Outline

ILLINOIS INSTITUTE OF TECHNOLOGY

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

6. What is Datawarehousing?

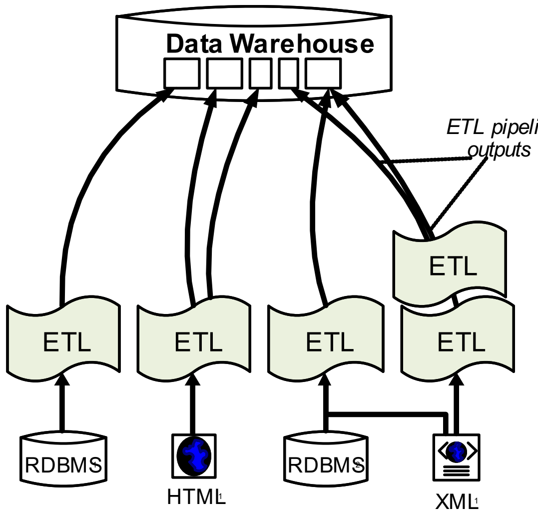
- **Problem: Data Analysis, Prediction, Mining**
  - **Example: Walmart**
  - Transactional databases
    - Run many "cheap" updates concurrently
    - E.g., each store has a database storing its stock and sales
  - Complex Analysis over Transactional Databases?
    - Want to analyze across several transactional 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

6. What is Datawarehousing?

- **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

ILLINOIS INSTITUTE OF TECHNOLOGY

- **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

- 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
  - …

6. Multidimensional Datamodel

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

		2014												2015				
		1. Quarter			2. Quarter			3. Quarter			4. Quarter			1. Quarter			2. Qu…	
		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May
Toy	car	3	7	6	37	7	92	37	7	92	37	7	92	37	7	92	2	…
	puppet	9	4	5	31	1	1	1	1	1	1	1	1	1	2	2	2	…
	Fishing rod	11	12	22	22	22	22	22	22	7	6	6	6	6	65	4	33	…
Books	Moby Dick	3	40	39	37	7	92	81	6	51	7	48	51	5	7	3	3	…
	Mobile devel.	3	2	5	43	7	0	81	6	51	7	48	51	5	7	3	3	…
	King Lear	3	9	6	37	7	92	5	6	51	7	48	51	5	7	3	3	…

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. Data cubes

ILLINOIS INSTITUTE OF TECHNOLOGY

- Given **n dimensions**
  - E.g., product type, location, time
- Given **m measures**
  - E.g., number of sales, items in stock, …
- A **datacube** (datahypercube) is an **n-**dimensional datastructure that maps values in the dimensions to values for the m measures
  - **Schema**: $D_1, \dots, D_n, M_1, \dots, M_m$
  - **Instance**: a function

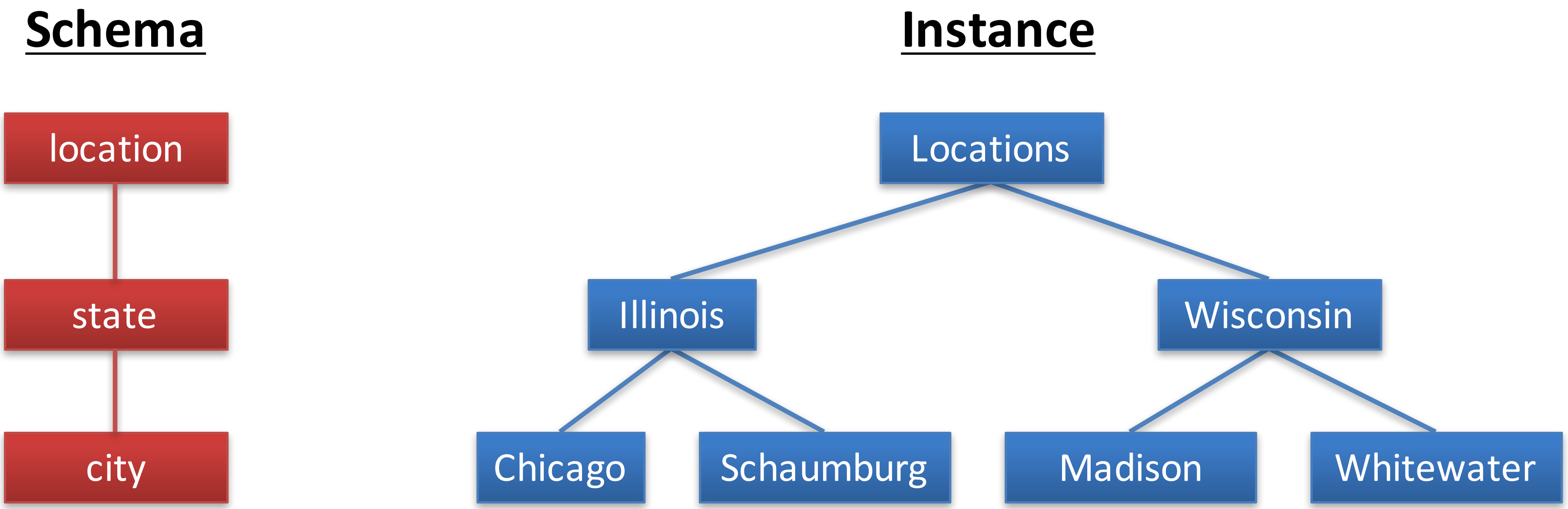
$\mathtt{dom}(D_1) \times \dots \times \mathtt{dom}(D_n) \rightarrow \mathtt{dom}(M_1) \times \dots \times \mathtt{dom}(M_m)$

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)

ILLINOIS INSTITUTE OF TECHNOLOGY

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



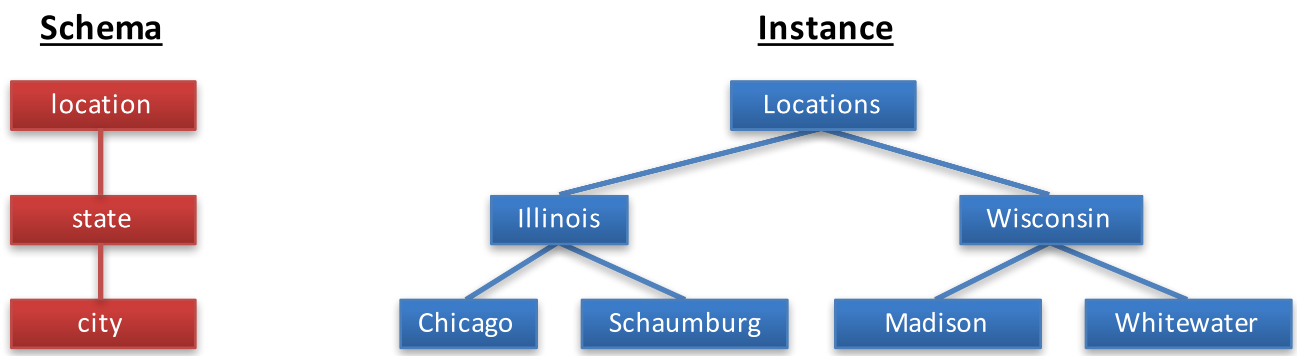
6. Dimension Schema

- **Schema of a Dimension**
  - A set **D** of category attributes $\mathbf{D_1}$, …, $\mathbf{D_n}$, $\mathbf{Top_D}$
    - 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
    - $\mathbf{Top_D}$ is larger than anything else
      - For every D_i: $D_i \rightarrow Top_D$
    - There exists $\mathbf{D_{min}}$ which is smaller than anything else
      - For every D_i: $D_{min} \rightarrow D_i$

6. Dimension Schema Example

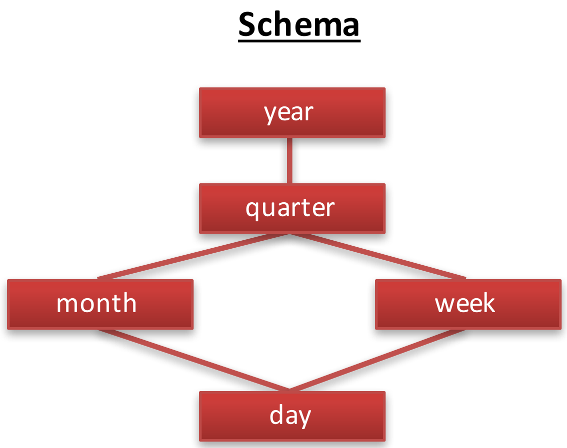
- **Schema of Location Dimension**
  - Set of categories D = {location, state, city}
  - Partial order

  $\{ \text{city} \rightarrow \text{state}, \text{city} \rightarrow \text{location}, \text{state} \rightarrow \text{location} \}$
  - Top_D = location
  - D_{min} = city

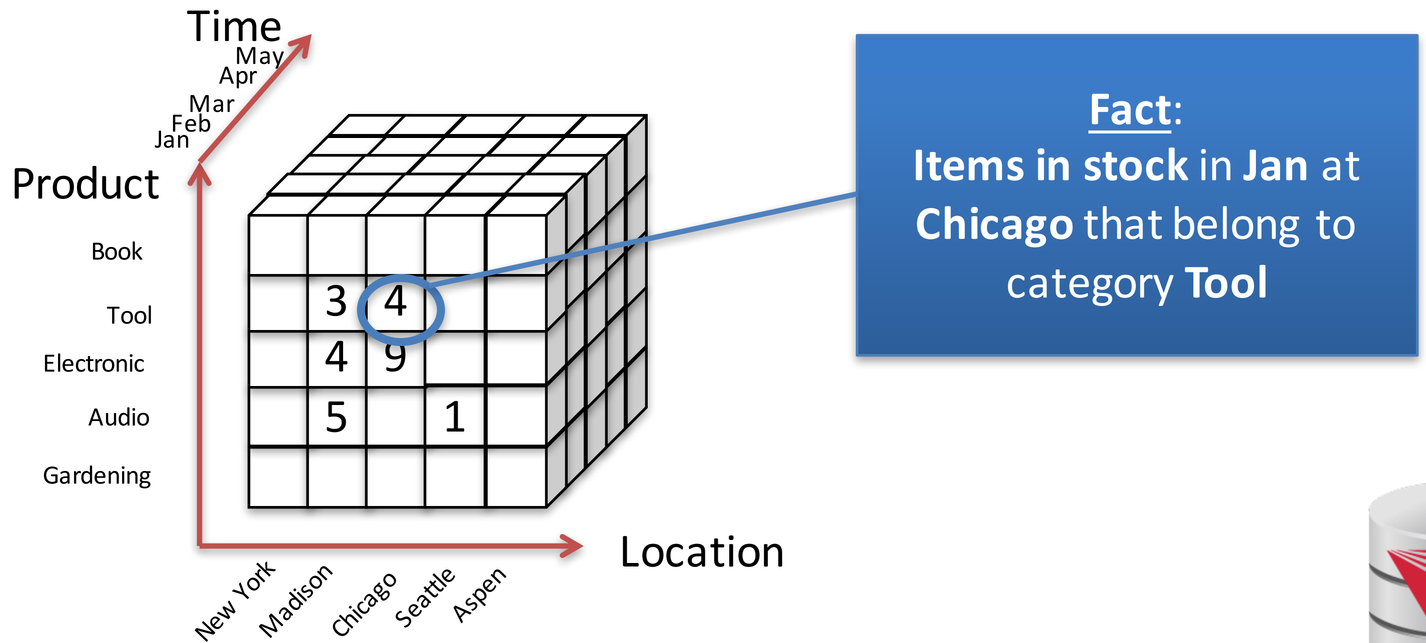


6. Remarks

- 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



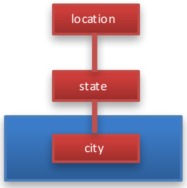
- Each **cell** in the cube corresponds to a combination of elements from each dimension
  - **Facts** are non-empty cells
  - Cells store **measures**
- Cube for a combination of levels of the dimension

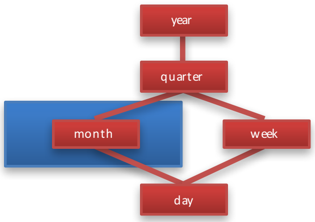


Facts

- 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 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., avg(revenue) = sum(revenue) / 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?
- Need algorithm to combine measures
  - **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)

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

- **1) Select relevant business processes**
  - E.g., order shipping, sales, support, stock management
- **2) Select granuarity**
  - 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**
  - Single sale?
  - Sale per branch/day?
  - Sale per city/year?

Design Process Example

ILLINOIS INSTITUTE OF TECHNOLOGY

- **1) Selecting process(es)**
  – sell coffee to customers
- **2) Select granularity**
  – Sale of type of coffee per branch per day
  – Sufficient for analysis
    - Save storage
- **3) Determine relevant dimensions**
  – Location
  – Time
  – Product, …

Design Process Example

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)

Design Process Example

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)
- **4) Select measures**

Design Process Example

ILLINOIS INSTITUTE OF TECHNOLOGY

- **1) Selecting process(es)**
  - sell coffee to customers
- **2) Select granularity**
  - Sale of type of coffee per branch per day
- **3) Determine relevant dimensions**
  - Location (country, state, city, zip, shop)
  - Time (year, month, day)
  - Product (type, brand, product)
- **4) Select measures**
  - cost, revenue, profit?

Relational representation

ILLINOIS INSTITUTE OF TECHNOLOGY

- How to model a **datacube** using the **relational datamodel**
- We start from
  - Dimension schemas
  - Set of measures

Star Schema

ILLINOIS INSTITUTE OF TECHNOLOGY

- A data cube is represented as a set of dimension tables and a fact table
- Dimension tables
  - For each dimension schema $D = (D_1,\ldots,D_k,Top_D)$ we create a relation
  - **D (<u>PK</u>, $D_1,\ldots,D_k$)**
  - Here PK is a primary key, e.g., D_{min}
- Fact table
  - **F(<u>FK_1</u>, …, <u>FK_n</u>, M_1, ..., M_m)**
  - Each **FK_i** is a foreign key to **D_i**
  - Primary key is the combination of all Fk_i

Star Schema - Remarks

ILLINOIS INSTITUTE OF TECHNOLOGY

- Dimension tables have redundancy
  - Values for higher levels are repeated
- Fact table is in 3NF
- 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 $D = (D_1,\ldots,D_k,Top_D)$ we create a relation multiple relations connected through FKs
  - D_i (<u>PK</u>, A_1, …, A_l, FK_j)
  - A_l is a descriptive attribute
  - FKj is foreign key to the immediate parent(s) of D_i
- Fact table
  - F(<u>FK_1</u>, …, <u>FK_n</u>, M_1, ..., M_m)
  - Each $\mathbf{FK_i}$ is a foreign key to $\mathbf{D_i}$
  - Primary key is the combination of all 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

Snowflake Schema - Example

– Coffee chain example

6. Extract-Transform-Load (ETL)

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

- **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
  - …

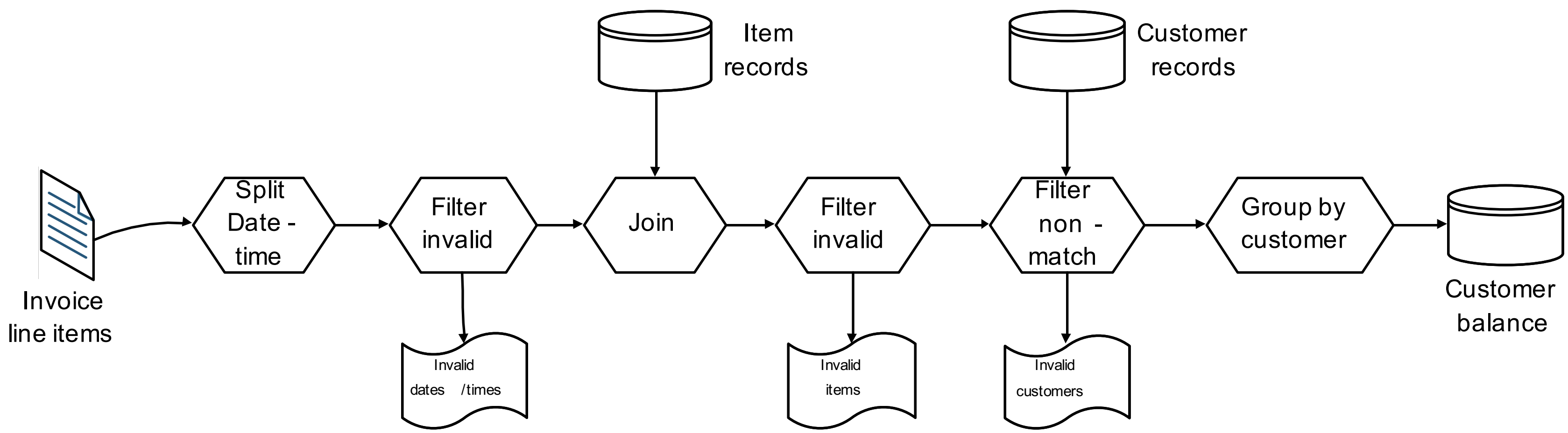
ILLINOIS INSTITUTE OF TECHNOLOGY

- **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

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Control flow operators**
  - AND/OR
  - Fork
  - Loops
  - Termination
    - Successful
    - With warning/errors

6. Typical ETL operators

- **Elementizing**
  - Split non 1NF data into individual elements
- **Examples**
  - **name: “Peter Gertsen” -> firstname: “Peter”, lastname: “Gertsen”**
  - **date: “12.12.2015” -> year: 2002, month: 12, day :12**
  - **Address: “10 W 31st, Chicago, IL 60616” -> street = “10 W 31st”, city = “Chicago”, state = “IL”, zip = “60616”**

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", …**

6. Typical ETL operators

- **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. Typical ETL operators

ILLINOIS INSTITUTE OF TECHNOLOGY

- **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
    - -> three subqueries with different group-by in SQL
  - Windowed aggregates and ranking
    - Show 10 most successful stores
    - Show cummulative sales for months of 2016
      - E.g., the result for Feb would be the sum of the sales for Jan + Feb

6. Querying DW

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

- **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. Cube operations

ILLINOIS INSTITUTE OF TECHNOLOGY

- **Drill-out**
  - Add additional dimensions
    - special case of drill-down starting from Top_D in dimension(s)
    - **E.g., sales per (city, product category) to Sales per (city,year, product category)**
- **Drill-in**
  - Remove dimension
    - special case for roll-up move to TopD for dimension(s)
    - **E.g., phonecalls per (city,month) to phonecalls per (month)**

6. Cube operations

- **Slice**
  - Select data based on restriction of the values of one dimension
    - **E.g., sales per (city,month) -> sales per (city) in Jan**
- **Dice**
  - Select data based on restrictions of the values of multiple dimensions
    - **E.g., sales per (city,month) -> sales in Jan for Chicago and Washington DC**

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. SQL Extensions

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

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_i
  - 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))
```

quarter	city	product_typ	profit
2010 Q1		Books	8347
2012 Q2		Books	7836
2012 Q2		Gardening	12300
2012 Q2	Chicago		12344
2012 Q2	Seattle		124345

6. GROUPING SETS

ILLINOIS INSTITUTE OF TECHNOLOGY

```
SELECT quarter, city, NULL AS 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 quarter, city
UNION
SELECT quarter, NULL AS 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 quarter, product_type
```

6. GROUPING SETS

ILLINOIS INSTITUTE OF TECHNOLOGY

- Problem:
  - 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)
```

quarter	city	product_typ	profit
2010 Q1			8347
2012 Q2			7836
2012 Q2			12300
2012 Q2	Chicago		12344
2012 Q2	Seattle		124345
2012 Q2	Seattle	Gardening	12343

Did not group on **product_typ** or this is the group for all NULL values in **product_typ**?

6. GROUPING SETS

- Solution:
  - GROUPING predicate
  - GOUPING(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)
```

quarter	city	product_typ	profit	grp_prd
2010 Q1		Books	8347	1
2012 Q2		Books	7836	1
2[illegible]	[illegible]	Gardening	12300	1
2[illegible]	[illegible]		12344	0
2012 Q2	Seattle		124345	1
2012 Q2	Seattle	Gardening	12343	1

Now it's clear!

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` **`CUBE (set)`**
- Group by all 2^n subsets of **`set`**

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

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. OVER clause

- `Agg OVER (partition-clause, order-by,window-specification)`
- New type of aggregation and grouping where

```
SELECT shop, sum(profit) OVER()
```

- aggregation over full table

```
SELECT shop, sum(profit) OVER(PARTITION BY state)
```

- like group-by

```
SELECT shop, sum(profit) OVER(ORDER BY month)
```

- rolling sum including everything with smaller month

```
SELECT shop, sum(profit) OVER(ORDER BY month 6
PRECEDING 3 FOLLOWING)
```

6. OVER clause

- `Agg OVER (partition-clause order-by,window-specification)`
- New type of aggregation and grouping where

```
<window frame preceding> ::= {
     UNBOUNDED PREDECEDING
         | n PRECEDING
         | CURRENT ROW }

<window frame following> ::= {
         UNBOUNDED FOLLOWING
           | n FOLLOWING
       | CURRENT ROW
}
```

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	92
2010	2	Chicago	5	92
2010	3	Chicago	20	92
2011	1	Chicago	45	92
2010	1	New York	12	92

6. OVER clause

ILLINOIS INSTITUTE OF TECHNOLOGY

```
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**

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	47
2010	2	Chicago	5	47
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	47

6. OVER clause

ILLINOIS INSTITUTE OF TECHNOLOGY

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	47
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	47

6. OVER clause

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause


```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause

ILLINOIS INSTITUTE OF TECHNOLOGY

```
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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	92
2010	1	New York	12	22

6. OVER clause

ILLINOIS INSTITUTE OF TECHNOLOGY

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

- Combining **PARTITION BY** and **ORDER BY**
  - First partition, then order tuples within each partition

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	27
2010	3	Chicago	20	47
2011	1	Chicago	45	45
2010	1	New York	12	22

6. OVER clause

```
SELECT year, month, city
       SUM(profit) OVER (PARTITION BY year ORDER BY 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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	27
2010	2	Chicago	5	47
2010	3	Chicago	20	25
2011	1	Chicago	45	45
2010	1	New York	12	27

6. OVER clause

```
SELECT year, month, city
      SUM(profit) OVER (ORDER BY year, month
                         ROWS 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

year	month	city	profit
2010	1	Chicago	10
2010	2	Chicago	5
2010	3	Chicago	20
2011	1	Chicago	45
2010	1	New York	12

year	month	city	profit	ttl
2010	1	Chicago	10	22
2010	2	Chicago	5	37
2010	3	Chicago	20	70
2011	1	Chicago	45	65
2010	1	New York	12	27

6. MDX

- **M**ulti**d**imensional e**x**pressions (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 Query

- Basic Query Structure

```
SELECT <axis-spec1>, …
FROM <cube-spec1>, …
WHERE ( <select-spec> )
```

- Note!
  - Semantics of SELECT, FROM, WHERE not what you would expect knowing SQL

6. MXD

ILLINOIS INSTITUTE OF TECHNOLOGY

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

- Meaning of
  - [] interpret number as name
  - {} set notation
  - () tuple in where clause

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

6. MXD

ILLINOIS INSTITUTE OF TECHNOLOGY

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

Determine result layout rows and columns of spreadsheet

Specify sets of dimensional concepts

Datacube(s) to use

Select measures to aggregate over

Slice (egg., here only aggregation over Spring calls)

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

6. MXD - SELECT

ILLINOIS INSTITUTE OF TECHNOLOGY

```
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, ON 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
  - **E.g., {2010, 2011 Jan, 2012 Jan, 2012 Feb, 2010 Jan 1st}**

	2010	2011 Jan	2011 Feb	2011 Mar	…	2011 Dec
Chicago	23423	5425234523	432	43243434	…	12231
Schaumburg	32132	12315	213333	123213	….	123153425

6. MXD - SELECT

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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]}`

6. MXD - SELECT

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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]}`

6. MXD - SELECT

- Nesting of sets: **CROSSJOIN**
  - Project two dimensions into one
  - Forming all possible combinations

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

Chicago	2010	123411
	2011	3231
Schaumburg	2010	32321132
	2011	12355

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
       { Chicago } ON COLUMNS
FROM PhoneCallsCube
WHERE ( Measures.numCalls )
```

6. Query Processing in DW

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

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

6. Partitioning

- **Why partitioning?**
  - Parallel/distributed query processing
    - read/write fragments in parallel
    - 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

- **Vertical Partitioning**
  - Fragments F_1 to Fn of relation R such that
    - $Sch(F_1)$ u $Sch(F_2)$ u … u $Sch(F_n)$ = Sch(R)
    - Store row id or PK of R with every fragment
    - Restore relation R through natural joins

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Rowid	Name	Salary
1	Peter	12,000
2	Alice	24,000
3	Bob	20,000
4	Gertrud	50,000
5	Pferdegert	14,000

Rowid	Age	Gender
1	45	M
2	34	F
3	22	M
4	55	F
5	23	M

- **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 Fn of relation R such that
    - $Sch(F_1) = Sch(F_2) = \dots = Sch(F_n) = Sch(R)$
    - $R = F_1 \cup \dots \cup F_n$

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Name	Salary	Age	Gender
Peter	12,000	45	M
Pferdegert	14,000	23	M

Salary
[0,15000]

Name	Salary	Age	Gender
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F

Salary
[15001,10000]

- **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**
    - $Sch(F_1) = Sch(F_2) = … = Sch(F_n) = Sch(R)$
    - $R = F_1 \cup … \cup F_n$

Name	Salary	Age	Gender
Peter	12,000	45	M
Alice	24,000	34	F
Bob	20,000	22	M
Gertrud	50,000	55	F
Pferdegert	14,000	23	M

Name	Salary	Age	Gender
Alice	24,000	34	F
Pferdegert	14,000	23	M

Salary
h(24,000) = 0
H(14,000) = 0

Name	Salary	Age	Gender
Peter	12,000	45	M
Bob	20,000	22	M
Gertrud	50,000	55	F

Salary
h(12,000) = 1
H(20,000) = 1
H(50,000) = 1

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