CS145 Introduction

About CS145

Relational Model, Schemas, SQL

Semistructured Model, XML

Content of CS145

- Design of databases.
  - E/R model, relational model, semistructured model, XML, UML, ODL.
- Database programming.
  - SQL, XPath, XQuery, Relational algebra, Datalog.
- Not DBMS implementation (that’s CS245, 346, 347, sometimes CS345).

Textbook “Situation”

- The closest text for the course is *First Course in Database Systems/3rd Edition*.
  - But it won’t be available until Friday.
  - First 2 chapters available on-line.
- You may prefer *Database Systems: Complete Book* (also used in CS245) or have *FCDB/2nd E*.
  - If so, we’ll give you a free copy of the major additions in *FCDB/3rd E*.

Do You Know SQL?

- Explain the difference between:

```
SELECT b
FROM R
WHERE a<10 OR a>=10;
```

and

```
SELECT b
FROM R;
```

a	b
5	20
10	30
20	40
…	…

R

And How About These?

```
SELECT a
FROM R, S
WHERE R.b = S.b;

SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Course Requirements

1. Project: a little eBay supported by a database.
   - Individual.
   - Uses Stanford Oracle system.
2. Homeworks: Gradiance (automated) and "challenge problems" (written).
3. Midterm and final.

Gradiance Homework System

- Automatic, fast-feedback system for taking you through standard homework problems and verifying your knowledge.
- Unusual: goal is to get 100% and learn.
  - Homework in CS145 is not a "mini-test."
  - You try as many times as you like and get help with each wrong answer.

Gradiance (GOAL) Access

- To get your account, you need:

  1. "Value-Pak" with any of the class texts, or purchase on-line.
  2. Class token: For FCDB/3e use 1B8B815E; for other books use A5DDE704.

- Details in the intro.html file.
- Advice on using Gradiance: www.gradiance.com/info.html

Interesting Stuff About Databases

- It used to be about boring stuff: employee records, bank records, etc.
- Today, the field covers all the largest sources of data, with many new ideas.
  - Web search.
  - Data mining.
  - Scientific and medical databases.
  - Integrating information.

More Interesting Stuff

- Database programming centers around limited programming languages.
  - Only area where non-Turing-complete languages make sense.
  - Leads to very succinct programming, but also to unique query-optimization problems (CS346).

Still More …

- You may not notice it, but databases are behind almost everything you do on the Web.
  - Google searches.
  - Queries at Amazon, eBay, etc.

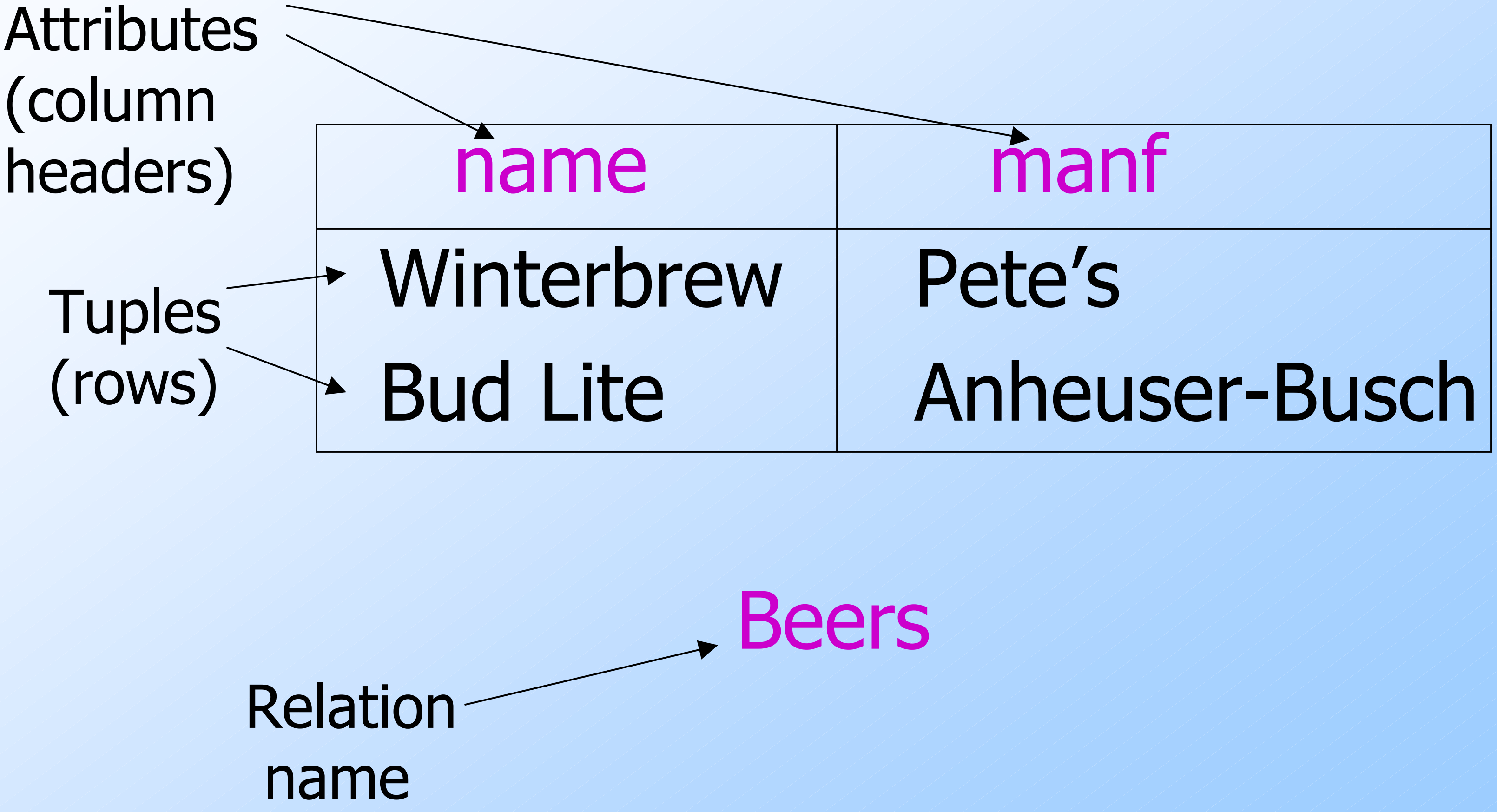
And More…

- Databases often have unique concurrency-control problems (CS245, CS347).
  - Many activities (transactions) at the database at all times.
  - Must not confuse actions, e.g., two withdrawals from the same account must each debit the account.

What is a Data Model?

1. Mathematical representation of data.
   - Examples: relational model = tables; semistructured model = trees/graphs.
2. Operations on data.
3. Constraints.

A Relation is a Table



Attributes (column headers)

name	manf
Winterbrew	Pete’s
Bud Lite	Anheuser-Busch

Tuples (rows)

Beers

Relation name

Schemas

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes.
  - Example: Beers(name, manf) or Beers(name: string, manf: string)
- *Database schema* = set of all relation schemas in the database.
- *Database* = collection of relations.

Why Relations?

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, the most important database language today.

Our Running Example

Beers(name, manf)

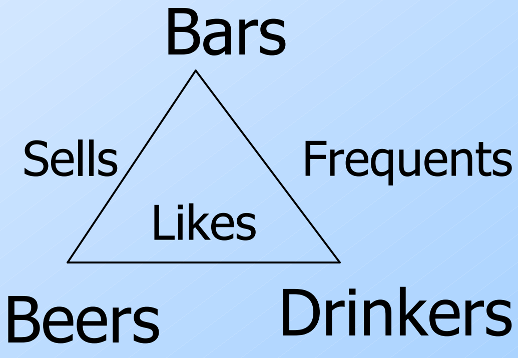
Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)



- Underline = *key* (tuples cannot have the same value in all key attributes).
  - Excellent example of a constraint.

Database Schemas in SQL

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Elements of Table Declarations

- Most basic element: an attribute and its type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.

Example: Create Table

```
CREATE TABLE Sells (
    bar    CHAR(20),
    beer   VARCHAR(20),
    price  REAL
);
```

SQL Values

- Integers and reals are represented as you would expect.
- Strings are too, except they require single quotes.
  - Two single quotes = real quote, e.g., `’Joe’’s Bar’`.
- Any value can be NULL.

Dates and Times

- DATE and TIME are types in SQL.
- The form of a date value is:

  DATE ’yyyy-mm-dd’

  - Example: `DATE ’2007-09-30’` for Sept. 30, 2007.

Times as Values

- The form of a time value is:

  TIME ’hh:mm:ss’

with an optional decimal point and fractions of a second following.

  - Example: `TIME ’15:30:02.5’` = two and a half seconds after 3:30PM.

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.
- Either says that no two tuples of the relation may agree in all the attribute(s) on the list.
- There are a few distinctions to be mentioned later.

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (
    name    CHAR(20) UNIQUE,
    manf    CHAR(20)
);
```

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer      VARCHAR(20),
    price     REAL,
    PRIMARY KEY (bar, beer)
);
```

PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

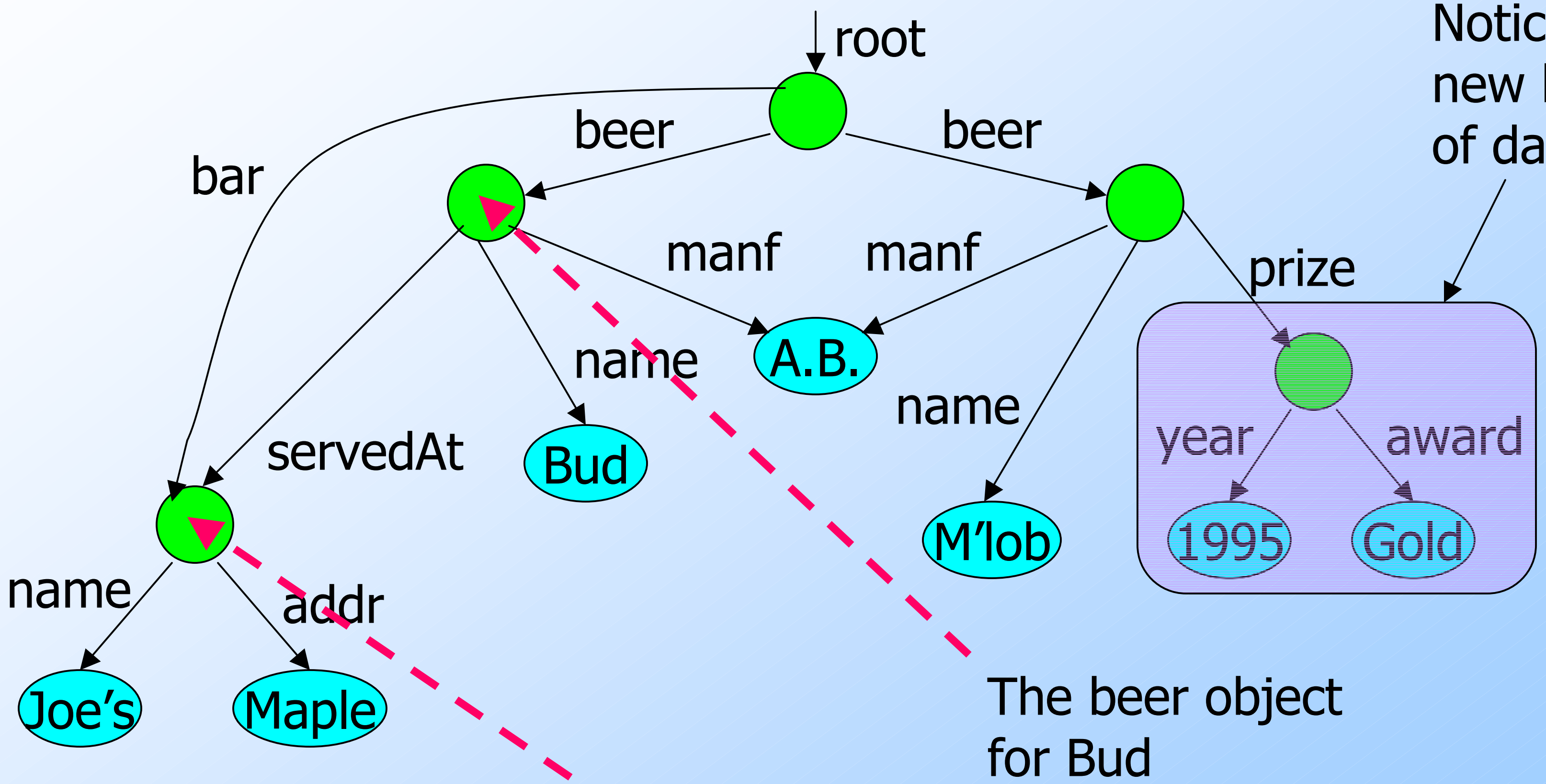
Semistructured Data

- A data model based on trees.
- Motivation: flexible representation of data.
- Motivation: sharing of *documents* among systems and databases.

Graphs of Semistructured Data

- Nodes = objects.
- Arc labels (properties of objects).
- Atomic values at leaf nodes (nodes with no arcs out).
- Flexibility: no restriction on:
  - Labels out of a node.
  - Number of successors with a given label.

Example: Data Graph



Notice a new kind of data.

The beer object for Bud

The bar object for Joe's Bar

XML

- XML = *Extensible Markup Language*.
- While HTML uses tags for formatting (e.g., “italic”), XML uses tags for semantics (e.g., “this is an address”).

XML Documents

- Start the document with a *declaration*, surrounded by <?xml … ?> .
- Typical:

```
<?xml version = “1.0” encoding
  = “utf-8” ?>
```

- Balance of document is a *root tag* surrounding nested tags.

Tags

- Tags, as in HTML, are normally open-close pairs, as <FOO> … </FOO>.
  - Optional single tag <FOO/>.
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

Example: an XML Document

```
<?xml version = “1.0” encoding = “utf-8” ?>
<BARS>
  <BAR><NAME>Joe’s Bar</NAME>
     <BEER><NAME>Bud</NAME>
           <PRICE>2.50</PRICE></BEER>
     <BEER><NAME>Miller</NAME>
           <PRICE>3.00</PRICE></BEER>
  </BAR>
  <BAR> …
</BARS>
```

A NAME subobject

A BEER subobject

Attributes

- Like HTML, the opening tag in XML can have attribute = value pairs.
- Attributes also allow linking among elements (discussed later).

Bars, Using Attributes

```
<?xml version = "1.0" encoding = "utf-8" ?>
<BARS>
  <BAR name = "Joe's Bar">
      <BEER name = "Bud" price = 2.50 />
      <BEER name = "Miller" price = 3.00 />
  </BAR>
  <BAR> ...
</BARS>
```

name and price are attributes

Notice Beer elements have only opening tags with attributes.

DTD's (Document Type Definitions)

- A grammatical notation for describing allowed use of tags.
- Definition form:

```
<!DOCTYPE <root tag> [
  <!ELEMENT <name>(<components>)>
  . . . more elements . . .
]>
```

Example: DTD

```
<!DOCTYPE BARS [
  <!ELEMENT BARS (BAR*)>
  <!ELEMENT BAR (NAME, BEER+)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT BEER (NAME, PRICE)>
  <!ELEMENT PRICE (#PCDATA)>
]>
```

A BARS object has zero or more BAR's nested within.

A BAR has one NAME and one or more BEER subobjects.

A BEER has a NAME and a PRICE.

NAME and PRICE are HTML text ("parsed character data").

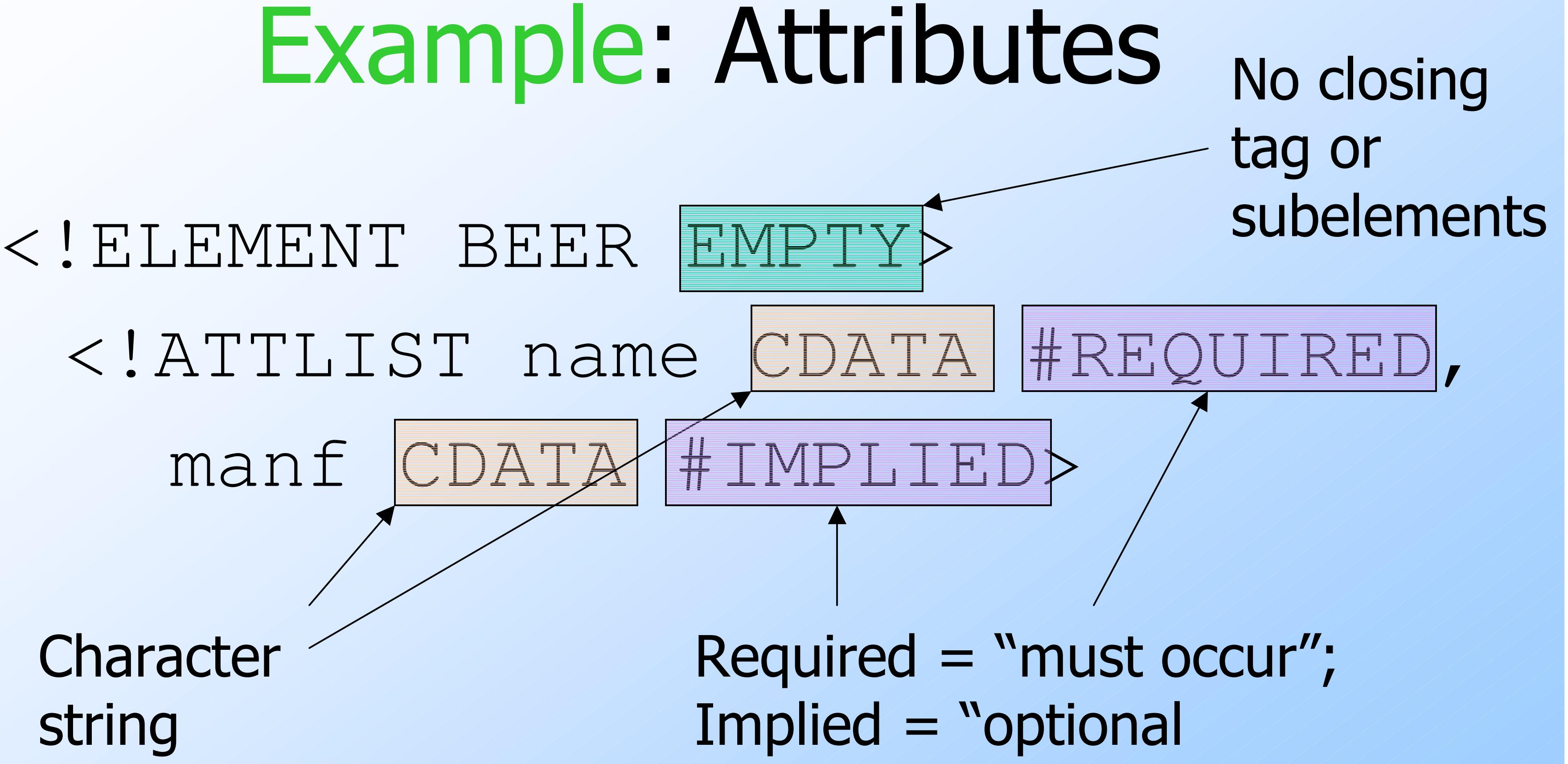
Attributes

- Opening tags in XML can have *attributes*.
- In a DTD,

`<!ATTLIST` *E* `...>`

declares an attribute for element *E*, along with its datatype.

Example: Attributes



```
<!ELEMENT BEER EMPTY>
  <!ATTLIST name CDATA #REQUIRED,
     manf CDATA #IMPLIED>
```

No closing tag or subelements

Character string

Required = “must occur”; Implied = “optional

Example use:
<BEER name=“Bud” />