Chapter 6

Object Oriented Database

What is Object Oriented Database?(OODB)

- A **database** system that incorporates all the important object-oriented concepts
- Object oriented database enable to represent data in the form of object
- Some additional features
  - Unique Object identifiers
  - Persistent object handling

Cont’d

- Object oriented databases or object data model to define data structures.
- The relationship implicit to the object and manifests is a database that can store the objects in their entirety, and methods are as usable ever stored in the database.
- Databases incorporate the object on which database operations store objects rather than data between various data is as object attributes and method

Relational model and Object

Relational model

- Clean and simple.
- Great for administrative and transactional data.
- Not as good for other kinds of complex data (e.g., multimedia, networks, CAD).

Object-Oriented models

- Complicated, but some influential ideas from Object Oriented
- Complex data types.
- Idea: Build DBMS based on OO model.

First approach: object-oriented model

- Object-Oriented DBMS(OODBMS) are DBMS based on an Object Oriented Data Model inspired by OO programming languages
- Relations are not the central concept, classes and objects are the main concept
- OODBMS are capable of storing complex objects,
  - I.e., objects that are composed of other objects, and/or multi multi-valued attributes

Cont’d

Main Features object-oriented model :

- ✓ Powerful type system
- ✓ Classes
- ✓ Object Identity
- ✓ Inheritance

Feature 1: powerful type system

- **Primitive types**

✓ Integer, string, date, Boolean, float, etc.

- **Structure type**

✓ Attribute can be a record with a schema

- **Collection type**

✓ Attribute can be a Set, Bag, List, Array of other types

- **Reference type**

✓ Attribute can be a Pointer to another object

Feature 2: classes

- A ‘class’ is in replacement of ‘relation’ Same concept as in OO programming languages
- All objects belonging to a same class share the same properties and behaviour

✓ An ‘object’ can be thought of as ‘tuple’

✓ Classes encapsulate data + methods + relationships

✓ Unlike relations that contain data only

✓ In OODBMSs objects are persistency

Feature 3: object identity

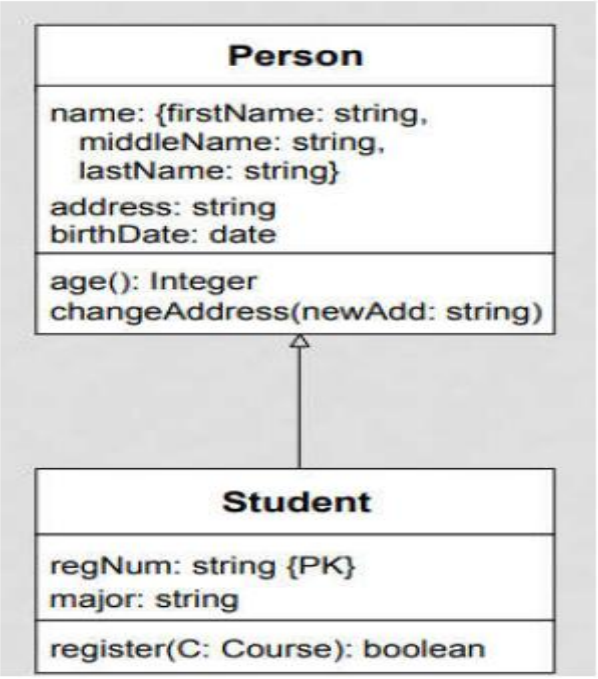
- OID is a unique identity of each object
- Even if all attributes are the same, easier for references
- An object is made of two things:

✓ State: attributes (name, address, birthDate

✓ Behaviour: operations (age of a person is computed from and current date)

Feature 4: inheritance

- A class can be defined in terms of another one.
- Person is super-class and Student is sub-class
- Student class inherits attributes and operations of Person



Structured Types and Inheritance in SQL

- The user will create the **user-defined types (**UDTs) for a particular application as part of the database schema
- **UDT may** be specified in its simplest form using the following syntax:

  CREATE TYPE TYPE_NAME AS (<component declarations>);

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

create type *Name* **as**

(first*name* **varchar**(20),

lastname **varchar**(20))

final

Con't

- Each entity type specifies one or more properties.
- Properties are functions that apply to the instances of the type

Create Type Person

(Name char(20),

Age integer ,

Address address)

not final

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

Cont'd

- 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*

Select Name(p), State(Address(p)
for each Person p
where Age(p) > 21

Cont'd

- **User-defined row types**

  create type *PersonType* **as** (
  name Name,
  address Address,
  dateOfBirth **date**)
  not final

- Can then create a **table** whose rows are a user-defined type

  create table *customer* **of** *PersonType*

Con't

- it is possible to use the concept of **ROW TYPE to directly create a structured attribute by using the** keyword **ROW**
- Alternative using **unnamed row types**.

```
create table person_r(
        name row(firstname  varchar(20),
                            lastname  varchar(20)),
        address      row(street     varchar(20),
                         city       varchar(20),
                         zipcode    varchar(20)),
        dateOfBirth date)
```

Type Inheritance

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

```
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))
```

Cont’d

- **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

Table Inheritance

- Tables created from subtypes can further be specified as **subtables**
- E.g. **create table** *people* **of** *Person;*
  create table *students* **of** *Student* **under** *people;*
  create table *teachers* **of** *Teacher* **under** *people;*
- Tuples added to a subtable are automatically visible to queries on the supertable
  - E.g. query on *people* also sees *students* and *teachers*.
  - Similarly updates/deletes on *people* also result in updates/deletes on subtables
  - To override this behaviour, use "**only** *people*" in query

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;
```

Advantages of OODBS

- Designer can specify the structure of objects and their behavior (methods)
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types
- Encapsulation of operations and user-defined methods