CMPT 354
Database Systems I

Fall 2015

Introduction to Database Systems

What is a Database?

- A database is simply a collection of information that persists over a long period of time.
- This information is typically highly structured (e.g. in the case of the relational model, in tables)
- Operations: Create, add, delete, modify, … entities

Interesting Stuff About Databases

- Databases are traditionally about stuff like employee records, bank records, etc.

  → They still are.

- But today, the field also 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 centres around *limited programming languages.*
  - One of the only areas where *non-Turing-complete* languages make sense.
  - Leads to very succinct programming, but also to unique query-optimization problems (CMPT 454)….
  - So they exploit a tradeoff between
    - what you can compute and
    - how easy it is to compute something.
- When you think about it, 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 (CMPT 454).
  - 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
  - Can't have a transaction fail half-way through

Database Management System (DBMS)

- Database Applications:
  - Banking: financial transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
- A DBMS contains information about a particular enterprise
  - Collection of interrelated data (description + data)
  - Set of programs to access the data
  - An environment that is both *convenient* and *efficient* to use

Question: Why have database systems (and not just directly use a file system)?

Purpose of Database Systems

- In the early days, database applications were built directly on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Different programmers will create files & application programs over a long period of time
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become "buried" in program code rather than being stated explicitly
    - Hard to add new constraints or change existing ones

Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
  - Atomicity of updates
    - Failures may leave database in an inconsistent state with partial updates carried out
    - Example: Transfer of funds from one account to another should either complete or not happen at all
  - Concurrent access by multiple users
    - Concurrent accessed is needed for performance
    - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance and updating it at the same time
  - Security problems
    - Hard to provide user access to some, but not all, data
- Database systems offer solutions to all the above problems

Databases: Levels of Abstraction

- Physical level
- Logical level
- View level

Levels of Abstraction

- **Physical level:** Describes *how* a record (e.g., customer) is stored (again, CMPT 454)
  - Described in terms of low-level data structures

Levels of Abstraction

- **Logical level:** describes *what* data is stored in a database, and the relationships among the data.
  - Don't need to know physical representation
  - Analogy: record declaration:

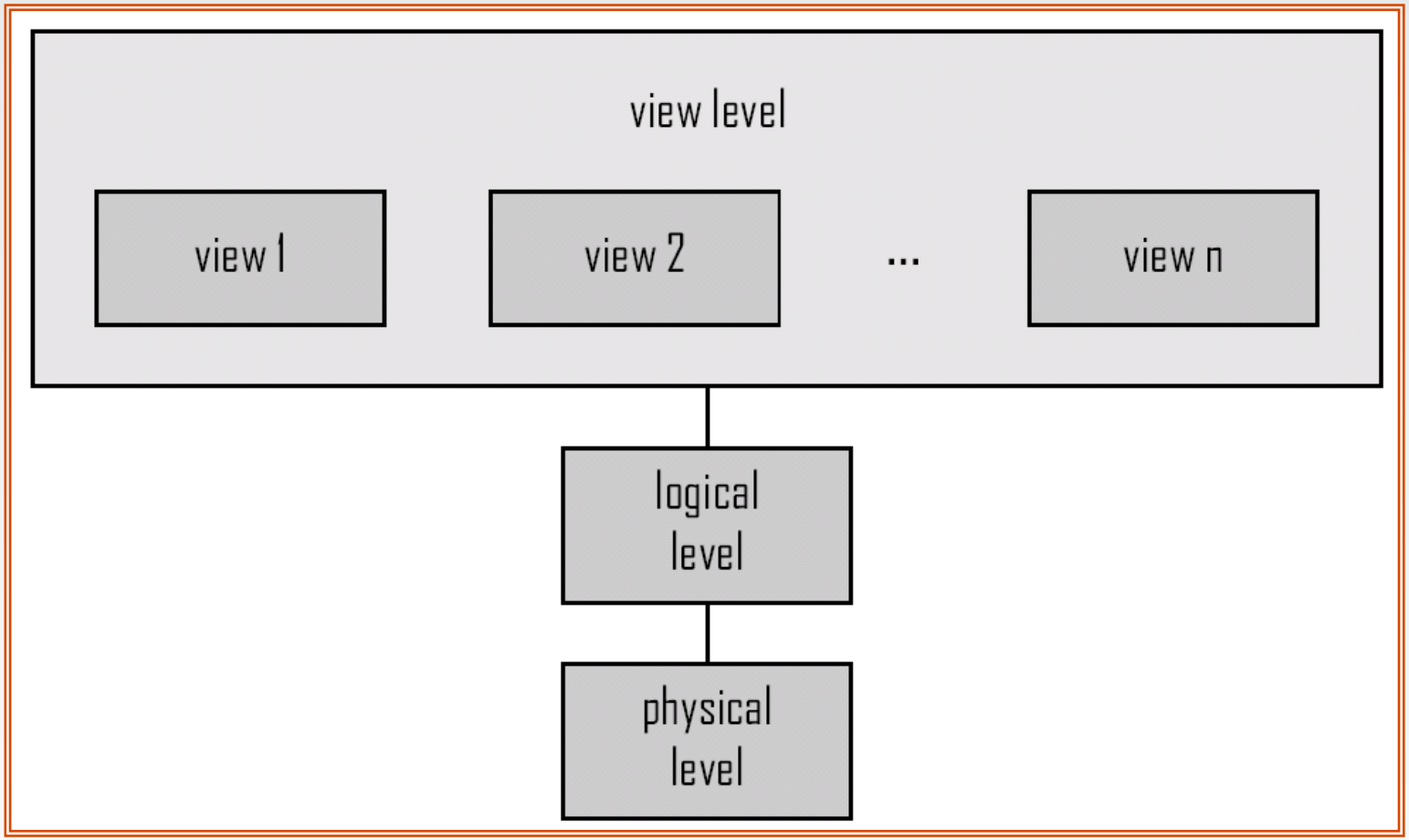
```
type customer = record
        customer_id : string;
        customer_name : string;
        customer_street : string;
        customer_city : integer;
    end;
```

Levels of Abstraction

- **View level:** describes only part of the full database.
  - Example: A teller may see bank account balances but not personal information
  - The view level simplifies interaction for users as well as provides (more) security
  - May have many views for the same database

Three Levels of Data Abstraction

An architecture for a database system



Data Models

- A data model is a notation for describing data or information
- A data model consists of a set of conceptual tools for *describing* data, data relationships, data semantics, and consistency requirements.
- Three parts:

1. Structure of the data.

   Examples:

   - relational model = tables;
   - entitly/relationship model = entities + relationships between them
   - semistructured model = trees/graphs.

2. Operations on data.
3. Constraints on the data.

Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - Eg: the database consists of information about customers and accounts, and the relationship between them
  - Analogous to type information of a variable in a program
  - Physical schema: database design at the physical level
  - Logical schema: database design at the logical level
- **Instance** – the actual content of the database at some point in time
  - Analogous to the value of a variable

Data Definition Language (DDL)

- The DDL is the language for defining the database schema
  - Eg: **create** table account (

    account-number **char**(10),

    balance **integer**)
- Need to be able to specify information such as
  - Database schema
  - Storage structure and access methods used
  - Integrity constraints
    - Domain constraints
    - Referential integrity
    - Assertions
  - Authorisation

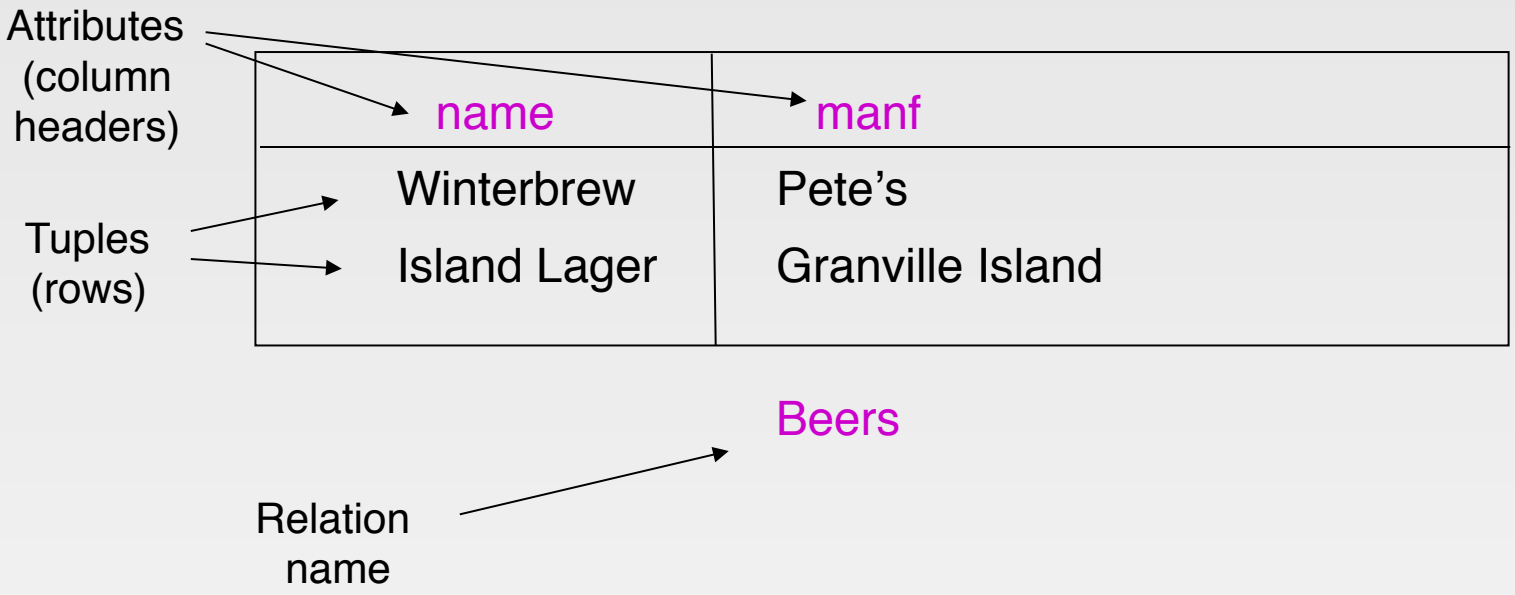
Data Manipulation Language (DML)

- The DML is the language for accessing and manipulating the data, organized by the appropriate data model
  - Also known as the query language
- Two classes of languages
  - **Procedural** – user specifies what is required and how to get it
  - **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get the data
- SQL is the most widely used query language
  - Nonprocedural

The Relational Model

- The central data model that we will look at is the relational model
- The relational model uses *tables* to represent both data and relationships among the data

Example Relation



Schemas

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes.
  - Example: Beers(name, manf) or Beers(name: string, manf: string)
  - *Describes* a relation
- *Relation instance* = actual data in a relation
- *Database* = collection of relations (instances).
  - Sometimes will refer to the *database instance* (in contrast to the database schema)
- *Database schema* = set of all relation schemas in the database.

Why Relations?

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

A Running Example: Beer Domain

Beers(<u>name</u>, manf)

Bars(<u>name</u>, addr, license)

Customers(<u>name</u>, addr, phone)

Likes(customer, <u>beer</u>)

Sells(<u>bar</u>, <u>beer</u>, price)

Frequents(<u>customer</u>, <u>bar</u>)

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

Another Example: Banking Domain

Branch(branch_name, branch_city, assets)

Customer(customer_name, customer_street, customer_city)

Loan(loan_number, branch_name, amount)

Borrower(customer_name, loan_number)

Account(account_number, branch_name, balance)

Depositor(customer_name, account_number)

Relational Approach: 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 are enclosed in single quotes.
- Any value can be NULL.

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.
- So keys provide a means of uniquely identifying tuples.
- 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) PRIMARY KEY,
        manf    CHAR(20)
);
```

Declaring Multiattribute Keys

- A key declaration can also be several elements 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)
);
```

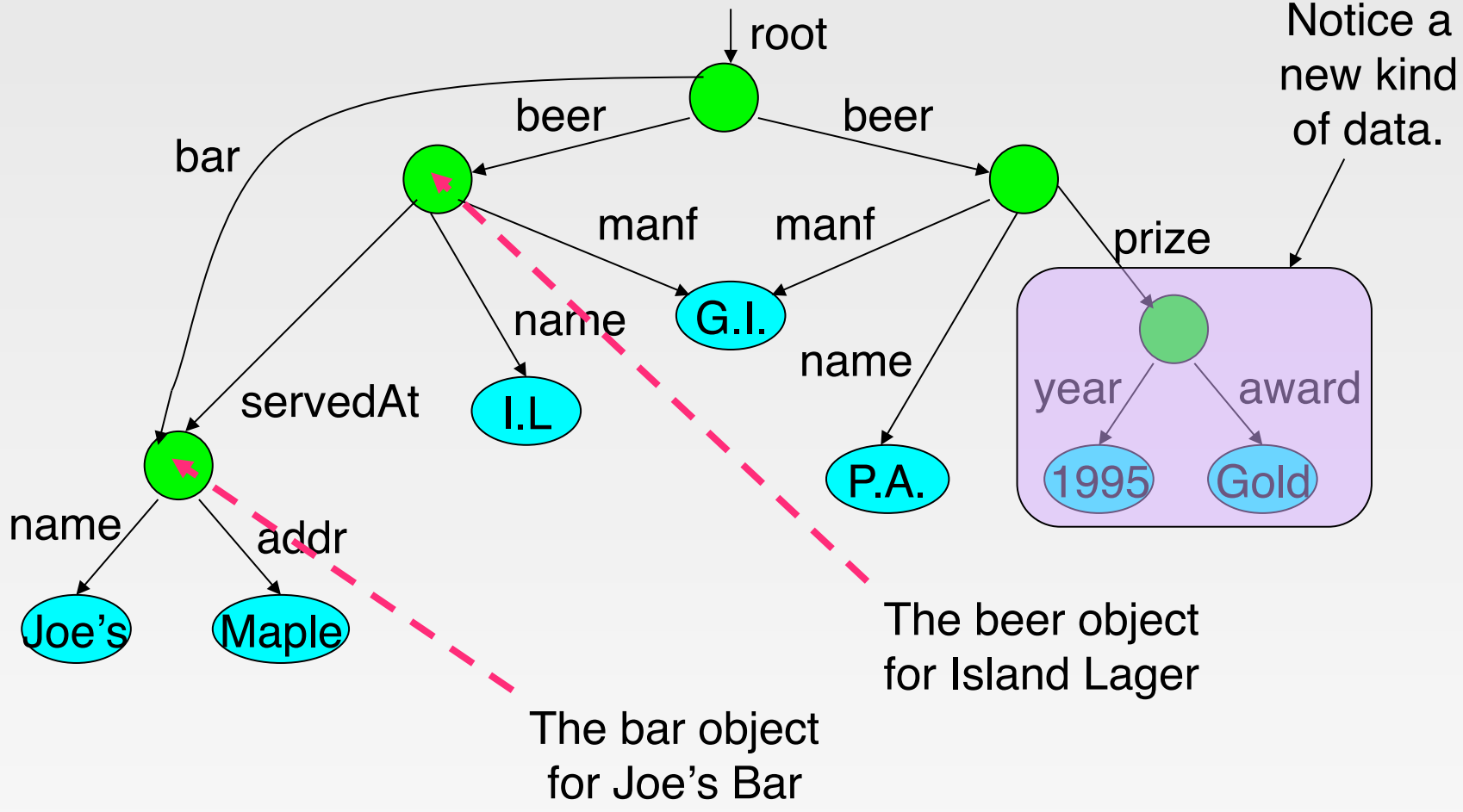
Semistructured Data: XML

- Another data model, based on *trees*.
  - Actually, trees + “cross links”, so *graphs*
- Motivation:
  - Flexible representation of data.
  - Sharing of *documents* among systems and databases.

Graphs of Semistructured Data

- Nodes = objects.
- Labels on arcs (like attribute names).
- 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



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”).
- Key idea:
  - create tag sets for a domain (e.g., genomics), and
  - translate all data into properly tagged XML documents.

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 matched pairs
  - E.g. <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>P.A.</NAME>
      <PRICE>2.50</PRICE></BEER>
    <BEER><NAME>G.I.</NAME>
      <PRICE>5.00</PRICE></BEER>
  </BAR>
  <BAR> … </BAR> …
</BARS>
```

A NAME subobject

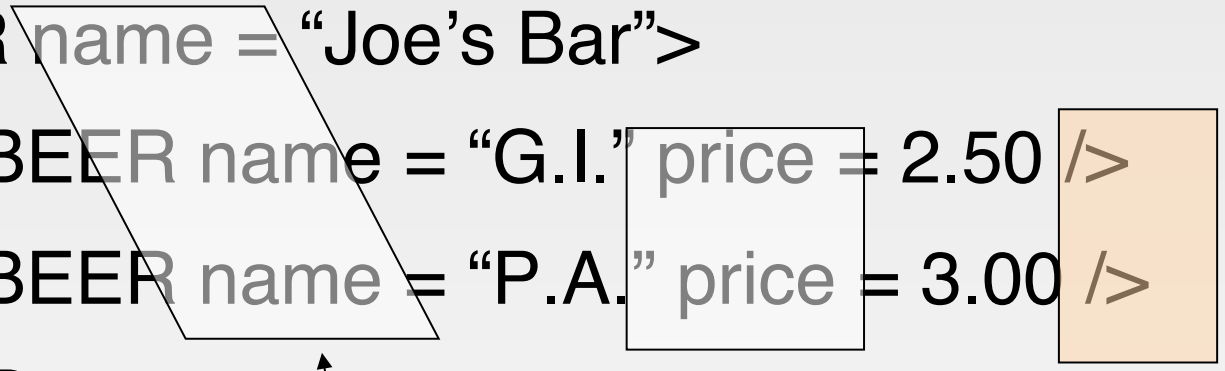
A BEER subobject

Attributes

- Like HTML, the opening tag in XML can have atttribute = 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 = “G.I.” price = 2.50 />
      <BEER name = “P.A.” 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 in XML.
- Definition form:

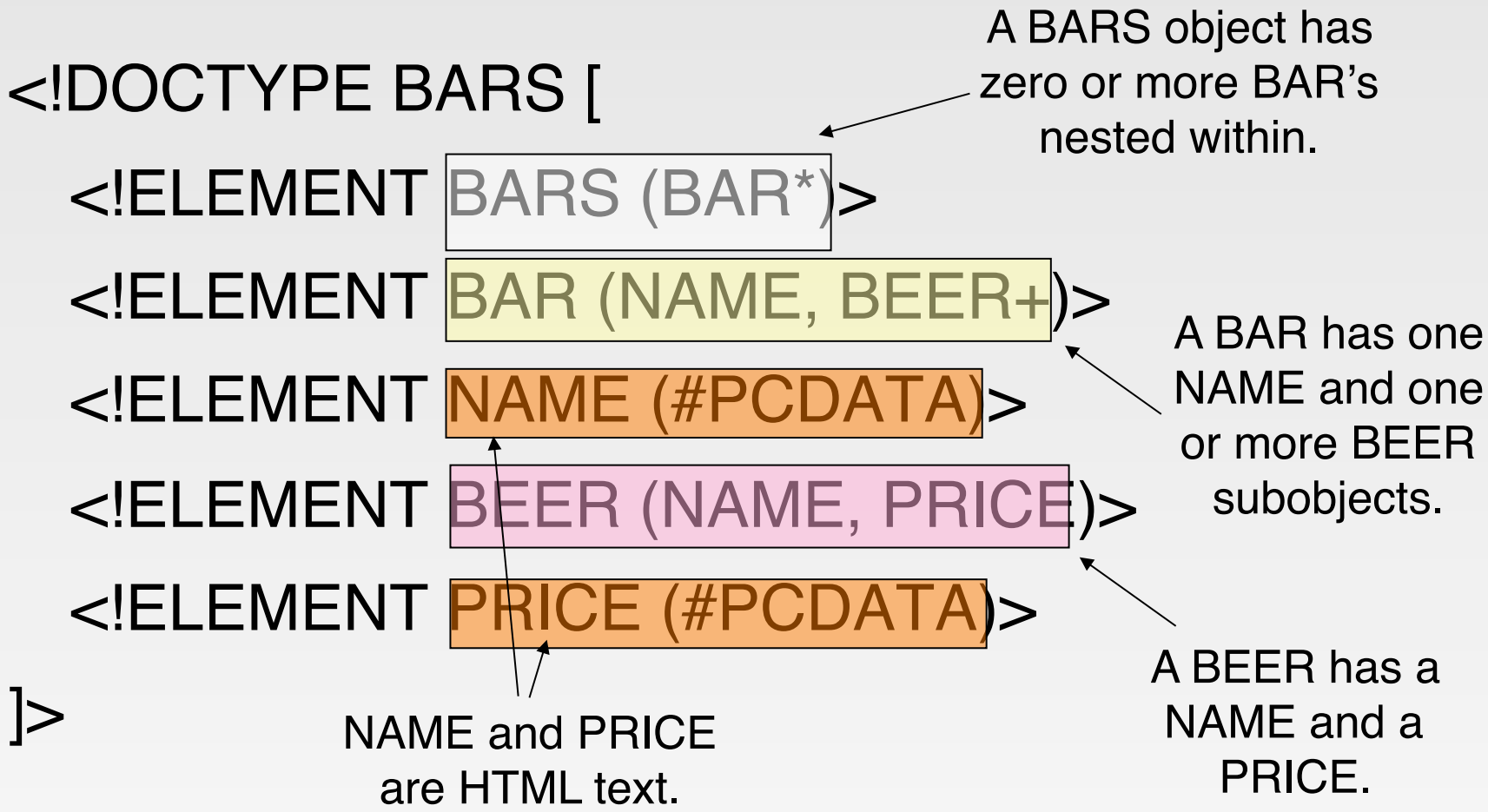
<!DOCTYPE <root tag> [

    <!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.

End of Introduction