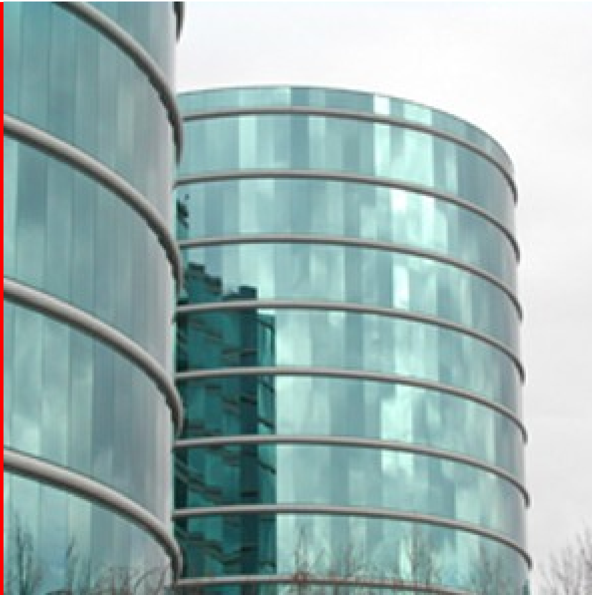


ORACLE®

# Partitioning under the hood in MySQL 5.5

**Mattias Jonsson, Partitioning developer**
**Mikael Ronström, Partitioning author**

# Who are we?

- Mikael is a founder of the technology behind NDB Cluster (MySQL Cluster)
- Mikael is also the author of the partitioning in MySQL 5.1 and COLUMNS extension in 5.5
- Mattias worked as a developer and MySQL consultant before joining MySQL in 2007, and have been fixing bugs and features in the partitioning code since.

## How is partitioning implemented?

- Extended syntax is added to the parser
- Common partitioning support routines
- A generic partitioning handler/engine for engines without native partitioning support
- NDB (MySQL Cluster) does partitioning natively
- Pruning as an extra optimizer step to only call into possible matching partitions

## Where is the code?

- All partitioning codes exists in the sql/ directory
- Structures in partition_info.{h,cc} and partition_element.h
- Common routines in sql_partition.{h,cc}
- Handler code in ha_partition.{h,cc}
- Pruning in opt_range.{h,cc}
- Minor partitioning specifics in sql_delete.cc, sql_update.cc, sql_select.cc (pruning), unireg.cc (frm handling), sql_table.cc (alter) etc.

# Execution flow

- Parsing
- Open and lock tables (including all partitions)
- Static pruning (only on partitioned tables)
- Query execution (including dynamic pruning)
- Sending results
- Unlock/close tables, cleaning up

# Overhead of open and lock a partitioned table

- Currently the hottest place for improvement for tables with many partitions (Bug#37252).
- In 5.1 all tables are opened and locked before the optimize step is done, and it is early in the optimizing step that the pruning is done.
- In 5.5 a new Meta Data Locking scheme is added which allow us to move the pruning step right after the table meta data lock and before the open/lock calls, so we also can prune open and locking.
    - Allows to prune inserts too.
    - Not yet implemented.

# How does partitioning work internally?

- For non native partitioned engines (all except NDB):
  - The handler (engine) for the table is set to ha_partition, which receives all calls from the server.
  - ha_partition handler creates a new handler for each partition with the 'real' storage engine (InnoDB, MyISAM, Memory, Archive etc.).
  - ha_partition forwards calls from the server to the partitions handlers depending on the type of operation.

# Examples of forwarding handler calls

- handler::write_row() - ha_partition calculates which partition the row belongs to and forward the call to that handler.
- handler::update_row() - ha_partition calculates which partitions the rows (old and updated/new) belongs to and if same it forward the call to that handler, if different partitions, it deletes from the old and inserts to new partition handler.
- hander::info() - Depends on the requested info; If simple only forward to first partition, else calculate from all partitions.

# Examples of forwarding handler calls. continued

- hander::index_* (select … order by <index> and more) starts by forward the call to all used partitions and creates a priority queue from the results. When one row is used it calls for the next value from that partition.
- handler::rnd_next (Scanning) calls one partition until no more rows, then continues with the next partition.

# Insert into a partitioned table

Yellow | Blue | Red | Green

| GOX 123 | $ 3000 | 2001 | Yellow |
|---|---|---|---|

# Update which results in change of partition

Yellow | Blue | Red | Green

Delete

Insert

| GOX 123 | $ 3000 | 2001 | Yellow |
|---|---|---|---|

| GOX 123 | $ 3000 | 2001 | Green |
|---|---|---|---|

# Index walking

Handler output

Merge Sort

Sorted output stream from Partition 1 index

Sorted output stream from Partition 4 index

Sorted output stream from Partition 5 index

Sorted output stream from Partition 8 index

## Why must all unique indexes include the partitioning functions columns?

- To ensure its uniqueness!
- Since the indexes is also partitioned, every unique tuple must be stored in the same partition
- So 'UNIQUE KEY (a,b) PARTITION BY HASH (c)' gives the possibility to place (1,2,3) and (1,2,4) in different partitions/indexes which can not enforce the uniqueness (1,2)!
- Support for global indexes is needed to solve this limitation.

# Pruning

- Only RANGE partitioning support full range pruning
- All other types support range pruning by partitioning walking (in 5.1 max 10 steps, in 5.5 max 32 steps)
- Remember that the optimizer does not handle func(col) = const, so use col = const instead, to let the optimizer its work
- Verify with 'EXPLAIN PARTITIONS'
- Its all about pruning, this is where you can win performance!

# EXPLAIN PARTITIONS

```
mysql> EXPLAIN PARTITIONS SELECT * FROM t1
  WHERE a BETWEEN 15 AND 25\G
*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: t1
partitions: p1,p2
...
```

# Dynamic Pruning

SELECT * FROM t1, t2 WHERE t1.a = t2.a

- If t1 is used as the inner loop, it is possible to select only one partition in each of its scan (one scan per record in outer table t2).
- If t1 is the outer loop it has to scan all partitions.
- Explanation: This works since there is an index on 'a' that contains all partitioning fields and this is bound for each scan in the inner loop.
- Cannot be seen in 'EXPLAIN PARTTIONS (WL#4128)

## How is the different partitioning types implemented?

- KEY works with a list of any column type by calculate a hash from all listed columns and then modulo by the number of partitions.
- COLUMNS is an extention for RANGE and LIST partitioning allowing the use of DATE, DATETIME, TIME and CHAR, VARCHAR columns as well as multi-column RANGE/LISTS.
- All other types works on integers only.
- HASH uses a simple modulo by number of partitions.

# RANGE partitioning

- The partition is found by binary search in the ranges.
- Pruning is also done on open ranges.
- Can also use COLUMNS for multi column ranges, and extended column types.
- Can also be subpartitioned by [LINEAR] HASH/KEY

# LIST partitioning

- All list values are stored in a sorted array.
- The partition is found by binary search in that array.
- Can also use COLUMNS for multi column ranges, and extended column types.
- Can also be subpartitioned by [LINEAR] HASH/KEY

# COLUMNS partitioning

- In MySQL 5.5
- Allows RANGE or LIST partitioning done on one or more columns with INT, DATE, DATETIME, TIME and CHAR, VARCHAR, BINARY types
- PARTITION BY {RANGE|LIST} COLUMNS (a,b,c) (PARTITION p1 VALUES {LESS THAN|IN} ('2010-04-14', 'Can you read this?', MAXVALUE));
- Also prunes complex WHERE clauses like ”a < '2010-04-15' and (b < 'Anyone' or b > 'None') and c between 10 and 100”

# LINEAR KEY/HASH partitioning

- The non linear KEY/HASH partitioning uses a modulo function for even distribution of records between partitions.
- Non linear KEY/HASH does a full rebuild of table for ADD/COALESCE partition
- By using a linear hashing algorithm some partitions can have twice as many rows as other partitions.
- But it only needs to rebuild one partition into two when adding and rebuild two partitions into one when coalesce partitions.
- Thus faster partition management.

## LINEAR HASH distribution

partition_name table_rows (14 additions ~ 78 s)

- p0 327687

ALTER TABLE t ADD PARTITION PARTITIONS 1

- p0 163843
- p1 163844

ALTER TABLE t ADD PARTITION PARTITIONS 1

- p0 81921
- p1 163844
- p2 81922

# Non LINEAR HASH distribution

partition_name   table_rows (14 additions ~  230 s)

- p0   327687

ALTER TABLE t ADD PARTITION PARTITIONS 1

- p0   163843
- p1   163844

ALTER TABLE t ADD PARTITION PARTITIONS 1

- p0   109229
- p1   109229
- p2   109229

# SUBpartitioning

- Combining RANGE/LIST with HASH/KEY.
- First level (partition) is done by RANGE/LIST.
- Second level (subpartition) is done by HASH/KEY
- The combination is done using:

  no_subpartitions * partition_id + subpartition_id

- If subpartitioned, then the partition is simply a group of subpartitions.

# ALTER TABLE t CMD PARTITION

- REORGANIZE, ADD, DROP, COALESCE and REBUILD is handled in mysql_alter_table
- From the function header comment of mysql_alter_table: 'This is a veery long function and is everything but the kitchen sink :)'
- Separate functions for handling the partitioning specifics:
    - Preparations are done in prep_alter_part_table, which analyzes if it is possible to do a 'fast' operation rather than a full table copy.
    - If possible to do a 'fast' alter, it is done in fast_alter_partition_table, which uses mysql_change_partitions, mysql_drop_partitions and mysql_rename_partitions.

# ALTER TABLE t CMD PARTITION

- ANALYZE, CHECK, OPTIMIZE and REPAIR is handled in mysql_admin_tables (just like their TABLE counterparts).
- Works by first mark given partitions, and then execute the operation only on those partitions.
- The handler functions is done like all others through the ha_partition handler.
- Note that InnoDB handler does not support OPTIMIZE, it is done by full table copy in the SQL layer followed by ANALYZE. For per partition OPTIMIZE use REBUILD + ANALYZE instead until bug#42822 is fixed.

# AUTO_INCREMENT handling

- ha_partition starts by initializing the auto_inc value from all partitions, and then keeps it in the table_share to avoid calling all partitions every time.
- If statement based replication is used it keeps a lock around the auto_inc value during the whole statement (as in multi-row insert/load) to keep it reproducable.
- It allows pre-allocation of values and release of non used values.
- The result is faster auto_increment handling and allows fewer gaps.

# INFORMATION_SCHEMA.PARTITIONS

- To get information about partition specifics like [sub]partition name, description, type, expression etc.
- And handler (table) statistics per partition such as rows, index/data size.
- Implemented by calling every partitions handler to get the data, so it is equivalent with INFORMATION_SCHEMA.TABLES

## ALTER TABLE t TRUNCATE PARTITION (p0, p3)

- In MySQL 5.5
- Uses the same code path as TRUNCATE TABLE with added partitioning pruning
- Uses the optimized delete_all_rows handler call to the partitions.

## Key caches per partition

- In MySQL 5.5
- Allows fine tuning of MyISAM key caches from table level to partition level.
- CACHE INDEX tbl PARTITION (ALL|p0[,p1...]) [INDEX|KEY (index_name[,index_name...])] IN key_cache_name
- LOAD INDEX INTO CACHE tbl PARTITION (ALL|p0[,p1...]) [INDEX|KEY (index_name[,index_name...])] [IGNORE LEAVES]

•Both assignment and preload per partition

# Experimental parallel ALTER TABLE

- Preview on launchpad lp:mysql-server/mysql-5.1-wl2550
- Can use all cores in one alter!
- Experimental!
- Two ways to copy in parallel:
  - Same partitioning → parallel data copy within groups of partitions
  - Multiple read threads (with one or more partition per thread) which sorts and feeds multiple write threads (with one or more partition per thread)

# EXCHANGE PARTITION WITH TABLE

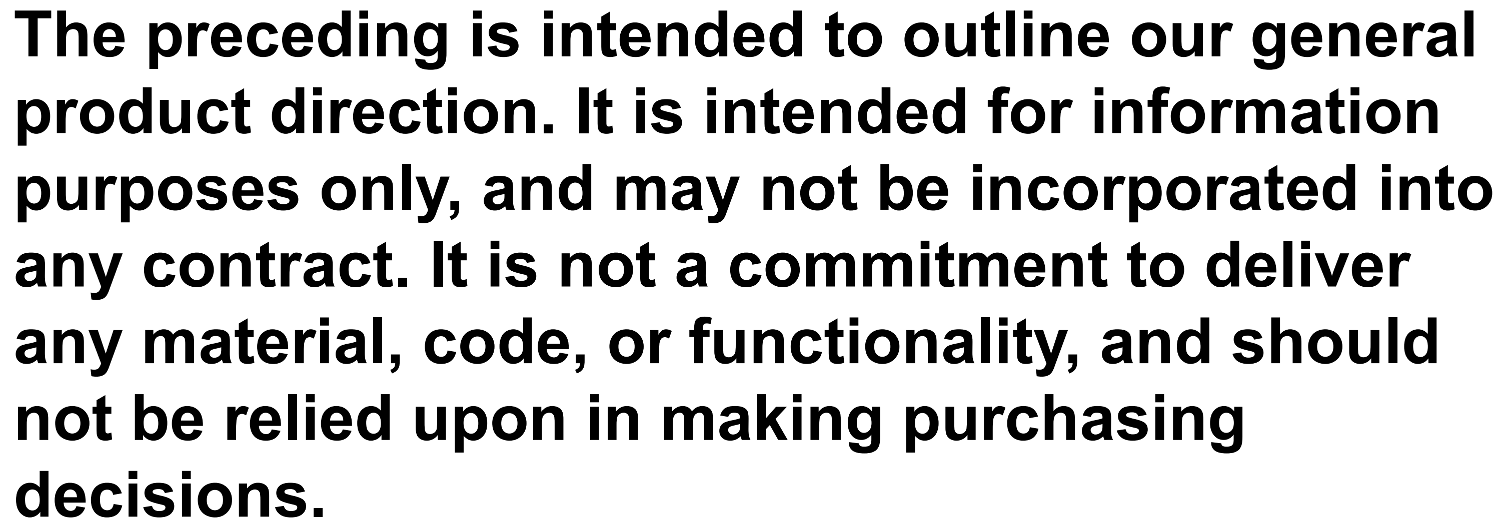
- Work in progress, WL#4445.
- Allows switching place of a table with a partition
- Both tables have to be created equally (accept that one is partitioned and the other is not).
- All rows in the table must belong to the exchanged partition (can be ignored for locking/performance reasons).
- Solves archiving of old partitions
- Allows import and export to/from a partitioned table (import – exchange with an empty partition, export – exchange with an empty table)

# EXCHANGE PARTITION WITH TABLE, continued

- ALTER TABLE tp EXCHANGE PARTITION p0 WITH TABLE t [IGNORE]
- If IGNORE is given (after verification is done by DBA) the operation is fast like a three way rename (t → t_tmp, tp_p0 → t, t_tmp → tp_p0)
- During verification both tables are write locked

## Explicit pruning

- Not yet implemented
- First step to change the pruning to avoid open and lock of pruned partitions.
- SELECT * FROM t1 PARTITION (p2, p5);
- Possibly also for insert, delete and update.
- Will allow to use the partitions instead as a WHERE clause.


The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.