

Documentation - Data Bases 2

Marco Fasanella

MsC Computer Science, Polimi
C.P. 10617541

Nicola Dean

MsC Computer Science, Polimi
C.P. 106---

This document describes design and implementation of Data Base 2 Project

1. Specifications

A telco company offers pre-paid online services to web users. Two client applications using the same database have been developed: a costumer application and an employee application.

1.1 Extra hypothesis

2. ER Diagram

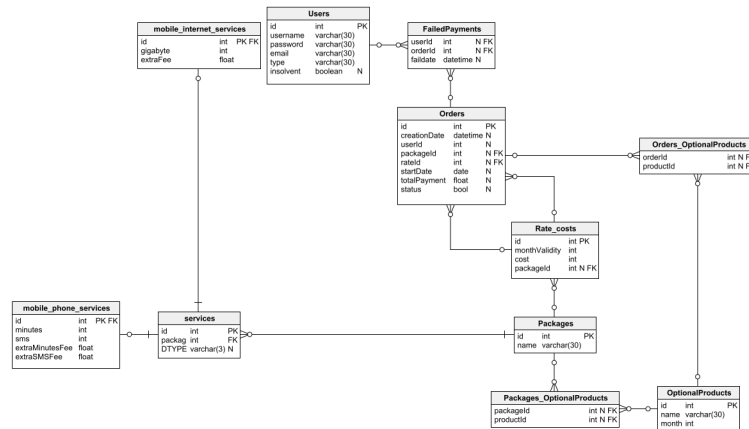


Figure 1. ER Diagram

■ 3. SQL Description

Detailed description of SQL code used in the project.

■ 3.1 Views

The following views are used for various Sales Reports.

Joining *Orders* and *Rate_costs* tables we performed a *selection* for each distinct package (depending on their month validity) and a *count* for their occurrences.

```
create view PurchasesCount as (  
    Select Packages.name as name, Rate_costs.monthValidity as validity, count(*) as count  
    from Orders as o join Packages join Rate_costs  
    on o.packageId=Packages.id and Rate_costs.packageId=o.packageId  
    and o.rateId=Rate_costs.id  
    group by o.packageId, Rate_costs.id  
);
```

Then for a less detailed view, from *PurchasesCount*, a *group by* on the name of the package gives a count of each one.

```
create view PurchasesCountGrouped as (  
    select p.name as name, sum(p.count) as count  
    from PurchasesCount as p  
    group by p.name  
);
```

To count the Optional Products was necessary a *left join* because it could exist a Package without any Optional Product, and as a consequence it wouldn't be stored in *Orders_OptionalProducts* table.

```
create view OptionalProductsCount as(  
    select o.packageId as packageId, count(opt.productId) as optcount  
    from Orders as o left join Orders_OptionalProducts as opt  
    on o.id=opt.orderId  
    group by o.id  
);
```

Then to have the average count of them, we performed an *avg* on the count of previous view.

```
create view OptionalProductsAverage as(  
    select p.name as name, avg(opc.optcount) as avg  
    from OptionalProductsCount as opc join Packages as p  
    where opc.packageId=p.id  
    group by id  
);
```

Following view sums the *totalPayment* of each Order grouped by *packageId*

```
create view ValueOfTotalSales as(
    select p.name as name, sum(o.totalPayment) as totalPayment
    from Orders as o join Packages as p
    where p.id=o.packageId
    group by o.packageId
);
```

Then in *OptionalProductsSales* the same method is used to get the totalCost of the Optional Products related to their corresponding Service Package in *Orders*

```
create view OptionalProductsSales as(
    select p.name as name, sum(op.monthlyFee*r.monthValidity) as totalOptionalProductsSales
    from Orders_OptionalProducts as orderop join Orders as o
    join OptionalProducts as op join Packages as p join Rate_costs as r
    where o.id=orderop.orderId and orderop.productId=op.id
    and o.packageId=p.id and o.rateId=r.id
    group by p.id
);
```

And finally these two views are *left joined* to have both *totalPayment* with and without Optional Products. A *left join* is required because as mentioned before it could exist a service package without Optional Product. In this case *totalPaymentWithoutOP* will return *null*.

```
create view ValueOfSalesDetailed as(
    select vot.name as name, vot.totalPayment as totalPayment,
           (vot.totalPayment-ops.totalOptionalProductsSales) as totalPaymentWithoutOP
    from ValueOfTotalSales as vot left join OptionalProductsSales as ops
    on vot.name=ops.name
);
```

In *InsolventReport* are selected all Users with *having count(*)=3* of insolvent orders stored in *FailedPayments*.

```
create view InsolventReport as(
    select u.id as id, u.username as username, u.email as email,
           (
            select max(fp1.faildate)
            from FailedPayments as fp1
            where fp1.userId=u.id) as lastDate,
           (
            select o.totalPayment
            from Orders as o join FailedPayments as fp2
            where o.id=fp2.orderId and lastdate=fp2.faildate) as amount
    from FailedPayments as fp join Users as u
```

```

where fp.userId=u.id
group by u.id
having count(*)>=3
);

```

Last view *OptionalProductBestSeller* is used to have a count and the value of total sales of each Optional Product.

```

create view OptionalProductBestSeller as(
    select op.name as name,count(op.id) as amountSold,
           sum(op.monthlyFee*r.monthValidity) as value
    from Orders_OptionalProducts as ordop join OptionalProducts as op
    join Rate_costs as r join Orders as o
    where ordop.productId=op.id and o.rateId=r.id and o.id=ordop.orderId
    group by op.id
);

```

The using two distinct queries we performed the *OptionalProductBestSeller-ForValue*

```

select o
from OptionalProductBestSeller o
where o.value=(
select max(o2.value)
from OptionalProductBestSeller o2
);

```

and *OptionalProductBestSellerForAmount*

```

select o from OptionalProductBestSeller o
where o.amountSold=(
select max(o2.amountSold)
from OptionalProductBestSeller o2
);

```

■ 3.2 Triggers

```

create trigger INSOLVENT_USER
after insert on Orders
for each row
begin
    if ( new.status = false) then
        update Users set Users.insolvent = true where Users.id = new.userId;
        insert into FailedPayments (userId,orderId,faildate)
        values (new.userId,new.id,CURRENT_TIMESTAMP);
    end if;

create trigger INSOLVENT_USER_REMOVAL

```

```

    after update on Orders
    for each row
begin
    if (new.status = true) AND
        (select count(*) from Orders as o where o.userId=new.userId and o.status = false) = 0
    then
        update Users set Users.insolvent = false where Users.id = new.userId;
    end if;
    if (new.status = false AND old.status = new.status) then
    insert into FailedPayments (userId,orderId,faildate)
values (new.userId,new.id,CURRENT_TIMESTAMP);
    end if;

```

4. ORM Description

4.1 Entities

Optional Product

```

@Entity
@Table(name = "OptionalProducts", schema = "test")
public class OptionalProduct{

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    int id;
    String name;
    int monthlyFee;

```

Order The Orders entity

```

@Entity
@Table(name = "Orders", schema = "test")
@NamedQuery(name="Orders.Id" , query="select o from Order o
    where o.Id = :orderId")
@NamedQuery(name="Orders.All" , query="select o from Order o")
//TODO adjust query for only accepted orders
@NamedQuery(name="Orders.Suspended" , query="select o from Order o
    where o.status=false")
@NamedQuery(name="Orders.RemoveSuspend" , query="update Order o set
    o.status = true where o.Id=:orderId")
@NamedQuery(name="PurchasesByPackages" , query = "select count
    (distinct o) from Order o group by o.pack")
@NamedQuery(name="PurchasesByPackagesID" , query = "select
    o.pack.name, count(o.pack) from Order o where o.status=true group
    by o.pack ")

```

```

@NamedQuery(name="Orders.UserInsolvances" , query = "select o from
    Order o where o.status=false and o.user.id = :userId")
//TODO think a new entity for statistical pourpose should be created
    to store pair "package-> quantity"

```

```

public class Order {

```

```

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    int Id;
    Date startDate;
    Date creationDate;
    float totalPayment;
    Boolean status =null;

```

RateCost

```

@Entity
@Table(name="Rate_costs", schema = "test")
public class RateCost {

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    int id;
    int monthValidity;
    int cost;
    int packageId;

```

Service

```

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "services", schema = "test")
public class Service {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;
    int packageId;
    @Column(name = "DTYPE")
    String type;

```

User

```

@Entity
@NamedQuery(name = "User.authentication", query = "select usr from
    User usr WHERE usr.username = :username and usr.password =
    :password")

```

```
@NamedQuery(name = "User.insolvent" , query = "select usr from User
    usr WHERE usr.insolvent = true")
@Table(name="Users", schema = "test")
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;

    String username;
    String password;
    String email;
    String type;
    boolean insolvent;
```

5. Application Components

6. UML sequence diagrams
