CPSC 504 – Background
(aka, all you need to know about databases for this course in two lectures)

Rachel Pottinger

September 13 and 18, 2006

Administrative notes

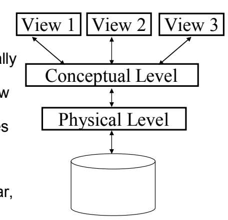
- Please note you're supposed to sign up for one topic presentation and one discussion… for *different* topics (send me mail)
- Please sign up for the mailing list
- HW 1 is on the web, due beginning of class a week from today

Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

Levels of Abstraction

- A major purpose of a DB system is to provide an abstract view of the data.
- Three abstraction levels:
  - **Physical level**: how data are actually stored
  - **Conceptual (or Logical) level**: how data is perceived by the users
  - **External (or View) level**: describes different part of the database to different users
    - convenience, security, etc.
  - Compare views of student, registrar, & database admin.



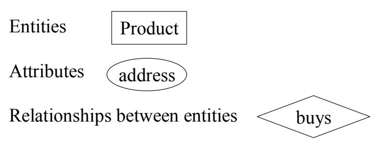
Schema and Instances

- We create the **schema** – the logical structure of the database (e.g., students take courses)
  - **Conceptual (or logical) schema**: db design at the logical level
  - **Physical schema**: db design at the physical level; indexes, etc
- Later we'll populate **instances** – the actual content of the database at a particular point in time
  - E.g., currently there are no grades for CPSC 504 Winter term 2
- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the conceptual schema
- **Logical Data Independence** – Provided by the views
  - Ability to change the conceptual scheme without changing the applications

Conceptual Database Design

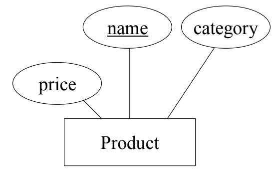
- What are the entities and relationships in the enterprise?
  - Entities are usually nouns, e.g., "course" "prof"
  - Relationships are statements about 2 or more objects. Often, verbs., e.g., "a prof teaches a course"
- What information about these entities and relationships should we store in the database?
- What integrity constraints or other rules hold?
- In relational databases, this data is generally encoded in an **Entity-Relationship (ER) Diagram**

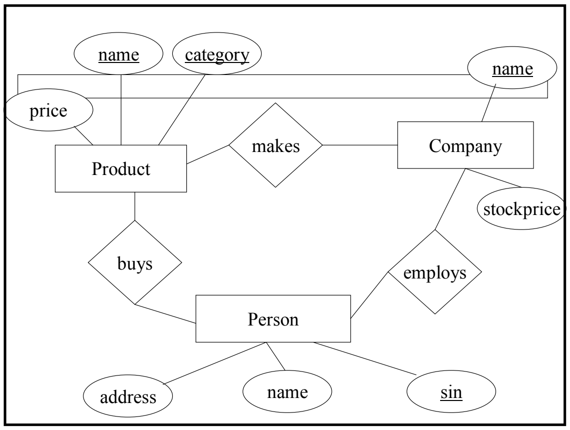
Entity / Relationship Diagrams



Keys in E/R Diagrams

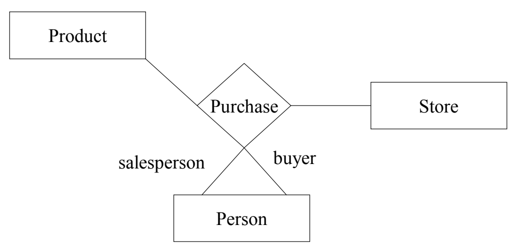
- Every entity set must have a key which is identified by an underline



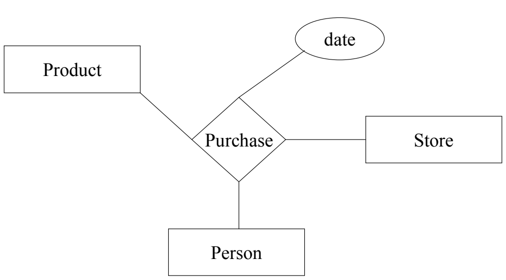


Roles in Relationships

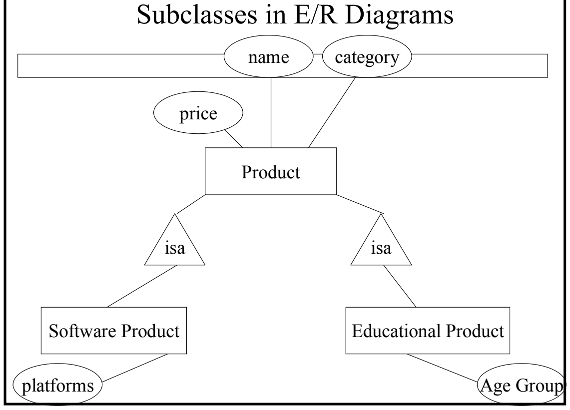
What if we need an entity set twice in one relationship?



Attributes on Relationships



Subclasses in E/R Diagrams



Summarizing ER diagrams

- Basics: entities, relationships, and attributes
- Also showed inheritance
- Has things other things like cardinality
- Used to design databases...

But how do you store data in them?

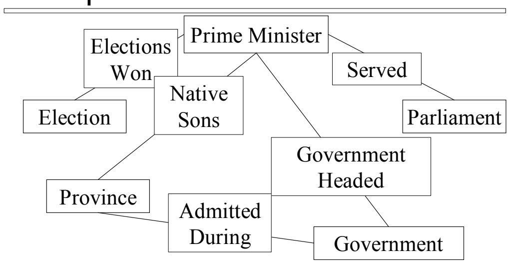
Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What's in a relational schema?
  - From ER to relational
  - Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

How did we get the relational model?

- Prior to the relational model, there were two main contenders
  - Network databases
  - Hierarchical databases
- Network databases had a complex data model
- Hierarchical databases integrated the application in the data model

Example Hierarchical Model

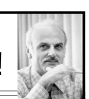


Example IMS (Hierarchical) query: Print the names of all the provinces admitted during a Liberal Government

```
DLITPLI:PROCEDURE (QUERY_PCB) OPTIONS (MAIN);

 DECLARE QUERY_PCB POINTER;
 /*Communication Buffer*/
 DECLARE 1 PCB BASED(QUERY_PCB),
  2 DATA_BASE_NAME CHAR(8),
  2 SEGMENT_LEVEL CHAR(2),
  2 STATUS_CODE CHAR(2),
  2 PROCESSING_OPTIONS CHAR(4),
  2 RESERVED_FOR_DLI FIXED BIRARY(31,0),
  2 SEGMENT_NAME_FEEDBACK CHAR(8)
  2 LENGTH_OF_KEY_FEEDBACK_AREA FIXED BINARY(31,0),
  2 NUMBER_OF_SENSITIVE_SEGMENTS FIXED BINARY(31,0),
  2 KEY_FEEDBACK_AREA CHAR(28);
 /* I/O Buffers*/
 DECLARE PRES_IO_AREA CHAR(65),
  1 PRESIDENT DEFINED PRES_IO_AREA,
  2 PRES_NUMBER CHAR(4),
  2 PRES_NAME CHAR(20),
  2 BIRTHDATE CHAR(8)
  2 DEATH_DATE CHAR(8),
  2 PARTY CHAR(10),
  2 SPOUSE CHAR(15);
 DECLARE SADMIT_IO_AREA CHAR(20),
  1 province_ADMITTED DEFINED SADMIT_IO_AREA,
  2 province_NAME CHAR(20);
 /* Segment Search Arguments */
 DECLARE 1 PRESIDENT_SSA STATIC UNALIGNED,
  2 SEGMENT_NAME CHAR(8) INIT('PRES '),
  2 LEFT_PARENTHESIS CHAR (1) INIT('('),
  2 FIELD_NAME CHAR(8) INIT ('PARTY '),
  2 CONDITIONAL_OPERATOR CHAR (2) INIT('='),
  2 SEARCH_VALUE CHAR(10) INIT ('Liberal '),
2 RIGHT_PARENTHESIS CHAR(1) INIT(')');
 DECLARE 1 province_ADMITTED_SSA STATIC UNALIGNED,
  2 SEGMENT_NAME CHAR(8) INIT('SADMIT ');
 /* Some necessary variables */
 DECLARE GU CHAR(4) INIT('GU '),
  GN CHAR(4) INIT('GN '),
  GNP CHAR(4) INIT('GNP '),
  FOUR FIXED BINARY (31) INIT (4),
  SUCCESSFUL CHAR(2) INIT(' '),
  RECORD_NOT_FOUND CHAR(2) INIT('GE');
 /*This procedure handles IMS error conditions */
 ERROR;PROCEDURE(ERROR_CODE);
  *
  *
  *
 END ERROR;
 /*Main Procedure */
 CALL PLITDLI(FOUR,GU,QUERY_PCB,PRES_IO_AREA,PRESIDENT_SSA);
 DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
  CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
  DO WHILE(PCB.STATUS_CODE=SUCCESSFUL);
   PUT EDIT(province_NAME)(A);
  CALL PLITDLI(FOUR,GNP,QUERY_PCB,SADMIT_IO_AREA,province_ADMITTED_SSA);
   END;
  IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
   THEN DO;
    CALL ERROR(PCB.STATUS_CODE);
    RETURN;
    END;
   CALL PLITDLI(FOUR,GN,QUERY_PCB,PRES_IO_AREA,PRESDIENT_SSA);
  END;
  IF PCB.STATUS_CODE NOT = RECORD_NOT_FOUND
   THEN DO;
    CALL ERROR(PCB.STATUS_CODE);
    RETURN;
    END;
END DLITPLI;
```

Relational model to the rescue!



- Introduced by Edgar Codd (IBM) in 1970
- Most widely used model today.
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- Competitor: object-oriented model
  - ObjectStore, Versant, Ontos
  - A synthesis emerging: *object-relational model*
    - Informix Universal Server, UniSQL, O2, Oracle, DB2
- Recent competitor: XML data model

Key points of the relational model

- Exceedingly simple to understand – main abstraction is represented as a table
- Simple query language separate from application language
- Lots of bells and whistles to do complicated things

Structure of Relational Databases

- ***Relational database**:* a set of ***relations***
- ***Relation:*** made up of 2 parts:
  - ***Schema*** : specifies name of relation, plus name and **domain** (type) of each **field** (or **column** or **attribute**).
    - e.g., Student (*sid*: string, *name*: string, *address*: string, *phone*: string, *major*: string).
  - ***Instance*** : a ***table***, with rows and columns. ***#Rows = cardinality, #fields = dimension / arity / degree***
- ***Relational Database Schema:*** collection of schemas in the database
- ***Database Instance:*** a collection of instances of its relations (e.g., currently no grades in CPSC 504)

Example of a Relation Instance

Product Attribute names or columns

Name	Price	Category	Manufacturer
gizmo	$19.99	gadgets	GizmoWorks
Power gizmo	$29.99	gadgets	GizmoWorks
SingleTouch	$149.99	photography	Canon
MultiTouch	$203.99	household	Hitachi

Tuples or rows Relation or table

Order of rows isn't important

Formal Definition:
Product(Name: string, Price: double, Category: string, Manufacturer: string)

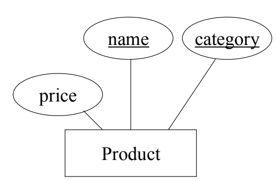
Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What's in a relational schema?
  - From ER to relational
  - Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation

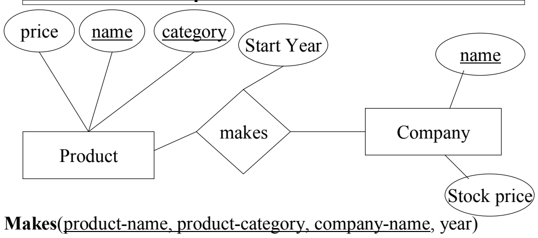
Entity Set to Relation



Product(<u>name, category</u>, price)

name	category	price
gizmo	gadgets	$19.99

Relationships to Relations

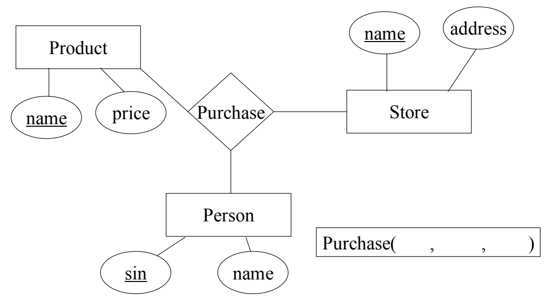


Makes(product-name, product-category, company-name, year)

Product-name	Product-Category	Company-name	Starting-year
gizmo	gadgets	gizmoWorks	1963

(watch out for attribute name conflicts)

Multi-way Relationships to Relations



Overview of the next two classes

- Entity Relationship (ER) diagrams
- Relational databases
  - How did we get here?
  - What's in a relational schema?
  - From ER to relational
  - Query Languages
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

Relational Query Languages

- A major strength of the relational model: simple, powerful *querying* of data.
- Queries can be written intuitively; DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.
- We'll look at 3: relational algebra, SQL, and datalog

Querying – Relational Algebra

- *Select* (σ)- chose tuples from a relation
- *Project* (π)- chose attributes from relation
- *Join* ($\bowtie$) - allows combining of 2 relations
- *Set-difference* (—) Tuples in relation 1, but not in relation 2.
- *Union* ($\cup$)
- *Cartesian Product* (×) Each tuple of R1 with each tuple in R2

Find products where the manufacturer is GizmoWorks

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Selection:

$\sigma_{Manufacturer = GizmoWorks}$Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Find the Name, Price, and Manufacturers of products whose price is greater than 100

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Selection + Projection:

$\pi_{\text{Name, Price, Manufacturer}}(\sigma_{\text{Price} > 100}\text{Product})$

Name	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Find the product names and price of products that cost less than $200 and have manufacturers where there is a Company that has a CName that matches the manufacturer, and its country is Japan

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

$\pi_{\text{Name, Price}}((\sigma_{\text{Price} < 200}\text{Product}) \bowtie_{\text{Manufacturer} = \text{Cname}} (\sigma_{\text{Country} = \text{'Japan'}}\text{Company}))$

Name	Price
SingleTouch	$149.99

When are two relations related?

- You guess they are
- I tell you so
- Constraints say so
  - A *key* is a set of attributes whose values are unique; we underline a key
    Product(<u>Name</u>, Price, Category, <u>Manfacturer</u>)
  - Foreign keys are a method for schema designers to tell you so
    - A foreign key states that an attribute is a reference to the key of another relation
      ex: Product.Manufacturer is foreign key of Company
    - Gives information and enforces constraint

The SQL Query Language

- Developed by IBM (System R) in the 1970s
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

SQL

- Data Manipulation Language (DML)
  - Query one or more tables
  - Insert/delete/modify tuples in tables
- Data Definition Language (DDL)
  - Create/alter/delete tables and their attributes
- Transact-SQL
  - Idea: package a sequence of SQL statements → server

Querying – SQL

Standard language for querying and manipulating data

Structured **Q**uery **L**anguage

Many standards out there:
- ANSI SQL
- SQL92 (a.k.a. SQL2)
- SQL99 (a.k.a. SQL3)
- Vendors support various subsets of these
- What we discuss is common to all of them

SQL basics

- Basic form: (many many more bells and whistles in addition)

Select attributes
From relations (possibly multiple, joined)
Where conditions (selections)

SQL – Selections

```
SELECT  *
FROM    Company
WHERE   country="Canada" AND stockPrice > 50
```

Some things allowed in the WHERE clause:

attribute names of the relation(s) used in the FROM.
comparison operators: =, <>, <, >, <=, >=
apply arithmetic operations: stockPrice*2
operations on strings (e.g., "||" for concatenation).
Lexicographic order on strings.
Pattern matching: s LIKE p
Special stuff for comparing dates and times.

SQL – Projections

Select only a subset of the attributes

```
SELECT  name, stock price
FROM    Company
WHERE   country="Canada" AND stockPrice > 50
```

Rename the attributes in the resulting table

```
SELECT  name AS company, stockPrice AS price
FROM    Company
WHERE   country="Canada" AND stockPrice > 50
```

SQL – Joins

```
SELECT  name, store
FROM    Person, Purchase
WHERE   name=buyer AND city="Vancouver"
                   AND product="gizmo"
```

Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Company (name, stock price, country)
Person(name, phone number, city)

Selection:
$\sigma_{Manufacturer = GizmoWorks}(Product)$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

What's the SQL?

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Selection + Projection:
$\pi_{Name, Price, Manufacturer}(\sigma_{Price > 100} Product)$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

What's the SQL?

Name	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

$\pi_{\text{Name, Price}}((\sigma_{\text{Price <= 200}}\text{Product}) \bowtie_{\text{Manufacturer = Cname}} (\sigma_{\text{Country = 'Japan'}}\text{Company}))$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What's the SQL?

Name	Price
SingleTouch	$149.99

Querying – Datalog
(Our final query language)

- Enables expressing recursive queries
- More convenient for analysis
- Some people find it easier to understand
- Without recursion but with negation it is equivalent in power to relational algebra and SQL
- Limited version of Prolog (no functions)

Datalog Rules and Queries

A datalog rule has the following form:

head :- atom1, atom2, …, atom,…

You can read this as

then :- if ...

Arithmetic comparison or interpreted predicate

ExpensiveProduct(N) :- Product(N,M,P) & P > $100

CanadianProduct(N) :- Product(N,M,P) & Company(M, "Canada", SP)

IntlProd(N) :- Product(N,M,P) & NOT Company(M, "Canada", SP)

Negated subgoal - also denoted by ¬

Conjunctive Queries

- A subset of Datalog
- Only relations appear in the right hand side of rules
- No negation
- Functionally equivalent to Select, Project, Join queries
- Very popular in modeling relationships between databases

Selection:
$\sigma_{\text{Manufacturer = GizmoWorks}}(\text{Product})$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

What's the Datalog?

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Selection + Projection:
$\pi_{\text{Name, Price, Manufacturer}} (\sigma_{\text{Price > 100}}\text{Product})$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

What's the Datalog?

Name	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

$\pi_{Name,Price}((\sigma_{Price <= 200} Product) \bowtie_{Manufacturer = Cname} (\sigma_{Country = 'Japan'} Company))$

Product

Name	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What's the Datalog?

Name	Price
SingleTouch	$149.99

Bonus Relational Goodness: Views

Views are stored queries treated as relations, Virtual views are not physically stored. Materialized views are stored

They are used (1) to define conceptually different views of the database and (2) to write complex queries simply.

View: purchases of telephony products:

```
CREATE VIEW telephony-purchases AS
  SELECT product, buyer, seller, store
  FROM Purchase, Product
  WHERE Purchase.product = Product.name
        AND Product.category = "telephony"
```

Summarizing Relational DBs

- Relational perspective: Data is stored in relations. Relations have attributes. Data instances are tuples.
- SQL perspective: Data is stored in tables. Tables have columns. Data instances are rows.
- Query languages
  - Relational algebra – mathematical base for understanding query languages
  - SQL – very widely used
  - Datalog – based on Prolog, very popular with theoreticians
- Views allow complex queries to be written simply

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

Object-Oriented DBMS's

- Started late 80's
- Main idea:
  - Toss the relational model!
  - Use the OO model – e.g. C++ classes
- Standards group: ODMG = Object Data Management Group.
- OQL = Object Query Language, tries to imitate SQL in an OO framework.

The OO Plan

ODMG imagines OO-DBMS vendors implementing an OO language like C++ with extensions (OQL) that allow the programmer to transfer data between the database and "host language" seamlessly.

A brief diversion: the impedance mismatch

OO Implementation Options

- Build a new database from scratch (O_2)
  - Elegant extension of SQL
  - Later adopted by ODMG in the OQL language
  - Used to help build XML query languages
- Make a programming language persistent (ObjectStore)
  - No query language
  - Niche market
  - ObjectStore is still around, renamed to Exelon, stores XML objects now

ODL

- ODL is used to define *persistent* classes, those whose objects may be stored permanently in the database.
  - ODL classes look like Entity sets with binary relationships, plus methods.
  - ODL class definitions are part of the extended, OO host language.

ODL – remind you of anything?

```
interface Person
   (extent People key sin)
{   attribute string  sin;
    attribute string  dept;
    attribute string  name;}

interface Course
   (extent Crs key cid)
{   attribute string cid;
    attribute string cname;
    relationship Person instructor;
    relationship Set<Student> stds
          inverse takes;}

interface Student extends Person
   (extent Students)
{   attribute string major;
    relationship Set<Course> takes inverse stds;}
```

Why did OO Fail?

- Why are relational databases so popular?
  - Very simple abstraction; don't have to think about programming when storing data.
  - Very well optimized
- Relational db are very well entrenched – not enough advantages, and no good exit strategy (we'll see more about this)

Merging Relational and OODBs

- Object-oriented models support interesting data types – not just flat files.
  - Maps, multimedia, etc.
- The relational model supports very-high-level queries.
- Object-relational databases are an attempt to get the best of both.
- All major commercial DBs today have OR versions – full spec in SQL99, but your mileage may vary.

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

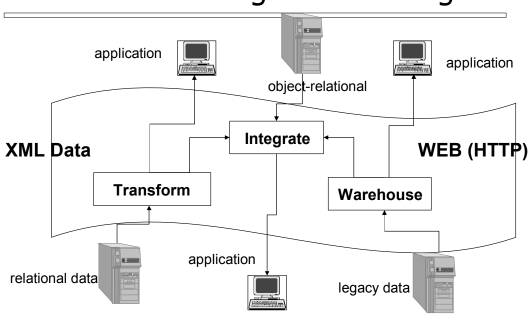
XML

- eXtensible Markup Language
- XML 1.0 – a recommendation from W3C, 1998
- Roots: SGML (from document community - works great for them; from db perspective, very nasty).
- After the roots: a format for sharing *data*

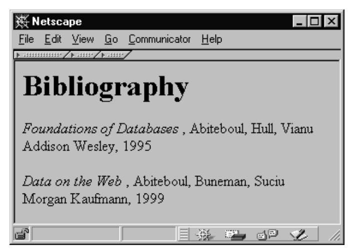
Why XML is of Interest to Us

- XML is just syntax for data
  - Note: we have no syntax for relational data
  - But XML is not relational: *semistructured*
- This is exciting because:
  - Can translate *any* data to XML
  - Can ship XML over the Web (HTTP)
  - Can input XML into any application
  - Thus: data sharing and exchange on the Web

XML Data Sharing and Exchange



From HTML to XML



HTML describes the presentation

HTML

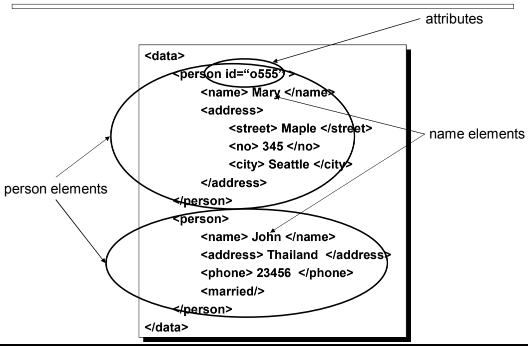
```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
        Abiteoul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999
```

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

XML Document



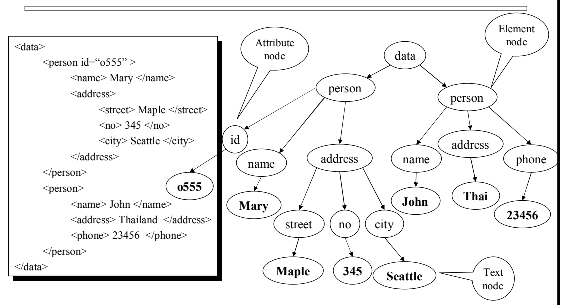
XML Terminology

- Elements
  - enclosed within tags:
    - <person> … </person>
  - nested within other elements:
    - <person> <address> … </address> </person>
  - can be empty
    - <married></married> abbreviated as <married/>
  - can have Attributes
    - <person id="0005"> … </person>
- XML document has as single ROOT element

Buzzwords

- What is XML?
  - W3C data exchange format
  - Hierarchical data model
  - Self-describing
  - Semi-structured

XML as a Tree !!



Minor Detail: Order matters !!!

XML is self-describing

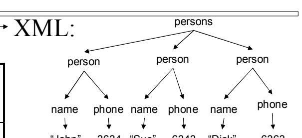
- Schema elements become part of the data
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
  - Relational schema: persons(name,phone) defined separately for the data and is fixed
- Consequence: XML is much more flexible

Relational Data as XML

person

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<persons>
    <person> <name>John</name>
             <phone> 3634</phone>
    </person>
    <person> <name>Sue</name>
             <phone> 6343</phone>
    </person>
    <person> <name>Dick</name>
             <phone> 6363</phone>
    </person>
</persons>
```

XML is semi-structured

- Missing elements:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

← no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

XML is semi-structured

- Repeated elements

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

← two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

XML is semi-structured

- Elements with different types in different objects

```
<person> <name>  <first> John </first>
                 <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

← structured name !

- Heterogeneous collections:
  - <persons> can contain both <person>s and <customer>s

So how would you store XML in a relational db?

Summarizing XML

- XML has first class elements and second class attributes
- XML is semi-structured
- XML is nested
- XML is a tree
- XML is a huge buzzword

Will XML replace relational databases?

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

Other data formats

- Makefiles
- Forms
- Application code

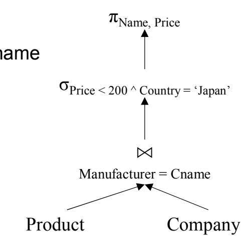
What format is your data in?

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
  - Query Optimization & Execution
  - Transaction Processing

How SQL Gets Executed: Query Execution Plans

Select Name, Price
From Product, Company
Where Manufacturer = Cname
AND Price <= 200
AND Country = 'Japan'



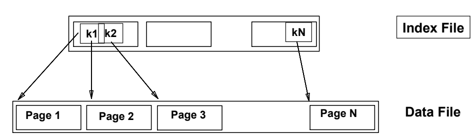
Query optimization also specifies the algorithms for each operator; then queries can be executed

Overview of Query Optimization

- *Plan: Tree of ordered Relational Algebra operators and choice of algorithm for each operator*
- Two main issues:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan.
  Practically: Avoid worst plans.
- Some tactics
  - Do selections early
  - Use materialized views
  - Use Indexes

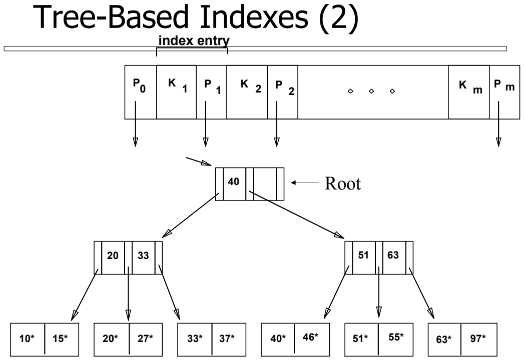
Tree-Based Indexes

- ``*Find all students with gpa > 3.0*''
  - If data is in sorted file, do binary search to find first such student, then scan to find others.
  - Cost of binary search can be quite high.
- Simple idea: Create an `index' file.



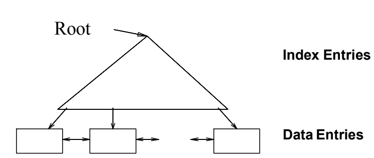
☒ *Can do binary search on (smaller) index file!*

Tree-Based Indexes (2)



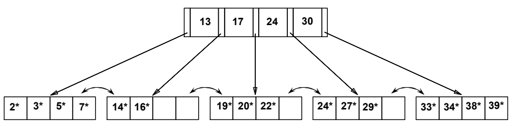
B+ Tree: The Most Widely Used Index

- Insert/delete at $\log_F N$ cost; keep tree *height-balanced*. (F = fanout, N = # leaf pages)
- Minimum 50% occupancy (except for root). Each node contains **d** <= *m* <= 2**d** entries. The parameter **d** is called the *order* of the tree.



Example B+ Tree

- Search begins at root, and key comparisons direct it to a leaf.
- Search for 5*, 15*, all data entries >= 24* ...



Query Execution

- Now that we have the plan, what do we do with it?
  - How do joins work?
  - How do deal with paging in data, etc.
- New research covers new paradigms where interleaved with optimization

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)
  - Query Optimization & Execution
  - Transaction Processing

Transactions

Address two issues:

- Access by multiple users
  - Remember the “client-server” architecture: one server with many clients
- Protection against crashes

Transactions

- Transaction = group of statements that must be executed atomically
- Transaction properties: ACID
  - ***Atomicity***: either all or none of the operations are completed
  - ***Consistency***: preserves database integrity
  - ***Isolation***: concurrent transactions must not interfere with each other
  - ***Durability***: changes from successful transactions must persist through failures

Transaction States

- A transaction can be in one of the following states:
  - ***active***:
    - makes progress or waits for resources; the initial state
  - ***failed:***
    - normal execution cannot continue; it may occur because of
      - system crash
      - cancellation by the user
  - ***aborted***:
    - DBMS cancels it due to problem in execution (e.g., Consistency)
  - ***committed***:
    - after successful completing a “commit” command
    - to undo its effects we need to run a compensating transaction
- Two options for aborted transactions:
  - restart it as a new transaction later (e.g. system failures)
  - kill it (e.g. internal logical errors)
- Failed transactions are eventually aborted

Transaction Example

- Consider two transactions:

```
T1:   READ(A)
      A=A+100
      WRITE(A)
      READ(B)
      B=B-100
      WRITE(B)

T2:   READ(A)
      A=1.1*A
      WRITE(A)
      READ(B)
      B=1.1*B
      WRITE(B)
```

- Intuitively, the first transaction transfers $100 to A's account from B's account. The second credits both accounts with a 10% interest payment.
- No guarantee that T1 executes before T2 or vice-versa. However, the end effect must be equivalent to these two transactions running serially in some order:
  T1, T2 or T2, T1

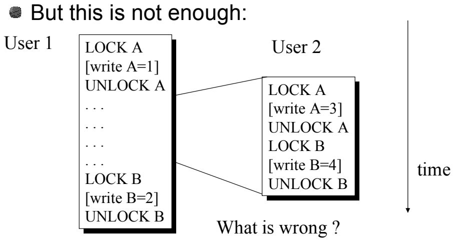
Transactions: Serializability

Serializability = the technical term for isolation

- An execution is ***serial*** if it is completely before or completely after any other function's execution
- An execution is ***serializable*** if it equivalent to one that is serial
- DBMS can offer serializability guarantees

Serializability Example

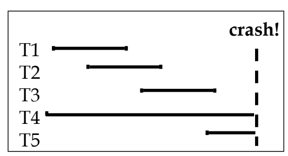
- Enforced with locks, like in Operating Systems !
- But this is not enough:



Okay, but what if it crashes?

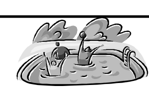
Enforcing Atomicity & Durability

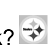
- Atomicity:
  - Transactions may abort ; Need to rollback changes
- Durability:
  - What if DBMS stops running? Need to "remember" committed changes.



- Desired Behaviour after system restarts:
  - T1, T2, & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen)

Handling the Buffer Pool



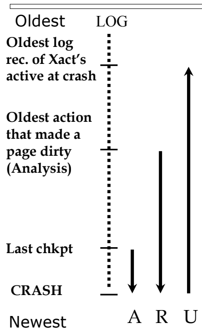
- Transactions modify pages in memory buffers
- Writing to disk is more permanent
- When should updated pages be written to disk? 
- Force every write to disk?
  - Poor response time.
  - But provides durability.
- Steal buffer-pool frames from uncommitted Xacts? (resulting in write to disk)
  - If not, poor throughput.
  - If so, how can we ensure atomicity?

	No Steal	Steal
Force	**Trivial**	
No Force		**Desired**

What to do?

- Basic idea: use steal and no-force
- Keep a log that talks about what's happened
- Make checkpoints where write down everything that's actually happened
- After a crash: assure Atomicity and Durability by keeping all committed transactions and getting rid of actions of uncommitted transactions

Crash Recovery: Big Picture



- Start from a checkpoint (Three phases. Need to:
  - Figure out which Xacts committed since checkpoint, which failed (Analysis)
  - REDO *all* actions
    - (repeat history)
  - UNDO effects of failed Xacts

Outline

- Entity Relationship (ER) diagrams
- Relational databases
- Object Oriented Databases (OODBs)
- XML
- Other data types
- Database internals (Briefly)

Now what?

- Time to read papers
- Prepare paper responses – it'll help you focus on the paper, and allow for the discussion leader to prepare better discussion
- You all have different backgrounds, interests, and insights. Bring them into class!