Chapter 6
DATABASES, DATA WAREHOUSES AND OLAP

Cios / Pedrycz / Swiniarski / Kurgan

Outline

- Introduction
- Database Management Systems and SQL
  - Architecture of Database Management Systems
  - Introduction to SQL
  - Data Retrieval with SQL
    - Select Command
    - Aggregate Functions
    - View Command
    - Insert Command
    - Update Command
    - Delete Command
    - Finalizing the Changes to the Database
  - Query Optimization
- Data Warehouses
- Data Warehouses vs. RDMS

Outline

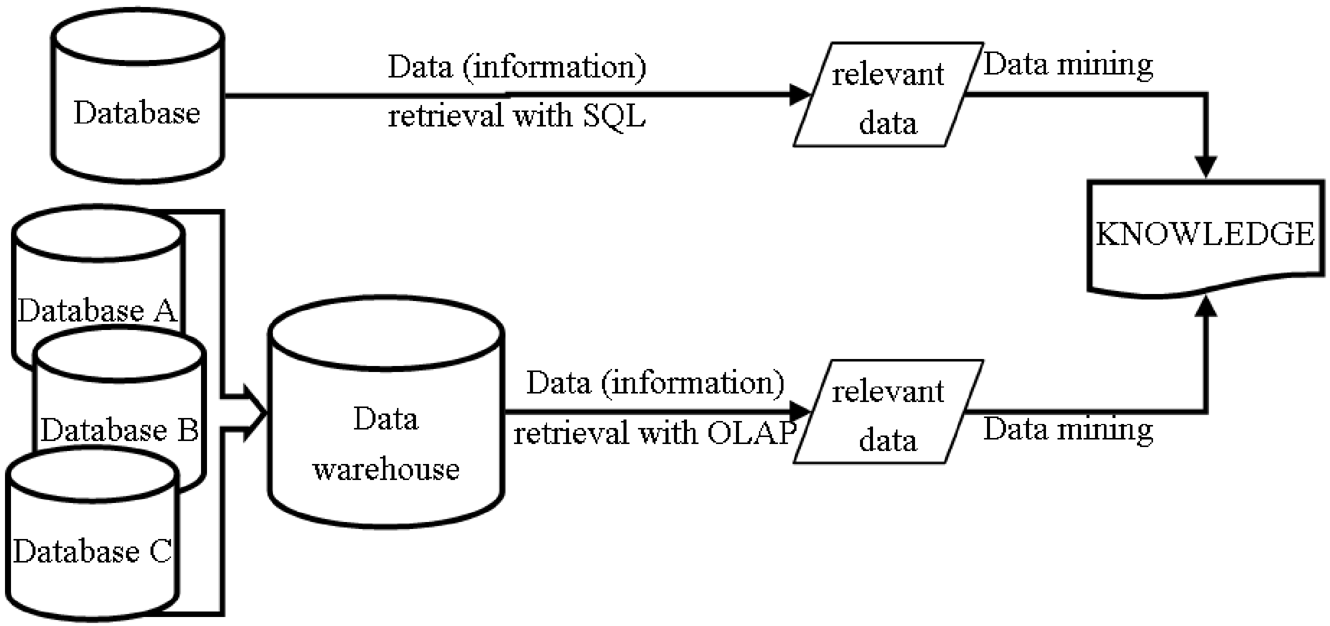
- **Virtual Data Warehouses, Data Marts and Enterprise Data Warehouses**
- **Architecture Of Data Warehouses**
  - **Star, Snowflake and Galaxy Schemas**
  - **Concept Hierarchy**
- **Multidimensional Data Models and Data Cubes**
- **On-Line Analytical Processing (OLAP)**
- **Data Retrieval with OLAP**
- **OLAP Server Architectures**
- **Efficiency of OLAP**
- **FASMI Test**
- **Example OLAP Tools**
- **Data Warehouses and OLAP for Data Mining**

Introduction

Databases and data warehouses provide an efficient data retrieval and summarization capabilities, necessary to prepare and select data for the subsequent steps of the knowledge discovery process.

Introduction

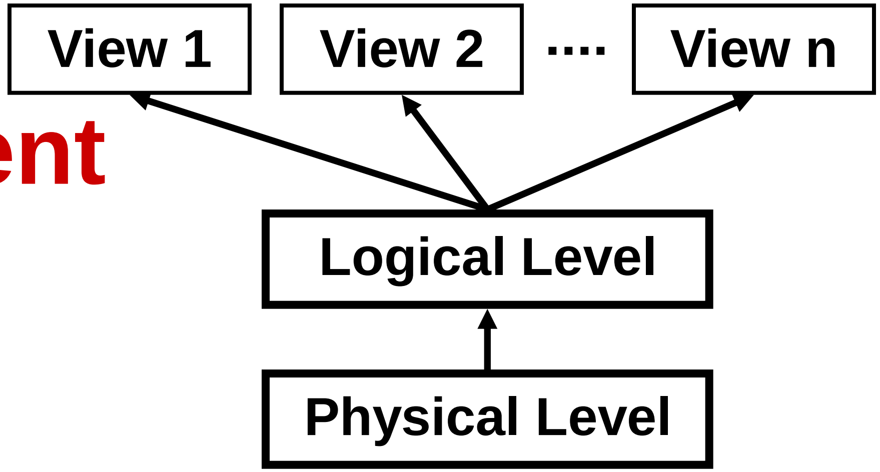
Relation between databases/data warehouses and Data Mining.



Database Management Systems

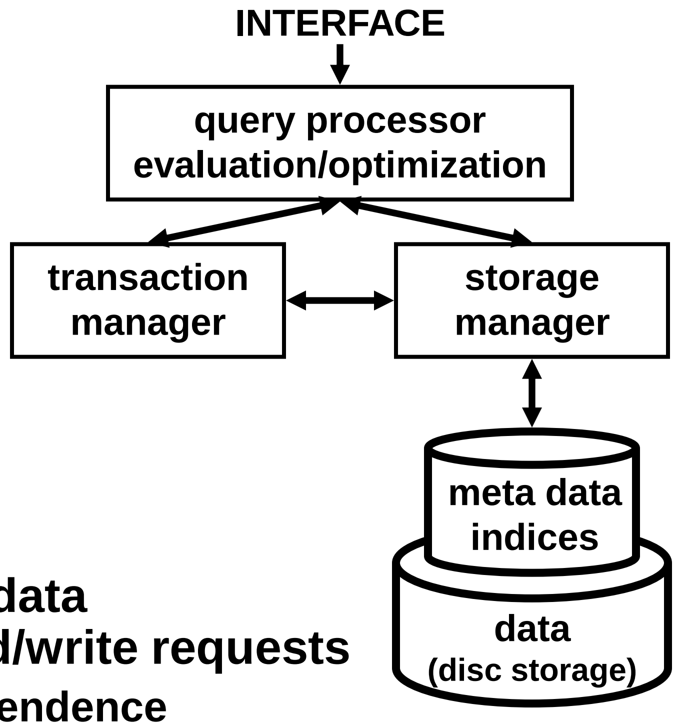
- **Collection of interrelated data and a set of programs to access those data**
  - **the primary goal is to provide an environment that is both convenient and efficient to use in retrieving and storing data**
  - **they also provide design, update, and maintenance capabilities**
- **We assume that such system contains information about a single enterprise**

Database Management Systems



- Three layer structure
  - view level
    - the part of the database that is interesting to the user
    - usually is it an extract consisting of a selected part of the data stored in the DBMS
  - logical level
    - describes what data is stored in the database, and that relationships exists among these data
  - physical level
    - describes how the actual data is stored
- Both, the physical and logical schema can be modified without the need to rewrite the entire DBMS application

Database Management Systems



- **Architecture**
  - **query processor**
    - **handles translation of queries or data manipulation statements into read/write requests**
      - **necessary because of data independence**
        i.e. queries are written in a language which hides the details of the storage representation of the data
    - **query optimization handles deciding on the best (most efficient) strategy for extracting the data needed to handle a particular query**
  - **storage manager**
    - **handles disk space allocation, read/write operations, buffer and cache management, etc.**
  - **transaction manager**
    - **handles issues related to concurrent multi-user access, and issues related to system failures**

Data Retrieval in DBMS

- **To retrieve and manipulate data, DBMS uses the following three types of languages:**
  - **Data Manipulation Language (DML) that retrieves or modifies data**
  - **Data Definition Language (DDL) that defines the structures of the data**
    - **i.e. statements that create, alter, or remove database objects**
  - **Data Control Language (DCL) that defines the privileges granted to database users**
    - **DDL and DCL are used only by a DBA (Database Administrator) or by the privileged user**
    - **DML is used by regular users**
- **all three of them are handled by SQL**

SQL

- Structured Query Language (SQL) allows users of relational DBMS to access and manipulate data, and to manipulate the database
  - examples include Oracle, Sybase, Informix, MS SQL Server, MS Access, and many others
  - it is a powerful, nonprocedural language
    - unlike other languages like C, Pascal, etc., it does not have control flow constructs (e.g. if-then-else, do-while), and function definitions
    - it has fixed set of data types, i.e. user cannot create own data types as it is possible with other languages
  - despite these limitations, it became a standard to perform data retrieval operations
    - other languages have extensions that enable using SQL

SQL

- **SQL programs consist of the following 5 steps:**
  1. **defining schema for each relation using SQL DDL**
     - **used to create and manage database objects**
       - **includes creation of tables and keys, which describe relationships between tables**
     - **example commands include: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, and DROP INDEX**
  2. **defining privileges for users using SQL DCL**
     - **used to create objects related to user access and privileges**
       - **includes giving and revoking permissions to see and alter data**
     - **example commands include: ALTER PASSWORD, GRANT, REVOKE, and CREATE SYNONYM**

SQL

3. populate the database by inserting tuples
   - used to populate the database with initial data
     - includes insertions of data into the created tables
   - example commands include: SELECT, and INSERT
4. writing SQL queries
   - used to perform various operations on the existing database
     - includes inserting new tuples, modifying existing tuples, creating views, updating privileges, etc
   - example commands include: SELECT, INSERT, UPDATE, DELETE, and VIEW
5. executing the queries
   - once the database is created and initially populated, new SQL statements are prepared and executed
     - this most often happens online, i.e. they are executed while the DBMS is running

SQL

- To write SQL queries we
  - specify attributes that will be retrieved in the SELECT clause
  - specify all tables (relations) that are involved/used in the FROM clause
  - specify conditions that constrain the desired operations (e.g. join, select, subtract) in the WHERE clause
  - words of wisdom
    - be aware that the same attributes may appear in different relations (tables) under different names
    - although SQL is case insensitive, you should be cautious when retrieving the contents of a field, since the stored data may be case sensitive
    - every SQL statement must be terminated by a single semicolon, even if it is extended over many lines

SQL

- The most popular DML statements are
  - SELECT, which is used to scan content of tables
    - it cannot create or modify neither the content, nor the table
  - VIEW, which is used to create a new database view
    - view is a new table used for example to help design complex queries (it is a soft filter, which is not physically created)
  - INSERT, which is used to insert new data into a table
  - UPDATE, which is used to modify existing data in a table
    - but not to remove or add new records
  - DELETE, which is used to remove a tuple from a table
- following, they are described in more details
  - they constitute core statements for data retrieval task

SQL

- **Example schema**

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

SELECT

- **Syntax**

SELECT	**[* | all | distinct] column1, column2, ...**
FROM	**table1 [, table2, ...]**
[WHERE	**condition1 | expression1]**
[AND	**condition2 | expression2]**
[GROUP BY	**column1, column2, ...]**
[HAVING	**conditions1 | expression1]**
[ORDER BY	**column1 | integer1, column2 | integer2, ... [ASC | DESC]]**

  - **[ ] define optional conditions**
  - **keywords are denoted by blue letters**

SELECT

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

- **some rules**
  - **it must contain the SELECT list (i.e. a list of columns or expressions to be retrieved) and the FROM clause (i.e. the table(s) from which to retrieve the data)**
    - **distinct keyword is used to prevent duplicate rows being returned**
    - **WHERE clause is used to filter out records that we are interested in**

- **example 1**

find all account numbers (and their balances) with loan balances bigger than 1000

```
SELECT      AccountNumber, Balance
FROM        Account
WHERE       Balance > 1000
AND         AccountType = ‘loan’
ORDER BY    Balance DESC;
```

AccountNumber	Balance
1000003	5000
1000006	1300

SELECT

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

- **example 2 (join between two tables)**
  find all customers who have both a loan and another account type

```
SELECT     distinct CustomerName
FROM       Own, Borrow
WHERE      Own.CustomerName = Borrow.CustomerName
ORDER BY   CustomerName;
```

CustomerName
Joe Dalton
Will Smith

- **example 3 (join with aliases between three tables)**
  find all customers, and their account types, who have both a loan and other type of account; rename corresponding columns as Name and Type

```
SELECT   distinct O.CustomerName Name, A.AccountType Type
FROM     Account A, Borrow B, Own O
WHERE    O.CustomerName = B.CustomerName
AND      (O.AccountNumber = A.AccountNumber OR
         B.AccountNumber = A.AccountNumber)
ORDER BY CustomerName;
```

Name	Type
Joe Dalton	saving
Joe Dalton	checking
Joe Dalton	loan
Will Smith	checking
Will Smith	loan

SELECT

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

– **query from example 2 can be written in several ways**

find all customers who have both a loan and other account

```
SELECT      distinct CustomerName
FROM        Own, Borrow
WHERE       Own.CustomerName = Borrow.CustomerName
ORDER BY    CustomerName;

SELECT      distinct CustomerName
FROM        Borrow
WHERE       CustomerName IN (SELECT CustomerName FROM Own)
ORDER BY    CustomerName;

SELECT      distinct CustomerName
FROM        Borrow
WHERE       EXISTS (SELECT CustomerName FROM Own WHERE
                 Own.CustomerName = Borrow.CustomerName)
ORDER BY    CustomerName;
```

CustomerName
Joe Dalton
Will Smith

SELECT

- **query from example 2 can be written in several ways**
  - **last two examples utilize so called nested queries**
    - **such query utilizes some other query or queries to compute its own result**
- **the redundancy in ability to express a given query in SQL is necessary since not all commercial products support all features of SQL**
  - **it also gives flexibility in designing complex queries**

Aggregate Functions

- **They map a collection of values into a single value**
  - **allow to compute simple statistics of the data, which can be used to make simple decisions**
  - **five aggregate functions are**
    - **avg(x) – average of a collection of numbers x**
    - **sum(x) – sum of a collection of numbers x**
    - **max(x) – max value among a collection of numbers or nonnumeric data x**
    - **min(x) – min value among a collection of numbers or nonnumeric data x**
    - **count(x) – cardinality of a collections of numbers or nonnumeric data x**

Aggregate Functions

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

- example 4 (using aggregate functions)

  find average balance and number of all loans

```
SELECT    avg(Balance) average loan, count(Balance) count of loans
FROM      Account
WHERE     AccountType = 'loan';
```

average loan	count of loans
2168.3	3

- example 5 (using aggregate functions with GROUP BY)
  - GROUP BY allows to compute values for a set of tuples

  find all account types, and their maximum balances
  but only if their average balance is more than 1000

```
SELECT    AccountType, max(Balance)
FROM      Account
GROUP BY  AccountType
HAVING    avg(Balance) > 1000
```

AccountType	max(Balance)
checking	1605
loan	5000

VIEW

- **Syntax**

  CREATE VIEW view [(column_name_list)]

  AS SELECT query

  - **view is the name of a view to be created**
  - **column_name_list is an optional list of names to be used for columns in the view**
    - **if given, these names override the column names that would be deduced from the SQL query**
  - **query**
    - **an SQL query that will provide the columns and rows of the view**
    - **usually given as a SELECT statement**

VIEW

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

– **example**

design a view that lists all customers that have a non loan account together with their account types

```
CREATE VIEW     CustomerAccounts (Name, Type)
AS              SELECT CustomerName, AccountType FROM Own, Account
                WHERE Own.AccountNumber = Account.AccountNumber;
```

CustomerAccounts

Name	Type
Will Smith	checking
Joe Dalton	saving
Joe Dalton	checking

INSERT

- **Syntax**

  INSERT INTO table_name [('column1', 'column2')]
  VALUES ('values1', 'value2', [NULL]);

  - **the SELECT statement can be used with the INSERT statement to insert data into the table based on the results of a query from another table**

    INSERT INTO table_name [('column1', 'colum2')]
    SELECT [* | ('column1', 'column2')]
    FROM table_name
    [WHERE condition(s)];

INSERT

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

- **example**
  add a new saving account for Will Smith with balance of 10000

```
INSERT INTO    Own (AccountNumber, CustomerName)
VALUES         (1000007,’Will Smith’);

INSERT INTO    Account
VALUES         (1000007,’saving’,10000);
```

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300
1000007	saving	10000

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004
Will Smith	1000007

UPDATE

- **Syntax**

```
UPDATE table_name
SET       column1  = 'value',
          [column2 = 'value',]
          [column3 = 'value']
[ WHERE  condition ];
```

  - **the UPDATE statement is usually used with the WHERE clause**
    - **otherwise, all records in the table for the specified column will be updated**

UPDATE

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300
1000007	saving	10000

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004
Will Smith	1000007

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

- **example**

  the new saving account for Will Smith should have balance of 1000 (human error)

```
UPDATE    Account
SET       Balance = 1000
WHERE     AccountNumber = 1000007
```

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300
1000007	saving	1000

DELETE

- **Syntax**

  DELETE FROM table_name

  [WHERE condition];

  - **removes an ENTIRE row of data from the specified table**
  - **as with the UPDATE statement, the DELETE statement is usually used with the WHERE clause**
    - **otherwise, all records in the table will be deleted**

DELETE

Account

AccountNumber	AccountType	Balance
1000001	checking	1605
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300
1000007	saving	1000

Own

CustomerName	AccountNumber
Will Smith	1000001
Joe Dalton	1000002
Joe Dalton	1000004
Will Smith	1000007

Borrow

CustomerName	AccountNumber
Will Smith	1000005
Will Smith	1000006
Joe Dalton	1000003

– **example**

Will Smith has closed his checking account with balance of 1605, and thus this accounts should be removed

- **we carefully select a row from Account table based on information from the Own table**

```
DELETE FROM     Account
WHERE           Account Number =
                 (SELECT Account.AccountNumber FROM Own, Account
                WHERE  Own.AccountNumber = Account.AccountNumber
                       AND  Account.Balance = 1605
                       AND Own.CustomerName = 'Will Smith');

DELETE FROM     Own
WHERE           CustomerName = 'Will Smith' AND AccountName = 1000001;
```

Account

AccountNumber	AccountType	Balance
1000002	saving	1000
1000003	loan	5000
1000004	checking	1216
1000005	loan	205
1000006	loan	1300
1000007	saving	1000

Own

CustomerName	AccountNumber
Joe Dalton	1000002
Joe Dalton	1000004
Will Smith	1000007

SQL

- **When using DML statements, such as INSERT, UPDATE and DELETE, the changes are finalized by using the following commands:**
  - **COMMIT, which makes the changes permanent**
  - **ROLLBACK, which undoes current transaction**
    - **transaction is understood as the last block of SQL statements**
  - **SAVEPOINT, which marks and names current point in processing a transaction**
    - **lets undo part of a transaction instead of the whole transaction**
  - **example**

```
DELETE FROM     Account
WHERE           AccountNumber = 1000002;
 1 row deleted
COMMIT;
 commit completed, i.e. state of the database was physically updated
```

Query Optimization

- **given a query, the DBMS interprets it and plans a strategy for carrying it out**
  - **user writes a query, the DBMS is responsible for evaluating it in the most efficient way**
    - **for all but the simplest queries there are several ways of execution with total processing costs that can vary even by several orders of magnitude**

Query Optimization

Steps

1. **Parsing**
   - **query if broke up into individual words, called tokens, and the query processor makes sure that query contains valid verb and legal clauses, i.e. syntax errors and misspellings are detected.**
2. **Validation**
   - **query is checked against the schema to verify that all tables named in the query exist in the database, all columns exist and their names are unambiguous, and if the user has the required privileges**

Query Optimization

Steps

3. Optimization
   - query processor explores various ways to carry out the query.
     - for instance, it may choose between first applying a condition to a table A and then merging it with table B, or first merging the two tables and then applying the condition
   - optimization aims to use predefined indices to speedup searching for data, and to avoid sequential searches through entire tables by first reducing them though applying conditions
   - after exploring alternatives, the optimal sequence of actions is chosen.

Query Optimization

Steps

4. **Execution plan preparation**
   - an execution plan for the query is generated
   - it includes generation of an "executable code" that translates the query into a sequence of low-level operations, such as read/write.
5. **Execution**
   - the query is executed according to the prepared execution plan

the cost of query evaluation can be computed in

- # of disc accesses
- CPU time to execute it
- cost of communication in a distributed system
- etc.

Data Warehouse

Data Warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process

W.H. Inmon

- **following each of these terms is explained**

- **the process of constructing and using data warehouses is called data warehousing**

Data Warehouse

Main features

- a database that is maintained separately from the organization's operational database for the purpose of decision support
  - provides integrated, company-wide, historical data for performing analysis
  - focuses on modeling and analysis of data for decision makers
    - NOT used for daily operations and transaction processing
  - subject-oriented
    - organized around major subjects, like customer, product, sales
      - » provides a simple and concise view around particular subject issues by excluding data that are not useful within the decision support process
      - » focuses on a subject defined by users
      - » contains all data needed by the users to understand the subject

Students

StudentNo	LastName	MiddleInit	FirstName	Status	…
234-99-9989	Doe	W	John	Sr	…
421-12-1121	Smith	A	William	Jr	…

Data Warehouse

Student Employees

StudentID	Address	Status	NoHoursWeek	…
234-99-9989	1001 West 11 St Apt 21	Sr	12	…
421-12-1121	3030 E 42 Ave	Jr	20	…

Student Health

Name	Address	Phone	ID	…
John Doe	1001 W. 11 St # 21	223-4454	234999989	…
William Smith	3030 East 42 Ave	341-9090	421121121	…

Main features

- integrated
  - it integrates multiple, heterogeneous data sources
    - relational databases, flat files, and on-line transaction records
  - during data warehousing, the data cleaning and integration techniques are used
    - main goal is to ensure consistency in naming conventions, attribute types, etc. among different data sources
      - » e.g. see tables above
        each comes from a different source: general DB, employment records, and health records
        inconsistencies in naming: StudentNo, StudentID, and ID
        inconsistencies in values: Address in Employees and in Health

Data Warehouse

Main features

- time-variant
  - data warehouse has much longer time horizon than operational systems
    - operational database keeps only current value data (data snapshot)
    - data warehouse provides information from a historical perspective
      - » e.g., past 5-10 years of data
  - every key in the data warehouse contains a time defining element, either explicitly or implicitly
    - the key from operational data may or may not contain the time defining element

Data Warehouse

Main features

- nonvolatile
  - data warehouse is a physically separate storage of data that is transformed from the operational data
  - the operational updates of data DO NOT occur in a data warehouse
    - NO update, insert, and delete operations
      - » in an operational DB repetition of the same query can give different results, but in a data warehouse they always give the same result
      - » thus there is NO need for transaction processing, recovery, and concurrency control
    - performs only two data accessing operations
      - » initial loading of data
      - » read

Data Warehousing

Why they are feasible

- use relational DBMS technology
  - well studied
  - very good performance
- use recent advances in hardware and software
  - high speed and large storage capacity
  - many end-user computing interfaces and tools
    - used to improve performance and provide user-friendly display if useful information

DBMS and Data Warehouse

- **Traditional DBMS uses OLTP** (on-line transaction processing)
  - used to perform transaction processing
    - transactions are used to read and update data for day-to-day operations
- **Data warehouse uses OLAP** (on-line analytical processing)
  - used to perform data analysis and decision making
    - static copy of data is used to generate useful information in a read-only fashion

feature	OLTP	OLAP
target of the analysis	customer oriented	market oriented
type of data	current and detailed	historical and integrated
type of underlying DB design	ER diagrams	star model
type of access	read and update	read-only
queries	less complex	very complex

DBMS and Data Warehouse

- DBMS
  - tuned for OLTP
    - access methods, indexing, concurrency control, recovery
- Data Warehouse
  - tuned for OLAP
    - complex OLAP queries, multidimensional views involving GROUP BY and aggregative operators
  - requires historical data that is not maintained by DBMS
  - requires integration of data from heterogeneous sources
    - uses reconciled and therefore consistent data representations, codes and formats
  - provides basis for analysis and exploration, that can be used to identify useful trends and create data summaries

DBMS and Data Warehouse

- **Long comparison**

feature	OLTP	OLAP
target users	clerks, IT professionals	decision support workers
# concurrent users	thousands	up to hundreds
goal	day-to-day operations	decision support
designed to provide	application-oriented solution	subject-oriented solution
type of data	current, flat relational, and isolated	historical, multidimensional, integrated, and summarized
unit of work	transaction	complex query
data accessing pattern	frequently	ad-hoc
type of access	read and update, indexing	read-only
# accessed records / work unit	tens	up to millions
size	MB to GB	MB to TB

Why not Heterogeneous DBMS?

- **Heterogeneous DB are integrated by building wrappers/mediators**
  - **use query driven approach**
    - **require complex information filtering, and thus are computationally expensive**
      - **when querying a client database, a meta-dictionary is used to translate the query into queries appropriate for individual heterogeneous databases involved**
      - **the returned results are integrated into a global answer**
- **Data warehouse**
  - **information from heterogeneous sources is integrated and stored in a warehouse for direct query and analysis**
    - **very high performance**
      - **possibility of precomputing frequently executed queries**

Data Warehouse

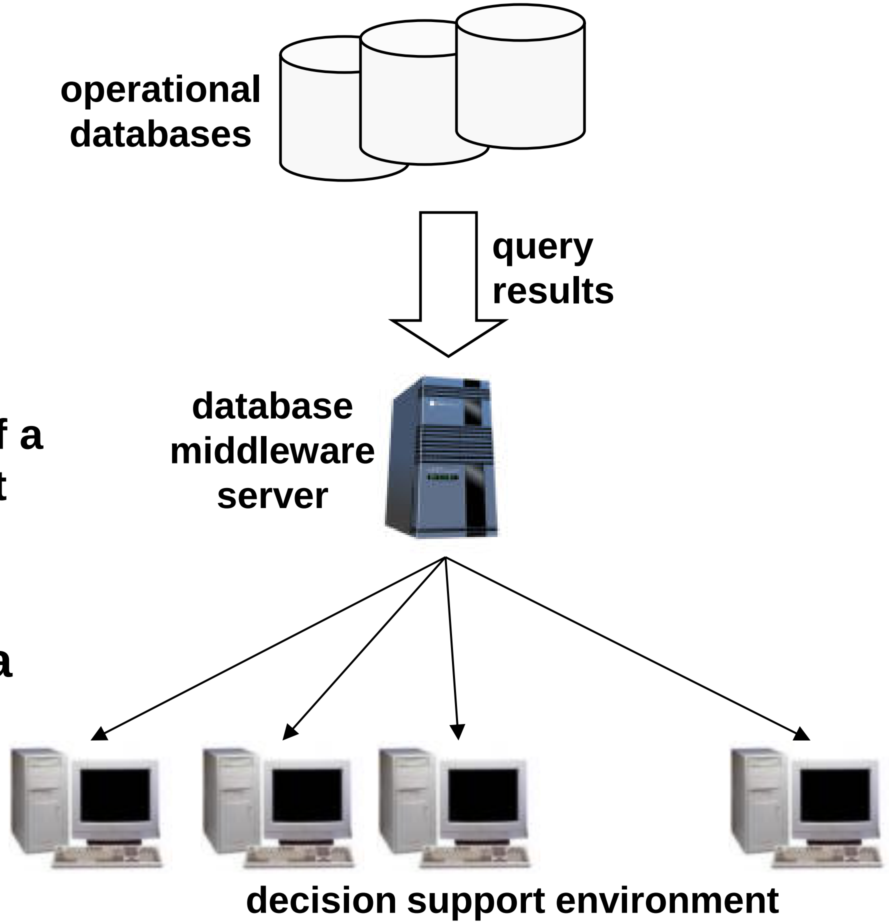
Three models

- enterprise warehouse
  - holds all information about subjects spanning the entire company
    - may take several years to design and build
- data mart
  - a subset of the company-wide data that is of value to a small group of users
    - scope is confined to a specific groups of users, like marketing or customer service
    - can be a precursor or a successor of the actual data warehouse
- virtual warehouse
  - a set of views over standard operational databases
    - only some views may be materialized because of the efficiency issues
    - easy to build but requires excess capacity on operational systems

Virtual Data Warehouse

End users directly access operational data via middleware tools

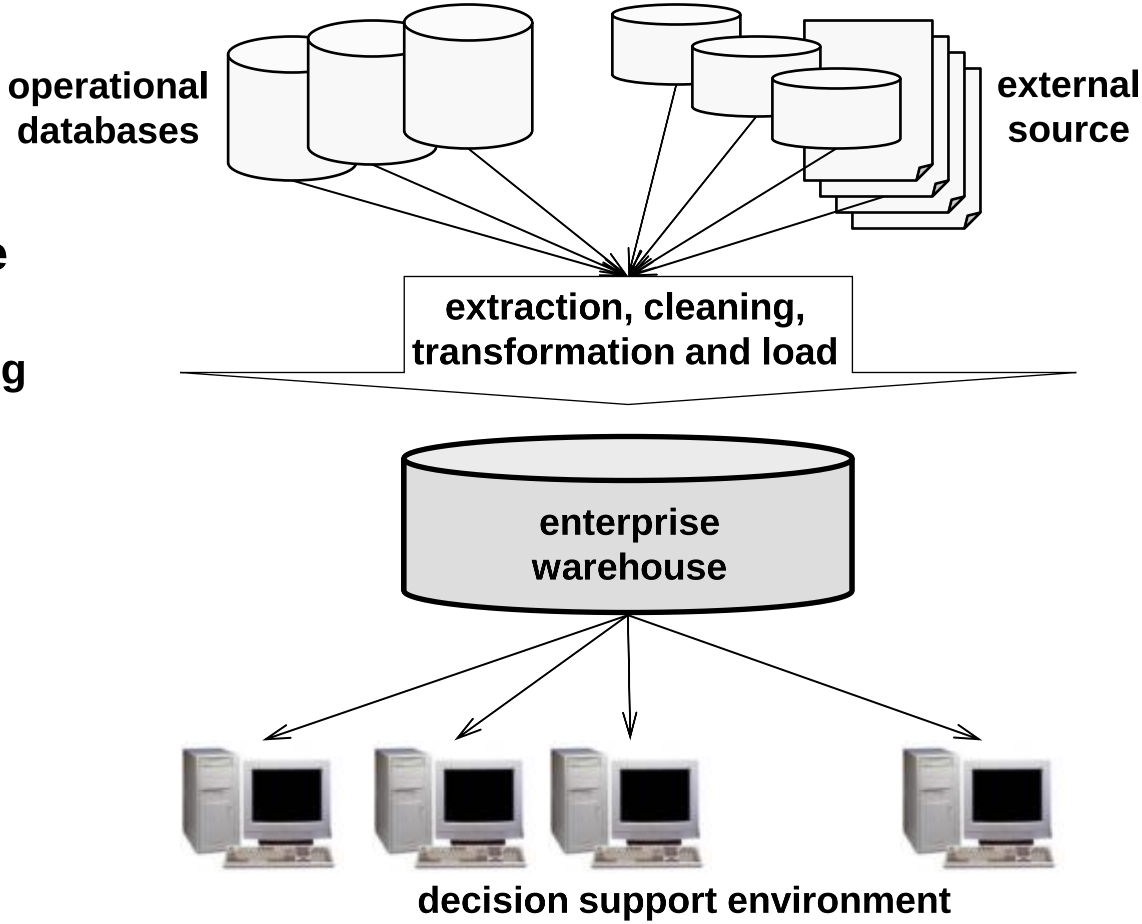
- feasible only if queries are posed infrequently
  - in this case the development of a separate data warehouse is not necessary
- can be used as a temporary solution until a permanent data warehouse is developed.



Generic Architecture of a Data Warehouse

Two level architecture

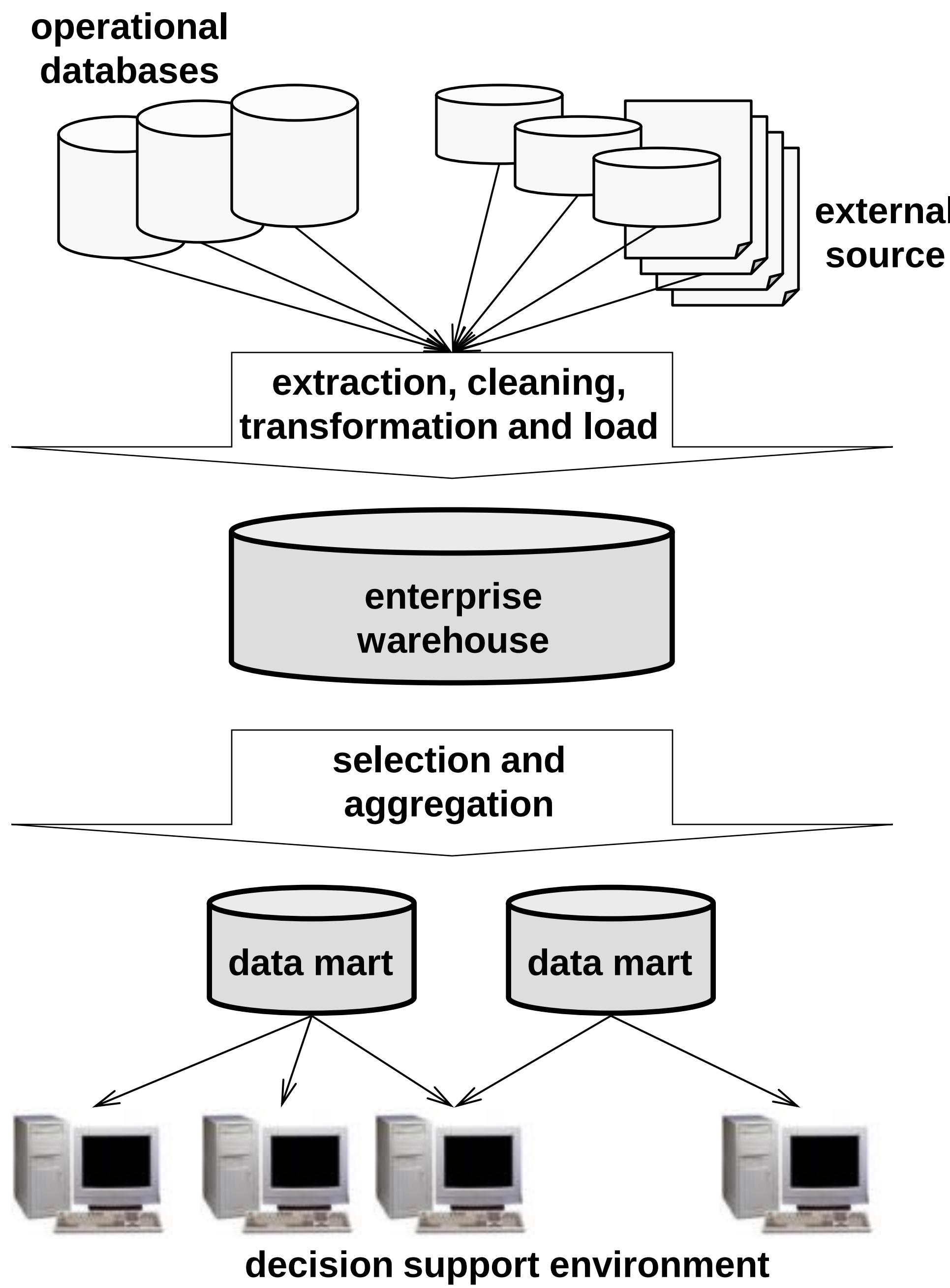
- **operational data**
- **enterprise data warehouse**
  - **used as a single source of data for decision making**



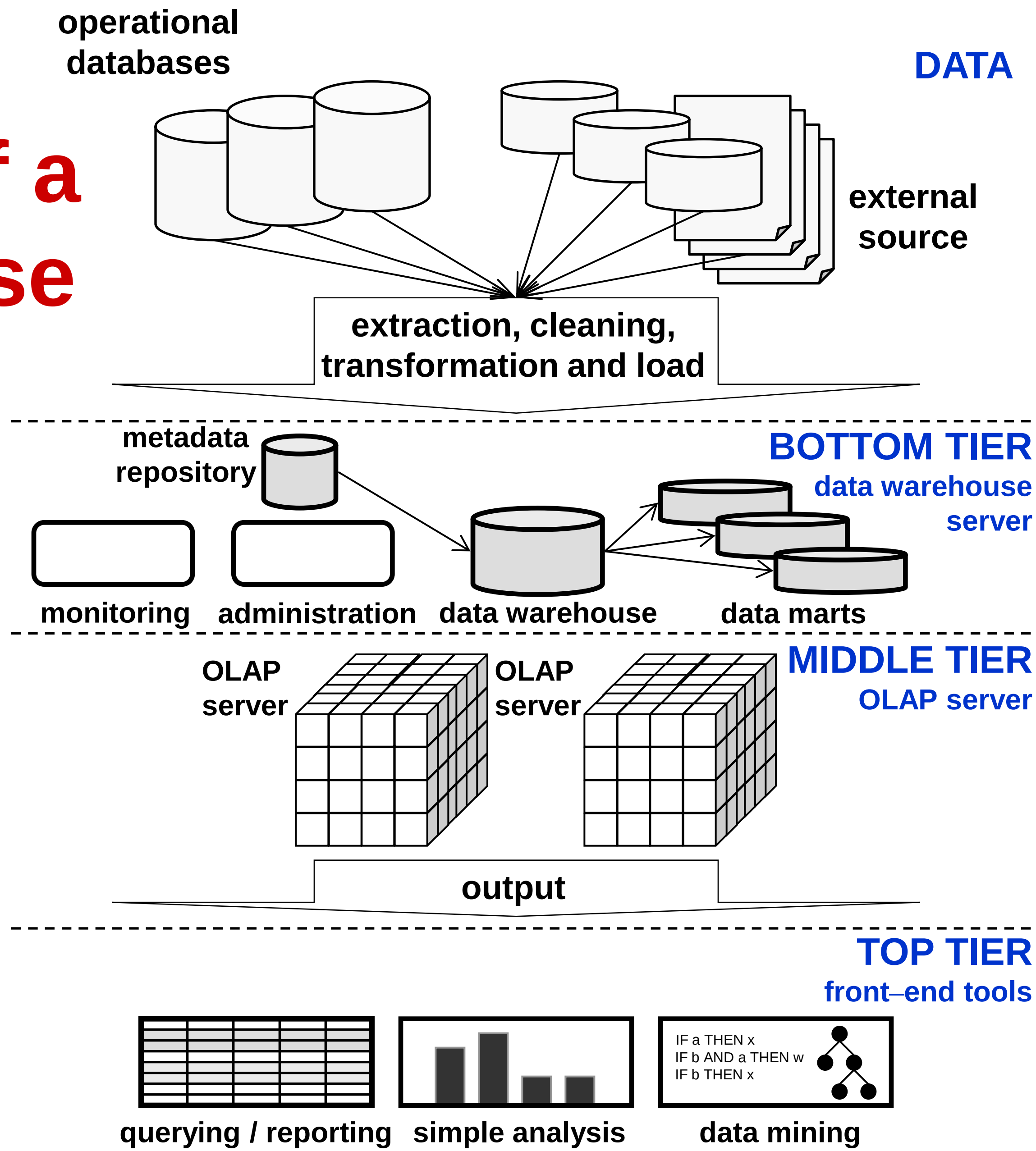
Generic Architecture of a Data Warehouse

Three level architecture

- operational data
- enterprise data warehouse
  - used as a single source of data for decision making
- data marts
  - provide limited scope data selected from a data warehouse



Three Tier Architecture of a Data Warehouse



Three level architecture

- **bottom tier**
  - **data warehouse server**
- **middle tier**
  - **OLAP server for fast querying of the data warehouse**
- **top tier**
  - **displaying results provided by OLAP**
  - **additional mining of the OLAP generated data**

Metadata Repository

Holds data defining warehouse objects

- provides parameters & information for middle and top tier apps
  - description of the structure of the warehouse
    - schema, dimensions, hierarchies, data mart locations and contents, etc.
  - operational meta-data
    - currency of data, i.e. active, archived or purged, and monitoring information, i.e. usage statistics, error reports, audit trails, etc.
  - system performance data
    - indices and hints to improve data access and retrieval performance
  - information about mapping from operational databases
    - source DBs and their contents, cleaning and transformation rules, etc.
  - summarization algorithms
  - business data
    - business terms and definitions, ownership information, etc.

Data Warehouse

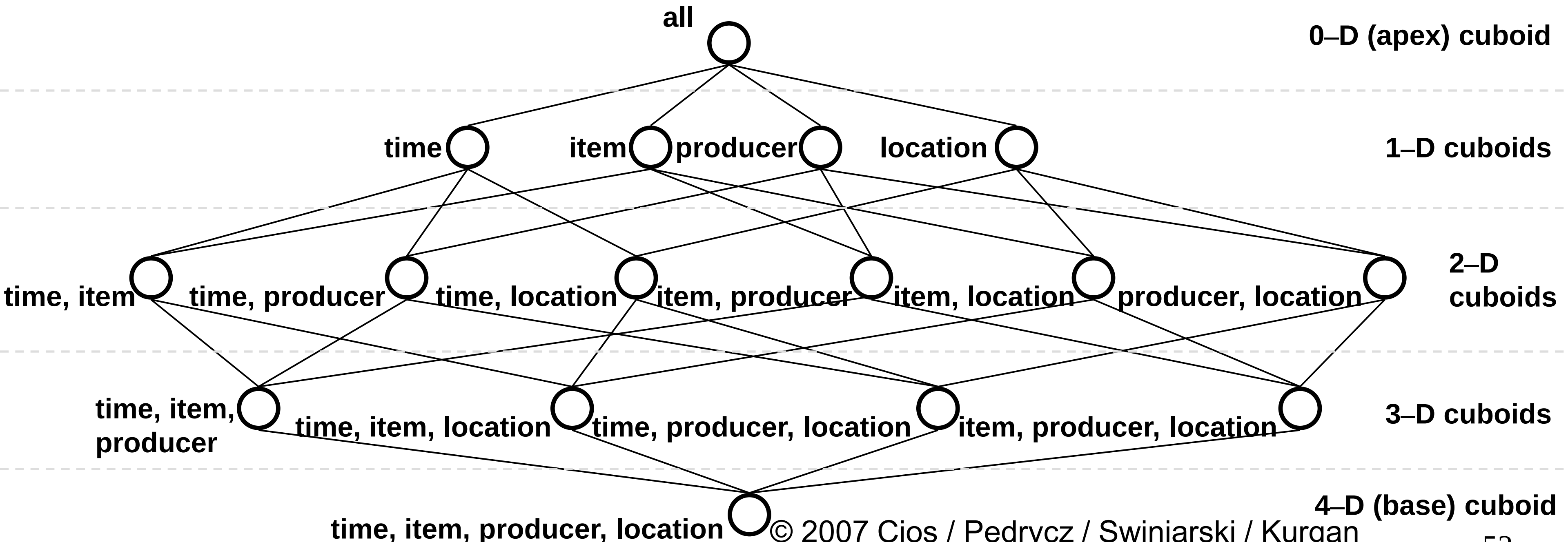
Data warehouse is based on a multidimensional data model

- data is viewed in the form of a data cube
- data cube allows data to be modeled and viewed in multiple dimensions
  - dimensions represent different information
    - item description (name, type)
    - producer information (name)
    - location information (cities)
    - time (day, week, month, quarter, year)
  - fact table spans multiple dimensions
    - contains keys to each of the related dimension tables
    - contains additional summary measures (like value of sold items in dollars)

Data Warehouse

Data cube

- lattice of cuboids forms a data cube
- apex cuboid is the top most 0-D cuboid
  - holds the highest-level of summarization
- base cuboid is an n-D base cube



2–D Data Model

Relational table

- for location = “Denver” and producer = “Company A”
- describes number of sold units

month	CPU_Intel	CPU_AMD	Prnt_HP	Prnt_Lexmark	Prnt_Canon
January2002	442	401	201	302	187
February2002	224	289	134	89	121
March2002	211	271	75	76	312
April2002	254	208	143	108	112
…	…	…	…	…	…

3–D Data Model

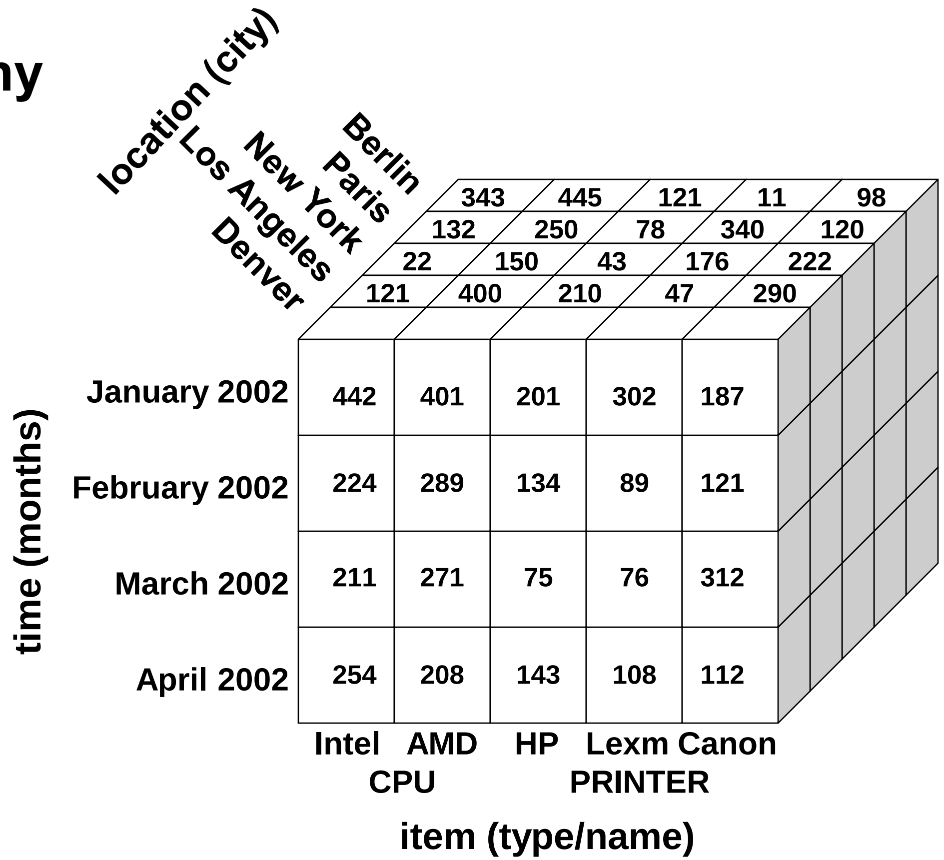
location = "Denver"

month	CPU_Intel	CPU_AMD	Prnt_HP	Prnt_Lexmark	Prnt_Canon
January2002	442	401	201	302	187
February2002	224	289	134	89	121
March2002	211	271	75	76	312
April2002	254	208	143	108	112
…	…	…	…	…	…

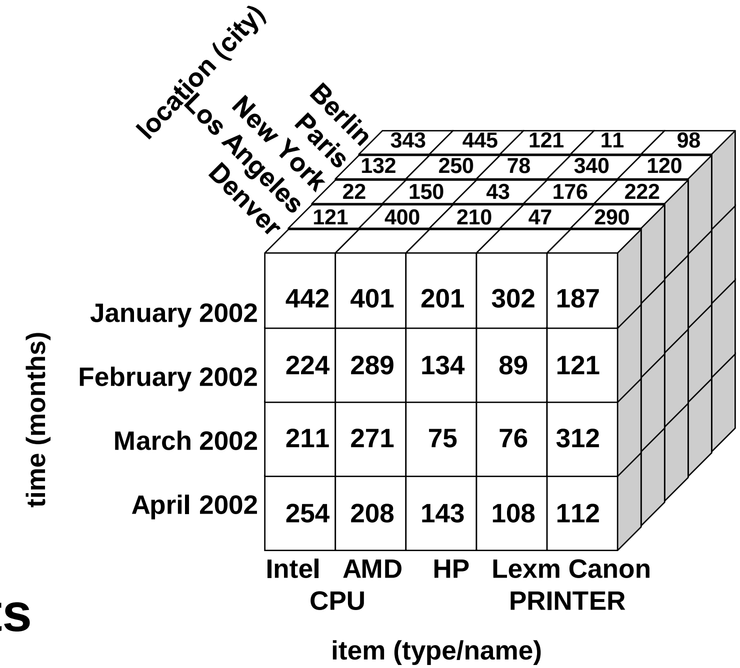
3–D cube

– producer = "Company A"

– describes number of sold units



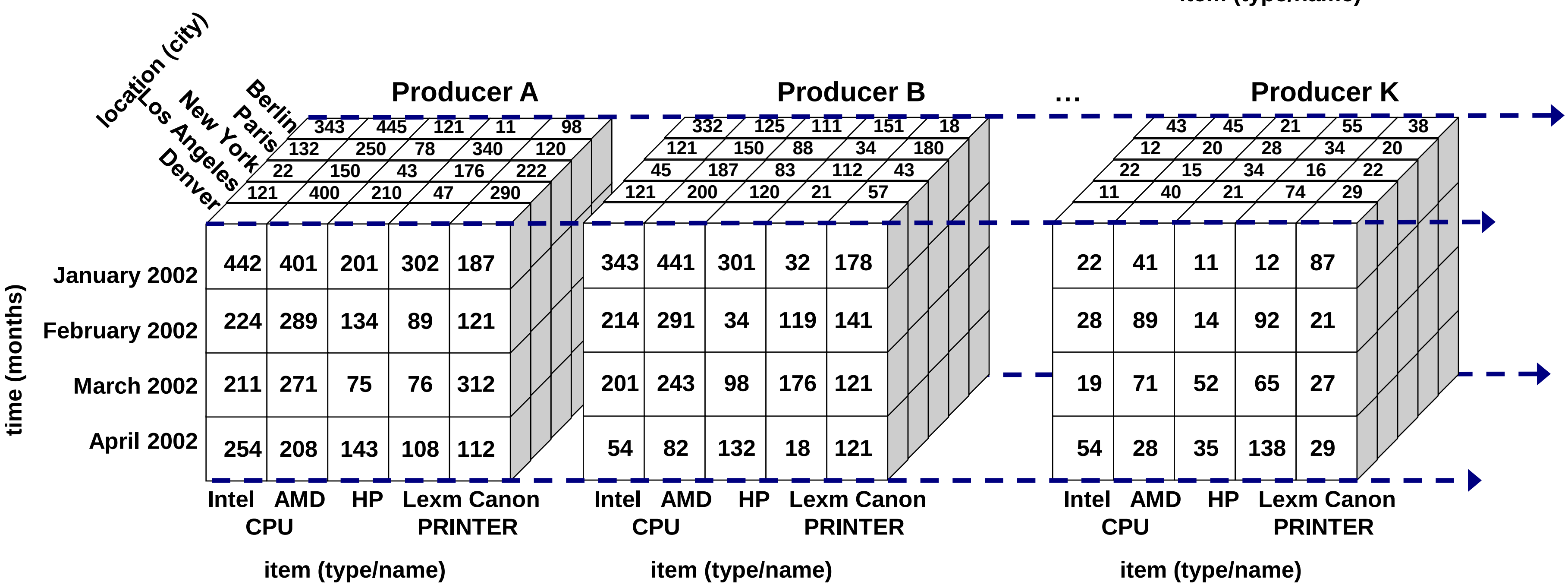
4–D Data Model



4–D cube

– different producers

– describes number of sold units



Data Warehouse

Modeling

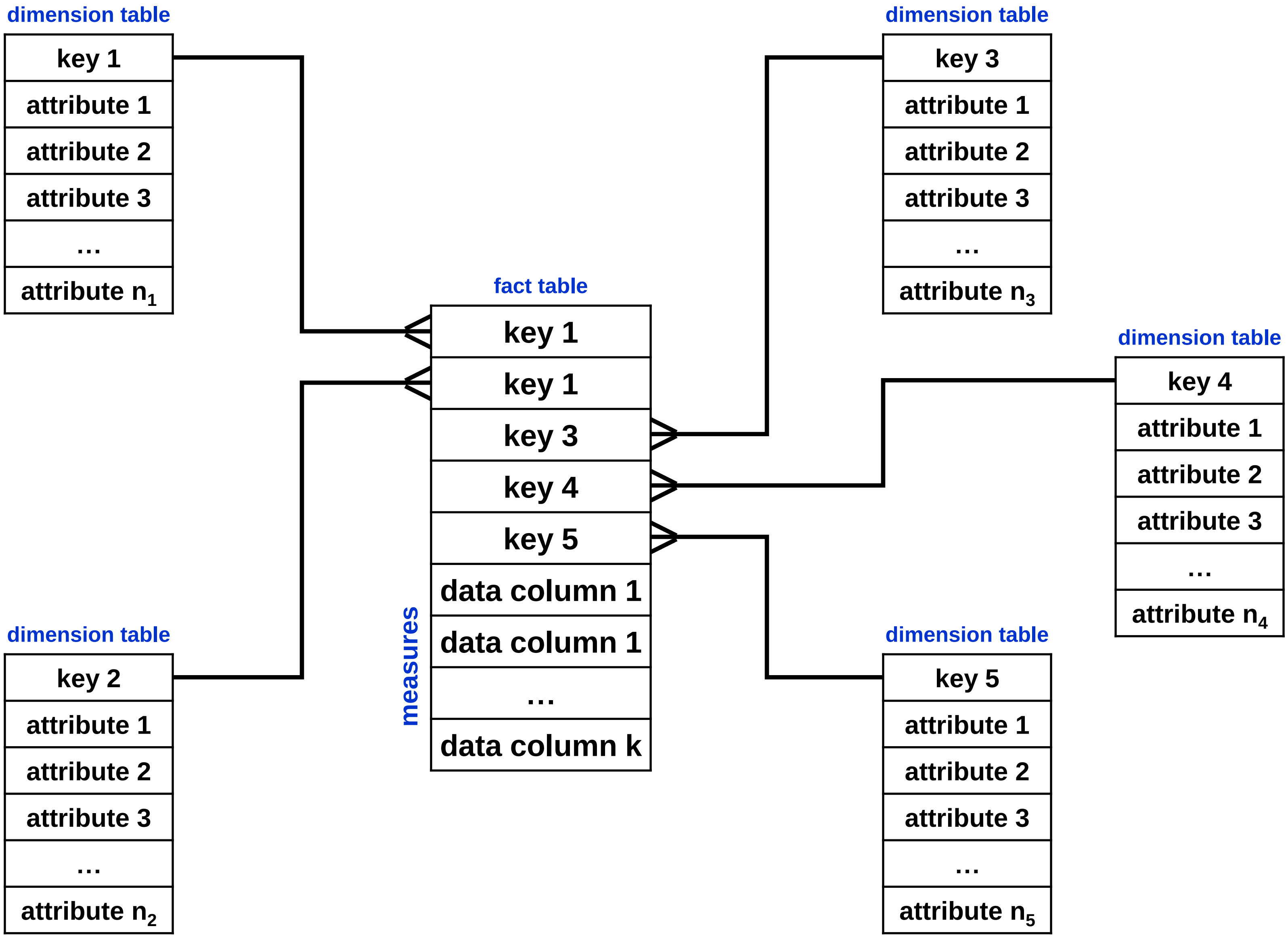
- provides subject-oriented schema to perform data analysis through use of dimensions and measures

– **star schema**
  - fact table in the middle, which is connected to a set of dimension tables

– **snowflake schema**
  - refinement of star schema where some dimensional tables are normalized into a set of smaller tables, forming a shape similar to a snowflake

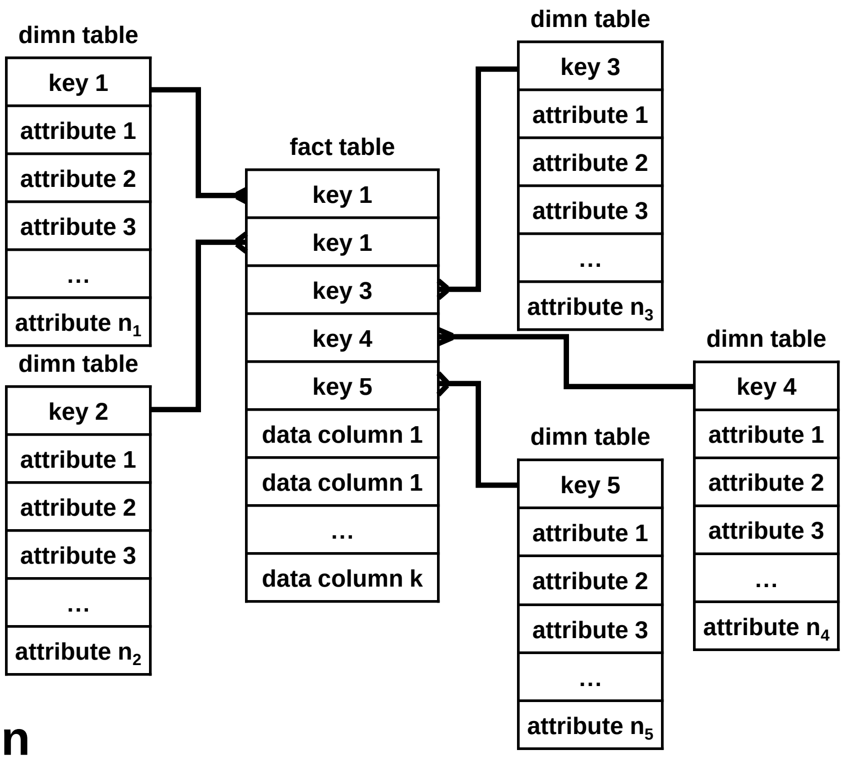
– **galaxy schema**
  - multiple fact tables share dimension tables
  - a collection of stars is also called fact constellation

Star Schema



dimension table
key 1
attribute 1
attribute 2
attribute 3
…
attribute n1
dimension table
key 3
attribute 1
attribute 2
attribute 3
…
attribute n3
fact table
key 1
key 1
key 3
key 4
key 5
data column 1
data column 1
…
data column k
measures
dimension table
key 4
attribute 1
attribute 2
attribute 3
…
attribute n4
dimension table
key 2
attribute 1
attribute 2
attribute 3
…
attribute n2
dimension table
key 5
attribute 1
attribute 2
attribute 3
…
attribute n5

Star Schema



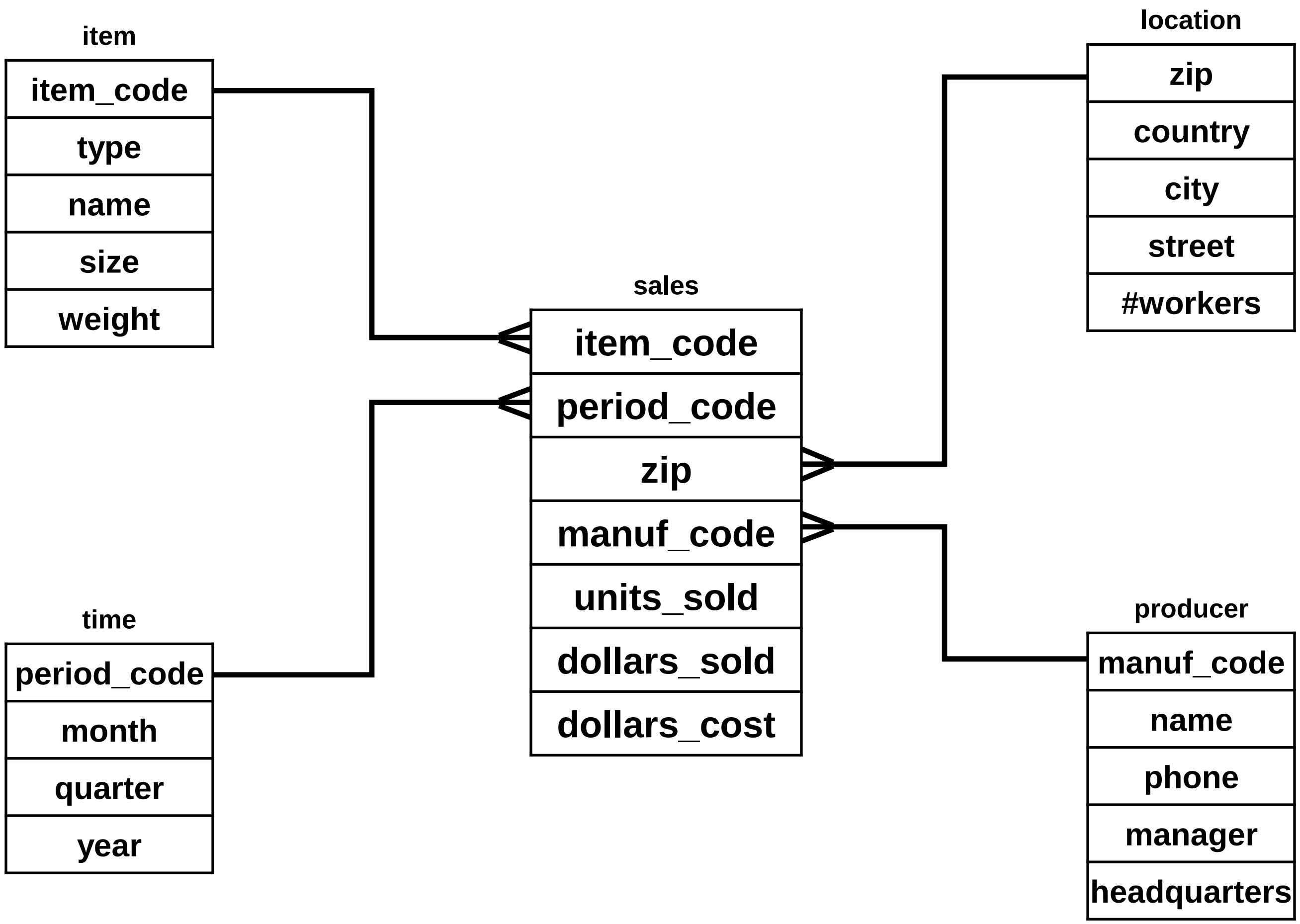
Consists of

- single fact table containing the data with no redundancy
  - it has a primary key has only one key column per dimension
  - for the sake of efficiency each key is generated
- single table per dimension
  - each dimension is a single table
  - highly denormalized
    - it may not follow the Boyce-Codd normalization
      - » e.g. may contain redundant data

Star Schema

- **Information is extracted by**
  - **performing join operation between the fact table and one or more dimension tables followed by projections and selection operations**
    - **projection selects particular columns**
    - **selection selects particular rows**
- **Benefits**
  - **easy to understand, reduces number of physical joins to extract information, requires very little maintenance**
- **Drawbacks**
  - **does not provide support for attribute hierarchies**

Example Star Schema



Example Star Schema with Sample Data

item

item_code	type	name	size	weight
100	CPU	Intel PIII	10x30x0.1	111
104	CPU	Intel PIV	10x30x0.1	101
121	CPU	Intel Pentium	10x30x0.1	104
…	…	…	…	…

location

zip	country	city	street	#workers
80234-17562	USA	Denver	14^{th} street	34
80541-90876	USA	LA	Broadway St	123
31546-89791	USA	NY	102^{th} Ave	54
…	…	…	…	…

sales

item_code	period_code	zip	manuf_code	units_sold	dollars_sols	dollars_cost
100	001	80234-17562	M01	315	315,000	215,000
100	001	80541-90876	M01	213	213,000	110,000
100	001	31546-89791	M01	24	24,000	45,000
121	004	98776-18765	M02	456	245,000	230,000
…	…	…	…	…	…	…

time

period_code	month	quarter	year
0001	January	1	2002
0002	February	1	2002
0003	March	1	2002
0004	April	1	2002
…	…	…	…

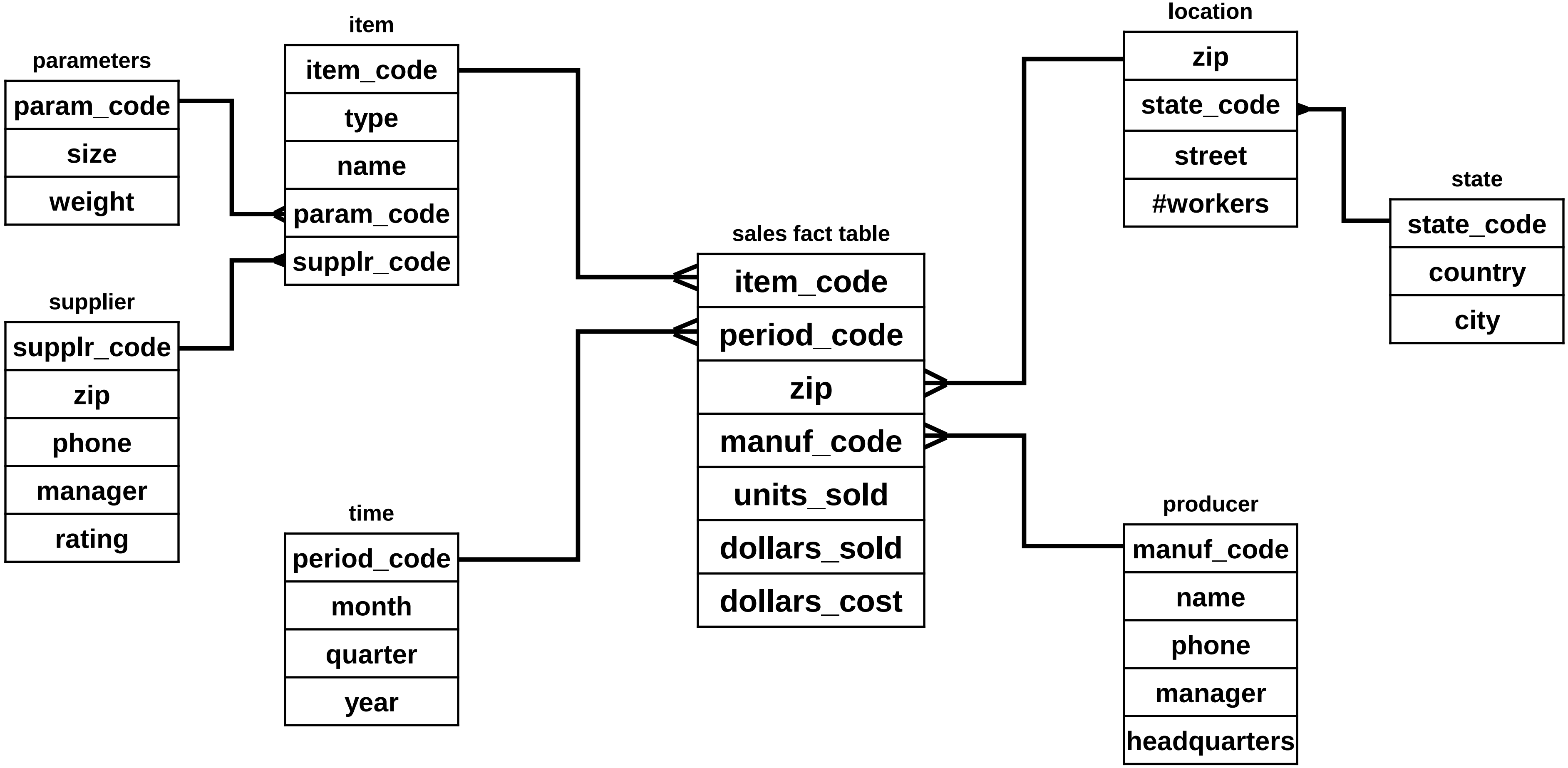
producer

manuf_code	name	phone	manager	headquarters
M01	CompuBus	451 334 5578	B.J. Smith	Colorado, USA
M02	CyberMax	213 443 9018	W. Red	Floryda USA
M04	MiniComp	776 552 1854	J. Mance	Alberta, CA
…	…	…	…	…

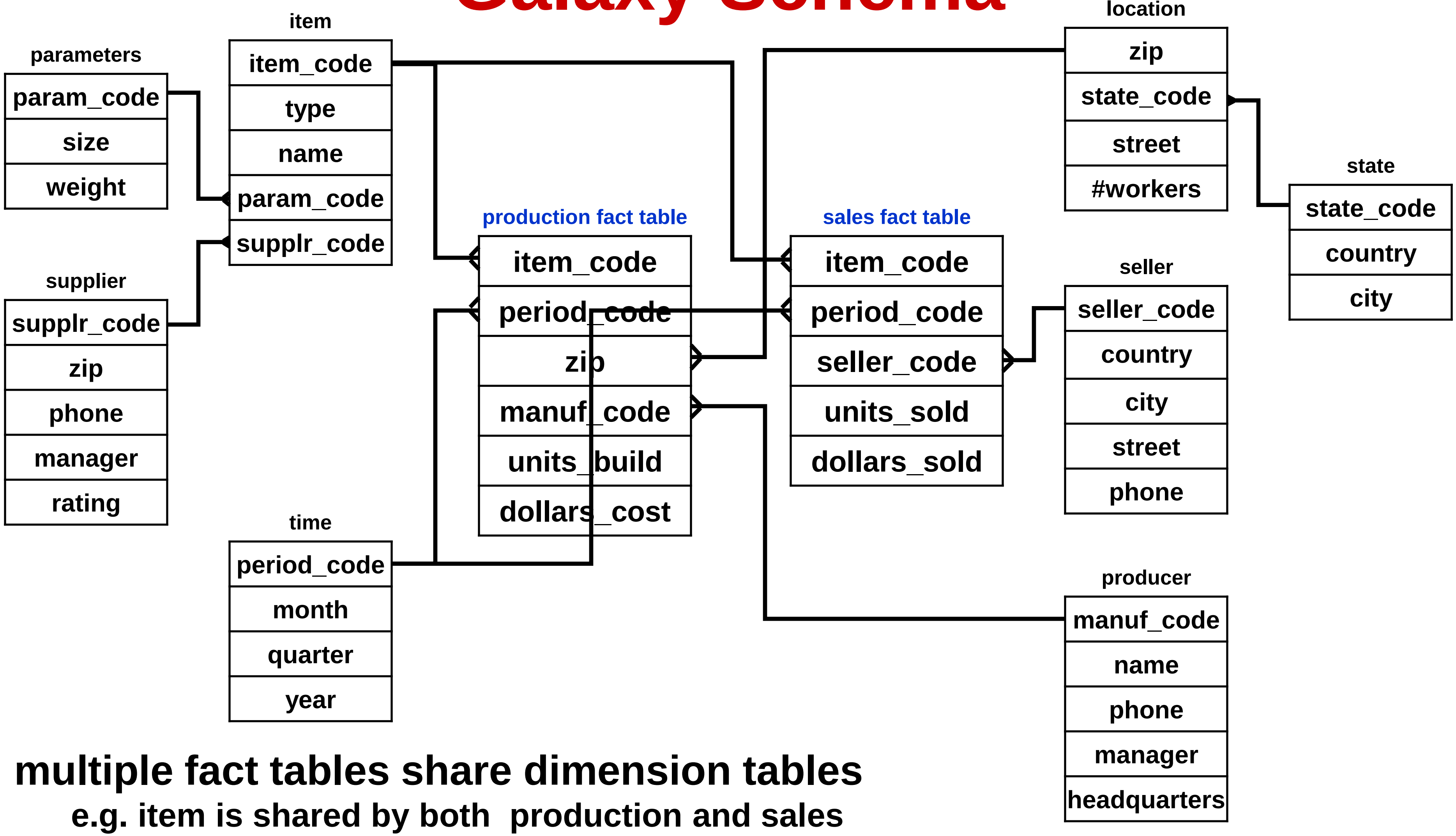
Snowflake Schema

- Refinement of star schema where some dimensional tables are normalized into a set of smaller tables, forming a shape similar to snowflake
  - normalized dimensions improve easiness of maintaining the dimension tables and save storage space
    - less redundancy
    - however, the saving of space is, in most cases, negligible in comparison to the typical magnitude of the size of the fact table
  - usually represents and exposes concept hierarchy which often relates to the aggregation levels
- Drawbacks
  - large number of tables must be joined to support even the most basic queries
  - worse performance

Snowflake Schema



Galaxy Schema



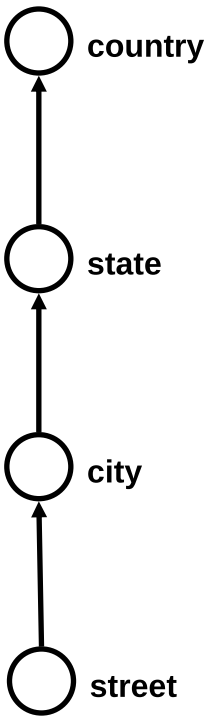
multiple fact tables share dimension tables

e.g. item is shared by both production and sales

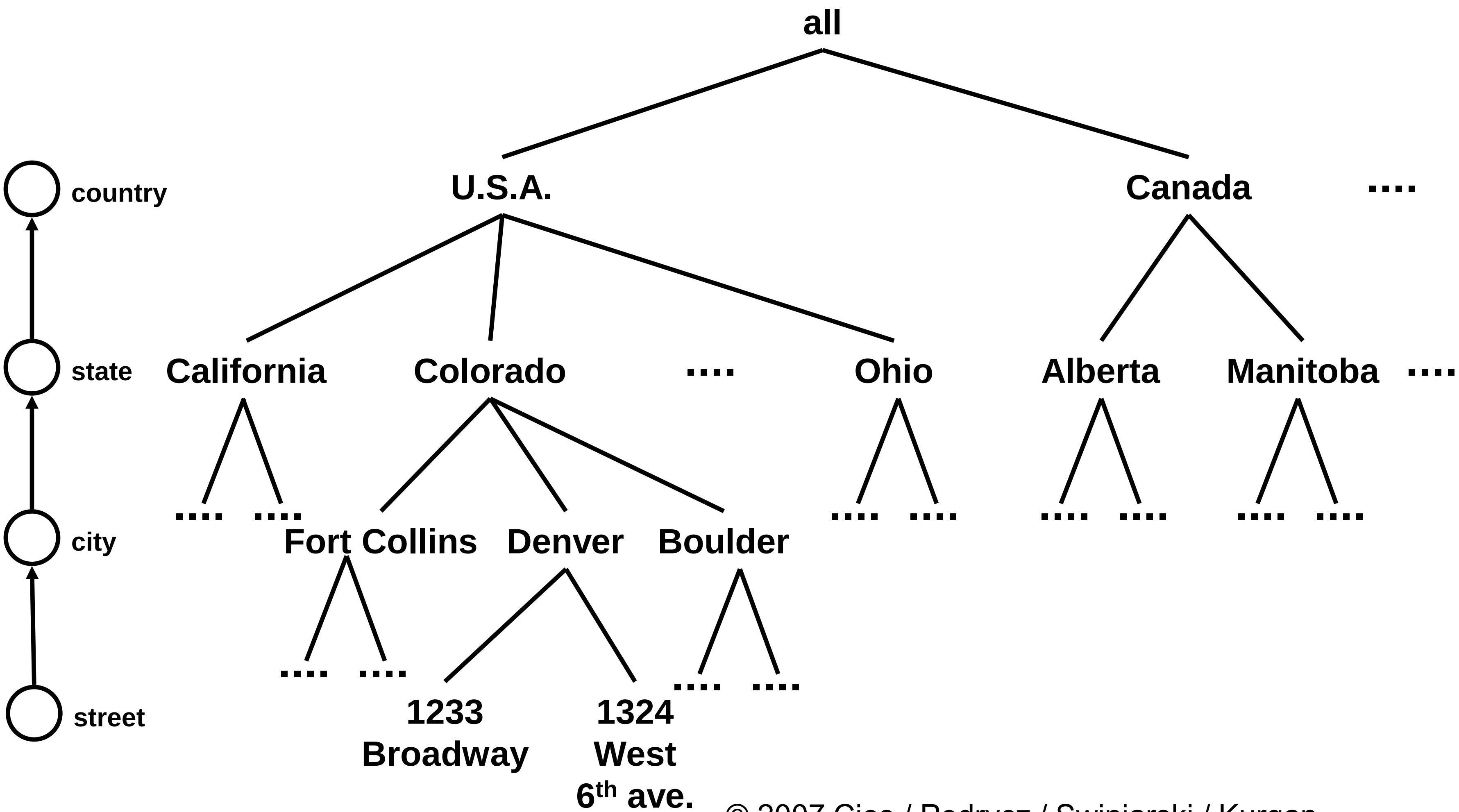
Concept Hierarchy

Defines a sequence of mappings from a set of very specific, low-level, concepts to more general, higher-level, concepts

- **e.g. concept of location**
  - **each city can host multiple shippers defined by their street address**
    - **city values include Denver and Los Angeles**
  - **each city is mapped to the state or province where it belongs**
  - **and finally state or province is mapped to the country to which they belong**



Example Concept Hierarchy



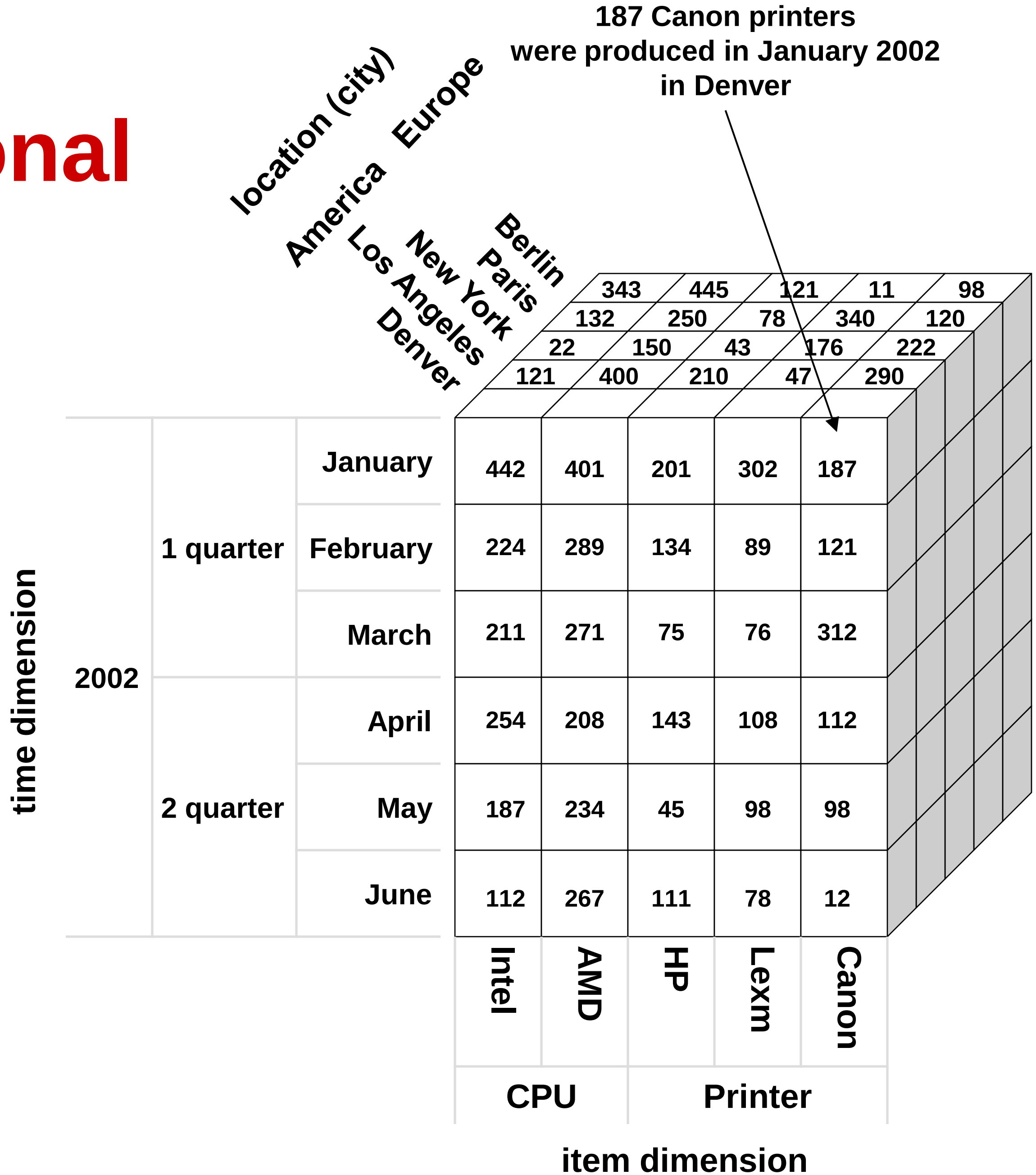
Concept Hierarchy

Concept hierarchies are useful to perform OLAP

- **data are organized in multiple dimensions where each dimension contains multiple levels of abstraction defined by concept hierarchies**
  - **it gives flexibility to summarize data on various levels of granularity**
  - **and OLAP operations enable materialization of such views**

Multi Dimensional Data Model

- 3–D cube
  - both time and item have a hierarchical structure



OLAP

DWs use on-line analytical processing (OLAP) to formulate and execute user queries

OLAP is an SLQ-based methodology that provides aggregate data (measurements) along a set of dimensions

OLAP

OLAP is a methodology that provides aggregate data (measurements) along a set of dimensions, where

- each dimension is described by a set of attributes
- each measure depends on a set of dimensions, which provide context for the measure
  - all dimensions are assumed to uniquely determine the measure

OLAP

Basic operations

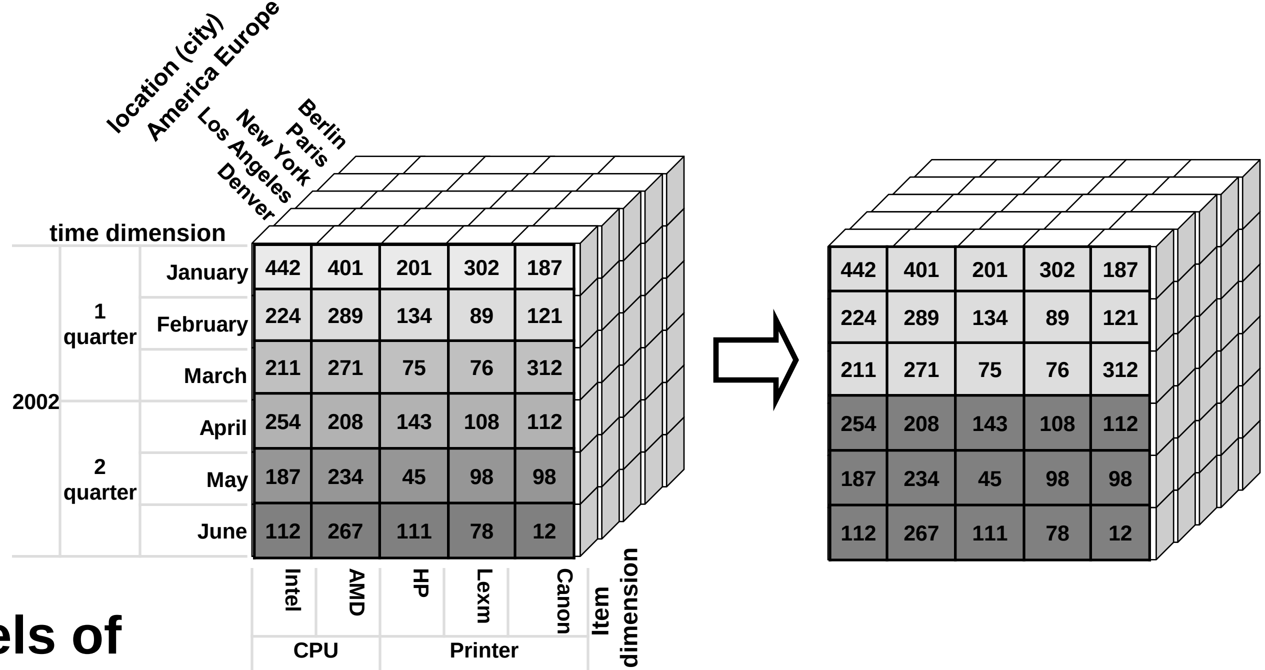
- Roll Up
  - navigates to lower levels of detail
    - takes current data object and performs a GROUP BY on one of the dimensions
    - example: given total production by month, it can provide production by a quarter
- Drill Down
  - navigates to higher levels of detail
    - converse of the roll-up
    - example: example: given total production in all regions, it can provide production in USA
- Slice
  - provides cut through the cube
  - enables users to focus on some specific perspectives
    - example: provides data concerning only production in LA

OLAP

Basic operations

- **Pivot**
  - rotates the cube to change the perspective
    - example: example: changing the perspective from "time item" to "time location"
- **Dice**
  - provides just one cell from the cube (the smallest slice)
    - example: provides data concerning the production of Canon printers in May 2002 in Denver
      - » city, product name, and month are the smallest members in location, product, time dimensions

Roll Up



Navigates to lower levels of detail

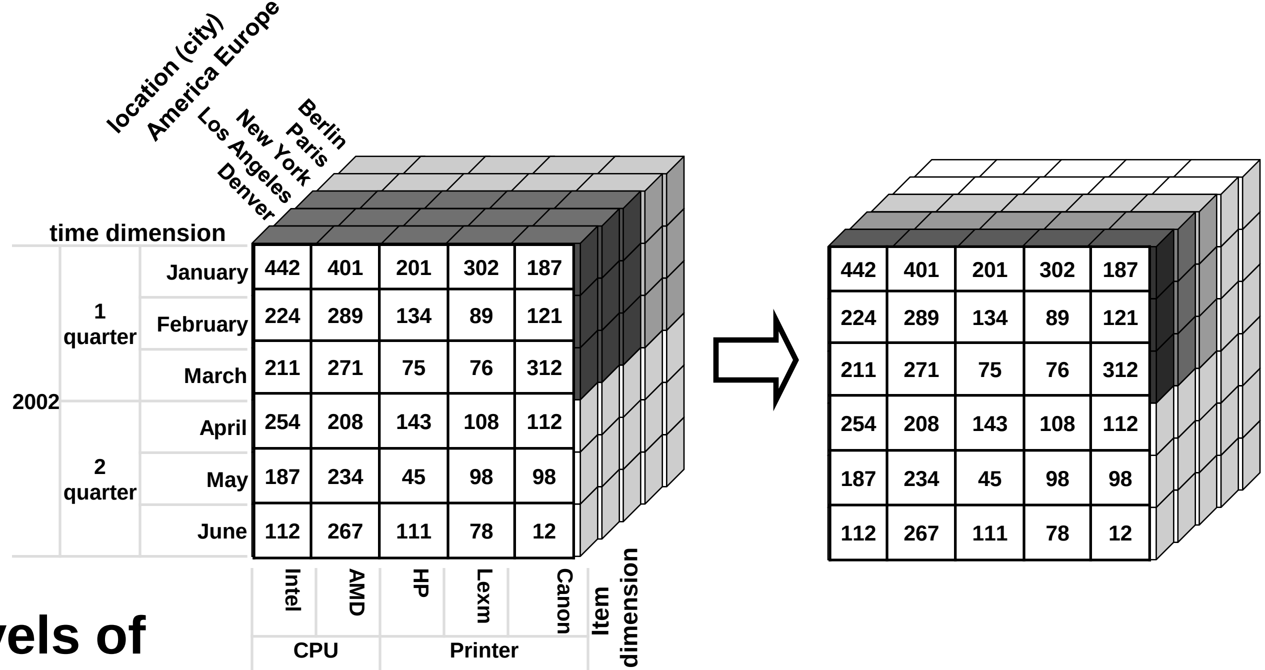
- takes current data object and performs a GROUP BY on one of the dimensions
- example: given total production by month, it can provide production by a quarter
  - production in Denver

# produced units		2002	
		Quarter 1	Quarter 2
CPU	Intel	877	553
	AMD	961	709

roll up on dimension time

# produced units		2002					
		January	February	March	April	May	June
CPU	Intel	442	224	211	254	187	112
	AMD	401	289	271	208	234	267

Drill Down



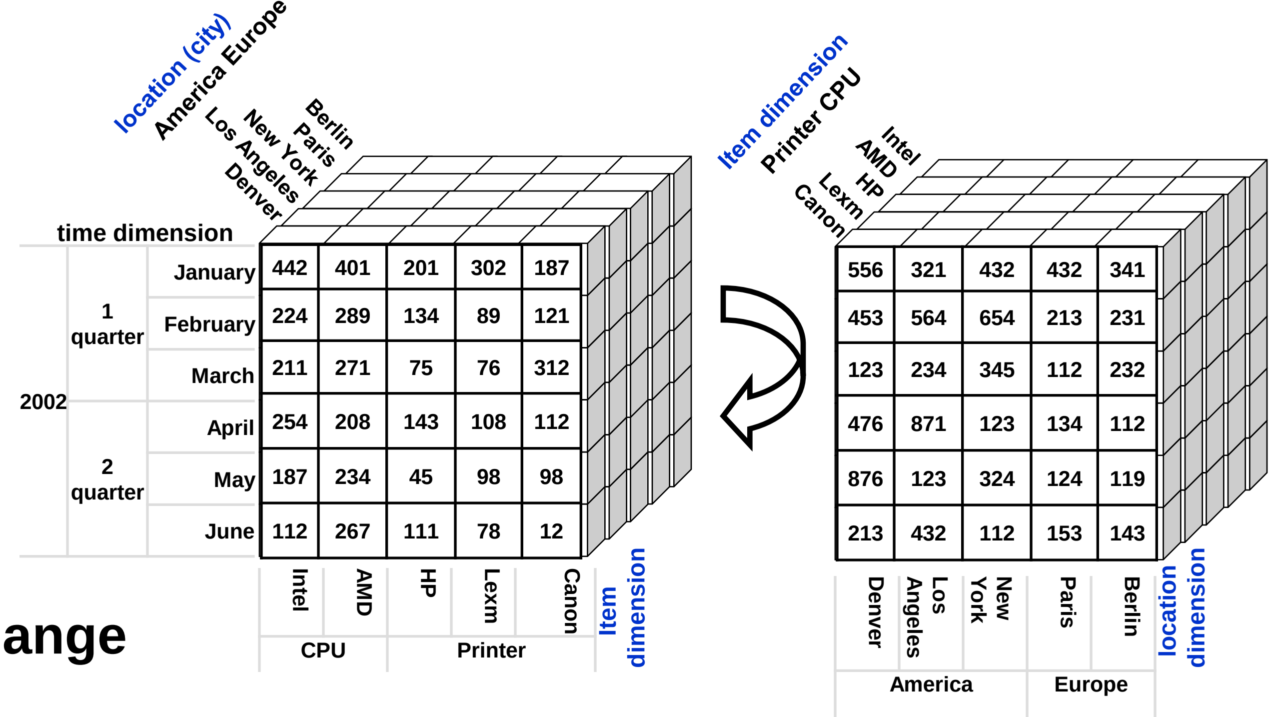
- Navigates to higher levels of detail
  - converse of the roll-up
  - example: given total production in all regions, it can provide production in USA
    - production in first quarter

# produced units		CPU		Printer		
		Intel	AMD	HP	Lexm	Canon
USA	Denver	877	961	410	467	620
	LA	833	574	621	443	213
	NY	521	599	770	650	296

drill down on dimension location America

# produced units		CPU		Printer		
		Intel	AMD	HP	Lexm	Canon
All	USA	2231	2134	1801	1560	1129
	Europe	1981	2001	1432	1431	1876

Pivot



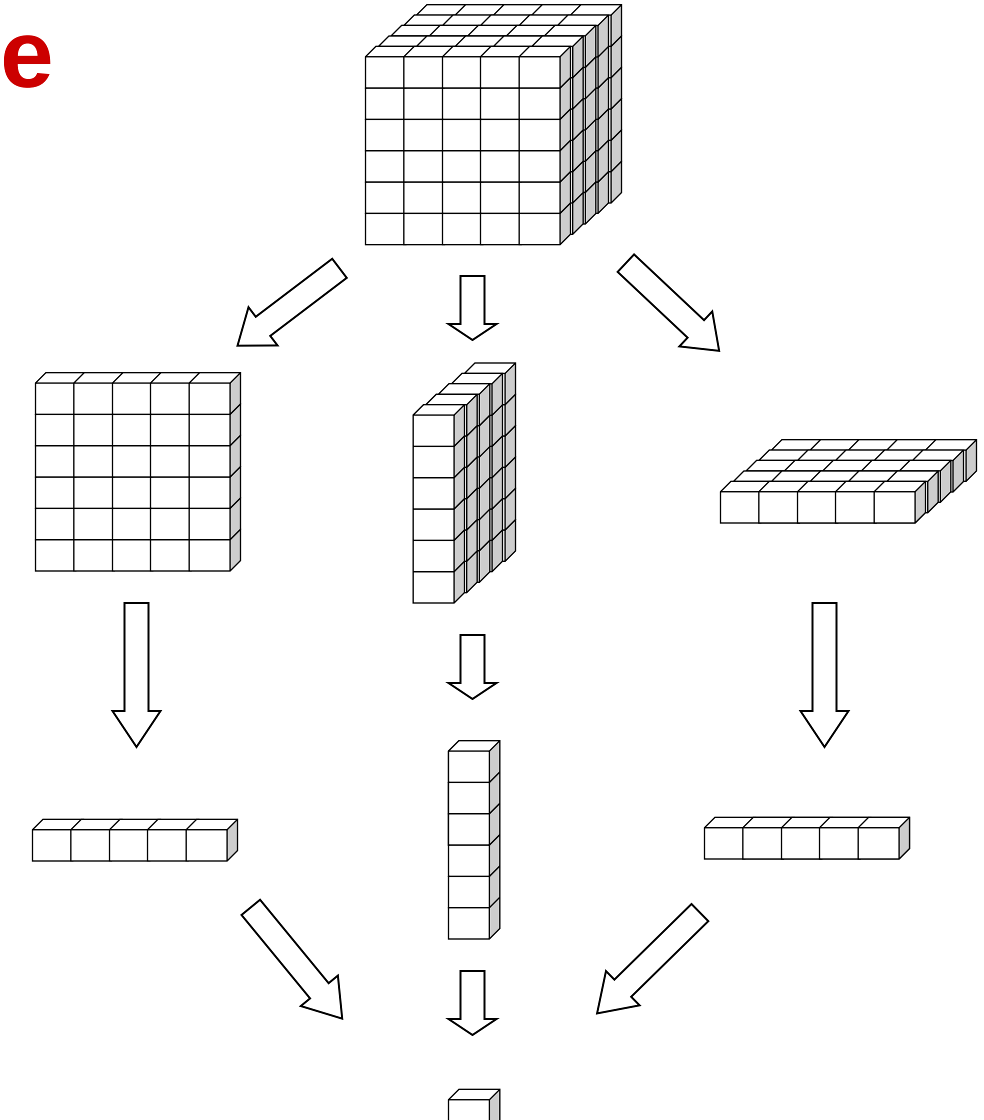
- **Rotates the cube to change the perspective**
  - example: changing the perspective from "time item" to "time location"
    - time is the fixed axis

# produced units		America			Europe	
		Denver	LA	NY	Paris	Berlin
quarter 1	January	556	321	432	432	341
	February	453	564	654	213	231
	March	123	234	345	112	232

pivot

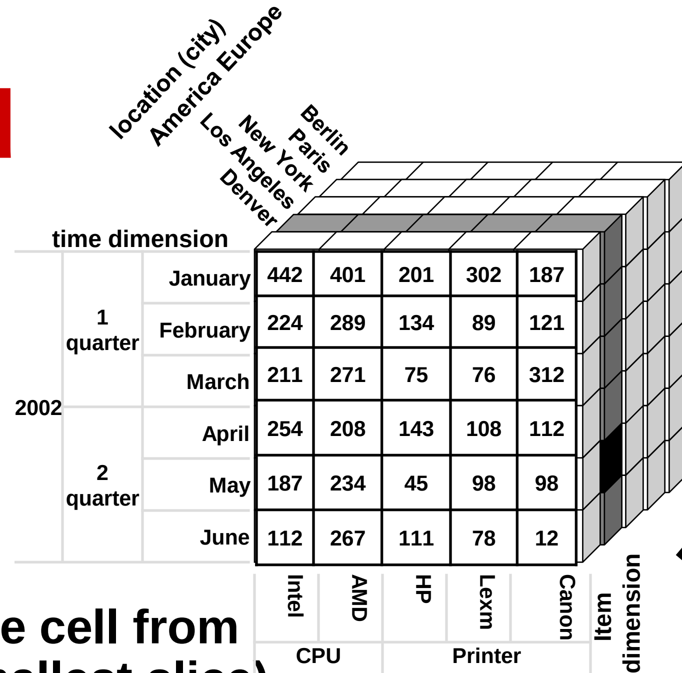
# produced units		CPU		Printer		
		Intel	AMD	HP	Lexm	Canon
quarter 1	January	442	401	201	302	187
	February	224	289	134	89	121
	March	211	271	75	76	312

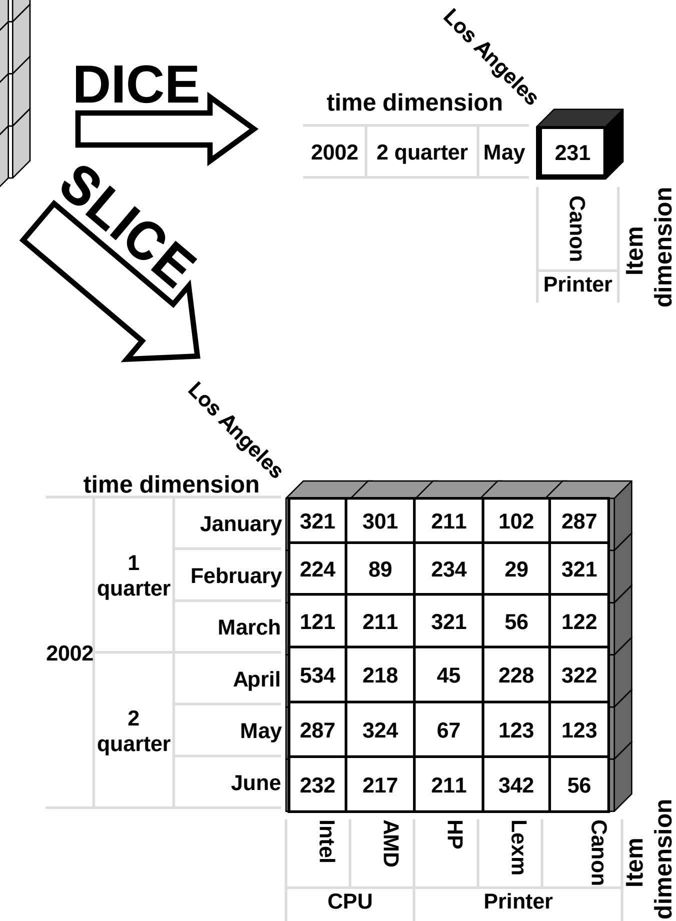
Slice and Dice



Perform selection and projection on one or more dimensions

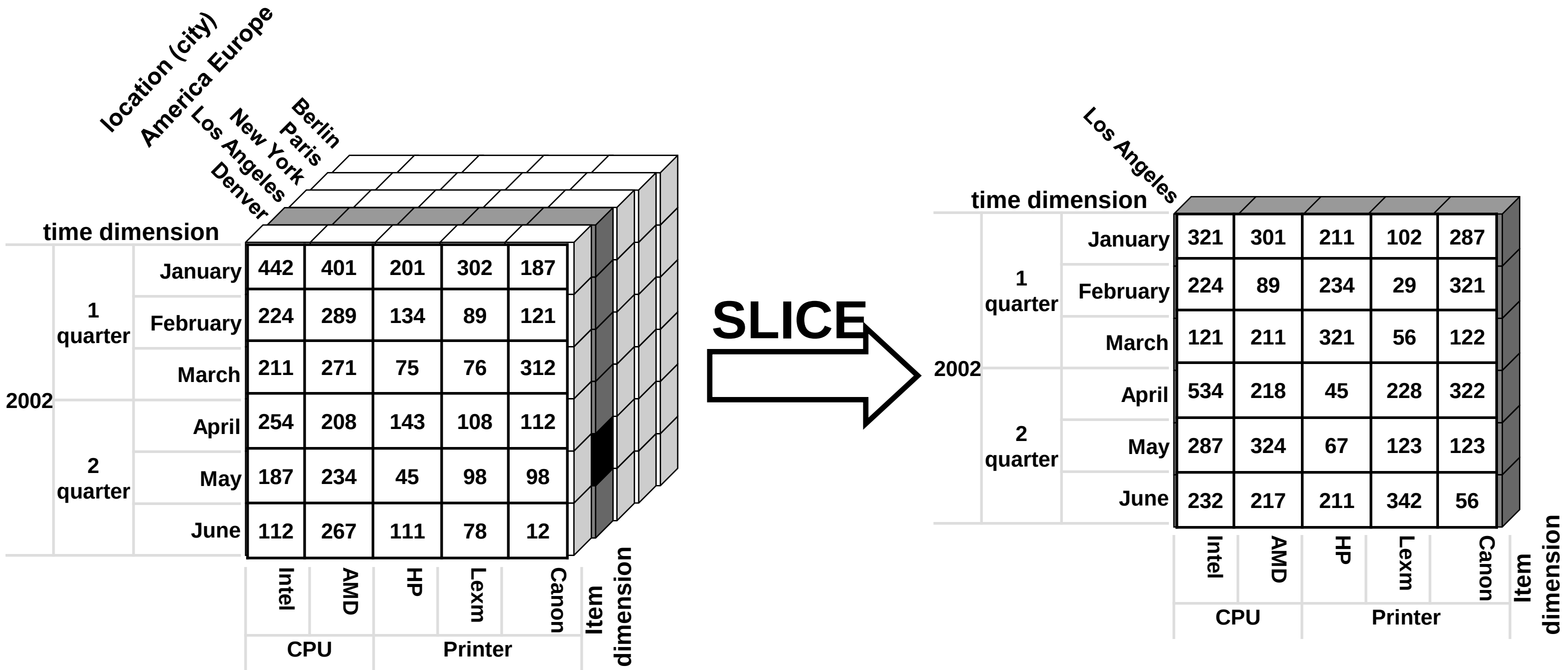
Slice and Dice





- **Dice**
  - provides just one cell from the cube (the smallest slice)
    - example: provides data concerning the production of Canon printers in May 2002 in LA
- **Slice**
  - provides cut through the cube
  - enables users to focus on some specific perspectives
    - example: provides data concerning only production in LA

Slice



- Provides cut through the cube
- Enables users to focus on some specific perspectives
  - example: provides data concerning only production in LA

production in Los Angeles

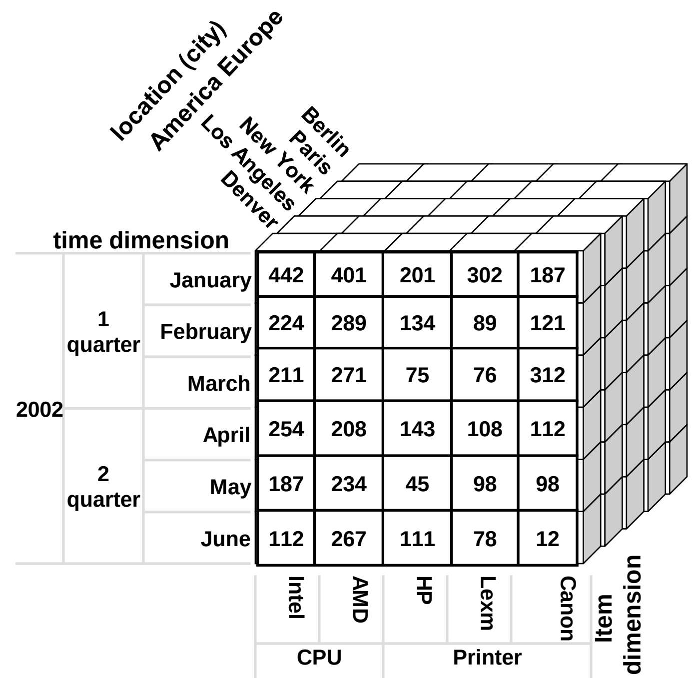
# produced units		CPU		Printer		
		Intel	AMD	HP	Lexm	Canon
2002	1 quarter	666	601	766	187	730
	2 quarter	1053	759	323	693	501

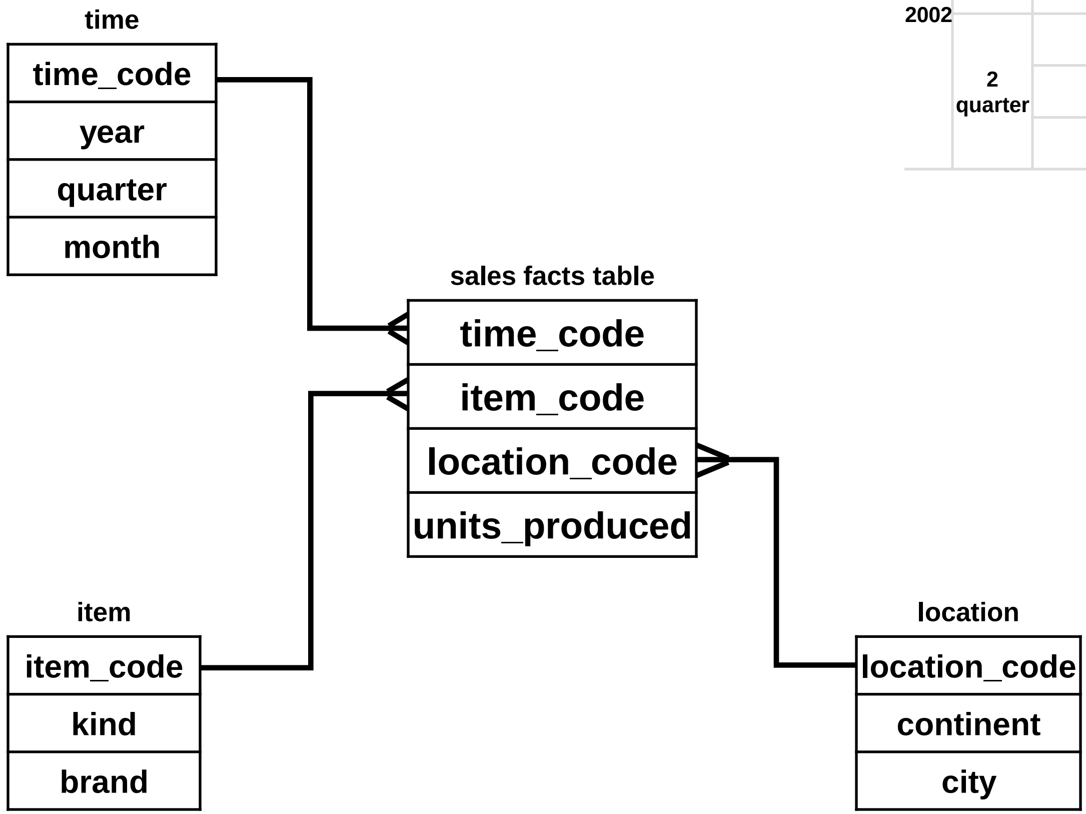
drill down on dimension location USA

production in all regions

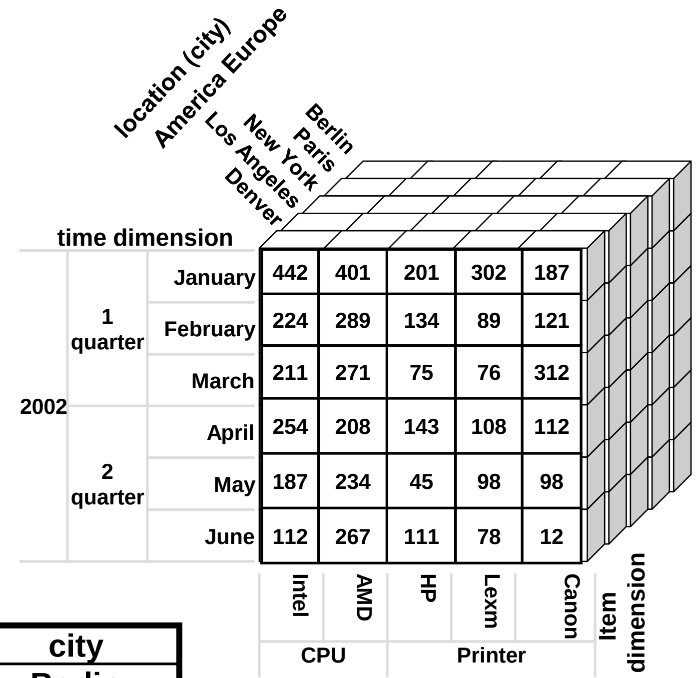
# produced units		CPU		Printer		
		Intel	AMD	HP	Lexm	Canon
2002	1 quarter	2231	2001	2390	1780	1560
	2 quarter	2321	2341	2403	1851	1621

Star Schema for Computer Sales





Relational Representation



Each dimension is represented as a relational table + a separate facts table

time dimension table

time_code	year	quarter	month
1	2002	1	January
2	2002	1	February
3	2002	1	March
4	2002	2	April
5	2002	2	May
6	2002	2	June

location dimension table

location_code	continent	city
1	Europe	Berlin
2	Europe	Paris
3	America	New York
4	America	Los Angeles
5	America	Denver

item dimension table

item_code	kind	brand
1	CPU	Intel
2	CPU	AMD
3	Printer	HP
4	Printer	Lexm
5	Printer	Canon

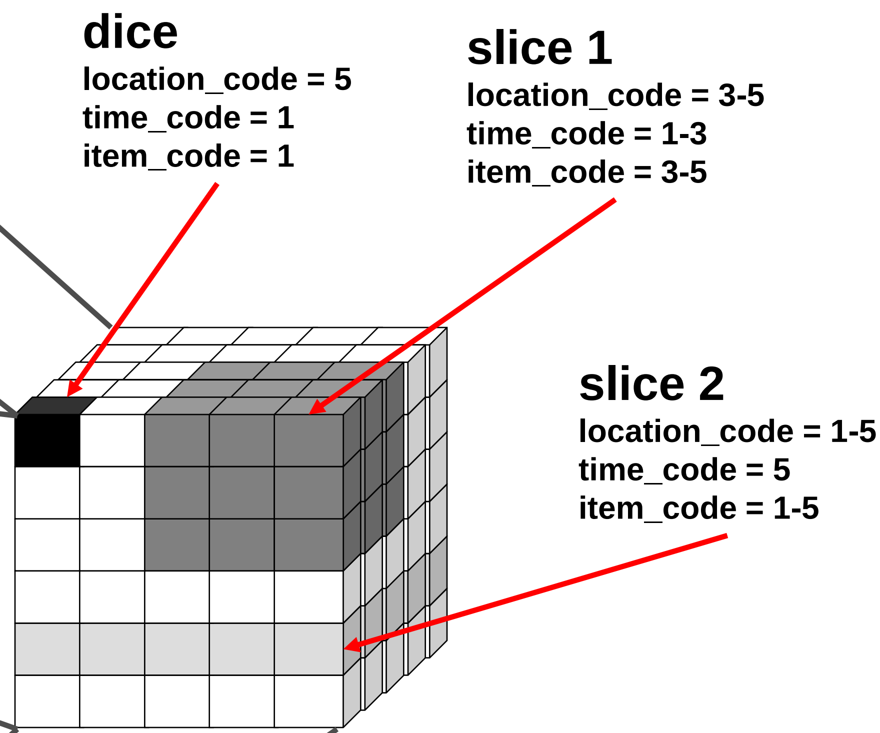
sales facts table

time_code	item_code	location_code	units_produced
1	1	1	111
1	1	2	232
1	1	3	123
1	1	4	322
1	1	5	442
1	2	1	401
1	2	2	276
…	...	…	…
6	5	5	12

OLAP Cube for Computer Sales

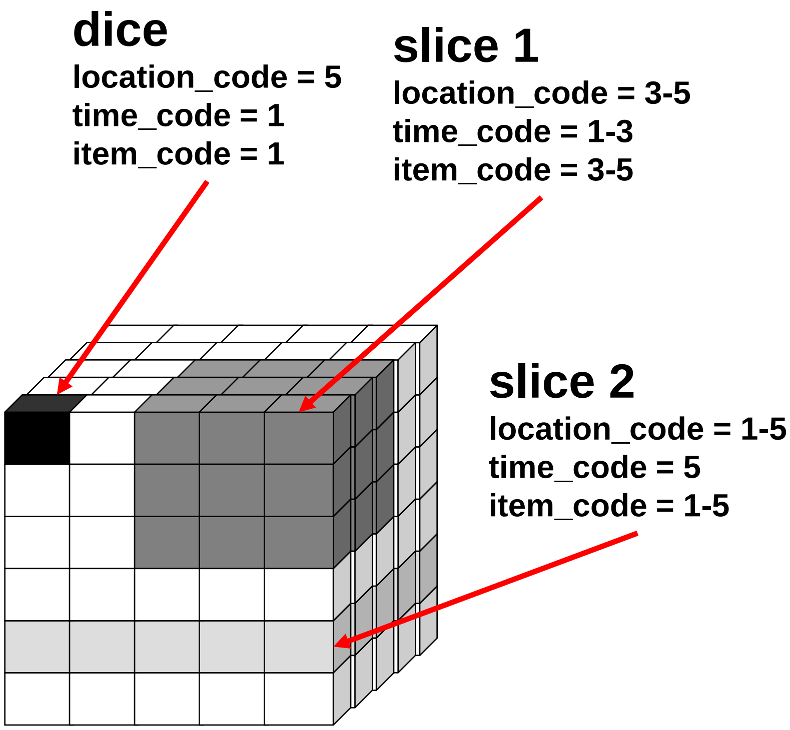
location_code	continent	city
1	Europe	Berlin
2	Europe	Paris
3	America	New York
4	America	Los Angeles
5	America	Denver

time_code	year	quarter	month
1	2002	1	January
2	2002	1	February
3	2002	1	March
4	2002	2	April
5	2002	2	May
6	2002	2	June



item_code	kind	brand
1	CPU	Intel
2	CPU	AMD
3	Printer	HP
4	Printer	Lexm
5	Printer	Canon

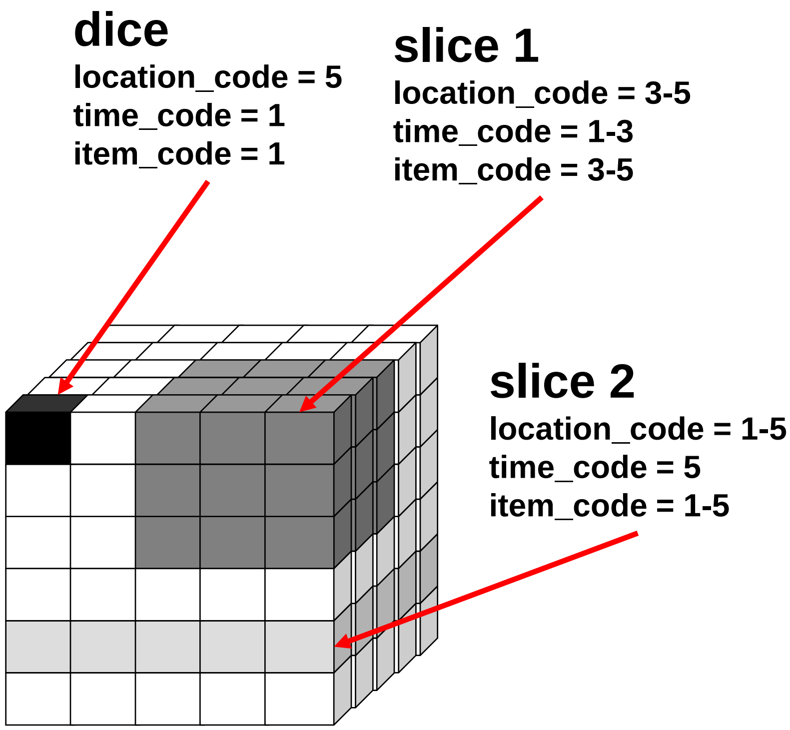
OLAP Cube for Computer Sales



SQL statement for dice

```
SELECT      units_produced
FROM        location L, time T, item I, facts F
WHERE       F.location_code = L.location_code
            AND F.time_code = T.time_code
            AND F.item_code = I.item_code
            AND L.city = 'Denver'
            AND T.month = 'January'
            AND I.brand = 'Canon';
```

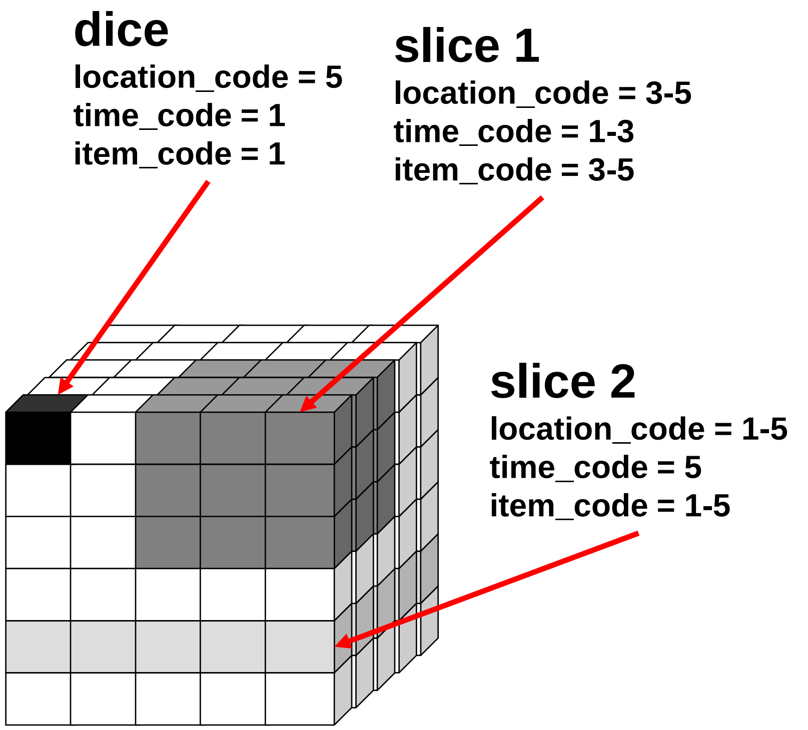
OLAP Cube for Computer Sales



SQL statement for slice 1

```
SELECT    units_produced
FROM      location L, time T, item I, facts F
WHERE     F.location_code = L.location_code
          AND F.time_code = T.time_code
          AND F.item_code = I.item_code
          AND L.continent = ‘America’
          AND T.quarter = ‘1’
          AND I.kind = ‘Printer’;
```

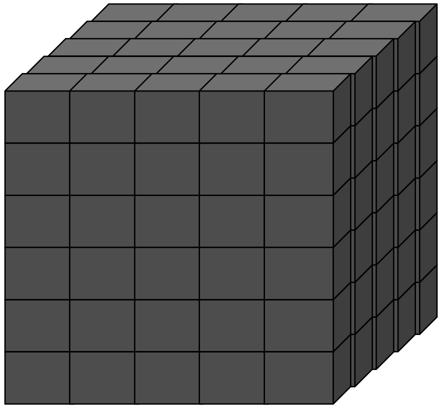
OLAP Cube for Computer Sales



SQL statement for slice 2

```
SELECT    units_produced
FROM      location L, time T, item I, facts F
WHERE     F.location_code = L.location_code
          AND F.time_code = T.time_code
          AND F.item_code = I.item_code
          AND T.month = 'May';
```

OLAP Cube for Computer Sales



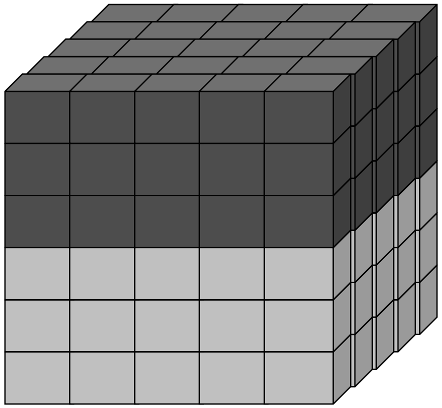
SQL statement for aggregative analysis

- **i.e. drill down and roll up**

- **e.g. analysis of production by year of production**

```
SELECT      SUM(units_produced)
FROM        location L, time T, item I, facts F
WHERE       F.location_code = L.location_code
            AND F.time_code = T.time_code
            AND F.item_code = I.item_code
GROUP BY    T.year;
```

OLAP Cube for Computer Sales



SQL statement for aggregative analysis

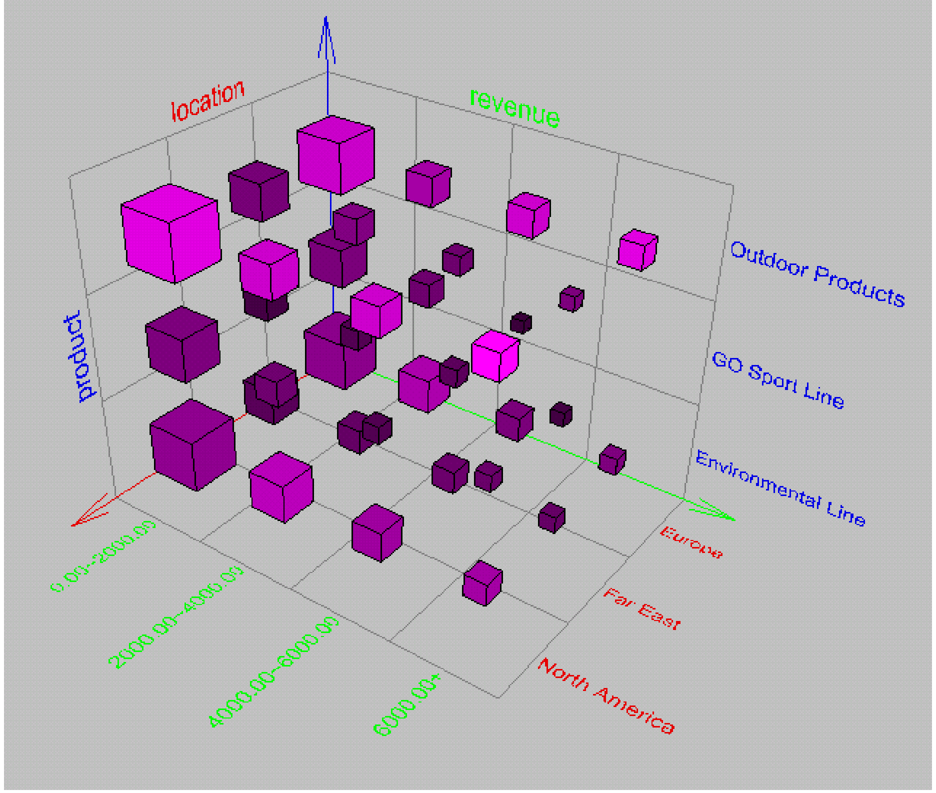
- i.e. drill down and roll up

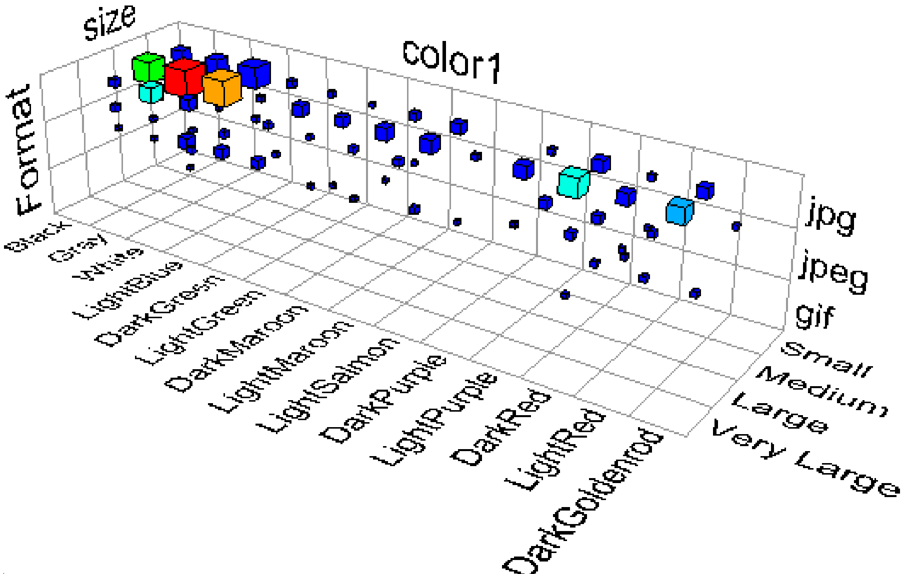
- e.g. analysis of production by quarter of production

```
SELECT      SUM(units_produced)
FROM        location L, time T, item I, facts F
WHERE       F.location_code = L.location_code
            AND F.time_code = T.time_code
            AND F.item_code = I.item_code
GROUP BY    T.quarter;
```

Browsing a Data Cube

- **Visual browsing**
  - **OLAP is used to pull out the data**
  - **data can be interactively manipulated**
    - **different angles and views**





Implementation of OLAP

Server Architectures

- Relational OLAP (ROLAP)
  - based on familiar, proven, and already known technologies
  - uses extended-relational DBMS and OLAP middle ware to store and manage warehouse data
    - usually stores aggregations also as relations
  - provides
    - optimization of DBMS backend
    - implementation of aggregation navigation logic
    - and some additional tools and services
  - good scalability
    - relational DBMS are very advanced technology, which is proven to be able to handle larger volumes of data

Implementation of OLAP

Server Architectures

- Multidimensional OLAP (MOLAP)
  - uses n-dimensional array based multidimensional storage engine and OLAP middle ware to manage warehouse data
    - multidimensional queries map to server capabilities in a straightforward way through direct addressing
  - has poor storage and performance utilization for sparse data
  - very good query performance by pre-calculation of transactional data
    - pre-calculates and stores every measure at every hierarchy summary level at load time and stores them for immediate retrieval using indexing
    - full pre-calculation requires an enormous amount of overhead both in processing time and in storage

Implementation of OLAP

Server Architectures

- **Hybrid OLAP (HOLAP)**
  - **user decides how to used multidimensional vs. relational models**
    - **e.g., relational for low level data, arrays for high-level data**

The assumption is that a data warehouses stores huge volumes of data

- **therefore, methodologies for efficient cube computation and indexing are necessary**

Efficiency in OLAP

Several step can be taken to improve performance of queries in OLAP:

- **materialization of cuboids**
  - **e.g. the most frequently accessed cuboids are materialized**
- **indexing**
  - **bitmap indexing**
    - **allows for very efficient search in data cuboids**
  - **join indexing**
    - **used for cross table searchers**
    - **most commonly used to join fact table with a dimension table in the start schema**

Materialization of a Data Cube

Full materialization

- physically materialize the whole data cube
- fastest query response, but requires heavy pre-computing and very large storage space
  - it is unrealistic to pre-compute and materialize all of the cuboids that can be generated for a given data cube
  - usually this approach is too expensive

No materialization

- nothing is materialized
- slowest query response, always requires dynamic query evaluation, but less storage space
  - very slow response time for complex queries causes necessity for some materialization

Materialization of a Data Cube

Partial materialization

- selected parts of a data cube are materialized
- gives a balance between the response time and required storage space
- requires
  - identification of a the subset of cuboids that will be materialized
  - exploitation of the materialized cuboids during query processing
  - efficient updating of the materialized cuboids during each load and refresh

Indexing in OLAP

Bitmap indexing

- index is performed on chosen columns
  - each value in the column is represented by a bit vector
    - the length of the bit vector is equal to the number of distinct records in the base table
    - the i^{th} bit is set if the i^{th} row of the base table has the value for the indexed column
- join and aggregation operators are reduced to bit arithmetic
  - and bit operations are very fast, even faster than hash and tree indexing
- works best for low cardinality domains
  - low number of values for an attribute
  - for high cardinality domains it may be adapted using compression techniques

Indexing in OLAP

item dimension table

item_code	kind	brand
1	DVD drive	HP
2	DVD drive	Intel
3	HDD	HP
4	HDD	Seagate
5	DVD drive	Samsung
6	HDD	Intel
7	HDD	Seagate

Bitmap indexing

– example

index on kind

record_code	DVD drive	HDD
1	1	0
2	1	0
3	0	1
4	0	1
5	1	0
6	0	1
7	0	1

index on brand

record_code	HP	Intel	Seagate	Samsung
1	1	0	0	0
2	0	1	0	0
3	1	0	0	0
4	0	0	1	0
5	0	0	0	1
6	0	1	0	0
7	0	0	1	0

to finding all rows where brand is either HP or Intel

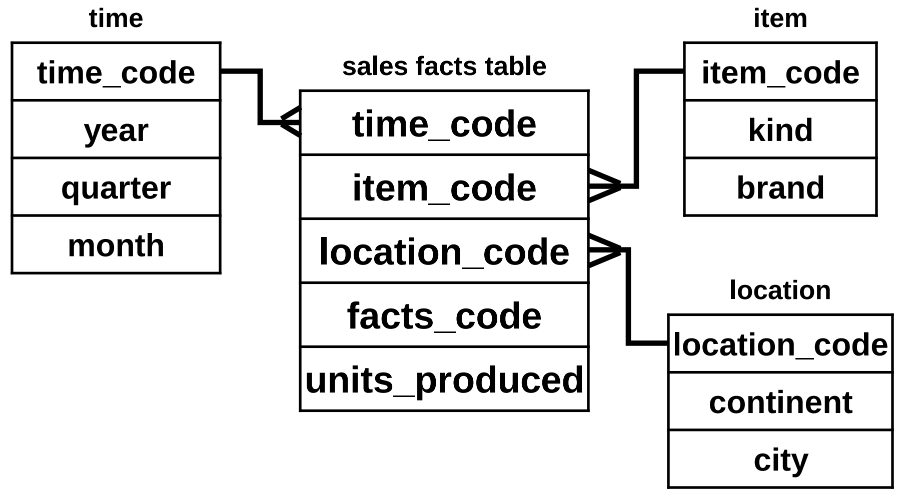
- **1000 OR 0100 = 1100**
- **thus rows 1, 2, 3, and 6 are selected**

Indexing in OLAP

Join indexing

- traditional indices map the values of an attribute to a list of record IDs
- join indices are used to register the joinable rows of two relations
  - they are used to speed up relational join, which is a very costly operation
  - applicable in data warehouses because of their design
    - they relate the values of the dimensions of a star schema to rows in the fact table
    - they can also relate multiple dimension tables
      - » composite join indices, which are used to select interesting cubes

Indexing in OLAP



Join indexing

- example

item dimension table

item_code	kind	brand
…	…	…
5	Printer	…
6	Printer	…
…	…	…
15	Printer	…
…	…	…

facts table

time_code	item_code	location_code	facts_code	units_produced
…	…	…	1	…
…	5	3	2	…
…	6	…	3	…
…	…	13	4	…
…	…	12	5	…
…	15	…	6	…
…	…	…	…	…

location dimension table

location_code	continent	city
…	…	…
3	America	…
…	…	…
12	America	…
13	America	…

join index for kind/facts_code

kind	facts_code
…	…
Printer	2
Printer	3
Printer	6
…	…

join index for kind/location/facts_code

kind	location	facts_code
…	…	…
Printer	America	2
…		…

composite join index

join index for continent/facts_code

continent	facts_code
…	…
America	2
America	5
America	4
…	…

Desired Features of an OLAP tool

How do we decide if a particular software tool is an OLAP tool?

- **many vendors claim to have 'OLAP compliant' products, but we should not rely on the vendors' own descriptions**

- **the FASMI test summarizes the OLAP definition in just five key words**
  Fast Analysis of Shared Multidimensional Information
  - **it was first used in early 1995 and has now been widely adopted and is cited in over 120 Web sites in about 30 countries**

taken from http://www.olapreport.com/fasmi.htm

Desired Features of an OLAP tool

FASMI test

- **Fast**
  - **the system must deliver most responses to users within about five seconds, with the simplest analyses taking no more than one second and very few taking more than 20 seconds**
    - **slow query response is consistently the most often-cited technical problem with OLAP products**
      - » **that is the result generated by the OLAP Survey 2 based on responses from 669 user organizations, see at http://www.survey.com/products/olap2/**
- **Analysis**
  - **the system must be able to cope with any business logic and statistical analysis that is relevant for the user of the system and application, and keep it easy enough for the target user**
    - **it must allow to define new ad hoc calculations, and to report on the data in any desired way, without having to program**

taken from http://www.olapreport.com/fasmi.htm

Desired Features of an OLAP tool

FASMI test

- Shared
  - the system must implement
    - security mechanisms necessary to provide confidentiality (possibly down to cell level)
    - concurrent update locking capabilities (if multiple write access is needed)
- Multidimensional
  - the key requirement since OLAP is multidimensional
  - the system must provide a multidimensional conceptual view of the data, including full support for hierarchies and multiple hierarchies
    - we assume that this is the most logical way to analyze businesses and organizations

taken from http://www.olapreport.com/fasmi.htm

Desired Features of an OLAP tool

FASMI test

- Information
  - information is defined as all of the data and derived information needed, wherever it is and however much is relevant for the application
  - an OLAP tool is evaluated in terms of how much input data it can handle, not how many Gb it takes to store the data
    - the largest OLAP products can hold at least a thousand times as much data as the smallest

taken from http://www.olapreport.com/fasmi.htm

Top 10 Commercial OLAP Tools

Recent report by www.olapreport.com gives top 10 commercial OLAP products together with their marker shares

1. **Microsoft (28.0%)**
2. **Hyperion (19.3%)**
3. **Cognos (14.0%)**
4. **Business Objects (7.4%)**
5. **MicroStrategy (7.3%)**
6. **SAP (5.9%)**
7. **Cartesis (3.8%)**
8. **Systems Union/MIS AG (3.4%)**
9. **Oracle (3.4%)**
10. **Applix (3.2%)**

taken from http://www.olapreport.com/fasmi.htm

OLAP Products

Specific commercial OLAP products include

- **Microsoft SQL Server 2000 and 2005 Analysis Services**
- **Hyperion Essbase 7X**
- **Cognos PowerPlay 7.3**
- **BusinessObjects XI**
- **MicroStrategy 7i**
- **SAP BW 3.1**
- **Cartesis Magnitude 7.4**
- **Oracle Express and the OLAP Option 6.4**
- **Applix TM1 8.3**

Also, a number of open source OLAP products, including Mondrian and Palo, were developed

taken from http://www.olapreport.com/fasmi.htm

Data Warehousing and OLAP for Data Mining

Data warehouse can be applied to perform three kinds of tasks

- information processing
  - by querying, providing basic statistical analysis, and reporting using tables, charts and graphs
- analytical processing
  - multidimensional analysis of data warehouse data by using basic OLAP operations, like slice and dice, drilling, pivoting, etc.
- Data Mining
  - knowledge discovery in terms of finding hidden patterns
  - supports discovery of associations, constructing analytical models, performing classification and prediction, and presenting the mining results using visualization tools

Data Warehousing and OLAP for Data Mining

Why Data mining systems should use Data Warehousing technology?

- **data warehouses contain high quality data**
  - **integrated, cleaned, and consistent data which is a high-quality source for data mining**
- **data warehouses provide information processing infrastructure like:**
  - **Open Database Connectivity (ODBC) that is a widely accepted application programming interface (API) for database access**
  - **Object Linking and Embedding for Databases (OLEDB) is a COM-based data access object that provides access to data in DBs**
  - **OLAP tools**
  - **reporting capabilities**
  - **web accessing**

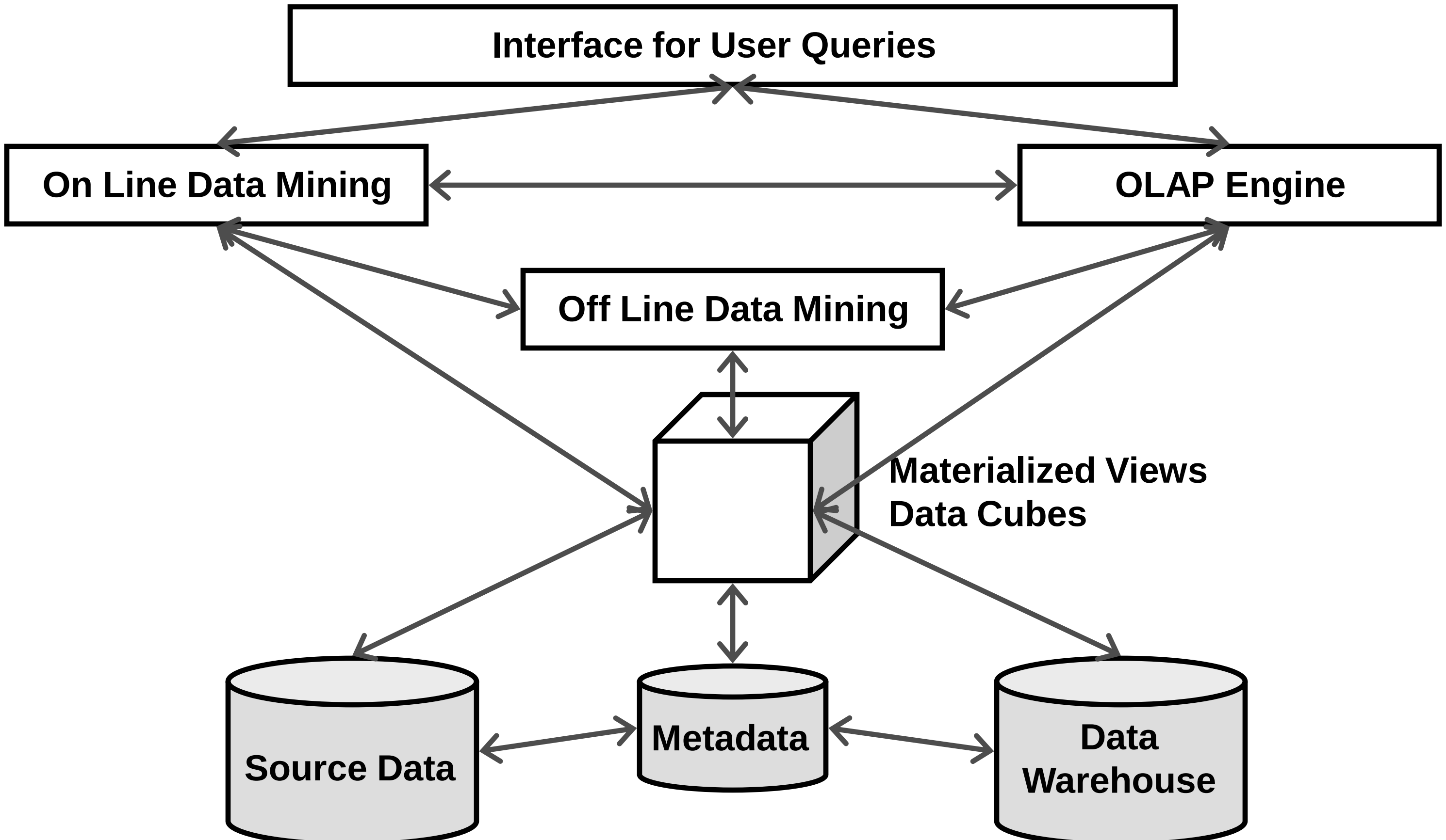
Data Warehousing and OLAP for Data Mining

Why Data mining systems should use Data Warehousing technology?

- **they provide OLAP-based exploratory data analysis**
  - **data can be pulled out of the database by means of drilling, dicing, pivoting, etc. operators**
    - **they enable very efficient selection of relevant portions of data for mining**

Data Warehousing and OLAP for Data Mining

Integrated architecture for OLAP and data mining in a data warehouse environment



References

Berson, A. and Smith, S.J. 1997. *Data Warehousing, Data Mining and OLAP*, McGraw-Hill

Codd, E., Codd, S. and Salley, C. 1993. Beyond Decision Support, *Computer World*, 27(30)

Inmon, W. 2005. *Building a Data Warehouse*, 4th edition, John Wiley and Sons

Jarke, M., Lenzerini, M., Vassiliou, Y. and Vassiliadis, P. 2003. *Fundamentals of Data Warehouses*, Springer

Thomsen, E. 1997. *OLAP Solutions: Building Multidimensional Information Systems*, John Wiley and Sons