Data bases 2 Project

Riccardo Luigi Aielli (996491)

Riccardo Casciotti (974768)

Index

- Specification
- Entity Relation diagram
  - Service pack in details
- Relational Model
- Triggers
- ORM relationship design
- Entities code
- Components
  - Business Tier
  - Web Tier
- UML sequence diagrams

Specifications: consumer application 1

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username (which can be assumed as the unique identification parameter), a password and an email. Login leads to the Home page of the consumer application. Registration leads back to the landing page where the user can log in.
The user can log in before browsing the application or browse it without logging in. If the user has logged in, his/her username appears in the top right corner of all the application pages.
The Home page of the consumer application displays the service packages offered by the telco company.
A service package has an ID and a name (e.g., “Basic”, “Family”, “Business”, “All Inclusive”, etc). It comprises one or more services. Services are of four types: fixed phone, mobile phone, fixed internet, and mobile internet. The mobile phone service specifies the number of minutes and SMSs included in the package plus the fee for extra minutes and the fee for extra SMSs. The fixed phone service has no specific configuration parameters. The mobile and fixed internet services specify the number of Gigabytes included in the package and the fee for extra Gigabytes. A service package must be associated with one validity period. A validity period specifies the number of months (12, 24, or 36). Each validity period has a different monthly fee (e.g., 20€/month for 12 months, 18€/month for 24 months, and 15€ /month for 36 months). A package may be associated with one or more optional products (e.g., an SMS news feed, an internet TV channel, etc.). The validity period of an optional product is the same as the validity period that the user has chosen for the service package. An optional product has a name and a monthly fee independent of the validity period duration. The same optional product can be offered in different service packages.
From the Home page, the user can access a Buy Service page for purchasing a service package and thus creating a service subscription. The Buy Service page contains a form for purchasing a service package. The form allows the user to select one package from the list of available ones and choose the validity period duration and the optional products to buy together with the chosen service. The form also allows the user to select the start date of his/her subscription. After choosing the service packages, the validity period and (0 or more) optional products, the user can press a CONFIRM button. The application displays a CONFIRMATION page that summarizes the details of the chosen service package, the validity period, the optional products and the total price to be pre-paid: (monthly fee of service package * number of months) + (sum of monthly fees of options * number of months).
If the user has already logged in, the CONFIRMATION page displays a BUY button. If the user has not logged in, the CONFIRMATION page displays a link to the login page and a link to the REGISTRATION page. After either logging in or registering and immediately logging in, the CONFIRMATION page is redisplayed with all the confirmed details and the BUY button.

Specifications: consumer application 2

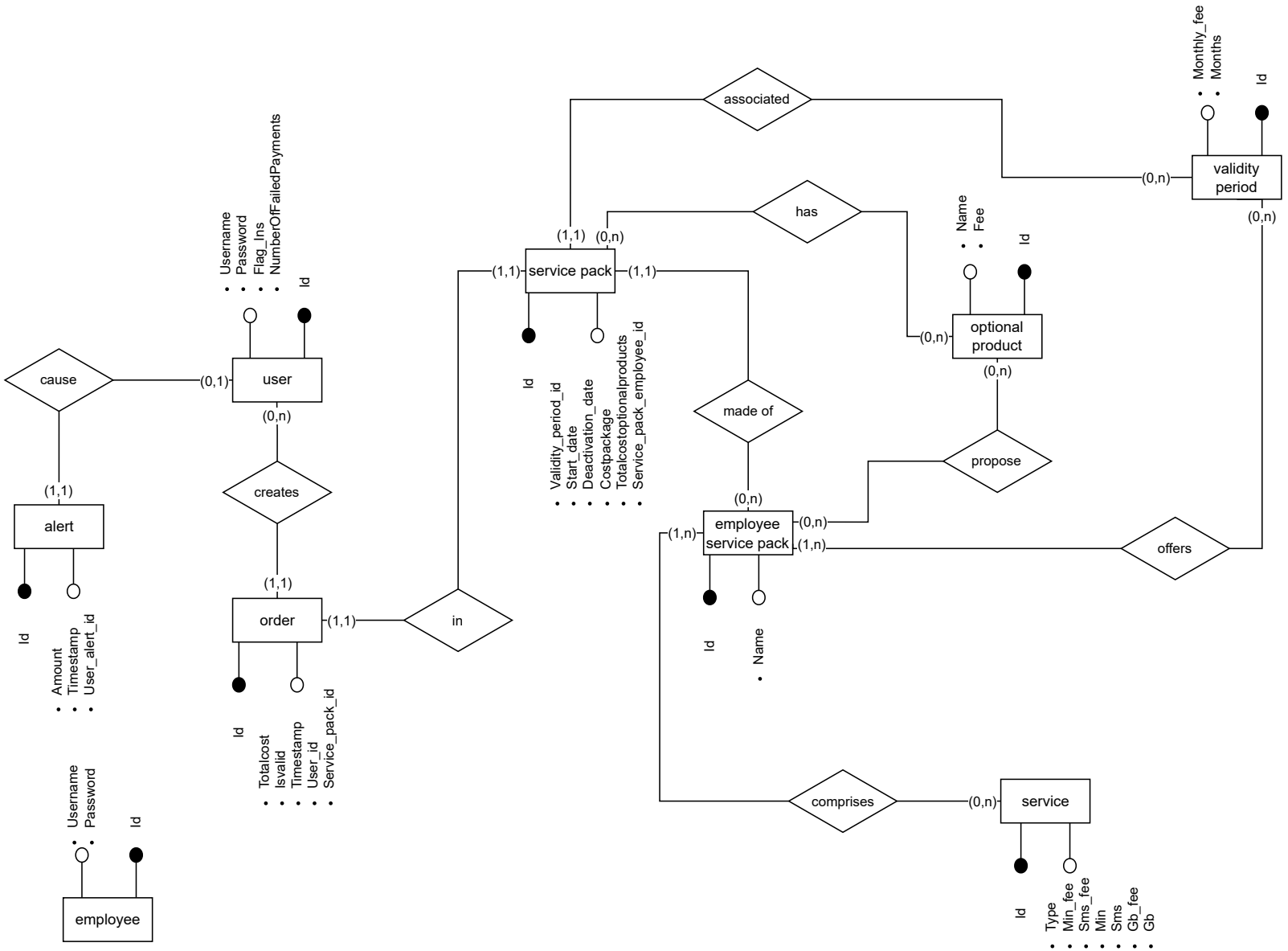
When the user presses the BUY button, an order is created. The order has an ID and a date and hour
of creation. It is associated with the user and with the service package, its validity period and the chosen optional products. It also contains the total value (as in the CONFIRMATION page) and the
start date of the subscription. After creating the order, the application bills the customer by calling an
external service. If the external service accepts the billing, the order is marked as valid and a service
activation schedule is created for the user. A service activation schedule is a record of the services and
optional products to activate for the user with their date of activation and date of deactivation.
If the external service rejects the billing, the order is put in the rejected status and the user is flagged
as insolvent. When an insolvent user logs in, the home page also contains the list of rejected orders.
The user can select one of such orders, access the CONFIRMATION page, press the BUY button and
attempt the payment again. When the same user causes three failed payments, an alert is created in a
dedicated auditing table, with the user Id, username, email, and the amount, date and time of the last
rejection.

Specifications: employee application

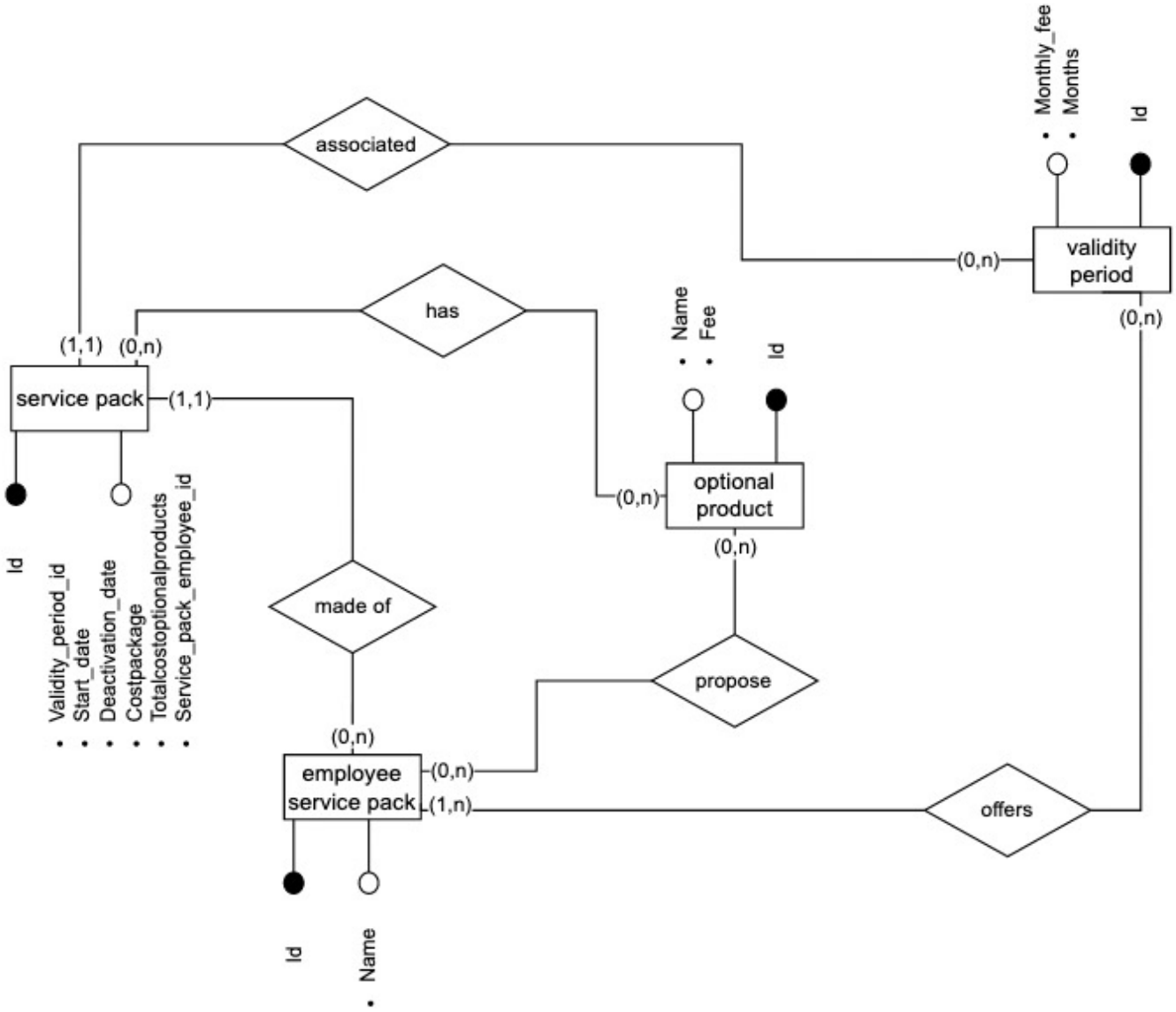
The employee application allows the authorized employees of the telco company to log in. In the Home page, a form allows the creation of service packages, with all the needed data and the possible optional products associated with them. The same page lets the employee create optional products as well.
A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- Number of total purchases per package.
- Number of total purchases per package and validity period.
- Total value of sales per package with and without the optional products.
- Average number of optional products sold together with each service package.
- List of insolvent users, suspended orders and alerts.
- Best seller optional product, i.e. the optional product with the greatest value of sales across all the sold service packages.

Entity Relationship



ER Service pack in detail



Motivations of the ER design

According to the specifications we have to create an entity that links validity periods and optional products to each service pack. We have thus created the entity “employee service pack” which represents the service packs created by an employee to which one or more validity periods and zero or more optional products are associated. These represent the options among which the user can choose when creating and purchasing his order.

Relational model

```
CREATE TABLE `user` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Username` varchar(45) NOT NULL,
  `Password` varchar(45) NOT NULL,
  `Email` varchar(90) NOT NULL,
  `Flag_Ins` boolean default 0 NOT NULL,
  `NumberOfFailedPayments` int default 0 NOT NULL,
  PRIMARY KEY (`id`),
  constraint Email
      unique (Email),
   constraint Id
      unique (Id),
   constraint Username
      unique (Username)
);
```

```
CREATE TABLE `employee` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Username` varchar(45) NOT NULL,
  `Password` varchar(45) NOT NULL,
  PRIMARY KEY (`Id`),
  constraint Username
      unique (Username)
);
```

```
CREATE TABLE `validity_period` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Monthly_fee` int NOT NULL,
  `Months` int NOT NULL,
  PRIMARY KEY (`Id`)
);
```

Relational model

```
CREATE TABLE `alert` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Amount` float NOT NULL,
  `Timestamp` datetime NOT NULL,
  `User_alert_id` int NOT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY (`User_alert_id`),
  CONSTRAINT `user_alert` FOREIGN KEY (`User_alert_id`)
                REFERENCES `user` (`Id`) ON DELETE RESTRICT
                ON UPDATE RESTRICT
);
```

```
CREATE TABLE `optional_product` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(45) NOT NULL,
  `Fee` float NOT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY (`Name`)
);
```

```
CREATE TABLE `service` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Type` varchar(45) NOT NULL,
  `Min_fee` float,
  `Sms_fee` float,
  `Min` int,
  `Sms` int,
  `Gb_fee` float,
  `Gb` int,
  PRIMARY KEY (`Id`)
);
```

Relational model

```
CREATE TABLE `service_pack` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Validity_period_id` int NOT NULL,
  `Start_date` date NOT NULL,
  `Deactivation_date` date NOT NULL,
  `Costpackage` float NOT NULL,
  `Totalcostoptionalproducts` float DEFAULT 0 NOT NULL,
  `Service_pack_employee_id` int NOT NULL,
  PRIMARY KEY (`Id`),
  CONSTRAINT `validity_period_fk` FOREIGN KEY (`Validity_period_id`) REFERENCES `validity_period` (`Id`),
  CONSTRAINT `service_pack_employee_fk` FOREIGN KEY (`Service_pack_employee_id`)
                REFERENCES `employeeServicePack` (`Id`)
);

CREATE TABLE `employeeServicePack` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(45) NOT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY (`Name`)
);
```

Relational model

```
CREATE TABLE `order` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Totalcost` float NOT NULL,
  `Isvalid` tinyint NOT NULL,
  `Timestamp` datetime NOT NULL,
  `User_id` int NOT NULL,
  `Service_pack_id` int NOT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY (`Service_pack_id`),
  CONSTRAINT `service_pack_id` FOREIGN KEY (`Service_pack_id`) REFERENCES `service_pack` (`Id`),
  CONSTRAINT `user_id` FOREIGN KEY (`User_id`) REFERENCES `user` (`Id`)
);
```

Number of total purchases per Employee Service Package

```
create table numberTotalPurchasesPerESP
(
    EmployeeServicePack_id int not null primary key,
    Numbertotalpurchases int default 0 not null,
    constraint numberOfTotalPurchasesPerPackage foreign key (EmployeeServicePack_id) references employeeServicePack (Id)
);
```

```
CREATE DEFINER  = CURRENT_USER TRIGGER purchaseToNumberTotalPurchasesPerESP_add
    AFTER INSERT ON `order` FOR EACH ROW
    BEGIN
        IF NEW.Isvalid = 1 THEN
            UPDATE numberTotalPurchasesPerESP SET Numbertotalpurchases = Numbertotalpurchases + 1
            WHERE EmployeeServicePack_id IN (   SELECT s.Service_pack_employee_id
                                                FROM service_pack s
                                                WHERE s.Id = NEW.Service_pack_id);
    END IF;
    end //
delimiter ;

DROP TRIGGER IF EXISTS numberTotalPurchasesPerESP_create;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER numberTotalPurchasesPerESP_create
    AFTER INSERT ON employeeServicePack FOR EACH ROW
    BEGIN
        INSERT INTO db2Project.numberTotalPurchasesPerESP(EmployeeServicePack_id)
        VALUES(NEW.Id);
    end //
delimiter ;


DROP TRIGGER IF EXISTS purchaseToNumberTotalPurchasesPerESP_update;
delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER purchaseToNumberTotalPurchasesPerESP_update
    AFTER UPDATE ON `order` FOR EACH ROW
    BEGIN
        IF NEW.Isvalid = 1 THEN
            UPDATE numberTotalPurchasesPerESP SET Numbertotalpurchases = Numbertotalpurchases + 1
            WHERE EmployeeServicePack_id IN (SELECT s.Service_pack_employee_id
                                             FROM service_pack s
                                             WHERE s.Id = NEW.Service_pack_id);
        END IF;
    end //
delimiter ;
```

After a user buys an order on a service package an INSERT operation on the "order" table is performed. This increases the number of times the package is bought and so the "numberTotalPurchasesESP" table is updated. The action on the table is performed only if the order is correctly payed (or an invalid order becomes payed).

Number of total purchases per package and validity period

```
create table numberTotalPurchasesPerESPAndValidityPeriod
(
    Id int not null AUTO_INCREMENT primary key,
    EmployeeServicePack_id     int not null,
    Validity_period_id   int not null,
    TotalPurchases int not null DEFAULT 0,
    constraint numberTotalPurchasesPerESPAndValidityPeriod_fk0
        foreign key (EmployeeServicePack_id) references employeeServicePack (Id),
    constraint numberTotalPurchasesPerESPAndValidityPeriod_fk1
        foreign key (Validity_period_id) references validity_period (Id)
);


DROP TRIGGER IF EXISTS numberTotalPurchasesPerESPAndVP_create;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER numberTotalPurchasesPerESPAndVP_create
    AFTER INSERT ON offers FOR EACH ROW
    BEGIN
        INSERT INTO numberTotalPurchasesPerESPAndValidityPeriod(EmployeeServicePack_id, Validity_period_id)
        VALUES(NEW.EmployeeServicePack_id, NEW.Validity_period_id);

    end //
delimiter ;



DROP TRIGGER IF EXISTS purchaseToNumberTotalPurchasesPerESPAndVP_new;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER purchaseToNumberTotalPurchasesPerESPAndVP_new
    AFTER INSERT ON `order` FOR EACH ROW
    BEGIN
        IF NEW.Isvalid = 1 THEN
            UPDATE numberTotalPurchasesPerESPAndValidityPeriod
            SET TotalPurchases = TotalPurchases + 1
            WHERE (EmployeeServicePack_id, Validity_period_id) IN (SELECT s.Service_pack_employee_id, s.Validity_period_id
                                                                   FROM db2Project.service_pack s
                                                                   WHERE s.Id = NEW.Service_pack_id);

        END IF;
    end //
delimiter ;
```

Every time a user creates an order in the system the "order" table is updated with an INSERT operation. Then the trigger starts the execution, if the order is marked as valid then the "numberTotalPurchasesESPandValidityPeriod" table is updated, which organizes the number of Employee Service Packages sold, by Validity periods.

Total value of sales per package with and without the optional products

```
create table salesPerPackage
(
    EmployeeServicePack_id int not null
        primary key,
    totalSalesWithOptionalProduct int not null DEFAULT 0,
    totalSalesWithoutOptionalProduct int not null DEFAULT 0,
    constraint salesPerPackage_fk0
        foreign key (EmployeeServicePack_id) references employeeServicePack (Id)
);


DROP TRIGGER IF EXISTS ESPAddEntryInSalesPerPackage_create;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER ESPAddEntryInSalesPerPackage_create
    AFTER INSERT ON employeeServicePack FOR EACH ROW
    BEGIN

        INSERT INTO salesPerPackage(EmployeeServicePack_id)
        VALUES(NEW.Id);

    end //
delimiter ;



DROP TRIGGER IF EXISTS salesPerPackage_add;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER salesPerPackage_add
    AFTER INSERT ON `order` FOR EACH ROW
    BEGIN
        DECLARE cp,tcop float;
        IF NEW.Isvalid = 1 THEN
            SET cp :=   (SELECT sp.Costpackage
                        FROM service_pack sp
                        WHERE sp.Id = NEW.Service_pack_id);

            SET tcop :=     (SELECT sp.Totalcostoptionalproducts
                            FROM service_pack sp
                            WHERE sp.Id = NEW.Service_pack_id);


            UPDATE salesPerPackage s
            SET s.totalSalesWithOptionalProduct = s.totalSalesWithOptionalProduct + cp + tcop,
                s.totalSalesWithoutOptionalProduct = s.totalSalesWithoutOptionalProduct + cp
            WHERE s.EmployeeServicePack_id IN (SELECT s.Service_pack_employee_id
                                               FROM service_pack s
                                               WHERE s.Id = NEW.Service_pack_id );
        END IF;
    end //
delimiter ;
```

Every time a user creates a successful (payment succeded) order and chooses to buy some Optional Products together with the chosen service package an INSERT on the table order is performed the trigger starts its execution and updates the table "salesPerPackage", which contains the total sales with and without optional products of every single Employee Service Package.

Average number of optional products sold together with each Employee Service Package

```
create table averageOPwithESP
(
 EmployeeServicePack_id int not null
        primary key,
    averageOPs float not null DEFAULT 0,
    totalOPsPerESP int not null DEFAULT 0,
    totalOrdersPerESP int not null DEFAULT 0,
    constraint averageOPs_fk0
        foreign key (EmployeeServicePack_id) references employeeServicePack (Id)
);

DROP TRIGGER IF EXISTS averageOPwithESP_new;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER averageOPwithESP_new
    AFTER INSERT ON employeeServicePack FOR EACH ROW
    BEGIN
        INSERT INTO averageOPwithESP(EmployeeServicePack_id)
        VALUES(NEW.Id);

end //
delimiter ;

DROP TRIGGER IF EXISTS averageOPwithESP_add;

delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER averageOPwithESP_add
    AFTER INSERT ON `order` FOR EACH ROW
    BEGIN
        DECLARE totalOPsPerESP_update integer;
        IF NEW.isValid = 1 THEN


            SET totalOPsPerESP_update := (  SELECT COUNT(h.Optional_product_id)
                                            FROM has h
                                            WHERE h.Service_pack_id = NEW.Service_pack_id);
            UPDATE averageOPwithESP
            SET totalOPsPerESP = totalOPsPerESP + totalOPsPerESP_update,
            totalOrdersPerESP = totalOrdersPerESP + 1,
            averageOPs = (totalOPsPerESP + totalOPsPerESP_update)/(totalOrdersPerESP + 1)
            WHERE EmployeeServicePack_id IN (SELECT s.Service_pack_employee_id
                                            FROM service_pack s
                                            WHERE s.Id = NEW.Service_pack_id);


        END IF;
    end //
delimiter ;
```

Every time a user creates a successful (payment succeded) order
and chooses to buy some Optional Products together with the chosen service package
an INSERT on the table "order" is performed and the trigger
starts its execution and updates the table "averageOPwithESP", which contains the divider to calculate the average and the total number of Optional Products associated with every Employee Service Pack.

Best Seller: the Optional Products with the highest number of sales

```
create table best_seller_OP
(
    Optional_product_id int   not null
        primary key,
    totalSales              float not null,
    constraint best_seller_OP_fk0
        foreign key (Optional_product_id) references optional_product (Id)
);


create table totalSalesPerOPO
(
    Optional_product_id int   not null
        primary key,
    totalSales     float default 0 not null
);



create table totalSalesPerOP
(
    Optional_product_id int             not null,
    totalSales     float default 0 not null
);

delimiter //
```

```
CREATE DEFINER  = CURRENT_USER trigger totalSales_add
    after insert
    on `order`
    for each row
begin


    IF NEW.Isvalid = 1 THEN
        DELETE FROM totalSalesPerOPO;
        INSERT INTO totalSalesPerOPO

            SELECT op.Id, (op.Fee * v.Months)
            FROM `order` o
                 JOIN service_pack s on s.Id = o.Service_pack_id
                 JOIN has h on h.Service_pack_id = s.Id
                 JOIN validity_period v on v.Id = s.Validity_period_id
                 JOIN optional_product op on op.Id = h.Optional_product_id
            WHERE s.Id = NEW.Service_pack_id;

        UPDATE totalSalesPerOP s, totalSalesPerOPO op
        SET s.totalSales = s.totalSales + op.totalSales
        WHERE s.Optional_product_id = op.Optional_product_id;


        DELETE FROM best_seller_OP;
        INSERT INTO best_seller_OP
            SELECT s1.Optional_product_id, s1.totalSales
            FROM totalSalesPerOP s1
            WHERE s1.Optional_product_id is not null and s1.totalSales IN (SELECT MAX(s2.totalSales) FROM totalSalesPerOP s2);
    end if;

end //
delimiter ;
```

```
delimiter //
CREATE DEFINER  = CURRENT_USER TRIGGER totalSales_new
    AFTER INSERT ON optional_product FOR EACH ROW BEGIN
    INSERT INTO totalSalesPerOP(Optional_product_id)
    VALUES(NEW.Id);
end //
delimiter ;
```

After a user creates a successful (payment succeded) order an INSERT is performed on "order" the trigger starts its execution by retrieving all the optional products associated with an order id and calculating the value purchased by the user. The results are put in "totalSalerOPO", which contains the total number of sales per optional product related to a particular order, It then takes all the values in the table and uses them to update the "totalSalesOP" table, which is used to extract the MAX value (best seller optional product).

Insolvent users, orders and alert

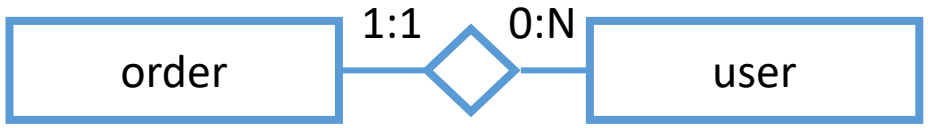
```
create table insolvent
(
    User_id int not null
        primary key,
    constraint insolvent_fk0
        foreign key (User_id) references `user` (Id)
);

DROP TRIGGER IF EXISTS insolventUser_update;

delimiter //
CREATE DEFINER  = CURRENT_USER trigger insolventUser_update
    after UPDATE on `user` FOR EACH ROW
    BEGIN
        IF NEW.Flag_Ins = 1 THEN
            IF(NEW.Id NOT IN (SELECT User_id FROM insolvent)) THEN
                INSERT INTO insolvent
                VALUES (NEW.Id);
        END IF;
        ELSE
            DELETE FROM insolvent i
            WHERE i.User_id = NEW.Id;
        END IF;
    end //
delimiter ;
```

```sql
create table rejectedOrders
(
    Order_id int not null
        primary key,
    constraint rejectedOrders_fk0
        foreign key (Order_id) references `order`  (Id)
);

DROP TRIGGER IF EXISTS rejectedOrder_new;

delimiter //
CREATE DEFINER  = CURRENT_USER trigger rejectedOrder_new
    AFTER INSERT on `order` FOR EACH ROW

BEGIN
    IF(NEW.Isvalid = 0) THEN
        IF(NEW.Id NOT IN (SELECT Order_id FROM rejectedOrders)) THEN
            INSERT INTO rejectedOrders(Order_id)
            VALUES(NEW.Id);
        END IF;
    END IF;
end //
delimiter ;



create table alerts
(
    Alert_id int not null
        primary key,
    constraint Alerts_fk0
        foreign key (Alert_id) references alert (Id)
);



DROP TRIGGER IF EXISTS alert_new;

delimiter //
CREATE DEFINER  = CURRENT_USER trigger alert_new
    AFTER INSERT on alert FOR EACH ROW

BEGIN
    INSERT INTO alerts
    VALUES (NEW.Id);
end //
delimiter ;
```

Every time a user creates an order and an INSERT on the table "order" is performed the triggers start the execution and if the order wasn’t already payed the user is inserted in the "insolvent" table, while the order is inserted in the "rejectedOrders" table and if the user has 3 or more failed payments an entry on the "alerts" table is created and the trigger starts the execution to insert the corresponding ID in the "alert" table.

ORM design

Relationship “creates”

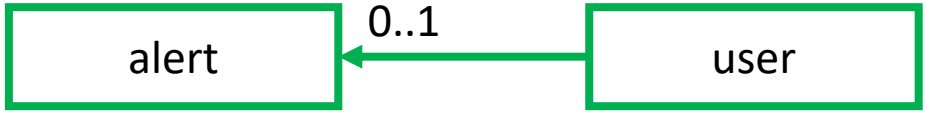


- order → user
- @ManyToOne
- Eager Fetch

- user → order
- @OneToMany
- Lazy Fetch
- Cascade Type: ***ALL***
- Orphan Removal

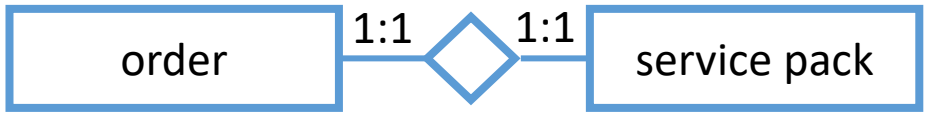
Relationship “cause”

- alert → user
- @OneToOne
- Eager Fetch



- user → alert
- @OneToOne
- Eager Fetch
- Cascade Type: ***ALL***
- Orphan Removal

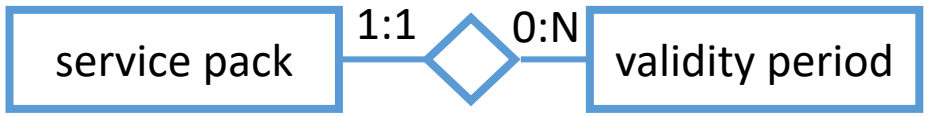
Relationship “in”



- order → service pack
- @OneToOne
- Lazy Fetch
- CascadeType: ***REMOVE, MERGE, REFRESH, DETACH***

- service pack → order
- @OneToOne
- Lazy Fetch
- CascadeType: ***ALL***
- Orphan Removal

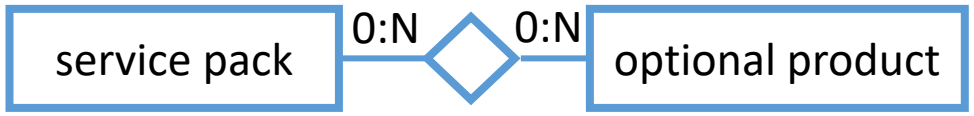
Relationship “associated”



- service pack → validity period
- @ManyToOne
- Eager Fetch

- validity period → service pack
- @OneToMany
- Lazy Fetch
- Cascade Type: ***ALL***
- Orphan Removal

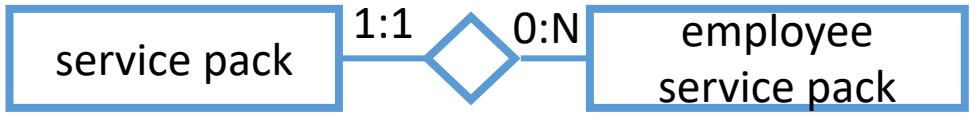
Relationship “has”



- service pack → optional product
- @ManyToMany
- Eager Fetch

- optional product → service pack
- @ManyToMany
- Eager Fetch

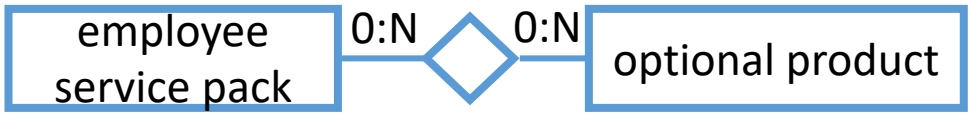
Relationship “made of”



- service pack → employee service pack
- @ManyToOne
- Eager Fetch

- employee service pack
- → service pack
- @OneToMany
- Eager Fetch
- Cascade Type: ***REMOVE, MERGE, REFRESH, DETACH***

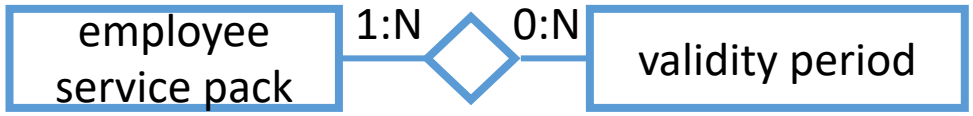
Relationship “propose”



- employee service pack
- → optional product
- @ManyToMany
- Eager Fetch

- optional product → employee service pack
- @ManyToMany
- Eager Fetch

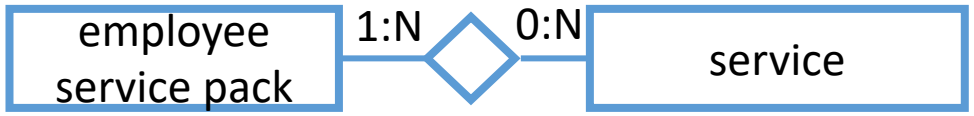
Relationship “offers”



- employee service pack
- → validity period
- @ManyToMany
- Eager Fetch

- validity period → employee service pack
- @ManyToMany
- Lazy Fetch

Relationship “comprises”



- employee service pack
- → service
- @ManyToMany
- Eager Fetch

- service → employee service pack
- @ManyToMany
- Lazy Fetch

AlertEntity

```
@Entity
@Table(name = "alert", schema = "db2Project")

@NamedQuery(name = "AlertEntity.findUser", query = "SELECT u FROM UserEntity u WHERE u.id = :user_alert")

public class AlertEntity {

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

   @Column(name = "Amount", nullable = false)
    private float amount;


    @Column(name = "Timestamp", nullable = false
    private Timestamp timestamp;

    @OneToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "User_alert_id", nullable = false)
    private UserEntity user_alert;
}
```

EmployeeEntity

```
@Entity
@Table(name = "employee", schema = "db2Project")
@NamedQuery(name = "EmployeeEntity.checkCredentials", query = "SELECT a FROM EmployeeEntity a WHERE a.username = :username AND a.password = :password")

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

    @Column(name = "Username", unique = true, nullable = false)
    private String username;

    @Column(name = "Password", nullable = false)
    private String password;
}
```

EmployeeServicePackEntity

```
@Entity
@Table(name = "employeeServicePack", schema = "db2Project")

@NamedQueries({
@NamedQuery(
    name = "EmployeeServicePack.findById",
    query = "SELECT esp FROM EmployeeServicePackEntity esp WHERE esp.id = :id"),
@NamedQuery(
    name = "EmployeeServicePack.findByName",
    query = "SELECT esp FROM EmployeeServicePackEntity esp WHERE esp.name = :name"),

@NamedQuery(
    name = "EmployeeServicePack.findAll",
    query = "SELECT esp FROM EmployeeServicePackEntity esp")
})

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

@Column(name = "Name", nullable=false)
private String name;

@OneToMany(mappedBy = "service_pack_employee_id", cascade = {CascadeType.REMOVE, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH }, fetch = FetchType.EAGER)
private List<ServicePackageEntity> servicePacks; // relation made_of

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name = "offers", joinColumns = @JoinColumn(name = "EmployeeServicePack_id"), inverseJoinColumns = @JoinColumn(name = "Validity_period_id"))
private List<ValidityPeriodEntity> validityPeriodEntity; // owner of the relation offers

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name = "propose", joinColumns = @JoinColumn(name = "EmployeeServicePack_id"), inverseJoinColumns = @JoinColumn(name = "Optional_product_id"))
private List<OptionalProductEntity> optionalProductEntity; // owner of the relation propose

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name = "comprises", joinColumns = @JoinColumn(name = "EmployeeServicePack_id"), inverseJoinColumns = @JoinColumn(name = "Service_id"))
private List<ServiceEntity> serviceEntities; // owner of the relation comprises
}
```

OptionalProductEntity

```
@Entity
@Table(name = "optional_product", schema = "db2Project")

@NamedQuery(
      name = "OptionalProduct.findOptProdOfEmployeeServicePackId",
      query = "SELECT o FROM OptionalProductEntity o " +
            "JOIN o.employeeServicePackEntity s " +
            "WHERE s.id = :employeeServicePack_id "
),
@NamedQuery(
         name = "OptionalProduct.findByName",
         query = "SELECT o FROM OptionalProductEntity o " +
               "WHERE o.name = :optionalProduct_name"
   ),
      @NamedQuery(name = "OptionalProductEntity.findAllOptionalProduct", query = "SELECT op FROM OptionalProductEntity op"),
      @NamedQuery(name = "OptionalProductEntity.findByName", query = "SELECT op FROM OptionalProductEntity op WHERE op.name = :name"),
      @NamedQuery(name = "OptionalProductEntity.findAssociatedESP", query = "SELECT esp FROM OptionalProductEntity esp WHERE esp.employeeServicePackEntity = :name"),
      @NamedQuery(
            name = "OptionalProduct.findByID",
            query = "SELECT o FROM OptionalProductEntity o " +
                  "WHERE o.id = :optionalProduct_id"
      )

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

   @Column(name = "Name", nullable = false)
   private String name;


   @Column(name = "Fee", nullable = false)
   private float fee;

   @ManyToMany(mappedBy = "optionalProductEntities", fetch = FetchType.EAGER) // relazione has
   private List<ServicePackageEntity> servicePackageEntities;

   @ManyToMany(mappedBy = "optionalProductEntity", fetch = FetchType.EAGER) // relation propose
   private List<EmployeeServicePackEntity> employeeServicePackEntity;
}
```

OrderEntity

```
@Table(name = "order", schema = "db2Project")
@NamedQuery(name = "OrderEntity.findOrderById", query = "SELECT o FROM OrderEntity o WHERE o.id = :order_id"),
@NamedQuery(name = "OrderEntity.findFailedOrdersByUserId", query = "SELECT o FROM OrderEntity o WHERE o.user_id = :user AND o.isvalid=false"),
@NamedQuery(
      name = "Order.findOrderScheduledByUserId",
      query = "SELECT DISTINCT o FROM OrderEntity o JOIN o.service_pack_id s WHERE o.user_id = :user AND o.isvalid=true AND s.start_date > CURRENT_TIMESTAMP ")
// @NamedQuery(name = "Order.findAllOrderByUser", query = "SELECT o FROM OrderEntity o WHERE o.userOwner = :user ")

@Entity
public class OrderEntity {

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

@Column(name = "Totalcost", nullable = false)
private float totalcost;

@Column(name = "Isvalid", nullable = false)
private boolean isvalid;

@Column(name = "Timestamp", nullable = false)
private Timestamp timestamp;

@ManyToOne(fetch = FetchType.EAGER, optional = false)
@JoinColumn(name = "User_id")
private UserEntity user_id;

@OneToOne(optional = false, cascade = {CascadeType.REMOVE, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn(name = "Service_pack_id")
private ServicePackageEntity service_pack_id;
}
```

ServiceEntity

```
@Entity
@Table(name = "service", schema = "db2Project")
@NamedQuery(name = "ServiceEntity.findServiceByName", query = "SELECT s FROM ServiceEntity s WHERE s.type=:name"),
@NamedQuery(name = "ServiceEntity.findServiceById", query = "SELECT s FROM ServiceEntity s WHERE s.id=:id"),
@NamedQuery(name="ServiceEntity.findAll", query="SELECT s FROM ServiceEntity s")

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

  @Column(name = "Type", nullable = false)
   private String type;


   @Column(name = "Min_fee")
   private float min_fee;

   @Column(name = "Sms_fee")
   private float sms_fee;

   @Column(name = "Min")
   private int min;

   @Column(name = "Sms")
   private int sms;

   @Column(name = "Gb_fee")
   private float gb_fee;

   @Column(name = "Gb")
   private int gb;

   @ManyToMany(mappedBy = "serviceEntities", fetch = FetchType.LAZY) // relation comprises
   private List<EmployeeServicePackEntity> employeeServicePackEntity;
}
```

ServicePackEntity

```
@Table(name = "service_pack", schema="db2Project")
@NamedQuery(name = "ServicePackageEntity.findAll", query = "SELECT sp FROM ServicePackageEntity sp")
@NamedQuery(name = "ServicePackageEntity.findById", query = "SELECT sp FROM ServicePackageEntity sp WHERE sp.id = :id")

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

@Temporal(TemporalType.DATE)
@Column(name = "Start_date")
private Date start_date;

@Temporal(TemporalType.DATE)
@Column(name = "Deactivation_date")
private Date deactivation_date;

@Column(name = "Costpackage", unique = true, nullable = false)
private float costpackage;

@Column(name = "Totalcostoptionalproducts", unique = true, nullable = false)
private float totalcostoptionalproducts;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "Validity_period_id") // owner della relazione associate
private ValidityPeriodEntity validity_period_id;

@ManyToMany(fetch=FetchType.EAGER )
@JoinTable(name = "has", joinColumns = {@JoinColumn(name = "Service_pack_id")}, inverseJoinColumns = {@JoinColumn(name = "Optional_product_id")})
private List<OptionalProductEntity> optionalProductEntities; // owner of the relation has

@OneToOne(mappedBy = "service_pack_id", cascade = CascadeType.ALL, orphanRemoval = true)
private OrderEntity orders; // relazione in


@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "Service_pack_employee_id")
private EmployeeServicePackEntity service_pack_employee_id; // owner della relazione made_of
}
```

UserEntity

```
@Table(name = "user", schema = "db2Project")
@NamedQuery(name = "UserEntity.checkCredentials", query = "SELECT u FROM UserEntity u WHERE u.username = :username AND u.password = :password"),
@NamedQuery(name = "UserEntity.findByUsername", query = "SELECT u FROM UserEntity u WHERE u.username = :username"),
@NamedQuery(name = "UserEntity.findByEmail", query = "SELECT u FROM UserEntity u WHERE u.email = :email"),
@NamedQuery(name = "UserEntity.findById", query = "SELECT u FROM UserEntity u WHERE u.id = :user_id")

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

    @Column(name = "Username", unique=true, nullable=false)
    private String username;

    @Column(name = "Password", nullable = false)
    private String password;

    @Column(name = "Email", unique=true, nullable=false)
    private String email;

    @Column(name = "Flag_ins")
    private boolean flag_ins;

    @Column(name = "NumberOfFailedPayments")
    private int numberOfFailedPayments;

    @OneToMany(mappedBy = "user_id", fetch = FetchType.LAZY, cascade = CascadeType.ALL, orphanRemoval = true)
    private List<OrderEntity> orders;

    @OneToOne(mappedBy = "user_alert", fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true)
    private AlertEntity alert;
}
```

ValidityPeriodEntity

```
@Table(name = "validity_period", schema = "db2Project")
@NamedQuery(name = "ValidityPeriod.findByID", query = "SELECT v FROM ValidityPeriodEntity v WHERE v.id = :validityPeriod_id"),
@NamedQuery(name = "ValidityPeriod.findValidityPeriodsByEmployeeServicePack", query = "SELECT v FROM ValidityPeriodEntity v
        JOIN v.employeeServicePackEntity s WHERE s.id = :employeeServicePack_id "
@NamedQuery( name = "ValidityPeriod.getValidityPeriods", query = "SELECT x FROM ValidityPeriodEntity x"),
@NamedQuery(name = "ValidityPeriodEntity.findAllValidityPeriod", query = "SELECT vp FROM ValidityPeriodEntity vp"),
@NamedQuery(name = "ValidityPeriodEntity.findValidityPeriodById", query = "SELECT vp FROM ValidityPeriodEntity vp WHERE vp.id= :id")

@Entity
public class ValidityPeriodEntity {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "Id", nullable = false)
private int id;
@Column(name = "Monthly_fee", nullable = false)
private int monthly_fee;

@Column(name = "Months", nullable = false)
private int months;

@OneToMany(mappedBy = "validity_period_id", fetch=FetchType.LAZY, cascade = CascadeType.ALL, orphanRemoval = true) // relation associate
   private List<ServicePackageEntity> servicePackageEntities;

@ManyToMany(mappedBy = "validityPeriodEntity", fetch = FetchType.LAZY) // relation offers
private List<EmployeeServicePackEntity> employeeServicePackEntity = new ArrayList<>();
}
```

Components

Business Tier

UserService

- findUserById
- findUserByUsername
- findUserByEmail
- checkCredentials
- addNewUser
- incrementsFailedPayments
- setUserInsolvent

EmployeeServicePackService

- findAllEmployeeServicePack
- findEmployeeServicePackByName
- findEmployeeServicePackById
- addNewEmployeeServicePack

OptionalProductService

- findOptionalProductByName
- findAllOptionalProduct
- findOptionalProductAssociatedESP
- addNewOptionalProduct
- findOptProdOfEmployeeServicePackId
- findByOptProdID

OrderService

- createOrder
- findOrderByID
- updateOrder
- findFailedOrdersByUserId
- findOrderScheduledByUserId

SalesReportService

- findAllSalesPerPackage
- findAllAverageOPwithESP
- findAllNumberTotalPurchasesPerESP
- findAllNumberTotalPurchasesPerESPAndValidityPeriod
- findAllAlert
- findAllInsolvent
- findAllRejectedOrder
- findAllBest_seller_OP

ServicePackageService

- findAllServices
- findServicePackById
- createServicePackage

ServiceService

- findServiceByName
- findAllService
- findServiceById

ValidityPeriodService

- findAllValidityPeriod
- findValidityPeriodById
- findValidityPeriodsOfEmployeeServicePackId

EmployeeService

- checkCredentials

AlertService

- createAlert

Components 2

Web Tier

User

- BuypageServlet
- ConfirmationpageServlet
- HomepageServlet
- IndexServlet
- LoginServlet
- LogoutServlet
- RegisterServlet
- ServiceActivationScheduleServlet

Employee

- CreateESPServlet
- CreateOPServlet
- SalesReportServlet
- HomepageServlet
- IndexServlet
- LoginServlet
- LogoutServlet

Components 3

Client Tier

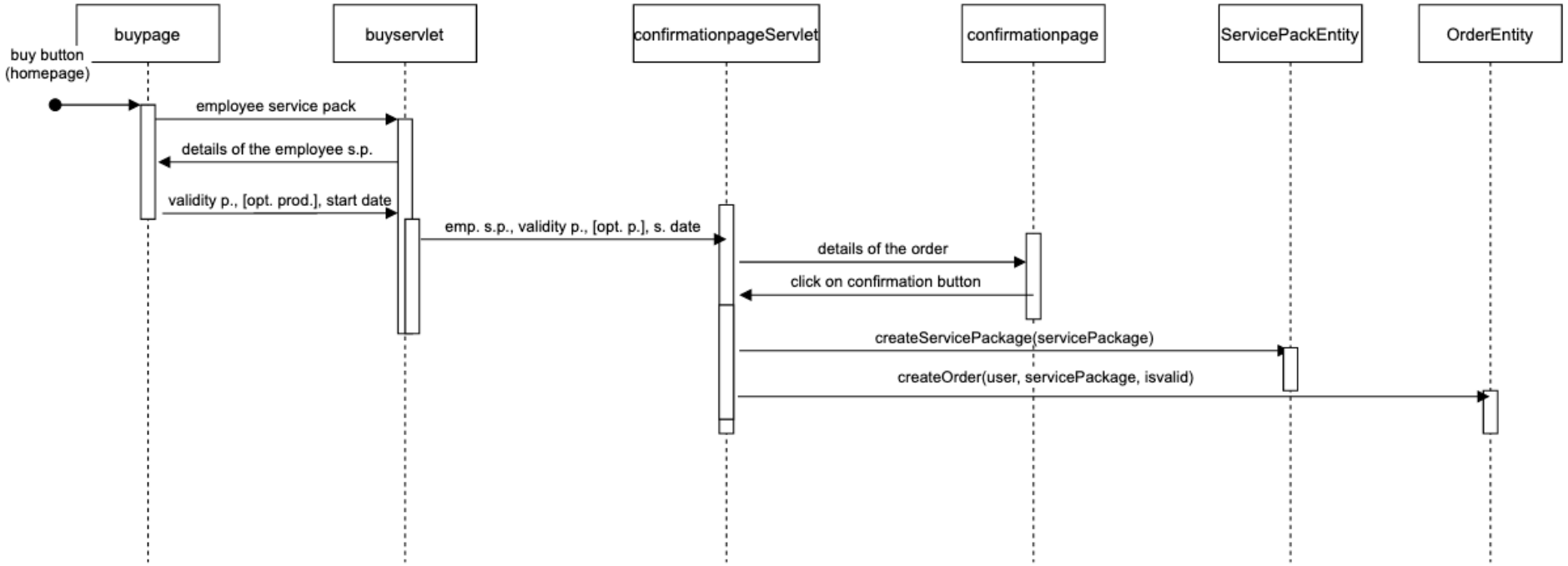
User

- index.html
- confirmationpage.html
- homepage.html
- buypage.html
- serviceactivationschedulepage.html

Employee

- index.html
- homepage.html
- newESP.html
- newOP.html
- salesReport.html

Order sequence diagram (Login already done)



The sequence diagram is intended as a general overview of the execution flow, some details are omitted for example the login during the purchase phase if the user is logged in as a guest.