



The new MariaDB 11.0 (Starting the optimizer)

MariaDB FOSDEM 2023

**Michael Widenius
CTO @ MariaDB**

11.0

Or the dawn of better optimizer plans

- 11.0 is one of the bigger milestones in MariaDB history!
 - The last comparable one was the big optimizer changes done in MariaDB 5.3)
- This is the first big cost change in MySQL/MariaDB, where we go away from the notion that 1 “optimizer cost” == 1 IO (disk read) cost.
- In the new model, **1 “optimizer cost” = “1 ms”**
 - Costs are calculated for all storage engine read calls, all optimizer strategies, checking the WHERE clause, sorting, rowid_filter. Different storage engines has different costs!
- This means that some plans will change (hopefully for the better)!
 - One should do some **production testing** before installing 11.0 on a production server with complex queries (to avoid the wrong kind of surprises).
- That said, 11.0 should be able to better handle complex production queries. There is also many new options to fix ‘bad queries’ than every before!
- As MariaDB is increasingly used to replace old expensive legacy databases, having a better optimizer is critical.

The intention of the optimizer changes

- Be able to find the '**best table combination and best access plan**' for a query.
 - The original optimizer cost model was not very good if there were no good indexes!
 - Replace most of the **remaining rule** based choices with **cost based** choices.
 - Take into account that different engines have different characteristics (Memory vs InnoDB)
- Allow the user to **fine tune the optimizer costs** for their environment.
 - (Hopefully they never need to do that, but it is now possible)
- Easier to compare query costs and also quickly see 'if a cost is reasonable'.
 - Having **costs in microseconds** helps to verify if a cost is 'totally wrong'
 - Optimizer trace now writes out a lot more information about the costs!

The intention for the future is to be able to enable all optimizer_switch options by default! For this we needed a better cost model, like the new one, as a base!

When optimizer cost calculation changes, unexpected things can happen to existing applications

- The new optimizer should be able to do a better choice when to use **table scan**, **index scan**, **index_merge**, **hash** and other join methods needed when key lookup cannot be used.
- Most applications, which are properly using keys, should be unaffected.
 - Simple queries will work as before
 - Most complex queries (with many tables) should perform equal *or better* than before.
- The new optimizer costs may need future tuning to be ‘perfect’ for most. If needed, this will be done over a few MariaDB releases..
- If things does not work, they can usually be fixed by changing an optimizer variable.

Optimizer trace added to MariaDB 10.4

The new optimizer trace has made it possible to start improving the optimizer. To use it one should do:

```
set optimizer_trace="enabled=on";  
SELECT ...  
select * from information_schema.OPTIMIZER_TRACE;
```

In an mtr test (mariadb-test-run) one can (starting with 10.5) use **--optimizer_trace** before a **SELECT** or **EXPLAIN** query that produces wrong results, to find out what is different from before.

optimizer_trace enabled me to start working on 11.0!

As part of the 11.0 work, much more information was added to optimizer trace!

Selectivity (bug fix)

- Starting from MariaDB 10.4.1 has **optimizer_use_condition_selectivity=4**
- In some cases the selectivity calculation is wrong (selectivity becomes > 1) and one gets a bad plan.
- Current workaround is to use **optimizer_use_condition_selectivity=1** if a plan is bad.

- Selectivity calculations are now fixed. There are now **asserts** in place in all selectivity calculations that ensure this cannot happen again.
- **The optimizer now uses the most optimistic (smallest number of rows) access method among all plans when estimating row count.**
 - One effect of this is that ‘explain extended’ now has a more accurate number for “filtered”.

Derived tables and union can now create distinct key

- Temporary derived tables are now creating unique keys to speed up searches.

Here is a diff from the commit: now eq_ref (unique key lookup) instead of ref

```
EXPLAIN UPDATE          t1, t2 SET a = 10 WHERE a IN (SELECT * FROM (SELECT b FROM t2 ORDER BY b
LIMIT 2,2) x);
id      select_type  table  type  possible_keys  key  key_len  ref  rows  Extra
1       PRIMARY t1     ALL   NULL           NULL 3        Using where
-1      PRIMARY <derived3>  ref    key0    key0    5      test.t1.a    2    FirstMatch(t1)
+1      PRIMARY <derived3>  eq_ref distinct_key distinct_key 5      test.t1.a    1
1       PRIMARY t2     ALL   NULL           NULL 3        Using where
3       DERIVED t2     ALL   NULL           NULL 3        Using filesort
```

New cost calculations

- Cost calculations for filesort, Unique, Rowid filters, join_cache and materialization are updated.
 - The consequences for these are:
 - MariaDB is more likely to use an index for order by
 - MariaDB will use filters a bit more than before.
 - Materialization costs are now a bit higher (and thus used a bit less)
- Cost of “Using index for group-by” corrected.
 - MariaDB will use “index for group by” optimization more optimal now.
- The disk access cost is now **assuming SSD!**
- When counting disk accesses, we assume that **all read rows are cached** for the duration of the query. If this calculation would not be done, the cost of joining a big table with a small one would be unreasonable high!

The new storage engine costs

- Cost calculations changed from using 'disk/row/index' access to **microseconds**.
- As part of this, the base costs (table_scan, index_scan, key_look, row_lookup) have been split into smaller parts:

select * from information_schema.optimizer_costs where engine="innodb"

	InnoDB	Aria
OPTIMIZER_DISK_READ_COST:	10.240000	10.240000
OPTIMIZER_INDEX_BLOCK_COPY_COST:	0.035600	0.035600
OPTIMIZER_KEY_COMPARE_COST:	0.011361	0.011361
OPTIMIZER_KEY_COPY_COST:	0.015685	0.015685
OPTIMIZER_KEY_LOOKUP_COST:	0.791120	0.435777
OPTIMIZER_KEY_NEXT_FIND_COST:	0.099000	0.082347
OPTIMIZER_DISK_READ_RATIO:	0.020000	0.020000
OPTIMIZER_ROW_COPY_COST:	0.060870	0.060866
OPTIMIZER_ROW_LOOKUP_COST:	0.765970	0.130839
OPTIMIZER_ROW_NEXT_FIND_COST:	0.070130	0.045916
OPTIMIZER_ROWID_COMPARE_COST:	0.002653	0.002653
OPTIMIZER_ROWID_COPY_COST:	0.002653	0.002653

Note that the **above costs are in microseconds**, while query costs (in optimizer_trace) are in **milliseconds!**

The new (important) SQL level costs

show variables like "optimizer%cost";

Variable_name	Value
optimizer_disk_read_cost	10.240000
...	
optimizer_where_cost	0.032000

show variables like "optimizer%ratio"

Variable_name	Value
optimizer_disk_read_ratio	0.020000

Verifying the optimizer costs (All data is in memory for this test)

check_costs.pl --engine=InnoDB

...

Timing table access for query: table scan

```
select sum(l_quantity) from test.check_costs_innodb
```

explain:

```
1 SIMPLE check_costs_innodb ALL 1000000
```

```
table_scan time: 113.629434 ms cost-where: 131.3373 cost: 163.3373
```

...

Cost/time ratio for different scans types

table scan	cost: 131.3373	time: 113.6294	cost/time: 1.1558
index scan	cost: 113.7515	time: 102.5823	cost/time: 1.1089
range scan	cost: 954.8773	time: 998.5945	cost/time: 0.9562
eq_ref_index_join	cost: 854.8883	time: 846.3483	cost/time: 1.0101
eq_ref_cluster_join	cost: 874.9233	time: 1010.3653	cost/time: 0.8659
eq_ref_join	cost: 1701.6433	time: 1792.2118	cost/time: 0.9495

Rule based -> Cost based

- The decision to use an index (and which index) for resolving ORDER BY/GROUP BY where only partly cost based before.
- The old optimizer would limit the number of 'expected key lookups' to 10% of the number of rows. This would cause the optimizer to use an index to scan a big part of a table when a full table scan would be much faster.
This code is now removed.
- InnoDB would limit the number of rows in a range and ref_per_key estimates to 50% of the total rows, which would confuse the optimizer for big ranges.
 - These caps are now removed.
- If there was a usable filter for an index, it was sometimes used without checking the complete cost of the filter.
- 'Aggregate distinct optimization with indexes' is now cost based.
 - Changes "Using index for group-by (scanning)" → "Using index for group-by"

Other things

- A lot of small changes to improve performance
 - Changed some critical functions to be inline
 - Improved rowid_filter filling code (faster)
 - More caching of values
 - Simplified code (removed extra calls that were not needed)
- Many (!) more code comments to existing code.
- Some old Mariadb bugs in Jira were solved by the new code.
- Some small improvements to **LIMIT**
- Indexes can now be used for ORDER BY/GROUP BY in sub queries (instead of filesort)
- Aria tables now supports rowid_filtering

Some other plan changes

- We now prefer indexes with more used key parts if the number of resulting rows is the same.
 - Where `key_part_1 = 1` **and** `key_part_2 < 10`
- For very small tables, index lookup is preferred over table scan
 - This is mainly because of the mysql-run-test (mtr) test suite which has mostly small tables.
 - This can be changed by setting `OPTIMIZER_SCAN_SETUP_COST=0`
 - Normally this should not matter for end users.
- Do not report in EXPLAIN scans on clustered primary keys as 'Using index'.
 - This is not an index scan, it is a table scan!
 - Maybe we should instead report 'Using clustered index' ?

The most important new optimizer cost variables

<code>optimizer_disk_read_ratio</code>	0.020000	The chance that an engine-block is cached
<code>optimizer_disk_read_cost</code>	10.240000	Time to read a 4K block from an SSD
<code>optimizer_where_cost</code>	0.032000	Time to execute the WHERE clause This time is added to all 'accepted' rows
<code>optimizer_scan_setup_cost</code>	10.000000	Cost added to all full table or index scans

The above variables (*in microseconds*) will ensure that if tuning is needed for the new cost calculations, one should be able to fix it by just adjusting one of the above variables in the MariaDB config file.

For example, increasing **`optimizer_where_cost`** will cause the optimizer to choose plans with less estimated rows.

Changing cost variables

All engine and “sql level” cost variables can be changed via mariadb startup options, in config files or dynamically using SQL.

```
set session optimizer_where_cost=1.0;  
set global innodb.OPTIMIZER_DISK_READ_COST=100;
```

- The “default” engine contains the default costs for all storage engines.
 - When a new engine is loaded, the default costs are taken from the “default” engine and then the engine updates its own internal costs and adds the user configured costs.
- To keep things fast, engine specific costs are stored in the table definition (TABLE_SHARE). This means that if one changes the cost for an engine, it will only take effect when new, not previously cached tables are accessed.
- You can use `flush tables` to force the table to use the new costs at next access.

When does the optimizer changes matter to you

- The new optimizer should be able to find a better plan
 - If you are using queries with more than two tables
 - If you have indexes with a lot of identical values
 - If you are using ranges that cover more than 10% of a table
 - ... WHERE key between 1 and 1000 -- Table has values 1-2000
 - If you have complex queries when not all used columns are or can be indexed
 - In which case you may need to depend on selectivity to get the right plan
 - If you are using queries mixing different storage engines
 - Like using both InnoDB and Memory engine in the same query.
 - If you have had to use FORCE INDEX to get a good plan.
 - If using ANALYZE TABLE made your plans worse (or not good enough)
 - If your queries have lots of derived tables (subselects)
 - Using ORDER BY / GROUP BY that could be resolved via indexes

State of things

- All changes (except minor tuning based on input from user or performance testing) are already done!
- The code can be found in the bb-11.0 tree, to be merged to 11.0 withing 2 weeks.
- The optimizer will be the main major change in this tree!
- We want the release to become 'stable' quickly and many testers would help!
- We encourage everyone to test this and give us feedback (through Jira) so that we can fix any bugs ASAP!
- **Please** consider putting 11.0-beta as an extra slave to your existing production and give us FEEDBACK!

A last note: Going through the optimizer code has given me a lot of ideas for cleanups that could be done. This will be done in next major MariaDB releases.

State of the optimizer

- All changes (except minor tuning based on input from user or performance testing) are already done!
- The code can be found in the bb-11.0 tree, to be merged to 11.0 withing 2 weeks.
- The optimizer will be the main major change in this tree!
- We want the release to become 'stable' quickly and many testers would help!
- We encourage everyone to test this and give us feedback (through Jira) so that we can fix any bugs ASAP!
- **Please** consider putting 11.0-beta as an extra slave to your existing production and give us FEEDBACK!

A last note: Going through the optimizer code has given me a lot of ideas for cleanups that could be done. This will be done in next major MariaDB releases.

Future optimizer plans

- Parallel query (work will start soon)
- More optimizer cleanups
 - Adding a cleaner interface for the different stages in query execution (do_select)
 - Taking some ideas from the volcano model, but keeping the current push model
 - This will make the code a bit faster as we can have more specialized methods, like separate code for normal join and left join.
- Enable all optimizer_switch options by default
 - Row_cache based hash_joins
 - Will first be enabled for joins where there are no good keys
- Hash joins
- Bushy plans

Acknowledgments

- A big thanks to Sergei Petrunia for optimizer_trace and for reviews and always been available to explain some of the optimizer internals to me!
- Thanks to Vicențiu Ciorbaru for adding cost calculations for filesort!
- Thanks to Andrew Hutchings for helping on the ColumnStore front.



Thank you

Any questions?