# Optimizing Refresh of a Set of Materialized Views

Nathan Folkert, Abhinav Gupta, Andrew Witkowski
Sankar Subramanian, Srikanth Bellamkonda, Shrikanth Shankar
Tolga Bozkaya, Lei Sheng

Oracle Corporation

February 14, 2006

# Roadmap

- Typical Schema in data Warehouses [with windowing] [1]
- Motivating Example
  - Conventional refresh
- PCT $(Oracle9i)$
- EPCT Refresh for a MV
- Managing refresh of a set of MVs

# Schema

| Sales | | |
|---|---|---|
| day | city | amt |

Facts table.

| Times | | | |
|---|---|---|---|
| day | month | quarter | year |

Dimension table : 5 levels

| Geog | | |
|---|---|---|
| city | state | region |

Dimension table : 4 levels

# Schema: Sample MV

```
CREATE MATERIALIZED VIEW quart_state_MV AS
PARTITION BY RANGE (quart)
(
PARTITION VALUES LESS THAN 'Q1 03'
PARTITION VALUES LESS THAN 'Q2 03'
...
)
SELECT t.quart, g.state, sum(s.amt) amt
FROM sales s, geog g, times t
WHERE t.day = s.day and g.city = s.city
GROUP BY t.quart, g.state
```

# Partitioning : Advantages

- Each Partition is similar to separate relation
- Operations on different partitions can proceed concurrently
- Faster bulk operations
  - TRUNCATE/DROP partition
  - MULTI-TABLE INSERT SELECT

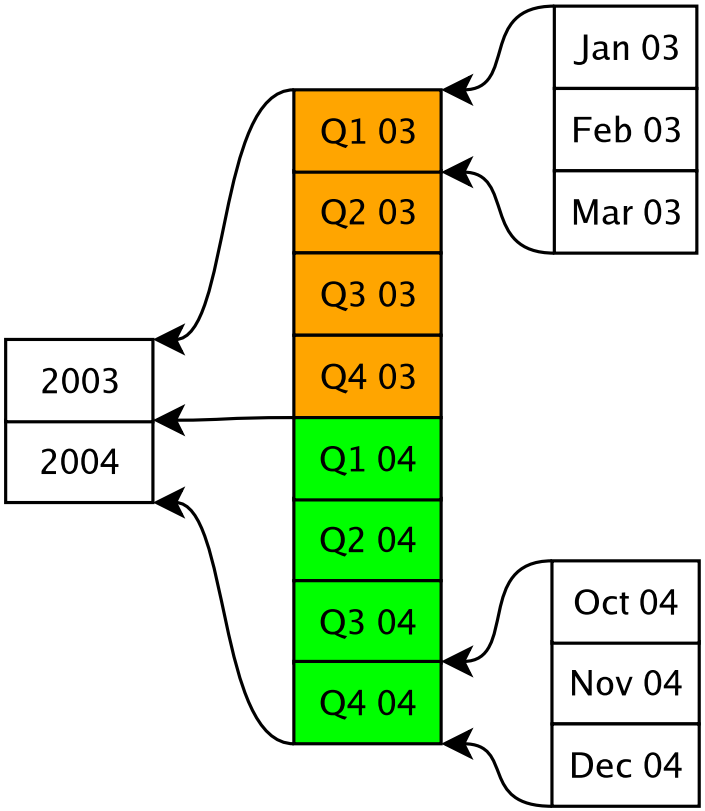
# Partitioning : MULTI-TABLE INSERTS [3]

```
INSERT ALL
       WHEN quart <' Q1 03'
            INTO quart_state_MV
            PARTITION quart_state_MV_part1
       WHEN quart <' Q2 03'
            INTO quart_state_MV
            PARTITION quart_state_MV_part2
       WHEN quart <' Q3 03'
            INTO quart_state_MV
            PARTITION quart_state_MV_part3
       ...
       SELECT t.quart, g.state, sum(s.amt) amt
       FROM sales s, geog g, times t
       WHERE t.day = s.day and g.city = s.city
       GROUP BY t.quart, g.state
```

# Partitioning : MULTI-TABLE INSERTS

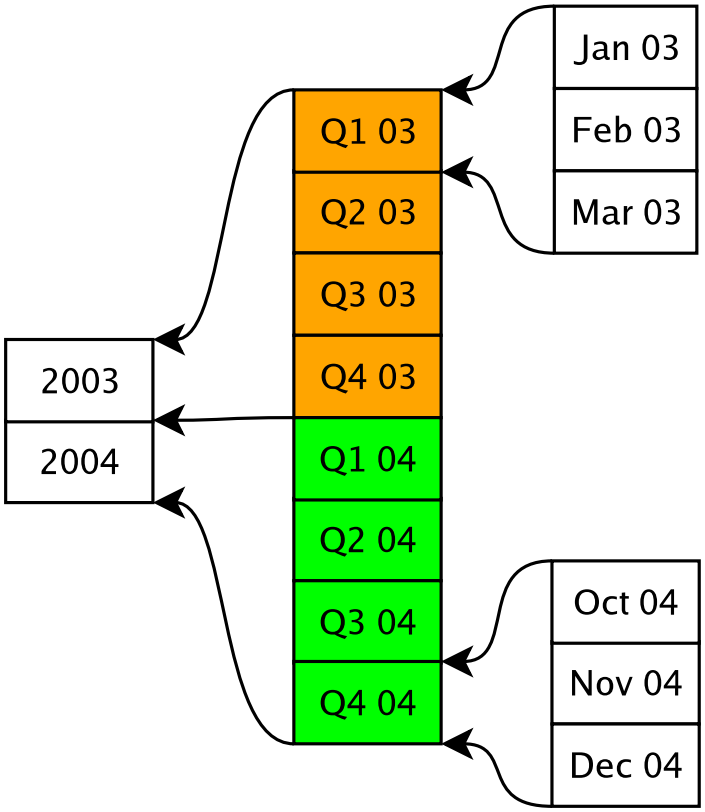
- Reusing locks
- Concurrent operations on partition
- If first time populating the partition then minimal undo logging (mark new extents invalid)
- drop partition equivalent to drop relation
- Index updation considerations

# Motivating Example

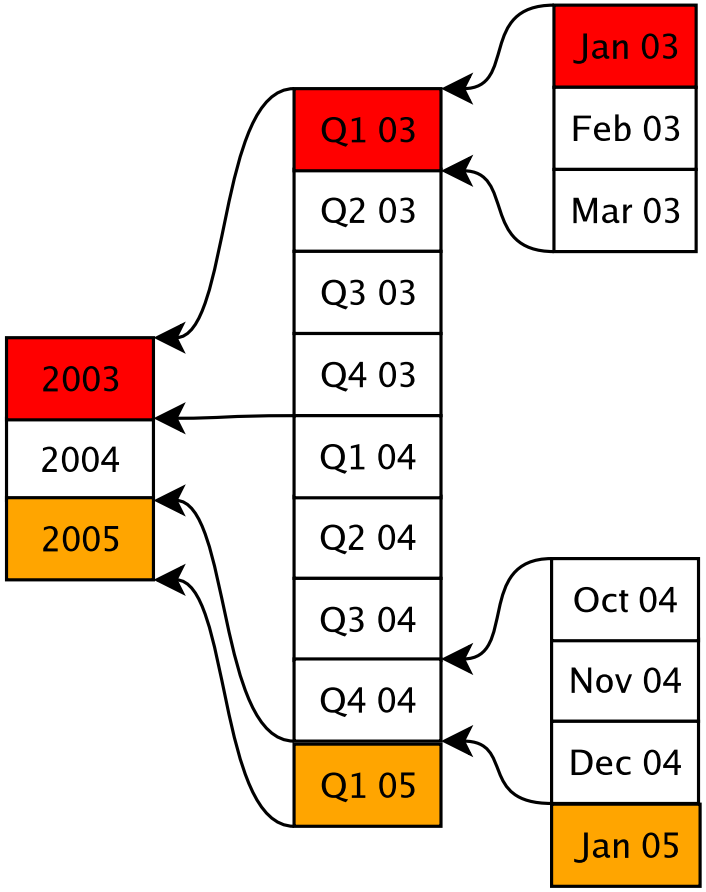



Windowed sales database.(for 24 months)

# Motivating Example

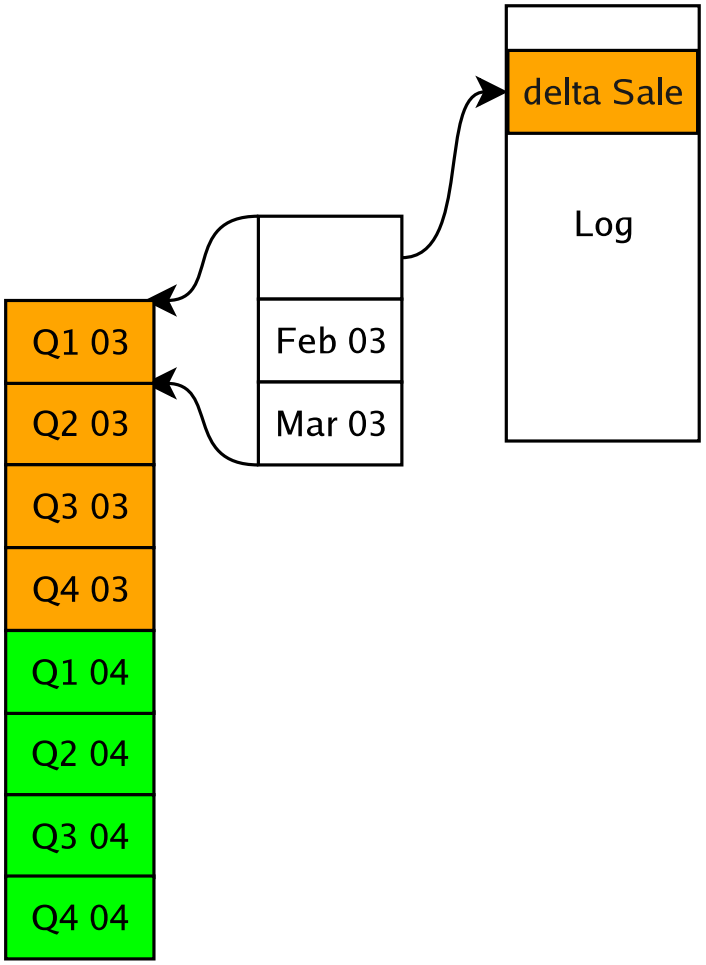



Windowed sales database.(for 24 months)




Jan 03 is deleted. Jan 05 is added.

# Motivating Example : Conventional Refresh




Deleted $Jan03$ data is in log file (as delta Sale)

Compute $delta\{quart\_state\_MV\}$

Update MV by taking its join with $delta\{quart\_state\_MV\}$

# Motivating Example : Conventional Refresh

```
{ SELECT t.quart, g.state, sum(s.amt) amt
FROM delta_sales s, geog g, times t
WHERE t.day = s.day and g.city = s.city
GROUP BY t.quart, g.state } AS delta
```

```
UPDATE quart_state_MV mv
SET mv.amt = (
              SELECT mv.amt - delta.amt
              FROM {....}AS delta
              WHERE mv.quart = delta.quart and
              mv.state=delta.state
              )
```

# PCT Refresh

1. ORACLE Specific Features :

   - Partition Maintenace Operations (ADD/DROP/TRUNCATE) are cheaper than INSERT/DELETE
   - INSERTS with Direct Path are much cheaper than Conventional INSERTS

2. PCT : Partition Change Tracking Refresh (ORACLE 9i)

   - Uses partitioned views
   - TRUNCATE affected partition
   - Recompute truncated partition from base partition
   - Use Direct Path INSERT

# PCT Refresh

- Very efficient if MV and Base table partitioned on same field

# PCT Refresh

- Very efficient if MV and Base table partitioned on same field
- Otherwise
  - Given an affected base partition, how to find out affected partition in MV.

# Enhanced PCT Refresh (EPCT)

- Implemented in ORACLE 10g
- Concept of Partition Join Dependent Expression

Methods:

- EPCT with DELETE
- EPCT with TRUNCATE

# EPCT : PJoin Dependent Expression of Table T

Definition:

- Columns (in the view) which are from tables directly/indirectly equi-joined with partioning key of T
- Value of Partioning key determines value of PJoin dependent expression

# EPCT : PJoin Dependent Expression of Table T

Example:

- ```
  SELECT t.quart, g.state, sum(s.amt) amt
  FROM sales s, geog g, times t
  WHERE t.day = s.day and g.city = s.city
  GROUP BY t.quart, g.state
  ```
- Base table $Sales$ partioned by $day$
- table equi-joined with $sales.day$ = $times$
- Columns in view which are from $times$ = $quart$
  ```

# EPCT : with DELETE

1. Subquery Predicate:

$pj\_dep\_exp\_list$ IN
(
SELECT $pj\_dep\_exp\_list$
FROM $tab\_list$
WHERE $join\_pred$ AND $part\_pred$
)

2. Used to generate:
   - DELETE statement
   - INSERT statement

3. Avoids join with facts table within IN clause

# EPCT : with DELETE

```
DELETE quart_state_MV WHERE
quart IN(
      SELECT quart
      FROM times
      WHERE day >=' 01 - Jan - 2003' AND day <' 01 - Feb - 2003')

INSERT INTO quart_state_MV
SELECT t.quart, g.state, sum(s.amt) amt
FROM sales s, geog g, times t WHERE
quart IN(
      SELECT quart
      FROM times
      WHERE day >=' 01 - Jan - 2003' AND day <' 01 - Feb - 2003')

GROUP BY t.quart, g.state
```

# EPCT : with TRUNCATE

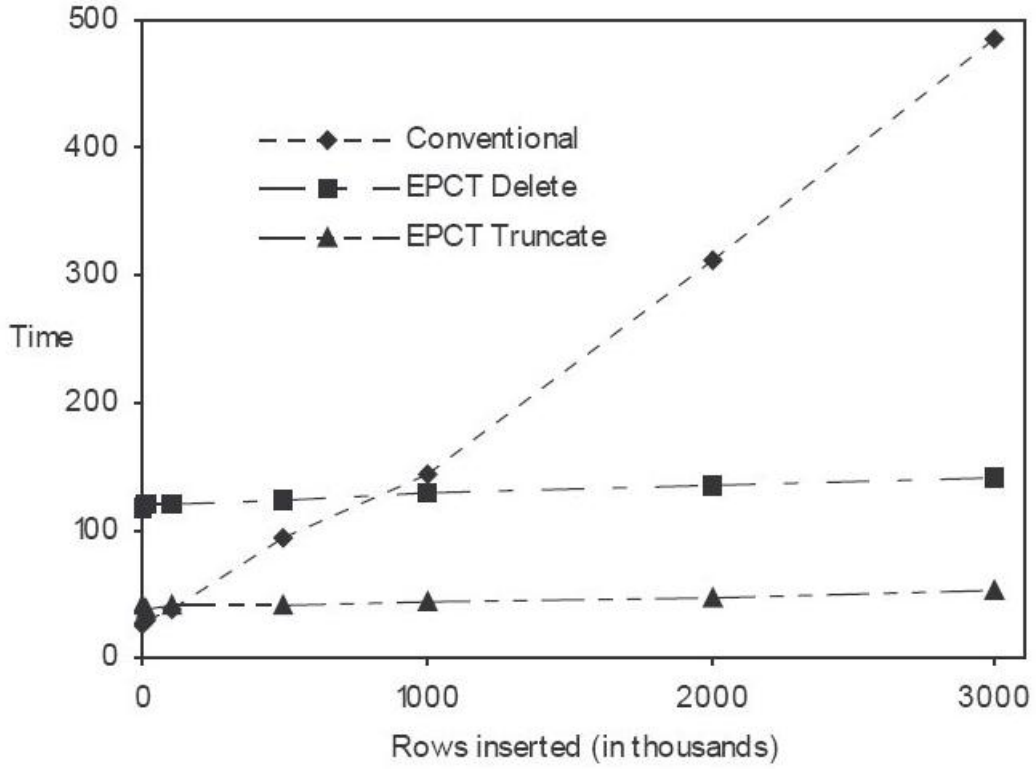
1. $Partition\_exp\_list \neq \phi$

2. $Partition\_exp\_list =$
   $\{Pjoin\_dep\_exp\_list \cap (partitioning\ columns\ of\ MV)\} \neq \phi$

3. Subquery Predicate:

```
SELECT DISTINCT PARTNAME(mv_name, Partition_exp_list)
FROM            (
                SELECT DISTINCT Partition_exp_list
                FROM tab_list
                WHERE join_pred AND part_pred
                )
```

# Performance: EPCT Vs. Conventional refresh




MV :

- Partitioned on quarter
- has 22 million rows

rows inserted in fact table.

# Functional Dependencies & Query Rewrites

- Functional Dependency specification syntax
  - Foreign Key / Primary Key
  - Dimensions $day \rightarrow month \rightarrow quart \rightarrow year$

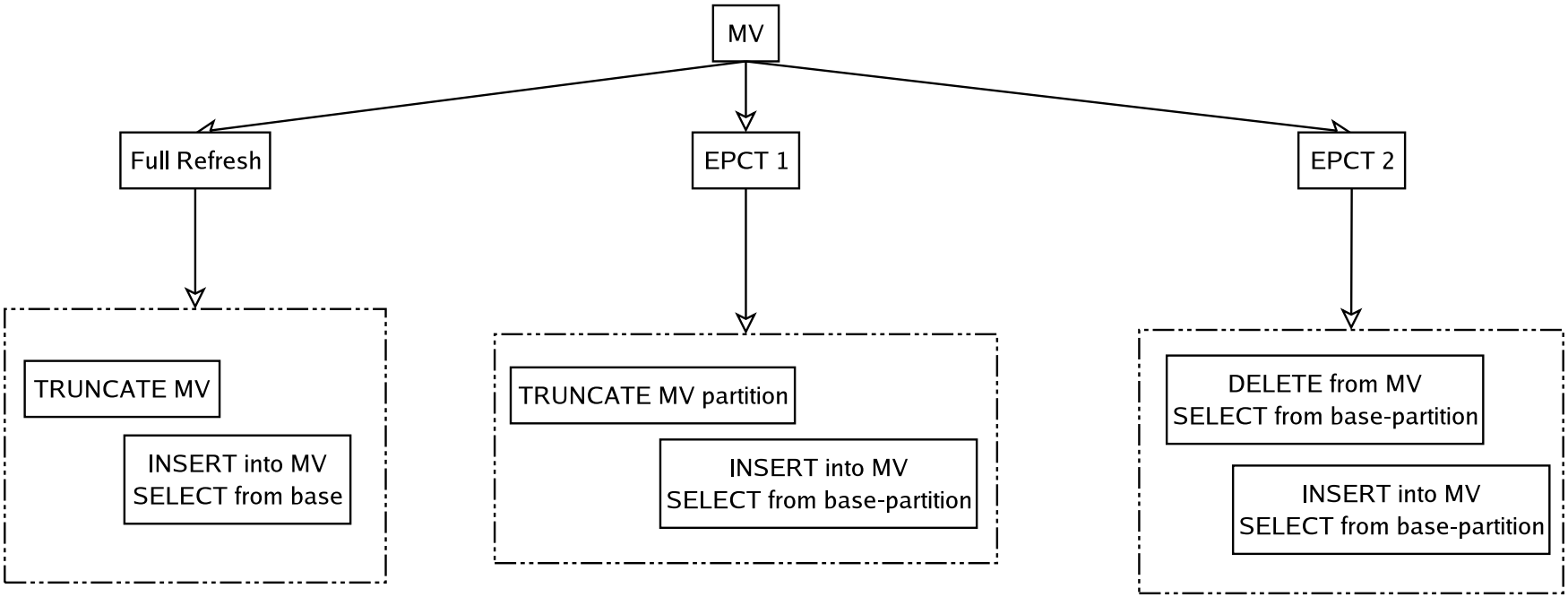
```
CREATE DIMENSION time_dim
       LEVEL day IS times.day
       LEVEL month IS times.month
       LEVEL quart IS times.quarter
       LEVEL year IS times.year
       HIERARCHY calender
       (day CHILD OF month CHILD OF quart CHILD OF year);
```

- Query rewrite : Use $month\_state\_MV$ to refresh $quart\_state\_MV$

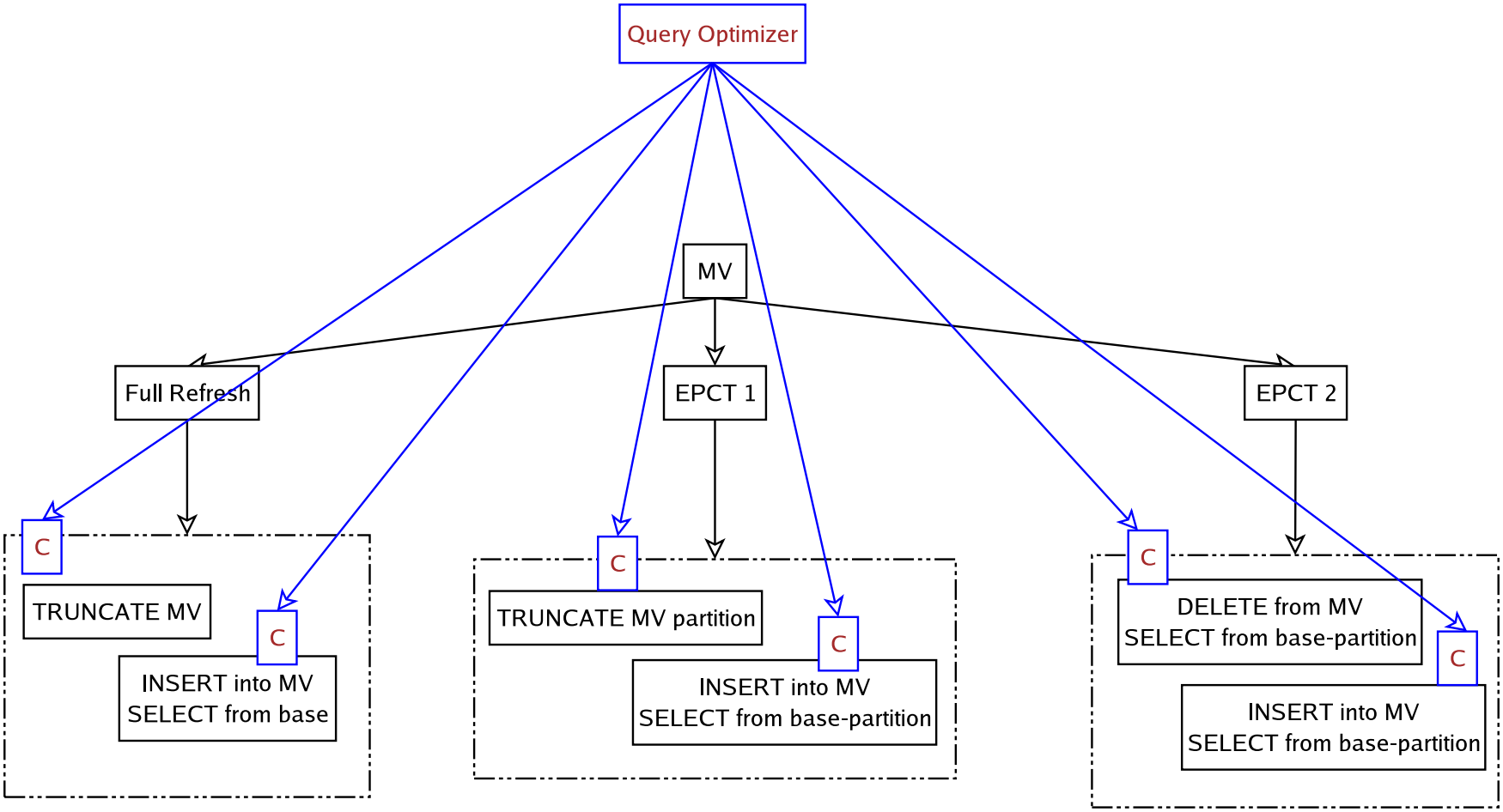
# Scheduling Refresh of MVs

- Creating best rewrite graph
- Finding an acyclic graph
- Executing refresh

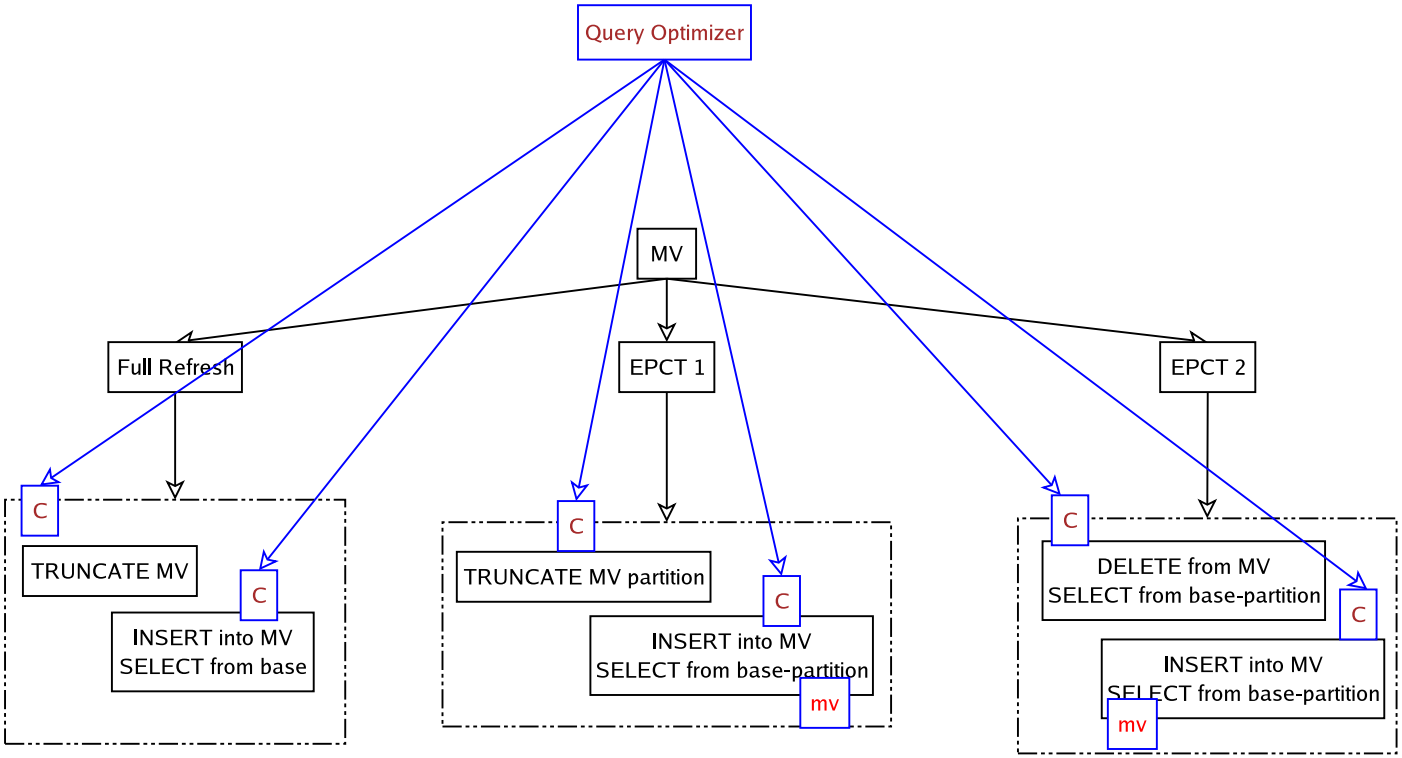
# Creating Best Refresh Graph



# Creating Best Refresh Graph



# Creating Best Refresh Graph




Assumes availability of all the MVs in fresh state

# Finding an acyclic graph

- Cycles can be present : eg. two views defined on same query
- Tarjan's Algorithm to detect SCC
- Break cycles by removing edges (arbitrarily/heuristically) : no longer optimal

# Executing refresh

- Consideration for resources required
- Find out threshold : cost at which node requires majority of system resources
- for each $node$ in topological sort order
  - if $(\exists\ node | cost(node) > threshold)$ then $schedule\ node$
  - otherwise $schedule\ nodes\ in\ batches$ with $number\ of\ processes \propto cost(node)$

```
get source nodes
if any source nodecost exceeds threshold
   refresh node MV with full parallelism
   return
for each node 1...n, ordered by cost
    if(nodecost/runningsum . processes) < 1
      break
    runningsum+ = nodecost
    k++
for each node k...1
    p = ⌊(nodecost/runningsum . processes)⌋
    processes− = p
    runningsum− = nodecost
    refresh node MV with parallelism = p
```

# Conclusion

- A new approach to refresh MV using partitioning of base tables and MVs
- Optimal Algorithm to schedule refresh of a set of MVs
- Performace Improvements

# References


- VLDB Presentation: $http://www.vldb2005.org/program/slides/tue/s1043-folkert.ppt$
- Oracle Documentation : Data Warehousing Guide
- Multi table Insert :

  $http://web.njit.edu/info/oracle/DOC/server.920/a96540/statements_913a.htm\#2125349$

# Questions?