Data Bases 2

Project presentation

Giovanni De Novellis & Carlo Lazzari

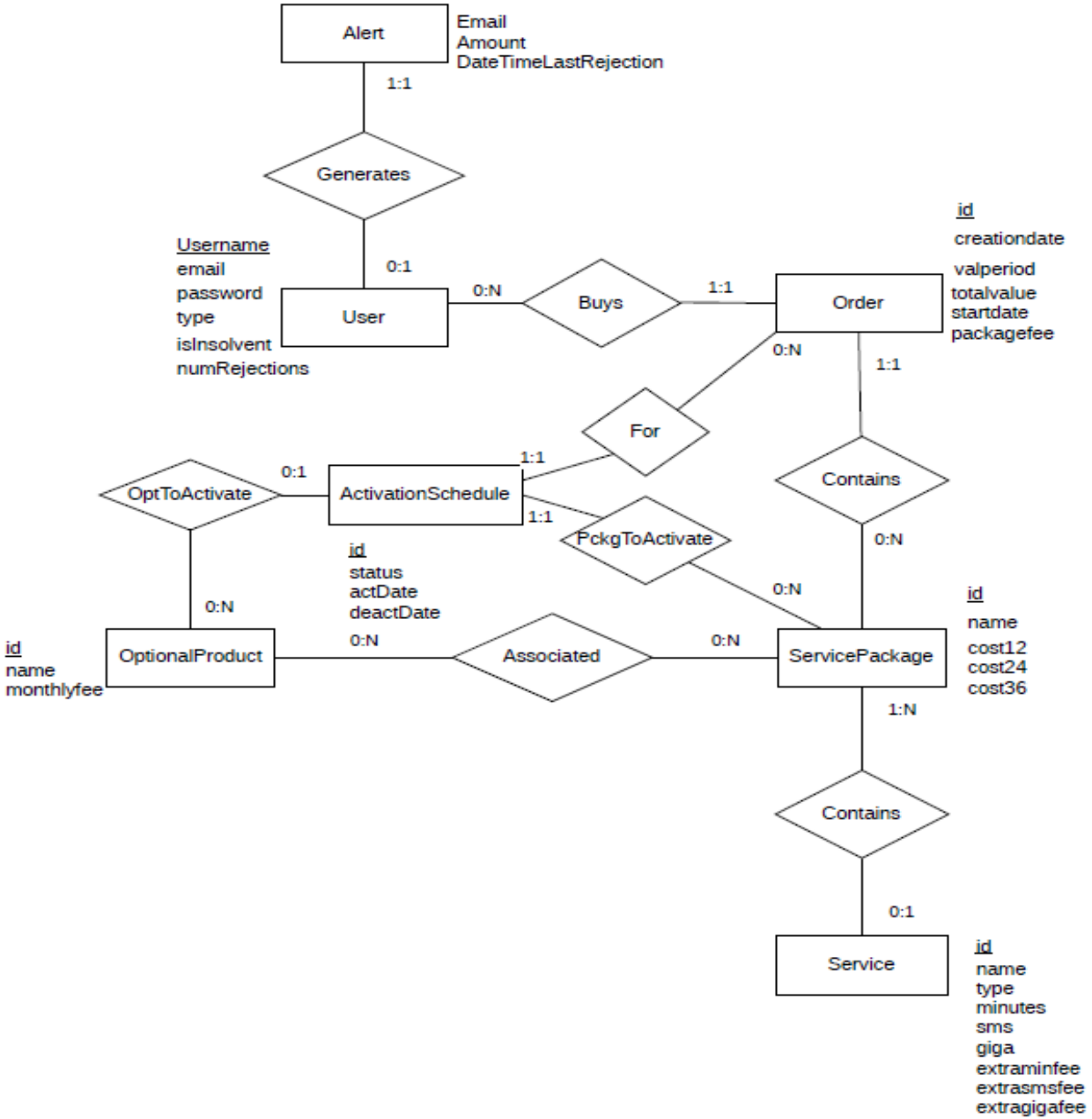
Specification summary

- A telco company offers pre-paid online services to web users. Two client applications using the same database need to be developed.
- The customer application allows the client to login, register, place orders and purchase service packages with various services and optional products tied with the order.
- The employee application allows the customer to login, create new services, service packages, optional products and check data on the sales report page.

Specification Interpretations

- A service can be associated to only one package, while optional products can be associated to multiple packages.
- An order can be either in a Valid, if the payiment succeded or Suspended if the payimen failed. A Valid order can’t become Suspended.
- After the third failed payment for the user, an Alert is created and is updated after every payment fail for the same user. Every «version» of the Alert is saved in an Alert History table by means of triggers.

Er Diagram


Alert
Email
Amount
DateTimeLastRejection
1:1
Generates
0:1
Username
email
password
type
isInsolvent
numRejections
User
0:N
Buys
1:1
Order
id
creationdate
valperiod
totalvalue
startdate
packagefee
0:N
1:1
For
Contains
1:1
0:1
OptToActivate
ActivationSchedule
1:1
PckgToActivate
0:N
id
status
actDate
deactDate
0:N
0:N
id
name
cost12
cost24
cost36
0:N
id
name
monthlyfee
OptionalProduct
0:N
Associated
0:N
ServicePackage
1:N
Contains
0:1
Service
id
name
type
minutes
sms
giga
extraminfee
extrasmsfee
extragigafee

Er comments

- The activation schedule can have 0 optional products associated when no product has been bought and only the services have to be activate.
- The tables regarding the materialized views have been omitted from the diagram for simplicity, their structure is usually the id of the table they are referring to and the aggregate data computed. The triggers that compute them act on the tables presented in the diagram.

Logical model: DDL Code Activation schedule

```
CREATE TABLE activation-schedule (
  package int NOT NULL,
  orderid int NOT NULL,
  optproduct int DEFAULT NULL,
  actdate datetime DEFAULT NULL,
  deactdate datetime DEFAULT NULL,
  bridge_id int NOT NULL AUTO_INCREMENT,
  status varchar(45) NOT NULL,
  PRIMARY KEY (bridge_id),
  UNIQUE KEY bridge_id_UNIQUE (bridge_id),
  KEY package_idx (package),
  KEY order_idx (orderid),
  KEY product_idx (optproduct),
  CONSTRAINT orderid FOREIGN KEY (orderid) REFERENCES order (id),
  CONSTRAINT package FOREIGN KEY (package) REFERENCES service-package (ID),
  CONSTRAINT product FOREIGN KEY (optproduct) REFERENCES optional-product (id)
)
```

Alert

```
CREATE TABLE alert (
  username varchar(45) NOT NULL,
  email varchar(45) NOT NULL,
  amount float NOT NULL,
  datetimelastrejection datetime NOT NULL,
  PRIMARY KEY (username),
  KEY mail_idx (email),
  CONSTRAINT alertusername FOREIGN KEY (username) REFERENCES user (username),
  CONSTRAINT mail FOREIGN KEY (email) REFERENCES user (email)
)
```

Optional Product

```
CREATE TABLE optional-product (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(45) NOT NULL,
  monthlyfee float NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY id_UNIQUE (id)
)
```

Order

```
CREATE TABLE order (
  id int NOT NULL AUTO_INCREMENT,
  creationdate datetime NOT NULL,
  valperiod int NOT NULL,
  totalvalue int NOT NULL,
  startdate datetime NOT NULL,
  status varchar(45) NOT NULL,
  username varchar(45) DEFAULT NULL,
  packageid int DEFAULT NULL,
  fee float NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY id_UNIQUE (id),
  KEY username_idx (username),
  KEY packageid_idx (packageid),
  CONSTRAINT packageid FOREIGN KEY (packageid) REFERENCES service-package (ID),
  CONSTRAINT username FOREIGN KEY (username) REFERENCES user (username)
)
```

Package opt association

```
CREATE TABLE package-opt-association (
  packageid int NOT NULL,
  optprodid int NOT NULL,
  PRIMARY KEY (packageid,optprodid),
  KEY assproduct_idx (optprodid),
  CONSTRAINT asspackage FOREIGN KEY (packageid) REFERENCES service-package (ID),
  CONSTRAINT assproduct FOREIGN KEY (optprodid) REFERENCES optional-product (id)
)
```

Service

```
CREATE TABLE service (
  serviceid int NOT NULL AUTO_INCREMENT,
  type varchar(45) NOT NULL,
  minutes int DEFAULT '0',
  sms int DEFAULT '0',
  extraminfee float DEFAULT '0',
  extrasmsfee float DEFAULT '0',
  giga int DEFAULT '0',
  extragigafee float DEFAULT '0',
  service_package_id int DEFAULT NULL,
  name varchar(45) NOT NULL,
  PRIMARY KEY (serviceid),
  UNIQUE KEY serviceid_UNIQUE (serviceid),
  KEY service_package_id_idx (service_package_id),
  CONSTRAINT service_package_id FOREIGN KEY (service_package_id) REFERENCES service-package (ID)
)
```

Service package

```
CREATE TABLE service-package (
  ID int NOT NULL AUTO_INCREMENT,
  name varchar(45) NOT NULL,
  monthscost12 float NOT NULL,
  monthscost24 float NOT NULL,
  monthscost36 float NOT NULL,
  PRIMARY KEY (ID,name),
  UNIQUE KEY ID_UNIQUE (ID)
)
```

User

```
CREATE TABLE user (
  username varchar(64) NOT NULL,
  email varchar(64) NOT NULL,
  password varchar(64) NOT NULL,
  type varchar(8) NOT NULL,
  isInsolvent varchar(1) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL,
  numrejections int DEFAULT '0',
  PRIMARY KEY (username),
  UNIQUE KEY username_UNIQUE (username),
  UNIQUE KEY email_UNIQUE (email)
)
```

Materialized views logical model: alert history

```
CREATE TABLE `alert-history` (
  `username` varchar(64) NOT NULL,
  `amount` float NOT NULL,
  `datetimerejection` datetime NOT NULL,
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `usernickname_idx` (`username`),
  CONSTRAINT `user` FOREIGN KEY (`username`) REFERENCES `user` (`username`)
)
```

Materialized views logical model: avg-opt-for-package

```
CREATE TABLE `avg-opt-for-package` (
  `servicePackage` int NOT NULL,
  `numopttot` int DEFAULT '0',
  `numsales` int NOT NULL DEFAULT '0',
  `avgoptforsale` float DEFAULT '0',
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `avgpckgid_idx` (`servicePackage`),
  CONSTRAINT `avgpckgid` FOREIGN KEY (`servicePackage`) REFERENCES `service-package` (`ID`)
)
```

Materialized views logical model: best-opt-product

```
CREATE TABLE `best-opt-product` (
  `productid` int NOT NULL,
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `opprod_idx` (`productid`),
  CONSTRAINT `opprod` FOREIGN KEY (`productid`) REFERENCES `optional-product` (`id`)
)
```

Materialized views logical model: insolvent-users

```
CREATE TABLE `insolvent-users` (
  `idinsolventuser` varchar(64) NOT NULL,
  PRIMARY KEY (`idinsolventuser`),
  CONSTRAINT `idinsolvent` FOREIGN KEY (`idinsolventuser`) REFERENCES `user` (`username`) ON DELETE CASCADE ON UPDATE CASCADE
)
```

Materialized views logical model: num-purch-package

```
CREATE TABLE `num-purch-package` (
  `packageid` int NOT NULL,
  `numpurchases` int DEFAULT '0',
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `packageid_idx` (`packageid`),
  CONSTRAINT `pkid` FOREIGN KEY (`packageid`) REFERENCES `service-package` (`ID`)
)
```

Materialized views logical model: num-purch-package-val-period

```
CREATE TABLE `num-purch-package-val-period` (
  `packageid` int NOT NULL,
  `valperiod` int NOT NULL,
  `numpurchases` int DEFAULT '0',
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `packid_idx` (`packageid`),
  CONSTRAINT `packid` FOREIGN KEY (`packageid`) REFERENCES `service-package` (`ID`)
)
```

Materialized views logical model: sales-package

```
CREATE TABLE `sales-package` (
  `servicePackage` int NOT NULL,
  `totalwithopt` float DEFAULT '0',
  `totalwithoutopt` float DEFAULT '0',
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `spid_idx` (`servicePackage`),
  CONSTRAINT `spid` FOREIGN KEY (`servicePackage`) REFERENCES `service-package` (`ID`)
)
```

Materialized views logical model: suspended-orders

```
CREATE TABLE `suspended-orders` (
  `idsuspendedorders` int NOT NULL,
  `id` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`),
  KEY `ordid` (`idsuspendedorders`),
  CONSTRAINT `ordid` FOREIGN KEY (`idsuspendedorders`) REFERENCES `order` (`id`)
)
```

Views

```
create view salesPackageWithoutOpt as
        select sum(o.fee*o.valperiod) as sum1 , o.packageid from new_schema.order o
   where o.status='Valid'
   group by o.packageid;

create view salesPackageOnlyOpt as
        select sum(opt.monthlyfee*ord.valperiod) as sum2, p.id from new_schema.`activation-
schedule` a join new_schema.`optional-product` opt join new_schema.`service-package` p
        join new_schema.`order` ord
        on a.optproduct = opt.id and a.package=p.id and a.orderid=ord.id
        where a.status = 'Valid' and a.optproduct is not null
        group by p.ID;

create view salesPackageWithOpt as
        select s1.sum1+s2.sum2 as totalVal, s1.packageid
   from salesPackageWithoutOpt s1 join SalesPackageOnlyOpt s2
   on s1.packageid=s2.id;
```

Views

```
create view numProdForPackage as
SELECT count(*), a.package FROM new_schema.`activation-schedule` a
WHERE a.optproduct is not null
GROUP BY a.package;

create view numpurchpackage as
SELECT packageid, COUNT(packageid) AS numpurchases
FROM new_schema.order
GROUP BY packageid;

CREATE view numpurchpackagevalperiod as
SELECT packageid, COUNT(packageid) AS numpurchases, valperiod
FROM new_schema.order
GROUP BY packageid;
```

Views

```
CREATE view salesoptionalproduct as
SELECT optproduct AS optproductid,
sum(monthlyFee * valperiod) AS totalsalesvalue, package AS servicepackageid
FROM new_schema.`activation-schedule` JOIN new_schema.`order` ORD ON orderid = ORD.id
JOIN new_schema.`optional-product` OPT ON optproduct = OPT.id
GROUP BY optproduct;

CREATE VIEW bestoptproductview as
SELECT optproductid, totalsalesvalue FROM salesoptionalproduct
LIMIT 1;

create view numProdForPackage as
SELECT count(*) as numProd, a.package as package
FROM new_schema.`activation-schedule` a
WHERE a.optproduct is not null
GROUP BY a.package;

create view avgOptForPackage as
SELECT (n2.numpurchases/n1.numProd) as average, n1.package
FROM numProdForPackage n1 join numpurchpackage n2 on n1.package=n2.packageid;
```

Triggers design and code

- List of triggers:
  - A trigger that updates the num-purch-package table whenever a new service package has been bought, with information the service package id, and number of purchases.
  - A trigger that updates the num-purch-package-val-period table whenever a new service package has been bought, with information the service package id, number of purchases, and validity period.
  - A trigger that updates the total value of sales for each service package, whenever a new service package has been bought, with and without possible optional products.
  - A trigger that updates the avg-opt-for-package table; whenever a new order is finalized, ie status is changed to valid this trigger updates the table by updating the number of sales and the average number of optional product for each sale.

- A trigger that updates the table alert; when an order fails, the user is flagged as insolvent. If the order fails for three times in a row, an alert is created with information about the order. Another trigger keeps track of previous alerts by inserting the entry on alert-history too. (?)
- A trigger that updates the table insolvent-users, which keeps track of which users are currently insolvent(the order a user placed fails). Whenever a new user becomes insolvent, his id gets added to the table via a trigger and when he is not insolvent anymore, the entry gets deleted from the table.
- A trigger that updates the table suspended-orders; whenever a new order is placed but the purchase fails, its id gets added to the table via an after-insert (on order) trigger. When the status of the order changes to valid, with an after-update trigger the id is removed from the table suspended-order.
- A trigger that updates the table best-opt-product. Whenever a new order is finalized and before the sales-optional-product is updated, it updates and keeps track of the id of the optional product with the highest number of sales.

Trigger code: activation-schedule

```
CREATE DEFINER=`root`@`localhost` TRIGGER `activation-schedule_AFTER_INSERT` AFTER INSERT ON `activation-schedule` FOR EACH ROW
BEGIN
        IF new.status = 'Valid' AND new.optproduct IS NOT NULL
        THEN
                UPDATE `new_schema`.`avg-opt-for-package`
                SET numopttot = numopttot + 1,
                avgoptforsale = numopttot / numsales
                WHERE servicePackage = new.package;

                UPDATE `new_schema`.`sales-package`
                SET totalwithopt = totalwithopt + (SELECT monthlyfee FROM `optional-product` WHERE ID = new.optproduct) *
                        (SELECT valperiod from `order` WHERE id = new.orderid) WHERE servicePackage = new.package;

                UPDATE `new_schema`.`sales-optional-product`
                SET totalsalesvalue = totalsalesvalue + (SELECT monthlyfee FROM `optional-product` WHERE ID = new.optproduct) *
                        (SELECT  valperiod from `order` WHERE id = new.orderid)  WHERE optproductid = new.optproduct;
        END IF;
END
```

```
CREATE DEFINER=`root`@`localhost` TRIGGER `activation-schedule_AFTER_INSERT` AFTER UPDATE ON `activation-schedule` FOR EACH ROW
BEGIN
        IF new.status = 'Valid' AND old.status <> 'Valid'  AND new.optproduct IS NOT NULL
        THEN
                UPDATE `new_schema`.`avg-opt-for-package`
                SET numopttot = numopttot + 1,
                avgoptforsale = numopttot / numsales
                WHERE servicePackage = new.package;

                UPDATE `new_schema`.`sales-package`
                SET totalwithopt = totalwithopt + (SELECT monthlyfee FROM `optional-product` WHERE ID = new.optproduct) *
                        (SELECT valperiod from `order` WHERE id = new.orderid) WHERE servicePackage = new.package;

                UPDATE `new_schema`.`sales-optional-product`
                SET totalsalesvalue = totalsalesvalue + (SELECT monthlyfee FROM `optional-product` WHERE ID = new.optproduct) *
                        (SELECT  valperiod from `order` WHERE id = new.orderid)  WHERE optproductid = new.optproduct;
        END IF;
END
```

Activation-schedule triggers motivation

When an activation schedule row is inserted or updated, with the new status as ‘Valid’, it means the order has been paid and the data of that order can be counted in the sales data. Therefore, there are triggers that will update the materialized views concerning the optional product to be activated in the considered schedule line.

Trigger code: alert

```
CREATE DEFINER=`root`@`localhost` TRIGGER `alert_AFTER_INSERT` AFTER INSERT ON `alert` FOR EACH ROW BEGIN
    INSERT INTO `new_schema`.`alert-history`(username, amount, datetimerejection)
    VALUES (new.username, new.amount, new.datetimelastrejection);
END

CREATE DEFINER=`root`@`localhost` TRIGGER `alert_AFTER_UPDATE` AFTER UPDATE ON `alert` FOR EACH ROW BEGIN
    INSERT INTO `new_schema`.`alert-history`(username, amount, datetimerejection)
    VALUES (new.username, new.amount, new.datetimelastrejection);
END
```

Motivations

When an user fails to pay more than 3 times, the system will generate a new alert for any failed payiment. Since the alert is overwritten every time with the new date and time of the last rejection, there are triggers responsible for saving the old value in a dedicated Alert-History table, where i have the complete list of all alerts generated by all the users.

Trigger code: optional-product

```
CREATE DEFINER=`root`@`localhost` TRIGGER `optional-product_AFTER_INSERT` AFTER INSERT ON `optional-product` FOR EACH ROW BEGIN
    INSERT INTO `new_schema`.`sales-optional-product`(optproductid, totalsalesvalue)
    VALUES(new.id, 0);
END
```

Motivations

When an user fails to pay more than 3 times, the system will generate a new alert for any failed payiment. Since the alert is overwritten every time with the new date and time of the last rejection, there are triggers responsible for saving the old value in a dedicated Alert-History table, where i have the complete list of all alerts generated by all the users.

Trigger code: order

```
CREATE DEFINER=`root`@`localhost` TRIGGER `order_AFTER_INSERT` AFTER INSERT ON `order` FOR EACH ROW BEGIN
    IF new.status = 'Valid'
    THEN
    UPDATE `new_schema`.`num-purch-package`
        SET numpurchases = numpurchases + 1
        WHERE packageid = new.packageid;

        UPDATE `new_schema`.`num-purch-package-val-period`
        SET numpurchases = numpurchases + 1
        WHERE packageid = new.packageid
        AND valperiod = new.valperiod;

        UPDATE `new_schema`.`sales-package`
        SET totalwithoutopt = totalwithoutopt + new.fee * new.valperiod
        WHERE servicePackage = new.packageid;
```

```
    UPDATE `new_schema`.`sales-package`
        SET totalwithopt = totalwithopt + new.fee * new.valperiod
        WHERE servicePackage = new.packageid;

UPDATE `new_schema`.`avg-opt-for-package`
        SET numsales = numsales + 1,
        avgoptforsale = numopttot / numsales
        WHERE servicePackage = new.packageid;

    END IF;

    IF new.status = 'Suspended' THEN
        INSERT INTO `new_schema`.`suspended-orders`(idsuspendedorders)
        VALUES(new.id);
    END IF;

END
```

Trigger code: order

```
CREATE DEFINER=`root`@`localhost` TRIGGER `order_AFTER_UPDATE` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
    IF new.status = 'Valid' AND old.status <> 'Valid'
    THEN
        UPDATE `new_schema`.`num-purch-package`
        SET numpurchases = numpurchases + 1
        WHERE packageid = new.packageid;

        UPDATE `new_schema`.`num-purch-package-val-period`
        SET numpurchases = numpurchases + 1
        WHERE packageid = new.packageid
        AND valperiod = new.valperiod;

        UPDATE `new_schema`.`sales-package`
        SET totalwithoutopt = totalwithoutopt + new.fee * new.valperiod
        WHERE servicePackage = new.packageid;
```

•

```
    UPDATE `new_schema`.`sales-package`
    SET totalwithopt = totalwithopt + new.fee * new.valperiod
    WHERE servicePackage = new.packageid;

    UPDATE `new_schema`.`avg-opt-for-package`
    SET numsales = numsales + 1,
    avgoptforsale = numopttot / numsales
    WHERE servicePackage = new.packageid;

  END IF;

  IF old.status = 'Suspended' AND new.status <> 'Suspended'
  AND new.id IN (SELECT idsuspendedorders FROM `suspended-orders`)
  THEN
    DELETE FROM `new_schema`.`suspended-orders`
    WHERE idsuspendedorders = new.id;
  END IF;

END
```

Order triggers motivation

When an order is created with a Valid state, or its status is changed to Valid, it means that the order can be counted in the sales data. Therefore, the triggers willl update the number of purchases and the total sales value for the order’s package.

Also, if an order is inserted with a Suspended order there is a trigger that will insert it into the suspended orders list and a trigger to remove it if it is being updated from a Suspended to a Valid state.

Trigger code: sales-optional-product

```
CREATE DEFINER=`root`@`localhost` TRIGGER `sales-optional-product_BEFORE_UPDATE` BEFORE UPDATE ON `sales-optional-product` FOR EACH ROW BEGIN
    IF NOT EXISTS (SELECT * FROM `best-opt-product`)
    THEN
        INSERT INTO `new_schema`.`best-opt-product`(productid)
        VALUES(new.optproductid);
    ELSEIF new.totalsalesvalue > (SELECT totalsalesvalue FROM `sales-optional-product`
            WHERE optproductid = (SELECT productid from `best-opt-product`))
    THEN
        UPDATE `new_schema`.`best-opt-product`
        SET productid = new.optproductid;
    END IF;
END
```

Motivation

In this table we update the total value of sales generated by an optional product. Therefore, every time we update it, we have to check if the updated product has become the one with the greatest value, and eventually update the best optional product table.

Trigger code: service package

```
CREATE DEFINER=`root`@`localhost` TRIGGER `service-package_AFTER_INSERT` AFTER INSERT ON `service-package` FOR EACH ROW BEGIN
   INSERT INTO `new_schema`.`num-purch-package`(packageid, numpurchases)
   VALUES(new.ID, 0);
   INSERT INTO `new_schema`.`num-purch-package-val-period`(packageid, numpurchases, valperiod)
   VALUES(new.ID, 0, 12);
   INSERT INTO `new_schema`.`num-purch-package-val-period`(packageid, numpurchases, valperiod)
   VALUES(new.ID, 0, 24);
   INSERT INTO `new_schema`.`num-purch-package-val-period`(packageid, numpurchases, valperiod)
   VALUES(new.ID, 0, 36);
   INSERT INTO `new_schema`.`sales-package`(servicePackage, totalwithopt, totalwithoutopt)
   VALUES(new.ID, 0, 0);
   INSERT INTO `new_schema`.`avg-opt-for-package`(servicePackage, numopttot, numsales, avgoptforsale)
   VALUES(new.ID, 0, 0, 0);
END
```

Motivation

When i create a new service package, i have to create the corresponding rows on the materialized view tables.

Trigger code: user

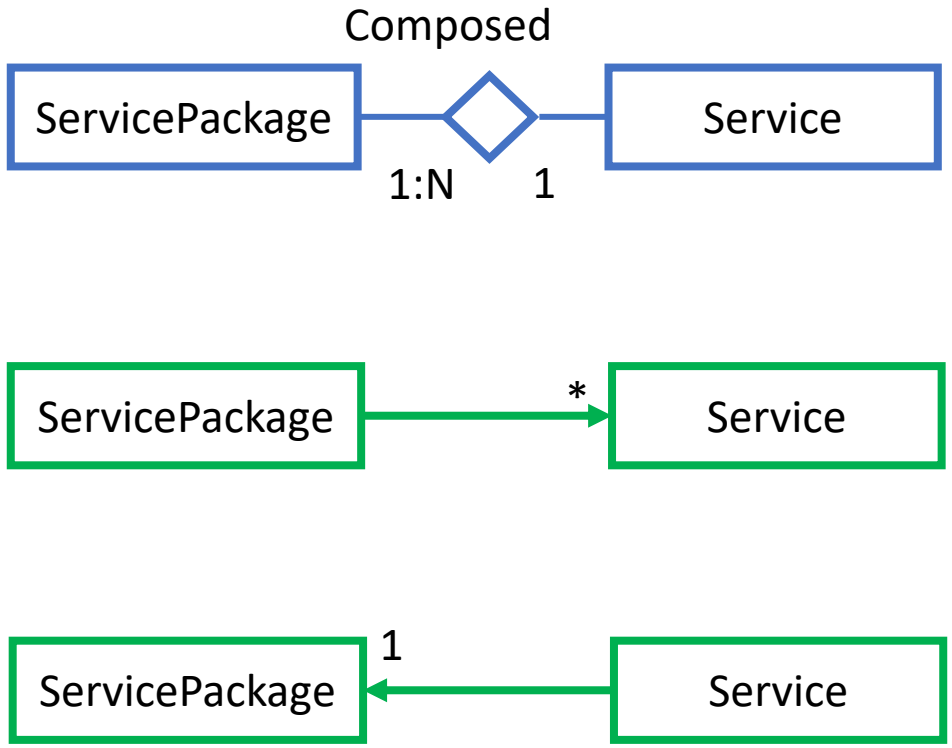
```
CREATE DEFINER=`root`@`localhost` TRIGGER `user_AFTER_UPDATE` AFTER UPDATE ON `user` FOR EACH ROW BEGIN
    IF new.isInsolvent = '1' AND old.isInsolvent = '0' THEN
        INSERT INTO `new_schema`.`insolvent-users`(idinsolventuser)
        VALUES (new.username);
    END IF;

    IF new.isInsolvent = '0' AND old.isInsolvent = '1'
    AND new.username IN (SELECT idinsolventuser FROM `new_schema`.`insolvent-users`)
    THEN
        DELETE FROM `new_schema`.`insolvent-users`
        WHERE idinsolventuser = new.username;
    END IF;
END
```

Motivation

The trigger on the user table are used to insert o remove an user from the Insolvent Users table if he is marker or removed as insolvent.

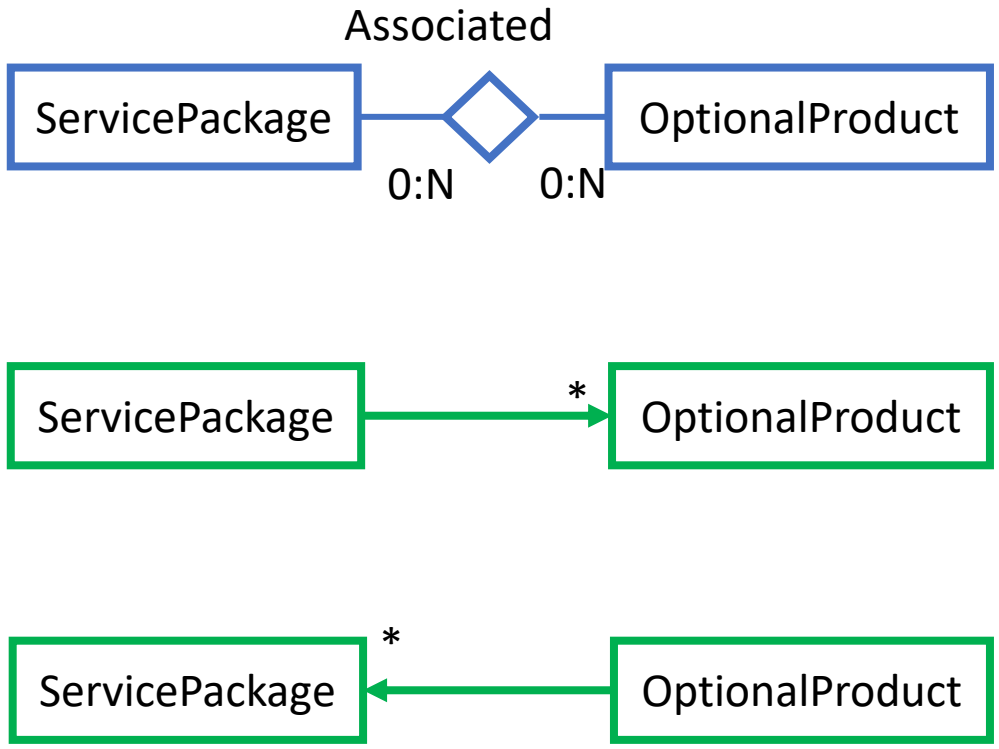
ORM design

Relationship “Composed”



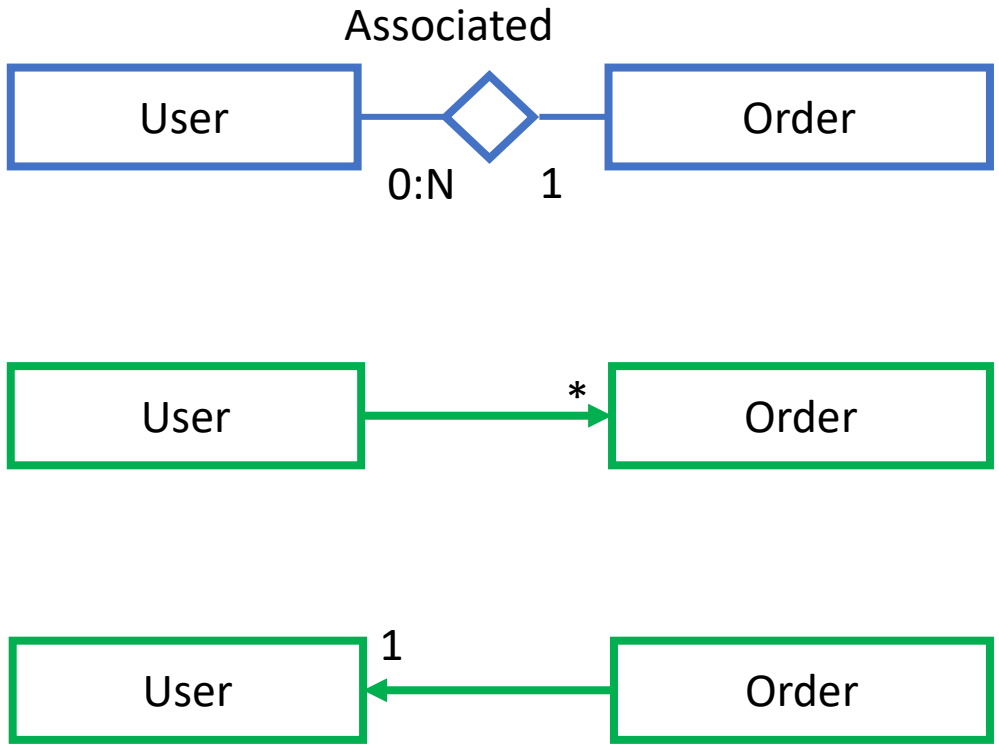
- ServicePackage -> Service @OneToMany
- Service -> ServicePackage @ManyToOne
- Only the ServicePackage -> Service side used.
- FetchType Eager because i always want to retrieve the services associated to a package.
- Cascade Type Remove because the services are bound to the specific package.

Relationship “Associated”



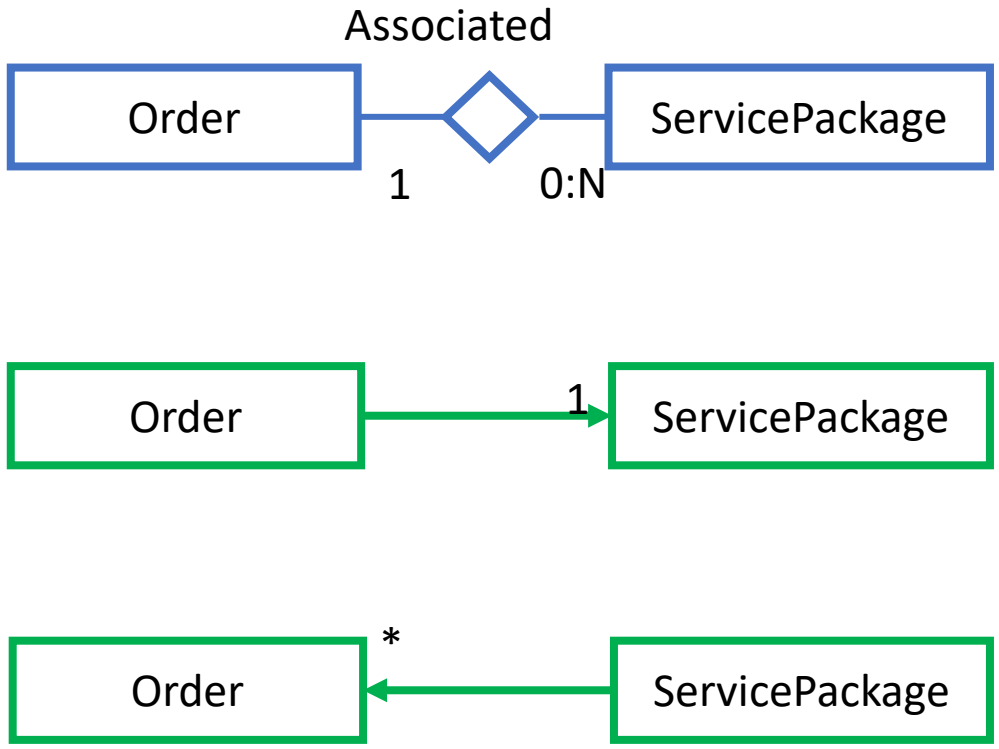
- ServicePackage – Optional Product @ManyToMany
- ServicePackage -> Optional Product side used.
- FetchType Eager because i always want to retrieve the services associated to a package.
- No operation cascaded because the ServicePackage and OptionalProducts are independent one from the other.

Relationship “OwnedOrders”



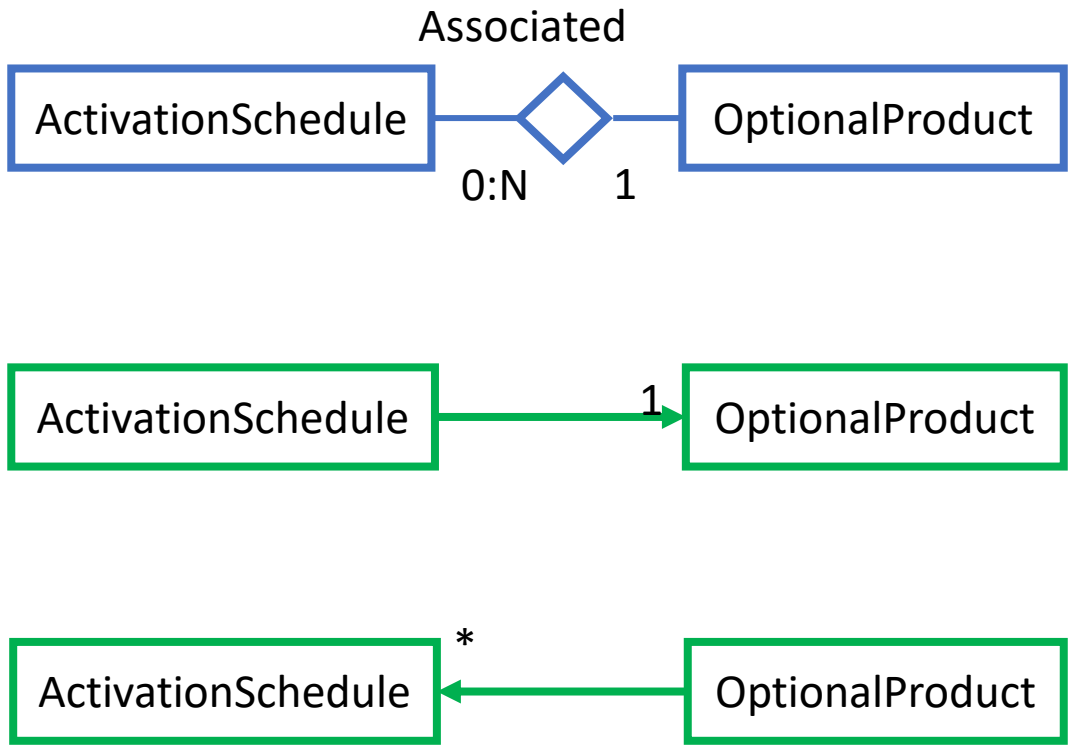
- User -> Order @OneToMany
- Order -> User @ManyToOne
- Order->User side used through a named query.
- Remove cascaded because it’s pointless to keep the user’s order if he is deleted.

Relationship “BoughtPackage”



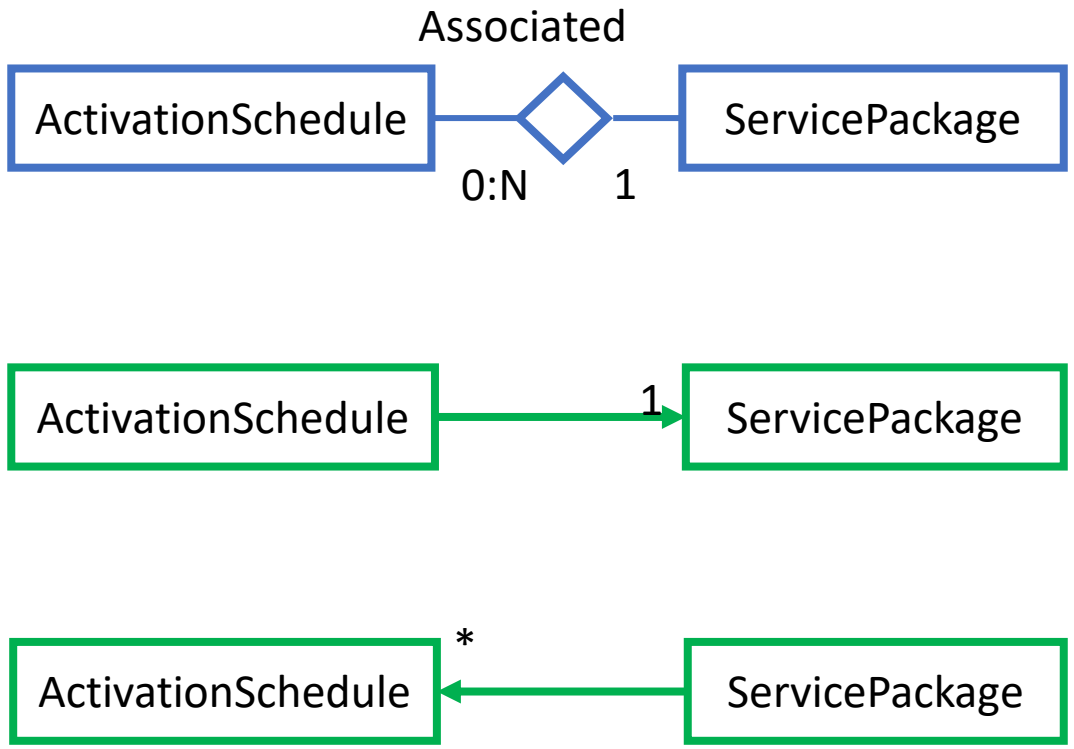
- Order -> ServicePackage @ManyToOne
- ServicePackage -> Order @OneToMany
- Order->User side used through a named query.
- Remove cascaded because it’s pointless to keep the user’s order if he is deleted.

Relationship “ActivationScheduleProduct”



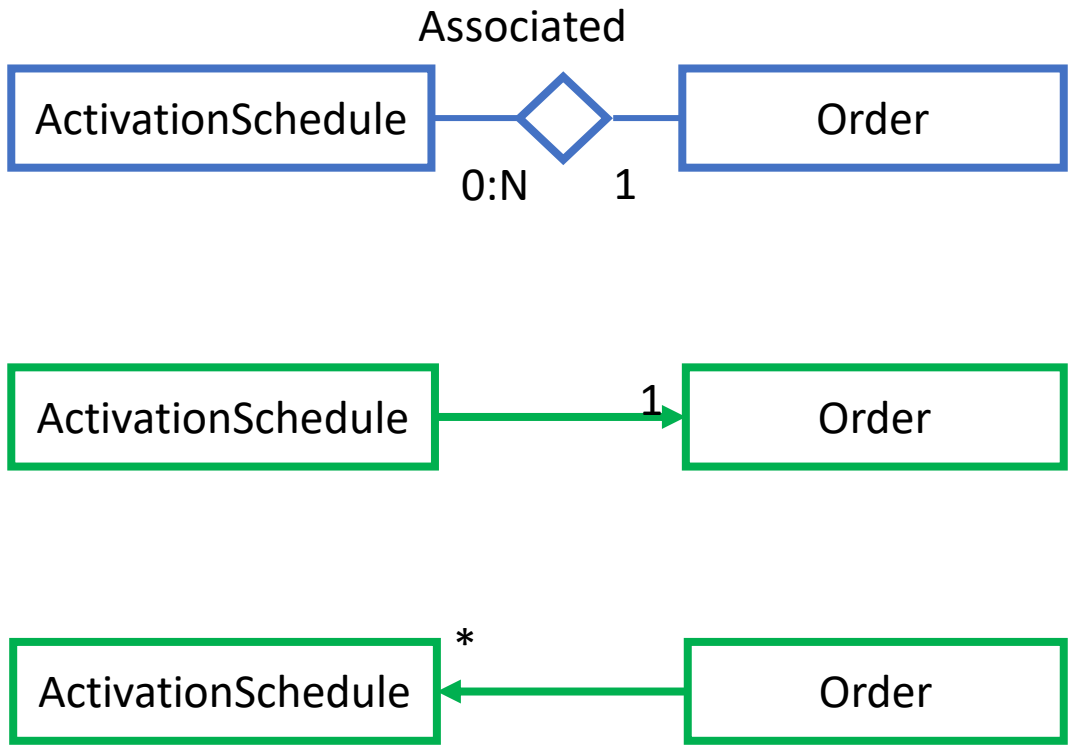
- ActivationSchedule -> OptionalProduct @ManyToOne
- OptionalProduct -> ActivationSchedule @OneToMany
- Only the ManyToOne is needed to print the data of the schedule in the employee view

Relationship “ActivationSchedulePackage”



- ActivationSchedule -> ServicePackage @ManyToOne
- OptionalProduct -> ServicePackage @OneToMany
- Only the ManyToOne is needed to print the data of the schedule in the employee view

Relationship “ActivationScheduleOrder”



- ActivationSchedule -> Order @ManyToOne
- OptionalProduct -> Order @OneToMany
- Only the ManyToOne is needed to print the data of the schedule in the employee view

ActivationSchedule

```java
@Entity

@Table(name = "activation-schedule", schema = "new_schema")

@NamedQuery(name = "activationSchedule.findByOrderID", query = "SELECT o FROM ActivationSchedule o WHERE o.orderid.id = ?1")

@NamedQuery(name = "activationSchedule.findOptByOrderID", query = "SELECT o.optionalProduct FROM ActivationSchedule o  " +
        "WHERE o.orderid.id = ?1")

@NamedQuery(name = "activationSchedule.findAll", query = "SELECT a FROM ActivationSchedule a")
public class ActivationSchedule {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long bridge_id;

    @ManyToOne
    @JoinColumn(name = "package")
    private ServicePackage servicePackage;


    @ManyToOne
    @JoinColumn(name = "orderid")
    private Order orderid;


    @ManyToOne
    @JoinColumn(name = "optproduct")
    private OptionalProduct optionalProduct;

    private Date actdate;

    private Date deactdate;

    private String status;
```

Alert

```
@Entity

@Table(name = "alert")

@NamedQuery(name = "Alerts.getAllAlerts", query = "SELECT a FROM Alert a")

@NamedQuery(name = "Alerts.getAllAlertsByUser", query = "SELECT a FROM Alert a
        WHERE a.user.username = ?1")
public class Alert {
    @Id
    @OneToOne
    @JoinColumn(name = "username")
    private User username;

    private String email;

    private float amount;

    private Date datetimelastrejection;
```

AlertHistory

```java
@Entity

@Table(name = "alert-history")
@NamedQuery(name = "AlertsHistory.getAllAlerts", query = "SELECT a FROM AlertHistory a")
public class AlertHistory {

    @Id
    private long id;
    @ManyToOne
    @JoinColumn(name = "username")
    private User user;

    private float amount;

    private Date datetimerejection;
```

AvgOptForPackage

```java
@Entity
@Table(name = "avg-opt-for-package")
@NamedQuery(name = "AvgOptForPackage.findByPackageID", query = "SELECT p FROM AvgOptForPackage p WHERE p.servicePackage.ID = ?1")
@NamedQuery(name = "AvgOptForPackage.getAllAvgOptForPackages", query = "SELECT p FROM AvgOptForPackage p")
public class AvgOptForPackage {
    @Id
    private long id;

    @OneToOne
    @JoinColumn(name = "servicePackage")
    private ServicePackage servicePackage;

    private int numOptTot;

    private int numsales;

    private float avgOptForSale;
```

BestOpProduct

```
@Entity
@Table(name = "best-opt-product")
@NamedQuery(name = "BestOptProduct.findAllBestOptProduct", query = "SELECT b
        FROM BestOptProduct b")
public class BestOptProduct {
    @Id
    private long id;

    @OneToOne
    @JoinColumn(name = "productid")
    private OptionalProduct optionalProduct;
```

InsolventUsers

```
@Entity
@Table(name = "insolvent-users", schema = "new_schema")
@NamedQuery(name = "InsolventUsers.findAllInsolventUsers", query = "SELECT i FROM
       InsolventUsers i")
public class InsolventUsers {

    @Id
    @OneToOne
    @JoinColumn(name = "idinsolventuser")
    private User user;
```

NumPurchPackage

```java
@Entity
@Table(name = "num-purch-package")
@NamedQuery(name = "NumPurchPackage.findByPackageID", query = "SELECT p FROM NumPurchPackage p WHERE
p.servicePackage.ID = ?1")
@NamedQuery(name = "NumPurchPackage.getAllNumPurchPackages", query = "SELECT p FROM NumPurchPackage p")
public class NumPurchPackage {
    @Id
    private long id;
    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "packageid")
    private ServicePackage servicePackage;

    private int numpurchases;
```

NumPurchPackageValPeriod

```
@Entity
@Table(name = "num-purch-package-val-period")
@NamedQuery(name = "NumPurchPackageValPeriod.findByPackageIdValPeriod", query = "SELECT
p FROM NumPurchPackageValPeriod p WHERE " +
        "p.servicePackage.ID = ?1 AND p.valperiod = ?2")
@NamedQuery(name = "NumPurchPackage.getAllNumPurchPackageValPeriod", query = "SELECT p
FROM NumPurchPackageValPeriod p")
public class NumPurchPackageValPeriod {

    @Id
    private long id;
    @ManyToOne
    @JoinColumn(name = "packageid")
    private ServicePackage servicePackage;

    private int valperiod;

    private int numpurchases;
```

OptionalProduct

```
@Entity
@Table(name = "optional-product")
@NamedQuery(name = "OptionalProduct.findAllProducts", query = "SELECT o FROM OptionalProduct o")
@NamedQuery(name = "OptionalProduct.findOptProductByID", query = "SELECT o FROM OptionalProduct o
        WHERE o.id = ?1")
public class OptionalProduct {

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    private float monthlyFee;
```

Order

```
@Entity
@Table(name = "order", schema = "new_schema")
@NamedQuery(name = "Order.checkOrder", query = "SELECT o FROM Order o WHERE o.id = ?1")
@NamedQuery(name = "Order.findByStatusAndUsername", query="SELECT o FROM Order o WHERE o.status = ?1 AND
o.user.username = ?2")
@NamedQuery(name = "Order.findOrderByID", query = "SELECT o FROM Order o WHERE o.id = ?1")
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Temporal(TemporalType.DATE)
    private Date creationdate;

    private int valperiod;

    private int totalvalue;

    @Temporal(TemporalType.DATE)
    private Date startdate;

    private String status;

    private float fee;

    @ManyToOne
    @JoinColumn(name = "username")
    private User user;

    @ManyToOne
    @JoinColumn(name = "packageid")
    private ServicePackage servicePackage;
```

SalesOfPackage

```java
@Entity
@Table(name = "sales-package", schema = "new_schema")
@NamedQuery(name = "SalesOfPackage.findByPackageID", query = "SELECT s FROM SalesOfPackage s WHERE s.servicePackage.ID = ?1")
@NamedQuery(name = "SalesOfPackage.findAllSalesOfPackages", query = "SELECT s FROM SalesOfPackage s")
public class SalesOfPackage {

    @Id
    private long id;

    @OneToOne
    @JoinColumn(name = "servicePackage")
    private ServicePackage servicePackage;

    private float totalwithopt;

    private float totalwithoutopt;
```

SalesOptionalProduct

```
@Entity
@Table(name = "sales-optional-product", schema = "new_schema")
@NamedQuery(name = "SalesOptionalProduct.getAllSalesOptionalProducts", query = "SELECT s FROM
SalesOptionalProduct s")
@NamedQuery(name = "SalesOptionalProduct.findByProductId", query = "SELECT s FROM
SalesOptionalProduct s WHERE s.optionalProduct.id=?1")
public class SalesOptionalProduct {

    @Id
    private long id;

    @OneToOne
    @JoinColumn(name = "optproductid")
    private OptionalProduct optionalProduct;

    private float totalsalesvalue;
```

Service

```
@Entity
@Table(name = "service", schema="new_schema")
@NamedQuery(name = "service.findUnassigned", query = "SELECT s FROM Service s WHERE s.servicePackage IS NULL")
public class Service {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long serviceid;

    private String name;

    private String type;

    private int minutes;

    private int sms;

    private Float extraminfee;

    private Float extrasmsfee;

    private int giga;

    private Float extragigafee;

    @ManyToOne
    @JoinColumn(name = "service_package_id")
    private ServicePackage servicePackage;
```

SuspendedOrders

```java
@Entity
@Table(name = "suspended-orders", schema = "new_schema")
@NamedQuery(name = "SuspendedOrders.findAllSuspendedOrders", query = "SELECT s FROM SuspendedOrders s")
public class SuspendedOrders {

    @Id
    private long id;

    @OneToOne
    @JoinColumn(name = "idsuspendedorders")
    private Order order;
```

User

```java
@Entity
@Table(name = "user", schema = "new_schema")
@NamedQuery(name = "User.checkCredentials", query = "SELECT u FROM User u WHERE u.username = ?1 AND u.password = ?2
          AND u.type = ?3")
@NamedQuery(name = "User.checkExisting", query = "SELECT u FROM User u WHERE u.username = ?1 OR u.email = ?2")
public class User {

    @Id
    private String username;

    private String email;

    private String password;

    private String type;

    private boolean isInsolvent;

    private int numRejections;

    @OneToMany(mappedBy = "user", cascade = CascadeType.REMOVE)
    private Collection<Order> orders;
```

Textual Functional Analysis of the interactions - Consumer

The Consumer application has a landing page with two forms for registering and logging in. After filling the register form, the user is redirected again to the landing page with a message with the status of the action. After filling the login form, the user is redirected to the home page if he inserted valid credentials, or to the landing page again in the other case.

The home page can also be accessed without logging in by clicking a link in the landing page. If the page is accessed after a log in, it will display the list of available service packages, the list of the owned ones with the associated services and the optional products and the list of suspended orders, that are the orders that the user hasn't paid for yet. If the page is accessed without logging in, only the list of the service packages is shown. In both cases, after the list of available packages, there is a link to buy a package.

After clicking it, the purchase page shows a form where the user is able to select a package. After selecting a package, and submitting, the same page displays a form where the user is able to select a starting date, a validity period and some optional products to associate to the package. After choosing all the needed data and submtting, the system sends the user to a Confirmation page with the recap of the order.

If the user is not logged in, there is a link that redirects to the login page, otherwise there will be two buttons to purchase the order. One button is used to simulate the success of the payiment, while the other one is used to simulate the failure of the payiment. In both cases, after the purchase the user will be redirected to the Home Page.

In every page, if the user is logged in, there is a link on the right to log out from the system.

Legend: Pages (views), view components, events, actions

Textual Functional Analysis of the interactions - Employee

The Employee application has a Landing Page with a form to log into the system. If the user fills it with invalid data, the same page is shown again with an error message, while if the user inputs valid username and password, the user will be redirected to the Home Page.

The Home Page displays a link to access the Sales Data Page, the list of all the services package created, the list of the services not assigned to a package, a form to create a service package, a form to create an optional product and a form to create a service, one for each type of service available in the system. If the user fills them, the home page will be displayed again, with the updated informations if the operation went well or with an error message if something wrong happened.

If the user clicks to the sales link, the Sales Data Page is shown. The page displays the list of the insolvent users, the list of all alerts created, the list of suspended orders, the total purchases for each package, the total purchases for each package and validity period, the total value of sales per package with and without the products, the average number of optional products sold with each package and the best selling optional product.

In every page, if the user is logged in, there is a link on the right to log out from the system.

Legend: Pages (views), view components, events, actions

Components

- Client components
  - Servlets
    - BuyPackage
    - CheckLogin
    - Confirmation
    - CreateOptionalProduct
    - CreateService
    - CreateServicePackage
    - HomeCustomer
    - HomeEmployee
    - IndexCustomer
    - IndexEmployee
    - Logout
    - PurchController
    - Register
    - SalesReportServlet
  - Views
    - BuyPackage.html
    - Confirmation.html
    - HomeCustomer.html
    - HomeEmployee.html
    - indexCustomer.html
    - indexEmployee.html
    - SalesReport.html

- Back end components
  - Entities
    - ActivationSchedule
    - Alert
    - AlertHistory
    - AvgOptForPackage
    - BestOpProduct
    - InsolventUsers
    - NumPurchPackage
    - NumPurchPackageValPeriod
    - OptionalProduct
    - Order
    - SalesOfPackage
    - SalesOptionalProduct
    - Service
    - ServicePackage
    - SuspendedOrders
    - User

Components

- Back end components
  - Business components
    - AuthService
      - registerUser(username,password,email,type)
      - authenticateUser(username,password,type)
    - CustomerService
      - getServicePackages()
      - getSingleServicePackage(id)
      - getOrdersByStatusAnNickname(username,status)
      - getSingleOptionalProduct(id)
      - addOrder(creationDate, valPeriod, startDate, fee, packageID, username, optProductIds, status)
      - calculateTotalValue(valPeriod, fee, optProductIds)
      - createOrderSchedule(order, optProdIds, startDate, valperiod, servicePackage, status)
      - validateOrder(orderId)
      - validateOrderSchedule(orderId)
      - checkInsolventRemove(username)
      - failAgainOrder(orderId)
      - checkAlert(user, amount)
      - getOptionalProducts()
      - getOrder(id)
      - findBoughtOptional(id)
      - findSchedulesByOrderIds(orders)
      - findUnassignedServices()
    - EmployeeService
      - createServicePackage(name, cost12, cost24, cost36, servicesIds, productsIds)
      - createOptionalProduct(name, monthlyfee)
      - createService(name, type, minutes, sms, extraMinFee, extraSMSFee, giga, extragigafee)
      - findAllSchedules()
    - SalesReportService
      - getInsolventUsers()
      - getSuspendedOrders()
      - getAlertsHistory()
      - getAlerts()
      - getBestSeller()
      - getBestSellerData()
      - getSalesOfPackage()
      - getSalesOptionalProduct()
      - getAllNumPurchPackages()
      - getAllNumPurchPackagesValPeriod()
      - getAllAvgOptForPackage()

Component Design Comments

We decided to split the back end functionalities in 4 main components:

- AuthService: Responsible for the registration and login functionalities.
- CustomerService: Responsible for functionalities mainly concerning the Customer application.
- EmployeeService: Responsible for functionalities mainly concerning the Employee application.
- SalesReportService: Responsible for the retrieval of informations concerning the sales report.

This is done mainly for «logical» separation, because in this way each client servlet should be able to use only one service and get every data needed. Of course, this is not guaranteed for every single servlet, but for most of them it is.