Database Management Systems
CSE 594

Lecture #1

April 4^{th}, 2002

Staff

- Instructor: Alon Halevy
  - Sieg, Room 310, alon@cs.washington.edu
  - Office hours: Thursday before class
  - or by appointment, email.
- TA: Maya Rodrig
  - Sieg 226a, rodrig@cs.washington.edu
  - Office hours: TBA

Communications

- Web page: http://www.cs.washington.edu/594/
- Mailing list: send email to majordomo@cs saying (in body of email): subscribe cse594

Goals of the Course

- Purpose:
  - Principles of building database applications
  - Foundations of database management systems.
  - Issues in building database systems.
  - Have fun: *databases are not just bunches of tuples.*
  - *Not an introduction to the nitty gritty of any specific commerical system.*

Grading

- Paper homeworks: 30%
  - Very little regurgitation.
  - Meant to be challenging (I.e., fun).
- Programming project: 30%
  - Work in pairs.
  - Build a database application
- Final Exam: 30% (June 14^{th}).
- Intangibles (e.g., participation): 10%

Textbook

- **Database Systems: The Complete Book,** by Garcia-Molina, Ullman and Widom, 2002
- Comments on the textbook.

Other Texts

- Database Management Systems, *Ramakrishnan*
  - very comprehensive
- Fundamentals of Database Systems, *Elmasri and Navathe*
  - very widely used

- Foundations of Databases, *Abiteboul, Hull and Vianu*
  - Mostly theory of databases
- Data on the Web, *Abiteboul,Buneman,Suciu*
  - XML and other new/advanced stuff

Available on reserve, at the library

Prerequisites

Real Prerequisites

- Operating systems
- Data structures and algorithms
- Distributed systems
- Complexity theory
- Mathematical Logic
- Knowledge Representation
- User interface design
- Programming languages
- Artificial Intelligence (Search)
- *Greek, Hebrew, French*

Why use a DBMS?

Suppose we are building a system to store the information pertaining to the university.

Several questions arise:

- how do we store the data? (file organization, etc.)
- how do we query the data? (write programs…)
- make sure that updates don't mess things up?
- Provide different views on the data? (registrar versus students)
- how do we deal with crashes?

Way too complicated! Go buy a database system!

Functionality of a DBMS

- Persistent storage management
- Transaction management
- Resiliency: recovery from crashes.
- Separation between logical and physical views of the data.
  - High level query and data manipulation language.
  - Efficient query processing
- Interface with programming languages

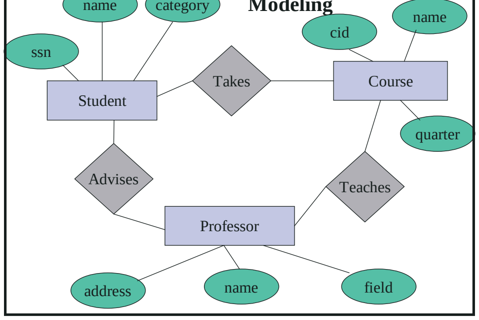
Bird’s Eye View of

- How to build a database application
- The different components of a database system.

Building an Application with a Database System

- Requirements modeling (conceptual, pictures)
  - Decide what entities should be part of the application and how they should be linked.
- Schema design and implementation
  - Decide on a set of tables, attributes.
  - Define the tables in the database system.
  - Populate database (insert tuples).
- Write application programs using the DBMS
  - way easier now that the data management is taken care of.

Conceptual Modeling



Schema Design and Implementation

- Tables:

Students:

SSN	Name	Category
123-45-6789	Charles	undergrad
234-56-7890	Dan	grad
	…	…

Takes:

SSN	CID
123-45-6789	CSE444
123-45-6789	CSE444
234-56-7890	CSE142
	…

Courses:

CID	Name	Quarter
CSE444	Databases	fall
CSE541	Operating systems	winter

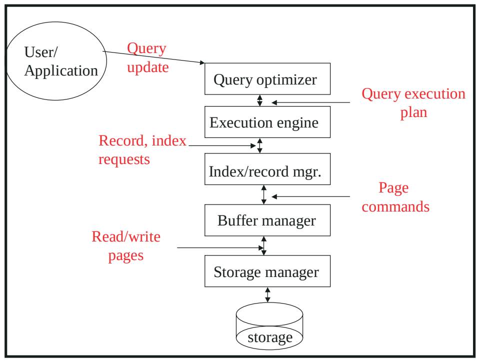
- Separates the logical view from the physical view of the data.

Querying a Database

- Find all courses that “Mary” takes
- S(tructured) Q(uery) L(anguage)

```
select  C.name
from    Students S, Takes T, Courses C
where   S.name=“Mary” and
        S.ssn = T.ssn and T.cid = C.cid
```

- Query processor figures out how to answer the query efficiently.



Storage Management

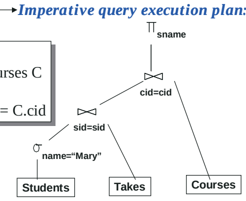
- Becomes a hard problem because of the interaction with the other levels of the DBMS:
  - What are we storing?
  - Efficient indexing, single and multi-dimensional
  - Exploit “semantic” knowledge
- Issue: interaction with the operating system. Should we rely on the OS?

Query Optimization

Goal:

Declarative SQL query* → *Imperative query execution plan:

select C.name
from Students S, Takes T, Courses C
where S.name=“Mary” and
S.ssn = T.ssn and T.cid = C.cid



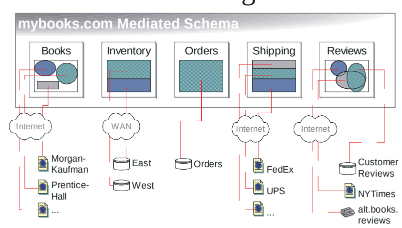
Plan: tree of Relational Algebra operators, choice of algorithms at each operator

Ideally: Want to find best plan. Practically: Avoid worst plans!

TP and Recovery

- For efficient use of resources, we want concurrent access to data.
- Systems sometimes crash.
- A “real” database guarantees **ACID**:
  - Atomicity: all or nothing of a transaction.
  - Consistency: always leave the DB consistent.
  - Isolation: every transaction runs as if it’s the only one in the system.
  - Durability: if committed, we really mean it.
- *Do we really want **ACID**?*

Data Integration



Uniform query capability across *autonomous*, *heterogeneous* data sources on LAN, WAN, or Internet

XML: Semi-structured Data

eXtensible Markup Language:

- *Emerging format for data exchange on the web and between applications.*

```
<db>
 <book>
    <title>Complete Guide to DB2</title>
    <author>Chamberlin</author>
 </book>
 <book>
    <title>Transaction Processing</title>
    <author>Bernstein</author>
    <author>Newcomer</author>
 </book>
 <publisher>
   <name>Morgan Kaufman</name>
   <state>CA</state>
 </publisher>
</db>
```

Traditional and Novel Data Management

- Traditional Data Management:
  - relational data for enterprise applications
  - storage
  - query processing/optimization
  - transaction processing
- Novel Data Management:
  - Integration of data from multiple databases, warehousing.
  - Data management for decision support, data mining.
  - Exchange of data on the web: XML.

The Study of DBMS

- Several aspects:
  - Modeling and design of databases
  - Database programming: querying and update operations
  - Database implementation
- DBMS study cuts across many fields of Computer Science: OS, languages, AI, Logic, multimedia, theory...

Database Industry

- Relational databases are a great success of theoretical ideas.
- $20B industry.
- Main players: Oracle, IBM, MS, Sybase, Informix
- Trends:
  - warehousing and decision support
  - data integration
  - XML, XML, XML.

Course (Rough) Outline

- The basics: (quickly)
  - Conceptual design
  - The relational model
  - SQL
  - Views, integrity constraints
- XML
- Physical representation:
  - Index structures.

Course Outline (cont)

- Query execution:
  - Algorithms for joins, selections, projections.
- Query Optimization
- Data Integration
- semi-structured data
- Transaction processing and recovery (not much, really)

Projects

- Goal: identify and solve a problem in database systems.
- (almost) anything goes.
- Groups of 2-3
- Groups assembled end of week 2;
- Proposals, end of week 3.
- Specs – end of week 5
- End-to-end skeleton – end of week 7.
- Start Early.
- Be creative
- Demos on last week

Database Design

Building an Application with a DBMS

- Requirements modeling (conceptual, pictures)
  - Decide what entities should be part of the application and how they should be linked.
- Schema design and implementation
  - Decide on a set of tables, attributes.
  - Define the tables in the database system.
  - Populate database (insert tuples).
- Write application programs using the DBMS
  - way easier now that the data management is taken care of.

Outline

- ODL - Object Definition Language (2.1)
- E/R - Entity relationship diagrams (2.2)
- Design Principles (2.3)

Database Design

- Why do we need it?
  - Agree on structure of the database before deciding on a particular implementation.
- Consider issues such as:
  - What entities to model
  - How entities are related
  - What constraints exist in the domain
  - How to achieve ***good*** designs

Database Design Formalisms

1. Object Definition Language (ODL):
   - Closer in spirit to object-oriented models
2. Entity/Relationship model (E/R):
   - More relational in nature.

- Both can be translated (semi-automatically) to relational schemas
- ODL to OO-schema: direct transformation (C++ or Smalltalk based system).

1. Object Definition Language

- ODL is part of ODMG
- superset of Corba's IDL
- Resembles C++ (and Smalltalk).

ODL Principles

- Basic design paradigm in ODL:
  - Model ***objects*** and their ***properties.***
- For abstraction purposes:
  - Group objects into ***classes.***
- What qualifies as a ***good*** class?
  - Objects should have common properties.

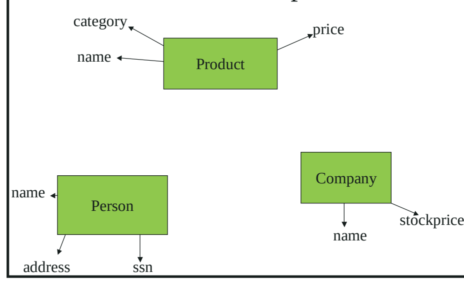
ODL Class Declarations

Class declaration:

```
Interface <name> {
   attributes:  <type>  <name>;
   relationships  <range type>  <name>;
   methods <type> <name>(param)
}
```

Methods: arbitrary function, of little concern for us here

ODL Example

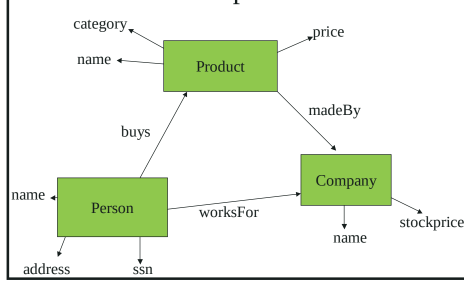


ODL Declarations

```
Interface Product {
      attribute  string name;
      attribute  float  price;
      attribute  enum Categories
                     {electronics, communications, sports …} category
      }
Interface Company {
      attribute  string  name;
      attribute  float   stockprice;
      }
Interface Person {
      attribute integer ssn;
      attribute string  name;
      attribute struct Address {string street, string city} address; }
```

So far just simplified C++ with slightly different syntax

ODL Example Extended



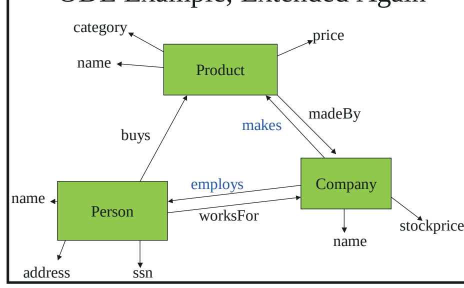
ODL Declarations, Extended

```
Interface Product {
      attribute  string name;
      attribute  float  price;
      attribute  enum Categories
                     {electronics, communications, sports …} category;
      relationship  <Company>  madeBy;
      }

Interface Person {
      attribute integer ssn;
      attribute string  name;
      attribute Struct Address {string street, string city} address;
      relationship  set <Product>  buys;
      relationship  set <Company>  worksFor;}
```

relationship corresponds somewhat to pointers in C++

ODL Example, Extended Again



ODL Declarations, Extended Again

```
Interface Company {
      attribute   string  name;
      attribute   float    stockprice;

      relationship  set <Product> makes
                         inverse  Product::madeBy;

      relationship   set <Person>  employs
                         inverse  Person::worksFor;
      }
```

Types in ODL

Basic types:

Atomic types (e.g., string, integer, …)
Interface types (e.g., Person, Product, Company)

Constructors:

collection types:
- Set: {1, 5, 6}
- Bag: {1, 1, 5, 6, 6 }
- List: [1, 5, 6, 1, 6]
- Array: integer[17]

structured types:
- Struct {string street, string city, integer zipcode}

Collection Types

- Sets:
  - order, number of occurrences don't matter
  - {4,7,9} = {7,9,7,4} = {9,4,7}
- Bags:
  - number of occurrences matter, order not:
  - {7,9,7,4}={7,7,9,4}, is different from {4,7,9}
- Lists:
  - order, number of occurrences matter:
  - [4,7,9] different from [9,4,7]

Allowable Types in ODL

For attributes: atomic/struct, or collection of atomic/struct

OK: string, set<integer>

Not OK: Product, set<set<integer>>

For relationships: interface, or collection of interface.

OK: Product, set<Product>, list<Person>

Not OK: **struct** {pname Product, cname Company}
set<bag<Product>>
integer

2. Entity / Relationship Diagrams

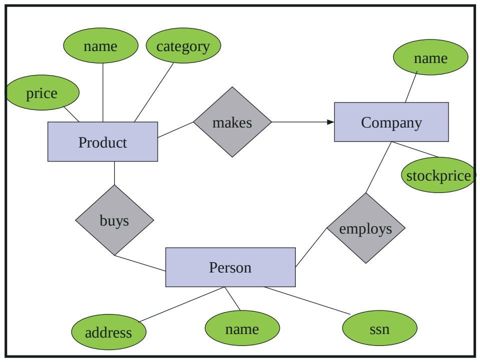
Objects → entities
Classes → entity sets   Product

Attributes are like in ODL.   address

Relationships: like in ODL except   buys

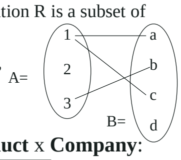
- first class citizens (not associated with classes)

- not necessarily binary



What is a Relation ?

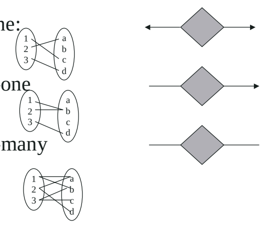
- A mathematical definition:
  - if A, B are sets, then a relation R is a subset of A x B
- A={1,2,3}, B={a,b,c,d},
  R = {(1,a), (1,c), (3,b)}



- **makes** is a subset of **Product** x **Company**:

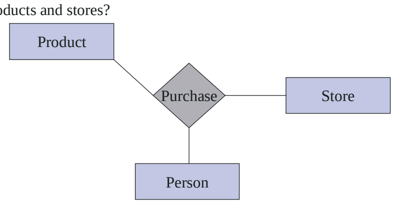
Multiplicity of E/R Relations

- one-one:
- many-one
- many-many



Multi-way Relationships

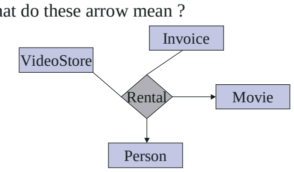
How do we model a purchase relationship between buyers, products and stores?



Can still model as a mathematical set (how ?)

Arrows in Multiway Relationships

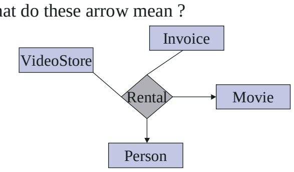
Q: what do these arrow mean ?



A: store, person, invoice determines movie and store, invoice, movie determines person

Arrows in Multiway Relationships

Q: what do these arrow mean ?

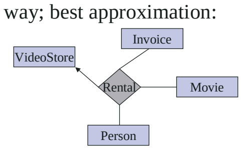


A: store, invoice determines movie and store, invoice determines person

Arrows in Multiway Relationships

Q: how do I say: “invoice determines store” ?

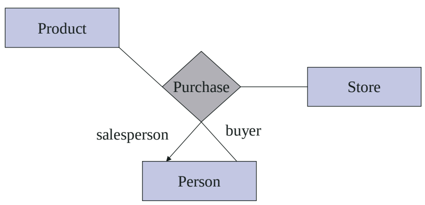
A: no good way; best approximation:



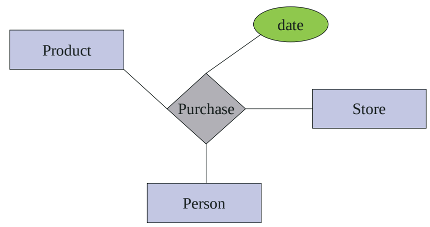
Q: Why is this incomplete ?

Roles in Relationships

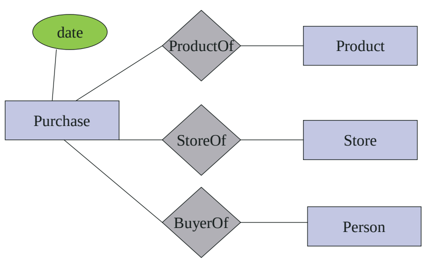
What if we need an entity set twice in one relationship?



Attributes on Relationships

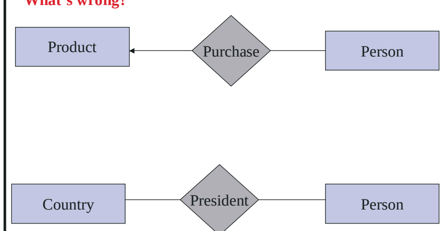


Converting Multi-way Relationships to Binary



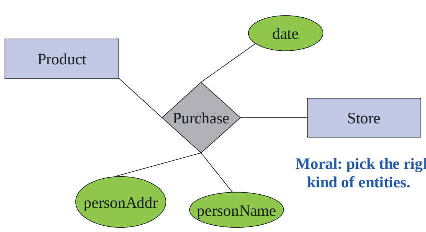
3. Design Principles

What's wrong?



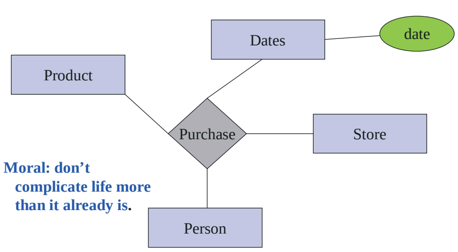
Moral: be faithful!

Design Principles: What's Wrong?



Moral: pick the right kind of entities.

Design Principles: What's Wrong?



Moral: don't complicate life more than it already is.