Agenda

- Quick overview of Partitioning
- Use cases (4)
- Details
- Postlog
Introduction

What is Partitioning
What is PARTITIONing?

- Bunch of sub-tables, each with subset of data
- “Pruning” picks which partition(s) to use
  - `SELECT ... WHERE ...`
- Added in 5.1 (2005)
Partition Types

PARTITION BY RANGE(col)  (  
PARTITION p000 VALUES LESS THAN (123),  
PARTITION p123 VALUES LESS THAN (246),  
PARTITION rest VALUES LESS THAN MAXVALUE  )

Also PARTITION BY (though not useful)  
- KEY, LINEAR KEY, LIST, LIST COLUMNS, HASH, RANGE COLUMNS  
- SUBPARTITION BY...
Specific Disk Layout

- DATA/INDEX DIRECTORY = ...
  - Better to stripe the drives (RAID-5/10)

- PARTITIONing is not the same as Sharding
RANGE Key

- **Limitations**
  - *INTs, not FLOAT/DECIMAL*
  - *DATE/DATETIME/TIMESTAMP – TO_DAYS() and a few other functions*
  - *BY RANGE COLUMNS allows VARCHAR*
  - *(not much else allowed)*
No Intrinsic Benefit

- Partitioning rarely provides any benefit
  - Never space savings
  - Rarely speed improvement
- I present 4 cases where partitioning can speed up some aspect
Use Cases

Where PARTITIONing can shine
Use Case 1: Sliding Time

- **Situation**
  - News Articles; need to purge after 30 days

- **Problem:**
  - DELETE is slow

- **Solution**
  - CREATE TABLE ... PARTITION BY RANGE(TO_DAYS(dt))
  - DROP PARTITION – much faster than DELETE
  - Nightly script to DROP & REORGANIZE
Use Case 2: 2D index needed

- **Situation**
  - 2 ranges in WHERE
  
  ```
  WHERE lat BETWEEN 52.3 AND 52.5
  AND lng BETWEEN 4.6 AND 5.1
  ```

- **Problem:**
  - INDEXing can handle only one range

- **Solution...**
Use Case 2: 2D index needed (solution)

- **Solution**
  - *Use lat*10000, lng*10000 (Scale to MEDIUMINT)*
    
    PARTITION BY RANGE(lat)
    WHERE lat BETWEEN 523000 AND 525000
    AND lng BETWEEN 46000 AND 51000
  - PRIMARY KEY(lng, ...)
  - *After getting items from 'square', filter by distance*
Use Case 3: Last Partition’s Index is hot

• Situation
  – Most activity is in “latest” partition, and
  – Table is too big to cache, but
  – One partition can be cached

• Problem
  – Some keys (esp. GUID) are I/O bound
  – Some queries (esp. scans) are I/O bound

• Solution...
Use Case 3: hot Partition (solution)

- Solution
  - *Include Partition key in WHERE (to get pruning)*
  - *Only last partition is cached*
Use Case 4: Export/Import by Partition

• Situation
  – You want to “archive” old data

• Problem:
  – No good way to carve out a chunk of a table
  – DELETE and OPTIMIZE are slow

• Solution
  – Partition such that you archive exactly 1 partition at a time
  – “Transportable Tablespaces”
  – (5.7 has cleaner code; possible in 5.6)
Further Details

Miscellany notes
Limitations

<table>
<thead>
<tr>
<th></th>
<th>Abs Limit</th>
<th>Practical Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows:</td>
<td>&gt;= 0</td>
<td>&gt;= 1M</td>
</tr>
<tr>
<td>Partitions:</td>
<td>1 - 8192</td>
<td>5 - 50</td>
</tr>
</tbody>
</table>

- No diffs between Galera/MariaDB/PXC/Oracle
- No parallel actions → no benefit from multiple CPUs
Index Limitations

- No FOREIGN KEY support
- UNIQUE & PRIMARY keys must include Partition key
  - Suggest tacking onto end
  - Hence, UNIQUE is rarely useful
Point Query not Faster

- **SELECT * FROM t WHERE id = 12345;**
- Given a billion rows:
  - *Plain table:* 5-level Btree
  - *Partitioned:* Pick partition, then 4-level Btree
- Not much difference
AUTO_INCREMENT

PRIMARY KEY(id, partition_key)
  – versus

PRIMARY KEY(cols, partition_key, id)
INDEX(id)

• index(id) is sufficient for auto_inc
• PK starting with other cols gives clustering advantage
• Rarely useful to partition on PK.
Partition Pruning

- Most useful when WHERE clause restricts the Partition key
- Pruning may involve unnecessarily opening all partitions
- Pruning not done on writes (fixed in later versions)
- New in 5.6 (but rarely useful):
  - `SELECT * FROM ... PARTITION (p0, p2) WHERE ...`
EXPLAIN

- EXPLAIN PARTITIONS SELECT ...
  - (to see if pruning worked)

- BY RANGE(datetime)
  - 'first' partition contains 'bad' dates; always checked
Subtle issues

- **ALTER TABLE ... REORGANIZE, not OPTIMIZE PARTITION**

- **SHOW TABLE STATUS ... - Data_free:**
  - `innodb_file_per_table = OFF`: free space in `ibdata1`
  - `innodb_file_per_table = ON`: usually 4-7M (in larger tables/partitions)
Postlog

Futures, References
Summary

- Only BY RANGE is useful
- Only 4 use cases provide performance benefit
Futures

- “Global Index” / UNIQUE / FOREIGN KEY – someday
- “Native partitioning” now in 5.7
  - Decreases 'handler' overhead
  - Single file?
    - Impact on 'transportable' partitions??
References

- http://mysql.rjweb.org/doc.php/partitionmaint -- Use case 1
- mysql@rjweb.org -- Rick James