Materialized Views

Dr. Rim Moussa

University of Carthage
rim.moussa@gmail.com

Aggregate Tables

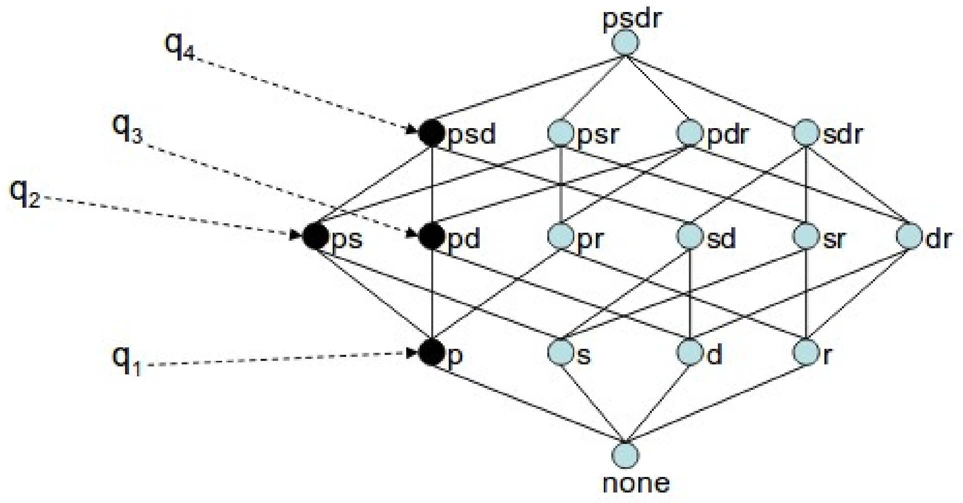
- Aggregate tables are also known as :
  - Snapshots, summary table, materialized views (Oracle), materialized query tables (DB2), indexed views (SQL Server)
- Definition
  - A *materialized view* is a table that contains the result of a query
  - if a view is used frequently enough, it may even be efficient to materialize it
  - Aggregates are pre-calculated summaries derived from the most granular fact table
- *Goals*: Aggregate table are used to:
  - Cache expensive queries in a data warehouse with summarized table. Then use the cache to process queries.
  - Replicate data to non-master sites in a replication environment
- MVs are recommended when queries are known:
  - OLAP queries are usually ad-hoc!

Cost of Aggregate Tables

- Complex calculus
  - optimized through smart data fragmentation and distributed (parallel) calculus
- Storage requirement
  - Volume in bytes of each MV
- Refresh
  - recompute parts of the materialized view each time one of the underlying base tables changes.
  - How frequent are changes?
  - Always compare
    - incremental refresh performances to
    - recomputation for source base tables performances

What to materialize?

- MVs Advisor
  - Exhaustive enumeration
  - Recommendations for a specific workload
  - *m* number of dimensions, number of nodes in the lattice = 2^m
- Data cube lattice: 4 dimensions: product,store,day,reduction



q_1 = total sales per product

q_2 = total sales per product and store

q_3 = total sales per product and day

q_4 = total sales per product, store and day

Example Q12 of TPC-H benchmark

```
SELECT l_shipmode,
sum(case when o_orderpriority = '1-URGENT' OR o_orderpriority
= '2-HIGH' then 1 else 0 end) as high_line_count,
sum(case when o_orderpriority <> '1-URGENT' AND
o_orderpriority <> '2-HIGH' then 1 else 0 end) AS
low_line_count

FROM  orders, lineitem

WHERE
    o_orderkey = l_orderkey
    AND l_shipmode in ('MAIL', 'SHIP')
    AND l_commitdate < l_receiptdate
    AND l_shipdate < l_commitdate
    AND l_receiptdate >= date '1994-01-01'
    AND l_receiptdate < date '1994-01-01' + interval '1' year

GROUP BY  l_shipmode
ORDER BY  l_shipmode;
```

Parameters:
_list of values for *l-shipmode*
_1st day of a given year

MV12 of TPC-H benchmark (CTAS statement)

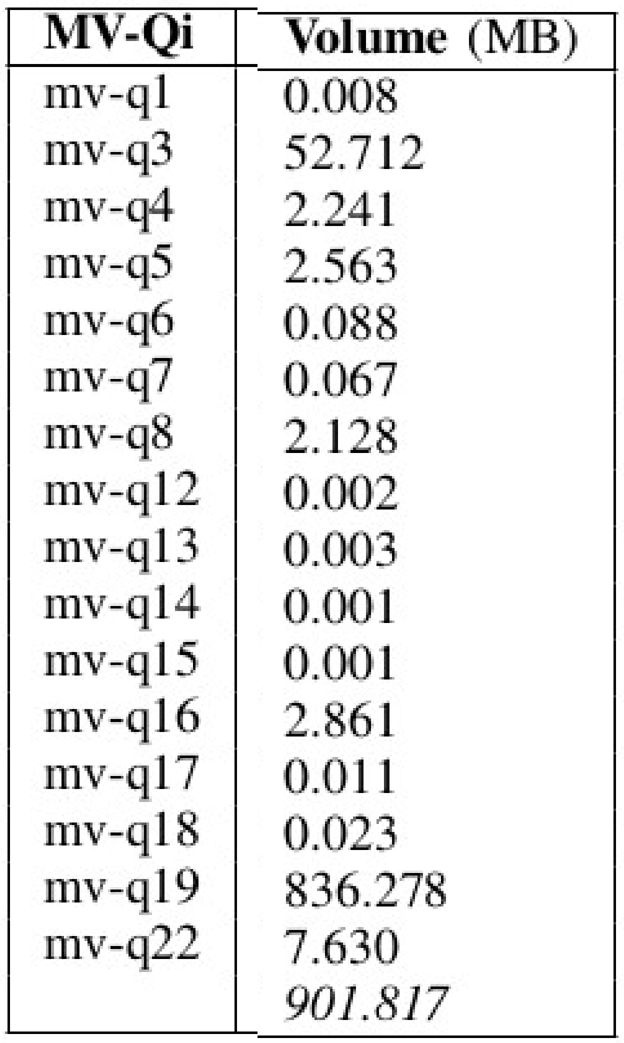
```
CREATE TABLE agg_c12 AS

SELECT year, l_shipmode,
sum(case when o_orderpriority ='1-URGENT' or o_orderpriority
='2-HIGH' then 1 else 0 end) as high_line_count,
sum(case when o_orderpriority <> '1-URGENT' and
o_orderpriority <> '2-HIGH' then 1 else 0 end) as
low_line_count
FROM orders, lineitem, time

WHERE o_orderkey = l_orderkey
AND timekey_receiptdate = time.timekey
AND o_orderkey = l_orderkey
AND l_commitdate < l_receiptdate
AND l_shipdate < l_commitdate

GROUP BY year,l_shipmode
ORDER BY year,l_shipmode;

Calculate agg_c12 metadata:
_volume (bytes)
_cardinality: DW age (number of years) X |l_shipmode|
_time to build (sec)
```

Q12 re-written

```
SELECT l_shipmode, high_line_count, low_line_count

FROM  agg_c12

WHERE l_shipmode in ('MAIL', 'SHIP')  AND year = '1994'

ORDER BY  l_shipmode;
```

Gain in performance because we don't
_ filter *lineitem* (l_commitdate < l_receiptdate AND l_shipdate < l_commitdate) : full table scan
  Lineitem has 6M X SF records
  SF is the TPC-H scale factor
  SF = 1 --> TPC-H warehouse volume is 1GB of raw data
_ run expensive joins
  orders $\bowtie$ *lineitem* is not performed
  Lineitem has 6M X SF records
  Orders has 1.5M X SF records
And measures are pre-computed

Materialized Views Data for TPC-H benchmark



MV-Qi	Volume (MB)
mv-q1	0.008
mv-q3	52.712
mv-q4	2.241
mv-q5	2.563
mv-q6	0.088
mv-q7	0.067
mv-q8	2.128
mv-q12	0.002
mv-q13	0.003
mv-q14	0.001
mv-q15	0.001
mv-q16	2.861
mv-q17	0.011
mv-q18	0.023
mv-q19	836.278
mv-q22	7.630
	901.817

```
--MonetDB
SELECT table, sum(columnsize)
FROM storage()
GROUP BY table;
```

Less than 1GB for MVs whether is the scale factor!

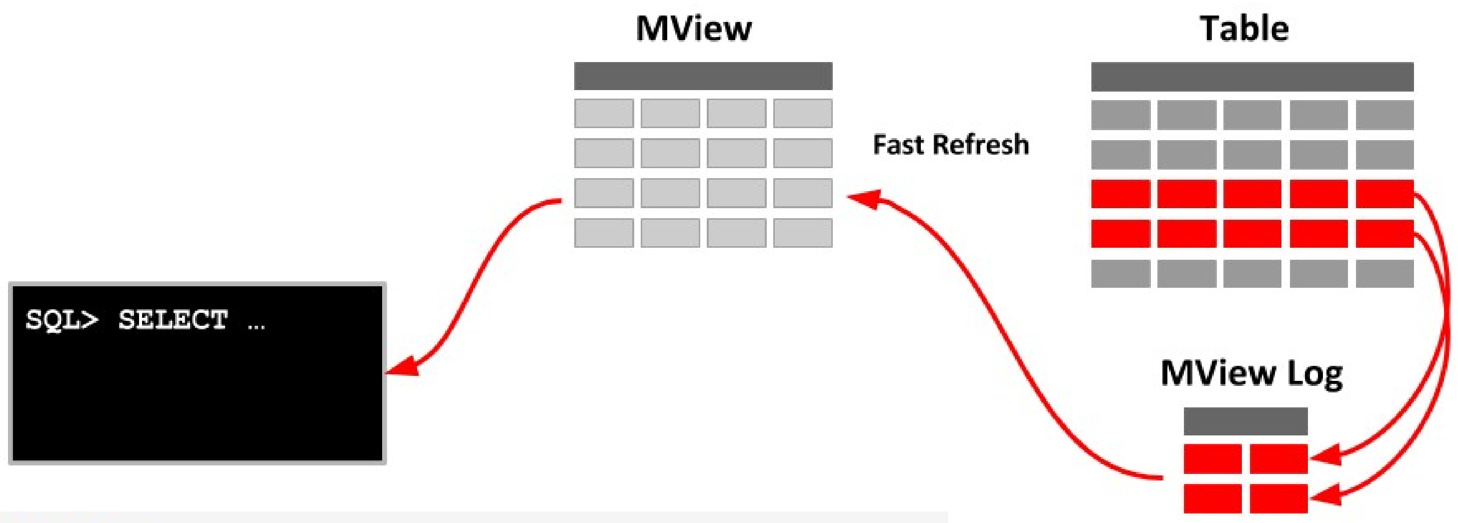
Materialized Views Refresh
RF1: new inserts in ORDERS and LINEITEM

```
-- create a temp lineitem table
CREATE TABLE LINEITEM_TEMP (L_ORDERKEY INTEGER NOT NULL,
L_PARTKEY .. L_PROFIT  DECIMAL(15,2));
-- create a temp orders table
CREATE TABLE ORDERS_TEMP  (O_ORDERKEY INTEGER NOT NULL, ...
O_SUM_LOST_REVENUE DECIMAL(15,2));
--load data into temp table
--Oracle
sqlldr system/manager@XE control = ins_orders.ctl
sqlldr system/manager@XE control = ins_lines.ctl
.ctl
load data
infile '/home/oracle/TPCCDATA/orders_*.tbl'
into table orders_tmp
fields terminated by "|"
(O_ORDERKEY, ...)
--MonetDB
COPY INTO LINEITEM_TEMP  FROM '$HOME\refresh\ins_lines.tbl'
USING DELIMITERS '|', '\n' ;
COPY INTO ORDERS_TEMP  FROM '$HOME\refresh\ins_orders.tbl'
USING DELIMITERS '|', '\n' ;
```

Incremental MVs Refresh (PL or `Merge Statement`)

```
CREATE TABLE agg_c12_ins AS
SELECT time.year as year, l_shipmode,
sum(case when o_orderpriority ='1-URGENT' or o_orderpriority ='2-HIGH' then 1
else 0 end) as high_line_count,
sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH'
then 1 else 0 end) as low_line_count
FROM orders_temp, lineitem_temp, time
WHERE timekey_receiptdate = time.timekey  AND o_orderkey = l_orderkey
AND l_commitdate < l_receiptdate  AND l_shipdate < l_commitdate
GROUP BY time.year,l_shipmode;

DECLARE
CURSOR delta IS SELECT * FROM agg_c12_ins;
BEGIN
FOR d IN delta LOOP
   UPDATE agg_c12 SET
         high_line_count = high_line_count + d.high_line_count,
         low_line_count = low_line_count + d.low_line_count,
         WHERE year = d.year and l_shipmode = d.l_shipmode;
END LOOP;
END;
/
drop table agg_c12_ins;
COMMIT;
```

Oracle Materialized Views: scenario



```
SQL> SELECT …
```

```
CREATE MATERIALIZED VIEW view-name
BUILD [IMMEDIATE | DEFERRED]
REFRESH [FAST | COMPLETE | FORCE ]
ON [COMMIT | DEMAND ]
[[ENABLE | DISABLE] QUERY REWRITE]
AS
SELECT ...;
```

```
CREATE MATERIALIZED VIEW LOG ON scott.emp
TABLESPACE users
WITH PRIMARY KEY
INCLUDING NEW VALUES;
```

Oracle Materialized Views Creation

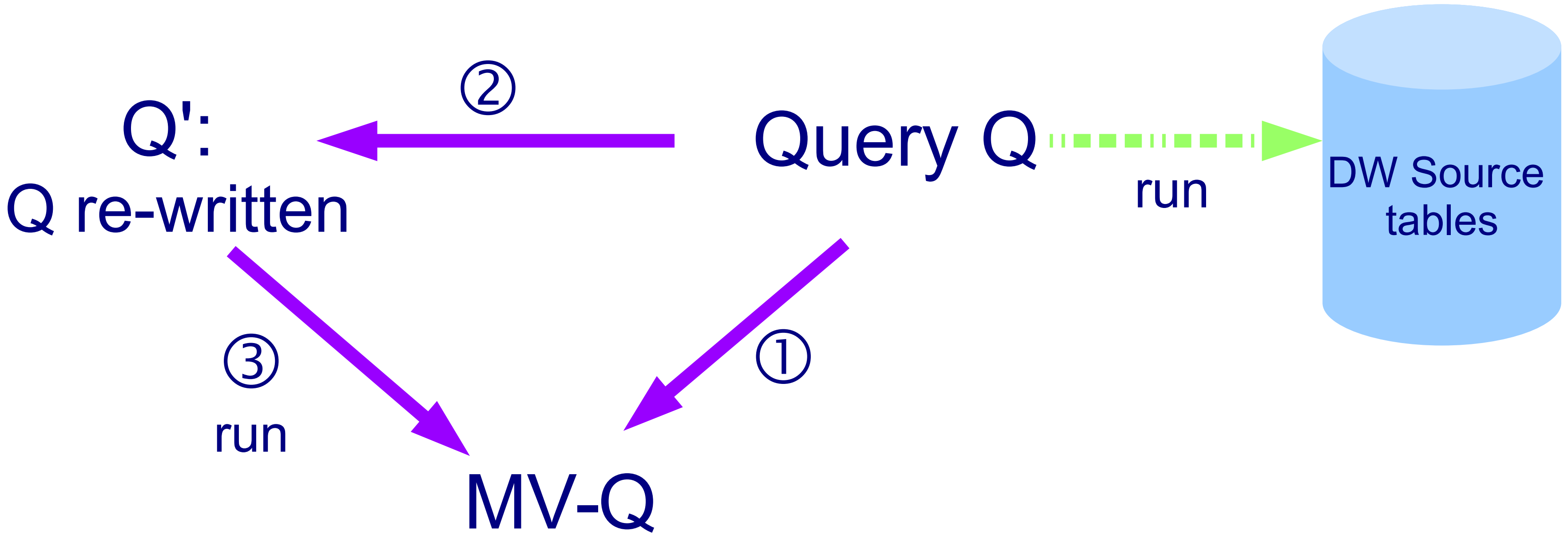
- BUILD
  - IMMEDIATE: the MV is populated immediately.
  - DEFERRED: The MV is populated on the first requested refresh.
- REFRESH
  - FAST: a fast refresh is attempted
    - Need to create an MV log
    - Incremental refresh only for simple SQL
  - COMPLETE: The table segment supporting the MV is truncated and repopulated completely using the associated query
  - FORCE: A fast refresh is attempted else a complete refresh is performed.
  - COMMIT: The refresh is triggered by a committed data change in one of the dependent tables.
  - DEMAND: The refresh is initiated by a manual request or a scheduled task.

```
EXEC DBMS_MVIEW.refresh('EMP_MV');
```

Oracle Materialized Views

```
--privileges to create an MV
CREATE MATERIALIZED VIEW
CREATE TABLE --an MV is a table
GRANT SELECT  --on tables required for building the MV

--enable QUERY REWRITE
GRANT QUERY REWRITE TO user|role;
ALTER SESSION SET
QUERY_REWRITE_ENABLED = TRUE;

--data about refresh jobs
SELECT job, SCHEMA_USER,
        TO_CHAR(last_date,‘DD/MM/YYY HH:MI’) “last refresh”,
        TO_CHAR(next_date,‘DD/MM/YYY HH:MI’) “next refresh”,
        interval “when ”,
        what “what”
FROM dba_jobs
WHERE what LIKE ‘%refresh%’ AND SCHEMA_USER=’SCOTT’;
```

Big Picture



②
Q':
Q re-written
Query Q
run
DW Source tables
③
run
①
MV-Q