ORACLE

Practical Partitioning

Brian Miezejewski
Senior Manager

Copyright Oracle 2010

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
- Sub-Partitioned Tables
- Q&A

ORACLE

Copyright Oracle 2010

What is Partitioning?

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 ;
```

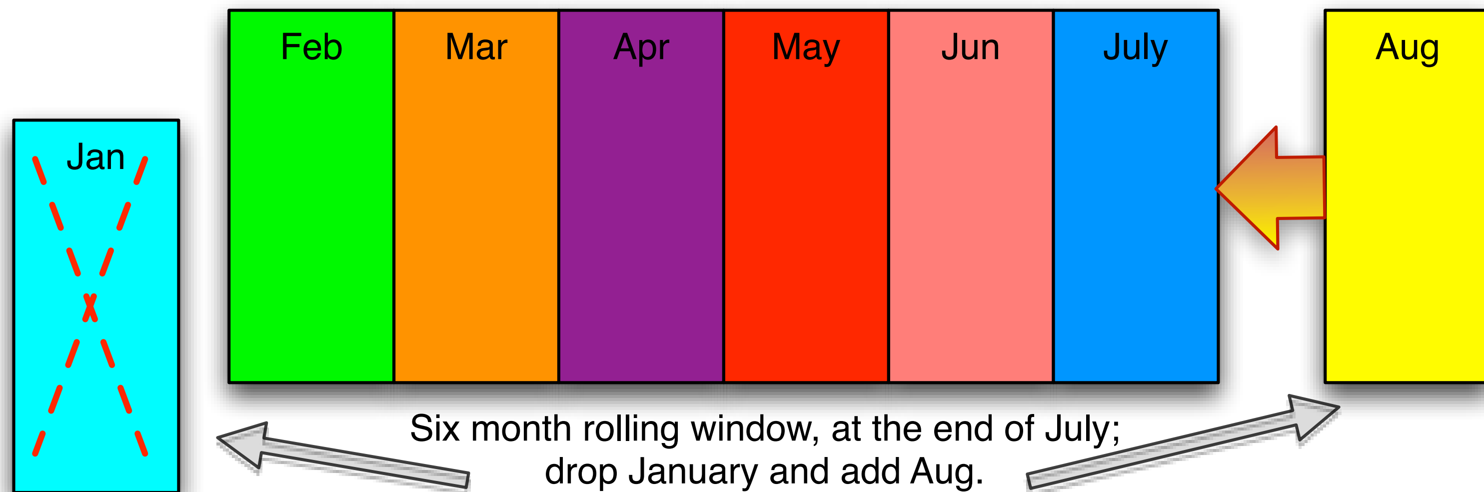
- MySQL knows how the table was divided into smaller parts and uses this information to speed up queries
- Operations on many smaller parts are often faster than on one big table
  - Optimize
  - Create INDEX, etc.

ORACLE

Copyright Oracle 2010

Why Partition?
1) Deleting Data by Partition

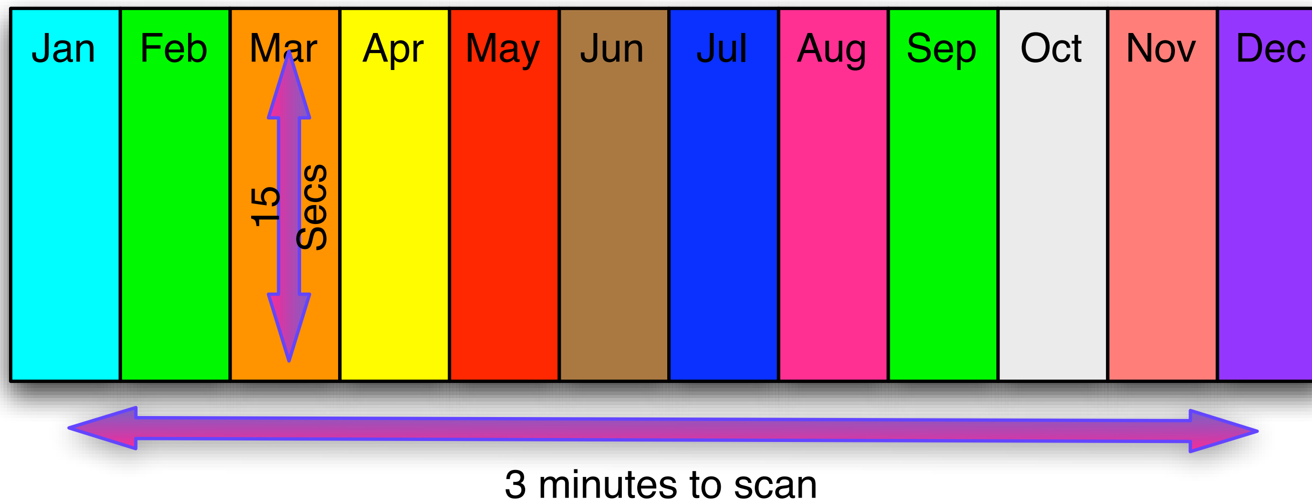
- Partitions can support deleting data by dropping a partition
  - If you insert 1,000,000 rows a day, eventually you need to delete 1,000,000 rows a day
- Only works with **Range** and **List** partitioning
- Very useful for rolling date/time range
- Can be very useful even for small increments, i.e.1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

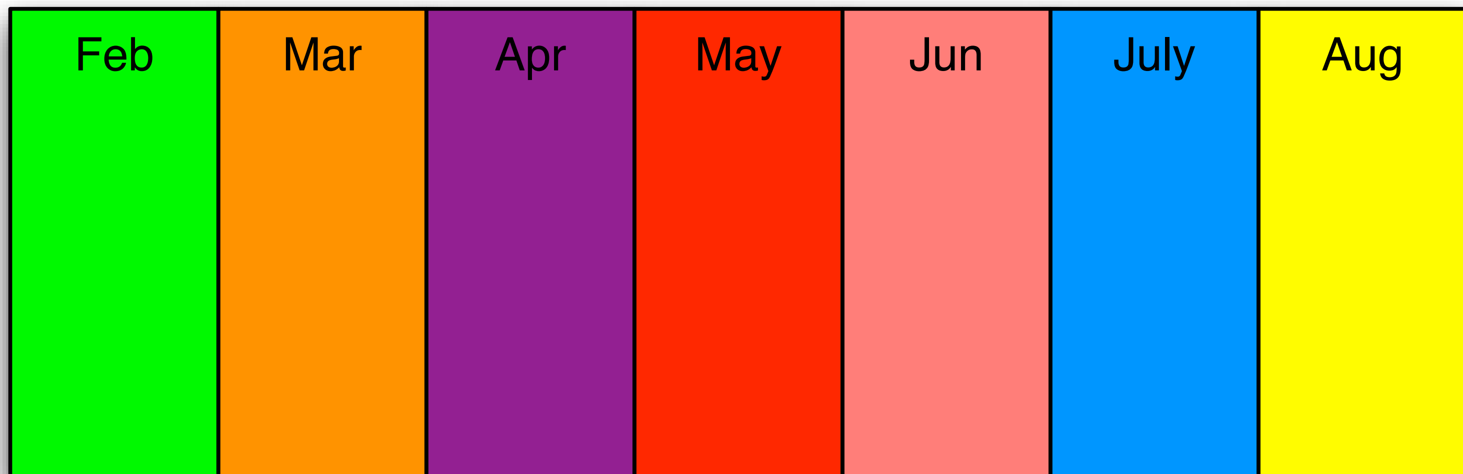


ORACLE

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



Optimize on: Mon Tue Wed Thu Fri Sat Sun

ORACLE

Copyright Oracle 2010

Types of MySQL Partitioning

Copyright Oracle 2010

Types of MySQL Partitioning

- **Key/Hash - Not useful for Deleting by partition**
  - **Key**(***column list***) - Uses internal hash function
    - no expressions ,specify columns and number of partitions
  - **Hash**(**INT *expr***) - Mod on user expression
    - effectively MOD on an integer expression or column
  - 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

ORACLE

Copyright Oracle 2010

Key Partitioning

- Uses internal hash function (MD5 for cluster/NDB)
- Partitioning is by zero or more columns
  - zero defaults to the primary key or a unique key
- Use COALESCE to merge data into fewer partitions
- Use ADD to split data into more partitions
- Supports “LINEAR” distribution
  - Distribution not as good, but much faster for ADD and COALESCE
- Normally gives **good** distribution of data
- Difficult to determine what partition a particular record is in
- Partitioning key can be any type except blob or text
-
```
CREATE TABLE Sales ( saleDate date, ... )
      PARTITION BY KEY(saleDate)
      PARTITIONS 16 ;
```

ORACLE

Copyright Oracle 2010

ORACLE

Copyright Oracle 2010

Partitioning Expressions

- **HASH**, **RANGE** and **LIST** can use expressions
- A Partitioning Expression must return an integer 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

ORACLE

Copyright Oracle 2010

ORACLE

Copyright Oracle 2010

Hash Partitioning

- Easy to define like key, but supports expressions
- Internally uses MOD(<expression>, N) where N is the number of partitions to determine the partition
- Even distribution ... **maybe (more later...)**
- Supports “LINEAR” distribution
  - Distribution not as good, but much faster for ADD and COALESCE
-

```
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
- All possible values ***must*** be listed in the partition expression

```
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) );
```

- No overlapping lists!
- Can be declared in any order
- Distribution is 100% up to your design!

ORACLE

Copyright Oracle 2010

Range Partitioning

ORACLE

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, if you want allow this, then add a final range just for these special cases with a “maximum” value as needed.

ORACLE

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, if you want allow this, then add a final range just for these special cases with a "maximum" value as needed.

```
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

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

Partitioning and NULL

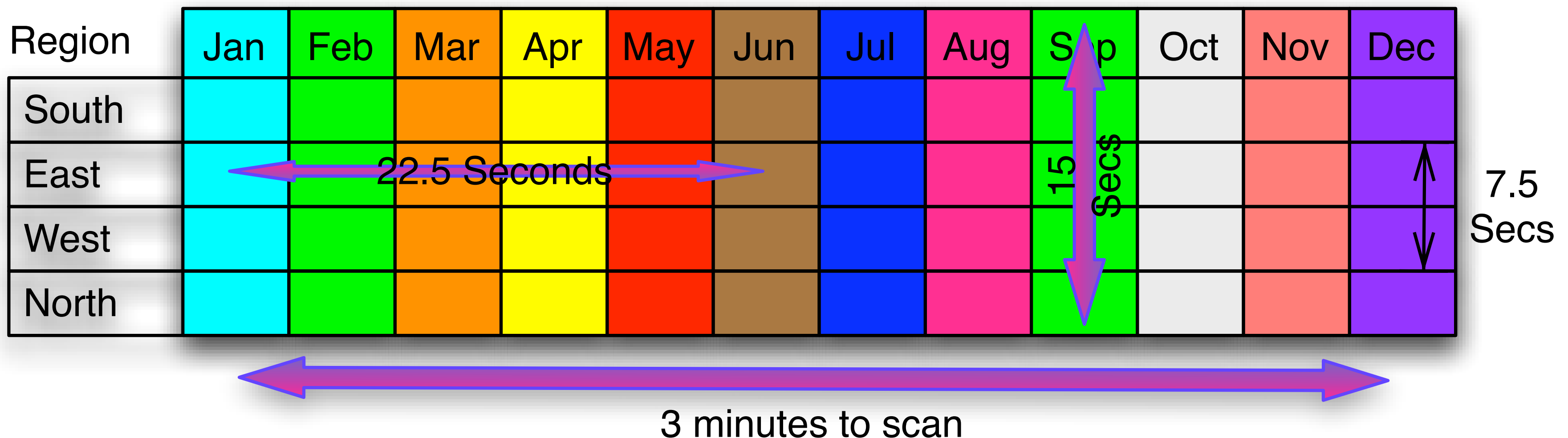
- **RANGE** Partitioning
  - NULL < than any other value
- **LIST** Partitioning
  - THere must be one and only one partition defined with a NULL in its list
- **HASH**
  - Treated as 0 (ZERO)
- **KEY**
  - Treated as 0 (ZERO)

ORACLE

Copyright Oracle 2010

Sub-partitioning

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



ORACLE

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, only that 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;
```

- If the partitioning column(s) are the PKEY or the leading part of a compound PKEY then MySQL will always choose the PKEY over partition elimination

```
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(col1)
    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)

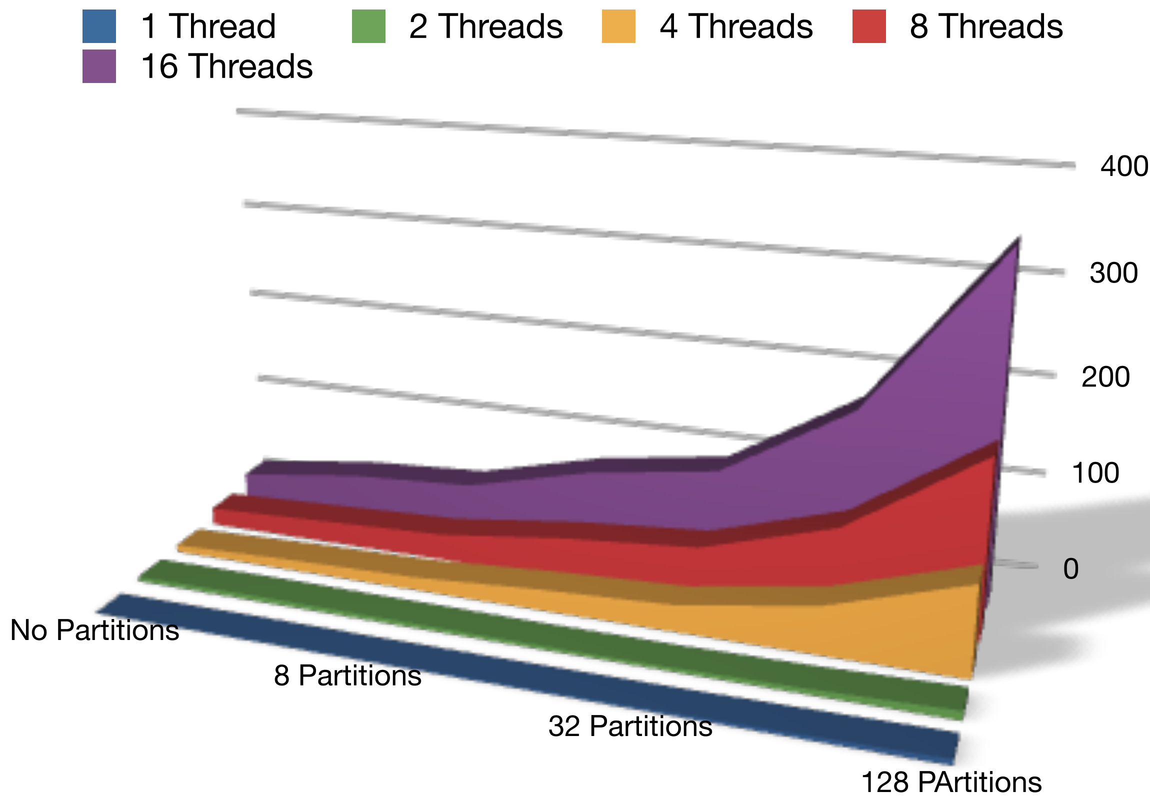
	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
No Partitions	0.47	4.43	6.93	17.63	28.25
4 Partitions	1.25	5.13	10.63	22.25	44.37
8 PArtitions	1.25	4.32	13.11	26.87	51.35
16 Partitions	1.67	3.62	21.20	42.57	83.16
32 Partitions	2.29	3.62	26.88	52.84	102.20
64 Partitions	3.85	5.42	48.70	92.44	180.23
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!***
- Don’t expect to just partition your tables and improve performance without reevaluating your indexes in respect to the partitioning of the table!
- Do not expect to get any performance improvement due to partition elimination if partitioning column(s) lead or make up a unique index for that table
- Too many partitions over 124 start to really slow down non-unique index lookups

Copyright Oracle 2010

Short Term Rolling Data

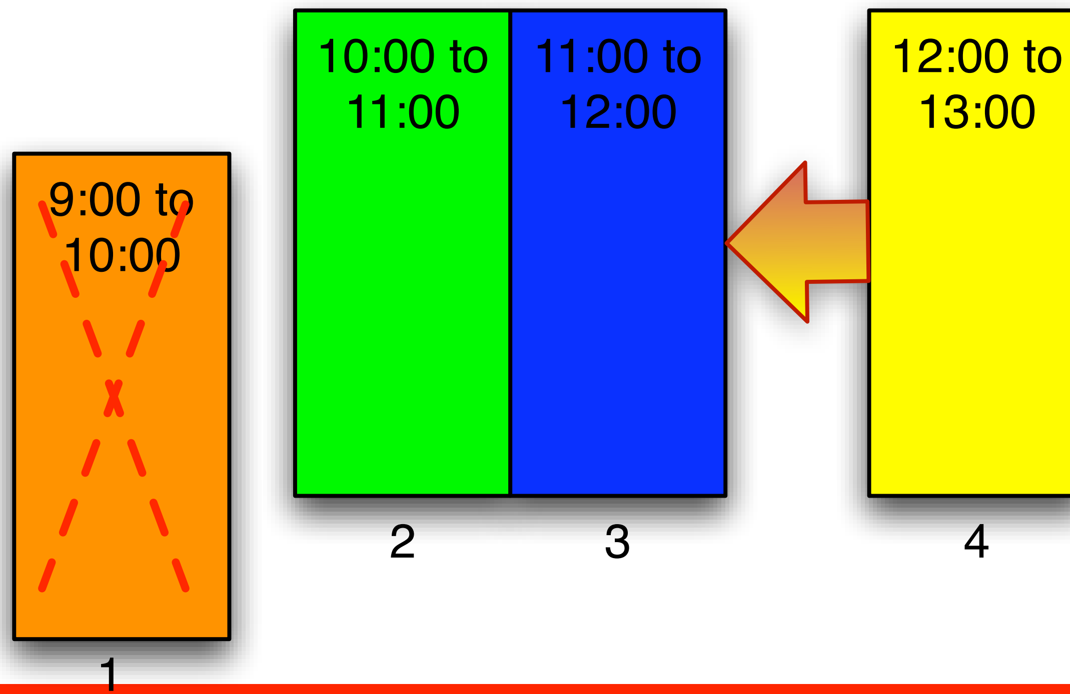
- **When to Use:**
  - Data only needs to be kept for a few hours or days
  - Not a lot of data, indexes work fine for lookups
  - Its hard to balance the deletes against other operations
- **Goal:**
  - Reduce or eliminate delete overhead
- **Steps**:
  - Create a table (**LIST** or **RANGE**) with at least three partitions
  - Let partitions 1 and 2 fill
  - During the filling of part 3, perform any analysis you need on part 1, then drop part 1 and add part 3
- Its OK if the primary key is also the partitioning expression for this use case, the partitioning is not for select query performance, only for fast deletes

ORACLE

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”



ORACLE

Copyright Oracle 2010

Short Term Rolling

- Original 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

- **When to use:**
  - Very large tables 10, 100 GB or more, **too big for cache**
  - There is a date column or similar to partition on
  - Most of your queries filter on the partitioning column
  - Many or most queries currently do large index scans
  - Inserts are getting too slow
  - Optimize, add index, etc. are taking far too long
- **Goals:**
  - Partition elimination to speed up selects
  - Speed up inserts
  - Reduce optimize and other maintenance overhead
  - Eliminate Delate Overhead
- **Steps:**
  - Partition the table into many (usually 32+) partitions
  - Roll N partitions out for each N you add.

ORACLE

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

2007	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2008	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2009	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec

ORACLE
Copyright Oracle 2010

Long Term Rolling Data

- Original Table definition

```
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 ;
```

ORACLE

Copyright Oracle 2010

Optimize, Analyze, etc. by Partition

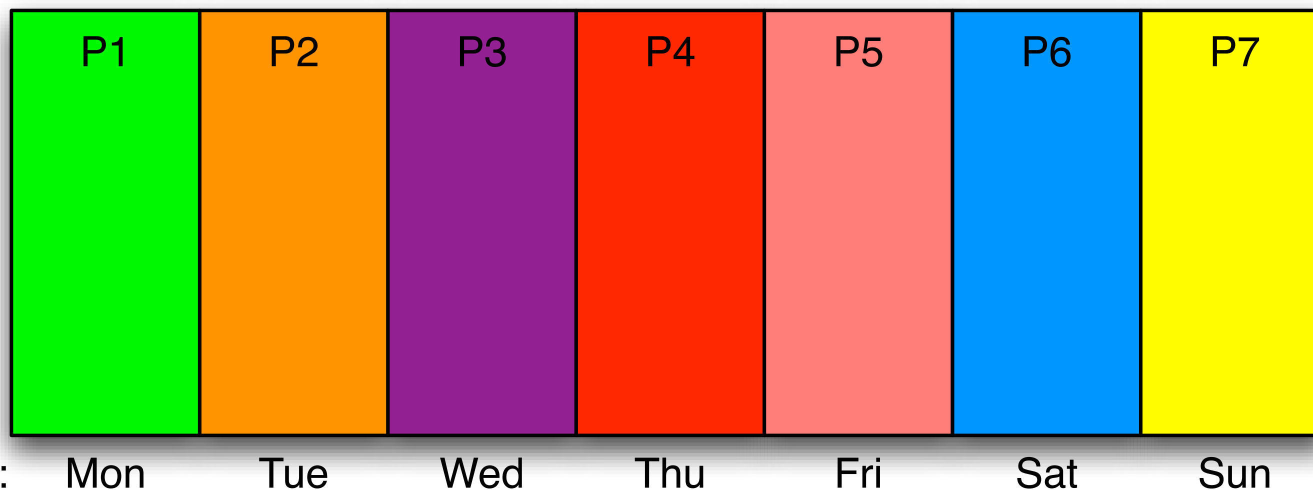
- **When to use:**
  - A table has a lot of inserts, updates and deletes
  - For medium sized tables, around 1-10GB
  - Optimize, Analyze, etc. takes too long and is needed
  - You have a unique column you can HASH on.
- **Goals:**
  - Reduce optimize, Analyze, etc. overhead
- **Steps:**
  - Partition 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
- Best with only one unique index that is auto increment (or similar) and is used for the partitioning column

ORACLE

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



ORACLE

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 ;
```

update

ORACLE

Copyright Oracle 2010

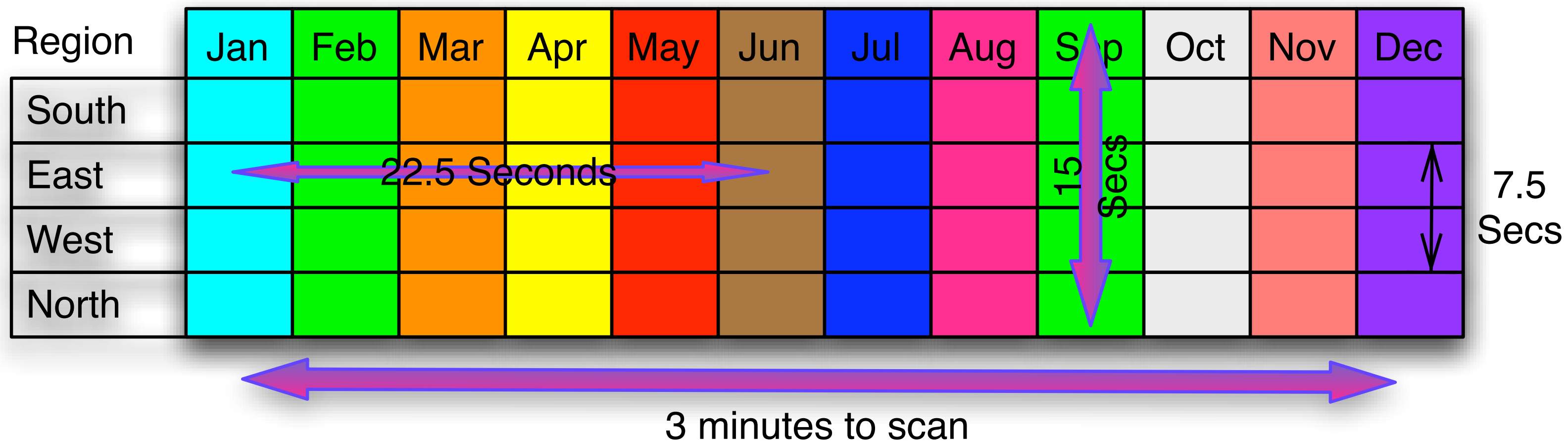
Sub-Partitioned Tables

- **When to use:**
  - Very large tables 10, 100 GB or more, **too big for cache**
  - 2 columns, Column A and Column B where either column A or Column B or both are used in most (> 95%) of the queries
  - Many or most queries do large index scans
  - Optimize, add index, etc. are taking far too long, inserts may be slow
  - Small < 5-10 queries running at a time
- **Goals:**
  - Support partition elimination queries to speed up selects
    - For either columns, even faster with both
  - Speed up inserts
  - Reduce optimize and other maintenance overhead
  - Eliminate Delate Overhead
- **Steps:**
  - Partition the table into many (32+) partitions each with 4 or more sub-partitions
  - Roll N partitions out for each N you add.

Copyright Oracle 2010

Sub-Partitioned Tables

- Top level partition is by month
- Sub-Partition is by region
  - May be **HASH** or **KEY** only!
  - **HASH** Partitioning gives best control of distribution
- The more records accessed in a scan the better the performance will be versus using an Index



Copyright Oracle 2010

Optimize, Analyze, etc. by Partition

- Original Table definition

```
CREATE TABLE Sales (
       salesDate     TIMESTAMP,
       storeID       smallint unsigned ,
       regionID      tinyint unsigned ,
       amount        decimal(10,2)
) PARTITION BY RANGE ( UNIX_TIMESTAMP(salesDate) )
        SUBPARTITION BY HASH(RegionID) SUBPARTITIONS 4
(
    PARTITION p200801 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-02-01 00:00:00') ),
    PARTITION p200802 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-03-01 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 Partitions for the first partition (Partitions are P0 to P6)

```
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') ) ) ;
```

ORACLE

Copyright Oracle 2010

Do not assume that the distribution is good just because the manual says so!

- Use the INFORMATION_SCHEMA PARTITIONS table to see what it really is!

```
mysql> select table_schema, table_name, partition_name, table_rows
from INFORMATION_SCHEMA.PARTITIONS where table_name = "geoH4" ;
+--------------+------------+----------------+------------+
| table_schema | table_name | partition_name | table_rows |
+--------------+------------+----------------+------------+
| mypart       | gx         | p0             |     117839 |
| mypart       | gx         | p1             |     103519 |
| mypart       | gx         | p2             |     197823 |
| mypart       | gx         | p3             |      93121 |
+--------------+------------+----------------+------------+
4 rows in set (0.01 sec)
```

- Do a show create \G on the INFORMATION_SCHEMA PARTITIONS table to see the available columns

ORACLE

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.

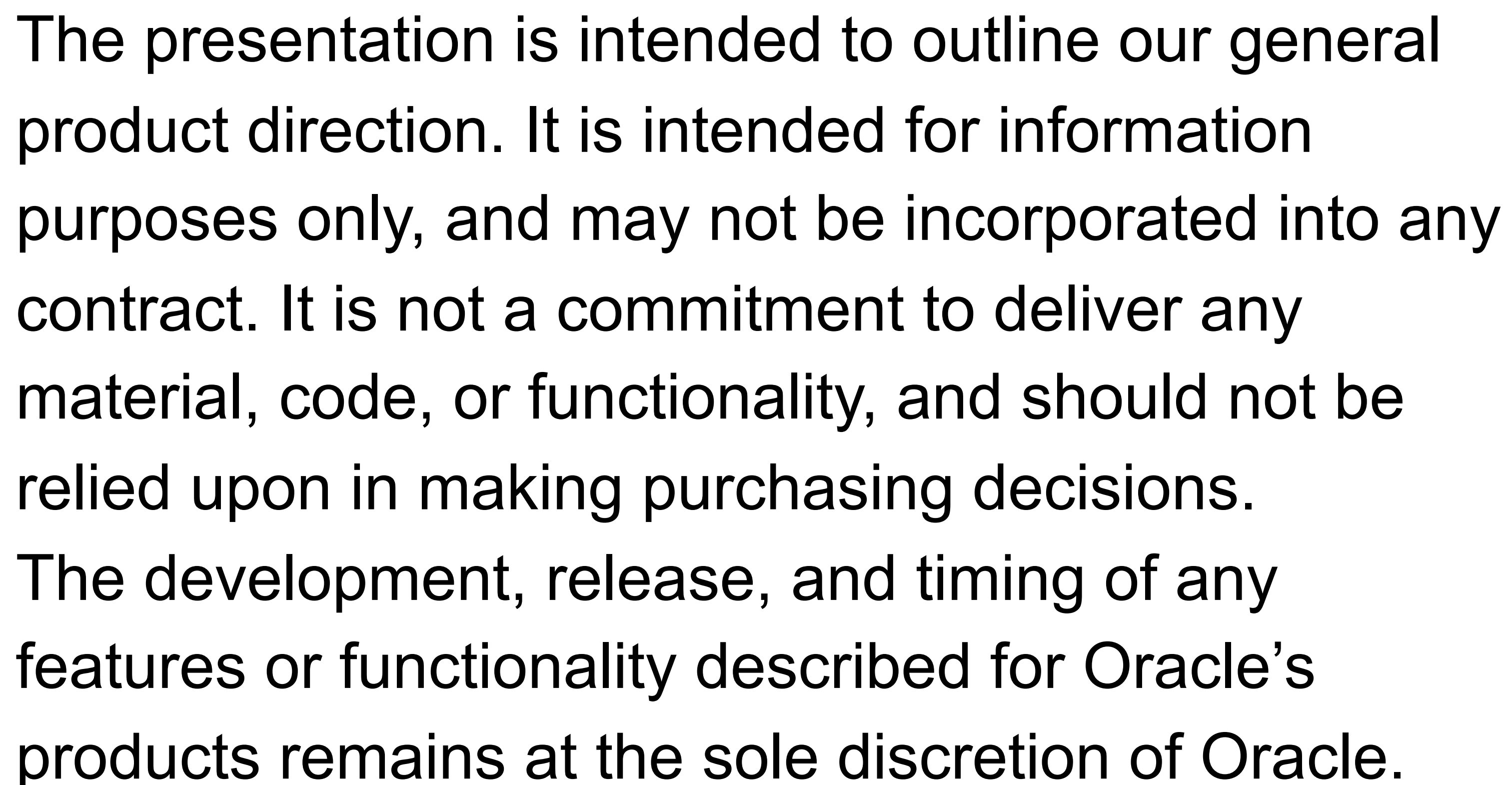
Copyright Oracle 2010

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

ORACLE

Copyright Oracle 2010



ORACLE