Other Data Models

• New edition (DBS the complete book): Chapter 4
• Old edition (First course in DBS):
  – Section 2.1
  – Section 2.3.4
  – Section 2.4.1, 2.4.2
  – Section 2.5.2
  – Section 3.2 (ALL)

Emerging Data Models

• Not as popular as relational model or E/R model but become more and more important
• Include:
  – Object-oriented data model
  – Object-relational data model
  – Semistructured data model
    • DOC
    • XML

Text

• New edition (DBS the complete book): Chapter 4
• Old edition (First course in DBS):
  – Section 2.1
  – Section 2.3.4
  – Section 2.4.1, 2.4.2
  – Section 2.5.2
  – Section 3.2 (ALL)

Example of an Object

OID

( #32,
  SSN : 111-22-333,
  Name: Joe Public,
  PhoneN: { “505-011-2222”, “505-011-2334” },
  Child: { #445, #73 } )

VALUE
Objects

• Every object has a unique id (oid)
  – not depend on the value of the object
  – assigned to object by the system when it is created
  – not change during its lifetime
  – not visible to programmer

Definition

• An object is a pair (oid, val) where
  – oid – object id
  – val – can be
    • Primitive (integer, string, float, boolean)
    • Reference value (an oid of an object)
    • Record value \[A_1:v_1, \ldots, A_n:v_n\], where the A_1,\ldots, A_n are distinct attribute names and v_1,\ldots, v_n are values
    • Set value like \{v_1,\ldots, v_n\}

Example of an Object and Values

\[
\left( \#32, \begin{array}{|c|}
\hline
\text{OID} \\
\hline
\#32 \\
\hline
\end{array}, \begin{array}{|c|}
\hline
\text{VALUE} \\
\hline
\text{Reference Value} \\
\hline
\end{array}\right)
\]

[ SSN : 111-22-333,
Name: Joe Public,
PhoneN: \{ “505-011-2222”,
  “505-011-2334”\},
Child: \{\#445, \#73\} ]

Classes

• Group of semantically similar objects
  – Exp: Persons, Students, House, …
• Play the roles of relations in relational databases
• Typed (each class has a type) – describes its structure
• Method Signatures (declaration of methods – not the implementation of the methods that will be stored in the DB server)
• ISA hierarchy (Class – Subclass relationship)
Types

- **Basic types**: (integer, string, float, boolean)
- **Reference types**: (user defined names)
- **Record types**: \([A_1:T_1, \ldots, A_n:T_n]\), where the \(A_1, \ldots, A_n\) are distinct attribute names and \(T_1, \ldots, T_n\) are types
- **Set types**: \(\{T\}\) where \(T\) is a type

Subtypes

- **Student**
  - [SSN : String, Name : String, Address : [StNumber : Integer, StName: String]
  - Majors: {String}, Enrolled : {Course}
- **Subtype and Supertype**
  - Reflect the richness of a structure
  - \(T\) subtype of \(T'\) if \(T \neq T'\) and one of the following condition holds:
    - \(T\) and \(T'\) are reference types and \(T\) is a subclass of \(T'\)
    - \(T\) = \([A_1:T_1, \ldots, A_n:T_n, A_{n+1}:T_{n+1}, \ldots, A_m:T_m]\) and \(T' = [A_1:T'_1, \ldots, A_n:T'_n]\) and either \(T_i=T'_i\) or \(T_i\) is a subtype of \(T'_i\) for each \(i=1, \ldots, m\)
    - \(T = \{T_0\}\) and \(T' = \{T'_0\}\) and \(T_0\) is a subtype of \(T'_0\)
Example of Subtype

**Person**: [SSN: String, Name: String, Address: [StNumber: Integer, StName: String]]

**Student**: [SSN: String, Name: String, Address: [StNumber: Integer, StName: String], Majors: [String], Enrolled: [Course]]

*Student* is a subtype of *Person*

---

Domain of a Type T - \( \text{domain}(T) \)

Determines all objects that conform to type \( T \)
- **Basic types**: (integer, string, float, boolean)
- **Reference types**: set of all oids of all objects of the class \( T \)
- **Record types**: \([A_1:T_1, \ldots, A_n:T_n]\) then \( \text{domain}(T) = \{[A_1:w_1, \ldots, A_n:w_n] \mid w_i \in T_i \} \)
- **Set types**: \([T_0]\) then \( \text{domain}(T_0) = \{[w_1, \ldots, w_n] \mid n \geq 0, w_i \in T_0 \} \)

---

Database Schema and Instance

- **Schema** – specification of classes that can be stored in the database
  - For each class:
    - Type
    - Method signature
    - Subclass-of relationship
    - Integrity constraint
- **Instance** – specific objects for the classes

---

Object Definition Language (ODL)

- Uses to describe database schema
  - (schema is stored in the system catalog and uses at runtime to evaluate queries and perform database updates)
- Gap between ODL and database language
  - Data definition vs. data manipulation
Object Definition Language

- Objects with similar properties are grouped into class – like entities are collected into entity set
- Objects have
  - attributes (values associated with the object)
  - relationships (connections with other objects)
  - methods (functions that may be applied to the objects of the class)

Class declarations – Attributes

class Movie{
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enum Film {color, blackAndWhite} filmType;
};

Attributes: title, year, length, filmType
string, integer: basic types
e num Film {…}: enumeration type named Film whose possible values are listed in {…}

Class declarations – relationships

class Movie{
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enum Film {color, blackAndWhite} filmType;
    relationship Set<Star> stars;
};

relationship Set<Star> stars: in each object of the class Movie there is a set of references to Star objects
Class declarations – relationships

```java
class Movie{
    (1) attribute string title;
    (2) attribute integer year;
    (3) attribute integer length;
    (4) attribute enum Film {color, blackAndWhite} filmType;
    (5) relationship Set<Star> stars;
};
```

“relationship Set<Star> stars”: in each object of the class Movie there is a set of references to Star objects.

Should the Star class contains “relationship Set<Movie> starredIn”? Integrity constraint: starredIn of star must contain the name of the movie in which the star plays.

(5) and (6): the relationship stars of Movie is the inverse relation of starredIn of Star.

Multiplicity of Relationships

- C, D – classes
- R: relationship from C to D
  - R is many to many, C contains: `relationship Set<D> nameRelationshipRCD` inverse D:: nameRelationshipRDC`
  - R is many-one, C contains: `relationship D nameRelationshipRCD` inverse D:: nameRelationshipRDC`
  - R is one-many, C contains: `relationship Set<D> nameRelationshipRCD` inverse D:: nameRelationshipRDC`
  - R is one-to-one, C contains: `relationship D nameRelationshipRCD` inverse D:: nameRelationshipRDC`

Methods

```java
class Movie{
    (1) attribute string title;
    (2) attribute integer year;
    (3) attribute integer length;
    (4) attribute enum Film {color, blackAndWhite} filmType;
    (5) relationship Set<Star> stars;
    (6) inverse Star::starredIn;
};
```

(float, void): type of function value
parameters: in, out, inout – input, output, input/output or both parameters.