MySQL 5.1 and 5.5 partitions

Giuseppe Maxia
MySQL Community Team Lead
about me - Giuseppe Maxia

- a.k.a. The Data Charmer
- MySQL Community Team Lead
- Long time hacking with MySQL features
- Formerly, database consultant, designer, coder.
- A passion for QA
- An even greater passion for open source
- ... and community
- Passionate blogger
- http://datacharmer.blogspot.com
Updated presentation slides

- you will find an up to date copy of these slides at:
  - http://datacharmer.org/tutorial_uc2010/
We love your feedback!

- Tell it to the Twitter community
  - #mysqlconf "your feedback here"
- Twitter me
  - @datacharmer "your feedback here"
- Blog about it (if you already have a blog)
- Find my boss in the corridors and tell him I was great!
Feedback examples

- Tell it to the Twitter community

#mysqlconf Tutorial on partitions is fabulous!

Tutorial_attendee
Feedback examples

- Tell it to the Twitter community

#mysqlconf Tutorial on partitions sucks big time!
Feedback examples

- Twitter me

@datacharmer This tutorial kicks ass!

o_O Tutorial_attendee
Feedback examples

- Twitter me
Feedback examples

- Twitter me

@datacharmer Love it!
Will you marry me?

o_O Tutorial_attendee
Feedback examples

- Find my line manager: Kaj Arnö
- Tell him how good you feel about this tutorial
Topics

- Understand the partitioning types (range, hash, list, key)
- Partition pruning and EXPLAIN examples
- Benchmarking partitions
- Partitions and indexes
- How to partition by dates
- Benefits of partitioning
- Limits of partitioning
- Partitioning with MyISAM
- Partitioning with InnoDB
- Partitioning with Archive
- Partition maintenance
- Advanced partitioning techniques
- New features in 5.5
The Time Has Come...
MySQL Server 5.1
Defining the problem

YOUR NEEDS
The problem(s)

- Too much data
- Not enough RAM
- Historical data
- Growing data
- Rotating data
Too much data
Not enough RAM
Not enough RAM
MySQL partitions

WHAT
What exactly is this "partitions" thing?

- Logical splitting of tables
- Transparent to user
Remember the MERGE tables?

- separate tables
- risk of duplicates
- insert in each table
- no constraints
It isn't a merge table!

- One table
- No risk of duplicates
- Insert in one table
- Constraints enforced
Wait a minute ...

- WHAT THE HELL DOES "LOGICAL SPLIT" REALLY MEANS?
- LET ME EXPLAIN ...
Physical partitioning (1)

• Take a map
Physical partitioning (2)

- cut it into pieces

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Physical partitioning (3)

- What you have, is several different pieces
Physical partitioning (4)

• If you want the map back, you need some application (adhesive tape) and you may get it wrong
Logical partitioning (1)

• Take a map
Logical partitioning (2)

- fold it to show the piece you need
Logical partitioning (3)

- what you have is still a map, even if you see only one part.
Logical partitioning (4)

- if you unfold the map, you still have the whole thing
MySQL partitions

Partition pruning
Partition pruning
1a - unpartitioned table - SINGLE RECORD

```
select *
from
table_name
where colx = 120
```
Partition pruning

1a - unpartitioned table - SINGLE RECORD

```
select *
from table_name
where colx = 120
```
Partition pruning
1b - unpartitioned table - SINGLE RECORD

```sql
select * 
from 
table_name 
where colx = 350
```
Partition pruning

1c - unpartitioned table - RANGE

```
select *
from
table_name
where colx
between 120 and 230
```
Partition pruning
2a - table partitioned by colx - SINGLE REC

select * 
from 
table_name 
where colx = 120
Partition pruning
2a - table partitioned by colx - SINGLE REC

```
select *
from table_name
where colx = 120
```
Partition pruning
2b - table partitioned by colx - SINGLE REC

select *
from
table_name
where colx = 350
Partition pruning
2c - table partitioned by colx - RANGE

```
select *
from table_name
where colx between 120 and 230
```
Partition pruning

**Before**

```sql
EXPLAIN
select *
from table_name
where colx = 120
```

**In 5.1**

```sql
EXPLAIN PARTITIONS
select *
from table_name
where colx = 120
```
Partition pruning - unpartitioned table

```
explain partitions select count(*)
from table_name where colx=120\G

***** 1. row ****

    id: 1
    select_type: SIMPLE
    table: table_name
    partitions: NULL
    type: index

...
Partition pruning - unpartitioned table

```
explain partitions select count(*)
from table_name where colx between 120 and 230\G

***** 1. row ****
   id: 1
   select_type: SIMPLE
   table: table_name
   partitions: NULL
   type: index
...
```
Partition pruning - table partitioned by colx

```
explain partitions select count(*)
from table_name where colx between 120 and 230

***** 1. row ****
   id: 1
   select_type: SIMPLE
   table: table_name
   partitions: p02,p03
   type: index
```
MySQL partitions

HOW TO
Partitioning Types

- Partition a table using CREATE TABLE or ALTER TABLE

  ```sql
  CREATE TABLE <table_name> (<columns>)
  ENGINE=<engine_name>
  PARTITION BY <type> ( <partition_expression> );
  ```

- `<type>` can be RANGE, LIST, HASH or KEY
CREATE TABLE Employee (
  emp_id INT AUTO_INCREMENT,
  fname VARCHAR(50),
  lname VARCHAR(50),
  store_id TINYINT,
  PRIMARY KEY (emp_id)
) ENGINE=MyISAM

PARTITION BY RANGE (emp_id) (
  PARTITION p0 VALUES LESS THAN (10000),
  PARTITION p1 VALUES LESS THAN (20000),
  PARTITION p2 VALUES LESS THAN (30000),
  PARTITION p3 VALUES LESS THAN (40000),
  PARTITION p4 VALUES LESS THAN MAXVALUE)

- MAXVALUE is optional
- Partition ranges must be listed smallest to greatest and must be integers
CREATE TABLE Employee (  
    emp_id INT,  
    fname VARCHAR(50),  
    lname VARCHAR(50),  
    store_id TINYINT  
) ENGINE=MyISAM  
PARTITION BY LIST (store_id) (  
    PARTITION pNorth VALUES IN (2,8,12),  
    PARTITION pEast VALUES IN (1,4,7),  
    PARTITION pWest VALUES IN (3,5,6,10),  
    PARTITION pSouth VALUES IN (9,11)  
)

Notice there is no primary key – the primary key must include all columns in the partitioning expression
HASH

- CREATE TABLE Employee (emp_id INT AUTO_INCREMENT, fname VARCHAR(50), lname VARCHAR(50), store_id TINYINT, PRIMARY KEY (emp_id)) ENGINE=MyISAM PARTITION BY HASH (emp_id) PARTITIONS 4;

- HASH(expr) – expr must return an integer
- MySQL decides row placement using mod
- e.g., for emp_id=9, mod(9,4) = 1 which would put the row in p1
CREATE TABLE Employee (  
  emp_id INT,  
  fname VARCHAR(50),  
  lname VARCHAR(50),  
  store_id TINYINT  
) ENGINE=MyISAM  
PARTITION BY KEY (fname)  
PARTITIONS 4;

Unlike HASH, the partitioning expression does not have to be an integer
The hashing algorithm is similar to PASSWORD()
Partitioning expression

- RANGE, LIST and HASH must use integer values
- Some functions are permitted (that return an integer):
  
  - ABS()
  - CEILING()
  - DAY()
  - DAYOFMONTH()
  - DAYOFWEEK()
  - DAYOFYEAR()
  - DATEDIFF()
  - EXTRACT()
  - FLOOR()
  - HOUR()
  - MICROSECOND()
  - MINUTE()
  - MOD()
  - MONTH()
  - QUARTER()
  - SECOND()
  - TIME_TO_SEC()
  - TO_DAYS()
  - WEEKDAY()
  - YEAR()
  - YEARWEEK()

- No stored functions, stored procedures, or UDFs
Partition pruning

- PARTITION BY RANGE (emp_id) :
  WHERE emp_id = 1392
  WHERE emp_id > 45000
  WHERE emp_id BETWEEN 30000 AND 31000

- PARTITION BY LIST(store_id) :
  WHERE store_id IN(3,7,10,12)

- PARTITION BY HASH(emp_id) :
  WHERE emp_id = 1392

- PARTITION BY KEY(lname) :
  WHERE lname = 'Sproehnle'
Benchmarking partitions

- Compare results
- Unpartitioned vs partitioned
- ISOLATION
- Repeatability
- Check your resources!
Benchmarking partitions - Compare results

- Execute query
- Record execution time
- CHECK RETRIEVED RECORDS!
Benchmarking partitions - Unpartitioned vs partitioned

- Make it simple.
- Do not change structure.
- If needed, remove PK from partitioned table.
Benchmarking partitions - ISOLATION

- Try to reproduce working conditions
- No other servers running while benchmarking
- Restart the server before each test
- Do NOT mix partitioned and unpartitioned tables in the same server
- Use MySQL Sandbox
Benchmarking partitions - Repeatability

- Measure more than once
- Make sure you have the same conditions
- Make sure your results are consistent between runs
Benchmarking partitions - Check resources

- InnoDB
  - check disk space (uses more than MyISAM)
  - check CPU usage

- Partitioned MyISAM tables
  - use 2 file handles per partition
  - If you use more than one partitioned table, count total file handles

- If you use Archive partitioned tables
  - check CPU usage
  - check memory usage
Interlude - MySQL Sandbox

- (introduction to testing and benchmarking with MySQL Sandbox)
- (about 10 minutes)

http://launchpad.net/mysql-sandbox

Sunday, 11 April 2010
Indexes

- Indexes are partitioned along with the data

- **Primary/Unique keys:** All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

- Incorrect:
  UNIQUE (col1, col2)...PARTITION BY HASH(col3)

- Correct:
  UNIQUE (col1, col2, col3)..PARTITION BY HASH(col3)
Partitioning with MyISAM

- Each partition has a .MYD and .MYI file:
  Employee#P#p0.MYD
  Employee#P#p0.MYI

- DATA DIRECTORY and INDEX DIRECTORY can be used per partition

- Examples...
Partitioning with InnoDB

- Without `--innodb_file_per_table` the partitions are in the centralized table space.

- With `--innodb_file_per_table` each partition has an .ibd file:
  Employee#P#p0.ibd
  Employee#P#p1.ibd

- Examples...
Partitioning by date - limits of partitioning

- Can partition only by INTEGER columns
- OR you can partition by an expression, which must return an integer
Partitioning by date - your options

- BAD ideas
  - Convert the date to UNIX_TimeStamp
  - Using unformatted dates (20050215)
  - Using expressions

- GOOD ideas
  - YEAR(date_column)
  - TO_DAYS(date_column)
Partitioning by date - your (BAD) options

BAD ideas
- Convert the date to UNIX(TimeStamp)
- Using unformatted dates (20050215)
- using expressions

WHY?
- you can't use easily the server date arithmetic
- Column value not human readable
- Partition pruning may not kick in
Partitioning by date - your (GOOD) options

- GOOD ideas
  - YEAR(date_column)
  - TO_DAYS(date_column)

- WHY?
  - Both functions optimized for partitions
  - Partition pruning does not kick with other functions
CREATE TABLE t1 (  
    d date  
)  
PARTITION BY RANGE (YEAR(d))  
(  
    PARTITION p01 VALUES LESS THAN (1999),  
    PARTITION p02 VALUES LESS THAN (2000),  
    PARTITION p03 VALUES LESS THAN (2001),  
    PARTITION p04 VALUES LESS THAN (MAXVALUE)  
);
CREATE TABLE t1 (  
    d date)  
PARTITION BY RANGE (TO_DAYS(d))  
(  
PARTITION p01 VALUES LESS THAN 
   (TO_DAYS('2008-01-01')),  
PARTITION p02 VALUES LESS THAN 
   (TO_DAYS('2008-02-01')),  
PARTITION p03 VALUES LESS THAN 
   (TO_DAYS('2008-03-01')),  
PARTITION p04 VALUES LESS THAN (MAXVALUE)  
);
Partitioning by date - How TO

- How it works
  - partition BY FUNCTION
  - query BY COLUMN
Partitioning by date -WRONG!!!

PARTITION BY RANGE(year(from_date))

select count(*) from salaries where year(from_date) = 1998;

+----------+
| count(*) |
+----------+
| 247489   |
+----------+
1 row in set (2.25 sec)
Partitioning by date - RIGHT

PARTITION BY RANGE(year(from_date))

select count(*) from salaries where from_date between '1998-01-01' and '1998-12-31';

+----------+
<table>
<thead>
<tr>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>247489</td>
</tr>
</tbody>
</table>
+----------+

1 row in set (0.46 sec)
Partitioning by date - EXPLAIN

```
explain partitions select count(*) from salaries where year(from_date) = 1998\G

***** 1. row ****

    id: 1
    select_type: SIMPLE
    table: salaries
    partitions: p01,p02,p03,p04,p05,p06,p07,p08,p09,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19
    type: index

...
Partitioning by date - EXPLAIN

explain partitions select count(*) from salaries where from_date between '1998-01-01' and '1998-12-31'

***** 1. row ****

id: 1

select_type: SIMPLE

table: salaries

partitions: p14, p15

...
Partitioning by date with different sizes

- Mixing partitions by year, month, day in the same table

**HOW TO:**
- Use the "TO_DAYS" function
- set appropriate intervals
Partitioning by date with different sizes

ALTER TABLE salaries

partition by range (to_days(from_date))

(  
  # 5 years
  partition p01 values less than (to_days('1985-01-01')),
  partition p06 values less than (to_days('1990-01-01')),

  # 1 year
  partition p11 values less than (to_days('1995-01-01')),
  partition p12 values less than (to_days('1996-01-01')),
  partition p13 values less than (to_days('1997-01-01')),
  partition p14 values less than (to_days('1998-01-01')),
  partition p15 values less than (to_days('1999-01-01')),
  partition p16 values less than (to_days('2000-01-01'))),
ALTER TABLE salaries
partition by range (to_days(from_date))
(
  # 1 month
  partition p17 values less than (to_days('2001-01-01')),
  partition p18 values less than (to_days('2001-02-01')),
  partition p19 values less than (to_days('2001-03-01')),
  partition p20 values less than (to_days('2001-04-01')),
  partition p21 values less than (to_days('2001-05-01')),
  partition p22 values less than (to_days('2001-06-01')),
  partition p23 values less than (to_days('2001-07-01')),
  partition p24 values less than (to_days('2001-08-01')),
  partition p25 values less than (to_days('2001-09-01')),
  partition p26 values less than (to_days('2001-10-01')),
  partition p27 values less than (to_days('2001-11-01')),
  partition p28 values less than (to_days('2001-12-01')))
Benefits of Partitions

- To make single inserts and selects faster
- To make range selects faster
- To help split the data across different paths
- to store historical data efficiently
Benefits of Partitions

- to add a check constraint to a table
- to add control to ARCHIVE tables
- to differentiate roles on master and slave (fast insert on the master with HASH partitions, fast statistics in a slave with RANGE partitions)
when to use partitions

- if you have large tables
- if you know that you will always query for the partitioning column
- if you have historical tables that you want to purge quickly
- if your indexes are larger than the available RAM
MySQL partitions

HOW TO
Maintaining RANGE and LIST

- For RANGE partitioning, partitions can only be added to the end:
  
  ```sql
  CREATE TABLE Employee ( ... )
  PARTITION BY RANGE (emp_id) (
    PARTITION p0 VALUES LESS THAN (10000),
    PARTITION p1 VALUES LESS THAN (20000),
    PARTITION p2 VALUES LESS THAN (30000),
    PARTITION p3 VALUES LESS THAN (40000);
  
  ALTER TABLE Employee ADD PARTITION (PARTITION p4 VALUES LESS THAN (50000));
  ```

- For LIST partitioning, partitions can be added that use different values

- Dropping a partition DELETES the data in that partition:
  
  ```sql
  ALTER TABLE Employee DROP PARTITION p1
  ```
Maintaining HASH and KEY

- Adding partitions reorganizes the data using the hashing algorithm

  **ALTER TABLE .. ADD PARTITION PARTITIONS 2;**

  This increases the number of partitions by 2

- DROP PARTITION does not work; instead, use COALESCE to merge partitions

  **ALTER TABLE .. COALESCE PARTITION 2;**

  This decreases the number of partitions by 2
Other maintenance commands

- **REORGANIZE PARTITION** – this can split or merge partitions
- **REMOVE PARTITIONING** – reverts to an unpartitioned table
- **CHECK PARTITION** – similar to CHECK TABLE
- **REPAIR PARTITION** – similar to REPAIR TABLE
- **ANALYZE PARTITION**
- **OPTIMIZE PARTITION**
Advanced Partitioning Features

- Subpartitioning
- Partition Maintenance using Events
- Using the Partition Helper
Subpartitioning

- RANGE or LIST partitions can be subpartitioned with HASH or KEY

- PARTITION BY RANGE(emp_id) SUBPARTITION BY HASH(store_id) SUBPARTITIONS 2 (
  PARTITION p0 VALUES LESS THAN(10000),
  PARTITION p1 VALUES LESS THAN(20000),
  PARTITION p2 VALUES LESS THAN(30000),
  PARTITION p3 VALUES LESS THAN(40000),
  PARTITION p4 VALUES LESS THAN(50000)
);
MySQL partitions

HANDS ON
Hands on - Partitioning with MyISAM

- Live examples
- Primary key matters - not always needed
- Size of indexes matters
components used for testing

- MySQL Sandbox
  - created MySQL instances in seconds
  - [http://launchpad.net/mysql-sandbox](http://launchpad.net/mysql-sandbox)

- MySQL Employees Test Database
  - has ~ 4 mil records in 6 tables
  - [http://launchpad.net/test-db](http://launchpad.net/test-db)

- Command line ability
- fingers
- ingenuity
employees test database

[Database diagram showing relationships and tables]
How many partitions

from information_schema.partitions

| pname | expr            | descr    |
|-------|-----------------+----------|
| p01   | year(from_date) | 1985     |
| p02   | year(from_date) | 1986     |
| p13   | year(from_date) | 1997     |
| p14   | year(from_date) | 1998     |
| p15   | year(from_date) | 1999     |
| p16   | year(from_date) | 2000     |
| p19   | year(from_date) | MAXVALUE |
How many records

```sql
select count(*) from salaries;
```

<table>
<thead>
<tr>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2844047</td>
</tr>
</tbody>
</table>

Sunday, 11 April 2010
How many records in 1998 not partitioned

```
select count(*) from salaries where
from_date between '1998-01-01' and '1998-12-31';
```

```
+----------+
| count(*) |
+----------+
|   247489 |
```

1 row in set (1.52 sec)

# NOT PARTITIONED
How many records in 1998 partitioned

```
select count(*) from salaries where from_date between '1998-01-01' and '1998-12-31';
+----------+
| count(*) |
+----------+
|   247489 |
+----------+
1 row in set (0.41 sec)
```

# partition p15
Deleting records in 1998
NOT partitioned

delete from salaries where from_date between '1998-01-01' and '1998-12-31';
Query OK, 247489 rows affected (19.13 sec)

# NOT PARTITIONED
Deleting records in 1998 partitioned

```
alter table salaries drop partition p15;
Query OK, 0 rows affected (1.35 sec)
```
Partitions with InnoDB

- Slower than MyISAM
- But more robust
- Requires more storage
Partitioning with InnoDB - File per table

CREATE TABLE table_name (  
  . . .  
  . . .  
) ENGINE = INNODB  
ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8  
PARTITION BY XXXX
Partitions with InnoDB (laptop)

- Key points:
- Takes much more storage than other engines

<table>
<thead>
<tr>
<th>engine</th>
<th>storage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>innodb</td>
<td>221</td>
</tr>
<tr>
<td>myisam</td>
<td>181</td>
</tr>
<tr>
<td>archive</td>
<td>74</td>
</tr>
<tr>
<td>innodb partitioned (whole)</td>
<td>289</td>
</tr>
<tr>
<td>innodb partitioned (file per table)</td>
<td>676</td>
</tr>
<tr>
<td>myisam partitioned</td>
<td>182</td>
</tr>
<tr>
<td>archive partitioned</td>
<td>72</td>
</tr>
</tbody>
</table>
## Benchmarking results (laptop)

<table>
<thead>
<tr>
<th>engine</th>
<th>query year 2000</th>
<th>query year 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnoDB</td>
<td>1.25</td>
<td>1.25</td>
</tr>
<tr>
<td>MyISAM</td>
<td>1.72</td>
<td>1.73</td>
</tr>
<tr>
<td>Archive</td>
<td>2.47</td>
<td>2.45</td>
</tr>
<tr>
<td>InnoDB partitioned whole</td>
<td>0.24</td>
<td>0.10</td>
</tr>
<tr>
<td>InnoDB Partitioned (file per table)</td>
<td>0.45</td>
<td>0.10</td>
</tr>
<tr>
<td>MyISAM partitioned</td>
<td>0.18</td>
<td>0.12</td>
</tr>
<tr>
<td>Archive partitioned</td>
<td>0.22</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Partitions with InnoDB (huge server)

- Key points:
  - Takes much more storage than other engines

<table>
<thead>
<tr>
<th>engine</th>
<th>storage (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>innodb (with PK)</td>
<td>330</td>
</tr>
<tr>
<td>myisam (with PK)</td>
<td>141</td>
</tr>
<tr>
<td>archive</td>
<td>13</td>
</tr>
<tr>
<td>innodb partitioned (no PK)</td>
<td>237</td>
</tr>
<tr>
<td>myisam partitioned (no PK)</td>
<td>107</td>
</tr>
<tr>
<td>archive partitioned</td>
<td>13</td>
</tr>
</tbody>
</table>
## Benchmarking results (huge server)

<table>
<thead>
<tr>
<th>engine</th>
<th>6 month range</th>
</tr>
</thead>
<tbody>
<tr>
<td>InnoDB</td>
<td>4 min 30s</td>
</tr>
<tr>
<td>MyISAM</td>
<td>25.03s</td>
</tr>
<tr>
<td>Archive</td>
<td>22 min 25s</td>
</tr>
<tr>
<td>InnoDB partitioned by month</td>
<td>13.19</td>
</tr>
<tr>
<td>MyISAM partitioned by year</td>
<td>6.31</td>
</tr>
<tr>
<td>MyISAM partitioned by month</td>
<td>4.45</td>
</tr>
<tr>
<td>Archive partitioned by year</td>
<td>16.67</td>
</tr>
<tr>
<td>Archive partitioned by month</td>
<td>8.97</td>
</tr>
</tbody>
</table>
Partitions with Archive

- Key points:
- REMOVE INDEXES (not supported by archive)
- For pure statistics applications, ARCHIVE can be ALMOST AS FAST AS MyISAM!
CREATE TABLE titles (  
  emp_no INT NOT NULL,  
  title VARCHAR(50) NOT NULL,  
  from_date DATE NOT NULL,  
  to_date DATE,  
  KEY (emp_no),  
  FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,  
  PRIMARY KEY (emp_no,title,from_date)  
) ENGINE = InnoDB;
Partitions with ARCHIVE - in practice

CREATE TABLE titles (  
  emp_no INT NOT NULL,  
  title VARCHAR(50) NOT NULL,  
  from_date DATE NOT NULL,  
  to_date DATE #,  
  # KEY (emp_no),  
  # FOREIGN KEY (emp_no)  
  # REFERENCES employees  
  # (emp_no) ON DELETE CASCADE,  
  # PRIMARY KEY (emp_no,title,  
  #   from_date)  
) ENGINE = ARCHIVE;
annoyances

- CREATE TABLE STATEMENT hard to read
annoyances

- CREATE TABLE STATEMENT hard to read
annoyances (no more!)

- hard to read - Fixed in 5.1.31!
annoyances (no more!)

- hard to read - Fixed in 5.1.31!
annoyances

- CREATE TABLE only keeps the result of the expression for each partition.
- (you can use the information_schema to ease the pain in this case)
annoyances - table creation details

CREATE TABLE t1 (  
  d date  
) ENGINE=InnoDB
PARTITION BY RANGE (to_days(d))
(  
  PARTITION P1 VALUES LESS THAN (to_days('1999-01-01')),
  PARTITION P2 VALUES LESS THAN (to_days('2000-01-01'))
)
CREATE TABLE `t1` (  `d` date DEFAULT NULL ) ENGINE=InnoDB DEFAULT CHARSET=latin1  /*!50100 PARTITION BY RANGE (to_days(d)) (PARTITION P1 VALUES LESS THAN (730120) ENGINE = InnoDB,  PARTITION P2 VALUES LESS THAN (730485) ENGINE = InnoDB) */
annoyances - table creation details - fixing

```sql
select partition_name part,
partition_expression expr,
partition_description val
from information_schema.partitions where
table_name='t1';
```

```
+------+------------+--------+
<table>
<thead>
<tr>
<th>part</th>
<th>expr</th>
<th>val</th>
</tr>
</thead>
</table>
+------+------------+--------+
| P1   | to_days(d) | 730120 |
| P2   | to_days(d) | 730485 |
+------+------------+--------+
```
### annoyances - table creation details - fixing

```sql
select partition_name part, partition_expression expr, from_days (partition_description) val from information_schema.partitions where table_name='t1';
```

<table>
<thead>
<tr>
<th>part</th>
<th>expr</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>to_days(d)</td>
<td>1999-01-01</td>
</tr>
<tr>
<td>P2</td>
<td>to_days(d)</td>
<td>2000-01-01</td>
</tr>
</tbody>
</table>
TOOLS

- The INFORMATION_SCHEMA.PARTITIONS table
- The partition helper
- A Perl script that creates partitioning statements
- The mysqldump_partition_backup
- ... anything you are creating right now :)

Sunday, 11 April 2010
<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>PARTITION_NAME</th>
<th>SUBPARTITION_NAME</th>
<th>PARTITION_ORDINAL_POSITION</th>
<th>SUBPARTITION_ORDINAL_POSITION</th>
<th>PARTITION_METHOD</th>
<th>SUBPARTITION_METHOD</th>
<th>PARTITION_EXPRESSION</th>
<th>SUBPARTITION_EXPRESSION</th>
<th>PARTITION_DESCRIPTION</th>
<th>TABLE_ROWS</th>
<th>PARTITION_COMMENT</th>
</tr>
</thead>
</table>
SELECT * FROM partitions WHERE table_name = 'salaries' AND table_schema = 'employees' LIMIT 1;

- **TABLE_SCHEMA**: employees
- **TABLE_NAME**: salaries
- **PARTITION_NAME**: p01
- **SUBPARTITION_NAME**: NULL
- **PARTITION_ORDINAL_POSITION**: 1
- **SUBPARTITION_ORDINAL_POSITION**: NULL
- **PARTITION_METHOD**: RANGE COLUMNS
- **SUBPARTITION_METHOD**: NULL
- **PARTITION_EXPRESSION**: from_date
- **SUBPARTITION_EXPRESSION**: NULL
- **PARTITION_DESCRIPTION**: '1985-12-31'
- **TABLE_ROWS**: 18238
SELECT partition_name, partition_description, table_rows
FROM partitions
WHERE table_name = 'salaries' AND
  table_schema = 'employees';

+----------------+-----------------------+------------+
<table>
<thead>
<tr>
<th>partition_name</th>
<th>partition_description</th>
<th>table_rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>'1985-12-31'</td>
<td>18238</td>
</tr>
<tr>
<td>p02</td>
<td>'1986-12-31'</td>
<td>37915</td>
</tr>
<tr>
<td>p03</td>
<td>'1987-12-31'</td>
<td>57395</td>
</tr>
<tr>
<td>p04</td>
<td>'1988-12-31'</td>
<td>76840</td>
</tr>
<tr>
<td>p05</td>
<td>'1989-12-31'</td>
<td>95890</td>
</tr>
<tr>
<td>p06</td>
<td>'1990-12-31'</td>
<td>114520</td>
</tr>
<tr>
<td>p07</td>
<td>'1991-12-31'</td>
<td>132578</td>
</tr>
<tr>
<td>p08</td>
<td>'1992-12-31'</td>
<td>151019</td>
</tr>
<tr>
<td>p09</td>
<td>'1993-12-31'</td>
<td>168103</td>
</tr>
<tr>
<td>p10</td>
<td>'1994-12-31'</td>
<td>185121</td>
</tr>
<tr>
<td>p11</td>
<td>'1995-12-31'</td>
<td>201576</td>
</tr>
<tr>
<td>p12</td>
<td>'1996-12-31'</td>
<td>218244</td>
</tr>
<tr>
<td>p13</td>
<td>'1997-12-31'</td>
<td>233144</td>
</tr>
<tr>
<td>p14</td>
<td>'1998-12-31'</td>
<td>247458</td>
</tr>
</tbody>
</table>

Sunday, 11 April 2010
The Partition helper

- Introduction, syntax, and examples (5 minutes)

- LIVE EXAMPLE
The partition helper

$ partition_helper --help

The Partition Helper, version 1.0.1
This program creates a ALTER TABLE statement to add or reorganize partitions for MySQL 5.1 or later
(C) 2008 Giuseppe Maxia

syntax: partitions_helper [options]
-t --table = name
-c --column = name
-i --interval = name
   "year", "month", or a number
-p --partitions = number
...
The partition helper

... 

--first_partition = number
--reorganize = name
-s --start = name
-e --end = name
-f --function = name
-l --list
--prefix = name
--explain
Partitions maintenance through events

- An article by Greg Haase about this topic
- http://dev.mysql.com/tech-resources/articles/partitioning-event_scheduler.html
mysqldump_partition_backup

- a script created by Roland Bouman
- detects the partitions for a table, and generates mysqldump statements for each one
Partitions maintenance through events

HOW TO

1. Gather information about partitions (from INFORMATION_SCHEMA)
2. dynamically create the partitions info
3. fire the event
Partitions maintenance through events
HOW TO - example

• we ant to drop the partition with the oldest data
• Let's do it manually first
finding the partition with the oldest data

```
select partition_name
from information_schema.partitions
where table_name='salaries' and
      table_schema='employees' and
      partition_ordinal_position = 1;
```

+------+
| part  |
+------+
| p01   |
+------+
dropping the oldest partition

```
alter table salaries drop partition p01;
Query OK, 0 rows affected (0.96 sec)
Records: 0  Duplicates: 0  Warnings: 0
```
putting things together (1)

delimiter //
drop procedure if exists drop_oldest_partition //
create procedure drop_oldest_partition (my_table varchar(100))
modifies SQL data
begin
  declare oldest_partition varchar(100);
  # ....
putting things together (2)

... set oldest_partition = ( select partition_name
from
information_schema.partitions
where table_name=my_table
and table_schema=schema()
and partitionOrdinal_position
= 1);

...

Sunday, 11 April 2010
putting things together (3)

```sql
set @q =CONCAT('alter table ', my_table , ' drop partition ', oldest_partition);
prepare q from @q;
execute q;
deallocate prepare q;
end//

delimiter ;
```
putting things together (4)

drop event if exists dop;
create event dop
on schedule every 1 month
do call drop_oldest_partition ('salaries');
Checking and repairing partitions

- transparent to the user
- use CHECK TABLE and REPAIR TABLE
Checking and repairing partitions - example corrupting a partition

$ ls -l salaries*p01*
  0 Apr 16 07:54 salaries#P#p01.MYD
  1024 Apr 16 07:56 salaries#P#p01.MYI

$ ls -l salaries*p13*
  3274635 Apr 16 07:56 salaries#P#p13.MYD
  5183488 Apr 16 07:56 salaries#P#p13.MYI

$ cp salaries#P#p01.MYI salaries#P#p13.MYI
overwrite salaries#P#p13.MYI? (y/n [n]) y
### Checking and repairing partitions - example checking partitions

```sql
check table salaries;
```

<table>
<thead>
<tr>
<th>Table</th>
<th>Op</th>
<th>Msg_type</th>
<th>Msg_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>employees.salaries</td>
<td>check</td>
<td>warning</td>
<td>Size of datafile is: 3274635 Should be: 0</td>
</tr>
<tr>
<td>employees.salaries</td>
<td>check</td>
<td>error</td>
<td>Record-count is not ok; is 218309 Should be: 0</td>
</tr>
<tr>
<td>employees.salaries</td>
<td>check</td>
<td>warning</td>
<td>Found 218309 key parts. Should be: 0</td>
</tr>
<tr>
<td>employees.salaries</td>
<td>check</td>
<td>error</td>
<td>Partition p13 returned error</td>
</tr>
<tr>
<td>employees.salaries</td>
<td>check</td>
<td>error</td>
<td>Corrupt</td>
</tr>
</tbody>
</table>

5 rows in set (1.56 sec)

```
# Partition p13 returned error

Sunday, 11 April 2010
Checking and repairing partitions - example fixing corruption

repair table salaries;

+--------------------+--------+----------+-----------------------------------------+
| Table              | Op     | Msg_type | Msg_text                                |
+--------------------+--------+----------+-----------------------------------------+
| employees.salaries | repair | warning  | Number of rows changed from 0 to 218309 |
| employees.salaries | repair | status   | OK                                      |
+--------------------+--------+----------+-----------------------------------------+
2 rows in set (8.31 sec)

ALTER table salaries REPAIR PARTITION p13;
Checking and repairing partitions - example checking partitions after fix

```
check table salaries;
```

```
+--------------------+-------+----------+----------+
| Table              | Op    | Msg_type | Msg_text |
+--------------------+-------+----------+----------+
| employees.salaries | check | status   | OK       |
+--------------------+-------+----------+----------+
1 row in set (2.07 sec)
```

```
# no errors
```

Sunday, 11 April 2010
QUIZ TIME

INSERT IGNORE INTO trickyTable (id) VALUES (1);
Query OK, 0 rows affected (0.00 sec)

SELECT * FROM trickyTable;
Empty set (0.00 sec)
Using partitions to create constraints

- That's what the quiz is about
Partitions as constraints - setting (quiz answer)

create table trickytable (id int)
partition by list (id)
(partition p1 values in (2));

#or

create table trickytable (id int)
partition by range (id)
(partition p1 values less than (1));
Partitions as constraints - fail

```
INSERT INTO trickytable (id) VALUES (1);
ERROR 1526 (HY000): Table has no partition for value 1
```
create table special_products (  
    part_no int not null,  
    category_id int not null,  
    primary key (part_no, category_id)  
) engine = InnoDB  
partition by list (category_id)  
(  
    partition p1 values in  
    (10, 25, 40, 53)  
);
Partitions as constraints - success and fail

insert into special_products values (1, 53), (2, 25);
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

insert into special_products values (1, 23);
ERROR 1526 (HY000): Table has no partition for value 23
Partitions as constraints (editing constraint)

```
alter table special_products
reorganize partition p1 into
 ( partition p1 values in
   (10, 23, 25, 40, 53));

alter table special_products
reorganize partition p1 into
 (partition p1 values
   in (10, 23, 25),
   partition p2 values in (40, 53));
```
MySQL 5.1 partitions

LEVERAGING REPLICAION
Replication schemes

- **MASTER**
  - INNODB NOT PARTITIONED
  - concurrent insert

- **SLAVE**
  - INNODB PARTITIONED BY RANGE
  - concurrent batch processing

- **SLAVE**
  - INNODB NOT PARTITIONED
  - concurrent read

- **SLAVE**
  - MyISAM PARTITIONED BY RANGE
  - large batch processing
Replication schemes

MASTER

INNODB
PARTITIONED
BY HASH

concurrent insert

SLAVE

MyISAM
PARTITIONED
BY RANGE

batch processing

SLAVE

INNODB
NON
PARTITIONED

concurrent reads

Sunday, 11 April 2010
Replication schemes - dimensions

MASTER

INNODB PARTITIONED BY HASH

concurrent insert

ARCHIVE PARTITIONED BY RANGE (date)

ARCHIVE PARTITIONED BY RANGE (locations)

dimensional processing

dimensional processing

SLAVE

SLAVE

SLAVE

ARCHIVE PARTITIONED BY RANGE (product)
MySQL 5.5 partitions

NEW FEATURES
MySQL 5.5 enhancements

- PARTITION BY RANGE COLUMNS
- PARTITION BY LIST COLUMNS
- TO_SECONDS
MySQL 5.5 enhancements

CREATE TABLE t (
    dt date
)
PARTITION BY RANGE (TO_DAYS(dt))
(
    PARTITION p01 VALUES LESS THAN (TO_DAYS('2007-01-01')),
    PARTITION p02 VALUES LESS THAN (TO_DAYS('2008-01-01')),
    PARTITION p03 VALUES LESS THAN (TO_DAYS('2009-01-01')),
    PARTITION p04 VALUES LESS THAN (MAXVALUE)
);
SHOW CREATE TABLE t \G
Table: t
Create Table: CREATE TABLE `t` ( `dt` date DEFAULT NULL ) ENGINE=MyISAM DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE (TO_DAYS (dt)) (PARTITION p01 VALUES LESS THAN (733042) ENGINE = MyISAM, [...]]
MySQL 5.5 enhancements

CREATE TABLE t (  
    dt date  
)  
PARTITION BY RANGE COLUMNS (dt)  
(  
    PARTITION p01 VALUES LESS THAN ('2007-01-01'),  
    PARTITION p02 VALUES LESS THAN ('2008-01-01'),  
    PARTITION p03 VALUES LESS THAN ('2009-01-01'),  
    PARTITION p04 VALUES LESS THAN (MAXVALUE)  
);
MySQL 5.5 enhancements

SHOW CREATE TABLE t
Table: t
Create Table: CREATE TABLE `t` (`dt` date DEFAULT NULL ) ENGINE=MyISAM DEFAULT CHARSET=latin1
/*!50500 PARTITION BY RANGE COLUMNS (dt)
(PARTITION p01 VALUES LESS THAN ('2007-01-01') ENGINE = MyISAM,
[...]
CREATE TABLE t ( 
  a int,
  b int
) PARTITION BY RANGE COLUMNS (a,b) 
  ( 
    PARTITION p01 VALUES LESS THAN (10,1),
    PARTITION p02 VALUES LESS THAN (10,10),
    PARTITION p03 VALUES LESS THAN (10,20),
    PARTITION p04 VALUES LESS THAN (MAXVALUE, MAXVALUE)) ;
### Partition Definition by Range (a, b)

<table>
<thead>
<tr>
<th>partition</th>
<th>LESS THAN</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
<td>MAXVALUE</td>
<td>MAXVALUE</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
<td>MAXVALUE</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>records</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>
### Partition Definition by Range \((a, b)\)

<table>
<thead>
<tr>
<th>partition</th>
<th>LESS THAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>10</td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>

**(1,10) < (10,10) ?**

\[(a < 10) \quad \text{OR} \quad ((a = 10) \text{ AND } (b < 10))\]

\[(1 < 10) \quad \text{OR} \quad ((1 = 10) \text{ AND } (10 < 10))\]

**TRUE**
(1, 10) < (10, 10) ?

(a < 10)
OR
((a = 10) AND (b < 10))

(1 < 10)
OR
((1 = 10) AND (10 < 10))

TRUE
(10,9) < (10,10) ?

(a < 10)
OR
((a = 10) AND (b < 10))

(10 < 10)
OR
((10 = 10) AND (9 < 10))

TRUE
(10,9) < (10,10) ?

(a < 10)
OR
((a = 10) AND (b < 10))

TRUE

(10 < 10)
OR
((10 = 10) AND (9 < 10))
partition definition by range \((a, b)\)

<table>
<thead>
<tr>
<th>partition</th>
<th>LESS THAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>10</td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>

\[(10,9) < (10,10) \?\]

\[(a < 10)\]
\[\text{OR}\]
\[((a = 10) \text{ AND } (b < 10))\]

\[(10 < 10)\]
\[\text{OR}\]
\[((10 = 10) \text{ AND } (9 < 10))\]

\[\text{TRUE}\]
**Partition Definition**

**By Range (a,b)**

<table>
<thead>
<tr>
<th>partition</th>
<th>LESS THAN</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
<td>MAXVALUE</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>

**Records**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td><strong>10</strong></td>
<td><strong>10</strong></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

**Question:** 

\[(10,10) \leq (10,10)\]?

\[(a < 10) \text{ \ OR \ } ((a = 10) \text{ AND } (b < 10))\]

\[(10 < 10) \text{ \ OR \ } ((10 = 10) \text{ AND } (10 < 10))\]

**FALSE**
(10,10) < (10,10) ?

(a < 10)
OR
((a = 10) AND (b < 10))

(10 < 10)
OR
((10 = 10) AND (10 < 10))

FALSE
(10,10) < (10,10) ?

(a < 10)
OR
((a = 10) AND (b < 10))

(10 < 10)
OR
((10 = 10) AND (10 < 10))

FALSE
<table>
<thead>
<tr>
<th>records</th>
<th>partition</th>
<th>LESS THAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>p01</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td><strong>10</strong></td>
<td><strong>10</strong></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

**partition definition by range \((a,b)\):**

\[(10,10) < (10,20)\]?

\[(a < 10)\]

**OR**

\[((a = 10) \text{ AND } (b < 20))\]

\[(10 < 10)\]

**OR**

\[((10 = 10) \text{ AND } (10 < 20))\]

**TRUE**
partitions definition by range \((a,b)\)

<table>
<thead>
<tr>
<th>partition</th>
<th>(\text{LESS THAN})</th>
</tr>
</thead>
<tbody>
<tr>
<td>p01</td>
<td>10</td>
</tr>
<tr>
<td>p02</td>
<td>10</td>
</tr>
<tr>
<td>p03</td>
<td>10</td>
</tr>
<tr>
<td>p04</td>
<td>10</td>
</tr>
<tr>
<td>p05</td>
<td>MAXVALUE</td>
</tr>
</tbody>
</table>

Records:

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

\((10,10) < (10,20)\) ?

\((a < 10)\)

\(\text{OR}\)

\(((a = 10) \text{ AND } (b < 20))\)

\((10 < 10)\)

\(\text{OR}\)

\(((10 = 10) \text{ AND } (10 < 20))\)

TRUE
(10,10) < (10,20) ?

(a < 10)
OR
((a = 10) AND (b < 20))

(10 < 10)
OR
((10 = 10) AND (10 < 20))

TRUE
CREATE TABLE employees (  
  emp_no int(11) NOT NULL,  
  birth_date date NOT NULL,  
  first_name varchar(14) NOT NULL,  
  last_name varchar(16) NOT NULL,  
  gender char(1) DEFAULT NULL,  
  hire_date date NOT NULL  
) ENGINE=MyISAM
PARTITION BY RANGE  
  COLUMNS(gender,hire_date)
(PARTITION p01 VALUES LESS THAN ('F','1990-01-01'),  
  PARTITION p02 VALUES LESS THAN ('F','2000-01-01'),  
  PARTITION p03 VALUES LESS THAN ('F',MAXVALUE),  
  PARTITION p04 VALUES LESS THAN ('M','1990-01-01'),  
  PARTITION p05 VALUES LESS THAN ('M','2000-01-01'),  
  PARTITION p06 VALUES LESS THAN ('M',MAXVALUE),  
  PARTITION p07 VALUES LESS THAN (MAXVALUE,MAXVALUE)
MySQL 5.5 enhancements

• TRUNCATE PARTITION
• TO_SECONDS()
Updated presentation slides

- you will find an up to date copy of these slides at:
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.
THANKS!

Question time

Comment on Twitter: @datacharmer

Sunday, 11 April 2010