Lecture 08

Optimizations

Summary – last week

- B-Trees are not fit for multidimensional data
- R-Trees
  - MBR as geometry to build multidimensional indexes
  - Inefficient because they allow overlapping between neighboring MBRs
  - R^+-trees - improve the search performance
- UB-Trees
  - Reduce multidimensional data to one dimension in order to use B-Tree indexes
  - Z-Regions, Z-Curve, use the advantage of bit operations to make optimal jumps
- Bitmap indexes
  - Great for indexing tables with set-like attributes e.g., Gender: Male/Female
  - Operations are efficient and easy to implement (directly supported by hardware)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

This week

Optimization

1. Partitioning
2. Joins
3. MaterializedViews

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning

- Breaking the data into several **physical units** that can be handled separately
- Granularity and partitioning are key to **efficient implementation of a warehouse**
- The question is not whether to use partitioning but how to do it

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning (cont'd.)

- Why partitioning?
  - **Flexibility** in managing data
  - Smaller physical units allow
    - Easy restructuring
    - Free indexing
    - Sequential scans, if needed
    - Easy reorganization
    - Easy recovery
    - Easy monitoring

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning (cont'd.)

- In DWs, partitioning is done to improve:
  - **Business query performance**, i.e., minimize the amount of data to scan
  - **Data availability**, e.g., back-up/restores can run at the partition level
  - **Database administration**, e.g., adding new columns to a table, archiving data, recreating indexes, loading tables

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

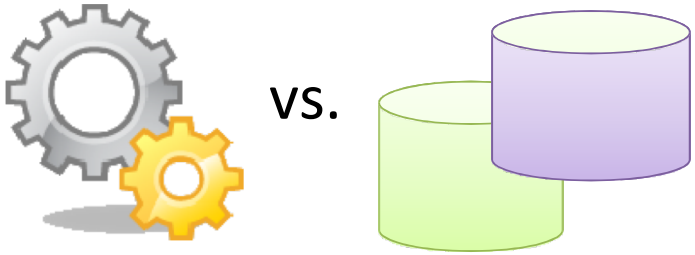
Partitioning (cont'd.)

- Possible approaches:
  - **Data partitioning**
    where data is usually partitioned by
    - Date
    - Line of business
    - Geography
    - Organizational unit
    - Combinations of these factors
  - **Hardware partitioning**
    - Hardware partitioning makes data available to different processing nodes by ensuring that sub-processes are capable of running on the different nodes

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning (cont'd.)

- At which **level** should data partitioning occur?
  - Possibilities are
    - on application
    - or DBMS level

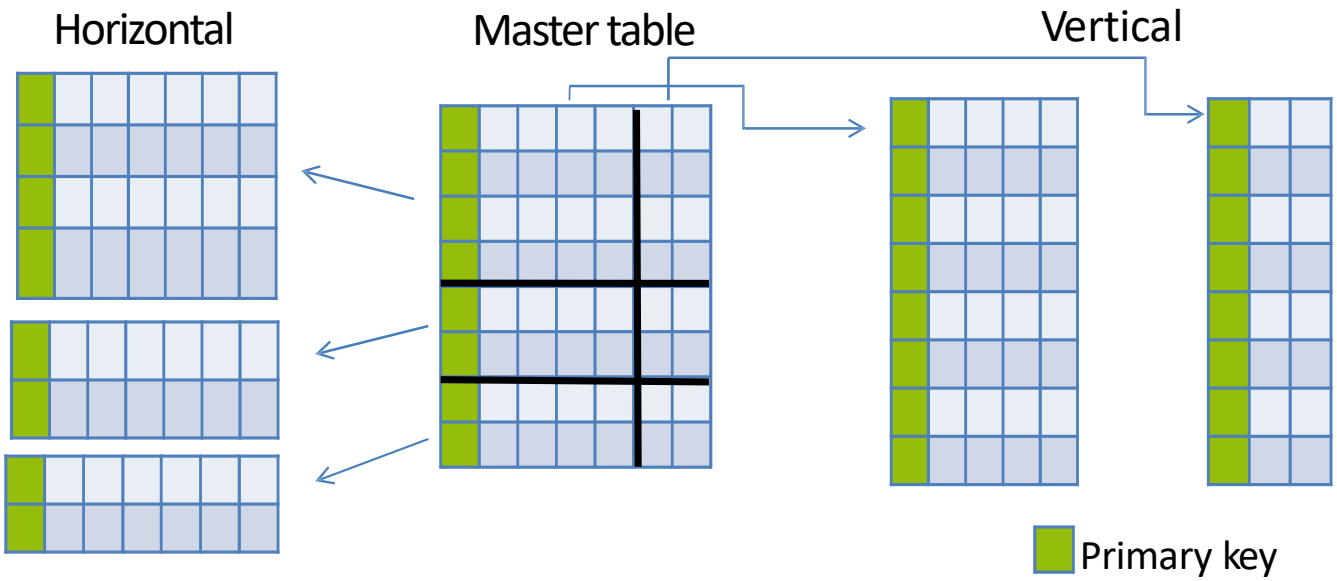


- Partitioning on DBMS level is clear, but it also makes sense to partition at **application level**
  - E.g., allows different definitions for each year
    - Important since DWs spans many years and as business evolves DWs change, too
    - Think for instance about changing tax laws

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Data Partitioning

- Data partitioning, involves:
  - Splitting out the rows of a table into multiple tables i.e., **horizontal partitioning**
  - Splitting out the columns of a table into multiple tables i.e., **vertical partitioning**



Which partitioning type is good tax laws changing every year?

Data Partitioning (cont'd.)

- Horizontal partitioning
  - The set of tuples of a table is split among **disjoint** table parts
  - Definition: A set of Relations $\{R_1, ..., R_n\}$ represent a **horizontal partitioning** of Master-Relation R if and only if $R_i \subseteq R$, $R_i \cap R_j = \emptyset$ and $R = \cup_i R_i$, for $1 \leq i, j \leq n$
  - According to the partitioning procedure we have different horizontal partitioning solutions
    - Range partitioning, List partitioning and Hash partitioning

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Horizontal Partitioning

- Range Partitioning
  - Selects a partition by determining if the partitioning key is inside a certain **range**
  - A partition can be represented as a restriction on the master-relation
    - $R_i = \sigma_{P_i}(R)$, where P_i is the partitioning predicate. The partitioning predicate can involve more attributes
      - P_1: Country = 'Germany' and Year = 2009
      - P_2: Country = 'Germany' and Year < 2009
      - P_3: Country ≠ 'Germany'

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Horizontal Partitioning (cont'd.)

- List Partitioning
  - A partition is assigned a **list of values**. If the partitioning key has one of these values, the partition is chosen
    - For example all rows where the column Country is either Iceland, Norway, Sweden, Finland or Denmark could build a partition for the Scandinavian countries
  - Is also expressed as a simple **restriction** on the master relation
    - The partitioning predicate involves just one attribute
      - P_1: City IN ('Hamburg', 'Hannover', 'Berlin')
      - P_2: City IN (DEFAULT) – represents tuples which do not fit to P_1

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Horizontal Partitioning (cont'd.)

- Hash Partitioning
  - The value of a hash function determines membership in a partition. Assuming there are four partitions, the hash function could return a value from 0 to 3
  - For each tuple t, of the master-table R, the hash function will associate it to a partition table R_i
    - $R_i = \{t_1, ..., t_m / t_j \in R \text{ and } H(t_j) = H(t_k) \text{ for } 1 \leq j, k \leq m\}$
  - This kind of partitioning is particularly used in parallel processing
  - The goal is to achieve an equal distribution of the data

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Horizontal Partitioning (cont'd.)

- Horizontal partitioning in DataWarehousing partitions data by
  - Time dimension
    - Periods, such as week or month can be used or the data can be partitioned by the age of the data
    - E.g., if the analysis is usually done on last month's data the table could be partitioned into monthly segments
  - A dimension other than time
    - If queries usually run on a grouping of data: e.g., each branch tends to query on its own data and the dimension structure is not likely to change then partition the table on this dimension
  - On table size
    - If a dimension cannot be used, partition the table by a predefined size. If this method is used, metadata must be created to identify what is contained in each partition

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Vertical Partitioning

- Vertical Partitioning
  - Involves creating tables with **fewer columns** and using additional tables to store the remaining columns
  - Different physical storage might be used e.g., storing infrequently used or very wide columns on a different device
  - Usually called **row splitting**
  - Row splitting creates a one-to-one relationship between the partitions

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Vertical Partitioning (cont'd.)

- In DW, common vertical partitioning means
  - Moving **seldom used** columns from a highly-used table to another table
  - Creating a **view** across the two newly created tables restores the original table with a performance penalty
    - However, performance will increase when accessing the highly-used data e.g., for statistical analysis

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Vertical Partitioning (cont'd.)

- In DWs with **very large** dimension tables (e.g., Amazon - a customer table with tens of millions of records) we have
  - Most of the attributes are rarely, if at all, queried
  - E.g., the address attribute is not as interesting as evaluating customers per age-group
  - But we must still maintain the link between the fact table and the **complete** customer dimension, which has **high performance costs**!

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

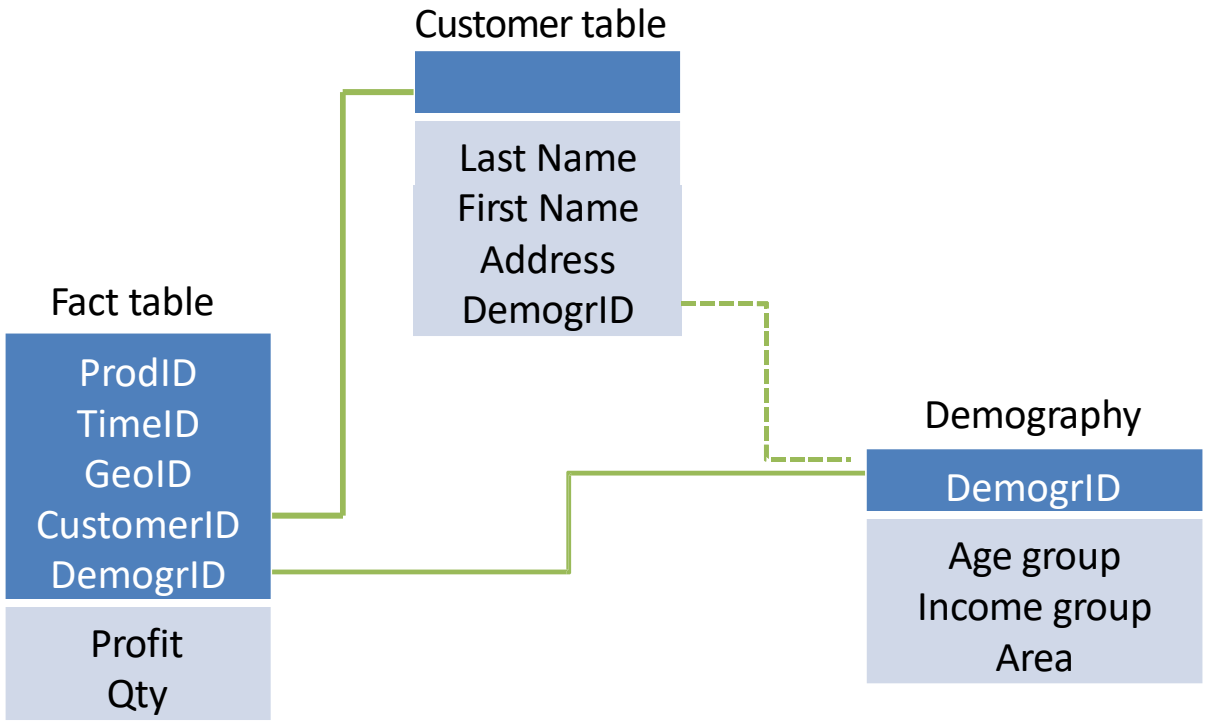
Vertical Partitioning (cont'd.)

- The solution is to use **Mini-Dimensions**, a special case of vertical partitioning
  - Many dimension attributes are used **very frequently** as browsing constraints
    - In big dimensions these constraints can be hard to find among the lesser used ones
  - Logical groups of often used constraints can be separated into **small dimensions** which are very well indexed and easily accessible for browsing

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

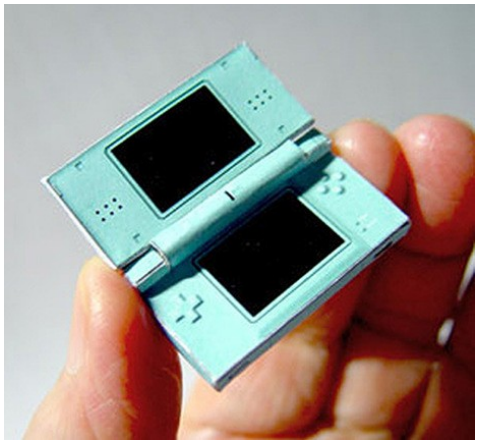
Vertical Partitioning (cont'd.)

- Mini-Dimensions, e.g., the **Demography table**



Vertical Partitioning (cont'd.)

- All variables in these mini-dimensions must be presented as **distinct classes**
- The key to the mini-dimension can be placed as a **foreign key** in both the fact table and dimension table from which it has been broken off
- Mini-dimensions, as their name suggests, should be kept small and compact



Partitioning

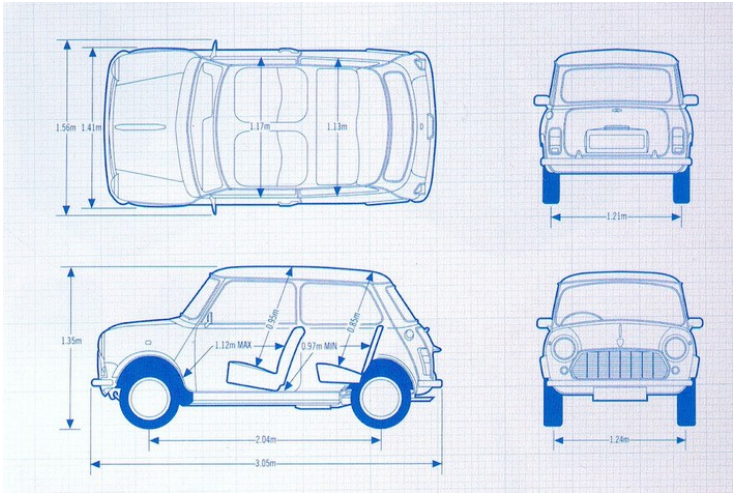
- Advantages
  - Records used together are grouped together
  - Each partition can be optimized for performance
  - Security, recovery
  - Partitions stored on different disks: contention
  - Take advantage of parallel processing capability
- Disadvantages
  - Slow retrieval across partitions (expensive joins)
  - Complexity

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning (cont'd.)

- Use partitioning when:
  - A table is > 2GB (from Oracle)
  - A Table is > 100 Million rows (praxis)
  - Think about it, if table is > 1 million rows

- Partitioning does not come for free!



Partitioning (cont'd.)

- Partitioning **management**
  - Partitioning should be transparent outside the DBMS
    - The applications work with the Master-Table at logical level
    - The conversion to the physical partition tables is performed internally by the DBMS
    - It considers also data consistency as if the data were stored in just one table
  - Partitioning transparency is not yet a standard. Not all DBMS support it!

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning Management

- Partitions in practice
  - Oracle supports Range-, List-, Hash-, Interval-, System-Partitions as well as combinations of these methods
  - E.g., partitioning in Oracle:
    - CREATETABLE SALES(


```
ProdID  NUMBER,
GeoID  NUMBER,
TimeID DATE,
Profit NUMBER)
PARTITION BY RANGE(timeID)(
        PARTITION before 2008
                VALUES LESSTHAN (TO_DATE ('01-JAN-
                2008','DD-MM-YYYY')),
        PARTITION 2008
                VALUES LESSTHAN (TO_DATE ('01-JAN-
                2009','DD-MM-YYYY'))
);
```

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning Management (cont'd.)

- Partitions in practice
  - In Oracle partitioning is performed with the help of the LESSTHAN function. How can we partition data in the current year?
    - ALTER TABLE Sales

      ADD PARTITION after2008 VALUES LESSTHAN

      (MAXVALUE);

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Partitioning Management (cont'd.)

RowID	ProdID	GeoID	TimeID	Profit
…	…	…	…	…
121	132	2	05.2007	8K
122	12	2	08.2008	7K
123	15	1	09.2007	5K
124	14	3	01.2009	3K
125	143	2	03.2009	1,5K
126	99	3	05.2007	1K
	…	…	…	…

RowID	ProdID	GeoID	TimeID	Profit
…	…	…	…	…
121	132	2	05.2007	8K
123	15	1	09.2007	5K
126	99	3	05.2007	1K

RowID	ProdID	GeoID	TimeID	Profit
122	12	2	08.2008	7K

RowID	ProdID	GeoID	TimeID	Profit
124	14	3	01.2009	3K
125	143	2	03.2009	1,5K

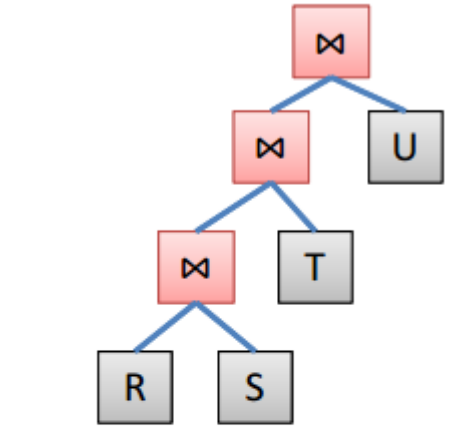
Join Optimization

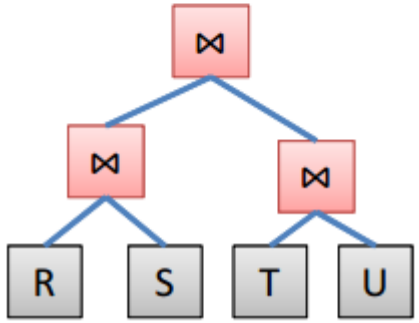
- Often queries over **several** partitions are needed
  - This results in **joins** over the data
  - Though joins are **generally expensive operations**, the overall cost of the query may strongly differ with the chosen evaluation plan for the joins
- Joins are **commutative** and **associative**
  - $R \bowtie S \equiv S \bowtie R$
  - $R \bowtie (S \bowtie T) \equiv (S \bowtie R) \bowtie T$

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Join Optimization (cont'd.)

- This allows to evaluate individual joins in **any order**
  - Results in **join trees**
    - Different join trees may show very different evaluation performance
  - Join trees have different **shapes**
  - Within a shape, there are different relation **assignments** possible
- Example: R ⋈ S ⋈ T ⋈ U





Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Join Optimization (cont'd.)

- Number of **possible join trees** grows rapidly with number of join relations
  - For n relations, there are T(n) different tree shapes
    - $T(1) = 1$
    - $T(n) = \sum_{i=1}^{n-1} T(i)T(n-i)$
    - "Any number of 1 ≤ i ≤ n-1 relations may be in the left subtree and ordered in T(i) shapes while the remaining n-i relations form the right subtree and can be arranged in T(n-i) shapes."

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

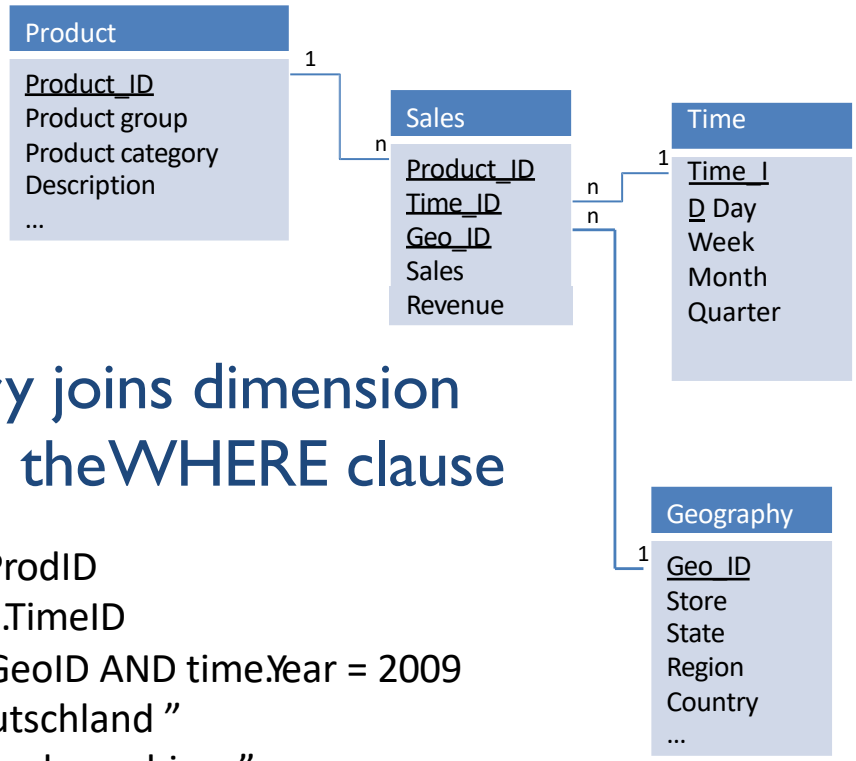
Join Optimization (cont'd.)

- Optimizer has 3 choices
  - Consider all possible join trees
    - Usually not possible
  - Consider a subset of all trees
    - i.e. restrict to trees of certain shapes
  - Use heuristics to pick a certain shape

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Join Optimization in DW

- Relational optimization of **star-joins**
  - Star schema comprises a **big** fact table and many **small** dimension tables



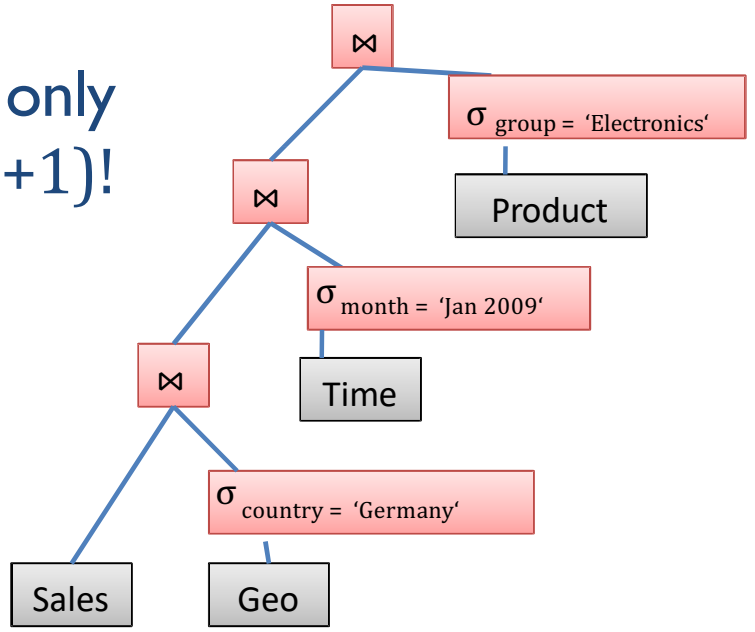
  - An **OLAP SQL** query joins dimension and fact tables usually in theWHERE clause

```
sales.ProdID = product.ProdID
AND sales.TimeID = time.TimeID
AND sales.GeoID = geo.GeoID AND time.Year = 2009
AND geo.Country = "Deutschland "
AND product.group = "wash machines"
```

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Join Optimization in DW (cont'd.)

- If the OLAP query specifies **restrictions** or **group by's** on n dimensions, an $n+1$ order join is necessary
  - Joins can be performed only pair-wise, resulting in $(n+1)!$ possible joining orders

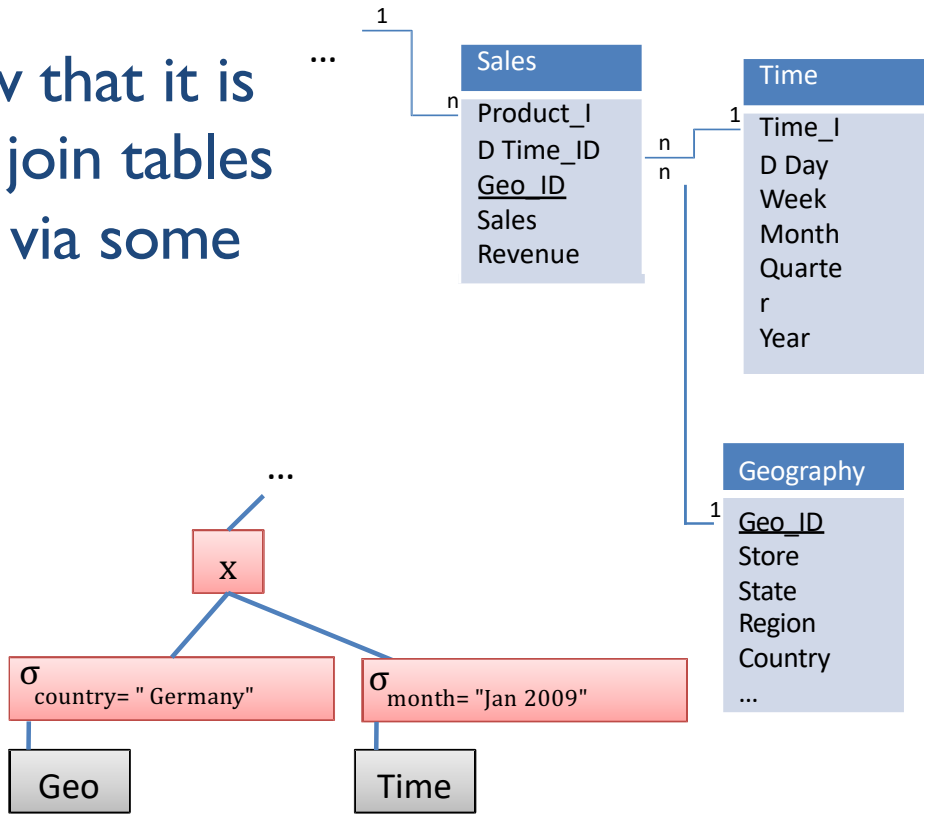


1. $\sigma_{country=Germany}(Sales \bowtie Geo)$
2. $Sales \bowtie (\sigma_{country=Germany} Geo)$

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

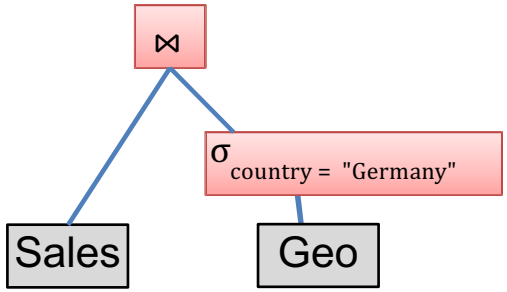
Join Heuristics

- To reduce the number of join-orders, **heuristics** are used
  - In OLTP heuristics show that it is **not a good idea** to join tables that are not connected via some attribute to each other
    - E.g., Geo with Time relation leads to a **Cartesian product**



Join Heuristics (cont'd.)

- But this heuristic rule from OLTP is **not suitable** for DW!
  - E.g., join Sales with Geo in the following case:
    - Sales has 10 mil records, in Germany there are 10 stores, in January 2009 there were products sold in 20 days, and the Electronics group has 50 products

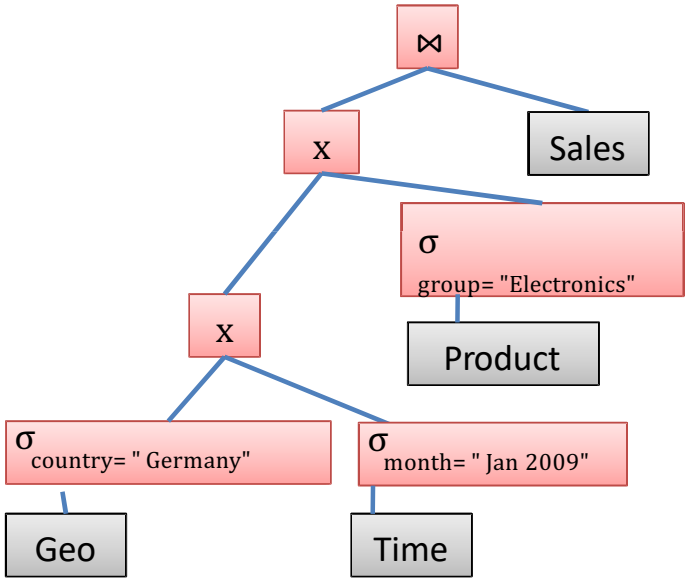


  - If 20% of our sales were performed in Germany, still we get 2mil intermediate results.
  - Also as we are interested in only one country from Geo dimension table so index on Geo table would not help that much either.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Dimensional Cross Product

- In star-joins a **cross product** of the dimension tables is recommended
  - Geo dimension – 10 stores
  - Time dimension 20 days
  - Product dimension 50 products
  - 10*20*50 = 10 000 records after performing the cross product of the dimensions
  - The selectivity is in this case 0.1% which is fit for using an index



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

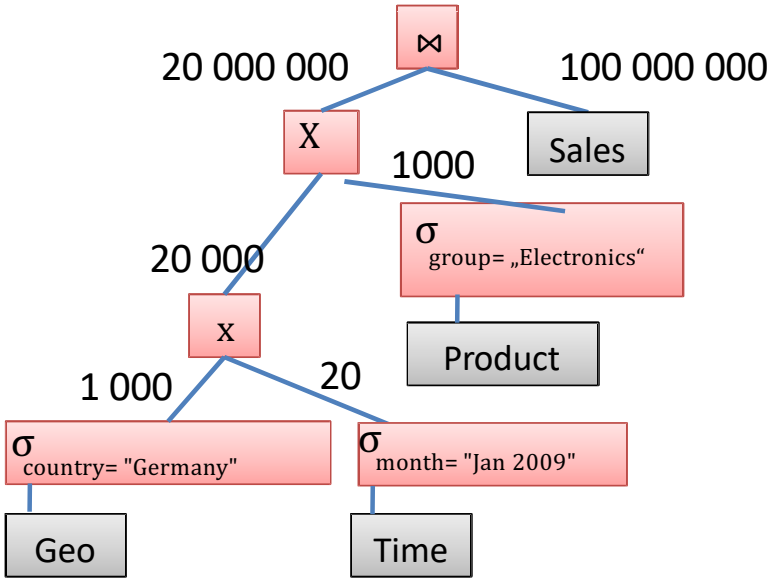
Dimensional Cross Product (cont'd.)

- But dimension cross products can also become **expensive**
  - If the restrictions on the dimensions are not restrictive enough or if there are many dimension tables
    - E.g., if we would have query on 1000 stores and 1000 customers, the cross product would comprise 1000 * 20 * 50 * 1000 = 100 mil records…

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

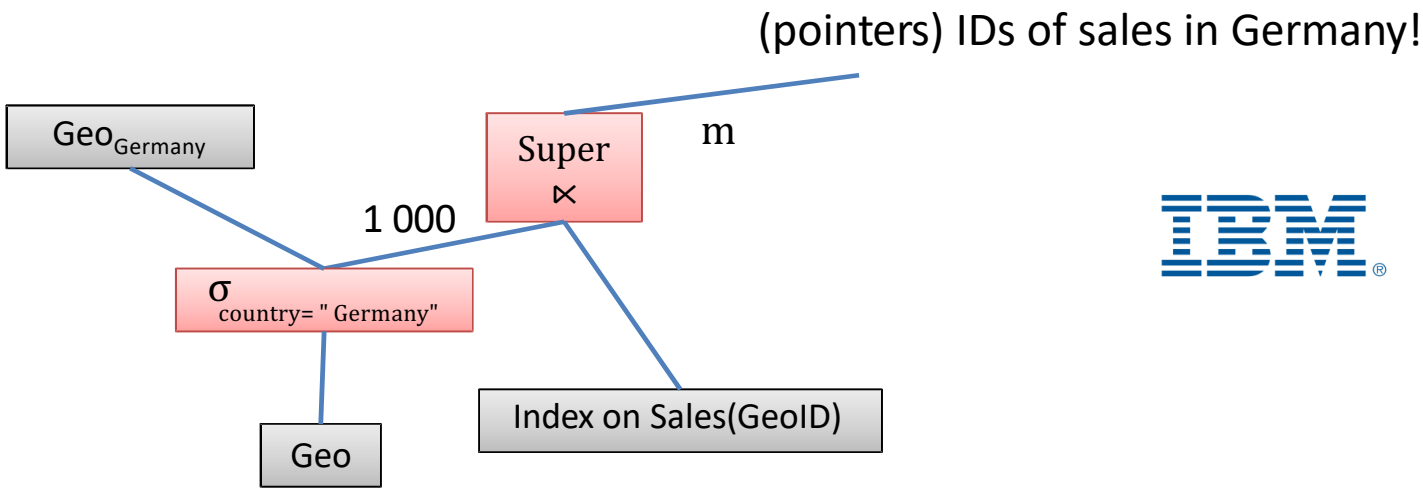
Dimensional Cross Product (cont'd.)

- E.g., In Germany the company has 1000 stores, it sold in 20 days in January, and it has 1000 types of electronics products
  - 1000 * 20 * 1000=20 mil records…



Star-join Optimization

- The **IBM DB2** solution for expensive dimension cross products is to build **semi-joins** of the dimension tables with indexes
  - A B*-Tree index will be on the fact table (sales) for each dimension (B*-Trees are unidimensional)



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

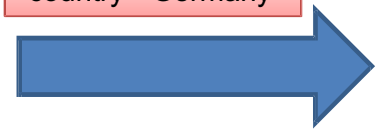
Star-join Optimization (cont'd.)

Geo

GeoID	Store	City	Country
1	S1	BS	Germany
2	S2	BS	Germany
3	S3	HAN	Germany
4	S4	Lyon	France

GeoID	ID
1	1
1	2
4	3
…	…

$\sigma_{country=\text{“Germany”}}$



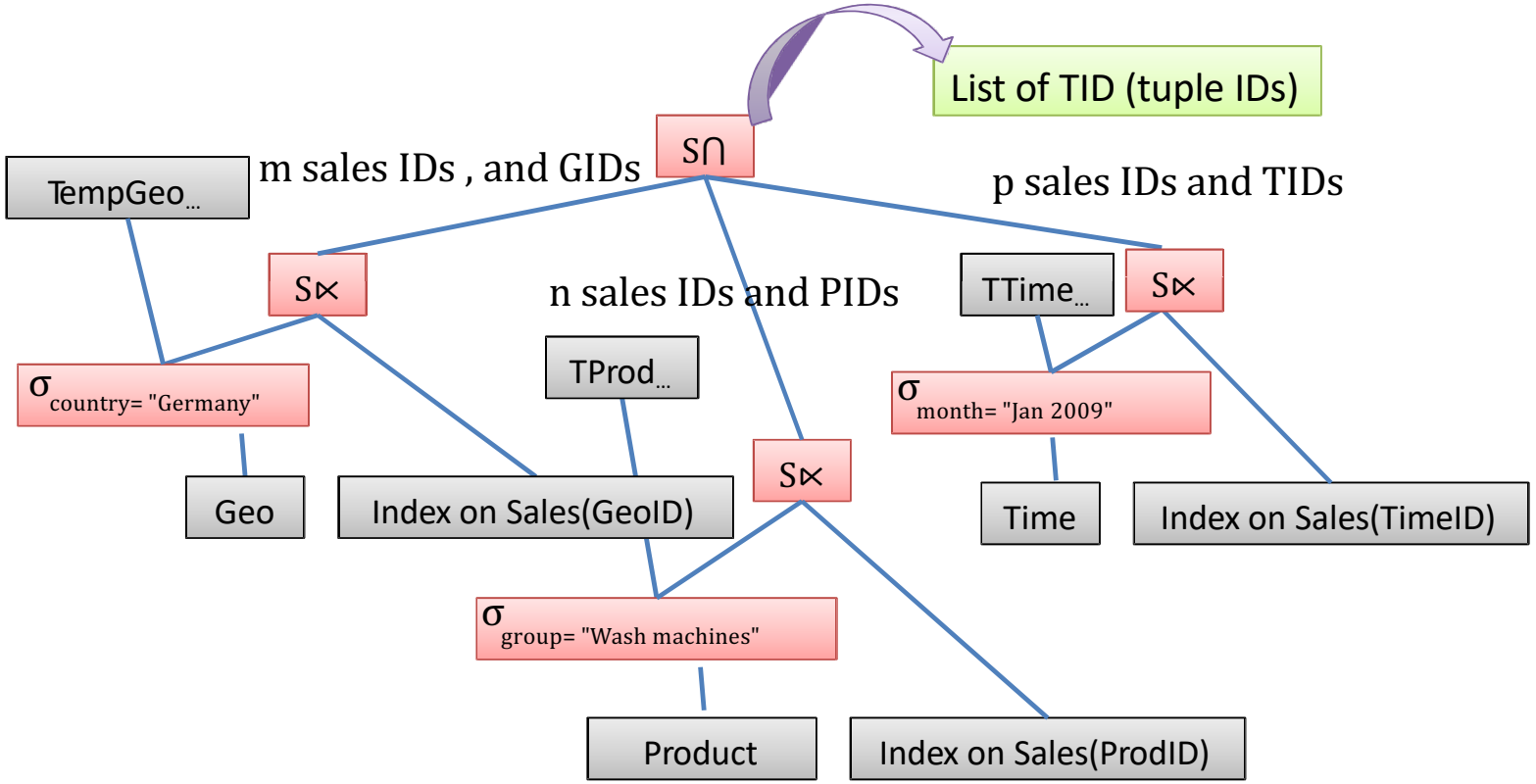
GeoID	Store	City	Country
1	S1	BS	Germany
2	S2	BS	Germany
3	S3	HAN	Germany

GeoID	ID
1	1
1	2

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Star-join Optimization (cont'd.)

- Reduces the fact table to what we need, based on indexes



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Materialized Views

- MaterializedViews (MV)
  - Views whose tuples are **stored** in the database are said to be materialized
  - They provides fast access, like a (very high-level) cache
  - Need to maintain the view as the underlying tables change
    - Ideally, we want incremental view maintenance algorithms

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Materialized Views (cont'd.)

- How can we use MV in DW?
  - E.g., we have queries requiring us to join the Sales table with another table and aggregate the result
    - SELECT P.Categ, SUM(S.Qty) FROM Product P, Sales S WHERE P.ProdID=S.ProdID GROUP BY P.Categ
    - SELECT G.Store, SUM(S.Qty) FROM Geo G, Sales S WHERE G.GeoID=S.GeoID GROUP BY G.Store
    - ….
  - There are more solutions to speed up such queries
    - Pre-compute the two joins involved (product with sales and geo with sales)
    - Pre-compute each query in its entirety
    - Or use an already materialized view

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Materialized Views (cont'd.)

- Having the following view materialized
  - CREATE MATERIALIZED VIEW Totalsales (ProdID, GeoID, total) AS SELECT S.ProdID, S.GeoID, SUM(S.Qty) FROM Sales S GROUP BY S.ProdID, S.GeoID
- We can use it in our 2 queries
  - SELECT P.Categ, SUM(T.Total) FROM Product P, Totalsales T WHERE P.ProdID=T.ProdID GROUP BY P.Categ
  - SELECT G.Store, SUM(T.Total) FROM Geo G, Totalsales T WHERE G.GeoID=T.GeoID GROUP BY G.Store

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Materialized Views (cont'd.)

- MV issues
  - **Utilization**
    - What views should we materialize, and what indexes should we build on the pre-computed results?
  - **Choice of materialized views**
    - Given a query and a set of materialized views, can we use the materialized views to answer the query?
  - **Maintenance**
    - How frequently should we refresh materialized views to make them consistent with the underlying tables?
    - And how can we do this incrementally?

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

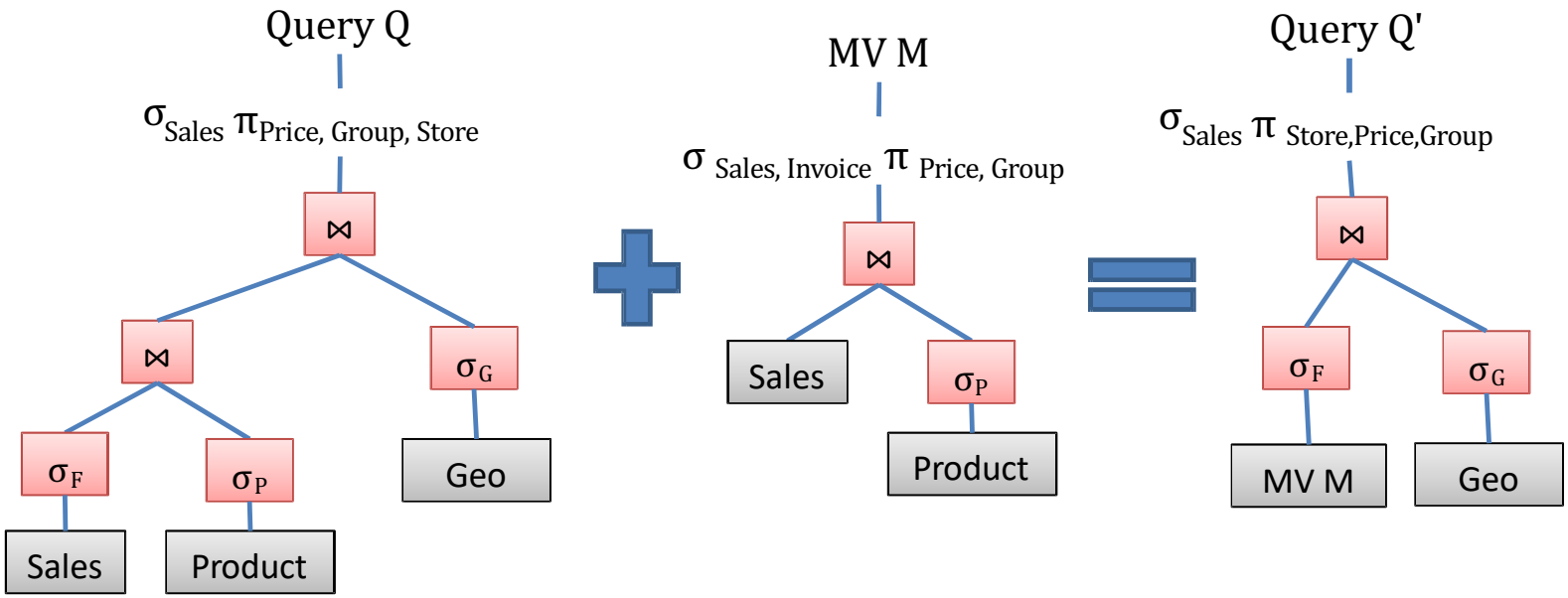
Utilization of MV

- Materialized views **utilization** has to be transparent
  - Queries are internally rewritten to use the available MVs by the **query rewriter**
  - The query rewriter performs integration of the MV based on the **query execution graph**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Utilization of MV (cont'd.)

- E.g., materialized views utilization, mono-block query



Integration of MV

- Integration of MV
  - **Valid replacement**: A query Q` represents a valid replacement of query Q by utilizing the materialized view M, if Q and Q` always deliver the same result set
  - For general relational queries, the problem of finding a valid replacement is **NP-complete**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Integration of MV (cont'd.)

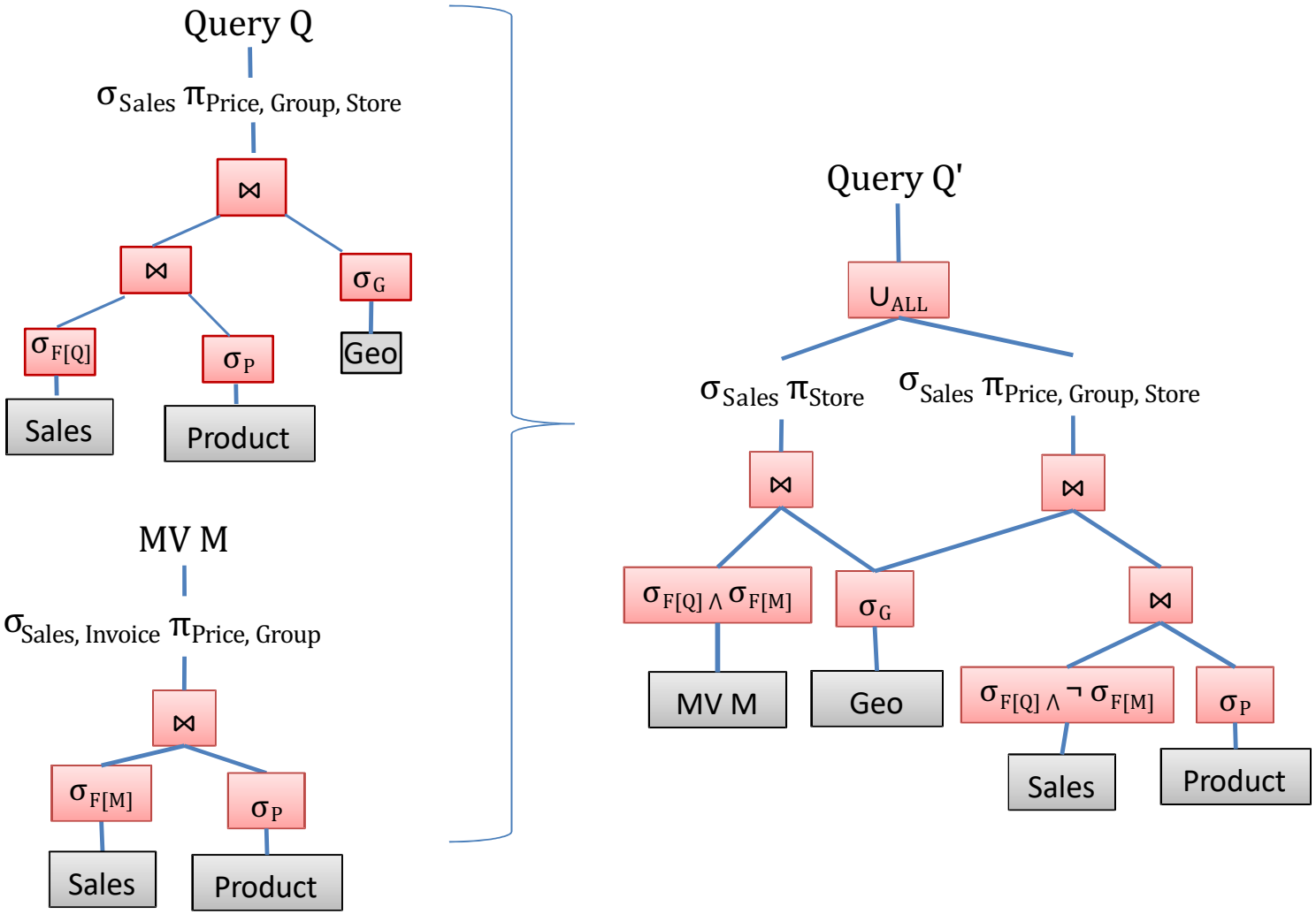
- In order to be able to integrate MV M in Q and obtain Q', the following **conditions** need to be respected
  - The selection condition in M cannot be more restrictive than the one in Q
  - The projection from Q has to be a subset of the projection from M
  - It has to be possible to derive the aggregation functions of $\pi(Q)$ from $\pi(M)$
  - Additional selection conditions in Q have to be possible also on M

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Integration of MV (cont'd.)

- How do we use MV even when there is no perfect match?
- If the selection in M is more restrictive than the selection in Q
  - Split Q in Q_a and Q_b such that $\sigma(Q_a) = (\sigma(Q) \wedge \sigma(M))$ and $\sigma(Q_b) = (\sigma(Q) \wedge \neg \sigma(M))$
  - Multiblock queries

Integration of MV (cont'd.)



Summary

- Partitioning: Horizontal or Vertical
  - Records used together are grouped together
  - However: slow retrieval across partitions
  - Mini-Dimensions
- Joins: for DW it is sometimes better to perform cross product on dimensions first
- Materialized Views: we can't materialize everthing
  - Static or Dynamic choice of what to materialize
  - The benefit cost function is decisive

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Next Lecture

- Queries!
  - OLAP queries
  - SQL for DW
  - MDX