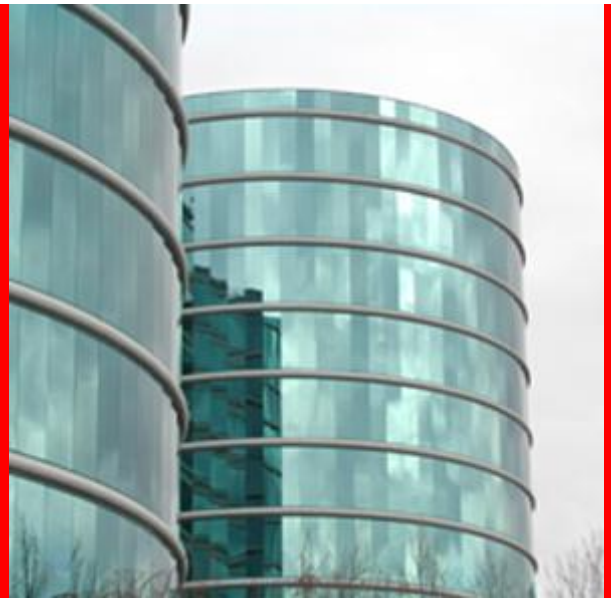


ORACLE

MySQL

Practical Partitioning

Alexander Rubin, Senior Consultant
Brian Miezejewski, Senior Manager

Copyright Oracle 2010

About MySQL

- Founded, first release in 1995
- Acquired by Sun in February 2008
- Acquired by Oracle in January 2010
- #1 Most Popular Open Source Database
- MySQL 5.5 RC
- Market-leading customers

Oracle's Plans for MySQL

- Complete Oracle's stack
- MySQL Global Business Unit
  - Managed by Edward Screven, Chief Corporate Architect
- Invest in MySQL!
  - "Make MySQL a Better MySQL"
  - Develop, Promote and Support MySQL
    - Improve engineering, consulting, and support
    - MySQL Sunday at Oracle Open World
    - Leverage World-Wide, 24x7 Oracle Support
- MySQL Community Edition
  - Source and binary releases
  - GPL license

MySQL 5.1 to 5.5 (RC) Improvements

- InnoDB becomes default
- Improved Availability, Improved Usability
- Better Instrumentation / Diagnostics
- InnoDB & MySQL Performance Improvements
  - More than 10x Improvement in recovery times

- Sysbench Results:
  - Linux: MySQL 5.5 vs. 5.1 - Read Only = 200%
  - Linux: MySQL 5.5 vs. 5.1 – Read Write = 369%
  - Windows: MySQL 5.5 vs. 5.1 - Read Only = 538%
  - Windows: MySQL 5.5 vs. 5.1 - Read Write = 1561%

Industry-Leading Customers

Contents

- What is Partitioning?
- Why Partition?
- Partitioning 101
  - Types of partitioning in MySQL
  - Managing Partitions
- Partitions and Indexes
- Short Term Rolling Data
- Long Term Rolling Data
- Optimize, Analyze, etc. by Partition
- New in MySQL 5.5
- Q&A

ORACLE

Copyright Oracle 2010

What is Partitioning?

- Partitioning divides a table into smaller parts called “partitions”
- Partitions are defined in a CREATE or ALTER

```
CREATE TABLE Sales ( saleDate date, ... )
       PARTITION BY KEY(saleDate)
       PARTITIONS 16 ;
```

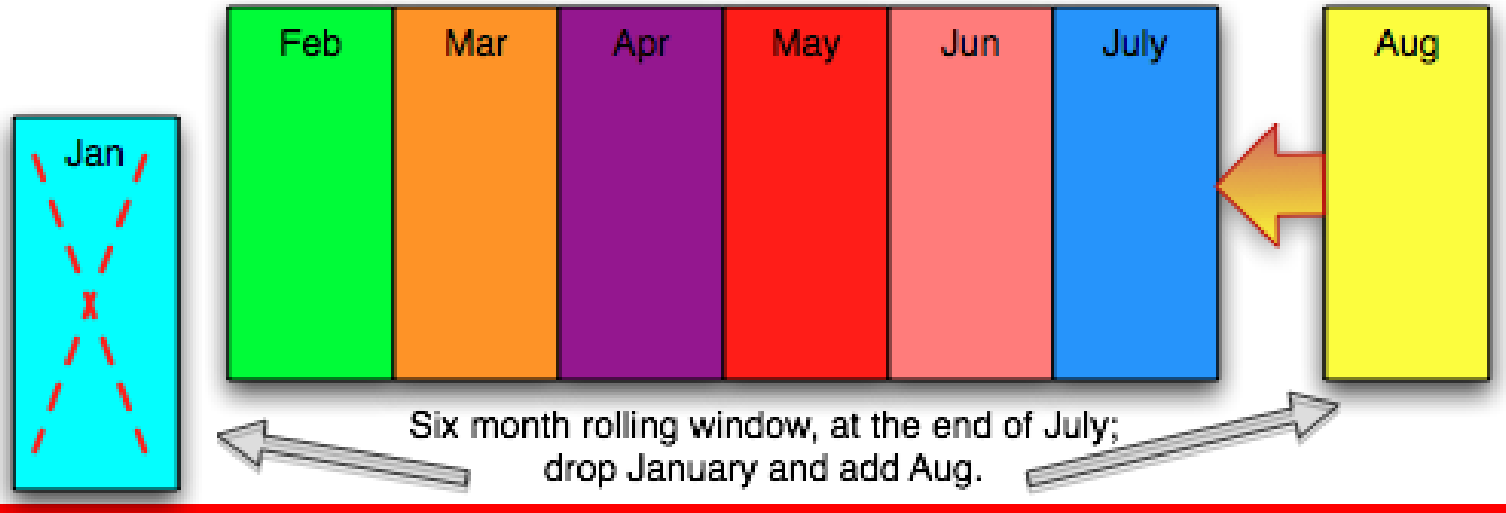
- MySQL knows how the table was divided into smaller parts and uses this information to speed up queries
- Operations on many smaller parts are often faster than on one big table
  - Optimize
  - Create INDEX, etc.

ORACLE

Copyright Oracle 2010

Why Partition?
1) Deleting Data by Partition

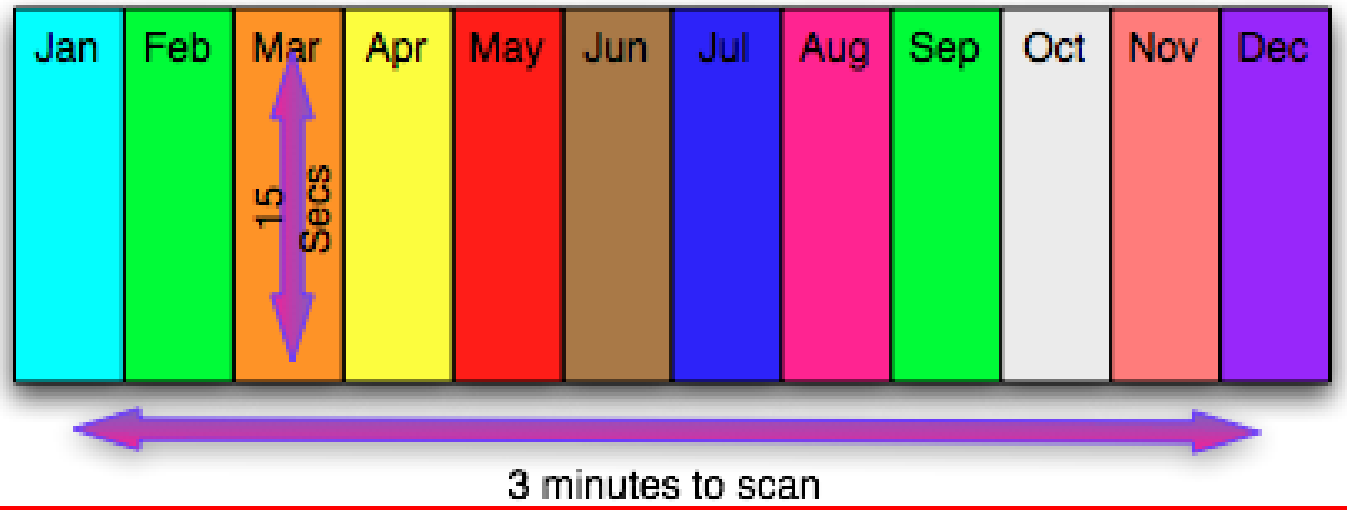
- Partitions can support deleting data by dropping a partition
  - If you insert 1,000,000 rows a day, eventually you need to delete 1,000,000 rows a day
- Only works with **Range** and **List** partitioning
- Very useful for rolling date/time range
- Can be very useful even for small increments, i.e.1 hour
- Very fast, deleting a file, can be ~ 1-2 Secs



Copyright Oracle 2010

Why Partition?
2) Faster non-index Data Access (Pruning or Partition Elimination)

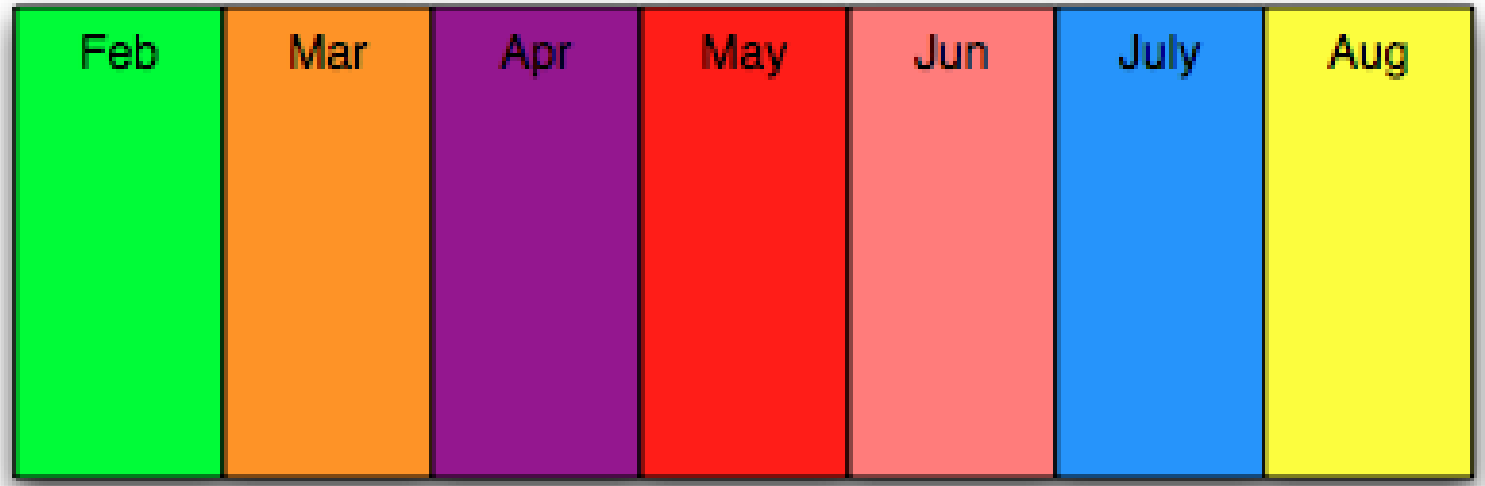
- The MySQL optimizer is aware of the partitioning expression and can eliminate partitions to scan when the columns used in the partitioning expression are in the queries where condition
- Reduce or even **eliminate** indexes!



Copyright Oracle 2010

Why Partition?
3) Some operations are faster

- Adding indexes can be faster
- Optimizations can be done by partition
  - If data is only being added to one partition then you can OPTIMIZE *only that partition* instead of running OPTIMIZE on the whole table



Copyright Oracle 2010

Types of MySQL Partitioning

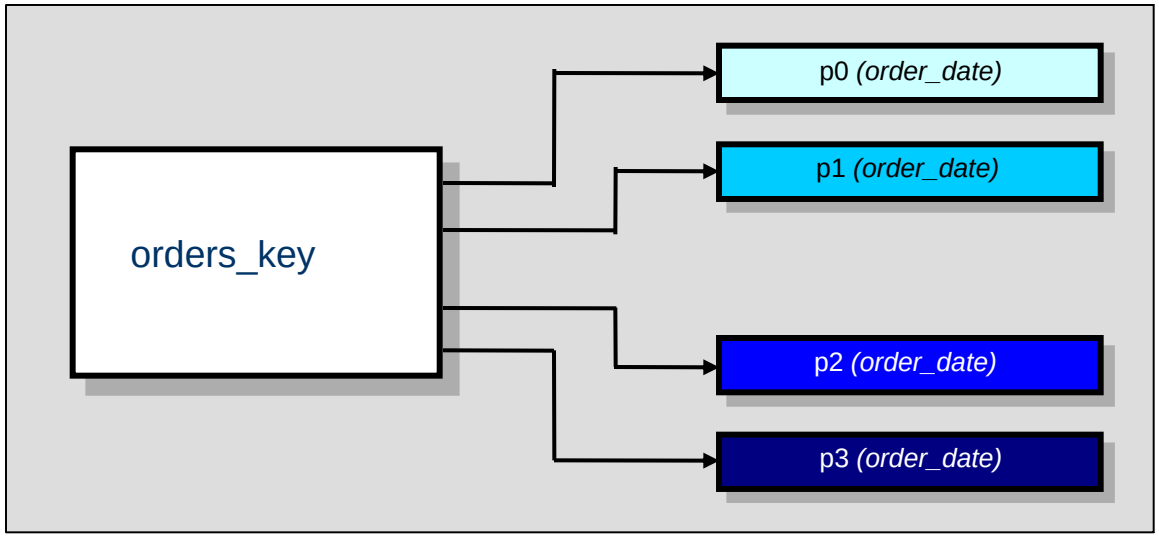
- **Key/Hash - Not useful for Deleting by partition**
  - **Key**(***column list***) - Uses internal hash function
  - **Hash**(**INT *expr***) - Mod on user expression
  - Both very easy to define and maintain
- **List/Range - Supports Deleting by Partition**
  - **List**(**INT *expr***) - IN list partitioning
    - List of IN (A, ..., N) expressions
  - **Range**(**INT *expr***) - Range expressions
    - list of less than expressions
- **List** and **Range** can be sub-partitioned by **Key** or **Hash**

ORACLE

Copyright Oracle 2010

Key Partitioning

- Uses internal hash function
- `CREATE TABLE Sales ( order_date date, ... )`
  `PARTITION BY` KEY(order_date)
  `PARTITIONS 4 ;`



Copyright Oracle 2010

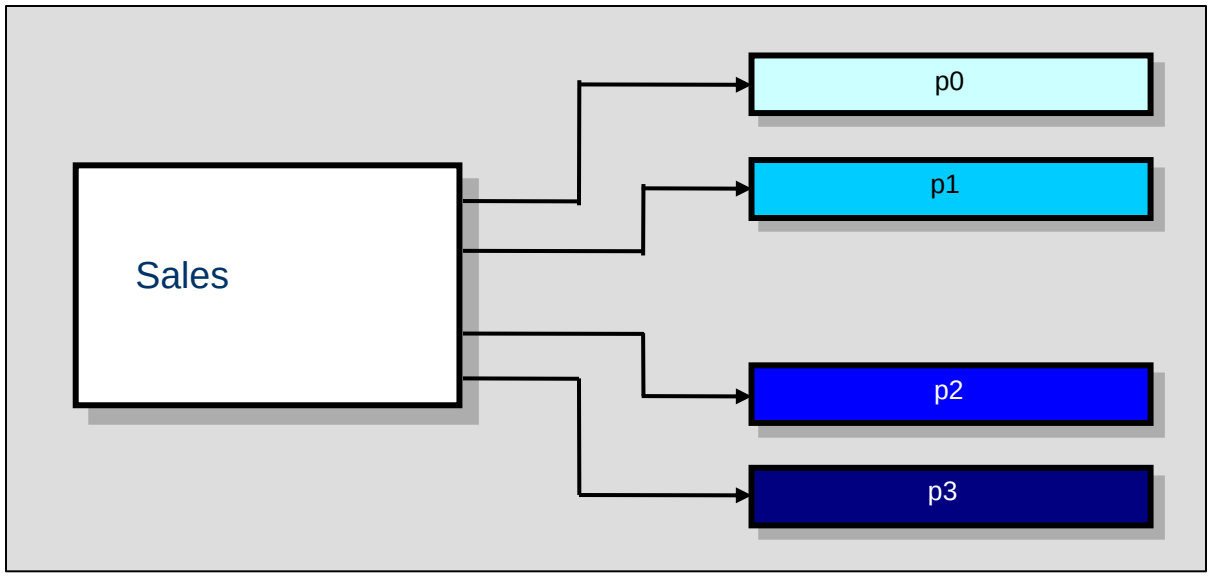
Partitioning Expressions

- **HASH**, **RANGE** and **LIST** can use expressions
- Expression **must** return an ***integer*** (**fixed in 5.5**) or NULL and can only use certain ***built-in functions***
  - ABS, CEILING, DAY, DAYOFMONTH, DAYOFWEEK,
  - DAYOFMONTH, DATEDIFF, EXTRACT, FLOOR, HOUR,
  - MICROSECEND, MINUTE, MOD, MONTH, QUARTER, SECOND,
  - TIME_TO_SEC, TO_DAYS, WEEKDAY, YEAR, YEARWEEK
- Not Allowed!
  - nested function calls, declared or user variables, stored functions or UDRs!
- http://dev.mysql.com/doc/refman/5.1/en/partitioning-limitations-functions.html

Copyright Oracle 2010

Hash Partitioning

- Easy to define like key, but supports expressions
- `CREATE TABLE Sales ( SaleDate date, ... )`
  `PARTITION BY HASH(MONTH(SaleDate))`
  `PARTITIONS 12 ;`



ORACLE

Copyright Oracle 2010

Managing KEY and HASH Partitions

- Both operations change the number of partitions, ***but keep the same number of records***
- ADD - Adds more partitions and redistribute the data
  - `ALTER TABLE Sales ADD PARTITION PARTITIONS 4 ;`
  - Adds 4 more partitions to `Sales`
  - If `Sales` had 16 partitions before, it now has 20
- COALESCE - Merges partitions
  - `ALTER TABLE Sales COALESCE PARTITION 6 ;`
  - Removes 6 partitions from `Sales`
  - If `Sales` had 16 partitions before, it now has 10

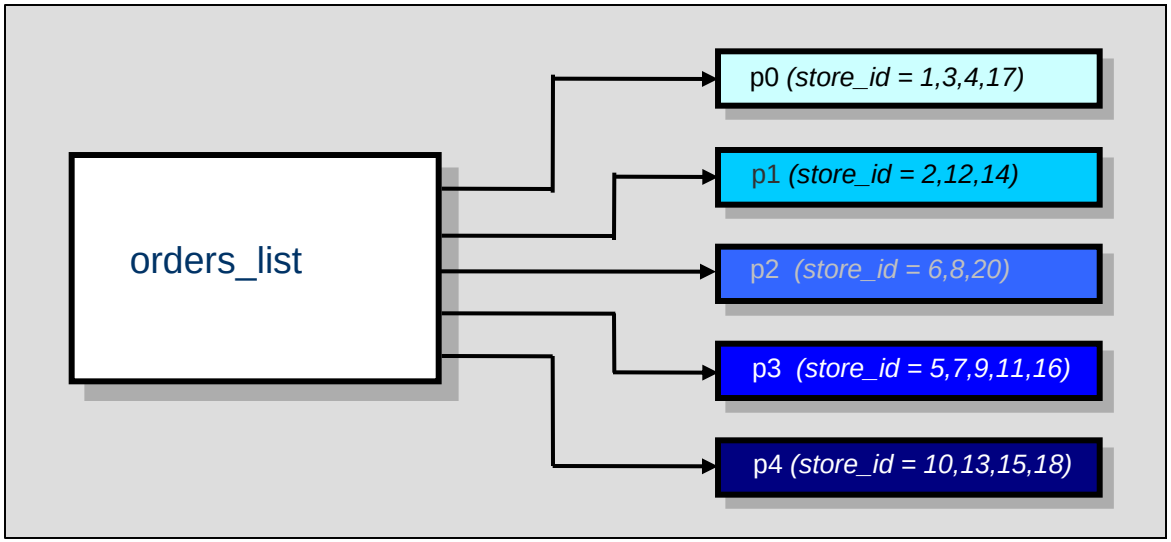
ORACLE

Copyright Oracle 2010

List Partitioning

- Each partition is defined by an IN LIST

```
CREATE TABLE orders_list (
     order_id int,
     order_date date,
     ... )
PARTITION BY LIST(DAYOFWEEK(order_date)) (
 PARTITION wend VALUES IN (1,7),PARTITION mon VALUES IN (2),
 PARTITION tue VALUES IN (3),PARTITION wed VALUES IN (4),
 PARTITION thr VALUES IN (5),PARTITION fri VALUES IN (6)  );
```



Copyright Oracle 2010

Range Partitioning

- Ranges **must be defined in order, lowest to highest**
- Cannot insert a record outside of the defined ranges
- Ranges must not overlap
- Note that you cannot add a value larger than the highest range
  - Use less than (maxval).

```
CREATE TABLE Sales ( id int, saleDate date, ... )
   PARTITION BY RANGE(YEAR(SaleDate)) (
   PARTITION p199X VALUES LESS THAN (2000),
   PARTITION p2003 VALUES LESS THAN (2004),
   ...
   PARTITION p2007 VALUES LESS THAN (2008),
   PARTITION p2008 VALUES LESS THAN (2009),
   PARTITION p2009 VALUES LESS THAN (2010)
   );
```

ORACLE

Copyright Oracle 2010

Managing List and Range Partitions

- **Add** - Add empty partitions

```
ALTER TABLE Sales ADD PARTITION (
    PARTITION p2011 VALUES IN (2011) ) ;
```

- **Drop** - Deletes the data in the partitions (not in NDB)

•Very fast!

•Requires DROP privilege

•Number rows dropped ***is not*** returned by server!

```
ALTER TABLE Sales DROP PARTITION p2003 ;
```

- **Reorganize** - Change the partitioning without losing data

•Can be used to split, merge, or change all partitions

•Reorganizes the data into the newly defined partitions

```
ALTER TABLE geoL REORGANIZE
    PARTITION p2002,p2004,p2003 INTO (
    PARTITION p20024 VALUES IN (2002,2003,2004)) ;
```

ORACLE

Copyright Oracle 2010

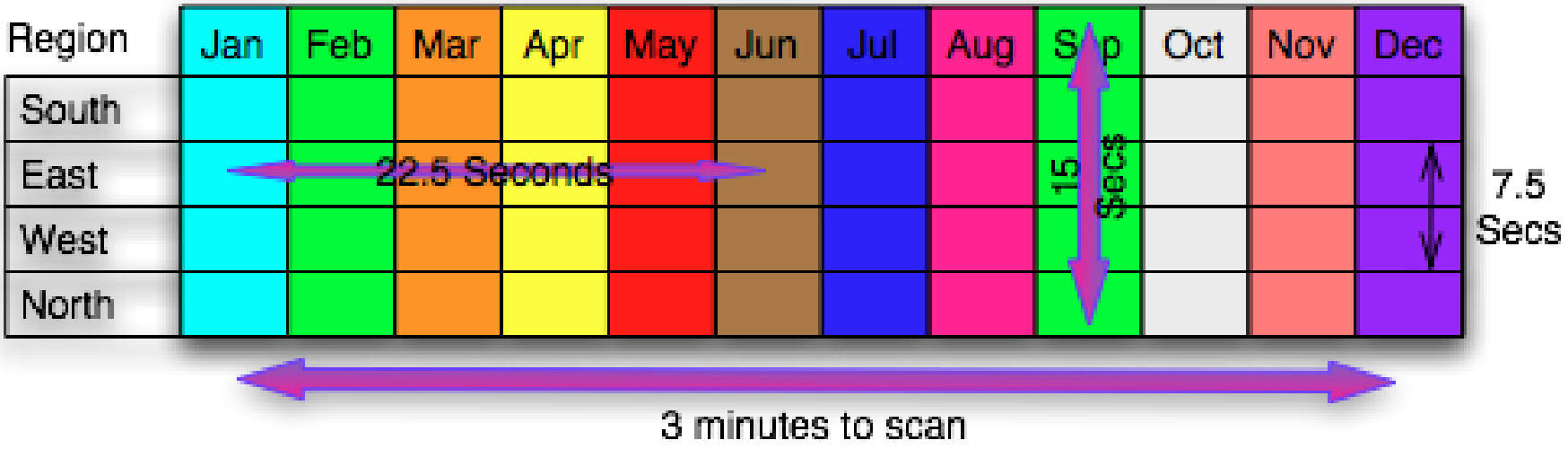
Rebuilding Partitions

- All Work with Partitioned tables
  - `REBUILD, CHECK, OPTIMIZE, ANALYZE and REPAIR`
- Examples:
  - `ALTER TABLE Sales REBUILD PARTITION P1,P2 ;`
  - `ALTER TABLE Inv OPTIMIZE PARTITION I4 ;`
- Smaller partitions make the above operations faster
  - Original Table 10 minutes
  - 16 way Partitioned table might be 10-15 seconds per partition

Copyright Oracle 2010

Sub-partitioning

- Range and List partitioning can be sub partitioned with
  - key
  - hash
- Range by Month sub-partitioned by region



Copyright Oracle 2010

Partition pruning

- Pruning happens when the MySQL optimizer *only* references partitions needed for a particular query
  - Optimizer's partition pruning mechanism provides performance increase
  - *"Do not scan partitions where there can be no matching values."*
- Example of **RANGE** pruning
  - Using the **orders_range** table, with the following partitions

```
+----------------+-------------------------+
| PARTITION_NAME | PARTITION_DESCRIPTION   |
+----------------+-------------------------+
| p0             | < 10000                 |
| p1             | < 20000                 |
| p2             | < 30000                 |
| p3             | < 40000                 |
| p4             | < 50000                 |
+----------------+-------------------------+
```

Copyright Oracle 2010

Partition pruning example

- See the effects of pruning using an **EXPLAIN PARTITIONS** statement

```
mysql> EXPLAIN PARTITIONS SELECT * FROM orders_range
    -> WHERE id > 19997 AND id < 20003 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: orders_range
   partitions: p1,p2
         type: system
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 2
        Extra:
1 row in set (0.06 sec)
```

- The partitions `p0`, `p3`, and `p4` are not scanned since they do not contain any of the values from the range of the query

Copyright Oracle 2010

Unique Indexes and Partitioning

- Every column used in a partitioning expression for a table ***must*** be part of every unique key on that table
  - This does not mean you must have unique keys,
  - If you do, then every one of them must include all of the values used in the partitioning expression!

- Partitioning column(s) can appear anywhere in the unique index:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3) )
PARTITION BY HASH(col3)
PARTITIONS 4;
```

ORACLE

Copyright Oracle 2010

Non-Unique Indexes

- You can always have non-unique indexes on a partitioned table.
- The partition engine will execute a separate non-parallel index lookup ***on each partition !***
- Performance may be OK with a very small number (4) of partitions, but gets really bad with large numbers of partitions
- If you must have non-unique indexes, keep the number of partitions low (<16)
- Added, and dropping them is faster

ORACLE

Copyright Oracle 2010

Multi-Query Non-Unique Index Performance with Partitions Table

(Simple index scan of ~1000 records on laptop)

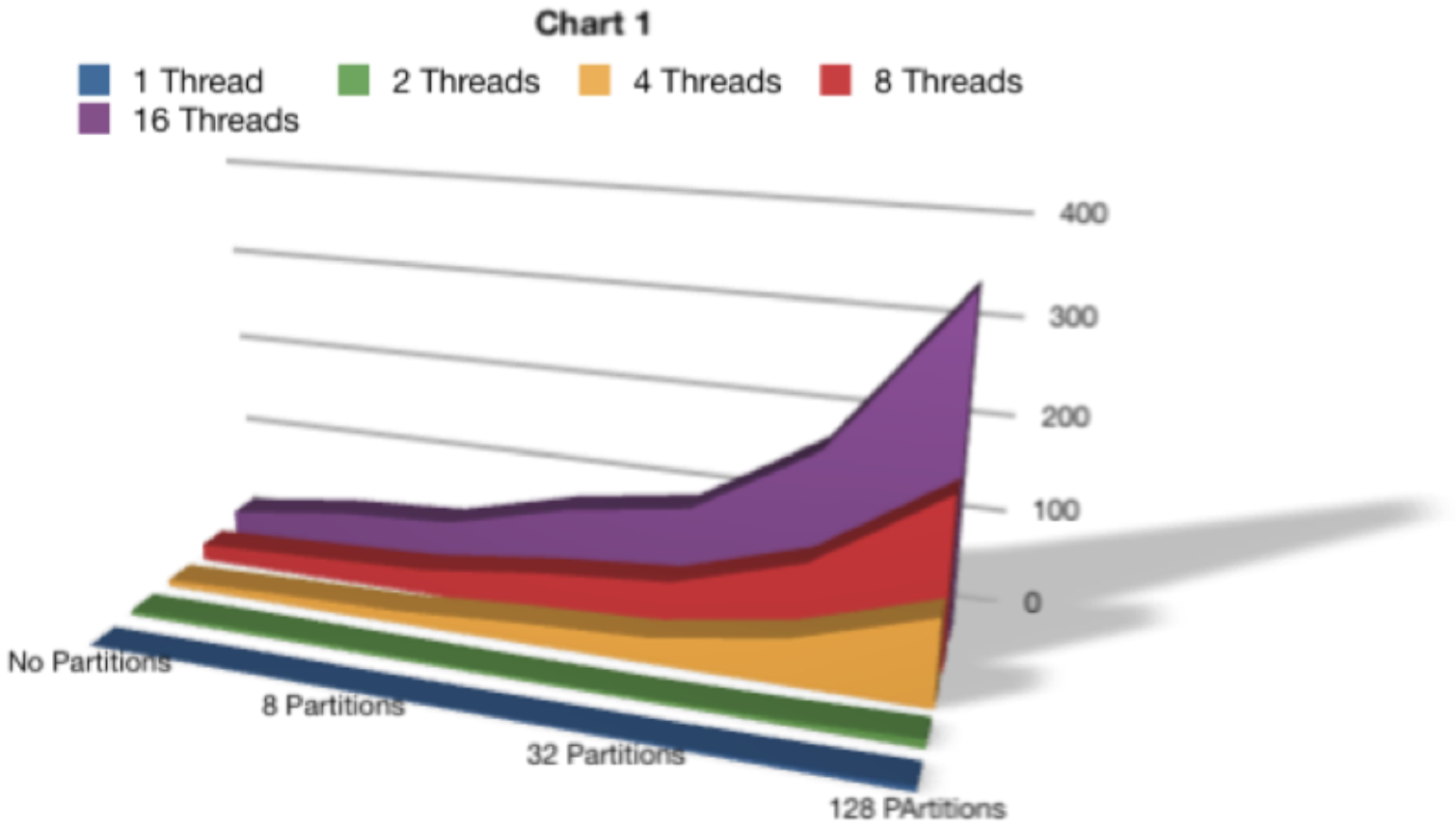
	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
No Partitions	0.47	4.43	6.93	17.63	28.25
4 Partitions	1.25	5.13	10.63	22.25	44.37
8 PArtitions	1.25	4.32	13.11	26.87	51.35
16 Partitions	1.67	3.62	21.20	42.57	83.16
32 Partitions	2.29	3.62	26.88	52.84	102.20
64 Partitions	3.85	5.42	48.70	92.44	180.23
128 PArtitions	5.99	9.06	90.57	179.64	348.28

ORACLE

Copyright Oracle 2010

Multi-Query Non-Unique Index Performance with Partitions Chart

```
select count(*) from geo where population between 1100000 and 1500000
```



Copyright Oracle 2010

Indexes and Partitioning

- A well designed partitioned table has few or even **NO** indexes!
  - ***Should always have less than the non-partitioned table!***
- Need to re-design indexes!
- Too many partitions over 124 start to really slow down non-unique index lookups

Copyright Oracle 2010

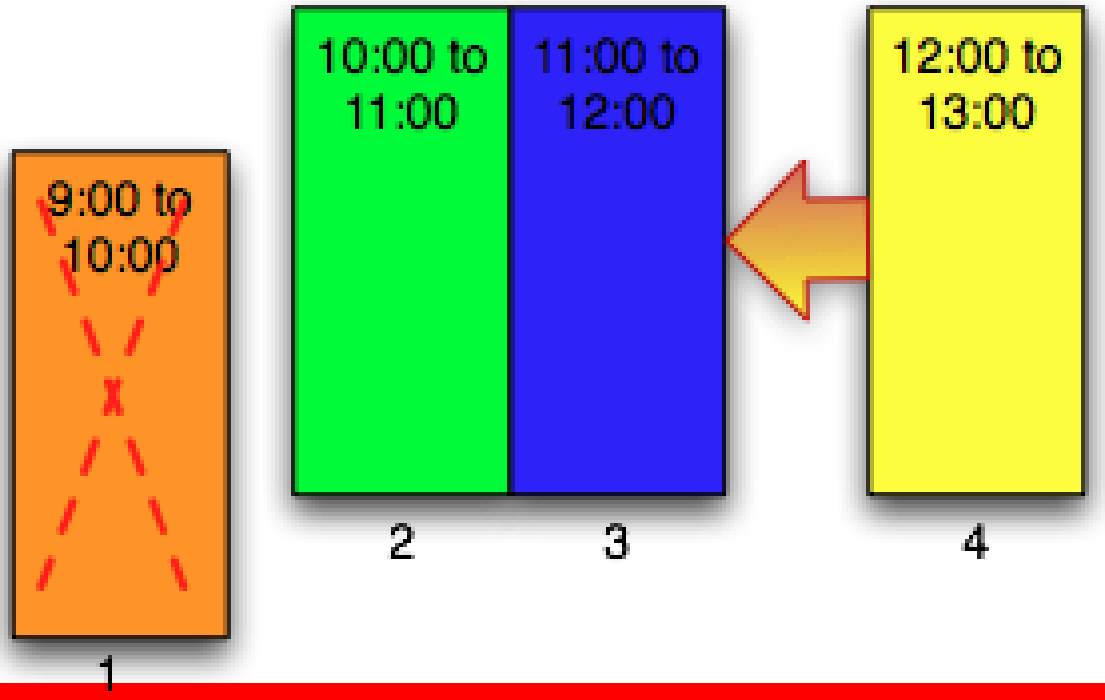
Short Term Rolling Data

- **Goal:**
  - Reduce or eliminate **delete overhead**
- **When to Use:**
  - Data only needs to be kept for a few hours or days
  - Not a lot of data, indexes work fine for lookups
  - Its hard to balance the deletes against other operations
- **Steps**:
  - Create a table (**LIST** or **RANGE**) with at least three partitions
  - Let partitions 1 and 2 fill
  - Drop part 1 and add part 3
- Goal: **not for select query performance**, only for **fast deletes**

Copyright Oracle 2010

Short Term Rolling Data

- Example: Session data is only needed for 1 hour after it is created.
  - If you create 10,000 sessions an hour you also need to delete 10,000 session an hour.
- Range Partition with hourly partitions on “CreateTime”



Copyright Oracle 2010

Short Term Rolling

- Table:

```
CREATE TABLE session (
    CreateTime time,
    SessionData varchar(2048) )
PARTITION BY LIST (hour(CreateTime))(
     PARTITION p09 VALUES IN (9),
     PARTITION p10 VALUES IN (10),
     PARTITION p11 VALUES IN (11) ) ;
```

- Add a partition for the 12:00 to 13:00 Sessions

```
alter table session add partition (
    partition p12 values in (12) ) ;
```

- Drop the 9:00 to 10:00 Sessions

```
alter table session drop partition p09 ;
```

ORACLE

Copyright Oracle 2010

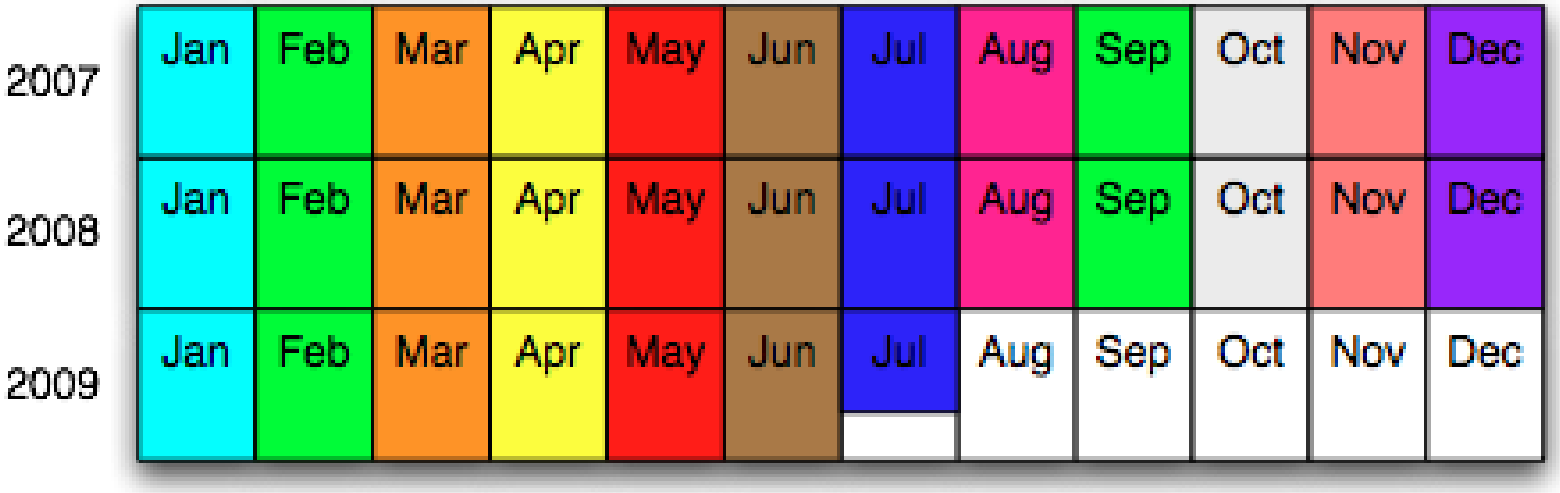
Long Term Rolling Data

- **Goals:**
  - ***Reduce optimize and other maintenance overhead***
  - ***Eliminate delete overhead***
  - Speed up selects and inserts
- **When to use:**
  - Very large tables 100 GB or more, **too big for cache**
  - **date column** or similar to partition on
  - Most of your queries filter on the partitioning column
  - Most queries currently do **large index scans**
  - Inserts are getting too slow
  - **Optimize, add index**, etc. are taking far too long
- **Steps:**
  - Partition the table into many (usually 32+) partitions
  - Roll N partitions out for each N you add.

Copyright Oracle 2010

Long Term Rolling Data

- More traditional Data Warehouse usage
- Avoid Index = Partitioning column
- Only the active month needs optimization, etc.
- Data can be deleted by month, quarter, or year



Copyright Oracle 2010

Long Term Rolling Data

```
CREATE TABLE Sales (
       salesDate    TIMESTAMP,
       storeID      smallint,
       regionID     tinyint,
       amount       decimal(10,2)
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(SalesDate) ) (
    PARTITION p200701 VALUES LESS THAN ( UNIX_TIMESTAMP('2007-02-01 00:00:00') ),
    PARTITION p200702 VALUES LESS THAN ( UNIX_TIMESTAMP('2007-03-01 00:00:00') ),
   ...
    PARTITION p200911 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-12-01 00:00:00') ),
    PARTITION p200912 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') )
);
```

- Add new partitions

```
alter table Sales add partition (
  PARTITION p201001 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-02-01 00:00:00')),
  ...
  PARTITION p201012 VALUES LESS THAN ( UNIX_TIMESTAMP('2011-01-01 00:00:00')) ) ;
```

- Drop old partitions for Jan 2007 to Dec 2007

```
alter table sales drop partition p200701, p200702, ... , p200712 ;
```

Copyright Oracle 2010

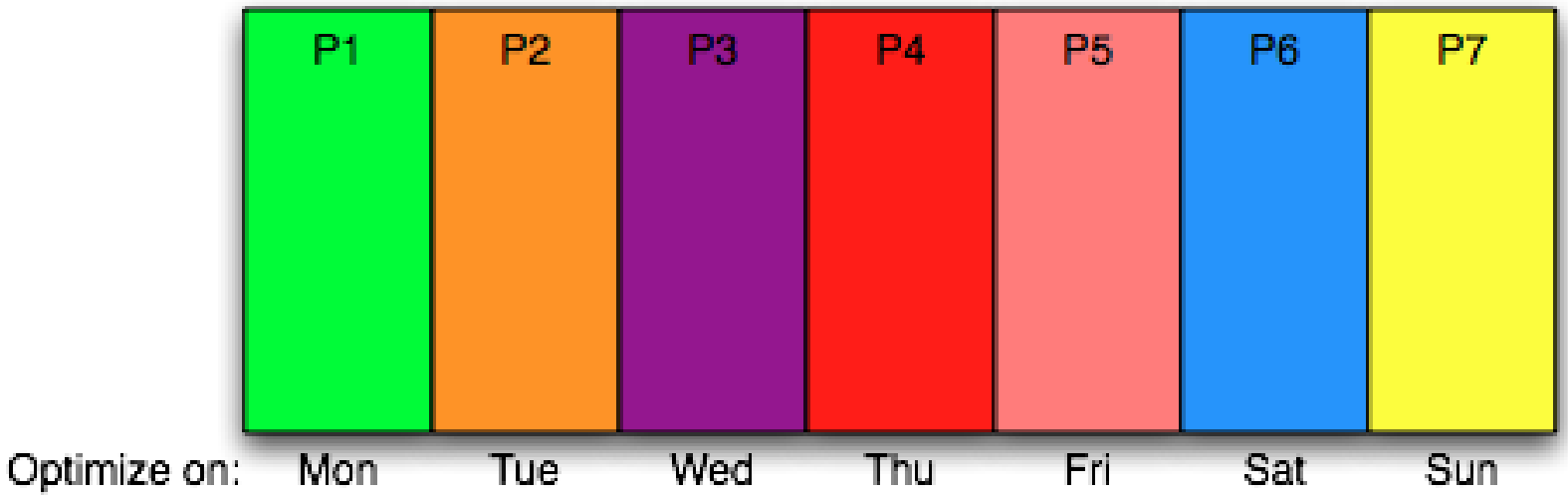
Optimize, Analyze, etc. by Partition

- **Goals:**
  - Reduce optimize, analyze, etc. overhead
- **When to use:**
  - Optimize, Analyze, etc. takes too long and is needed
  - A table has a lot of inserts, updates and deletes
  - For medium sized tables, around 1-10GB
  - You have a unique column you can HASH on.
- **Steps:**
  - Partition the table into partitions using HASH on an ID or similar
  - Optimize, Analyze, etc. 1 partition a night or as needed
- Can turn Hour+ process to 5-10 minutes a night

Copyright Oracle 2010

Optimize, Analyze, etc. by Partition

- Insert, Update, and Delete as usual
- Fix indexes if needed
- Try to keep to 16 partitions or less, 8 or less best
- **Cycle Optimize, Analyze**, etc. through the partitions



Copyright Oracle 2010

Optimize, Analyze, etc. by Partition

- Original Table definition

```
CREATE TABLE Sale (
       saleID        INT AUTO_INCREMENT PRIMARY KEY,
       salesDate     TIMESTAMP,
       storeID       smallint,
       amount        decimal(10,2)
);
```

- Add partitions

```
Alter table Sale Partition by hash(saleID) partitions 7 ;
```

- Optimize the first partition (Partitions are P0 to P6)

```
alter table sale optimize partition P0 ;
```

ORACLE

Copyright Oracle 2010

Partitioning by date in MySQL 5.1

The MySQL optimizer will recognize 2 date-based functions for partition pruning purposes:

- TO_DAYS()
- YEAR()

```
mysql> CREATE TABLE part_date3
    ->(c1 int default NULL, c2 varchar(30) default NULL,
    ->  c3 date default NULL)
    ->      partition by range (to_days(c3))
    -> (PARTITION p0 VALUES LESS THAN (to_days('1995-01-01')),
    -> PARTITION p1 VALUES LESS THAN (to_days('1996-01-01')),
…
    -> PARTITION p11 VALUES LESS THAN MAXVALUE );
```

http://dev.mysql.com/tech-resources/articles/mysql_5.1_partitioning_with_dates.html

Copyright Oracle 2010

MySQL 5.5 partitioning enhancements

Use dates directly (partition pruning will work)

- MySQL 5.1:

```
PARTITION BY RANGE (TO_DAYS(dt))
PARTITION p01 VALUES LESS THAN (733042),
 PARTITION p02 VALUES LESS THAN (733407),
 PARTITION p03 VALUES LESS THAN (733773),
 PARTITION p04 VALUES LESS THAN MAXVALUE)
```

- MySQL 5.5:

```
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));
```

Copyright Oracle 2010

MySQL 5.5 partitioning enhancements

- Partitioning expression do not have to return integer
- **COLUMNS** keyword instead of integer:

```
PARTITION BY LIST COLUMNS (category)
(
  PARTITION p01 VALUES IN ('lodging', 'food'),
  PARTITION p02 VALUES IN ('flights',
                'ground transportation'),
  PARTITION p03 VALUES IN ('leisure',
                'customer entertainment'),
  PARTITION p04 VALUES IN ('communications'),
  PARTITION p05 VALUES IN ('fees')
);
```

Copyright Oracle 2010

MySQL 5.5 partitioning enhancements

- Other enhancements
  - `ALTER TABLE t1 TRUNCATE PARTITION p0;`
  - multiple columns

```
PARTITION BY RANGE COLUMNS (a,b)
(
  PARTITION p01 VALUES LESS THAN (10,20),
  PARTITION p02 VALUES LESS THAN (20,30),
  PARTITION p03 VALUES LESS THAN (30,40),
  PARTITION p04 VALUES LESS THAN (40,MAXVALUE),
  PARTITION p05 VALUES LESS THAN
  (MAXVALUE,MAXVALUE)
);
```

- http://dev.mysql.com/tech-resources/articles/mysql_55_partitioning.html

Copyright Oracle 2010

Q & A : Prequel

- Is Partitioning with Partition Elimination always faster than using an Index?
  - No, many queries are much faster with indexes
- When is Partitioning faster?
  - When the index scan would have scanned 10-20% or more of the non-Eliminated partitions
  - Typically reporting queries
- Why is Partitioning faster in this case?
  - Because a table scan (used by partition based queries) is faster than an index scan on a row by row basis.
  - This advantage is multiplied when all of the data will not fit into cache.

Copyright Oracle 2010

Q & A : Prequel

- Can a partitioned table have NO INDEXES?
  - Yes in many cases
  - Best practice for very large tables (fact tables)
  - Use Memory engine for Dimensions

Copyright Oracle 2010

Resources and Q&A

- "Guide to MySQL Partitioning" white paper
  - http://mysql.com/why-mysql/white-papers/mysql_wp_partitioning.php
- "Getting the Best MySQL Performance in Your Products: Part 3, Query Tuning" webinar
  - **November 23**: http://mysql.com/news-and-events/web-seminars/display-582.html
  - Part 1, The Fundamentals: http://mysql.com/news-and-events/on-demand-webinars/display-od-552.html
  - Part 2, Beyond the Basics: http://mysql.com/news-and-events/on-demand-webinars/display-od-567.html
- ISV / OEM Resources
  - http://mysql.com/why-mysql/isv-oem-corner/
- Questions?
  - http://www.mysql.com/about/contact/sales.html?s=oem
  - Phone: USA=+1-866-221-0634; Outside USA = +1-208-327-6494

The presentation 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.