6. What is Datawarehousing?

- **Problem:** Data Analysis, Prediction, Mining  
  - Example: Walmart  
  - Transactional databases  
    - Run many “cheap” updates concurrently  
    - E.g., each store has a database storing its stock and sales  
  - Complex Analysis over Transactional Databases?  
    - Want to analyze across several transactional databases  
      - E.g., compute total Walmart sales per month  
      - Distribution and heterogeneity  
    - Want to run complex analysis over large datasets  
      - Resource consumption of queries affects normal operations on transactional databases

- **Solution:**  
  - Performance  
    - Store data in a different system (the datawarehouse) for analysis  
    - Bulk-load data to avoid wasting performance on concurrency control during analysis  
  - Heterogeneity and Distribution  
    - Preprocess data coming from transactional databases to clean it and translate it into a unified format before bulk-loading

6. Datawarehousing Process

- 1) Design a schema for the warehouse  
- 2) Create a process for preprocessing the data  
- 3) Repeat  
  - A) Preprocess data from the transactional databases  
  - B) Bulk-load it into the warehouse  
  - C) Run analytics

6. Overview

- The multidimensional datamodel (cube)  
  - Multidimensional data model  
  - Relational implementations  
- Preprocessing and loading (ETL)  
- Query language extensions  
  - ROLL UP, CUBE, …  
- Query processing in datawarehouses  
  - Bitmap indexes  
  - Query answering with views  
  - Self-tuning
6. Multidimensional Datamodel

• Analysis queries are typically aggregating lower level facts about a business
  – The revenue of Walmart in each state (country, city)
  – The amount of toy products in a warehouse of a company per week
  – The call volume per zip code for the Sprint network
  – …

• Commonality among these queries:
  – At the core are facts: a sale in a Walmart store, a toy stored in a warehouse, a call made by a certain phone
  – Data is aggregated across one or more dimensions
    • These dimensions are typically organized hierarchically: year – month – day – hour, country – state - zip
  • Example
    – The revenue (sum of sale amounts) of Walmart in each state

6. Example 2D

<table>
<thead>
<tr>
<th></th>
<th>1st Quarter</th>
<th>2nd Quarter</th>
<th>3rd Quarter</th>
<th>4th Quarter</th>
<th>1st Quarter</th>
<th>2nd Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>toy</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>puppet</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>fishing rod</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Mobile device</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>King Lear</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

6. Generalization to multiple dimensions

• Given a fixed number of dimensions
  – E.g., product type, location, time
• Given some measure
  – E.g., number of sales, items in stock, …
• In the multidimensional datamodel we store facts: the values of measures for a combination of values for the dimensions

6. Dimensions

• Purpose
  – Selection of descriptive data
  – Grouping with desired level of granularity
• A dimension is define through a containment-hierarchy
• Hierarchies typically have several levels
• The root level represents the whole dimensions
• We may associate additional descriptive information with a elements in the hierarchy (e.g., number of residents in a city)
6. Dimension Example

- **Location**
  - **Levels:** location, state, city

    ![Diagram of Location Dimensions](image)

6. Dimension Schema Example

- **Schema of Location Dimension**
  - Set of categories $D = \{\text{location}, \text{state}, \text{city}\}$
  - Partial order
    - $\text{city} \rightarrow \text{state}, \text{city} \rightarrow \text{location}, \text{state} \rightarrow \text{location}$
  - $\text{Top}_D = \text{location}$
  - $\text{D}_{\text{min}} = \text{city}$

6. Cells, Facts, and Measures

- Each cell in the cube corresponds to a combination of elements from each dimension
  - **Facts** are non-empty cells
  - Cells store **measures**
- Cube for a combination of levels of the dimension

6. Remarks

- In principle there does not have to exist an order among the elements at one level of the hierarchy
  - E.g., cities
- Hierarchies do not have to be linear

6. Dimension Schema

- **Schema of a Dimension**
  - A set $D$ of category attributes $D_1, \ldots, D_n$ on $D$
    - These correspond to the levels
  - A partial order $\rightarrow$ over $D$ which represents parent-child relationships in the hierarchy
    - These correspond to upward edges in the hierarchy
    - $\text{Top}_D$ is larger than anything else
    - For every $D_i$, $D_i \rightarrow \text{Top}_D$
    - There exists $\text{D}_{\text{min}}$ which is smaller than anything else
    - For every $D_i$, $\text{D}_{\text{min}} \rightarrow D_i$

6. Facts

- **Targets of analytics**
  - E.g., revenue, #sales, #stock
- A fact is uniquely defined by the combination of values from the dimensions
  - E.g., for dimensions time and location
    - **Revenue in Illinois during Jan 2015**
- **Granularity:** Levels in the dimension hierarchy corresponding to the fact
  - E.g., state, month
Facts (Event vs. Snapshot)

- **Event Facts**
  - Model real-world events
  - E.g., Sale of an item

- **Snapshot Facts**
  - Temporal state
  - A single object (e.g., a book) may contribute to several facts
  - E.g., number of items in stock

Measures

- A measure describes a fact
  - May be derived from other measures

- **Two components**
  - **Numerical value**
  - **Formula** (optional): how to derive it
    - E.g., \( \text{avg(revenue)} = \frac{\text{sum(revenue)}}{\text{count(revenue)}} \)

- We may associate multiple measures to each cell
  - E.g., number of sales and total revenue

Measures - Granularity

- Similar to facts, measures also have a granularity

- How to change granularity of a measure?

  - **Additive measures**
    - Can be aggregated along any dimension
  
  - **Semi-additive/non-additive**
    - Cannot be aggregated along some/all dimensions
      - E.g., snapshot facts along time dimension
        - Number of items in stock at Jan + Feb + … / items in stock during year
        - Median of a measure

Design Process (after Kimball)

- **Comparison to classical relational modeling**
  - **Analysis driven**
    - No need to model all existing data and relationships relevant to a domain
    - Limit modeling to information that is relevant for predicted analytics

  - **Redundancy**
    - Tolerate redundancy for performance if reasonable
      - E.g., in dimension tables to reduce number of joins

Design Process – Steps

1) **Select relevant business processes**
   - E.g., order shipping, sales, support, stock management

2) **Select granularity**
   - E.g., track stock at level of branches or regions

3) **Design dimensions**
   - E.g., time, location, product, ...

4) **Select measures**
   - E.g., revenue, cost, #sales, items in stock, #support requests

Design Process Example

- **Coffee shop chain**
  - **Processes**
    - Sell coffee to customers
    - Buy ingredients from suppliers
    - Ship supplies to branches
    - Pay employees
    - HR (hire, advertise positions, …)

  - Which process is relevant to be analysed to increase profits?
Design Process Example

1) Selecting process(es)
   – sell coffee to customers

2) Select granularity
   – Sale of type of coffee per branch per day
   – Location (country, state, city, zip, shop)
   – Time (year, month, day)
   – Product (type, brand, product)

3) Determine relevant dimensions
   – Location (country, state, city, zip, shop)
   – Time (year, month, day)
   – Product (type, brand, product)

4) Select measures
   – cost, revenue, profit?
**Star Schema**

- A data cube is represented as a set of dimension tables and a fact table.
- **Dimension tables**
  - For each dimension schema $D = (D_1, \ldots, D_k, \text{Top}_D)$ we create a relation $D$.
  - Here $PK$ is a primary key, e.g., $D_{\text{min}}$.
- **Fact table**
  - $F(\text{FK}_1, \ldots, \text{FK}_n, M_1, \ldots, M_m)$
  - Each $\text{FK}_i$ is a foreign key to $D_i$.
  - Primary key is the combination of all $\text{FK}_i$.

**Star Schema - Remarks**

- Dimension tables have redundancy
  - Values for higher levels are repeated
- Fact table is in 3NF
- $\text{Top}_D$ does not have to be stored explicitly
- Primary keys for dimension tables are typically generated (surrogate keys)
  - Better query performance by using integers

**Snowflake Schema**

- A data cube is represented as a set of dimension tables and a fact table.
- **Dimension tables**
  - For each dimension schema $D = (D_1, \ldots, D_k, \text{Top}_D)$ we create a relation multiple relations connected through FKs.
  - $D_i$ is a descriptive attribute.
  - $\text{FK}_i$ is foreign key to the immediate parent(s) of $D_i$.
- **Fact table**
  - $F(\text{FK}_1, \ldots, \text{FK}_n, M_1, \ldots, M_m)$
  - Each $\text{FK}_i$ is a foreign key to $D_i$.
  - Primary key is the combination of all $\text{FK}_i$.

**Snowflake Schema - Remarks**

- Avoids redundancy
- Results in much more joins during query processing
- Possible to find a compromise between snowflake and star schema
  - E.g., use snowflake for very fine-granular dimensions with many levels

**6. Extract-Transform-Load (ETL)**

- The preprocessing and loading phase is called extract-transform-load (ETL) in datawarehousing.
- Many commercial and open-source tools available
- ETL process is modeled as a workflow of operators
  - Tools typically have a broad set of build-in operators: e.g., key generation, replacing missing values, relational operators,
  - Also support user-defined operators

**Snowflake Schema - Example**

- Coffee chain example

**6. Extract-Transform-Load (ETL)**

- The preprocessing and loading phase is called extract-transform-load (ETL) in datawarehousing.
- Many commercial and open-source tools available
- ETL process is modeled as a workflow of operators
  - Tools typically have a broad set of build-in operators: e.g., key generation, replacing missing values, relational operators,
  - Also support user-defined operators
6. Extract-Transform-Load (ETL)

- Some ETL tools
  - Pentaho Data Integration
  - Oracle Warehouse Builder (OWB)
  - IBM Infosphere Information Server
  - Talend Studio for Data Integration
  - CloverETL
  - Cognos Data Manager
  - Pervasive Data Integrator
  - …

6. Operators supported by ETL
- Many of the preprocessing and cleaning operators we already know
  - Surrogate key generation (like creating existentials with skolems)
  - Fixing missing values
    - With default value, using trained model (machine learning)
  - Relational queries
    - E.g., union of two tables or joining two tables
  - Extraction of structured data from semi-structured data and/or unstructured data
  - Entity resolution, data fusion

6. ETL Process
- Operators can be composed to form complex workflows

6. Typical ETL operators
- Elementizing
  - Split values into more fine-granular elements
- Standardization
- Verification
- Matching with master data
- Key generation
- Schema matching, Entity resolution/ Deduplication, Fusion

6. Typical ETL operators
- Elementizing
  - Split non 1NF data into individual elements
- Examples
  - name: “Peter Gertsen” -> firstname: “Peter”, lastname: “Gertsen”
6. Typical ETL operators

- **Standardization**
  - Expand abbreviation
  - Resolve synonyms
  - Unified representation of, e.g., dates

- **Examples**
  - “IL” -> “Illinois”
  - “m/w”, “M/F” -> “male/female”
  - “Jan”, “01”, “January”, “january” -> “January”
  - “St” -> “Street”, “Dr” -> “Drive”, …

- **Verification**
  - Same purpose as constraint based data cleaning but typically does not rely on constraints, but, e.g., regular expression matching

- **Examples**
  - Phone matches “[0-9][3][0-9][3]-[0-9][4]”
  - For all t in Tokens(product description), t exists in English language dictionary

6. Matching master data (lookup)

- Check and potentially repair data based on available master data

- **Examples**
  - E.g., using a clean lookup table with (city, zip) replace the city in each tuple if the pair (city, zip) does not occur in the lookup table

6. Metadata management

- As part of analysis in DW data is subjected to a complex pipeline of operations
  - Sources
  - ETL
  - Analysis queries

- Important, but hard, to keep track of what operations have been applied to data and from which sources it has been derived

- Need metadata management
  - Including provenance (later in this course)

6. Querying DW

- Targeted model (cube vs. relational)
  - Design specific language for datacubes
  - Add suitable extensions to SQL

- Support typical analytical query patterns
  - Multiple parallel grouping criteria
    - Show total sales, subtotal per state, and subtotal per city
    - E.g., the result for Feb would be the sum of the sales for Jan + Feb
  - Windowed aggregates and ranking
  - Show 10 most successful stores

- **Targeted model (cube vs. relational)**
  - **Design specific language for datacubes**
    - MDX
  - **Add suitable extensions to SQL**
    - GROUPING SETS, CUBE, …
    - Windowed aggregation using OVER(), PARTITION BY, ORDER BY, window specification
    - Window functions: RANK, DENSE_RANK()
6. Cube operations

- **Roll-up**
  - Move from fine-granular to more coarse-granular in one or more dimensions of a datacube
  - E.g., sales per (city, month, product category) to Sales per (state, year, product category)

- **Drill-down**
  - Move from coarse-granular to more fine-granular in one of more dimensions
  - E.g., phonecalls per (city, month) to phonecalls per (zip, month)

6. SQL Extensions

- **Syntactic Sugar for multiple grouping**
  - GROUPING SETS
  - CUBE
  - ROLLUP
  - These constructs are allowed as expressions in the GROUP BY clause

6. SQL Extensions

- **Recall that grouping on multiple sets of attributes is hard to express in SQL**
  - E.g., give me the total sales, the sales per year, and the sales per month
  - Practice

6. GROUPING SETS

- **GROUP BY GROUPING SETS ((set_1), ..., (set_n))**
- Explicitly list sets of group by attributes
- Semantics:
  - Equivalent to UNION over duplicates of the query each with a group by clause GROUP BY set
  - Schema contains all attributes listed in any set
  - For a particular set, the attribute not in this set are filled with NULL values
6. GROUPING SETS

SELECT quarter, city, product_typ, SUM(profit) AS profit
FROM facttable F, time T, location L, product P
WHERE F.TID = T.TID AND F.LID = L.LID AND F.PID = P.PID
GROUP BY GROUPING SETS
( (quarter, city), (quarter, product_typ))

<table>
<thead>
<tr>
<th>quarter</th>
<th>city</th>
<th>product_typ</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 Q1</td>
<td>Books</td>
<td>Books</td>
<td>8347</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Books</td>
<td>Books</td>
<td>7836</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Seattle</td>
<td>Gardening</td>
<td>12344</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

• How to distinguish between NULLs based on grouping sets and NULL values in a group by column?

GROUP BY GROUPING SETS
( (quarter, city), (quarter, product_typ), (quarter, product_typ, city))

<table>
<thead>
<tr>
<th>quarter</th>
<th>city</th>
<th>product_typ</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 Q1</td>
<td>Books</td>
<td>Books</td>
<td>8347</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Books</td>
<td>Books</td>
<td>7836</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Chicago</td>
<td>Gardening</td>
<td>12344</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Seattle</td>
<td>Gardening</td>
<td>12345</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

• Solution:
  - GROUPING predicate
  - GROUPING(A) = 1 if grouped on attribute A, 0 else

SELECT - GROUPING(product_typ) AS grp_prd
GROUP BY GROUPING SETS
( (quarter, city), (quarter, product_typ), (quarter, product_typ, city))

<table>
<thead>
<tr>
<th>quarter</th>
<th>city</th>
<th>product_typ</th>
<th>profit</th>
<th>grp_prd</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010 Q1</td>
<td>Books</td>
<td>Books</td>
<td>8347</td>
<td>1</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Books</td>
<td>Books</td>
<td>7836</td>
<td>0</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Gardening</td>
<td>Gardening</td>
<td>12344</td>
<td>1</td>
</tr>
<tr>
<td>2012 Q2</td>
<td>Seattle</td>
<td>Gardening</td>
<td>12345</td>
<td>1</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

• Combining GROUPING SETS

GROUP BY A, B
= GROUP BY GROUPING SETS ((A,B))

GROUP BY GROUPING SETS ((A,B), (A,C), (A))
= GROUP BY A, GROUPING SETS ((B), (C), ())

GROUP BY GROUPING SETS ((A,B), (B,C),
GROUPING SETS ((D,E), (D))
= GROUP BY GROUPING SETS
( (A,B,D,E), (A,B,D), (B,C,D,E), (B,C,D) )
6. CUBE

- GROUP BY ROLLUP(A_1, ..., A_n)
- Group by all prefixes
- Typically different granularity levels from single dimension hierarchy, e.g., year-month-day
  - Database can often find better evaluation strategy

GROUP BY ROLLUP (A, B, C)
= GROUP BY GROUPING SETS {
  (A, B, C),
  (A, B),
  (A),
  ()
}

6. OVER clause

- Agg OVER (partition-clause, order-by, window-specification)
- New type of aggregation and grouping where
  - Each input tuple is paired with the aggregation result for the group it belongs too
  - More flexible grouping based on order and windowing
  - New aggregation functions for ranking queries
    - e.g., RANK(), DENSE_RANK()

6. GROUPING SETS

SELECT year, month, city, profit
SUM(profit) OVER () AS ttl
FROM sales

- For each tuple build a set of tuples belonging to the same window
  - Compute aggregation function over window
  - Return each input tuple paired with the aggregation result for its window
- OVER() = one window containing all tuples

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. OVER clause

- Agg OVER (partition-clause, order-by, window-specification)
- New type of aggregation and grouping where
  - Each input tuple is paired with the aggregation result for the group it belongs too
  - More flexible grouping based on order and windowing
  - New aggregation functions for ranking queries
    - e.g., RANK(), DENSE_RANK()

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (PARTITION BY year) AS ttl
FROM sales

- PARITION BY
  - only tuples with same partition-by attributes belong to the same window
- Like GROUP BY

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>
6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (ORDER BY year, month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>12</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>34</td>
</tr>
<tr>
<td>2011</td>
<td>2</td>
<td>New York</td>
<td>17</td>
</tr>
<tr>
<td>2011</td>
<td>3</td>
<td>New York</td>
<td>17</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (PARTITION BY year ORDER BY month) AS ttl
FROM sales

• Combining PARTITION BY and ORDER BY
  – First partition, then order tuples within each partition

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>New York</td>
<td>12</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (PARTITION BY year ORDER BY month) AS ttl
FROM sales

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>12</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>34</td>
</tr>
<tr>
<td>2011</td>
<td>2</td>
<td>New York</td>
<td>17</td>
</tr>
<tr>
<td>2011</td>
<td>3</td>
<td>New York</td>
<td>17</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (ORDER BY year, month) AS ttl
FROM sales

• ORDER BY
  – Order tuples on these expressions
  – Only tuples which are <= to the order as the current tuple belong to the same window

• E.g., can be used to compute an accumulate total

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>12</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>34</td>
</tr>
<tr>
<td>2011</td>
<td>2</td>
<td>New York</td>
<td>17</td>
</tr>
<tr>
<td>2011</td>
<td>3</td>
<td>New York</td>
<td>17</td>
</tr>
</tbody>
</table>

6. GROUPING SETS

SELECT year, month, city
SUM(profit) OVER (ORDER BY year, month)
RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING AS ttl
FROM sales

• Explicit window specification
  – Requires ORDER BY
  – Determines which tuples “surrounding” the tuple according to the sort order to include in the window

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>12</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>15</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>34</td>
</tr>
<tr>
<td>2011</td>
<td>2</td>
<td>New York</td>
<td>17</td>
</tr>
<tr>
<td>2011</td>
<td>3</td>
<td>New York</td>
<td>17</td>
</tr>
</tbody>
</table>
6. GROUPING SETS

```
SELECT year, month, city
    SUM(profit) OVER (ORDER BY year, month
    RANGE BETWEEN 1 PRECEDING
    AND 1 FOLLOWING) AS ttl
FROM sales
```

- Explicit window specification
  - Requires ORDER BY
  - Determines which tuples “surrounding” the tuple according to the sort order to
    include in the window

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>city</th>
<th>profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>1</td>
<td>Chicago</td>
<td>10</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td>Chicago</td>
<td>5</td>
</tr>
<tr>
<td>2010</td>
<td>3</td>
<td>Chicago</td>
<td>20</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>Chicago</td>
<td>45</td>
</tr>
<tr>
<td>2010</td>
<td>1</td>
<td>New York</td>
<td>12</td>
</tr>
</tbody>
</table>

6. MDX

- **Multidimensional expressions (MDX)**
  - Introduced by Microsoft
  - Query language for the cube data model
  - SQL-like syntax
    - Keywords have different meaning
    - MDX queries return a multi-dimensional report
      - 2D = spreadsheet
      - 3D or higher, e.g., multiple spreadsheets

6. MDX - SELECT

```
SELECT ( Chicago, Schaumburg ) ON ROWS
    ( [2010], [2011].CHILDREN ) ON COLUMNS
FROM PhoneCallsCube
WHERE ( Measures.numCalls, Carrier.Spring )
```

- Select specifies dimensions in result and how to visualize
  - ON COLUMNS, ON ROWS, ON PAGES, ON SECTIONS, OR
    CHAPTERS
- Every dimension in result corresponds to one dimension in the cube
  - Set of concepts from this dimensions which may be from different levels of
    granularity
6. MXD - SELECT

• Specify concepts from dimensions
  – List all values as set, e.g., { [2010], [2011] }
  – Not necessarily from same level of hierarchy (e.g., mix years and months)

• Language constructs for accessing parents and children or members of a level in the hierarchy
  – CHILDREN: all direct children
    • E.g.: [2010].CHILDREN = {{2010 Jan}, …, [2010 Dec]}
  – PARENT: the direct parent
    • E.g.: [2010 Jan].PARENT = [2010]
  – MEMBERS: all direct children
    • E.g.: Time.Years.MEMBERS = \{[1990], [1991], …, [2016]\}
  – LASTCHILD: last child (according to order of children)
    • E.g.: [2010].LASTCHILD = [2010 Dec]
  – NEXTMEMBER: right sibling on same level
    • E.g.: [2010].NEXTMEMBER = [2011]
  – [a]:[b]: all members in interval between a and b
    • E.g.: [1990]:[1993] = \{[1990], [1991], [1992], [1993]\}

78

6. MXD - SELECT

• Nesting of sets: CROSSJOIN
  – Project two dimensions into one
  – Forming all possible combinations

SELECT CROSSJOIN ( { Chicago, Schaumburg }, [2010], [2011] ) ON ROWS
FROM PhoneCallsCube
WHERE ( Measures.numCalls )

80

6. Query Processing in DW

• Large topic, here we focus on two aspects
  – Partitioning
  – Query answering with materialized views

82

6. MXD - SELECT

• Conditional selection of members: FILTER
  – One use members that fulfill condition
  – E.g., condition over aggregation result

• Show results for all month of 2010 where there are more Sprint calls than ATT calls

SELECT FILTER([2010].CHILDREN, (Sprint, numCalls) > (ATT, numCalls) ) ON ROWS
FROM PhoneCallsCube
WHERE ( Measures.numCalls )

81

6. Partitioning

• Partitioning splits a table into multiple fragments that are stored independently
  – E.g., split across X disks, across Y servers

• Vertical partitioning
  – Split columns across fragments
    • E.g., R = \{A,B,C,D\}, fragment F1 = \{A,B\}, F2 = \{C,D\}
    • Either add a row id to each fragment or the primary key to be able to reconstruct

• Horizontal partitioning
  – Split rows
  – Hash vs. range partitioning

83
6. Partitioning

• **Why partitioning?**
  • Parallel/distributed query processing
  • Distribute storage load across disks/servers
  • Avoid reading data that is not needed to answer a query
    - Vertical
      - Only read columns that are accessed by query
    - Horizontal
      - Only read tuples that may match queries selection conditions

---

6. Partitioning

• **Vertical Partitioning**
  – Fragments \( F_1 \) to \( F_n \) of relation \( R \) such that
    - \( \text{Sch}(F_1) \cup \text{Sch}(F_2) \cup \ldots \cup \text{Sch}(F_n) = \text{Sch}(R) \)
    - Store row id or PK of \( R \) with every fragment
    - Restore relation \( R \) through natural joins

---

6. Partitioning

• **Horizontal Partitioning**
  – **Range** partitioning on attribute \( A \)
    - Split domain of \( A \) into intervals representing fragments
    - E.g., tuples with \( A = 15 \) belong to fragment \([0,20]\)
  – Fragments \( F_1 \) to \( F_n \) of relation \( R \) such that
    - \( \text{Sch}(F_1) \cup \text{Sch}(F_2) \cup \ldots \cup \text{Sch}(F_n) = \text{Sch}(R) \)
    - \( R = F_1 \cup \ldots \cup F_n \)

---

6. Partitioning

• **Horizontal Partitioning**
  – **Hash** partitioning on attribute \( A \)
    - Split domain of \( A \) into \( x \) buckets using hash function
    - E.g., tuples with \( h(A) = 3 \) belong to fragment \( F_3 \)
    - \( \text{Sch}(F_1) \cup \text{Sch}(F_2) \cup \ldots \cup \text{Sch}(F_n) = \text{Sch}(R) \)
    - \( R = F_1 \cup \ldots \cup F_n \)

---

6. Partitioning

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>12,000</td>
<td>45</td>
<td>M</td>
</tr>
<tr>
<td>Alice</td>
<td>24,000</td>
<td>34</td>
<td>F</td>
</tr>
<tr>
<td>Bob</td>
<td>20,000</td>
<td>22</td>
<td>M</td>
</tr>
<tr>
<td>Gertrud</td>
<td>50,000</td>
<td>55</td>
<td>F</td>
</tr>
<tr>
<td>Pferdegert</td>
<td>14,000</td>
<td>23</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Age</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>12,000</td>
<td>45</td>
<td>M</td>
</tr>
<tr>
<td>Alice</td>
<td>24,000</td>
<td>34</td>
<td>F</td>
</tr>
<tr>
<td>Bob</td>
<td>20,000</td>
<td>22</td>
<td>M</td>
</tr>
<tr>
<td>Gertrud</td>
<td>50,000</td>
<td>55</td>
<td>F</td>
</tr>
<tr>
<td>Pferdegert</td>
<td>14,000</td>
<td>23</td>
<td>M</td>
</tr>
</tbody>
</table>

---

Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) **Big Data Analytics**
8) Data Provenance