



PARTITIONING UNDER THE HOOD

- **Mattias Jonsson**
 - M.Sc.
- **Mikael Ronström**
 - Ph.D.
- **Sun MySQL**

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
- Mattias worked as an developer and MySQL consultant before joining MySQL in 2007, and has worked with the partitioning code since.

How is partitioning implemented?

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

Where is the code?

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

Execution flow

- Parsing
- Open/lock tables (including all partitions)
- Static pruning (specific for partitioned tables)
- Query execution / sending result
- Unlock/close tables

Overhead of locking partitioned table

- Currently the hottest place for improvement
- The server always store locks for all tables used in the query before optimizing
- For a partitioned table, this also means store locks all partition
- After all locks are stored, they are sorted
- The sorting assumes only a few locks to sort, but for a heavily partitioned table, it can be many locks
- After sorting, they are locked in that order, to prevent deadlocks in the server.

What does this overhead mean?

- Since pruning is after the optimization, we always have to lock all partitions
- Having too many partitions will give every query a high overhead (easy to observe on small queries)
- For inserts, try to use multi-row inserts.
- If possible, encapsulate a set of queries with 'LOCK TABLE' (Also see bug#37252)
- It will not be fixed in 5.1, but the new meta-data locking in 5.4 will help to implement a fix for this limitation.

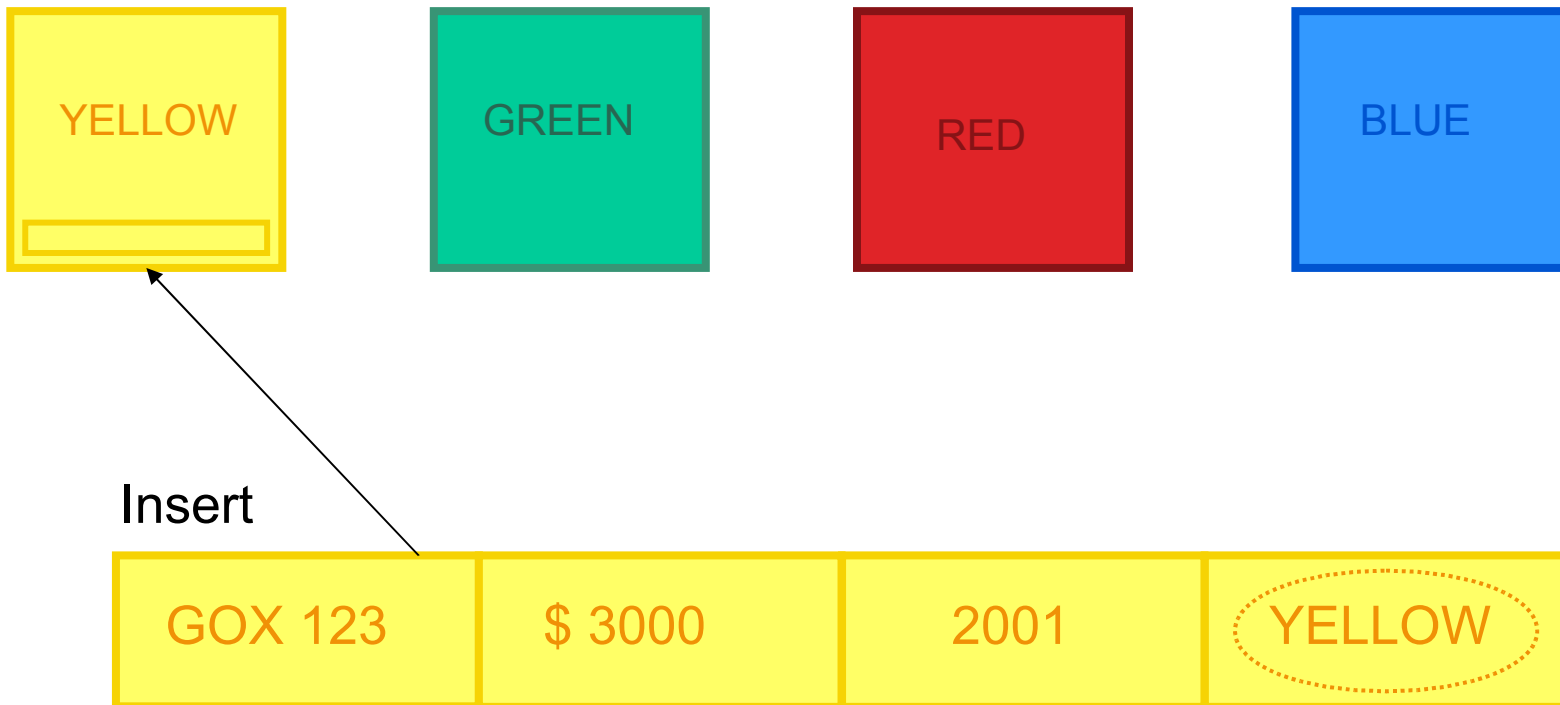
How does partitioning work?

- After opening and locking all tables, pruning is done to prevent scanning of non-matching partitions
- For non native partitioned engines (all except for NDB):
 - > The table engine/handler is the ha_partition, receiving all the server calls via the Storage Engine API
 - > Each partition is represented by an own underlying handler (InnoDB, MyISAM, Memory, Archive etc.) which in turn is called by the partitioning handler

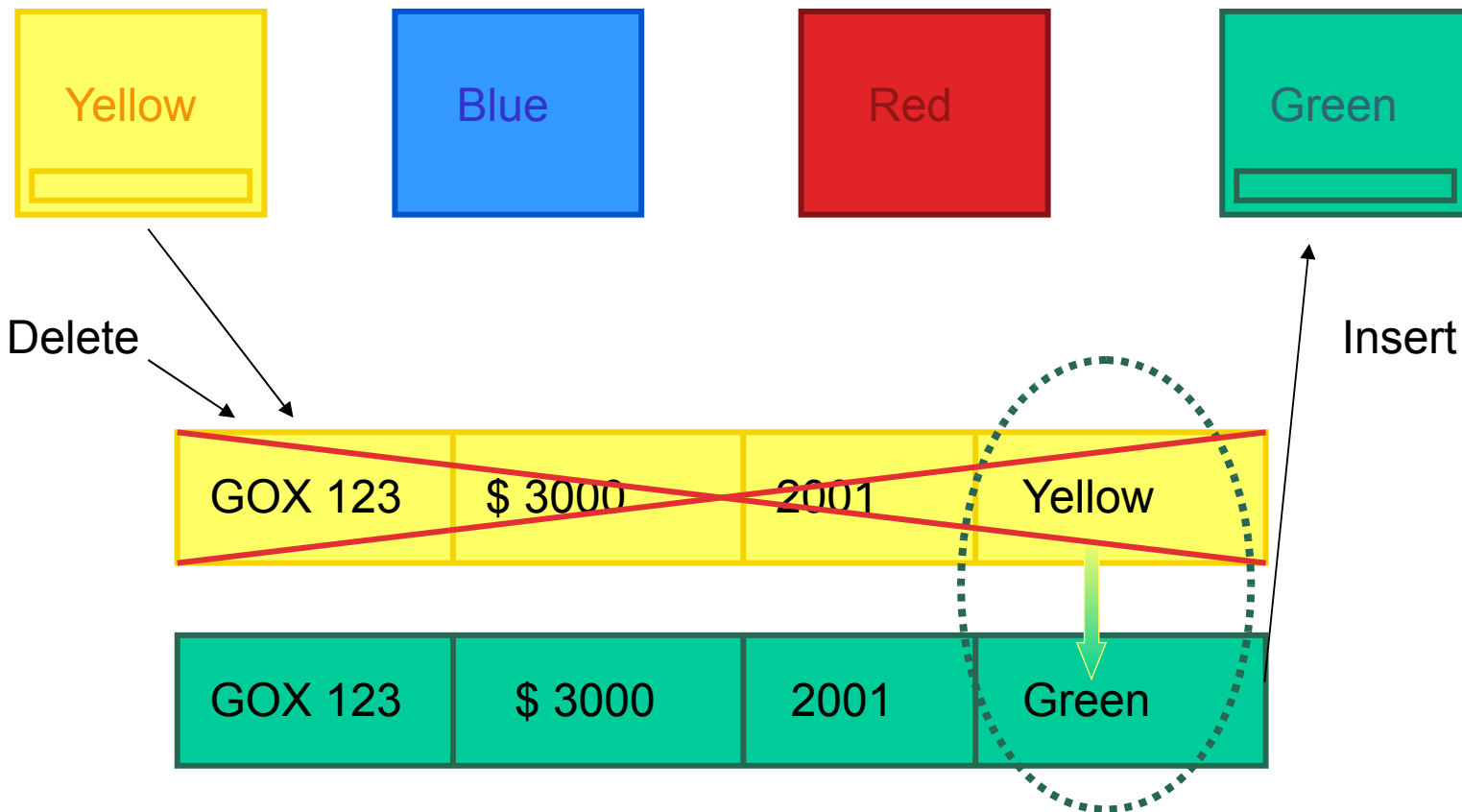
How is partitioning implemented?

- `handler::write_row` (insert) works by first calculate which partition it belongs to, and then redirect the call to that partitions handler
- `handler::update_row` (update) works the same, but if the row is to be moved, it is done by a delete in the old partition and insert in the new partition.
- `handler::index_*` (select ... order by <index>) has to forward the call to all partitions and create a priority queue to pick from
- For all scan and index reads, the pruning is used to avoid impossible partitions

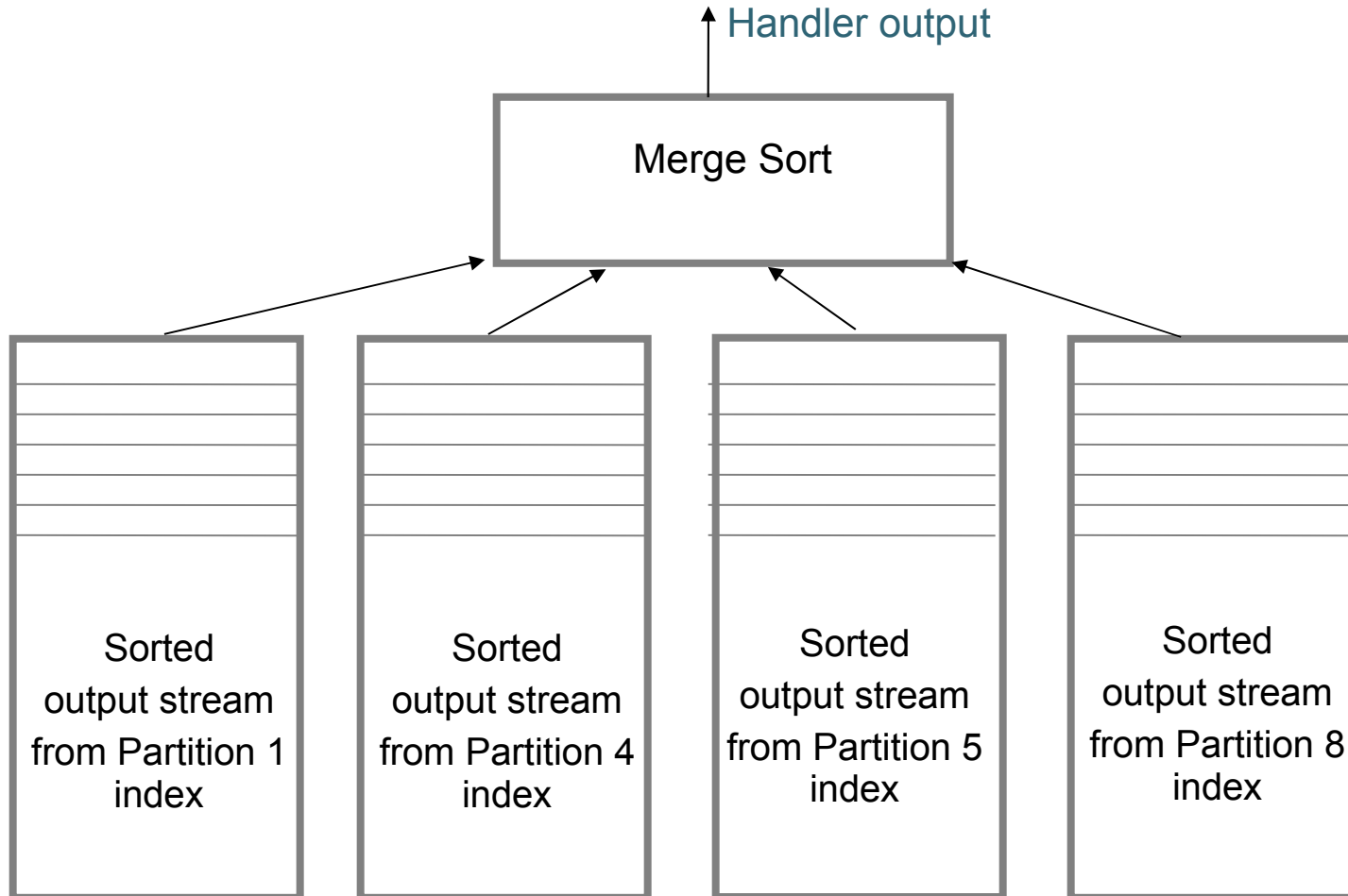
Insert



Update



Index walking



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 (c)' gives the possibility to place (1,2,3) and (1,2,4) in different partitions/indexes which can not enforce the uniqueness!
- Will be possible when global indexes is implemented

Pruning

- Only RANGE partitioning support full range pruning
- All other types support range pruning by partitioning walking (in 5.1 max 10 steps, in 5.4 max 32 steps)
- Remember that the optimizer cannot use `func(col) = val`, so use `col = val` instead and leave it to the optimizer
- Verify with EXPLAIN PARTITIONS
- It all about pruning, this is where the big gain is!

Dynamic Pruning

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

If t1 is 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 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 inner loop

Cannot be seen in EXPLAIN PARTITIONS
(WL#4128)

How is the different partitioning types implemented?

- Key works with a list of any column type (it calculates a hash/integer from hashing all listed columns and then modulo number of partitions)
- All other types works on integers only
- Hash use a simple modulo number of partitions
- The LINEAR variants is the same for both Hash and KEY, enabling less copying of data when adding/coalesce partitions; no need to rebuild all partitions, only splitting/merging the affected number

LINEAR KEY/HASH

- The normal HASH/KEY uses a modulo function to spread records => Even distribution of data among partitions
- Also leads to full rebuild of table for ADD/COALESCE Partition
- LINEAR HASH/KEY partitions use an algorithm based on linear hashing which means that some partitions will have twice as much data as others if the number of partitions is not on the form 2^n
- LINEAR HASH/KEY only requires rebuild of a number of the partitions, thus faster partition management at the cost of a slight imbalance

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

RANGE partitioning

- Partition found by binary search in the ranges.
- Pruning can also be done on open ranges.
- Can also be subpartitioned by [LINEAR] HASH/KEY

LIST partitioning

- All list values are stored in a sorted array
- Partition found by binary search of that array.
- Can also be subpartitioned by [LINEAR] HASH/KEY

SUBpartitioning

- Just combining RANGE/LIST with HASH/KEY
- First calculating which partition
- Then calculating which subpartition
- And finally combining those ($\text{no_subpartition} * \text{part_id} + \text{subpart_id}$)

ALTER TABLE t CMD PARTITION

- Handles REORGANIZE, ADD, DROP, COALESCE, REBUILD in `mysql_alter_table`
- Special functions for handling these ALTER PARTITION commands:
 - > Preparations done in `prep_alter_part_table`, which returns if it needs partition changes and if it can be done in a fast way (not copying the full table).
 - > If possible to do in a fast manner, it is done in `fast_alter_partition_table` which in turn uses `mysql_change_partitions/mysql_drop_partitions/mysql_rename_partitions`. For more information read the comments which explains the what and why.

ALTER TABLE t CMD PARTITION

- Handles ANALYZE, CHECK, OPTIMIZE, REPAIR in `mysql_admin_tables` (just like `ANALYZE/.. TABLE`)
- Operates just like the `TABLE` variants, but only affects the given partitions
- The handler methods is done through the `ha_partition` handler.
- Currently a bug in the `OPTIMIZE PARTITION` for InnoDB partitions due to no real support for `OPTIMIZE` in InnoDB, it proposes recreate which rebuild the full table. (see [bug#42822](#), use `REBUILD + ANALYZE` instead)

AUTO_INCREMENT Handling

- Was changed from always check with the partitions for the current auto_increment value to cache it in the table_share in the fix for bug#33479.
- It only loops through all partitions when initializing the shared auto_increment value
- Result is faster auto_increment handling in partitioned tables
- And fewer gaps

INFORMATION_SCHEMA.PARTITIONS

- [SUB]Partition name, description, type, expression etc
- Also some table statistics on per partition level such as rows, index/data size
- Implemented by calling every partitions handler to get the data, so it is equivalent with the INFORMATION_SCHEMA.TABLES.

How about TRUNCATE PARTITION?

- ALTER TABLE tbl_name TRUNCATE PARTITION (ALL|partition_name[,partition_name...])
- To be in 5.4 (patch in bug#19405)
- Uses the same code path as TRUNCATE TABLE with added partitioning pruning
- Uses the optimized delete_all_rows call

And using key caches per PARTITION?

- To be in 5.4 (patch in bug#39637)
- `CACHE INDEX tbl_name PARTITION (ALL|partition[, partition] ...) [INDEX|KEY (index_name[, index_name] ...)] IN key_cache_name`
- `LOAD INDEX INTO CACHE tbl_name PARTITION (ALL|partition[, partition] ...) [INDEX|KEY (index_name[,index_name] ...) [IGNORE LEAVES]`
- Both assignment and preload per partition
- Specific to partitioned MyISAM

Too many open files on partitioned MyISAM

- An experimental patch (in review) that splitted the open into open_handler and open_files
- Allows closing partition files based on LRU
- Specific to partitioned MyISAM, extensible to Archive.

Time consuming ALTER TABLE?

- Try doing it in parallel! Can use all cores in one alter!
- Experimental (need common thread handling in the server)
- Two implementations of the real copy data between tables
 - > No change in the partitioning – parallel data copy within clusters of partitions
 - > Otherwise several read threads which sorts/feeds several write threads (both clusters of one or more partitions).
- Preview at launchpad: lp:mysql-server/mysql-5.1-wl2550

Range partition on non integer column

- PARTITION BY RANGE COLUMN_LIST(a,b,c...)
(PARTITION p0 VALUES LESS THAN COLUMN_LIST('2009-04-22', 3.14, 'string'));
- Values can also be MINVALUE or MAXVALUE
- PARTITION BY LIST COLUMN_LIST(a,b,c...) also possible
- Pruning on TO_SECONDS function
- Preview at launchpad lp:~mikael-ronstrom/mysql-server/mysql-5.1-wl3352

Range partition on non integer column

```

CREATE TABLE t1
(joined DATE,
 category ENUM('beginner', 'medium', 'advanced', 'EOF'),
 name varchar(64))
PARTITION BY RANGE COLUMN_LIST(joined, category, name)
(PARTITION lt2008 VALUES LESS THAN (column_list('2008-01-01', 'beginner', "")),
 PARTITION bg2008 VALUES LESS THAN (column_list('2009-01-01', 'beginner', "")),
 PARTITION p12009 VALUES LESS THAN (column_list('2009-04-22', 'beginner', "")),
 PARTITION p22009 VALUES LESS THAN (column_list('2009-04-22', 'beginner', 'John')),
 PARTITION p32009 VALUES LESS THAN (column_list('2009-04-22', 'medium', "")),
 PARTITION p42009 VALUES LESS THAN (column_list('2009-04-22', 'advanced', "")),
 PARTITION p52009 VALUES LESS THAN (column_list('2009-04-22', 'EOF', "")),
 PARTITION future VALUES LESS THAN (column_list('2032-01-01', 'EOF', "")));

```

COLUMN_LIST partitioning

```
SELECT * FROM t1;
```

joined category	name
1999-01-01	advanced Lars
2001-02-01	medium Steven
2008-02-01	beginner Martin
2008-09-21	advanced John
2009-02-01	beginner Frank
2009-01-30	beginner Joe
2009-04-22	beginner Eve
2009-04-22	beginner Zoe
2009-04-22	medium Jona
2009-04-22	medium Sarah
2009-04-22	advanced Herb
2009-04-22	advanced Jane

COLUMN_LIST partitioning

```
SELECT PARTITION_NAME, TABLE_ROWS FROM  
INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME =  
't1' AND TABLE_SCHEMA = 'test';
```

PARTITION_NAME	TABLE_ROWS
Lt2008	2
Bg2008	2
P12009	2
P22009	1
P32009	1
P42009	2
P52009	2
future	0

Whats next?

- Moving partitions into tables
- Moving and checking tables into partitions
- Reworking the locking schedule, to allow pruning before locking (better support for this in 5.4), pruning for write operations
- Parallel scan (need more server support)
- Multi engine partitioning (Will be hard to get the optimizing properly)

PARTITIONING UNDER THE HOOD

- **Mattias Jonsson**
– mattias.jonsson at sun.com
- **Mikael Ronström**
– mikael.ronstrom at sun.com