

Chapter 16

Object Databases

We have so far discussed quite a few things about relational database. Indeed, RDB has been widely used in modeling and manipulating data since 1980's because of the simplicity of the well-defined relational data model and its appropriateness. Tables are just the right representation for much of the data in the business world.

On the other hand, the relational data model is not as appropriate for some of the other "non-traditional" applications such as CAD, GIS etc.. Even for the business oriented data, RDB is not perfect.

An example

Consider the following relational schema that describes people:

`Person(SSN:String,Name:String,PhoneN:String,Child:String)`

Since a person could have several phone numbers and several children, the key of this relation has to be a combination of SSN, PhoneN, and Child.

Below is an instance of this schema:

SSN	Name	PhoneN	Child
111-22-3333	Joe Public	516-123-4567	222-33-4444
111-22-3333	Joe Public	516-345-6789	222-33-4444
111-22-3333	Joe Public	516-123-4567	333-44-5555
111-22-3333	Joe Public	516-345-6789	333-44-5555
222-33-4444	Bob Public	212-987-6543	444-55-6666
222-33-4444	Bob Public	212-987-1111	555-66-7777
222-33-4444	Bob Public	212-987-6543	555-66-7777
222-33-4444	Bob Public	212-987-1111	444-55-6666

You know what's wrong, don't you?

This table is not in 3NF, since one of its FDs: $SSN \rightarrow Name$, violates the condition: SSN alone is not a key of this table; and $Name$ is not part of the key.

We thus can go through the normalization theory to decompose this schema to three subschemas:

$Person(SSN, Name)$, $Phone(Person, PnoneN)$, and $ChildOf(SSN, Child)$

This leads to the following instances

SSN	Name
111-22-3333	Joe Public
222-33-4444	Joe

SSN	PhoneN
111-22-3333	516-123-4567
111-22-3333	516-345-6789
222-33-4444	212-987-6543
222-33-4444	212-987-1111

Homework: 16.1.

SSN	Child
111-22-3333	222-33-4444
111-22-3333	333-44-5555
222-33-4444	444-55-6666
222-33-4444	555-66-7777

Although all the redundancy is taken out, we still have difficulties with its.

Consider the following query “*get the phone numbers of Joe’s grandchildren.*,” the following gets it from the original table,

```
Select G.PhoneN From Person P, Person C, Person G
Where P.Name='Joe Public' And P.Child=C.SSN
      And C.Child=G.SSN
```

while the following gets it from the decomposition

```
Select N.PhoneN From ChildOf C, ChildOf G, Person P, Phone N
Where P.Name='Joe Public' And P.SSN=C.SSN And
      C.Child=G.SSN And G.SSN=N.SSN
```

Both are awkward(?).

What are the issues?

One issue is that both phone number and children are fundamentally set valued, and the RDB is not good at handling this sort of data.

A much more appropriate schema for the original table might be the following:

```
Person(SSN:String,Name:String,  
       PhoneN:{String},Child:{String}),
```

where {} indicates set valued attributes. Thus, the type for `Child` tells that the value for this attribute should be a set. One of its tuples looks like the following:

```
(111-22-3333, Joe Public,  
  {516-123-4567,516-345-6789},  
  {222-33-4444,333-44-5555})
```

Another one is...

that it is awkward to query such tables.

Assume that the type of `Child` is `Person` rather than `String`, then an SQL query can treat the value of the `Child` attribute as a set of `Person`.

For example, the expression `P.Child.Child` means the children of the children of a person `P`.

Thus, it will be possible for us to formulate a much more natural query as follows:

```
Select P.Child.Child.PhoneN
From Person P
Where P.Name='Joe Public'
```

IsA hierarchies

Assume that some but not all people in our *Person* database are students. Since a student is a person, we can represent this fact by drawing an arrow from the *Student* entity to the *Person* entity as we did when working with an E/R diagram.

Since RDB does not support the IsA concept, we have to do a simulation by throwing out the general information to the *Person* entity and keep only those specific for students in the *Student* entity. This leads to the following

`Student(SSN:String, Major:String)`

What to say?

Question: Will the following query work, if we want to get the names of all the CS majors?

```
Select S.Name From Student S  
Where S.Major='CS'
```

Answer: I would expect you to say “No”, although you might think since every student is a person, she should have a name.

Question: What should we say?

Answer: We have to get someone's name from the Person table via a natural join:

```
Select P.Name From Student S, Person P  
Where S.Major='CS' And P.SSN=S.SSN
```

Question: Is the order of the conjuncts in the Where clause important?

Answer: Depends on DBMS.

A few examples

We will look at a few examples in the context of SQL:1999 and SQL:2003.

```
Crate Table Person (  
  Name Char(20),  
  Address Row(Number Integer,Street Char(20),Zip(Char(5)))
```

We can then refer to a value in a tuple using the path expression, e.g., as follows:

```
Select P.Name From Person P  
Where P.Address.Zip='11794'
```

```
Insert Into Person(Name,Address)  
Values('John Doe',Row(666,'Russell St.', '03264'))
```

```
Update Person  
Set Address=Row(21,'Main St.', '03264')  
Where Address=Row(666,'Russell St.', '03264')  
  And Name='John Doe'
```

Let's have our own

By SQL:1999/2003, we can define our own *abstract data type*, or rather user defined type (UDT), just as we did in Java.

For example,

```
Create Type PersonType As (  
    Name Char(20),  
    Address Row(Number Integer, Street Char(20), Zip(Char(5)));
```

```
Create Type StudentType Under PersonType As (  
    Id Integer,  
    Status Char(2))  
    Method award_degree() Returns Boolean;  
    Create Method award_degree() For StudentType  
    Language C  
    External Name 'file:/home/admin/award_degree';
```

Here, we define `StudentType` as a subtype of `PersonType`, with the `Under` clause. It then inherits all the properties of `PersonType`.

In addition, `StudentType` has its own type, plus a signature for a method, which is then defined using the `Create Method...For...` clause, telling where its implementation can be found.

Now the tables

We can do business as usual, e.g.

```
Create table Transcript (  
  Student StudentType,  
  CrsCode Char(6),  
  Semester Char(5),  
  Grade Char(1))
```

We can also create a table using a UDT, where the table must have the exact structure as given in the UDT:

```
Create Table Student of StudentType;
```

The only way to create an object in SQL is via an insertion into a table, whence it is given its unique oid, and the table itself will be regarded as a class, and the set of the objects as its extent.

Inheritance

Since `StudentType` is a subtype of `PersonType`, although such attributes of `Name`, `Address` are not explicitly mentioned in the `StudentType`, they are implicitly inherited.

Thus, if we create the `Student` table as follows:

```
Create Table Student Of StudentType Under Person;
```

then, when we insert a tuple into the `Student` table, as follows:

```
Insert Into Student(Name,Address,Id,Status)
Values
('John Jones',Row(123,'Main St.',11733),1122,'C2')
```

it will be automatically inserted into the `Person` table as well.

Notice that in this case, the `Person` table has become a *super table* of the `Student` table.

OOP in PHP

PHP supports object-oriented programming. We just look at a few simple examples. For details, please check out Chapter 20 of the Bible.

The following defines a class and then creates an instance.

```
class TextBox (  
    var $body_text="my text";  
    //the constructor  
    function _constructor($text_in){  
        $this->body_text=$text_in;  
    }  
    function display(){  
        print("<table border=1><tr><td>  
            $this->body_text");  
        print("</td></tr></table>");  
    }  
    //Let's create an instance.  
    $box=new TextBox("customer box");  
    $box->display();
```

An inheritance example

We now declare a child object, extending the just defined TextBox:

```
class TextBoxHeader extends TextBox{
  var $header_text;
  //constructor
  function _constructor($head_text_in,
                        $box_text_in){
    $this->header_text=$head_text_in;
    $this->body_text=$body_text_in;
  }
  function display(){
    //its code
  }
  function make_header($text){
    return($text);
  }
  function make_body($text){
    return($text);
  }
}
```