Relational Models

Juliane Marubayashi

2022-04-19

Contents

1 Object-relational model **3**
1.1 integration OO and relational . . . 3
1.1.1 Class = Relational . . . 3
1.1.2 Example . . . 3
1.1.3 Class = Domain . . . 3
1.2 Introduction to SQL3 . . . 4
1.2.1 Hierachical data . . . 4
1.2.2 Objects . . . 5
1.2.2.1 Object types . . . 5
1.2.2.2 Domain definition . . . 5
1.2.2.3 Objects as table columns . . . 5
1.2.2.4 Object as table rows . . . 6
1.2.2.5 Access object-row address . . . 7
1.2.2.6 references (REF) . . . 7

1 Object-relational model

1.1 integration OO and relational

1.1.1 Class = Relational

A class is represented as a table and the table rows are objects.
The advantages are:

- It has support for composite columns. The column can have a type, for example;
- It support tables as attribute values;
- There is a support for methods associated with the table.

1.1.2 Example

In the example below, the `CLIENT` is a **composite** column, since it have multiple attributes inside the column type. The `CONTAIN` is a table used as an attribute value.

```
CREATE TABLE ORDERTYPE
(
    O# NUMERIC,
    CONTAIN SETOF(IQTYPE) -- Multiple values
    CLIENT CLIENTTYPE -- User defined type
)
```

1.1.3 Class = Domain

How can domains ensure that classes follows the four principles (i.e encapsulation, inheritance, polymorphism and abstraction) of object oriented programming?

Domains may contain any object: vectors, lists, documents, photography, etc. Domains are able to encapsulate infomation, while relations don't and class hierarchies and polymorphism have their place in the construction of domains.

1.2 Introduction to SQL3

In this section we gonna see in more details the implementation of an object oriented implementation in SQL3.

1.2.1 Hierachical data

To approach the hierarchicaly, let's start with a query: show all the records sorted by the hierarcy. The sql classes are structured in the following way:

```
District(code, name, region)
Municipality(code, name, district)
Parish(code, name, municipality)
```

In the end, the `district`, `municipality` and `parish` can be represented as the following class:

```
Division(code, name, parent)
```

And the query to retrieve the desired information is:

```
select code, name
from division
start with parent is null
connect by prior code=parent;
```

We started by filtering root rows, then the prior code referes to the code in the parent rows (the parent in Division(code, name, parent) can be interpreted as a foreign key) and then we connect by starting with the root and giving the subtrees rows in preorder.

Another practical example is:

```
select code, name, level
from division where level < 4
start with name like 'Aveiro%'
connect by prior code=parent;
```

In this case the table contains a column called `level` which stores the levels of each row. The result is something like this:

CODE	NAME	LEVEL
1	Aveiro	1
101	Águeda	2
10101	Agadão	3
10102	Aguada de Baixo	3
10103	Aguada de Cima	3
102	Albergaria-a-Velha	2

Figure 1: Query result

1.2.2 Objects

The main goal of an object is to deal with complex object. There're two ways that an object can be represented in a databases:

- It can be **store as a table column**
- As the **row of a column**

The advantages of using objects are:

- It **encapsulates** the data and thus can be **reused**;
- We are able to store the OO programming structures without conversion. It eases the development.
- Representation of part-of hierarchies. Hierarchies are important, since it might ease the representation of some structures and allow a better understandment.
- It's more efficient, since fetch a group of objects in a a single access.

1.2.2.1 Object types

Firstly, an object contains attributes and methods at the same time and is semantically richer than the convetional databases, but this can be also a disadvantage: it hides part of the structure (as it will be shown later) and the access should must be made through specific methods. Also, it's possible to have type inheritance.

1.2.2.2 Domain definition

A type can be seen as a domain (i.e a static infinite set where the column may pick values).

```
create type address_t as object(
    street varchar2(30),
    zipcode varchar2(10),
    town varchar2(20),
    countryu varchar2(20)
);

create type position as object(
    position varchar2(20),
    start date,
    end date
);
```

The `varchar2` is different from `varchar`. The `varchar` sould not be used and is an internal type of oracle, while `varchar2` is an external variable and can be used by the user. More than that, `varchar2` does not distinguish between a `NULL` string and an empty string (i.e null = ''), however `varchar` makes this difference.

1.2.2.3 Objects as table columns

Types can be used as domains in table columns an in this case the objects are in the table columns.

```
create table members (
    ID number(10) primary key,
    name varchar2(20),
    address address_t,
    position position_t
);
```

Let's see and example of how to add a row for the `members` table:

```
insert into members(
    3658,
    'Besouro'
    address_t(
        "Rua de Soares dos Reis",
        "4400-163",
        "Vila Nova de Gaia",
        "Portugal"
    ),
    position_t(
        "President",
        "2010-01-01",
        "2012-12-31"
    )
);
```

To perform **queries** and select an object, a cursor is necessary:

```
select name, m.address.street, position
from members m where id=3658;
```

1.2.2.4 Object as table rows

```
create type department_t as object(
    acronym varchar2(5),
    designation varchar2(20),
    director number(10)
);

create deparments of deparment_t;
```

This is table was generated from type defined by the user and **contains objects**. This is not a normal table, since the object-row occupies the full row, while the object columns don't. The table **can be seen as a single column** of objects or multicolumn, like a relational table:

```
insert into departments values('CUL', 'Cultural', 3658); -- A multicolumn table
insert into departments values(department_t('BIB', 'Biblioteca', 3658)); -- A single column
```

This is an example of the relation class=table approach.

There're two ways that the values o the table can be retrieved. One of the ways of retrieving the information.

```
select * from departments where director=3658;
```

The result be something like this:

ACRONYM	DESIGNATION	DIRECTOR
CUL	Cultural	3658
BIB	Biblioteca	3658

And the other way is:

```
select value(d) from departments d where director=3658;
```

In this result we have an object view:

VALUE(D)
[DEPARTMENT_T]
[DEPARTMENT_T]

1.2.2.5 Access object-row address

The object rows possess address, while objects in columns don't.

```
select ref(d) from departments d;
```

The visualization, however, is automatically converted by SQL. In the result should be the same as performing a query with `value(d)`. This change may not affect the visualization, but might be useful programatically as we gonna see in the next.

1.2.2.6 references (REF)

What is the advantage of using references (REF)?
The explain this question, let's first change the object column `address` in the `members` table to a reference to the class table `department_t`.

```
-- Remembering what the format of the member row.
create table members (
    ID number(10) primary key,
    name varchar2(20),
    address address_t,
    position position_t
);

alter table members add (department ref department_t);
```

Now let's suppose we want to update department of some members:

```
update members set department = (select ref(d) from departments d where sigla='CUL')
where id=3658;
```

The advantage in this query is that we can have access to the table `departments` without performing a join.