Lecture 14

Optimizations (Continued)

Summary – last week

Optimization

1. Partitioning

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

This week

Optimization

1. Joins
2. MaterializedViews

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

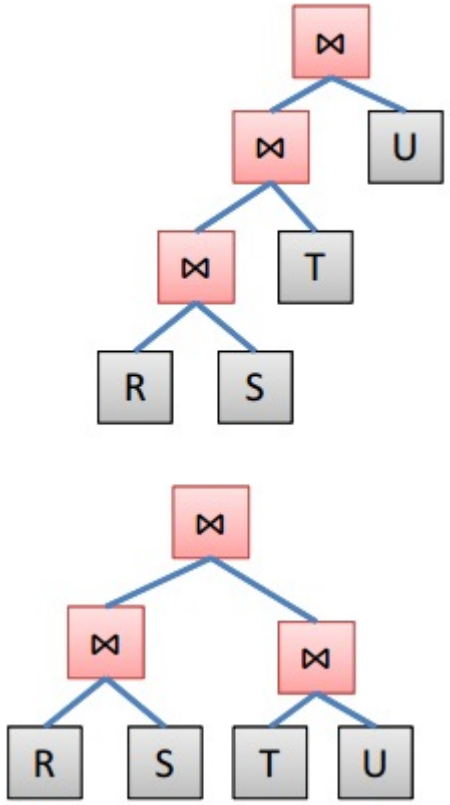
Join Optimization

- 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

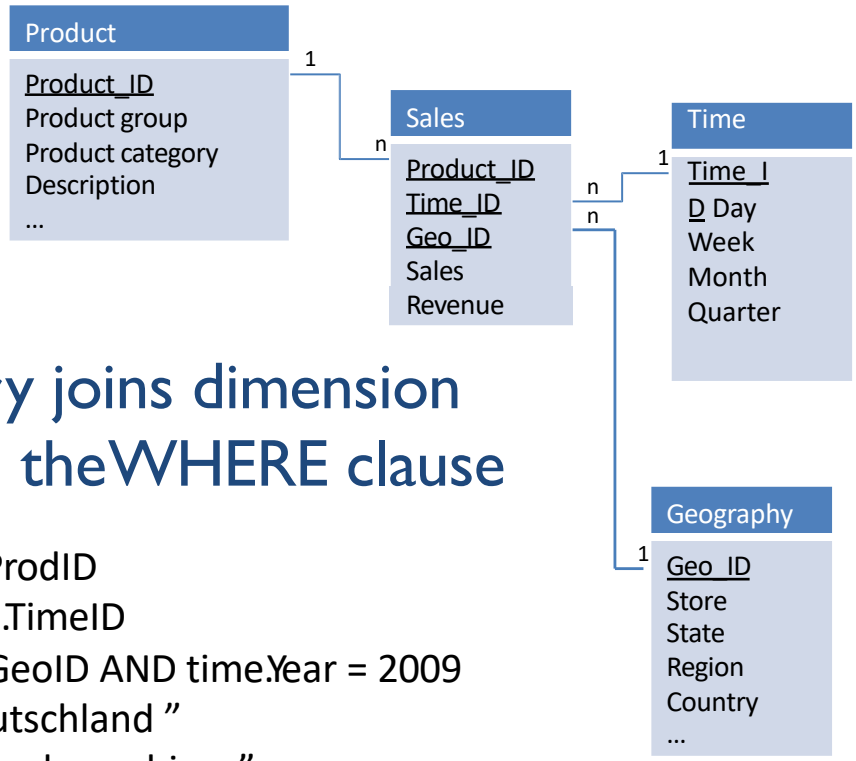
Join Optimization (cont'd.)

- 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



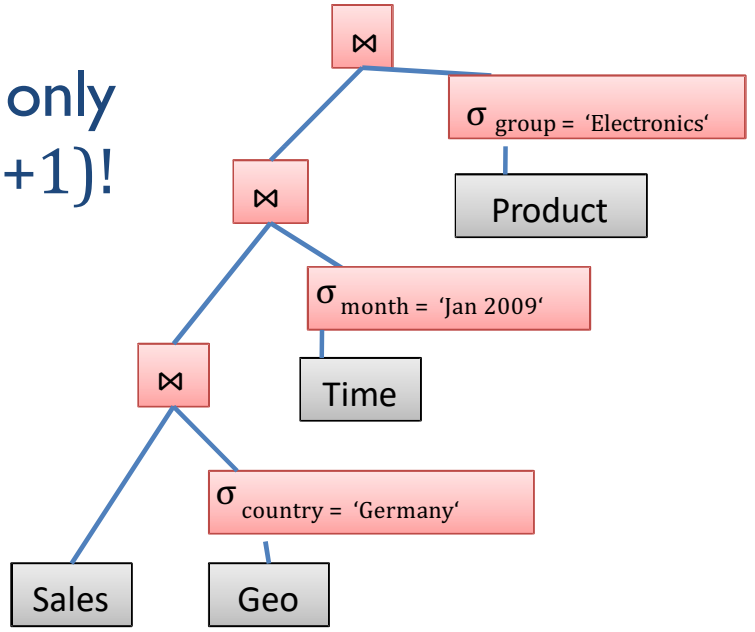
  - An **OLAP SQL** query joins dimension and fact tables usually in theWHERE clause

```
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.)

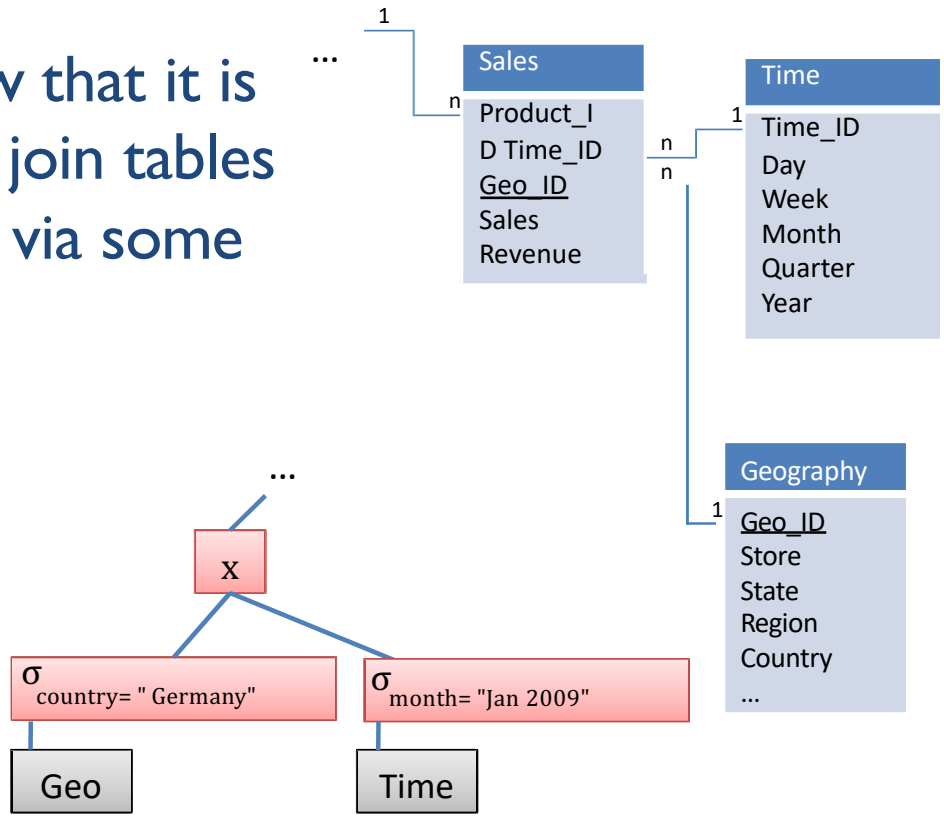
- If the OLAP query specifies **restrictions** or **group by's** on n dimensions, an $n+1$ order join is necessary
  - Joins can be performed only pair-wise, resulting in $(n+1)!$ possible joining orders



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

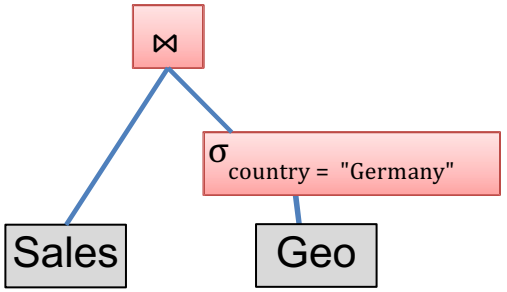
Join Heuristics

- 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 2016 there were products sold in 20 days, and the Electronics group has 50 products

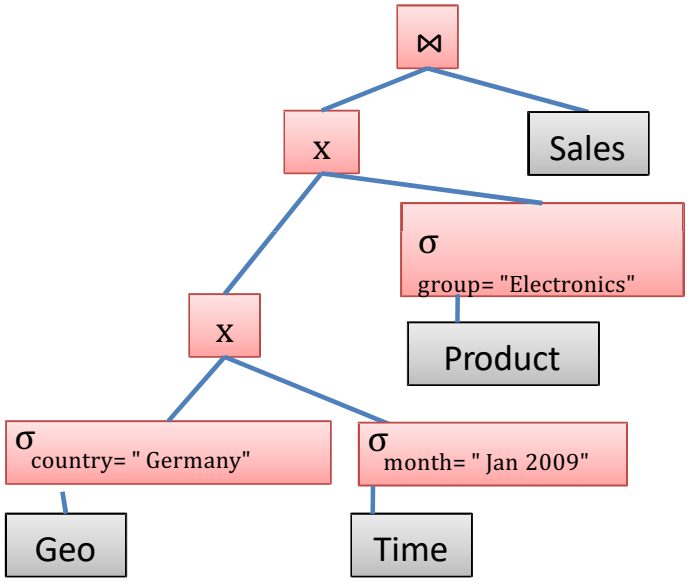


  - If 20% of our sales were performed in Germany, still we get 2mil intermediate results so index on Geo table would not help that much either.

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

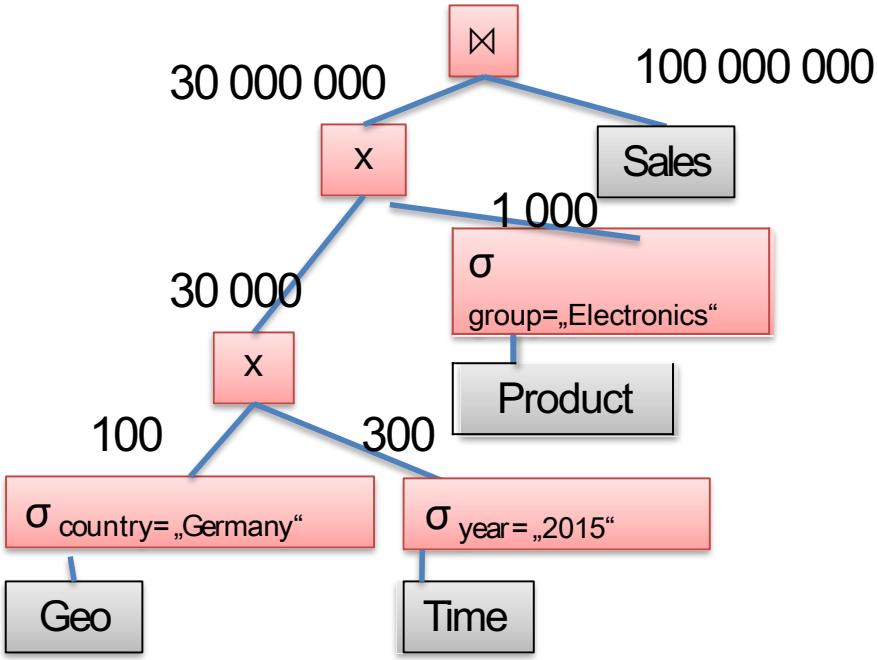
Dimensional Cross Product (cont'd.)

- 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. query for the sales of all electronic products of a company in 2015:
  - The company has **100 stores** in Germany and it sells **1000** types of electronics **products**
  - In 2015 it sold products in **300 working days**

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

Dimensional Cross Product (cont'd.)

•100 stores * 300 days * 1.000 products = **30 mil records…**

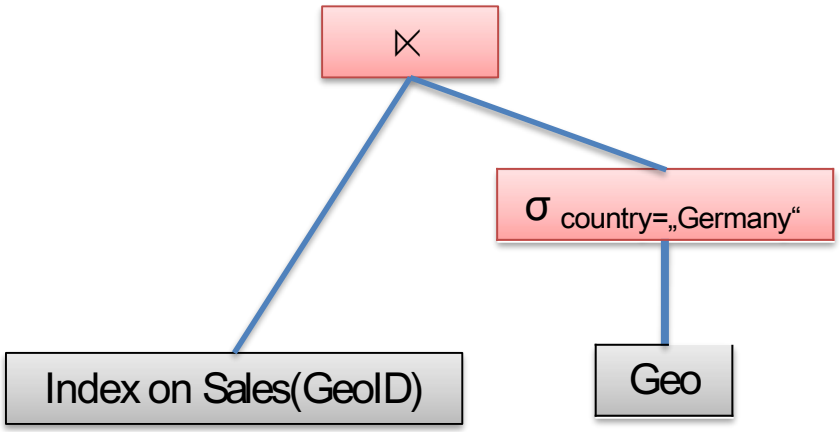


- Very **expensive** to compute

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

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



Keep all index entries for the Sales fact table for sales in Germany

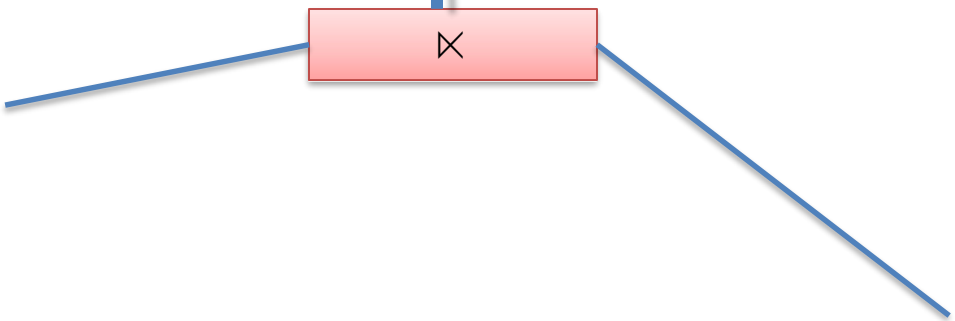
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Star-join Optimization (cont'd.)

Index on Sales for GeoID

GeoID	ID
1	1
1	2
4	3
…	…

GeoID	ID
1	1
1	2



Ge

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

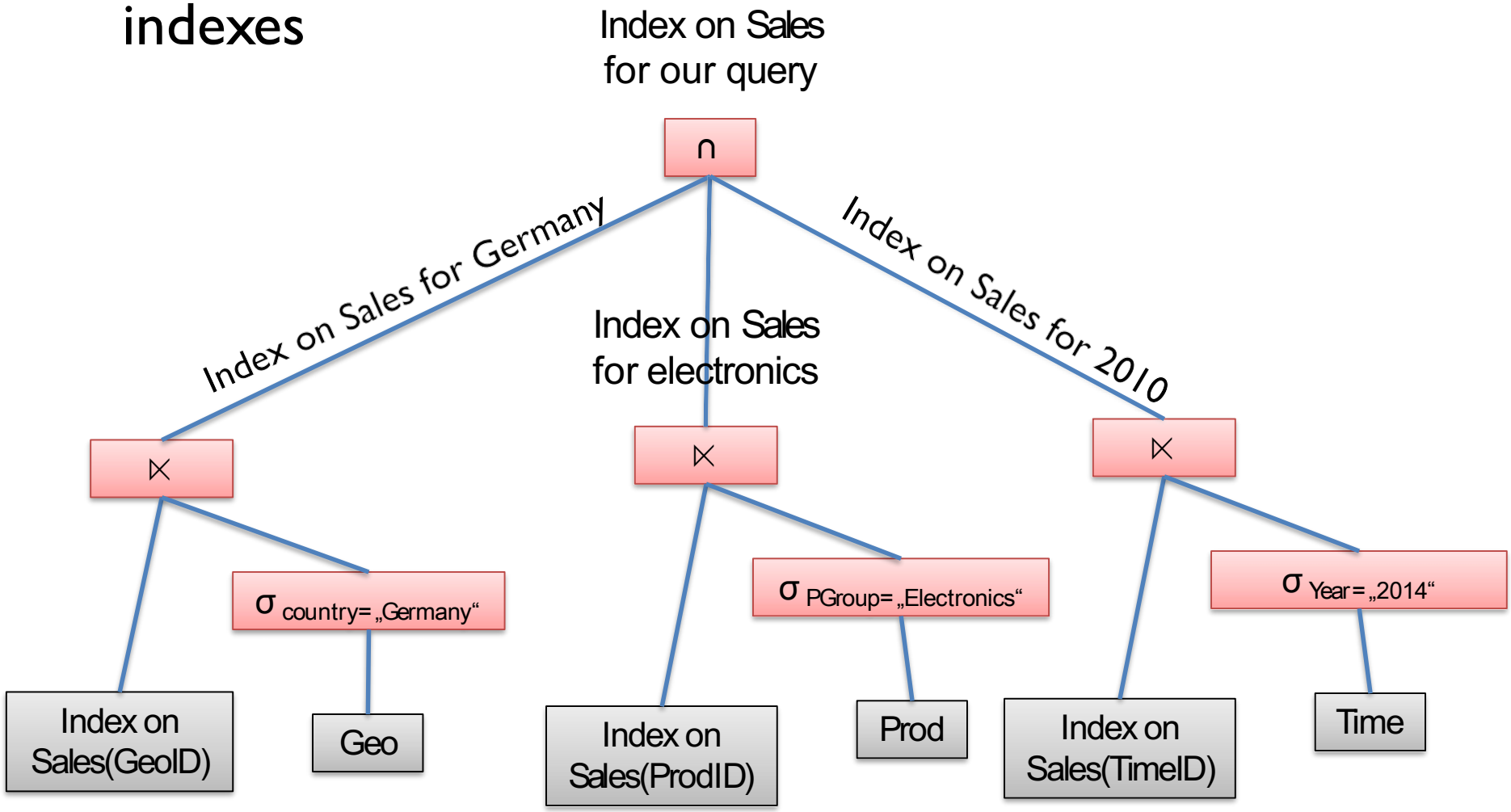
σ country=„Germany“

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

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

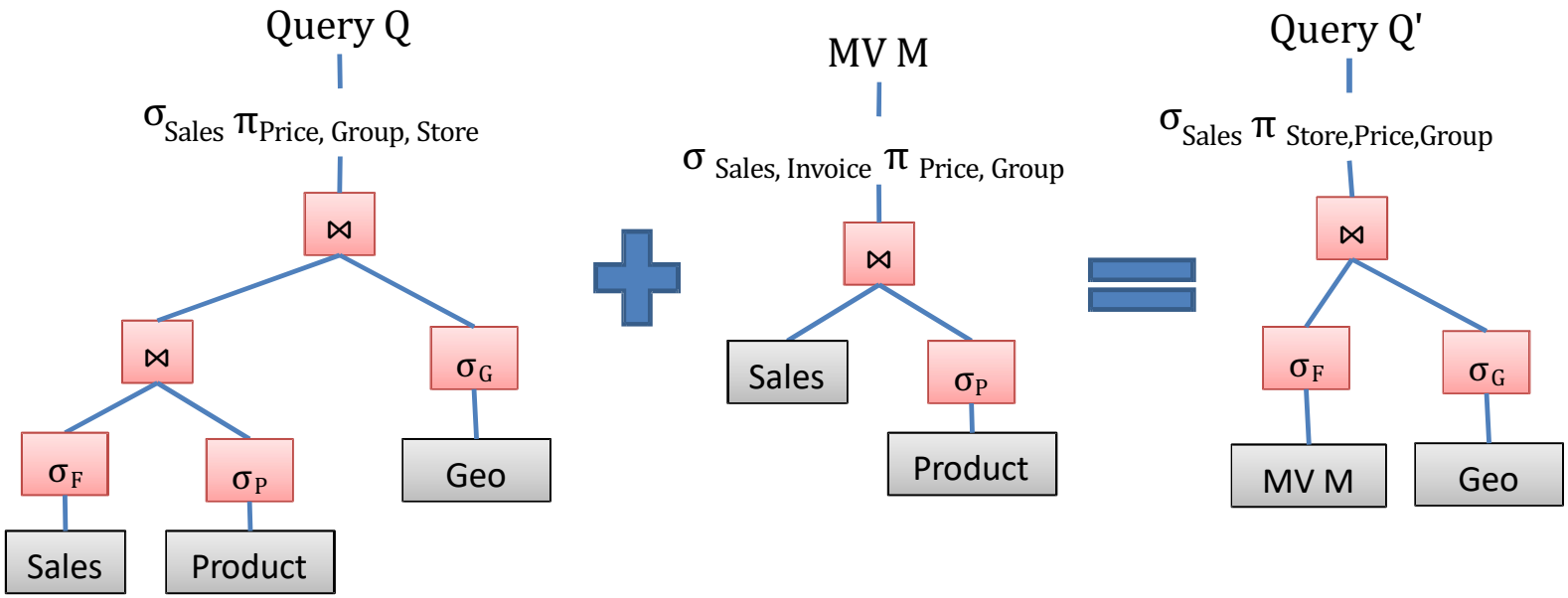
Utilization of MV

- 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

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

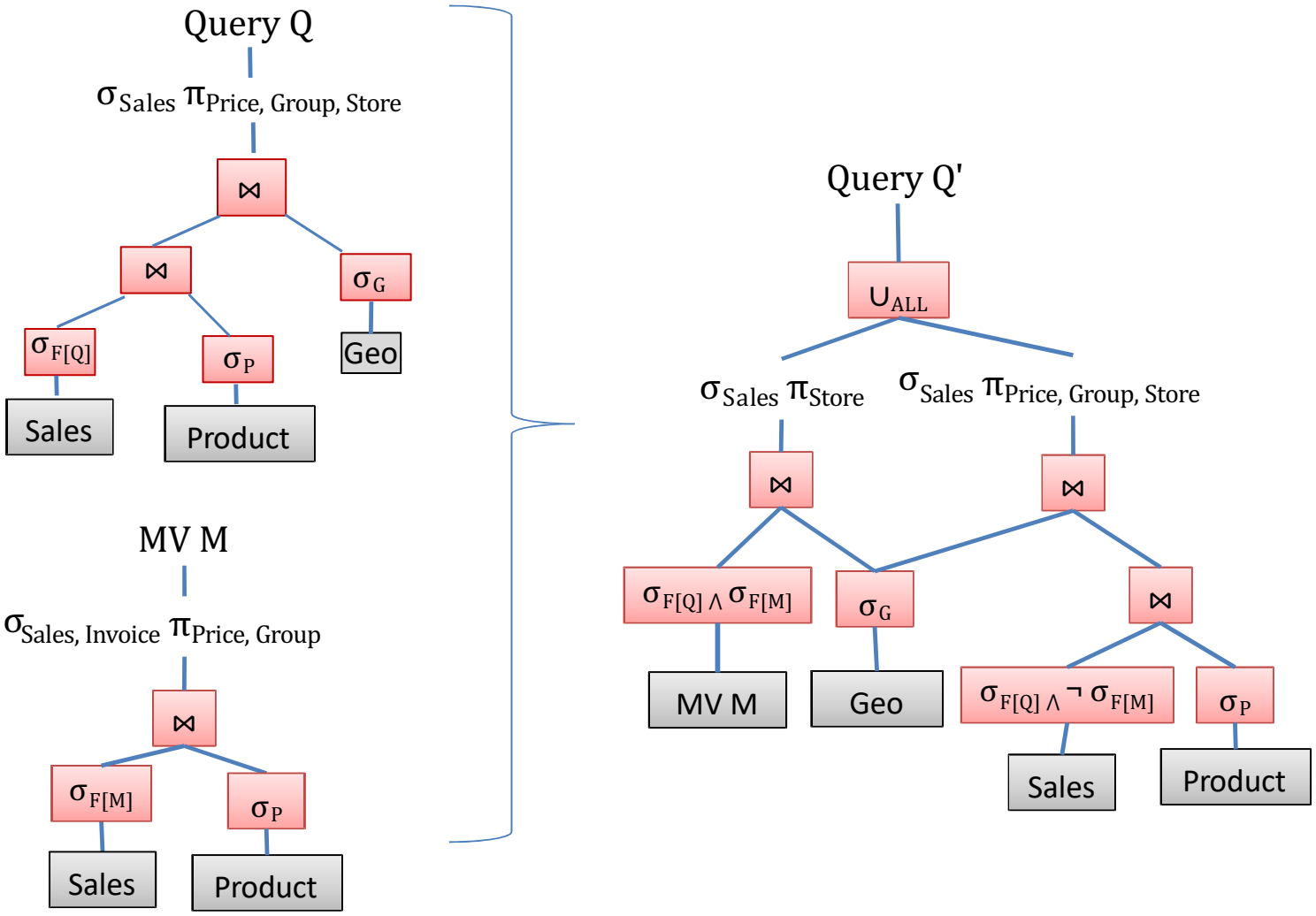
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Integration of MV (cont'd.)

- How do we use MV even when there is no perfect match? (Multiblock queries)
- 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
  - $\sigma(Q_b) = (\sigma(Q) \wedge \neg \sigma(M))$

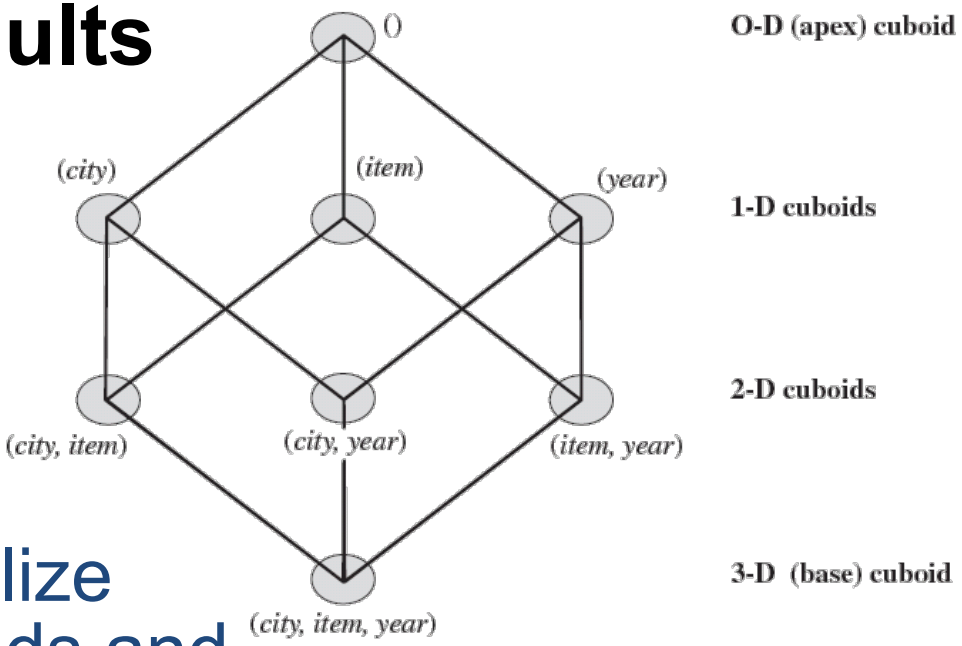
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Integration of MV (cont'd.)



MVs in DWs

- In DW, materialized views are often used to store **aggregated results**
  - The number of nodes in the lattice of cuboids is
    - $|n| = \prod_{j=1}^{n} 2 = 2^n$
  - n = 3, |n| = 8 and we would need to materialize 2-D cuboids 1-D cuboids and 0D cuboids; in total 7 views
  - n = 16, |n| = 65534, … too much to materialize
  - **What should we materialize?**



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

Maintenance of MV

- Maintenance of MV
  - Keeping a materialized view **up-to-date** with the underlying data
  - Important questions
    - **How** do we refresh a view when an underlying table is refreshed?
    - **When** should we refresh a view in response to a change in the underlying table?

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

How to Refresh a MV

- Materialized views can be maintained by re-computation on **every update**
  - – Not the best solution
- A better option is **incremental view maintenance**

When to Refresh a MV

- **Immediate**
  - As part of the transaction that modifies the underlying data tables
    - Advantage: materialized view is always consistent
    - Disadvantage: updates are slowed down
- **Deferred**
  - Some time later, in a separate transaction
    - Advantage: can scale to maintain many views without
    - slowing updates
    - Disadvantage: view briefly becomes inconsistent

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

Summary

- Joins: for DW it is sometimes better to perform cross product on dimensions first
- MaterializedViews: 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

- OLAP Queries!