



8. New Research and Application Fields



Main Contents

- Data warehouse
- OLAP
- Data mining
- Information retrieval
- Semi-structured data and XML



8.1 Data Warehouse & OLAP

- Challenges come from huge amount of data in network era
 - How to use them efficiently?
 - How to find useful information in such a data ocean?
 - If they cannot be used by human being, they are garbage.
- Data is the most important resources.
- The importance of decision-making scientifically
- Data requirements in decision-making



Data Requirements in Decision-making

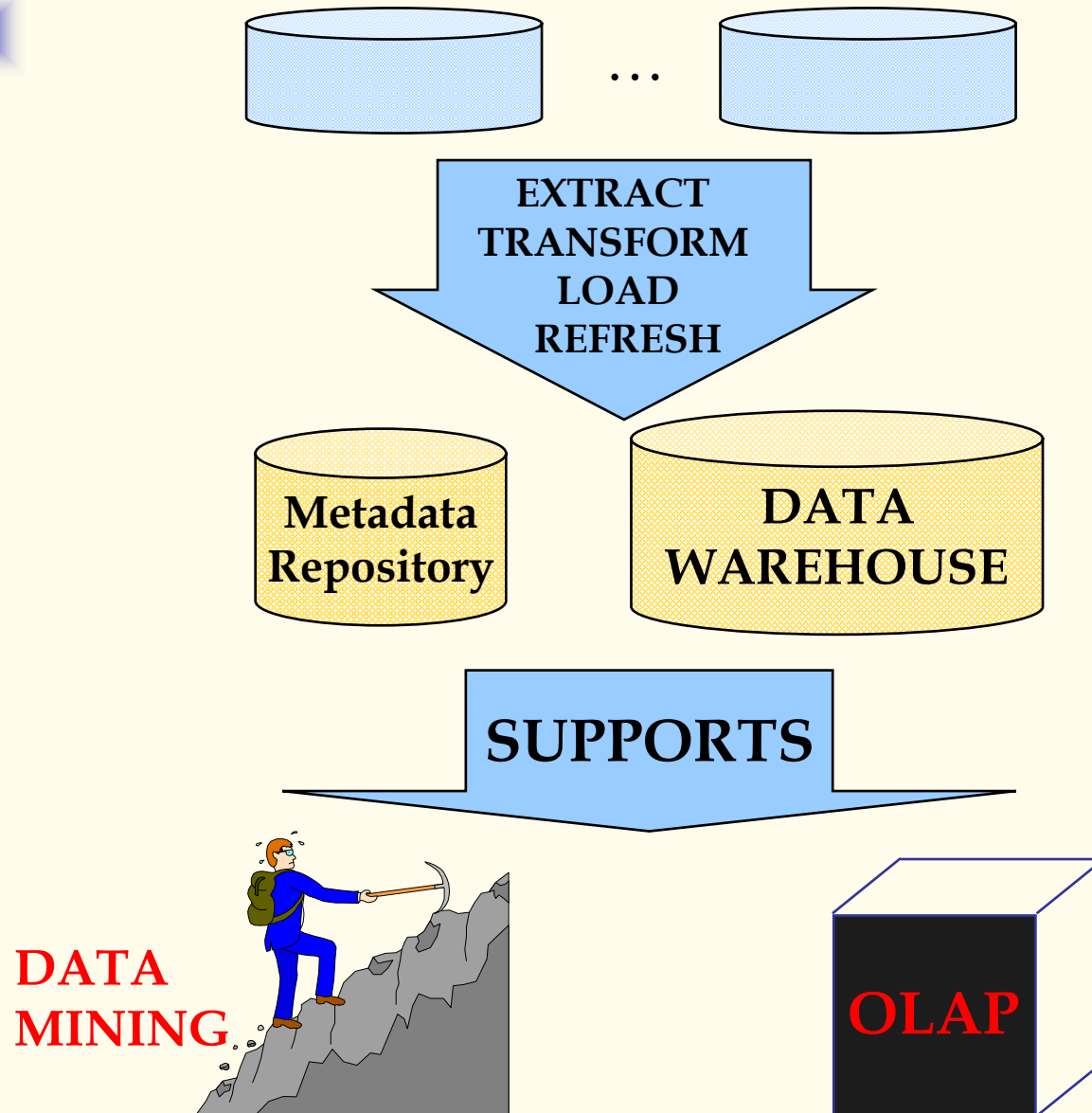
- Need summarized data while not detail data
- Need historical data
- Need large amount of external data (multi data source)
- Need decision subject oriented data, while not the data facing daily transaction process
- The data don't need real time updating. They are mainly read
- OLTP and OLAP



What is Data warehouse

- A data warehouse is a repository of information gathered from multiple sources.
 - Decision subject oriented
 - Provides a single consolidated interface to data
 - Data stored for an extended period, providing access to historical data
 - Mainly retrieved
 - Data/updates are periodically downloaded from online transaction processing (OLTP) systems.
 - Typically, download happens each night.
 - Data may not be completely up-to-date, but is recent enough for analysis.
 - Running large queries at the warehouse ensures that OLTP systems are not affected by the decision-support workload.

INNER / EXTERNAL DATA SOURCES





Database and Data Warehouse

	On-Line Transaction Processing (OLTP)	On-Line Analytical Processing (OLAP)
Data Feature	Detail data	Summarized data
Data prescription	Current data	Current and historical data
Data Source	Inner data of a enterprise	Inner & External data; Distributed; Heterogeneous
Data organization	Surrounding transaction processing	Surrounding decision subject
Data Updating	Updated instantly	Periodical or on demand
Data Amount	Involving less data in one operation	Involving huge amount of data in one operation
Operating Feature	Mainly simple and repeated short transactions	Mainly long transaction processing complex queries



Database and Data Warehouse

	On-Line Transaction Processing (OLTP)	On-Line Analytical Processing (OLAP)
Data Feature	Detail data	Summarized data
Data prescription	Current data	Current and historical data
Data Source	Inner data of a enterprise	Inner & External data; Distributed; Heterogeneous
Data organization	Surrounding transaction processing	Surrounding decision subject
Data Updating	Updated instantly	Periodical or on demand
Data Amount	Involving less data in one operation	Involving huge amount of data in one operation
Operating Feature	Mainly simple and repeated short transactions	Mainly long transaction processing complex queries



Warehousing Issues

- **Semantic Integration:** When getting data from multiple sources, must eliminate mismatches, e.g., different currencies, schemas.
- **Heterogeneous Sources:** Must access data from a variety of source formats and repositories.
 - Replication capabilities can be exploited here.
- **Load, Refresh, Purge:** Must load data, periodically refresh it, and purge too-old data.
- **Metadata Management:** Must keep track of source, loading time, and other information for all data in the warehouse.



Software Solutions that use Warehouse

- **Online Analytical Processing (OLAP)**
 - enables users to analyse data across multiple dimensions and hierarchies
- **Analysis and Query Reporting Solutions**
 - custom built analysis tools use mathematical models to produce specialized interactive solutions
- **Data Mining**
 - enable users to identify patterns and correlations within a set of data, or to create predictive models from the data



An Example

Color	Size	Number
<hr/>		
Light	small	3
Light	Medium	20
Light	Large	10
Dark	Small	4
Light	Small	5
Dark	Small	16
Light	Medium	5
Dark	Large	5
Light	Medium	10
Dark	Medium	5
Dark	Medium	5

	Small	Medium	Large	Total
Light Dark	8	35	10	53
	20	10	5	35
Total	28	45	15	88

Cross-tabulation of *number* by
size and *color* of sample relation
sales with the schema
Sales(color, size, number).



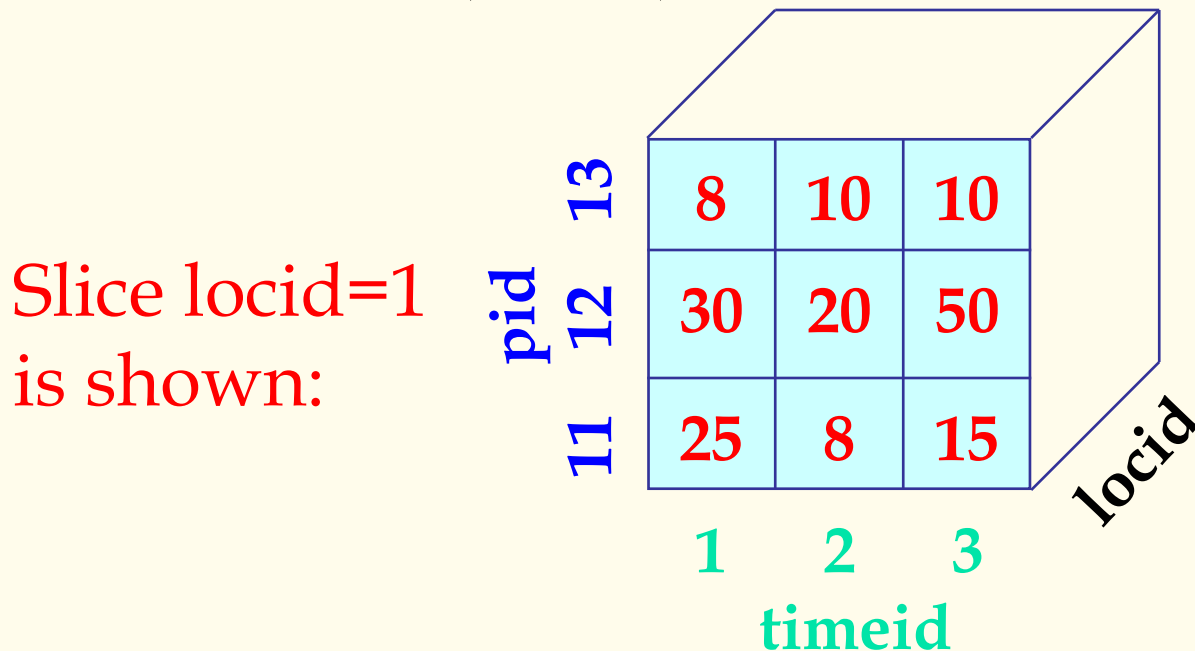
An Example (Cont.)

Color	Size	Number
Light	Small	8
Light	Medium	35
Light	Large	10
Light	all	53
Dark	Small	20
Dark	Medium	10
Dark	Large	5
Dark	all	35
all	Small	28
all	Medium	45
all	Large	15
all	all	88

- Can represent subtotals in relational form by using the value all
- E.g. : obtain (Light, all, 53) and (Dark, all, 35) by aggregating individual tuples with different values for size for each color.

Multidimensional Data Model

- Collection of numeric **measures**, which depend on a set of **dimensions**.
 - E.g., measure **Sales**, dimensions **Product** (key: pid), **Location** (locid), and **Time** (timeid).



pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35



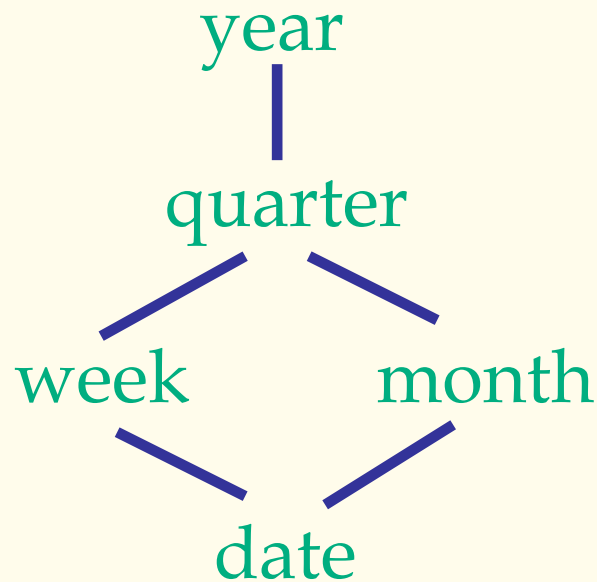
Dimension Hierarchies

- For each dimension, the set of values can be organized in a hierarchy:

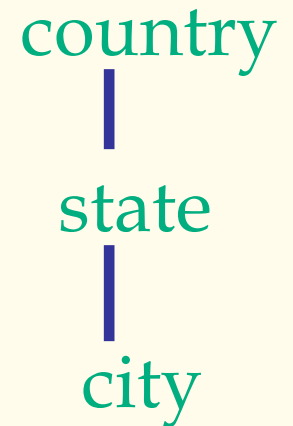
PRODUCT



TIME



LOCATION



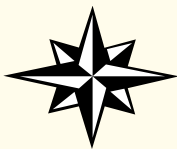
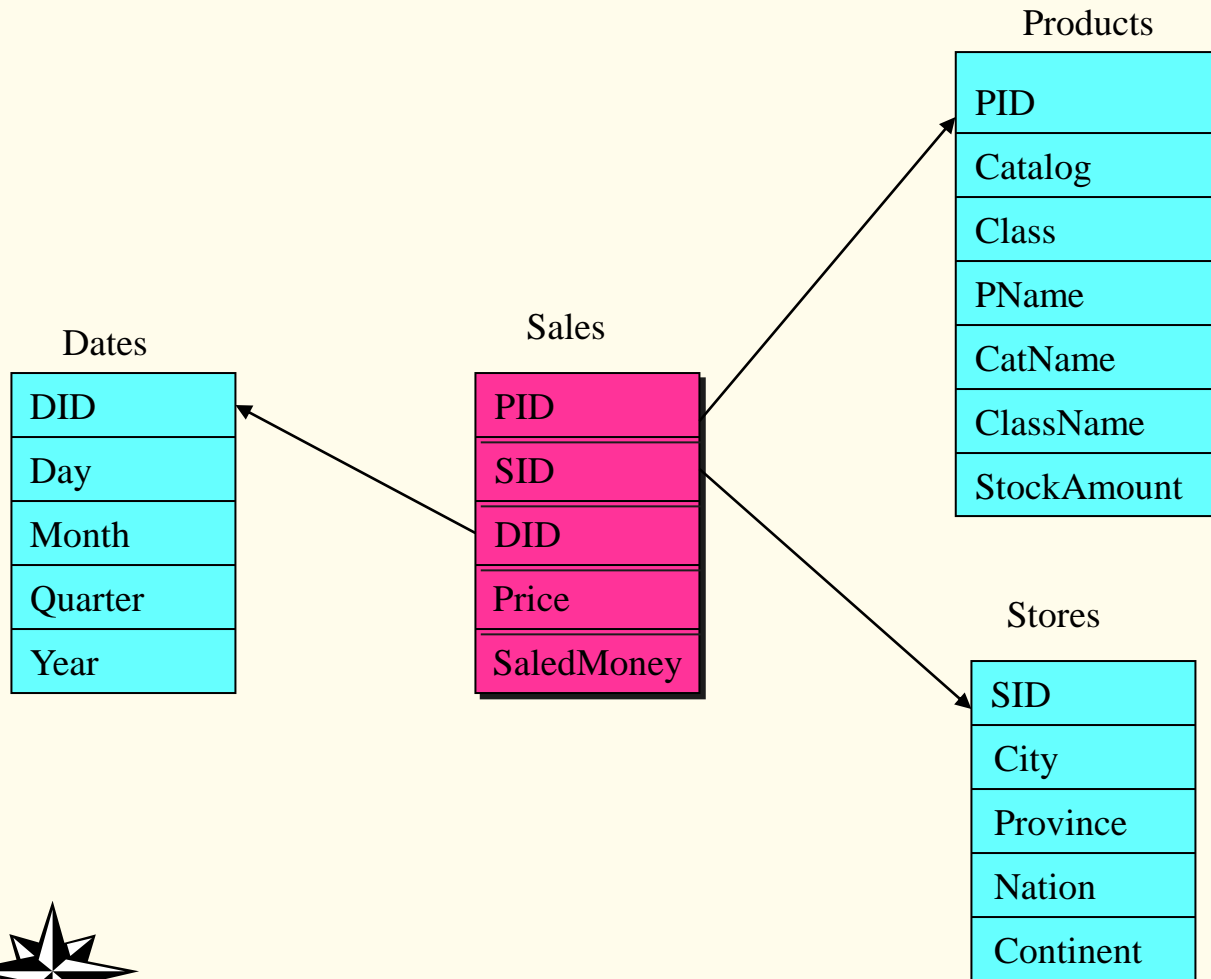


MOLAP vs ROLAP

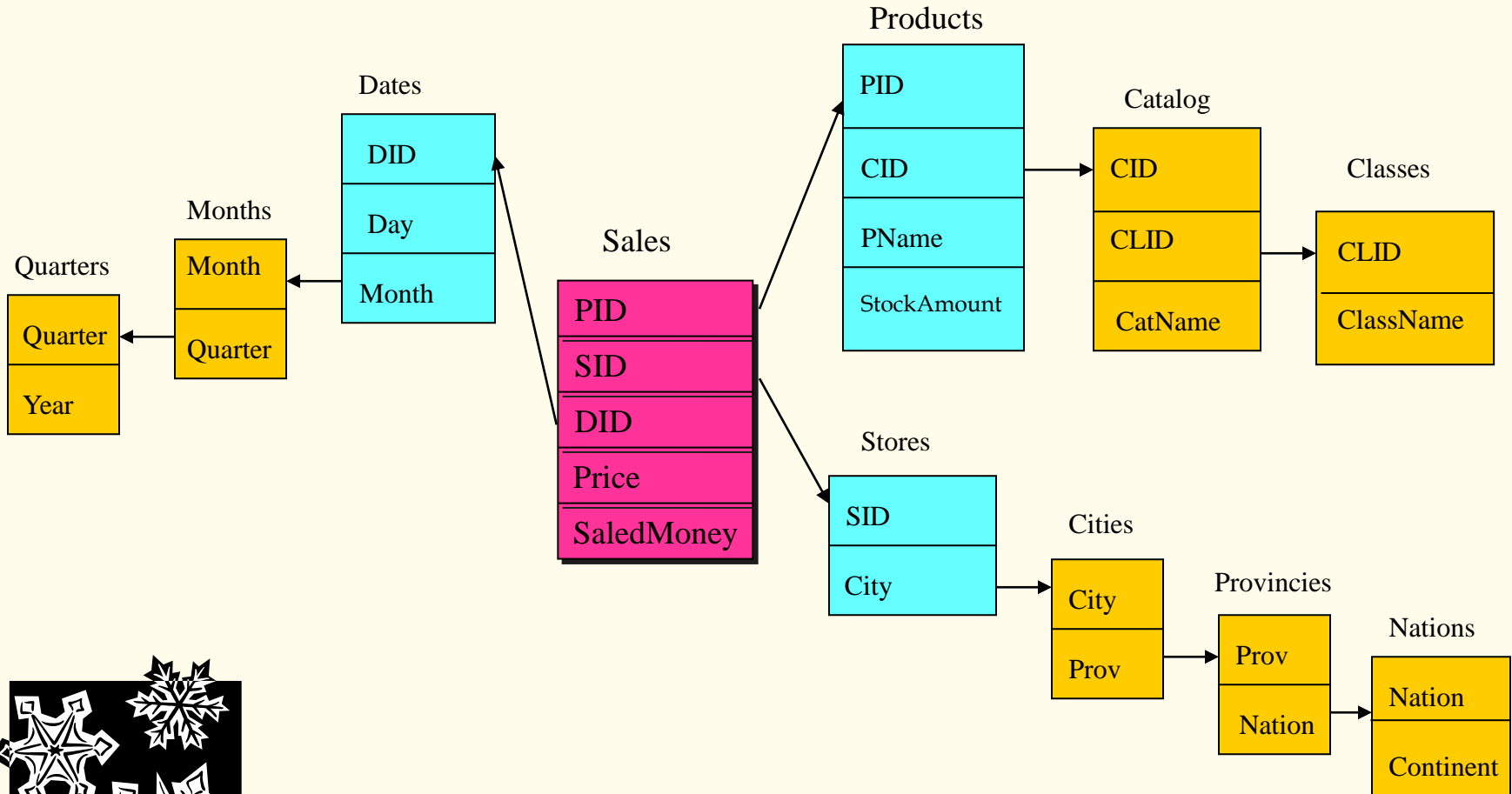
- Multidimensional data can be stored physically in an (disk-resident, persistent) array; called **MOLAP** systems. Alternatively, can store as a relation; called **ROLAP** systems.
- The main relation, which relates dimensions to a measure, is called the **fact table**. Each dimension can have additional attributes and an associated **dimension table**.
 - E.g., **Products(pid, pname, category, price)**
 - Fact tables are *much* larger than dimensional tables.



Star Schema



Snowflake Schema





Materialized View

- Star or snowflake schema are common storing schema in data warehouse. But decision-making are generally not based on star or snowflake schema directly. They are based on different kinds of summarized data computed from star or snowflake schema. Because the computation of aggregation function is very time-consuming, the computing results are often stored as *materialized view* in data warehouse.
- Take P (Products), S (Stores), D (Dates) as example. Their star schema as above.



PSD View

- Take the computation of SUM function as example. Other aggregation functions are similar.
- *Sales of every product in every store at every day.*

```
CREATE VIEW PSD (PID, SID, DID, TotalSales) AS  
SELECT PID, SID, DID, SUM (SaledMoney) AS TotalSales  
FROM Sales  
GROUP BY PID, SID, DID;
```



PS, SD, and PD View

- *Total sales of every product in every store (for all times)*
- SD and PD views are similar (for all products or for all stores)

```
CREATE VIEW PS (PID, SID, TotalSales) AS  
SELECT PID, SID, SUM(TotalSales) AS TotalSales  
FROM PSD  
GROUP BY PID, SID;
```

- PS View can be expressed as PS ALL



P, S, and D View

- *Total sales of every product (for all stores and all times)*
- S and D views are similar (aggregated according to store or date respectively)

```
CREATE VIEW P (PID, TotalSales) AS  
SELECT PID, SUM(TotalSales) AS TotalSales  
FROM PS          /* or PD */  
GROUP BY PID;
```

- P View can be expressed as P ALL ALL



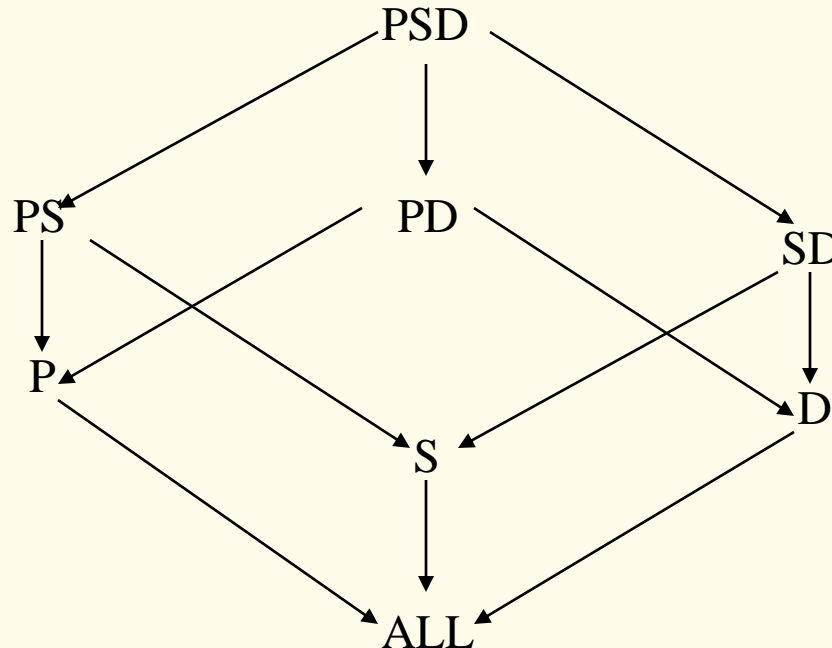
ALL View

- *Total sales for all products and all stores and all times*

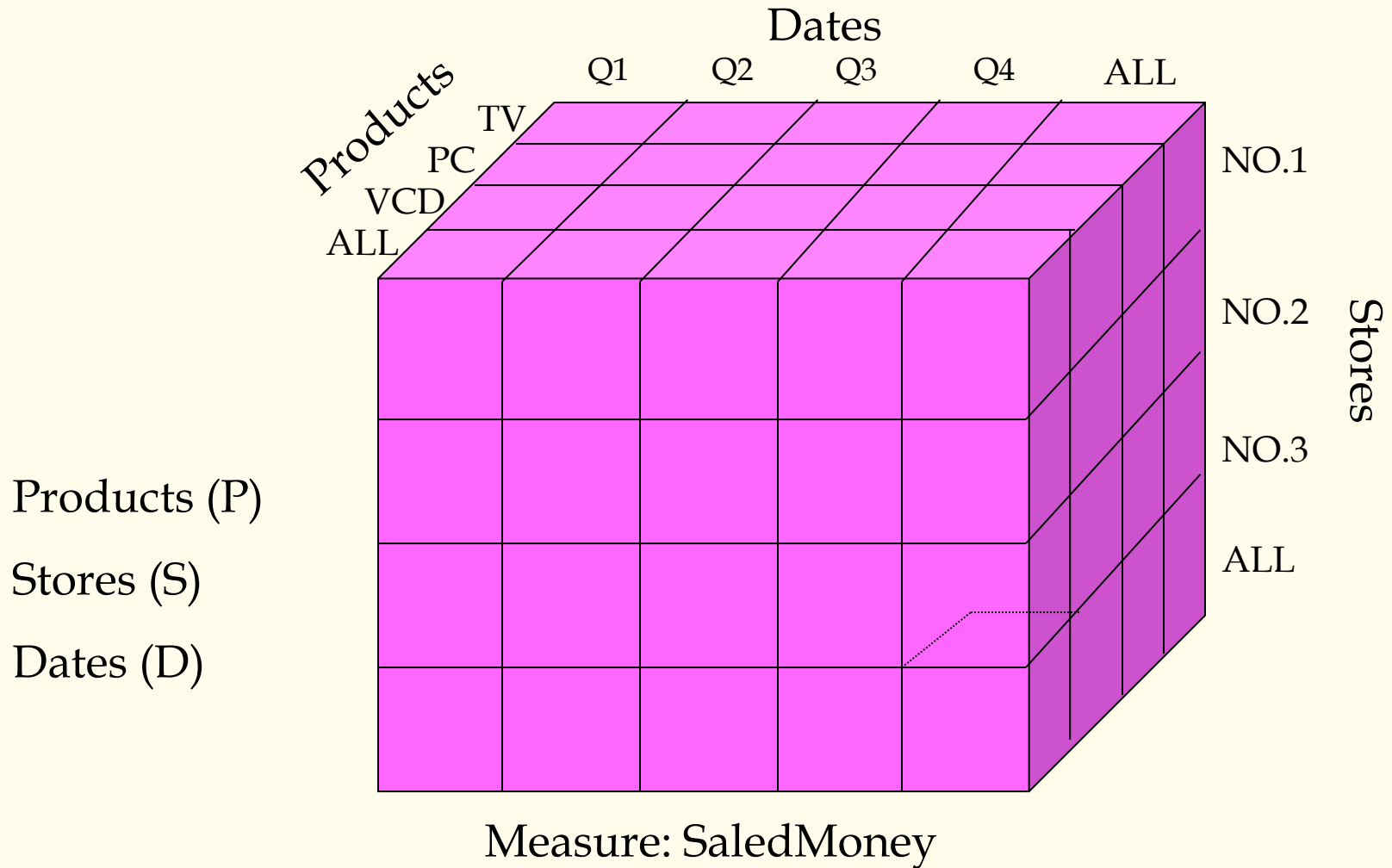
```
CREATE VIEW ALL (TotalSales) AS  
SELECT SUM(TotalSales) AS TotalSales  
FROM P          /* or S or D */
```

Reliant Relationships Between Views

- There are three dimensions in Sales: {P, S, D}, every sub-set of it is corresponding to a view. These are equivalent to 2^3 elements of power-set $\rho(\{P, S, D\})$.
- The following is the reliant relationships between these materialized views:



A Data Cube Example





OLAP Queries

- A common operation is to **aggregate** a measure over one or more dimensions.
 - Find total sales.
 - Find total sales for each city, or for each state.
 - Find top five products ranked by total sales.
- **Roll-up:** Aggregating at different levels of a dimension hierarchy.
 - E.g., Given total sales by city, we can roll-up to get sales by state.



OLAP Queries

- **Drill-down:** The inverse of roll-up.
 - E.g., Given total sales by state, can drill-down to get total sales by city.
 - E.g., Can also drill-down on different dimension to get total sales by product for each state.
- **Pivoting:** Aggregation on selected dimensions.
- **Slicing and Dicing:** Equality and range selections on one or more dimensions.



The CUBE Operator

- Generalizing the previous example, if there are k dimensions, we have 2^k possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.
- **CUBE PID, SID, DID BY SUM Sales**
 - Equivalent to rolling up Sales on all eight subsets of the set {pid, locid, timeid}; each roll-up corresponds to an SQL query of the form:

Lots of work on optimizing the CUBE operator!

```
SELECT SUM (S.sales)
FROM   Sales S
GROUP BY grouping-list
```




Examples of Queries on Cube

```
SELECT PID, SID, Quarter, SUM (SaledMoney) AS TotalSales  
FROM Sales S, Dates D  
WHERE S.DID=D.DID
```

CUBE BY PID, SID, Quarter;

--- *Generate a data cube including 2^3 views.*

- CUBE(TV,No1,Q1)
 - *Query TV's total sales of store No1 in first quarter.*
- CUBE(ALL,No1,Q1)
 - *Query total sales of store No1 in first quarter.*
- CUBE(ALL,ALL,Q1)
 - *Query total sales in first quarter.*
- CUBE(ALL,ALL,ALL)
 - *Query total sales in whole year.*
- CUBE(TV,No1,Q1) + CUBE(VCD,No1,Q1)
- CUBE(ALL,ALL,Q1) / CUBE(ALL,ALL,ALL) * 100%



Examples of Roll-up / Drill-down

```
SELECT PID, SID, DID, SUM (SaledMoney) AS TotalSales  
FROM Sales
```

```
GROUP BY PID, SID, DID ROLLUP;
```

--- Generate four views : PSD, PS, P, and ALL

```
SELECT PID, SID, DID, SUM (SaledMoney) AS TotalSales  
FROM Sales S, Dates D
```

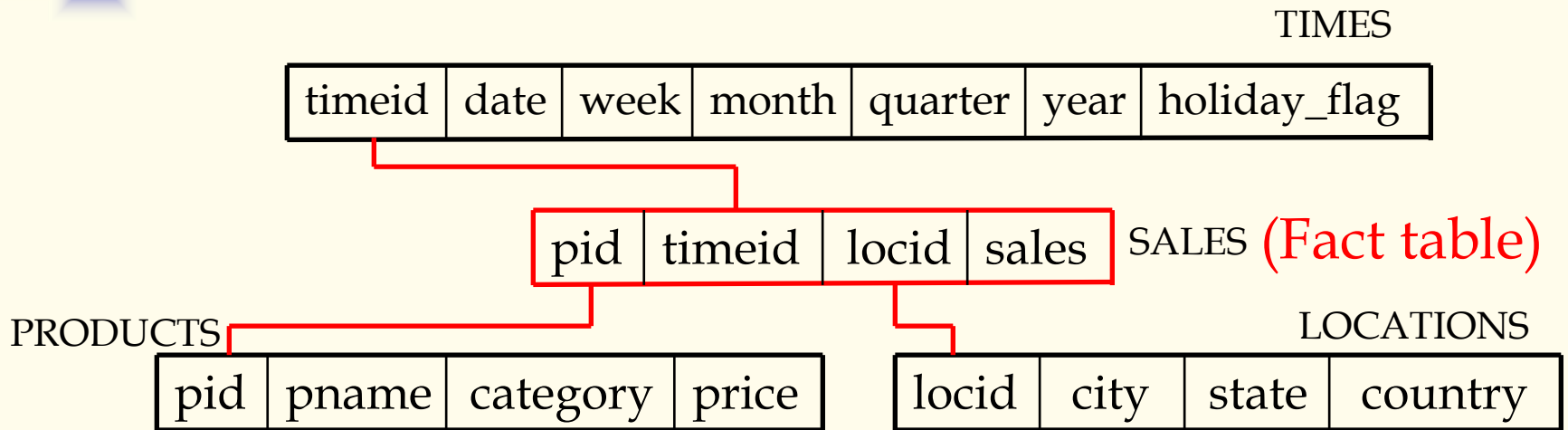
```
WHERE S.DID=D.DID AND Year BETWEEN 1997 AND 1998
```

```
CUBE BY PID, SID, DID
```

```
ROLLUP Day, Month, Quarter, Year;
```

- CUBE(TV, No1, 1998)
- CUBE(TV, No1, 1998.Q4)
- CUBE(TV, No1, 1998.12)

Design Issues



- Fact table in BCNF; dimension tables un-normalized.
 - Dimension tables are small; updates/inserts/deletes are rare. So, anomalies less important than query performance.
- This kind of schema is very common in OLAP applications, and is called a **star schema**; computing the join of all these relations is called a **star join**.

Bitmap index – index itself is data

Date	Store	State	Class	Sales
3/1/96	32	NY	A	6
3/1/96	36	MA	A	9
3/1/96	38	NY	B	5
3/1/96	41	CT	A	11
3/1/96	43	NY	A	9
3/1/96	46	RI	B	3
3/1/96	47	CT	B	7
3/1/96	49	NY	A	12

Bitmap Index for Sales

8bit	4bit	2bit	1bit
0	1	1	0
1	0	0	1
0	1	0	1
1	0	1	1
1	0	0	1
0	0	1	1
0	1	1	1
1	1	0	0

Bitmap Index for State

AK	AR	CA	CO	CT	MA	NY	RI
0	0	0	0	0	0	1	0		
0	0	0	0	0	1	0	0		
0	0	0	0	0	0	1	0		
0	0	0	0	1	0	0	0		
0	0	0	0	0	0	1	0		
0	0	0	0	0	0	0	1		
0	0	0	0	1	0	0	0		
0	0	0	0	0	0	1	0		

for Class

A	B	C
1	0	0
1	0	0
0	1	0
1	0	0
1	0	0
0	1	0
0	1	0
1	0	0

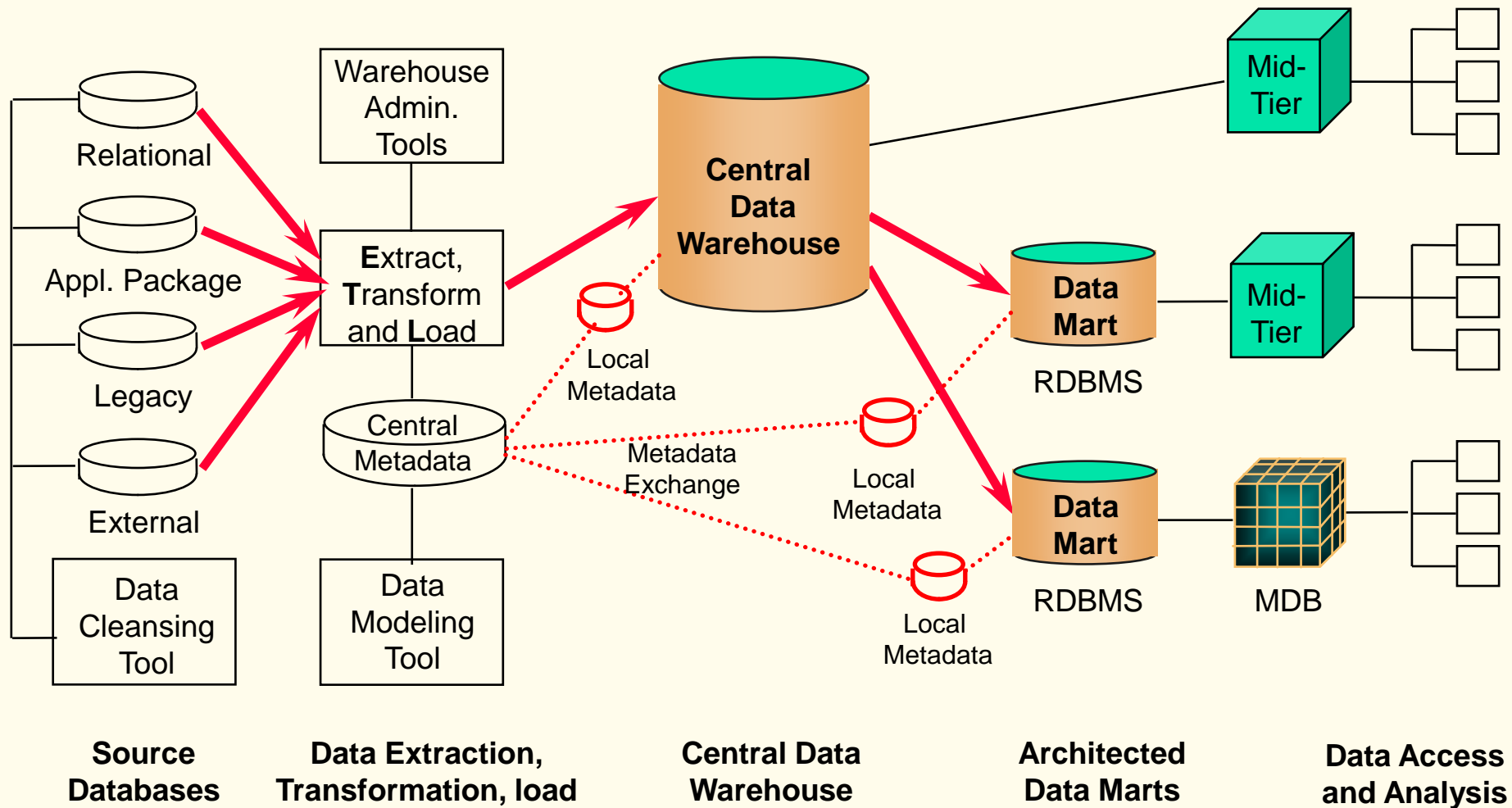
- Total sales = ? ($4*8+4*4+4*2+6*1=62$)
- How many class A store in NY ? (3)
- Sales of class A store in NY = ? ($2*8+2*4+1*2+1*1=27$)
- How many stores in CT ? (2)
- Join operation (query product list of class A store in NY)



The Procedure of Data Warehouse Engineering

- Requirements Analysis (Project Plan)
- Data Warehouse Architecture Design
- Environments Construction
- Data Warehouse Schema Design
- ETL Processing of Data
- Meta Data Management
- Front Applications Design & Implementation
- Testing
- Running & Maintaining

Data Warehouse Architecture Design



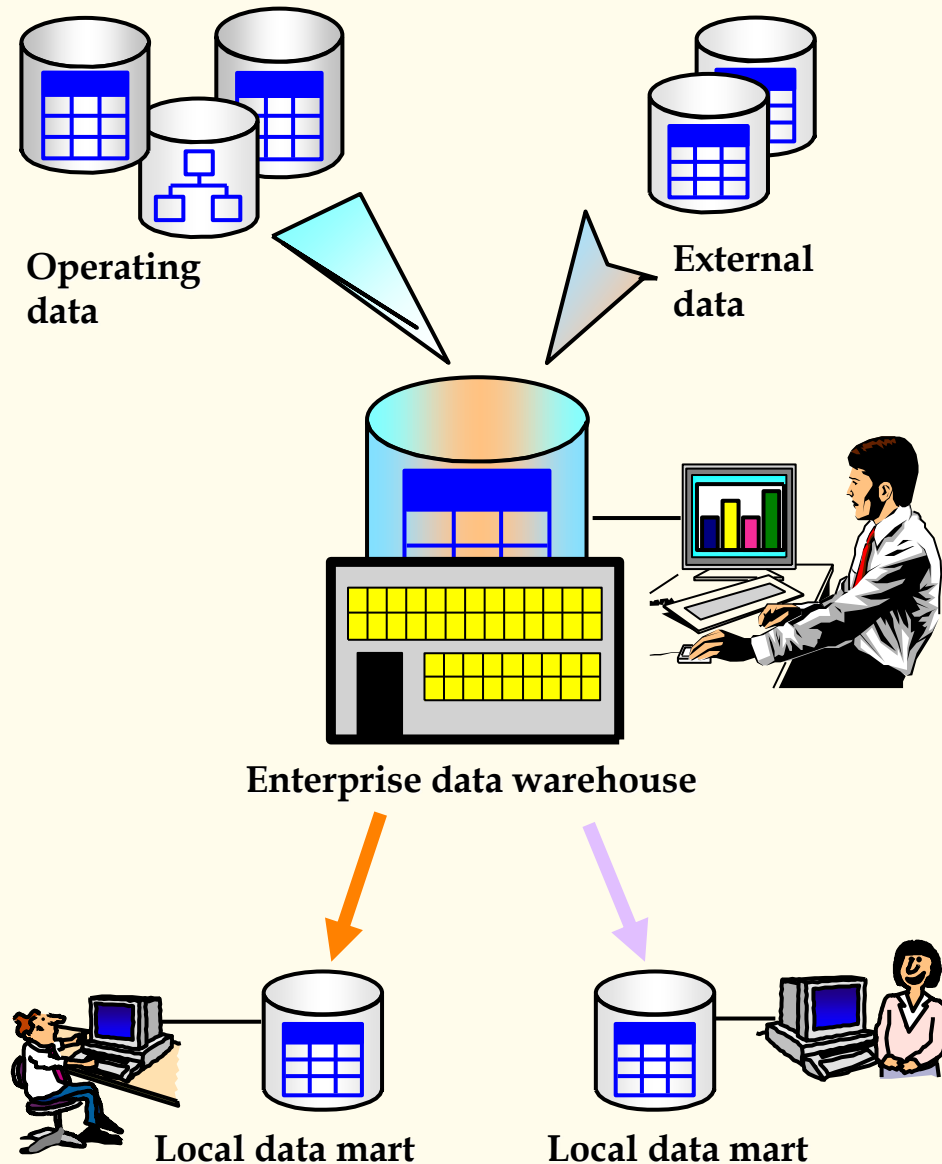


Data Warehouse Schema Design

- Top Down
- Bottom Up
- Mixed Method

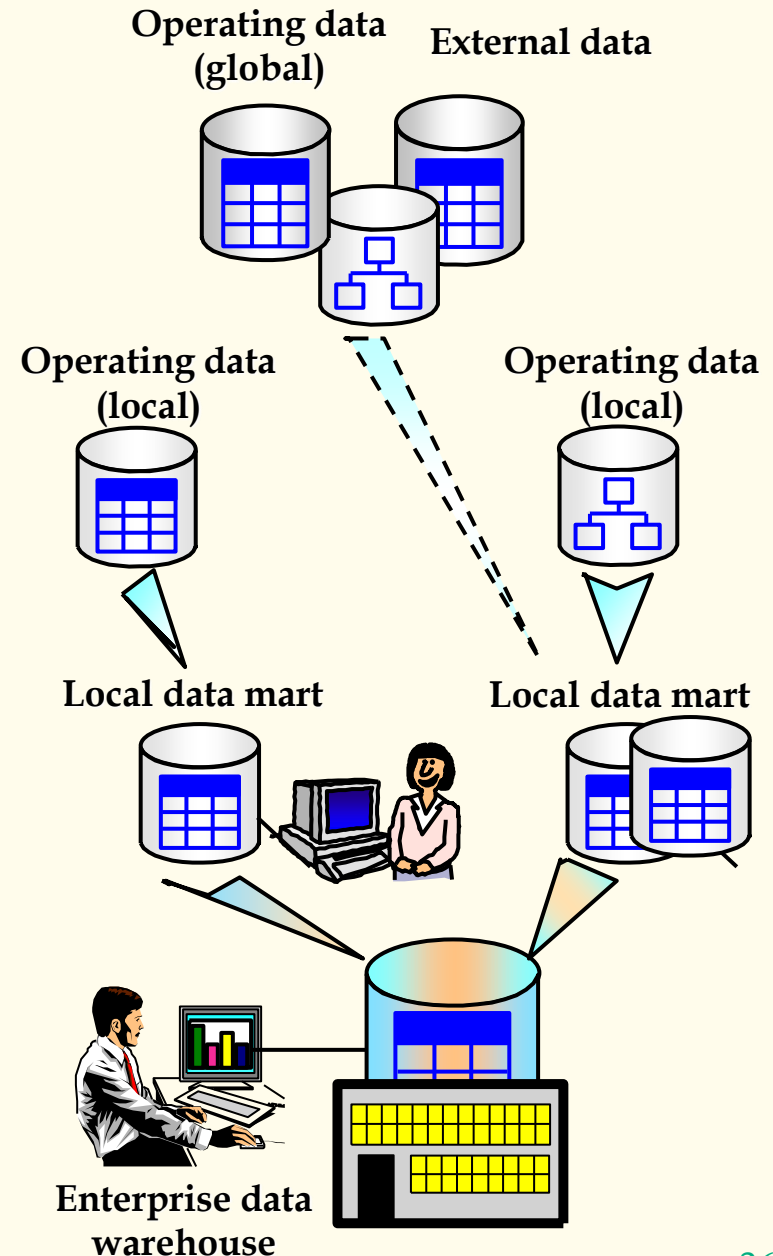
Top Down

- Construct enterprise data warehouse (EDW)
 - ✓ Common central data schema
 - ✓ One-off data reconstruction
 - ✓ Minimized data redundancy and inconsistency
 - ✓ Detail and historical data; global data exploration
- Construct data marts from EDW
 - ✓ Sub-set in EDW which related with a department
 - ✓ Summarized data
 - ✓ Rely on the availability of data in EDW



Bottom Up

- Construct data mart of department
 - ✓ Limited in a subject area
 - ✓ Local business requirements are fulfilled rapidly
 - ✓ Department autonomy
 - ✓ A good guidance to the construction of data marts of other departments
 - ✓ Need to do data reconstruction for every department
 - ✓ Have redundancy and inconsistency at a certain extent
 - ✓ A feasible way
- Enlarged to enterprise data warehouse (EDW)
 - ✓ Take the construction of EDW as a long-term target





Summary

- Decision support is an emerging, rapidly growing subarea of databases.
- Involves the creation of large, consolidated data repositories called data warehouses.
- Warehouses exploited using sophisticated analysis techniques: complex SQL queries and OLAP “multidimensional” queries (influenced by both SQL and spreadsheets).
- New techniques for database design, indexing, view maintenance, and interactive querying need to be supported.



Main Contents

- Data warehouse
- OLAP
- Data mining
- Information retrieval
- Semistructured data and XML



8.2 Data Mining

Definition:

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

Example pattern (Census Bureau Data):

If (relationship = husband), then (gender = male). 99.6%



Definition (Cont.)

Data mining is the exploration and analysis of large quantities of data in order to discover valid, novel, potentially useful, and ultimately understandable patterns in data.

- **Valid:** The patterns hold in general.
- **Novel:** We did not know the pattern beforehand.
- **Useful:** We can devise actions from the patterns.
- **Understandable:** We can interpret and comprehend the patterns.



Why Use Data Mining Today?

Human analysis skills are inadequate:

- Volume and dimensionality of the data
- High data growth rate

Availability of:

- Data
- Storage
- Computational power
- Off-the-shelf software
- Expertise



An Abundance of Data

- Supermarket scanners, POS data
- Preferred customer cards
- Credit card transactions
- Direct mail response
- Call center records
- ATM machines
- Demographic data
- Sensor networks
- Cameras
- Web server logs
- Customer web site trails



Much Commercial Support

- Many data mining tools
 - <http://www.kdnuggets.com/software>
- Database systems with data mining support
- Visualization tools
- Data mining process support
- Consultants



Why Use Data Mining Today?

Competitive pressure!

“The secret of success is to know something that nobody else knows.”

Aristotle Onassis

- Competition on service, not only on price (Banks, phone companies, hotel chains, rental car companies)
- Personalization, CRM
- The real-time enterprise
- “Systemic listening”
- Security, homeland defense



Data Mining is Supported by Three Sufficiently Mature Technologies

- **Massive data collections**

Commercial databases (using high performance engines) are growing at exceptional rates

- **Powerful multiprocessor computers**

cost-effective parallel multiprocessor computer technology

- **Data mining algorithms**

under development for decades, in research areas such as statistics, artificial intelligence, and machine learning, but now implemented as mature, reliable, understandable tools that consistently **outperform** older statistical methods



Applications, Operations, and Techniques

Applications	Database Marketing Customer Segmentation Customer Retention Fraud Detection Credit Checking Web Site Analysis, etc.
Operations	Classification and Prediction Clustering Association Analysis Forecasting
Techniques	Cluster Analysis Nearest Neighbour Neural Networks Naïve-Bayes Decision Trees



The Knowledge Discovery Process

Steps:

1. Identify business problem
2. Data mining
3. Action
4. Evaluation and measurement
5. Deployment and integration into businesses processes



Data Mining Step in Detail

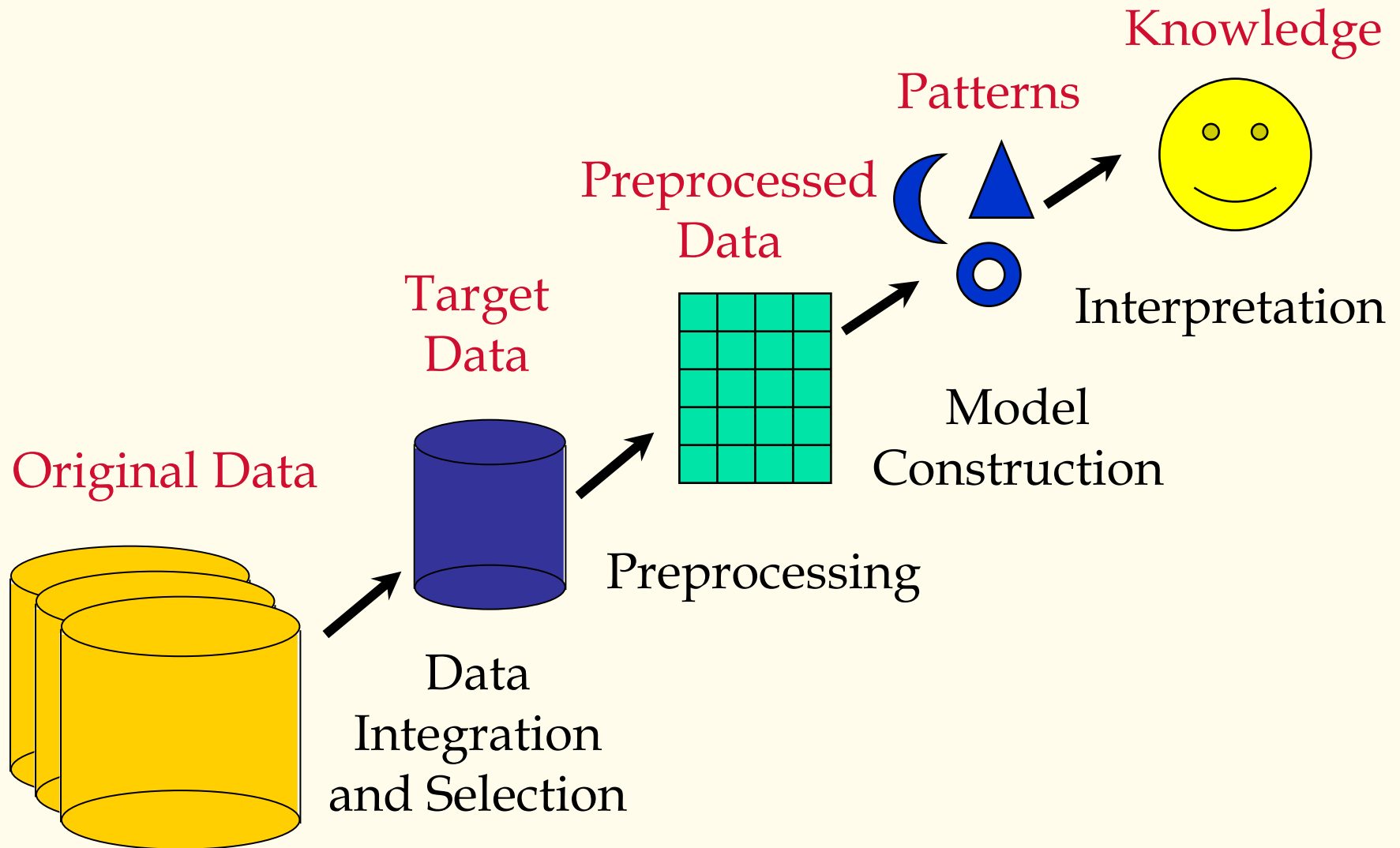
2.1 Data preprocessing

- Data selection: Identify target datasets and relevant fields
- Data cleaning
 - Remove noise and outliers
 - Data transformation
 - Create common units
 - Generate new fields

2.2 Data mining model construction

2.3 Model evaluation

Preprocessing and Mining





Example Application: Sky Survey

- Input data: 3 TB of image data with 2 billion sky objects, took more than six years to complete
- Goal: Generate a catalog with all objects and their type
- Method: Use decision trees as data mining model
- Results:
 - 94% accuracy in predicting sky object classes
 - Increased number of faint objects classified by 300%
 - Helped team of astronomers to discover 16 new high red-shift quasars in one order of magnitude less observation time



Example Application: Market Baskets

- An important form of mining involves *market baskets* = sets of “items” that are purchased together as a customer leaves a store.
- Summary of basket data is *frequent itemsets* = sets of items that often appear together in baskets.
- If people often buy hamburger and ketchup together, the store can:
 1. Put hamburger and ketchup near each other and put potato chips between.
 2. Run a sale on hamburger and raise the price of ketchup.



Data Mining: Types of Data

- Relational data and transactional data
 - Spatial and temporal data, spatial-temporal observations
 - Time-series data
 - Text
 - Images, video
 - Mixtures of data
 - Sequence data
-
- Features from processing other data sources



Types of Variables

- *Numerical*: Domain is ordered and can be represented on the real line (e.g., age, income)
- *Nominal* or *categorical*: Domain is a finite set without any natural ordering (e.g., occupation, marital status, race)
- *Ordinal*: Domain is ordered, but absolute differences between values is unknown (e.g., preference scale, severity of an injury)



Data Mining Operations

Each operation (or task) reflects a different way of distinguishing patterns or trends in complex datasets

- Classification and prediction
- Clustering
- Association analysis and sequential analysis
- Forecasting



Classification and Prediction

Classification is the operation most commonly supported by commercial data mining tools

- It is the process of sub-dividing a data set with regard to a number of specific outcomes.
 - For example, classifying customers into ‘high’ and ‘low’ categories with regard to credit risk.
- The category or ‘class’ into which each customer is placed is the ‘outcome’ of the classification.



Techniques for Classification and Prediction

- **Decision trees**
- **Neural networks**
- **Nearest neighbour algorithms**



Understanding v Prediction

- Sophisticated classification techniques enable us to discover new patterns in large and complex data sets.
- Classification is a powerful aid to understanding a particular problem.
- In some cases, improved understanding is sufficient. It may suggest new initiatives and provide information that improves future decision making.
- Often the reason for developing an accurate classification model is to improve our capability for prediction.



Training

- A classification model is said to be ‘trained’ on historical data, for which the outcome is known for each record.
- But beware overfitting: 100 per cent of customers called Smith who live at 28 Arcadia Street responded to our offer.
- One would then use a separate test dataset of historical data to validate the model.
- The model could then be applied to a new, unclassified data set in order to predict the outcome for each record.



Clustering

- Clustering is an unsupervised operation - no training.
- It is used to find groupings of similar records in a data set without any preconditions as to what that similarity may involve.
- Clustering is used to identify interesting groups in a customer base that may not have been recognised before.
- Often undertaken as an exploratory exercise before doing further data mining using a classification technique.



Techniques for Clustering

- Cluster analysis
- Neural networks
- + Good visualization support for “playing” with clusters, to see if they make sense and are useful in a business context



Association Analysis

- **Association Rule** looks for links between records in a data set.

Sometimes referred to as 'market basket analysis', its most common aim is to discover which items are generally purchased at the same time.

- **Time Sequential** looks for temporal links between purchases, rather than relationships between items in a single transaction.



Example of Association Rule

- Consider the following beer and nappy example:
 - 500,000 transactions
 - 20,000 transactions contain nappies (4%)
 - 30,000 transactions contain beer (6%)
 - 10,000 transactions contain both nappies and beer (2%)



Support (or prevalence)

- Measures how often items occur together, as a percentage of the total transactions.
- In this example, beer and nappies occur together 2% of the time ($10,000/500,000$).



Confidence (or predictability)

- Measures how much a particular item is dependent on another.
- Because 20,000 transactions contain nappies and 10,000 of these transactions contain beer, when people buy nappies, they also buy beer 50% of the time.
- The confidence for the rule:
When people buy nappies they also buy beer 50% of the time.
is 50%.



Confidence (cont)

- Because 30,000 transactions contain beer and 10,000 of these transactions contain nappies, when people buy beer, they also buy nappies 33.33% of the time.
- The confidence for the rule:
When people buy beer they also buy nappies 1/3 of the time.
is 33.33%.
- Both these rules have the same support: 2%



Expected Confidence

- In the absence of any knowledge about what else was bought, we can also make the following assertions from the available data:

People buy nappies 4% of the time.

People buy beer 6% of the time.

- These numbers - 4% and 6% - are called the **expected confidence** of buying nappies or beer, regardless of what else is purchased.



Lift

- Measures the ratio between the confidence of a rule and the expected confidence that the second product will be purchased. Lift is measures of the strength of an effect.
- In our example, the confidence of the nappies-beer buying rule is 50%, whilst the expected confidence is 6% that an arbitrary customer will buy beer.
- So, the lift provided by the nappies-beer rule is:
 $8.33 (= 50\% / 6\%)$.
- A key goal of an association analysis is to find rules that have a substantial lift like this.



Forecasting

- Forecasting (unlike prediction based on classification models) concerns the prediction of continuous values, such as person's income based on various personal details, or the level of the stock market.
- Simpler forecasting problems involve a single continuous value based on a series of unordered examples.
- More complex problem is to predict one or more values based on a sequential pattern.
- Techniques include statistical time-series analysis as well as neural networks.



Data Mining Techniques

Each technique is used to support a particular data mining operation

- Cluster Analysis
- Association Rules Algorithm
- Time Sequence Algorithm
- Decision Trees

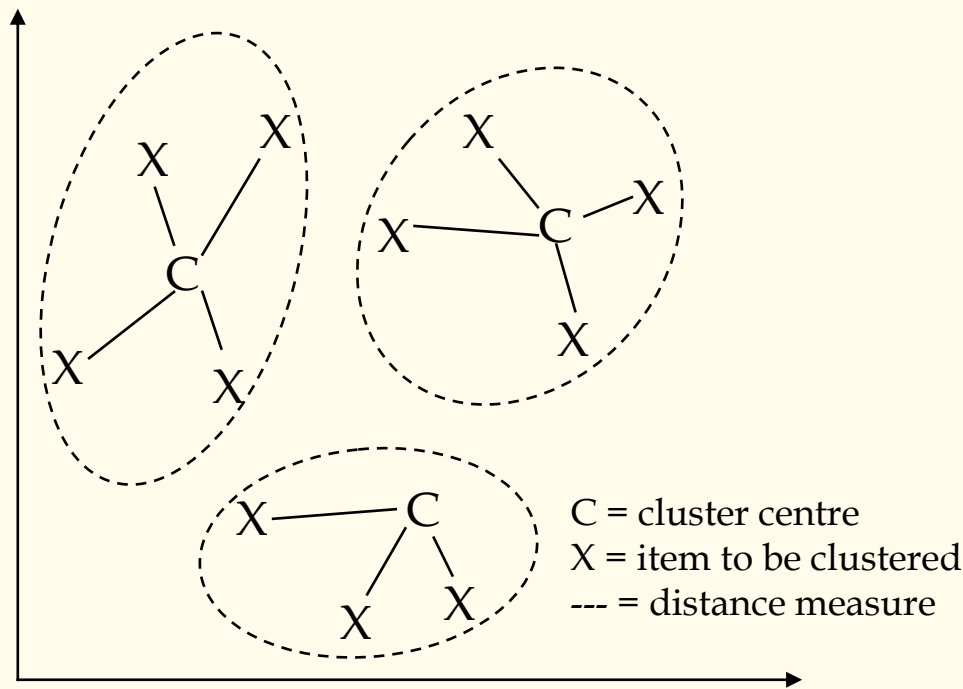


Cluster Analysis

- Identifies the relationships that exist between items on the basis of their similarity and dissimilarity.
- Clusters are typically based around a “centre” value.
- How centres are initially defined and adjusted varies between algorithms.
- One method is to start with a random set of centres, which are then adjusted, added to and removed as the analysis progresses.

Cluster Analysis (cont)

- To identify items that belong to a cluster, some measure must be used that gauges the similarity between items within a cluster and their dissimilarity to items in other clusters.



- This is typically measured as their distance from each other and from the cluster centres within a multi-dimensional space, where each dimension represents one of the variables being compared.
- The key is the design and implementation of distance function.



Association Rule Algorithm

- Apriori algorithm (R.Agrawal, etc.)
 1. Find out all frequent itemsets
 2. Generate candidate association rules based on frequent itemsets and verify their confidence.

Procedure AprioriAlg()

Begin

```
L1:={frequent 1-itemsets};    /* 1-itemsets is itemsets with single item */
for(k:=2; Lk-1≠Φ; k++) {
  Ck:= AprioriGen(Lk-1);      /* candidate itemset generated by AprioriGen */
  forall transactions in the dataset do
    {forall candidates c ∈ Ck contained in t do
      c.count++;
    }
  Lk := {c ∈ Ck | c.count >= min-support}
}
```

end



Time Sequence Algorithm

- AprioriALL algorithm
 1. Sort (according to customer ID and then transaction time)
 2. Find out frequent itemset (similar as Apriori, but count according to customer)
 3. Transform customer transaction sequence
 4. Find out frequent sequences (similar as Apriori, but AprioriGen() is some different)
 5. Select maximum frequent sequences, they are the answer.



Classification Algorithm

- The generation and maintenance algorithm of decision tree

partition(S)

if(all tuples in S are of the same class) then

 return; /*don't need to classify */

foreach value of attribute A do

evaluate the split on that value;

use best split found to partition S into S1 and S2;

partition S1; /* call partition() recursively */

partition S2;

- Evaluate each split:

$$\text{Gini}(S) = 1 - \sum_{j=1}^n p_j^2 \quad /* n \text{ is class number in } S; p_j \text{ is frequency of every class in } S */$$

$$\text{Gini}_{\text{split}}(S) = (N_1/N) * \text{gini}(S_1) + (N_2/N) * \text{gini}(S_2)$$

Select smallest $\text{Gini}_{\text{split}}(S)$ as split condition.



8. New Research and Application Fields



Main Contents

- Data warehouse
- OLAP
- Data mining
- Information retrieval
- Semistructured data and XML



8.3 Information Retrieval

- A research field traditionally separate from Databases
 - Goes back to IBM, Rand and Lockheed in the 50's
 - G. Salton at Cornell in the 60's
 - Lots of research since then
- Products traditionally separate
 - Originally, document management systems for libraries, government, law, etc.
 - Gained prominence in recent years due to web search



IR vs. DBMS

- Seem like very different beasts:

IR	DBMS
Imprecise Semantics	Precise Semantics
Keyword search	SQL
Unstructured data format	Structured data
Read-Mostly. Add docs occasionally	Expect reasonable number of updates
Page through top k results	Generate full answer

- Both support queries over large datasets, use indexing.
 - In practice, you currently have to choose between the two.



IR's “Bag of Words” Model

- Typical IR data model:
 - Each document is just a bag (multiset) of words (“terms”)
- Detail 1: “Stop Words”
 - Certain words are considered irrelevant and not placed in the bag
 - e.g., “the”
 - e.g., HTML tags like <H1>
- Detail 2: “Stemming” and other content analysis
 - Using English-specific rules, convert words to their basic form
 - e.g., “surfing”, “surfed” --> “surf”



Boolean Text Search

- Find all documents that match a Boolean containment expression:
 “Windows”
 AND (“Glass” OR “Door”)
 AND NOT “Microsoft”
- **Note:** Query terms are also filtered via stemming and stop words.
- When web search engines say “10,000 documents found”, that’s the Boolean search result size (subject to a common “max # returned’ cutoff).



Text “Indexes”

- When IR folks say “text index” ...
 - Usually mean more than what DB people mean
- In our terms, both “tables” and indexes
 - Really a logical schema (i.e., tables)
 - With a physical schema (i.e., indexes)
 - Usually not stored in a DBMS
 - Tables implemented as files in a file system
 - We’ll talk more about this decision soon



A Simple Relational Text Index

- Create and populate a table
`InvertedFile(term string, docURL string)`
- Build a B+-tree or Hash index on `InvertedFile.term`
 - Alternative 3 (<Key, list of URLs> as entries in index) critical here for efficient storage!!
 - Fancy list compression possible, too
 - Note: URL instead of RID, the web is your “heap file”!
 - Can also *cache* pages and use RIDs
- This is often called an “inverted file” or “inverted index”
 - Maps from **words** -> **docs**
- Can now do single-word text search queries!



An Inverted File

- Search for
 - “databases”
 - “microsoft”

term	docURL
data	http://www-inst.eecs.berkeley.edu/~cs186
database	http://www-inst.eecs.berkeley.edu/~cs186
date	http://www-inst.eecs.berkeley.edu/~cs186
day	http://www-inst.eecs.berkeley.edu/~cs186
dbms	http://www-inst.eecs.berkeley.edu/~cs186
decision	http://www-inst.eecs.berkeley.edu/~cs186
demonstrate	http://www-inst.eecs.berkeley.edu/~cs186
description	http://www-inst.eecs.berkeley.edu/~cs186
design	http://www-inst.eecs.berkeley.edu/~cs186
desire	http://www-inst.eecs.berkeley.edu/~cs186
developer	http://www.microsoft.com
differ	http://www-inst.eecs.berkeley.edu/~cs186
disability	http://www.microsoft.com
discussion	http://www-inst.eecs.berkeley.edu/~cs186
division	http://www-inst.eecs.berkeley.edu/~cs186
do	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www-inst.eecs.berkeley.edu/~cs186



Handling Boolean Logic

- How to do “term1” OR “term2”?
 - Union of two DocURL sets!
- How to do “term1” AND “term2”?
 - Intersection of two DocURL sets!
 - Can be done by sorting both lists alphabetically and merging the lists
- How to do “term1” AND NOT “term2”?
 - Set subtraction, also done via sorting
- How to do “term1” OR NOT “term2”?
 - Union of “term1” and “NOT term2”.
 - “Not term2” = all docs not containing term2. Large set !!!
 - Usually not allowed!
- Refinement: What order to handle terms if you have many ANDs/NOTs?



Boolean Search in SQL

“Windows” AND (“Glass” OR “Door”)
AND NOT “Microsoft”

- (SELECT docURL FROM InvertedFile
WHERE word = “windows”
INTERSECT
SELECT docURL FROM InvertedFile
WHERE word = “glass” OR word = “door”)
EXCEPT
SELECT docURL FROM InvertedFile
WHERE word=“Microsoft”
ORDER BY *relevance()*



Boolean Search in SQL

- Really only one SQL query in Boolean Search IR:
 - Single-table selects, UNION, INTERSECT, EXCEPT
- relevance () is the “secret sauce” in the search engines:
 - Combos of statistics, linguistics, and graph theory tricks!
 - Unfortunately, not easy to compute this efficiently using typical DBMS implementation.



Computing Relevance

- Relevance calculation involves how often search terms appear in doc, and how often they appear in collection:
 - More search terms found in doc → doc is more relevant
 - Greater importance attached to finding *rare* terms
- Doing this efficiently in current SQL engines is not easy:
 - “Relevance of a doc wrt a search term” is a function that is called once per doc the term appears in (docs found via inv. index):
 - For efficient fn computation, for each term, we can store the # times it appears in each doc, as well as the # docs it appears in.
 - Must also sort retrieved docs by their relevance value.
 - Also, think about Boolean operators (if the search has multiple terms) and how they affect the relevance computation!
 - An object-relational or object-oriented DBMS with good support for function calls is better, but you still have long execution path-lengths compared to optimized search engines.



Fancier: Phrases and “Near”

- Suppose you want a phrase
 - E.g., “Happy Days”
- Different schema:
 - InvertedFile (**term** string, count int, position int, DocURL string)
 - Alternative 3 index on **term**
- Post-process the results
 - Find “Happy” AND “Days”
 - Keep results where positions are 1 off
 - Doing this well is like *join processing*
- Can do a similar thing for “term1” NEAR “term2”
 - Position < k off



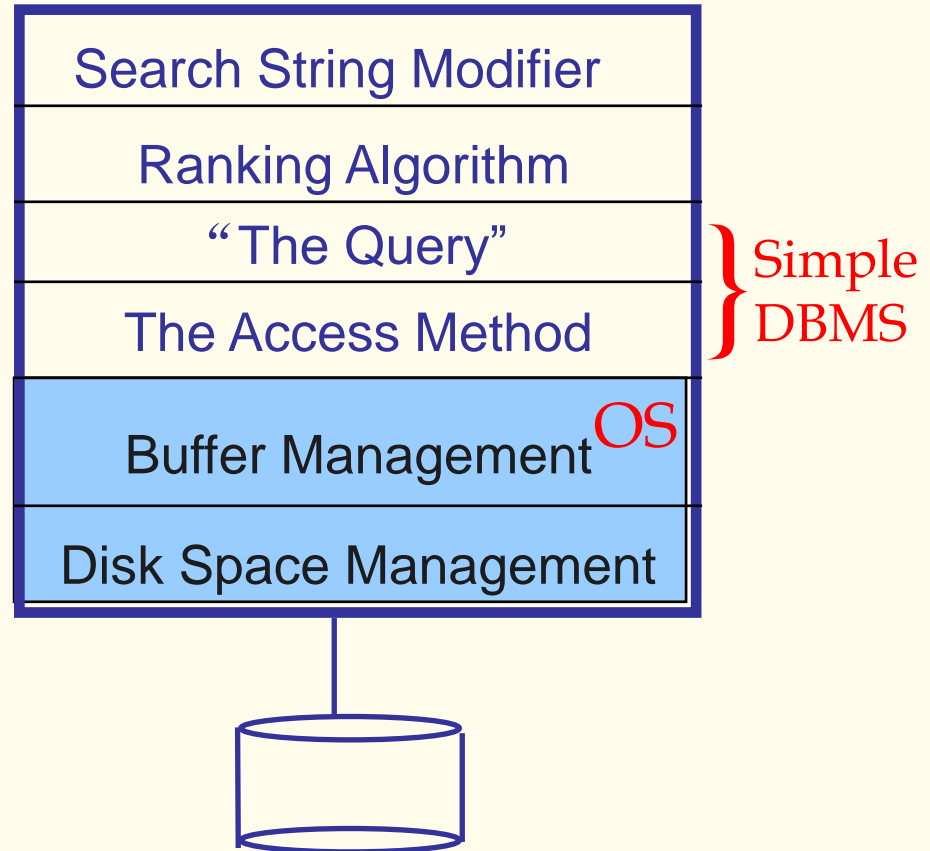
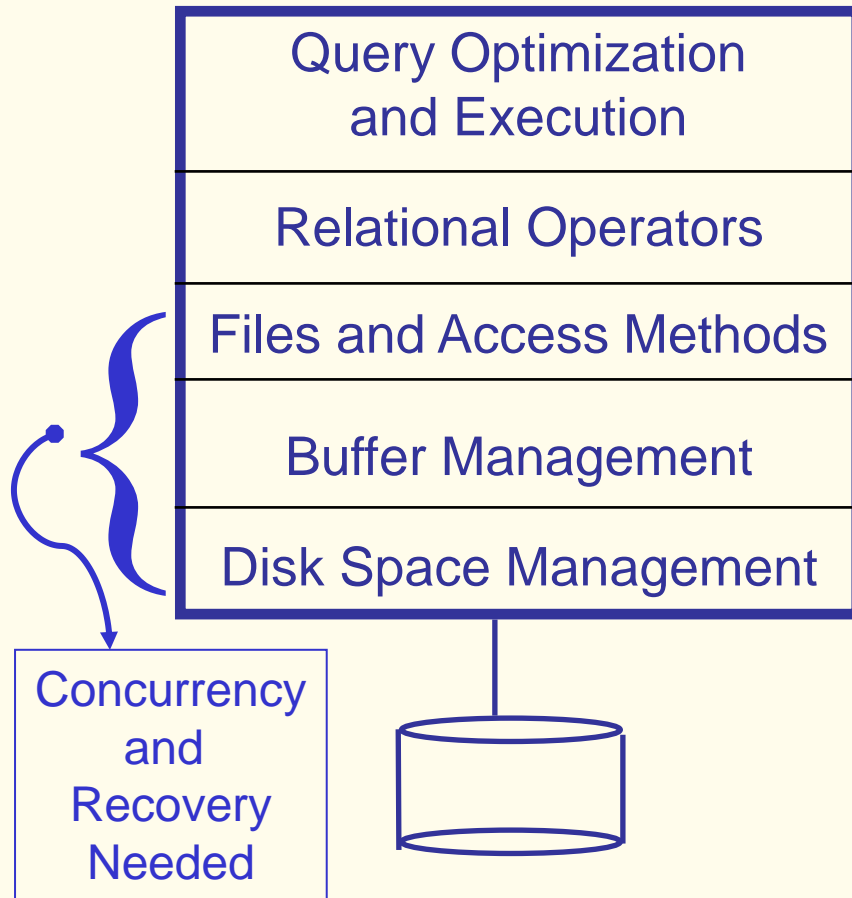
Updates and Text Search

- Text search engines are designed to be query-mostly:
 - Deletes and modifications are rare
 - Can postpone updates (nobody notices, no transactions!)
 - Updates done in batch (rebuild the index)
 - Can't afford to go off-line for an update?
 - Create a 2nd index on a separate machine
 - Replace the 1st index with the 2nd!
 - So no concurrency control problems
 - Can compress to search-friendly, update-unfriendly format
- Main reason why text search engines and DBMSs are usually separate products.
 - Also, text-search engines tune that one SQL query to death!

DBMS vs. Search Engine Architecture

DBMS

Search Engine





IR vs. DBMS Revisited

■ Semantic Guarantees

- DBMS guarantees transactional semantics
 - If inserting Xact commits, a later query *will see* the update
 - Handles multiple concurrent updates correctly
- IR systems do not do this; nobody notices!
 - Postpone insertions until convenient
 - No model of correct concurrency

■ Data Modeling & Query Complexity

- DBMS supports any schema & queries
 - Requires you to define schema
 - Complex query language hard to learn
- IR supports only one schema & query
 - No schema design required (unstructured text)
 - Trivial to learn query language



IR vs. DBMS, Contd.

■ Performance goals

- DBMS supports general SELECT
 - Plus mix of INSERT, UPDATE, DELETE
 - General purpose engine must always perform “well”
- IR systems expect only one stylized SELECT
 - Plus delayed INSERT, unusual DELETE, no UPDATE.
 - Special purpose, must run super-fast on “The Query”
 - Users rarely look at the full answer in Boolean Search



Lots More in IR ...

- How to “rank” the output? I.e., how to compute relevance of each result item w.r.t. the query?
 - Doing this well / efficiently is hard!
- Other ways to help users paw through the output?
 - Document “clustering”, document visualization
- How to take advantage of hyperlinks?
 - Really cute tricks here!
- How to use compression for better I/O performance?
 - E.g., making RID lists smaller
 - Try to make things fit in RAM!
- How to deal with synonyms, misspelling, abbreviations?
- How to write a good web crawler?



8. New Research and Application Fields



Main Contents

- Data warehouse
- OLAP
- Data mining
- Information retrieval
- Semistructured data and XML



8.4 Semistructured Data and XML

How the Web is Today

- HTML documents
 - often generated by applications
 - consumed by humans only
 - easy access: across platforms, across organizations
- No application interoperability:
 - HTML not understood by applications
 - screen scraping brittle
 - Database technology: client-server
 - still vendor specific



New Universal Data Exchange

Format: XML

A recommendation from the W3C

- XML = data
- XML generated by applications
- XML consumed by applications
- Easy access: across platforms, organizations



Paradigm Shift on the Web

- From documents (HTML) to data (XML)
- From information retrieval to data management
- For databases, also a paradigm shift:
 - from relational model to semistructured data
 - from data processing to data/query translation
 - from storage to transport

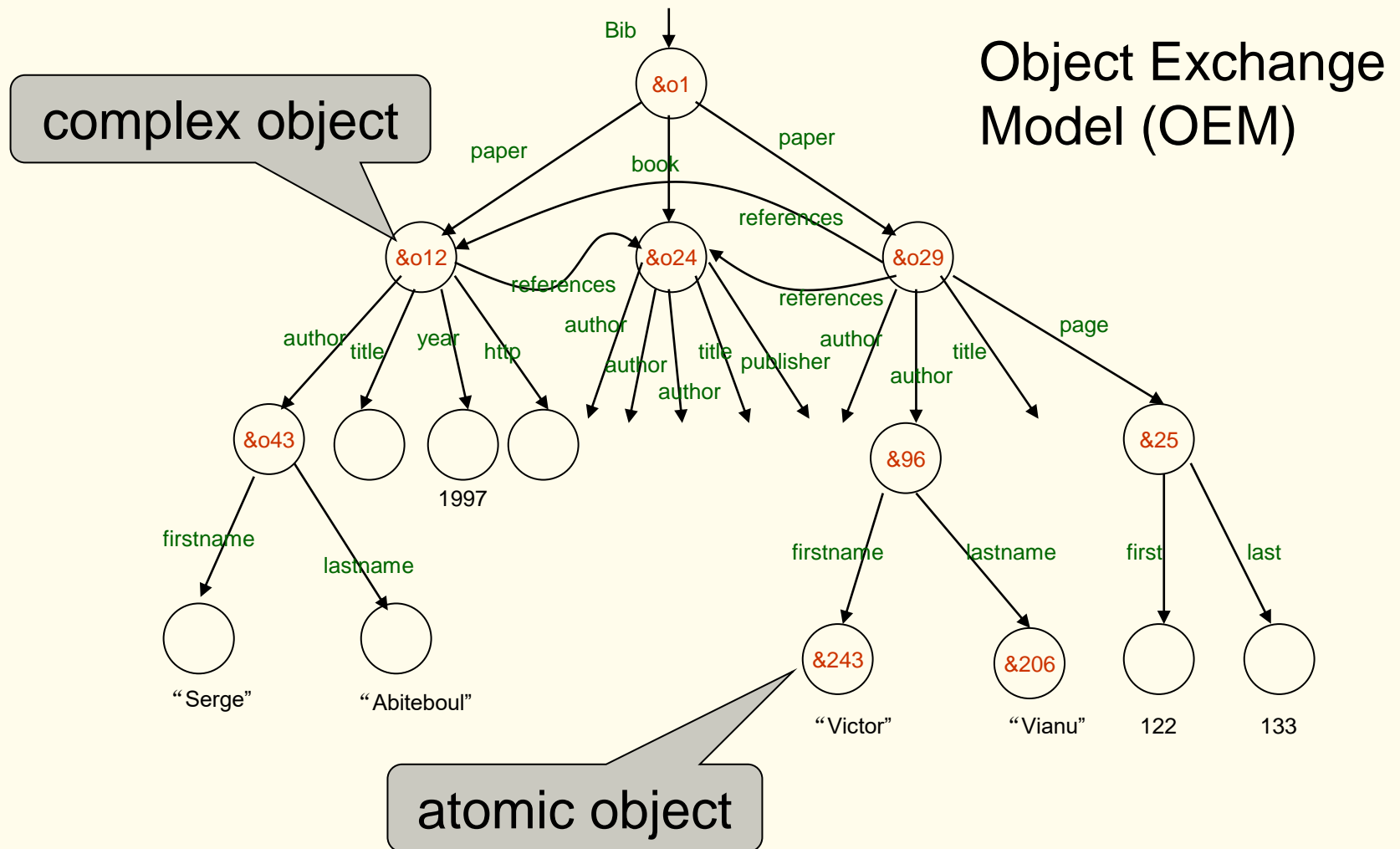


Semistructured Data

Origins:

- Integration of heterogeneous sources
- Data sources with non-rigid structure
 - Biological data
 - *Web data*

The Semistructured Data Model





Syntax for Semistructured Data

```
Bib: { paper: { ... },  
      book: { ... },  
      paper: {  
        author: "Abiteboul",  
        author: { firstname: "Victor",  
                  lastname: "Vianu"},  
        title: "Regular path queries with constraints",  
        references: {  
          references: {  
            pages: { first: 122, last: 133 }  
          }  
        }  
      }
```

Observe: Nested tuples, set-values, oids !



Syntax for Semistructured Data

May omit oids:

```
{ paper: { author: "Abiteboul",  
            author: { firstname: "Victor",  
                      lastname: "Vianu"},  
            title: "Regular path queries ...",  
            page: { first: 122, last: 133 }  
          }  
}
```



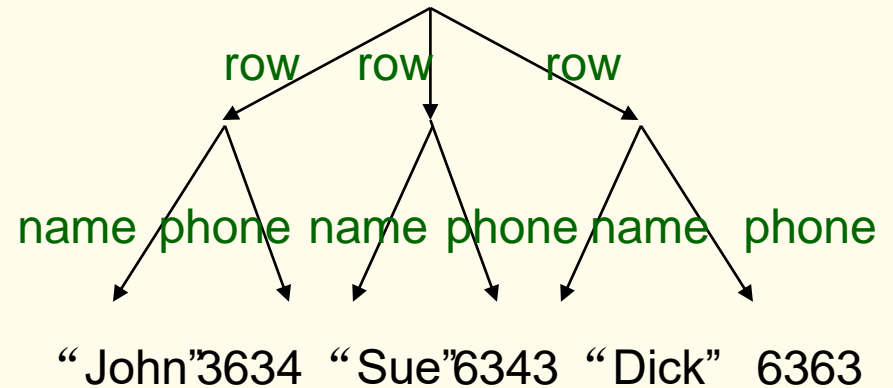
Characteristics of Semistructured Data

- Missing or additional attributes
- Multiple attributes
- Different types in different objects
- Heterogeneous collections

Self-describing, irregular data, no a priori structure

Comparison with Relational Data

name	phone
John	3634
Sue	6343
Dick	6363



```
{ row: { name: "John", phone: 3634 },  
  row: { name: "Sue", phone: 6343 },  
  row: { name: "Dick", phone: 6363 }  
}
```



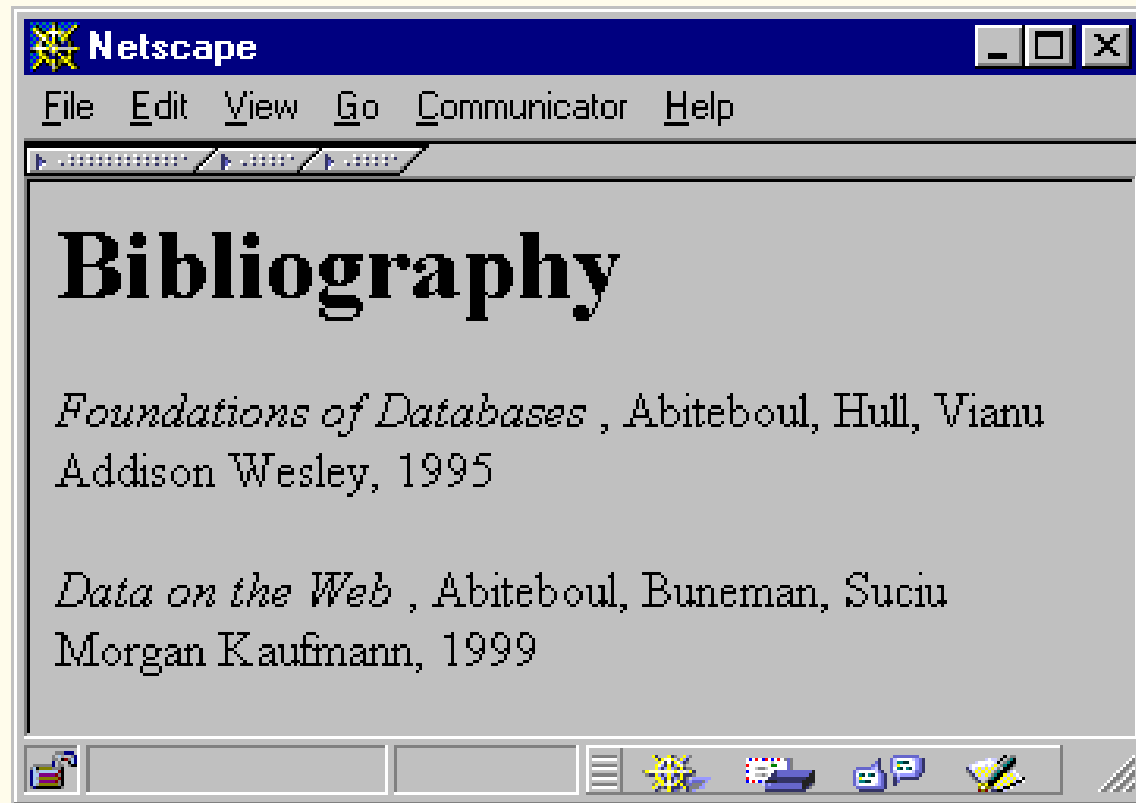

XML

- A W3C standard to complement HTML
- Origins: Structured text SGML
 - Large-scale electronic publishing
 - Data exchange on the web
- Motivation:
 - HTML describes presentation
 - XML describes content
- <http://www.w3.org/TR/2000/REC-xml-20001006> (version 2, 10/2000)

$$\text{HTML4.0} \in \text{XML} \subset \text{SGML}$$



From HTML to XML



HTML describes the presentation



HTML

`<h1> Bibliography </h1>`

`<p> <i> Foundations of Databases </i>`

Abiteboul, Hull, Vianu

`
` Addison Wesley, 1995

`<p> <i> Data on the Web </i>`

Abiteboul, Buneman, Suciu

`
` Morgan Kaufmann, 1999



XML

```
<bibliography>
  <book>  <title> Foundations... </title>
           <author> Abiteboul </author>
           <author> Hull </author>
           <author> Vianu </author>
           <publisher> Addison Wesley </publisher>
           <year> 1995 </year>
  </book>
  ...
</bibliography>
```

XML describes the content



Why are we DB'ers interested?

- It's data, stupid. That's us.
- Proof by Google:
 - database+XML – 1,940,000 pages.
- Database issues:
 - How are we going to model XML? (graphs)
 - How are we going to query XML? (XQuery)
 - How are we going to store XML (in a relational database? object-oriented? native?)
 - How are we going to process XML efficiently? (many interesting research questions !)



Document Type Descriptors

- Sort of like a schema but not really.

```
<!ELEMENT Book (title, author*) >
```

```
<!ELEMENT title #PCDATA>
```

```
<!ELEMENT author (name, address, age?)>
```

```
<!ATTLIST Book id ID #REQUIRED>
```

```
<!ATTLIST Book pub IDREF #IMPLIED>
```

- Inherited from SGML DTD standard
- BNF grammar establishing constraints on element structure and content
- Definitions of entities



Shortcomings of DTDs

Useful for documents, but not so good for data:

- Element name and type are associated globally
- No support for structural re-use
 - Object-oriented-like structures aren't supported
- No support for data types
 - Can't do data validation
- Can have a *single* key item (ID), but:
 - No support for multi-attribute keys
 - No support for foreign keys (references to other keys)
 - No constraints on IDREFs (reference *only* a Section)



XML Schema

- In XML format
- Element names and types associated locally
- Includes primitive data types (integers, strings, dates, etc.)
- Supports value-based constraints (integers > 100)
- User-definable structured types
- Inheritance (extension or restriction)
- Foreign keys
- Element-type reference constraints



Sample XML Schema

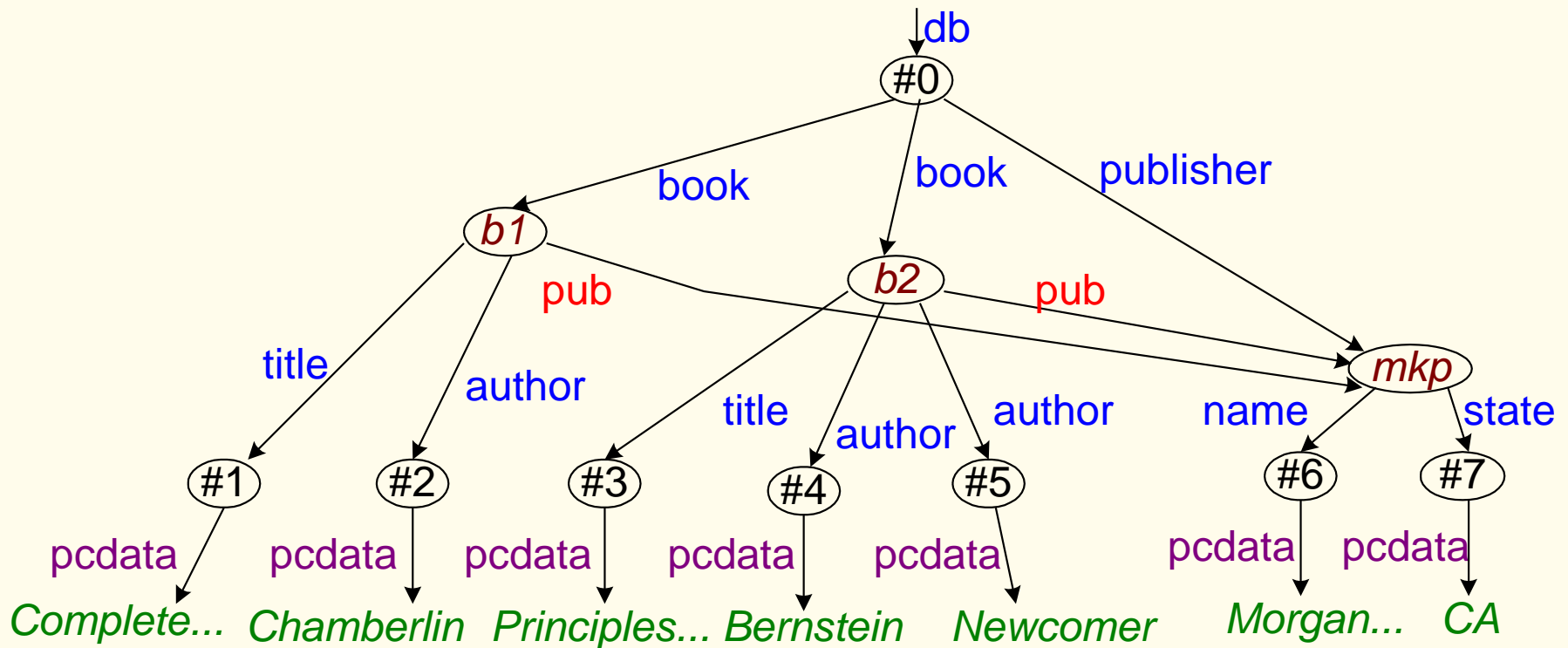
```
<schema version="1.0" xmlns="http://www.w3.org/1999/XMLSchema">
  <element name="author" type="string" />
  <element name="date" type = "date" />
  <element name="abstract">
    <type>
      ...
    </type>
  </element>
  <element name="paper">
    <type>
      <attribute name="keywords" type="string"/>
      <element ref="author" minOccurs="0" maxOccurs="*" />
      <element ref="date" />
      <element ref="abstract" minOccurs="0" maxOccurs="1" />
      <element ref="body" />
    </type>
  </element>
</schema>
```



Important XML Standards

- XSL/XSLT: presentation and transformation standards
- RDF: resource description framework (meta-info such as ratings, categorizations, etc.)
- Xpath/Xpointer/Xlink: standard for linking to documents and elements within
- Namespaces: for resolving name clashes
- DOM: Document Object Model for manipulating XML documents
- SAX: Simple API for XML parsing
- XQuery: query language

XML Data Model (Graph)



Issues:

- Distinguish between attributes and sub-elements?
- Should we conserve order?



XML Terminology

- **Tags:** book, title, author, ...
 - start tag: <book>, end tag: </book>
- **Elements:**
<book>...<book>,<author>...</author>
 - elements can be nested
 - empty element: <red></red> (Can be abbrev. <red/>)
- **XML document:** Has a single root element
- **Well-formed XML document:** Has matching tags
- **Valid XML document:** conforms to a schema



More XML: Attributes

```
<book price = "55" currency = "USD">  
  <title> Foundations of Databases </title>  
  <author> Abiteboul </author>  
  ...  
  <year> 1995 </year>  
</book>
```

Attributes are alternative ways to represent data



More XML: Oids and References

```
<person id="o555"> <name> Jane </name> </person>
```

```
<person id="o456"> <name> Mary </name>  
    <children idref="o123 o555" />  
</person>
```

```
<person id="o123" mother="o456"><name>John</name>  
</person>
```

oids and references in XML are just syntax



XML-Query Data Model

- Describes XML data as a tree
- **Node** ::= DocNode |
ElemNode |
ValueNode |
AttrNode |
NSNode |
PINode |
CommentNode |
InfoItemNode |
RefNode

<http://www.w3.org/TR/query-datamodel/2/2001>



XML-Query Data Model

Element node (simplified definition):

- $\text{elemNode} : (\text{QNameValue}, \{ \text{AttrNode} \}, [\text{ElemNode} \mid \text{ValueNode}])$
→ ElemNode
- QNameValue = means “a tag name”

Reads: “Give me a tag, a set of attributes, a list of elements / values, and I will return an element”



XML Query Data Model

Example:

```
<book price = "55"  
      currency = "USD">  
  <title> Foundations ... </title>  
  <author> Abiteboul </author>  
  <author> Hull </author>  
  <author> Vianu </author>  
  <year> 1995 </year>  
</book>
```

```
Book1 = elemNode(book,  
  {price2, currency3},  
  [title4,  
   author5,  
   author6,  
   author7,  
   year8])
```

```
price2 = attrNode(...) /* next */  
currency3 = attrNode(...)  
title4 = elemNode(title, string9)  
...
```



XML Query Data Model

Attribute node:

- $\text{attrNode} : (\text{QNameValue}, \text{ValueNode}) \rightarrow \text{AttrNode}$



XML Query Data Model

Example:

```
<book price = "55"  
      currency = "USD">  
  <title> Foundations ... </title>  
  <author> Abiteboul </author>  
  <author> Hull </author>  
  <author> Vianu </author>  
  <year> 1995 </year>  
</book>
```

```
price2 = attrNode(price,string10)  
string10 = valueNode(...) /* next */  
currency3 = attrNode(currency,  
                    string11)  
string11 = valueNode(...)
```



XML Query Data Model

Value node:

- ValueNode = StringValue | BoolValue | FloatValue ...
- stringValue : string \rightarrow StringValue
- boolValue : boolean \rightarrow BoolValue
- floatValue : float \rightarrow FloatValue



XML Query Data Model

Example:

```
<book price = "55"  
      currency = "USD">  
  <title> Foundations ... </title>  
  <author> Abiteboul </author>  
  <author> Hull </author>  
  <author> Vianu </author>  
  <year> 1995 </year>  
</book>
```


```
price2 = attrNode(price,string10)  
string10 = valueNode(stringValue("55"))  
currency3 = attrNode(currency, string11)  
string11 = valueNode(stringValue("USD"))  
  
title4 = elemNode(title, string9)  
string9 =  
valueNode(stringValue("Foundations..."))
```



XML vs. Semistructured Data

- Both described best by a graph
- Both are schema-less, self-describing
- XML is ordered, ssd is not
- XML can mix text and elements:

```
<talk> Making Java easier to type and easier to type  
    <speaker> Phil Wadler </speaker>  
</talk>
```
- XML has lots of other stuff: attributes, entities, processing instructions, comments



Management of XML and Semistructured Data

Based upon slides by Dan Suciu



Path Expressions

Examples:

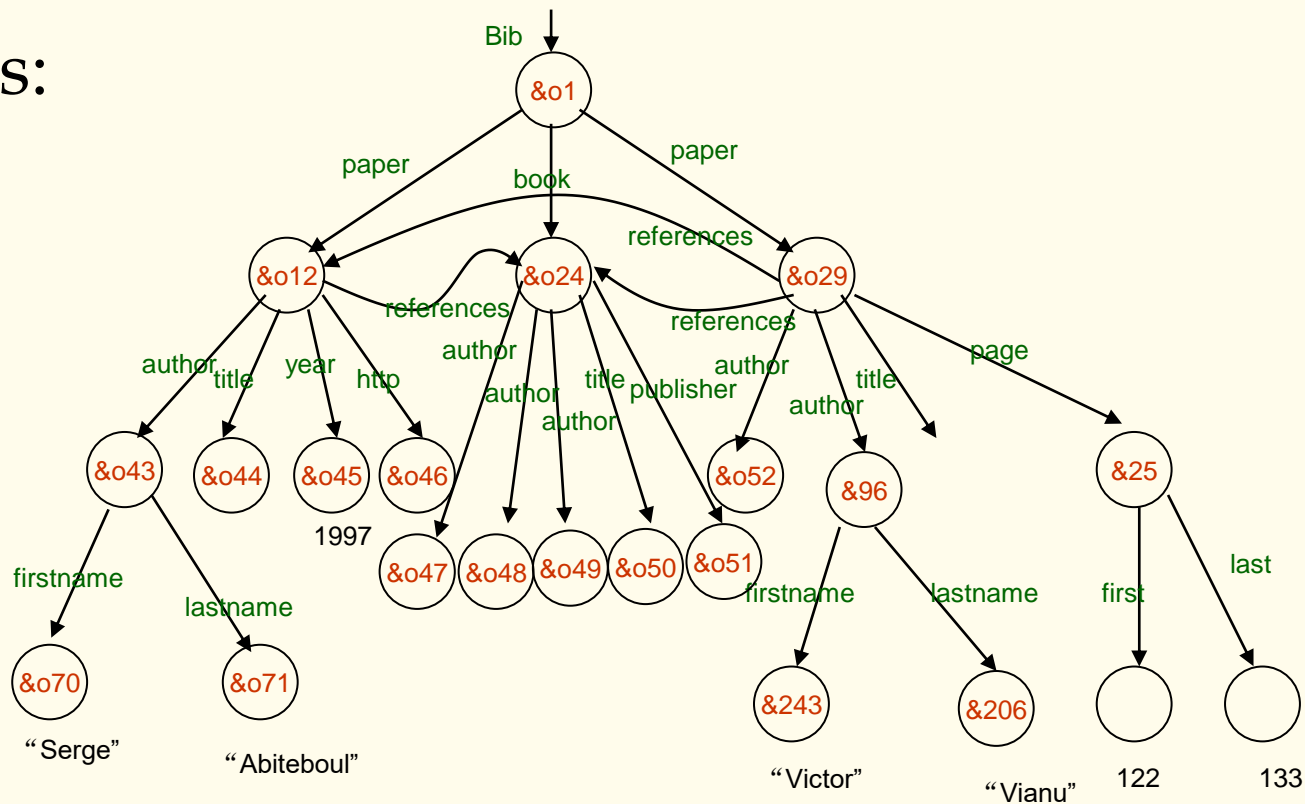
- Bib.paper
- Bib.book.publisher
- Bib.paper.author.lastname

Given an OEM instance, the *value* of a path expression p is a set of objects

Path Expressions

Examples:

DB =



Bib.paper={ &o12,&o29 }

Bib.book.publisher={ &o51 }

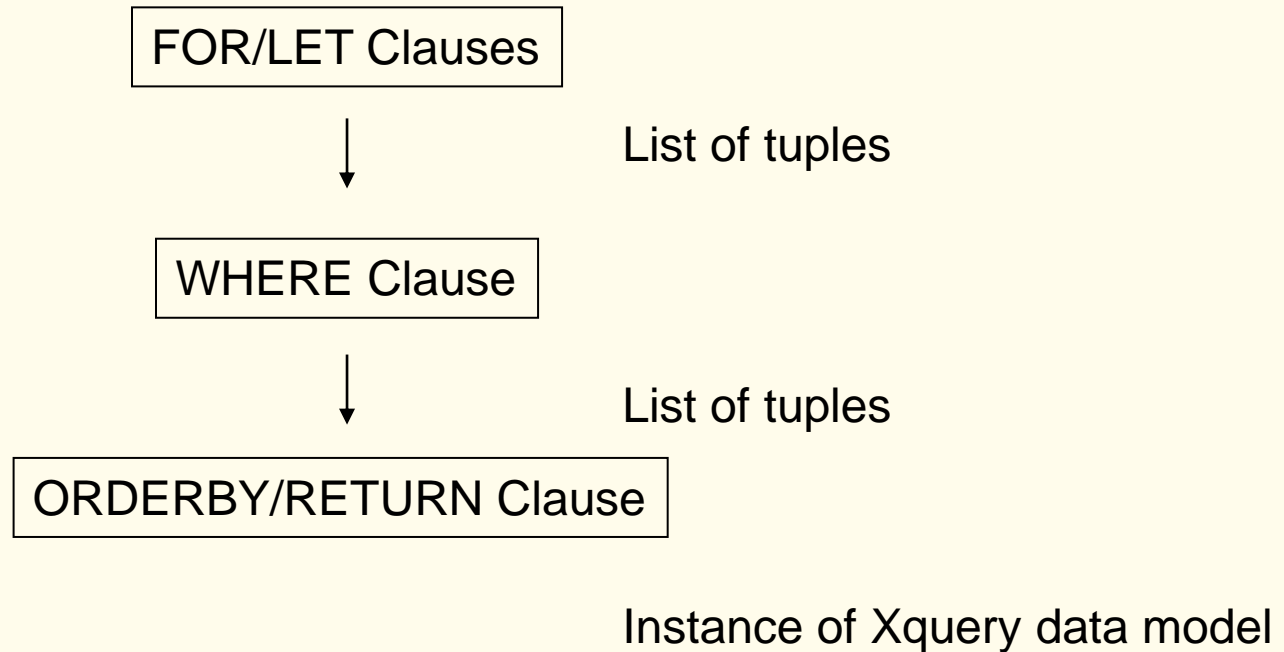
Bib.paper.author.lastname={ &o71,&206 }



XQuery

Summary:

- FOR-LET-WHERE-ORDERBY-RETURN = FLWOR





XQuery

- FOR $\$x$ in expr -- binds $\$x$ to each value in the list expr
- LET $\$x$ = expr -- binds $\$x$ to the entire list expr
 - Useful for common subexpressions and for aggregations



FOR v.s. LET

```
FOR $x IN document("bib.xml")/bib/book  
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book></result>  
<result> <book>...</book></result>  
<result> <book>...</book></result>  
...
```

```
LET $x IN document("bib.xml")/bib/book  
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book>  
        <book>...</book>  
        <book>...</book>  
        ...  
</result>
```



Path Expressions

- Abbreviated Syntax
 - `/bib/paper[2]/author[1]`
 - `/bib//author`
 - `paper[author/lastname="Vianu"]`
 - `/bib/(paper|book)/title`
- Unabbreviated Syntax
 - `child::bib/descendant::author`
 - `child::bib/descendant-or-self::* / child::author`
 - `parent, self, descendant-or-self, attribute`



XQuery

Find all book titles published after 1995:

```
FOR $x IN document("bib.xml")/bib/book  
WHERE $x/year > 1995  
RETURN $x/title
```

Result:

```
<title> abc </title>  
<title> def </title>  
<title> ghi </title>
```



XQuery

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")  
                    /bib/book[publisher="Morgan Kaufmann"]/author)  
RETURN <result>  
    $a,  
    FOR $t IN /bib/book[author=$a]/title  
    RETURN $t  
</result>
```

distinct = a function that eliminates duplicates



XQuery

Result:

```
<result>
  <author>Jones</author>
  <title> abc </title>
  <title> def </title>
</result>
<result>
  <author> Smith </author>
  <title> ghi </title>
</result>
```




XQuery

```
<big_publishers>  
  FOR $p IN distinct(document("bib.xml")//publisher)  
  LET $b := document("bib.xml")/book[publisher = $p]  
  WHERE count($b) > 100  
  RETURN $p  
</big_publishers>
```

count = a (aggregate) function that returns the number of elms



XQuery

Find books whose price is larger than average:

```
LET $a=avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN $b
```



FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value



Collections in XQuery

- Ordered and unordered collections
 - `/bib/book/author` = an ordered collection
 - `Distinct(/bib/book/author)` = an unordered collection
- LET `$a` = `/bib/book` → `$a` is a collection
- `$b/author` → a collection (several authors...)

```
RETURN <result> $b/author </result>
```

Returns:

```
<result> <author>...</author>
          <author>...</author>
          <author>...</author>
          ...
</result>
```



Collections in XQuery

What about collections in expressions ?

- $\$b/\text{price}$ → list of n prices
- $\$b/\text{price} * 0.7$ → list of n numbers??
- $\$b/\text{price} * \$b/\text{quantity}$ → list of n*m numbers ??
 - Valid only if the two sequences have at most one element
 - **Atomization**
- $\$book1/\text{author eq "Kennedy"}$ - Value Comparison
- $\$book1/\text{author} = \text{"Kennedy"}$ - General Comparison



Sorting in XQuery

```
<publisher_list>
  FOR $p IN distinct(document("bib.xml")//publisher)
  ORDERBY $p
  RETURN <publisher> <name> $p/text() </name> ,
    FOR $b IN document("bib.xml")//book[publisher = $p]
    ORDERBY $b/price DESCENDING
    RETURN <book>
      $b/title ,
      $b/price
    </book>
  </publisher>
</publisher_list>
```



If-Then-Else

```
FOR $h IN //holding  
ORDERBY $h/title  
RETURN <holding>
```

```
$h/title,
```

```
IF $h/@type = "Journal"
```

```
THEN $h/editor
```

```
ELSE $h/author
```

```
</holding>
```



Existential Quantifiers

```
FOR $b IN //book  
WHERE SOME $p IN $b//para SATISFIES  
    contains($p, "sailing")  
    AND contains($p, "windsurfing")  
RETURN $b/title
```




Universal Quantifiers

```
FOR $b IN //book  
WHERE EVERY $p IN $b//para SATISFIES  
    contains($p, "sailing")  
RETURN $b/title
```



Other Stuff in XQuery

- If-then-else
- Universal and existential quantifiers
- Sorting
- Before and After
 - for dealing with order in the input
- Filter
 - deletes some edges in the result tree
- Recursive functions



Group-By in Xquery ??

- No GROUPBY currently in XQuery
- A recent proposal (next)
 - What do YOU think ?



Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,  
    $y IN $b/@year  
WHERE $b/publisher="Morgan Kaufmann"  
RETURN   GROUPBY $y  
           WHERE count($b) > 10  
           IN <year> $y </year>
```

← with GROUPBY

Equivalent SQL →

```
SELECT year  
FROM Bib  
WHERE Bib.publisher="Morgan Kaufmann"  
GROUPBY year  
HAVING count(*) > 10
```

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,  
    $a IN $b/author,  
    $y IN $b/@year  
RETURN GROUPBY $a, $y  
    IN <result> $a,  
        <year> $y </year>,  
        <total> count($b) </total>  
    </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN document("http://www.bn.com")/ bib/book/author,  
    $y IN $a/./@year  
LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]  
RETURN <result> $a,  
    <year> $y </year>,  
    <total> count($b) </total>  
    </result>
```

Correct if the GROUPBY is node-identity based
Not equivalent if the GROUPBY is value-based

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,  
  $a IN $b/author,  
  $y IN $b/@year  
RETURN GROUPBY $a, $y  
  IN <result> $a,  
    <year> $y </year>,  
    <total> count($b) </total>  
  </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN distinct(document("http://www.bn.com")/ bib/book/author)  
  $y IN distinct(document("http://www.bn.com")/bib/book/@year)  
LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]  
RETURN  
  IF count($b) > 0  
  THEN  
    <result> $a,  
      <year> $y </year>,  
      <total> count($b) </total>  
    </result>
```

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,  
    $a IN $b/author,  
    $y IN $b/@year  
RETURN GROUPBY $a, $y  
    IN <result> $a,  
        <year> $y </year>,  
        <total> count($b) </total>  
    </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $Tup IN distinct (FOR $b IN document("http://www.bn.com")/bib,  
    $a IN $b/author,  
    $y IN $b/@year  
    RETURN <Tup> <a> $a </a> <y> $y </y> </Tup>),  
    $a IN $Tup/a/node(),  
    $y IN $Tup/y/node()  
LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]  
RETURN <result> $a,  
    <year> $y </year>,  
    <total> count($b) </total>  
    </result>
```



Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,  
    $a IN $b/author,  
    $y IN $b/@year,  
    $t IN $b/title,  
    $p IN $b/publisher  
RETURN  
    GROUPBY $p, $y  
    IN <result> $p,  
        <year> $y </year>,  
        GROUPBY $a  
        IN <authorEntry>  
            $a,  
            GROUPBY $t  
            IN $t  
            <authorEntry>  
        </result>
```

← Nested GROUPBY's