VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Materialized Views

Lecturer : *Dr. Pavle Mogin*

SWEN 432
Advanced Database Design and Implementation

Plan for Materialized Views

- Views, OLAP and Data Warehouse
- A motivating example
- Query rewriting as a motivating force for using materialized views
- View materialization
- Examples

  - *Reading* :
    - *Chaudhuri, Dayal : An Overview of Datawarehousing and OLAP Technologies*

Virtual Views

- A SQL view is a virtual table whose content is computed on demand, every time from beginning
- The syntax is:

  CREATE VIEW <view_name> [<view_attr_list>]

  AS SELECT …
- Once defined, view can be used in queries, or in defining new views as any other table
- Note: a SQL view is a **virtual** view, it is not materialized

Materialized Views and OLAP

- Views are frequently used in decision support systems to allow data analyst to consider just his/hers part of the business
- Decision support queries are typically aggregate queries over very large fact tables
- Fact tables have a constant content over a relatively long period of time
- To allow fast answers, view materialization is a viable alternative

Views and the Data Warehouse

- A Data Warehouse itself is a (materialized) view of the operational databases and external data sources
- Fast execution of decision support queries is frequently accomplished by materializing views on DW base tables

Using Virtual Views in Queries

- Consider the star schema
  {*Sales* (*ProdId*, *TimeId*, *LocId*, *Sales_Data*),
  Product (*ProdId*, *Prod_Name*, *Category*, *Price*),
  Location (*ShopId*, *Shop_Name*, *City*, *District*),
  Time (*TimeId*, *Date*, *Week*, *Month*, *Quarter*, *Year*)}

- And a virtual view:

```
CREATE VIEW Prod_Shop_Sales
(ProdId, Category, ShopId, City, TimeId, Sales
_Data) AS SELECT ProdId, Category, ShopId, City,
TimeId, Sales_Data
FROM Product NATURAL JOIN Location NATURAL JOIN
Sales;
```

Using Virtual Views in Queries

- Consider a query on *Prod_Shop_Sales* view (that is still not materialized) :

```
SELECT Category, City, SUM(Sales_Data)
FROM Prod_Shop_Sales
GROUP BY Category, City;
```

- It is very likely that the query processor will evaluate this query by replacing view *Prod_Shop_Sales* by its definition

Query Modification Using A Virtual View

```
SELECT Category, City, SUM(Sales_Data)
FROM (
SELECT p.ProdId, Category, l.ShopId,
City, TimeId, Sales_Data
FROM Product NATURAL JOIN Location
NATURAL JOIN Sales s)  AS view
GROUP BY Category, City;
```

View Materialization and Queries

- To speed up the repetitive execution of the previous query, the view *Prod_Shop_Sales* can be evaluated and stored
- This way doing two long lasting joins will be saved, every time you execute a query requiring join of:
  - Sales,
  - Product, and
  - Location

  tables, projected on:

  {*ProdId*, *Category*, *ShopId*, *City*, *TimeId*, *Sales_Data* }

  or some of it's proper subsets

Query Rewriting

- Suppose now the view

  `Prod_Shop_Sales`

  is **materialized**
- A clever query optimizer would rewrite the query

```
SELECT Category, City, SUM(Sales_Data)
FROM Product NATURAL JOIN Location NATURAL JOIN
Sales GROUP BY Category, City;
```

in the following way:

```
SELECT Category, City, SUM(Sales_Data)
FROM Prod_Shop_Sales GROUP BY Category, City;
```

and achieve a considerable saving

View Materialization and Queries

- The query Q:

```
SELECT Category, City, SUM(Sales_Data)
FROM Product NATURAL JOIN Location
NATURAL JOIN Sales NATURAL JOIN Time
WHERE Year = 2000
GROUP BY Category, City;
```

  can not be evaluated using **only** the materialized view *Prod_Shop_Sales*, since the view does not contain information about year

- But, the materialized view *Prod_Shop_Sales* can be joined back to*Time* in the rewritten query and thus be used to answer the query *Q*

Creating a Materialized View

- In a conventional Relational Database Management Engine:

```
CREATE TABLE <mat_view_name>…
INSERT INTO <mat_view_name> …
```

or

```
CREATE TABLE <mat_view_name>
AS
SELECT…
```

- In a Relational On Line Analytical Processing (ROLAP) Engine:

```
CREATE MATERIALIZED VIEW <mat_view_name>
AS
(SELECT…);
```

Choosing Views to Materialize

- The choice of views to materialize is complex, because the range of views that can be used for a query evaluation is very broad
- On the other hand, materialized views strongly influence the storage occupancy and DW maintenance time
- When deciding which views to materialize, one should consider the following issues:
  - How many queries potentially can be sped up,
  - How much space will be required to store the views, and
  - How will the views influence the DW maintenance (update)
- It is a goal to materialize a small, carefully chosen set of views that can be used to evaluate the majority of important queries

Two Symmetric Queries

- Consider the following two symmetric queries:

```
SELECT Category, SUM(Sales_Data)
FROM Product NATURAL JOIN Sales
GROUP BY Category;
```

```
SELECT City, SUM(Sales_Data)
FROM Location NATURAL JOIN Sales s
GROUP BY City;
```

Speeding Up Two Symmetric Queries

- There exist a number of possible ways to speed up these queries:
  - To precompute and store as materialized views both joins (*Sales* with *Product*, and *Sales* with *Location*),
  - To precompute and store as materialized views both queries in their entirety, or
  - To define and materialize the following view:

```
CREATE MATERIALIZED VIEW Prod_Loc_Sales (Category, City, Total) AS
SELECT Category, City, SUM(Sales_Data)
FROM Sales NATURAL JOIN Product NATURAL JOIN Location
GROUP BY Category, City;
```

  - There also exist some other ways

Question for You

- Suppose a clever query optimizer
- The view *Prod_Loc_Sales* (given on the previous slide) favors one of the two queries

```
SELECT Category, SUM(Total)
FROM Prod_Loc_Sales
GROUP BY Category;
```

```
SELECT City, SUM(Total)
FROM Prod_Loc_Sales
GROUP BY City;
```

- Which one and why?

Materialized View Advisor

- ROLAP Engines offer automatic advise to a DW DBA on which materialized views to build and which ones to drop
- This advising is done using statistical data gathered by Data Warehouse querying in a previous time period

Summary

- OLAP queries are typically aggregate queries over very large fact tables
- To allow fast answers, view materialization is a viable alternative
- Materialized views strongly influence the storage occupancy and DW maintenance time
- It is the goal to materialize a small, carefully chosen set of views that can be used to evaluate a majority of the most important queries