# Object-Based Databases

Database System Concepts, 6th Ed.

# Object-Relational Data Models

- Extend the relational data model by including object orientation and constructs to deal with added data types.
- Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
- Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
- Upward compatibility with existing relational languages.

# Complex Data Types

- Motivation:
  - Permit non-atomic domains (atomic ≡ indivisible)
  - Example of non-atomic domain:  set of integers,or set of tuples
  - Allows more intuitive modeling for applications with complex data

# Example of a Nested Relation

- Example: library information system
- Each book has
  - title,
  - a list (array) of authors,
  - Publisher, with subfields *name* and *branch*, and
  - a set of keywords
- Non-1NF relation *books*

| *title* | *author_array* | *publisher* | *keyword_set* |
|---|---|---|---|
| | | (*name*, *branch*) | |
| Compilers | [Smith, Jones] | (McGraw-Hill, NewYork) | {parsing, analysis} |
| Networks | [Jones, Frick] | (Oxford, London) | {Internet, Web} |

# Complex Types and SQL

- Extensions introduced in SQL:1999 to support complex types:
  - Collection and large object types
    - Nested relations are an example of collection types
  - Structured types
    - Nested record structures like composite attributes
  - Inheritance
  - Object orientation
    - Including object identifiers and references
- Not fully implemented in any database system currently
  - But some features are present in each of the major commercial database systems
    - Read the manual of your database system to see what it supports

# Structured Types and Inheritance in SQL

- **Structured types** (a.k.a. **user-defined types**) can be declared and used in SQL

```
create type Name as
    (firstname      varchar(20),
     lastname       varchar(20))
     final

create type Address as
    (street         varchar(20),
     city           varchar(20),
     zipcode        varchar(20))
     not final
```

  - Note: **final** and **not final** indicate whether subtypes can be created

- Structured types can be used to create tables with composite attributes

```
create table person (
    name        Name,
    address     Address,
    dateOfBirth date)
```

- Dot notation used to reference components: *name.firstname*

# Methods

- Can add a method declaration with a structured type.

  **method** *ageOnDate* (*onDate* **date**)

  **returns interval year**

- Method body is given separately.

  **create instance method** *ageOnDate* (*onDate* **date**)

  **returns interval year**

  **for** *CustomerType*

  **begin**

  **return** *onDate* - **self**.*dateOfBirth*;

  **end**

- We can now find the age of each customer:

  **select** *name.lastname, ageOnDate* (**current_date**)

  **from** *customer*

# Type Inheritance

- Suppose that we have the following type definition for people:

```
create type Person
    (name varchar(20),
    address varchar(20))
```

- Using inheritance to define the student and teacher types

```
create type Student
  under Person
  (degree       varchar(20),
   department   varchar(20))
create type Teacher
  under Person
  (salary       integer,
   department   varchar(20))
```

- Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

# Array and Multiset Types in SQL

- Example of array and multiset declaration:

```
create type Publisher as
    (name            varchar(20),
     branch          varchar(20));
create type Book as
    (title           varchar(20),
     author_array    varchar(20) array [10],
     pub_date        date,
     publisher       Publisher,
     keyword-set     varchar(20) multiset);
create table books of Book;
```

# Path Expressions

- Find the names and addresses of the heads of all departments:

  **select** *head ->name*, *head ->address*
  **from** *departments*

- An expression such as “head->name” is called a **path expression**
- Path expressions help avoid explicit joins
  - If department head were not a reference, a join of *departments* with *people* would be required to get at the address
  - Makes expressing the query much easier for the user

# Persistent Programming Languages

- Languages extended with constructs to handle persistent data
- Programmer can manipulate persistent data directly
  - no need to fetch it into memory and store it back to disk

- Persistent objects:
  - **Persistence by class** - explicit declaration of persistence
  - **Persistence by creation** - special syntax to create persistent objects
  - **Persistence by marking** - make objects persistent after creation
  - **Persistence by reachability** - object is persistent if it is declared explicitly to be so or is reachable from a persistent object

# Persistent Systems

- Persistent versions of C++ and Java have been implemented
  - C++
    - ODMG C++
    - ObjectStore
  - Java
    - Java Database Objects (JDO)

# Persistent Java Systems

- Standard for adding persistence to Java : **Java Database Objects (JDO)**
  - Persistence by reachability
  - Byte code enhancement
    - Classes separately declared as persistent
    - Byte code modifier program modifies class byte code to support persistence
  - Database mapping
    - Allows objects to be stored in a relational database

 

# Object-Relational Mapping

- **Object-Relational Mapping (ORM)** systems built on top of traditional relational databases
- Implementor provides a mapping from objects to relations
  - Objects are purely transient, no permanent object identity
- Objects can be retried from database
  - System uses mapping to fetch relevant data from relations and construct objects
  - Updated objects are stored back in database by generating corresponding update/insert/delete statements
- The **Hibernate** ORM system is widely used
  - Provides API to start/end transactions, fetch objects, etc
  - Provides query language operating directly on object model
    - queries translated to SQL
- Limitations: overheads, especially for bulk updates