Lecture 13

Optimizations

Summary – last week

- R+ Tree
  - Removes the overlaps in R Tree
- 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

Why Partitioning?

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

Partitioning (cont'd.)

- Why partitioning?
  - **Flexibility** in managing data
  - Smaller physical units allow
    - Easy restructuring e.g. adding weekly classification level
    - Cheap 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

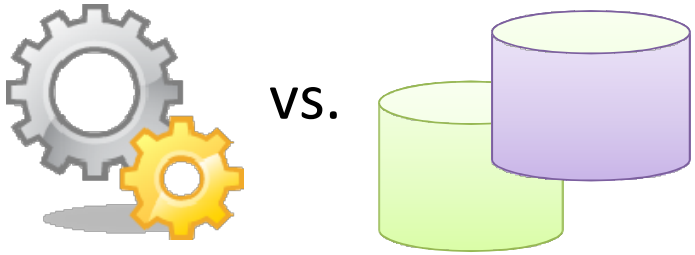
Partitioning (cont'd.)

- 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

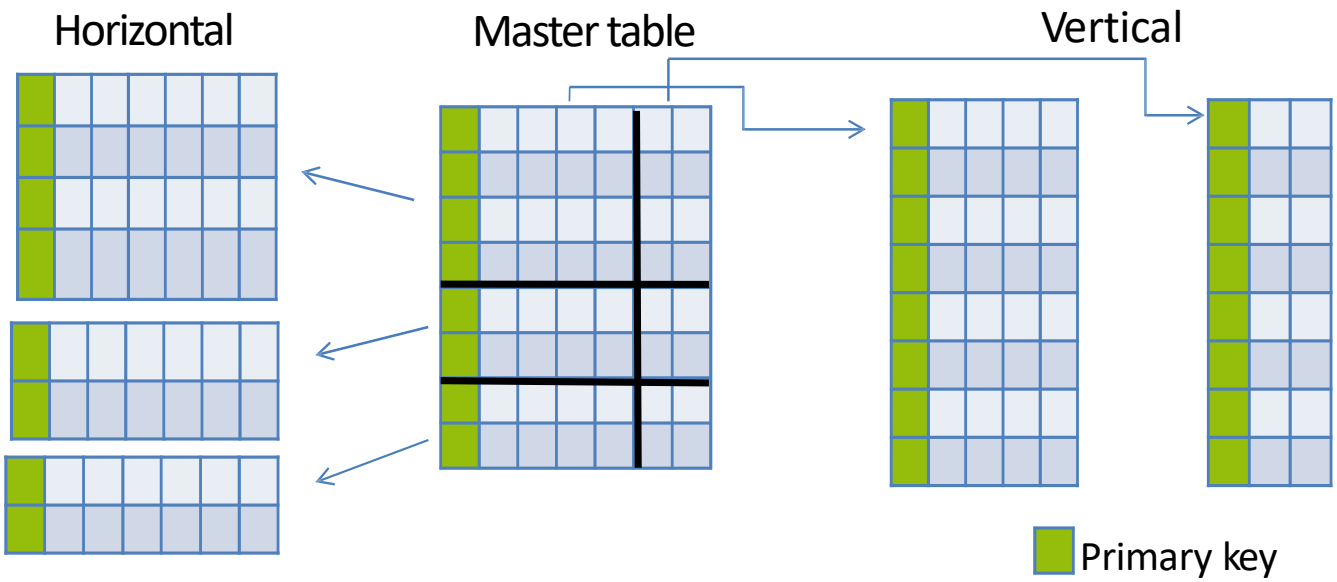


- Partitioning on DBMS level is clear, but it also makes sense to partition at **application level**
  - E.g., allows different definitions for each year
    - Important since DWs spans many years and as business evolves DWs change, too
    - Think for instance about changing tax laws

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 $R = U_i R_i$, 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

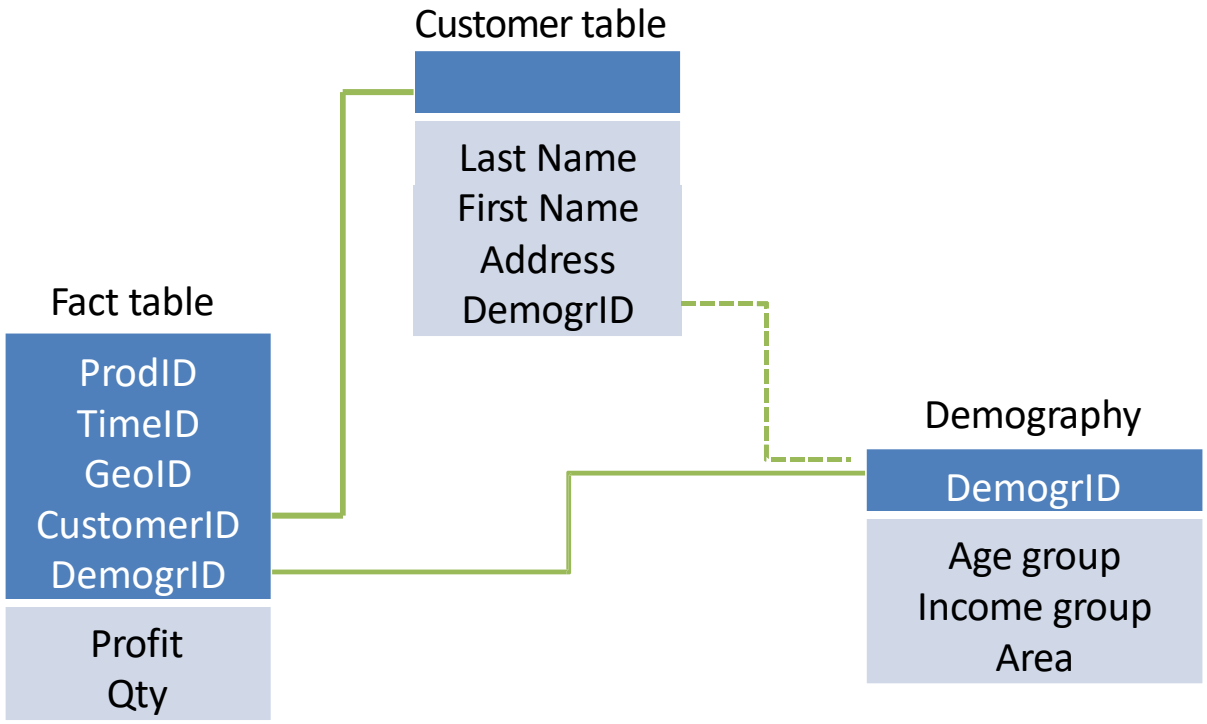
Vertical Partitioning (cont'd.)

- 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.)

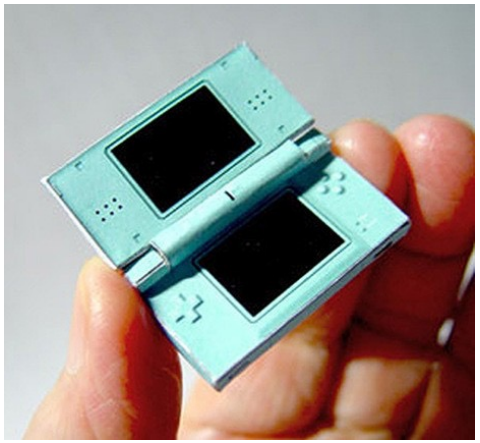
- Mini-Dimensions, e.g., the **Demography table**



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

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



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

Partitioning

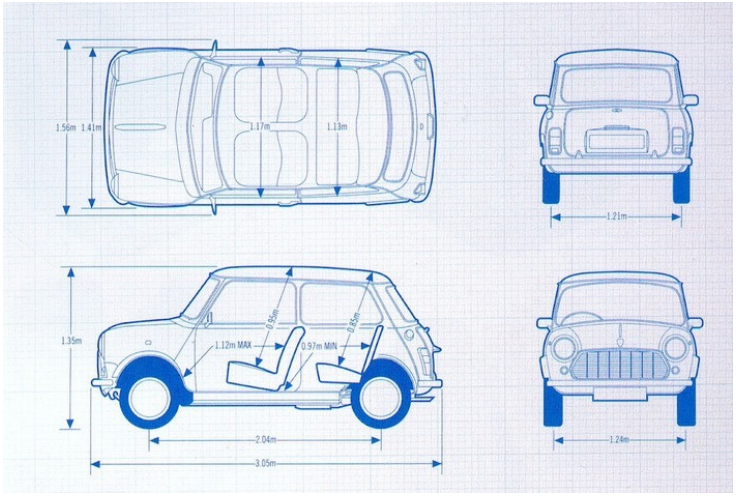
- Advantages
  - Records used together are grouped together
  - Each partition can be optimized for performance
  - Security, recovery
  - Partitions stored on different disks: contention
  - Take advantage of parallel processing capability
- Disadvantages
  - Slow retrieval across partitions (expensive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

Summary

- Partitioning: Horizontal or Vertical
  - Records used together are grouped together
  - However: slow retrieval across partitions
  - Mini-Dimensions

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

Next Lecture

1. Joins
2. MaterializedViews

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