Data bases 2

Optional JPA Project Documentation Group 197

Nicola della Volpe 10856875

Carlo Bellacoscia 10560632

Index

Specifications

The TlcProject Application is a web application that allow to buy service packages.

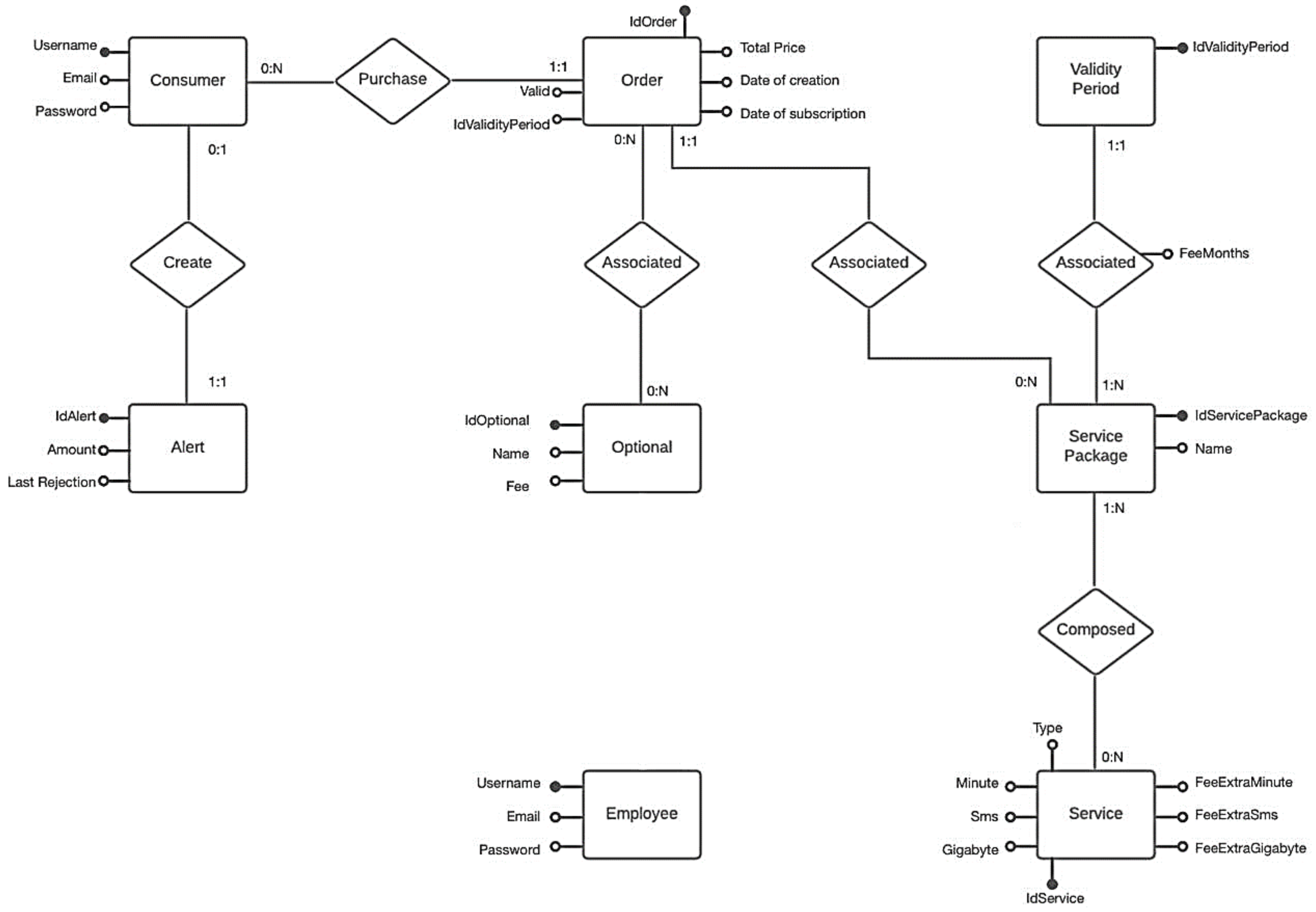
A service package is composed of an ID, a name, some services (which are of four types fixed phone, mobile phone, fixed internet, and mobile internet, each one with its own attributes). Each package has different validity period to choose from.

The TlcProject Application is composed of two parts, which are the consumerApplication and employeeApplication. The first page you find is the landingPage, which allow Login, Register or directly go to the homepage as a Guest. Either the consumer or the employee can login into the system through the Login form. Only the consumer can register whereas the employees’ credentials are already stored into the Database.

- **consumerApplication**: after logging in, the system displays the homePage, where there are all the packages bought by the user, the service packages available with a button which brings to the OderPage and the table of suspended order. In the OderPage the consumer can select which package he/she wants to buy, its validity period, the optional products (0 or more), and the start date (from which it will start the choosen validity period). The consumer clicks on the confirm order button, and goes to a Confirmation Page with the recap of the chosen preferences which can confirm. Then, an external service for the payment is called. If the transaction is successful, the order is valid, otherwise it is rejected, the consumer is marked as insolvent (by default it is not), and the order is added to the table of suspended orders in the home page. The consumer can select one of the orders in this table (in the homePage) and try to pay it again. If he/she causes three failed payments an alert will be created in the database. A guest can also access to the homePage without logging in. It is also possible to select the service package, the validity period, the optional products and the start date (in the OderPage), but the system does not allow you to create the order but displays a button that redirect you to the landing page and login first or eventually register and then login. After that, the new consumer is redirected to the Confirmation Page with the preferences chosen in the OrderPage, and now he/she can proceed with the payment.

- **employeeApplication**: after logging in the System displays the employeeHome, where he/she can creates a new service package, with all the needed data. The same page allow the employee to create optional products (providing the needed information, such as Name, Fee, Months). Also in the employeeHome there is the report button, which brings to the reportPage with all essential data about the sales and the users.

Entity Relationship



Motivations of the ER design

- Each cansumer has a counter indicating how many orders have not been paid for, if the count reaches three then an alert is created
- If an order is valid (if the payment is successful the attribute valid is set to 1). Otherwaise the order is rejected (the attribute valid is set to 0).
- Each time a consumer causes a failed payemnt, the attribute count increases, until three, when an alert is created.
- There are different Many to Many relationships representing:

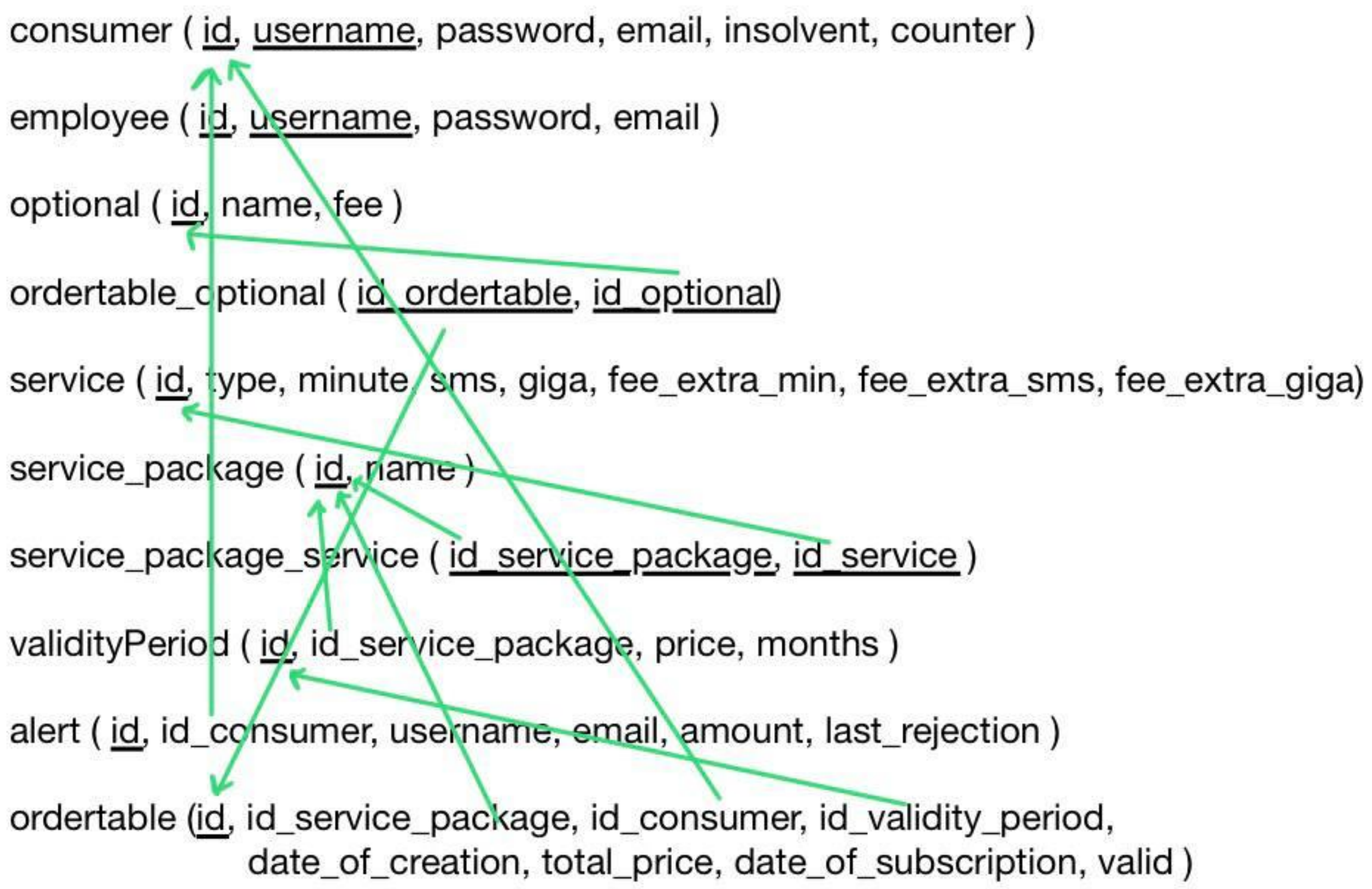
  Zero or many optional products can be selected in a Order.

  A servicePackage can have one or many validity period

  A servicePackage can have one or many Service

  A servicePackage can have zero or many optional products

Relational model



consumer (id, username, password, email, insolvent, counter)

employee (id, username, password, email)

optional (id, name, fee)

ordertable_optional (id_ordertable, id_optional)

service (id, type, minute, sms, giga, fee_extra_min, fee_extra_sms, fee_extra_giga)

service_package (id, name)

service_package_service (id_service_package, id_service)

validityPeriod (id, id_service_package, price, months)

alert (id, id_consumer, username, email, amount, last_rejection)

ordertable (id, id_service_package, id_consumer, id_validity_period,
date_of_creation, total_price, date_of_subscription, valid)

SQL DDL

```
CREATE TABLE `consumer` (
`username` VARCHAR(45) NOT NULL,
`password` VARCHAR(45) NOT NULL,
`email` VARCHAR(45) NOT NULL,
`count` INT NULL DEFAULT '0',
`insolvent` VARCHAR(45) NULL DEFAULT NULL,
PRIMARY KEY (`username`))

CREATE TABLE `employee` (

  `id` int NOT NULL AUTO_INCREMENT,

  `username` varchar(45) NOT NULL,

  `password` varchar(45) NOT NULL,

  `email` varchar(45) NOT NULL,

  PRIMARY KEY (`id`)
```

SQL DDL

```
CREATE TABLE `optional` (
  `id` int NOT NULL,
  `name` varchar(45) NOT NULL,
  `fee` int NOT NULL,
  PRIMARY KEY (`id`))
```

SQL DDL

```
CREATE TABLE `ordertable` (
  `id` int NOT NULL AUTO_INCREMENT,
  `date_of_creation` datetime NOT NULL,
  `total_price` int NOT NULL,
  `date_of_subscription` datetime NOT NULL,
  `id_service_package` int NOT NULL,
  `id_consumer` int NOT NULL,
  `valid` int NOT NULL,
  `id_validity_period` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_ordertable_service_package_idx` (`id_service_package`),
  KEY `fk_ordertable_usertable_idx` (`id_consumer`),
  KEY `fk_ordertable_validity_period_idx` (`id_validity_period`),
  CONSTRAINT `fk_ordertable_service_package` FOREIGN KEY
(`id_service_package`) REFERENCES `service_package` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_ordertable_usertable` FOREIGN KEY (`id_consumer`)
REFERENCES `consumer` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
```

SQL DDL

```
CREATE TABLE `service` (
  `id` int NOT NULL AUTO_INCREMENT,
  `type` varchar(45) NOT NULL,
  `minute` varchar(45) DEFAULT NULL,
  `sms` varchar(45) DEFAULT NULL,
  `giga` varchar(45) DEFAULT NULL,
  `fee_extra_min` varchar(45) DEFAULT NULL,
  `fee_extra_sms` varchar(45) DEFAULT NULL,
  `fee_extra_giga` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`))

CREATE TABLE `service_package` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`))
```

SQL DDL

```
CREATE TABLE `validity_period` (
  `id` int NOT NULL AUTO_INCREMENT,
  `id_service_package` int NOT NULL,
  `price` int NOT NULL,
  `months` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_validity_period_service_package_idx`
(`id_service_package`),
  CONSTRAINT `fk_validity_period_service_package` FOREIGN
KEY (`id_service_package`) REFERENCES `service_package`
(`id`) ON DELETE CASCADE ON UPDATE CASCADE
```

SQL DDL

```
CREATE TABLE alert (
  id_consumer int NOT NULL,
  username varchar(45) DEFAULT NULL,
  email varchar(45) DEFAULT NULL,
  amount int NOT NULL DEFAULT '0',
  last_rejection datetime DEFAULT NULL,
  PRIMARY KEY (id_consumer),
  KEY fk_alert_usertable_idx (id_consumer),
  CONSTRAINT fk_alert_usertable FOREIGN KEY (id_consumer) 
REFERENCES consumer (id) ON DELETE CASCADE ON UPDATE 
CASCADE
)
```

SQL DDL

```
CREATE TABLE ordertable_optional (
  id_ordertable int NOT NULL,
  id_optional int NOT NULL,
  id_service_package int not NULL,
  PRIMARY KEY (id_ordertable, id_optional)
   KEY fk_ordetable_optional_idx(id_optional),
  CONSTRAINT fk_ordetable_optional FOREIGN KEY
(id_optional) REFERENCES optional (id) ON DELETE CASCADE
ON UPDATE CASCADE,
  CONSTRAINT fk_optional_ordertable FOREIGN KEY
(id_ordertable) REFERENCES ordertable (id) ON DELETE
CASCADE ON UPDATE CASCADE
)
```

SQL DDL

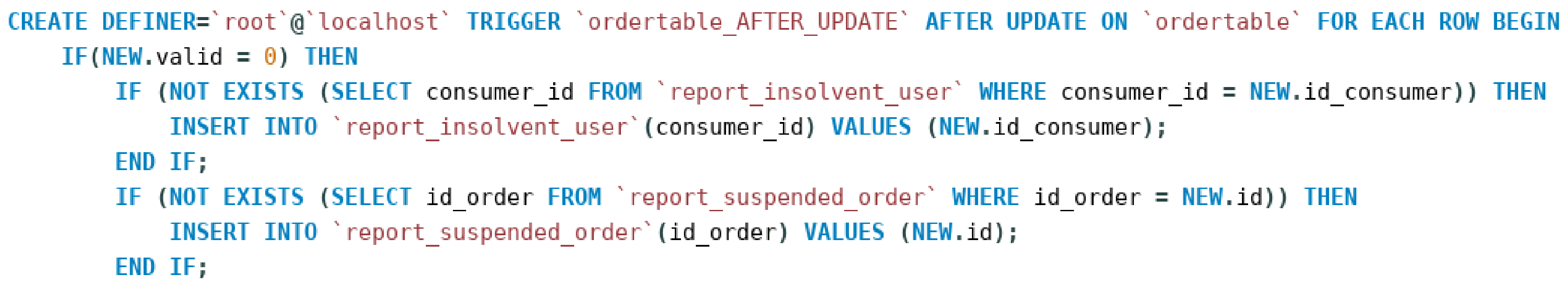
```
CREATE TABLE service_package_service (
  id_service_package int NOT NULL,
  id_service int NOT NULL,
  PRIMARY KEY (id_service_package,id_service),
  KEY fk_service_package_service_idx (id_service),
  CONSTRAINT fk_service_package_service FOREIGN KEY
(id_service) REFERENCES service (id) ON DELETE CASCADE ON
UPDATE CASCADE,
  CONSTRAINT fk_service_service_package FOREIGN KEY
(id_service_package) REFERENCES service_package (id) ON
DELETE CASCADE ON UPDATE CASCADE
)
```

Trigger

An Order is created the program simulates the payment and update the attribute valid of the

- Event : update on Order
- Condition :
    - The Order is not valid (is rejected) → valid = 0
- Actions
    - Update the corresponding Consumer count → increment count
    - Update the corresponding Consumer insolvent from “null” to “insolvent”
    - If count is equal to 3 then create the corresponding Alert Order

CODE



```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_UPDATE` AFTER UPDATE ON `ordertable` FOR EACH ROW BEGIN
    IF(NEW.valid = 0) THEN
        IF (NOT EXISTS (SELECT consumer_id FROM `report_insolvent_user` WHERE consumer_id = NEW.id_consumer)) THEN
            INSERT INTO `report_insolvent_user`(consumer_id) VALUES (NEW.id_consumer);
        END IF;
        IF (NOT EXISTS (SELECT id_order FROM `report_suspended_order` WHERE id_order = NEW.id)) THEN
            INSERT INTO `report_suspended_order`(id_order) VALUES (NEW.id);
        END IF;
```

Materialized view

- A sales reportage allow the employee to inspect the essential data about the sales and about the users. In order to show the different report, different materialized view are created :

1. Number of total purchase per package
2. Number of total purchases per package and validity period
3. Total value of sales per package with and without
4. Avarage number of optional product sold togheter with each service package
5. List of insolvent users
6. List of suspended Order
7. Best seller optional product

Materialized view : SQL DDL

```
CREATE TABLE
`report_avegare_optional_package` (
  `id_package` int NOT NULL,
  `total_sales` int DEFAULT NULL,
  `optional_sales` int DEFAULT NULL,
  `average_optional` float DEFAULT NULL,
  PRIMARY KEY (`id_package`))

CREATE TABLE `report_best_seller_optional` (
  `id_optional` int NOT NULL,
  `total_sales_optional` int NOT NULL,
  PRIMARY KEY (`id_optional`)
```

Materialized view : SQL DDL

```
CREATE TABLE `report_insolvent_user` (
  `consumer_id` int NOT NULL,
  PRIMARY KEY (`consumer_id`)

CREATE TABLE `report_purchase_package_validity_period` (
  `id_package` int NOT NULL,
  `id_validity_period` int NOT NULL,
  `total_purchase` int DEFAULT NULL,
  PRIMARY KEY (`id_package`,`id_validity_period`),
  KEY `fk_report_purchase_package_validity_period_validity_period_idx`
(`id_validity_period`),
  CONSTRAINT `fk_report_purchase_package_validity_period_service_package`
FOREIGN KEY (`id_package`) REFERENCES `service_package` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_report_purchase_package_validity_period_validity_period`
FOREIGN KEY (`id_validity_period`) REFERENCES `validity_period` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
```

Materialized view : SQL DDL

```
CREATE TABLE `report_purchases_package` (
  `id_service_package` int NOT NULL,
  `total_purchase` int DEFAULT NULL,
  PRIMARY KEY (`id_service_package`),
  CONSTRAINT `fk_report_purchases_package_service_package` FOREIGN
KEY (`id_service_package`) REFERENCES `service_package` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE))

CREATE TABLE `report_suspended_order` (
  `id_order` int NOT NULL,
  PRIMARY KEY (`id_order`))

CREATE TABLE `report_total_sales_package` (
  `id_package` int NOT NULL,
  `value_package` int DEFAULT NULL,
  PRIMARY KEY (`id_package`)
```

Trigger for the materialized view

1)Trigger for report_total_purchase_package

- Event: update on ordertable
- Condition : valid = 1, the order is valid
- Actions:

1. If not exist the id of Service Package then insert in report_total_purchcase_package the new values and then update to 1.
2. If the id of Service Package already exists then update the table report_total_purchase_package and increase the count to the corresponding id.

CODE

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_BEFORE_INSERT` BEFORE INSERT ON `ordertable` FOR EACH ROW BEGIN

    IF (NOT EXISTS (SELECT id_service_package FROM `report_purchases_package` WHERE id_service_package = NEW.id_service_package)) THEN
    INSERT INTO `report_purchases_package`(id_service_package,total_purchase) VALUES (NEW.id_service_package,0);
    END IF;



CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_UPDATE` AFTER UPDATE ON `ordertable` FOR EACH ROW BEGIN
    IF(NEW.valid = 1)THEN

        UPDATE`report_purchases_package` SET total_purchase = total_purchase+1
            WHERE id_service_package = NEW.id_service_package;
```

Trigger for the materialized view

2)Trigger for report_totalpurchase_validity_period

- Event: update on Ordetable
- Condition : valid = 1, the order is valid
- Actions:

1. If not exist the id of Service Package and the validity corresponding of the id, then insert in report_totalpurchase_validity_period the new values and and then update to 1.
2. If both the id of Service Package and the validity period already exists then update the table report_totalpurchase_validity_period and increase the count to the corresponding id.

CODE

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_BEFORE_UPDATE` BEFORE UPDATE ON `ordertable` FOR EACH ROW BEGIN

    IF (NOT EXISTS (SELECT id_package FROM `report_purchase_package_validity_period` WHERE id_package = NEW.id_service_package)) THEN
    INSERT INTO `report_purchase_package_validity_period`(id_package,id_validity_period,total_purchase)
    VALUES (NEW.id_service_package,NEW.id_validity_period,0);
    END IF;
```

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_UPDATE` AFTER UPDATE ON `ordertable` FOR EACH ROW BEGIN
    IF(NEW.valid = 1)THEN
        UPDATE`report_purchase_package_validity_period` SET total_purchase = total_purchase+1
            WHERE (id_package = NEW.id_service_package AND id_validity_period = NEW.id_validity_period);
```

Trigger for the materialized view

3)Trigger for report_total_sales_package

- Event: update on Ordertable
- Condition : valid = 1, the order is valid
- Actions:

1. If the IdService package doesn’t’ exist it the table

report_package_sales adds a new tuple in the table and update the corresponding total value.

2. If the IdService package already exists then updates the corresponding tuple and adds to the corresponding value without and with the new value calculated.

CODE

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_BEFORE_UPDATE` BEFORE UPDATE ON `ordertable` FOR EACH ROW BEGIN

    IF (NOT EXISTS (SELECT id_package FROM `report_total_sales_package` WHERE id_package = NEW.id_service_package)) THEN
    INSERT INTO `report_total_sales_package`(id_package,value_package) VALUES (NEW.id_service_package,0);
    END IF;



CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_UPDATE` AFTER UPDATE ON `ordertable` FOR EACH ROW BEGIN
    IF(NEW.valid = 1)THEN

        UPDATE`report_total_sales_package` SET  value_package = value_package + NEW.total_price
            WHERE id_package = NEW.id_service_package;
```

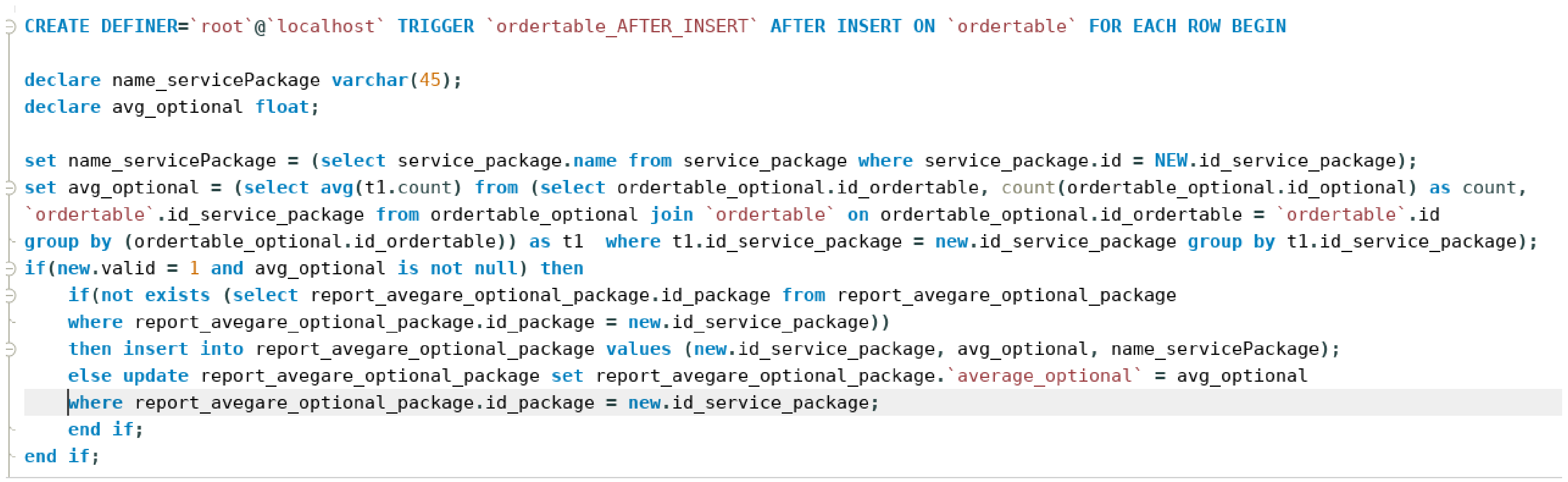
Trigger for the materialized view

4)Trigger for report_average_optional_package

- Event: update on Ordertable
- Condition : valid = 1, the order is valid
- Actions:

1. First we find how many optional products a consumer has selected in the new.IdOrder
2. Then we do the average for the corresponding idServicePackage
3. If the idServicePackage already exists in the table report_avg then update avg, else we add a new tuple with the calculated avg and the new idServicePackage.

CODE



```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_INSERT` AFTER INSERT ON `ordertable` FOR EACH ROW BEGIN

declare name_servicePackage varchar(45);
declare avg_optional float;

set name_servicePackage = (select service_package.name from service_package where service_package.id = NEW.id_service_package);
set avg_optional = (select avg(t1.count) from (select ordertable_optional.id_ordertable, count(ordertable_optional.id_optional) as count,
`ordertable`.id_service_package from ordertable_optional join `ordertable` on ordertable_optional.id_ordertable = `ordertable`.id
group by (ordertable_optional.id_ordertable)) as t1  where t1.id_service_package = new.id_service_package group by t1.id_service_package);
if(new.valid = 1 and avg_optional is not null) then
    if(not exists (select report_avegare_optional_package.id_package from report_avegare_optional_package
    where report_avegare_optional_package.id_package = new.id_service_package))
    then insert into report_avegare_optional_package values (new.id_service_package, avg_optional, name_servicePackage);
    else update report_avegare_optional_package set report_avegare_optional_package.`average_optional` = avg_optional
    where report_avegare_optional_package.id_package = new.id_service_package;
    end if;
end if;
```

Trigger for the materialized view

5)Trigger for report_suspended_order

- Event: update on Ordertable
- Condition :

1. Valid = 0
2. Valid = 1

- Actions case 1

If the username corresponding the new.Order doesn’t exist in the table report_suspended_order then insert the username in the table with the corresponding idOrder.

- Actions case 2

If the new.Order exists in the table then delete the report_suspended_order, decrement the count of the consumer and if the count of the consumer is 0 then delete the consumer from the table report_insolvent_user.

Trigger for the materialized view

6)Trigger for report_insolvet_user

- Event: update on Ordertable
- Condition :

1. Valid = 0
2. Valid = 1

- Actions case 1

If the username corresponding the new.Order doesn’t exist in the table report_suspended_order then insert the username in the table with the corresponding idConsumer.

- Actions case 2

If the new.Order exists in the table then delete the report_insolvet_user, decrement the count of the consumer and if the count of the consumer is 0 then delete the consumer from the table report_insolvent_user if there is no other invalid order.

CODE (for trigger 5 and 6)

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_AFTER_UPDATE` AFTER UPDATE ON `ordertable` FOR EACH ROW BEGIN
    IF(NEW.valid = 1)THEN

        IF (EXISTS (SELECT id_order FROM `report_suspended_order` WHERE id_order = NEW.id)) THEN
            DELETE FROM `report_suspended_order` WHERE id_order = NEW.id;
        END IF;
        IF (EXISTS (SELECT consumer_id FROM `report_insolvent_user` WHERE consumer_id = NEW.id_consumer) AND
            NOT EXISTS (SELECT id_consumer FROM `ordertable`
            WHERE id_consumer = (SELECT consumer_id FROM `report_insolvent_user` WHERE consumer_id = NEW.id_consumer) AND valid = 0)) THEN
            DELETE FROM `report_insolvent_user` WHERE consumer_id = NEW.id_consumer;
        END IF;


    END IF;

    IF(NEW.valid = 0) THEN
        IF (NOT EXISTS (SELECT consumer_id FROM `report_insolvent_user` WHERE consumer_id = NEW.id_consumer)) THEN
            INSERT INTO `report_insolvent_user`(consumer_id) VALUES (NEW.id_consumer);
        END IF;
        IF (NOT EXISTS (SELECT id_order FROM `report_suspended_order` WHERE id_order = NEW.id)) THEN
            INSERT INTO `report_suspended_order`(id_order) VALUES (NEW.id);
        END IF;
    END IF;
```

Trigger for the materialized view

7)Trigger for report_best_seller

- Event: update on ordertable_optional
- Condition : none
- Actions:

Count the total sales of the optionals in ordertable_optional, select the most purchased one and update the best seller in the table report_best_seller.

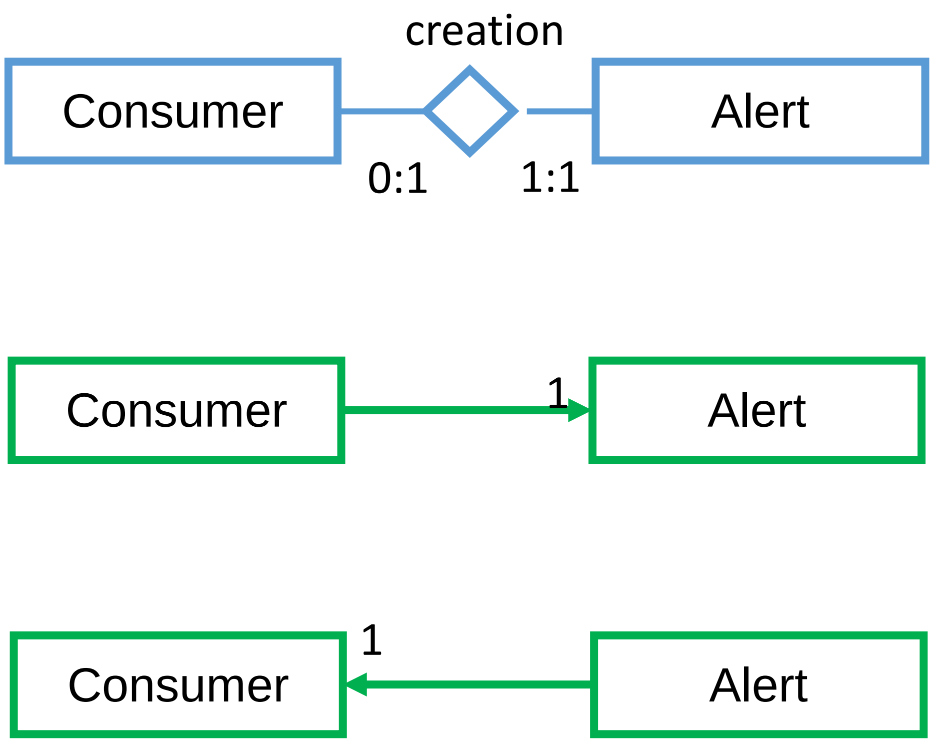
CODE

```sql
CREATE DEFINER=`root`@`localhost` TRIGGER `ordertable_optional_AFTER_UPDATE` AFTER UPDATE ON `ordertable_optional` FOR EACH ROW BEGIN

    IF( EXISTS (SELECT 1 FROM `ordertable_optional`)) THEN
    UPDATE `report_best_seller_optional`
        SET id_optional = (SELECT id_optional FROM `ordertable_optional` GROUP BY id_optional ORDER BY count(*) DESC LIMIT 1),
            total_sales_optional = (SELECT count(*) FROM `ordertable_optional` GROUP BY id_optional ORDER BY count(*) DESC LIMIT 1);
    ELSE UPDATE `report_best_seller_optional`SET id_optional = 0, total_sales_optional = 0;
    END IF;
END
```

ORM design

Relationship “consumer-alert”



- Consumer -> Alert

@OneToOne

show alerts of user

- Owner=Alert
- Fetch type LAZY

- Alert -> Consumer

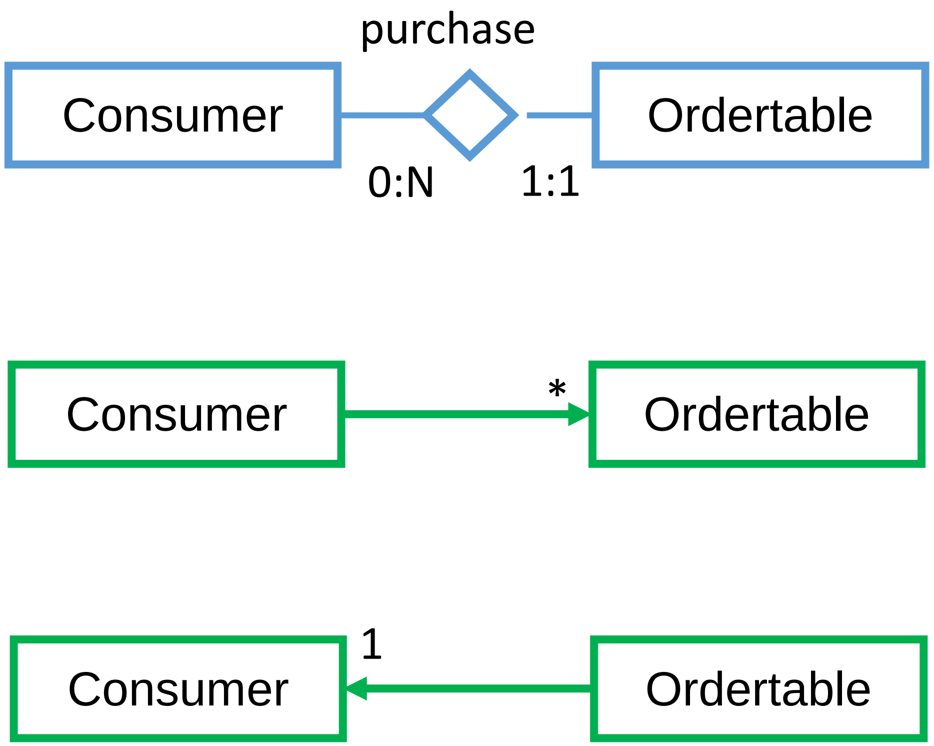
@OneToOne

show the an alert is

associated tu a

single consumer

Relationship "consumer-ordertable"



- Consumer -> Ordertable

@OneToMany

show list of order

Owner=Ordertable

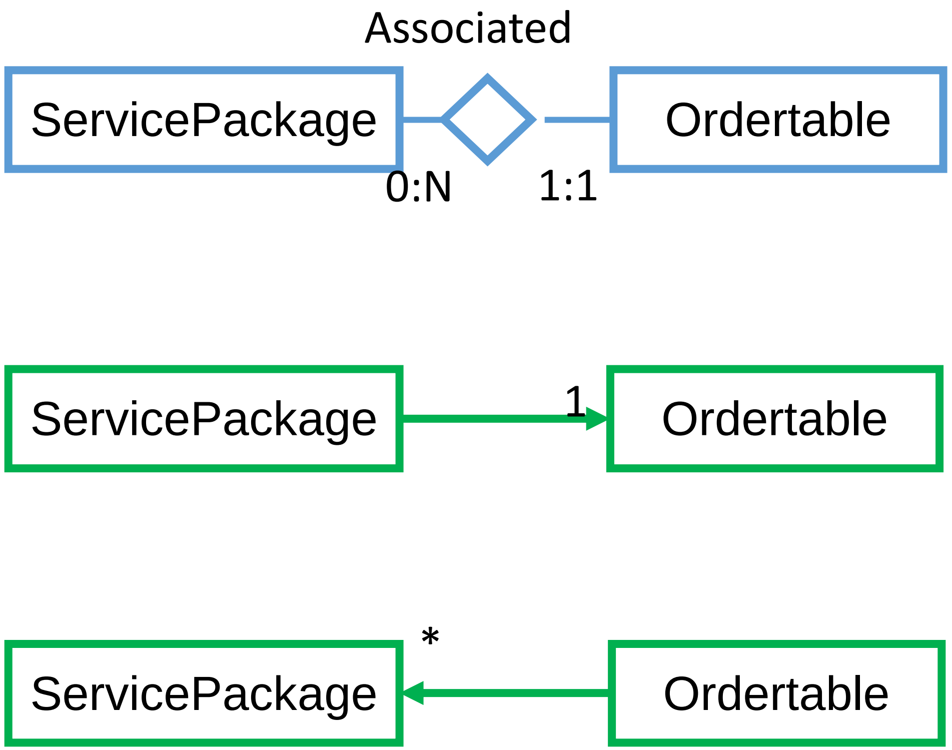
Fetch type EAGER

- Ordertable -> Consumer

@ManyToOne not necessary,

mapped for consistency

Relationship “servicePackage-ordertable”



- ServicePackage -> Ordertable

@OneToMany

mapped for consistency

Owner=Ordertable

Fetch = EAGER

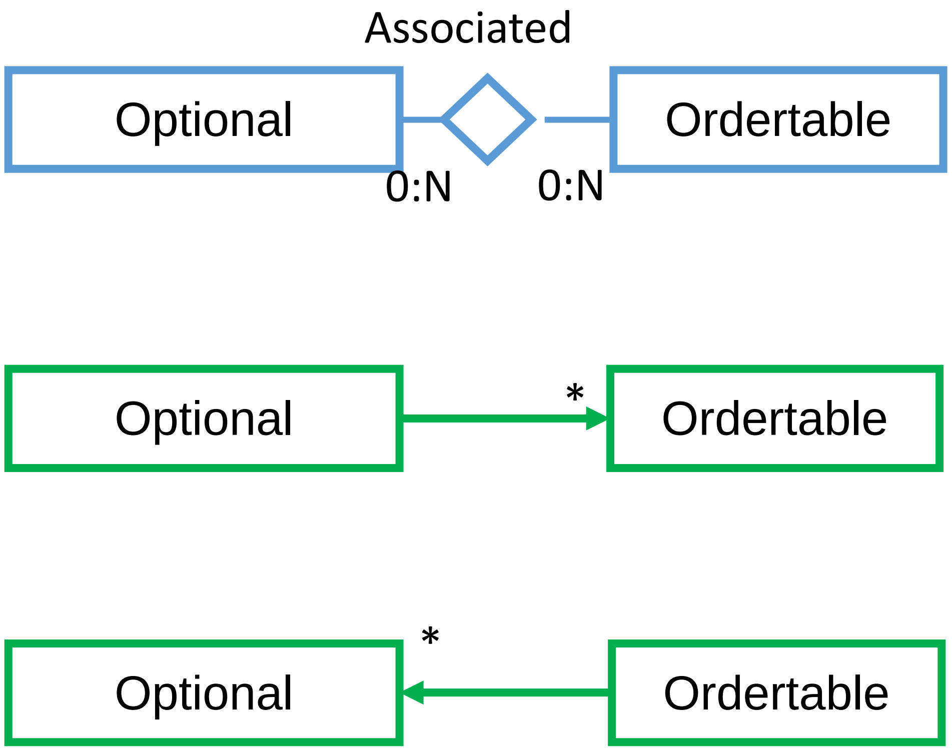
- Ordertable -> ServicePackage

@ManyToOne

show packages associated to

each order

Relationship “optional-ordertable”



- Optional -> Ordertable

@ManyToMany

mapped for consistency

Owner=Ordertable

Fetch = EAGER

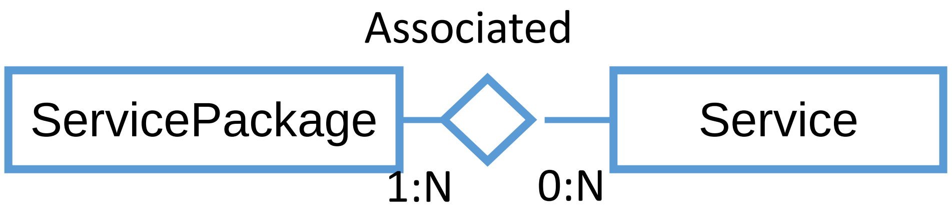
- Ordertable -> Optional

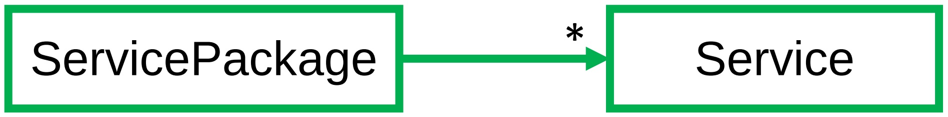
@ManyToMany

show optional associated to

each order

Relationship “servicePackage-service”





- ServicePackage -> Service

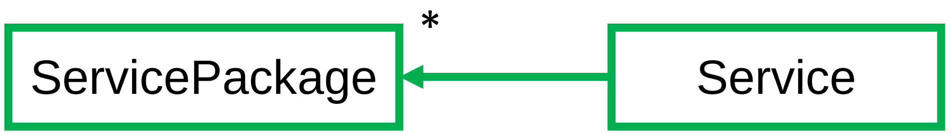
@ManyToMany

show services associated to

each package

Owner=Service

Fetch = EAGER

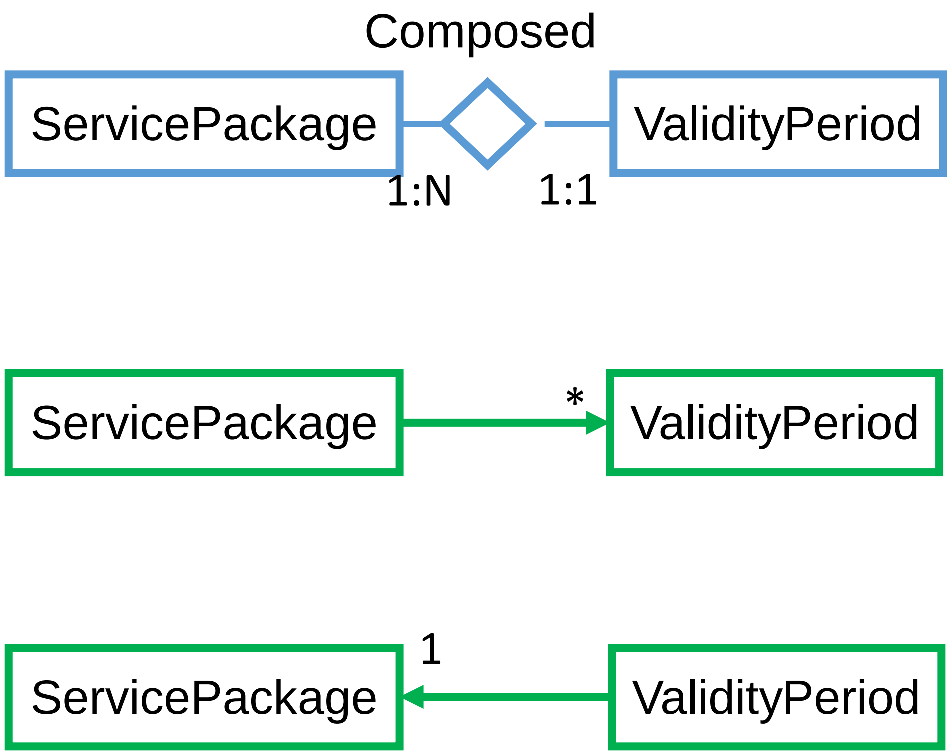


- Service -> ServicePackage

@ManyToMany

mapped for consistency

Relationship “servicePackage-validityPeriod”



- ServicePackage -> ValidityPeriod

@ManyToMany

show validity periods associated to each package

Owner=ValidityPeriod

Fetch = EAGER

- ValidityPeriod -> ServicePackage

@OneToOne

mapped for consistency

Entity Alert

```
@Entity
@Table(name="alert")
@NamedQuery(name="Alert.findAll", query="SELECT a FROM Alert a")
public class Alert implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  @Column(name="id_consumer")
  private int idConsumer;

  private int amount;

  private String email;

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name="last_rejection")
  private Date lastRejection;

  private String username;

  //bi-directional one-to-one association to Consumer
  @OneToOne
  @PrimaryKeyJoinColumn(name="id_consumer")
  private Consumer consumer;
```

Entity Consumer

```
@Entity
@Table(name="consumer")
@NamedQuery(name="Consumer.findAll", query="SELECT c FROM Consumer c")
@NamedQuery(name="Consumer.checkCredentials", query="SELECT c FROM Consumer c
WHERE c.username = ?1 and c.password = ?2")
public class Consumer implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  private int counter;

  private String email;

  private byte insolvent;

  private String password;

  private String username;

  //bi-directional one-to-one association to Alert
  @OneToOne(mappedBy="consumer")
  private Alert alert;

  //bi-directional many-to-one association to Ordertable
  @OneToMany(mappedBy="consumer")
  private List<Ordertable> ordertables;
```

Entity Employee

```
@Entity
@Table(name="employee")
@NamedQuery(name="Employee.findAll", query="SELECT e FROM Employee e")
@NamedQuery(name="Employee.checkCredentials", query="SELECT e FROM Employee e
WHERE e.username = ?1 and e.password = ?2")
public class Employee implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  private String email;

  private String password;

  private String username;
```

Entity Optional

```java
@Entity
@Table(name="optional")
@NamedQuery(name="Optional.findAll", query="SELECT o FROM Optional o")
public class Optional implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  private int fee;

  private String name;

  //bi-directional many-to-many association to ServicePackage
  @ManyToMany(mappedBy="optionals")
  private List<ServicePackage> servicePackages;
```

Entity Service

```
@Entity
@Table(name="service")
@NamedQuery(name="Service.findAll", query="SELECT s FROM Service s")
public class Service implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  @Column(name="fee_extra_giga")
  private String feeExtraGiga;

  @Column(name="fee_extra_min")
  private String feeExtraMin;

  @Column(name="fee_extra_sms")
  private String feeExtraSms;

  private String giga;

  private String minute;

  private String sms;

  private String type;

  //bi-directional many-to-many association to ServicePackage
  @ManyToMany(mappedBy="services")
  private List<ServicePackage> servicePackages;
```

Entity Ordertable

```
@Entity
@Table(name="ordertable")
@NamedQuery(name="Ordertable.findAll", query="SELECT o FROM Ordertable o")
public class Ordertable implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name="date_of_creation")
  private Date dateOfCreation;

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name="date_of_subscription")
  private Date dateOfSubscription;

  @Column(name="id_validity_period")
  private int idValidityPeriod;

  @Column(name="total_price")
  private int totalPrice;

  private int valid;

  //bi-directional many-to-one association to Consumer
  @ManyToOne
  @JoinColumn(name="id_consumer")
  private Consumer consumer;

  //bi-directional many-to-one association to ServicePackage
  @ManyToOne
  @JoinColumn(name="id_service_package")
  private ServicePackage servicePackage;
```

Entity Service_Package

```
@Entity
@Table(name="service_package")
@NamedQuery(name="ServicePackage.findAll", query="SELECT s FROM ServicePackage s")
public class ServicePackage implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  private String name;

  //bi-directional many-to-one association to Ordertable
  @OneToMany(mappedBy="servicePackage")
  private List<Ordertable> ordertables;

  //bi-directional many-to-one association to ReportPurchasePackageValidityPeriod
  @OneToMany(mappedBy="servicePackage")
  private List<ReportPurchasePackageValidityPeriod> reportPurchasePackageValidityPeriods;

  //bi-directional one-to-one association to ReportPurchasesPackage
  @OneToOne(mappedBy="servicePackage")
  private ReportPurchasesPackage reportPurchasesPackage;

  //bi-directional many-to-many association to Optional
  @ManyToMany
  @JoinTable(
    name="service_package_optional"
    , joinColumns={
      @JoinColumn(name="id_service_package")
      }
    , inverseJoinColumns={
      @JoinColumn(name="id_optional")
      }
    )
  private List<Optional> optionals;

  //bi-directional many-to-many association to Service
  @ManyToMany
  @JoinTable(
    name="service_package_service"
    , joinColumns={
      @JoinColumn(name="id_service_package")
      }
    , inverseJoinColumns={
      @JoinColumn(name="id_service")
      }
    )
  private List<Service> services;

  //bi-directional many-to-one association to ValidityPeriod
  @OneToMany(mappedBy="servicePackage")
  private List<ValidityPeriod> validityPeriods;
```

Entity Validity Period

```
@Entity
@Table(name="validity_period")
@NamedQuery(name="ValidityPeriod.findAll", query="SELECT v FROM ValidityPeriod
v")
@NamedQuery(name="ValidityPeriod.findByServicePackage", query="SELECT v FROM
ValidityPeriod v WHERE v.servicePackage.id = :packageId")
public class ValidityPeriod implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  private int id;

  private int months;

  private int price;

  //bi-directional many-to-one association to
ReportPurchasePackageValidityPeriod
  @OneToMany(mappedBy="validityPeriod")
  private List<ReportPurchasePackageValidityPeriod>
reportPurchasePackageValidityPeriods;

  //bi-directional many-to-one association to ServicePackage
  @ManyToOne
  @JoinColumn(name="id_service_package")
  private ServicePackage servicePackage;
```

Client Components

Servelets:

• CheckLogin: checks the Login credentials of the Employee or of the Consumer. If it is a consumer stores the info in the session and calls the GoToHomePage. Instead if it is an employee stores the info in the session and call the EmpCreate. If the Consumer logs in after he/she selects the service package to buy in the GoToConfirmation, it does not bring to the homePage, but directly to the ConfirmationPage.

• CreateOrder : verifies if the consumer wants to buy a suspended order, or wants to create a new one. Then it creates the order and calls the external payment service.

• EmpCreate: extracts the services and the optional products available and stores them in the web session. They are needed to create the service package in the employeePage.

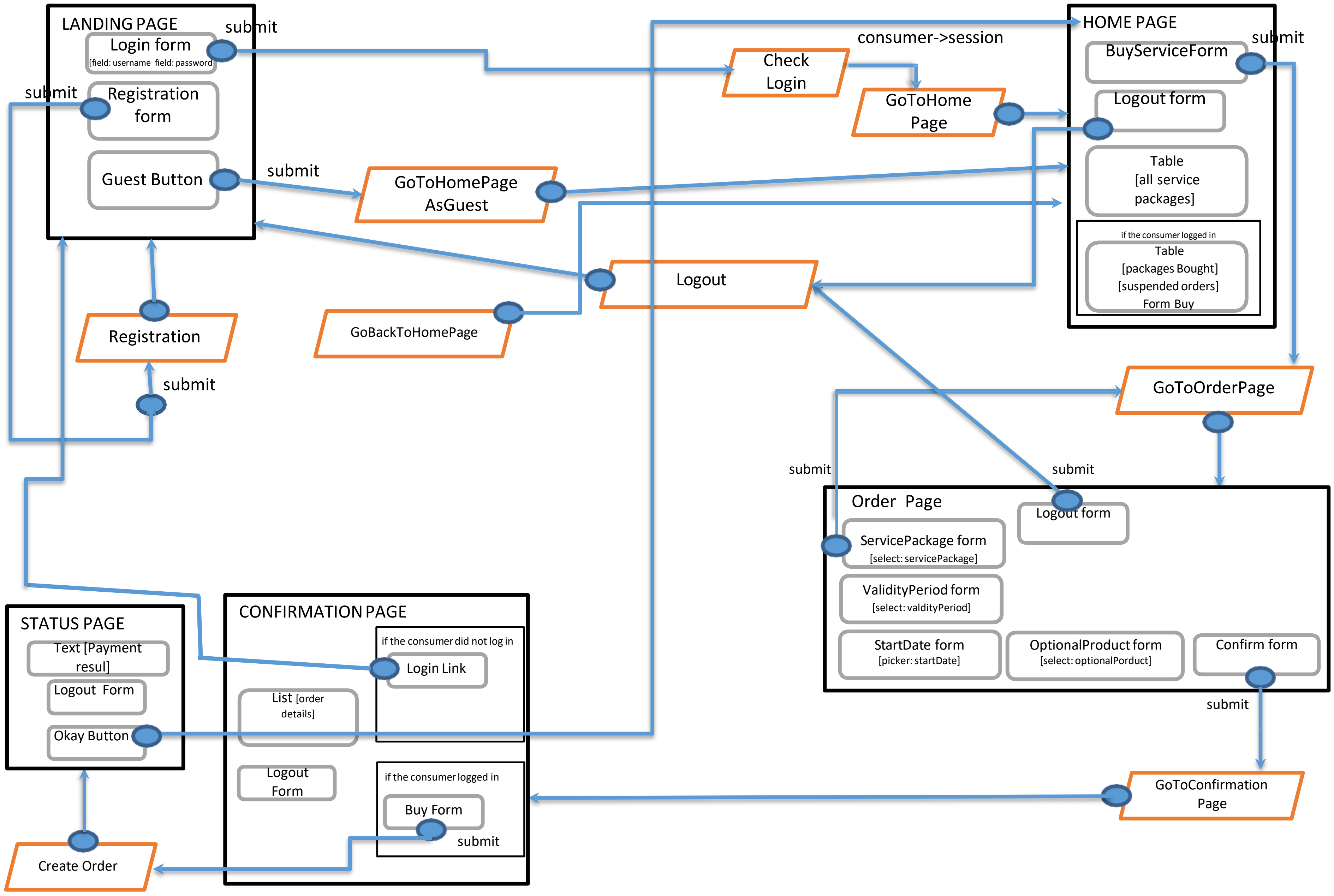
• GoToConfirmation: allows to display the information about what the consumer selected in the GoToOrderPage.

• GoToHomeAsGuest: creates a false consumer «Guest» and go to GoToHomePage.

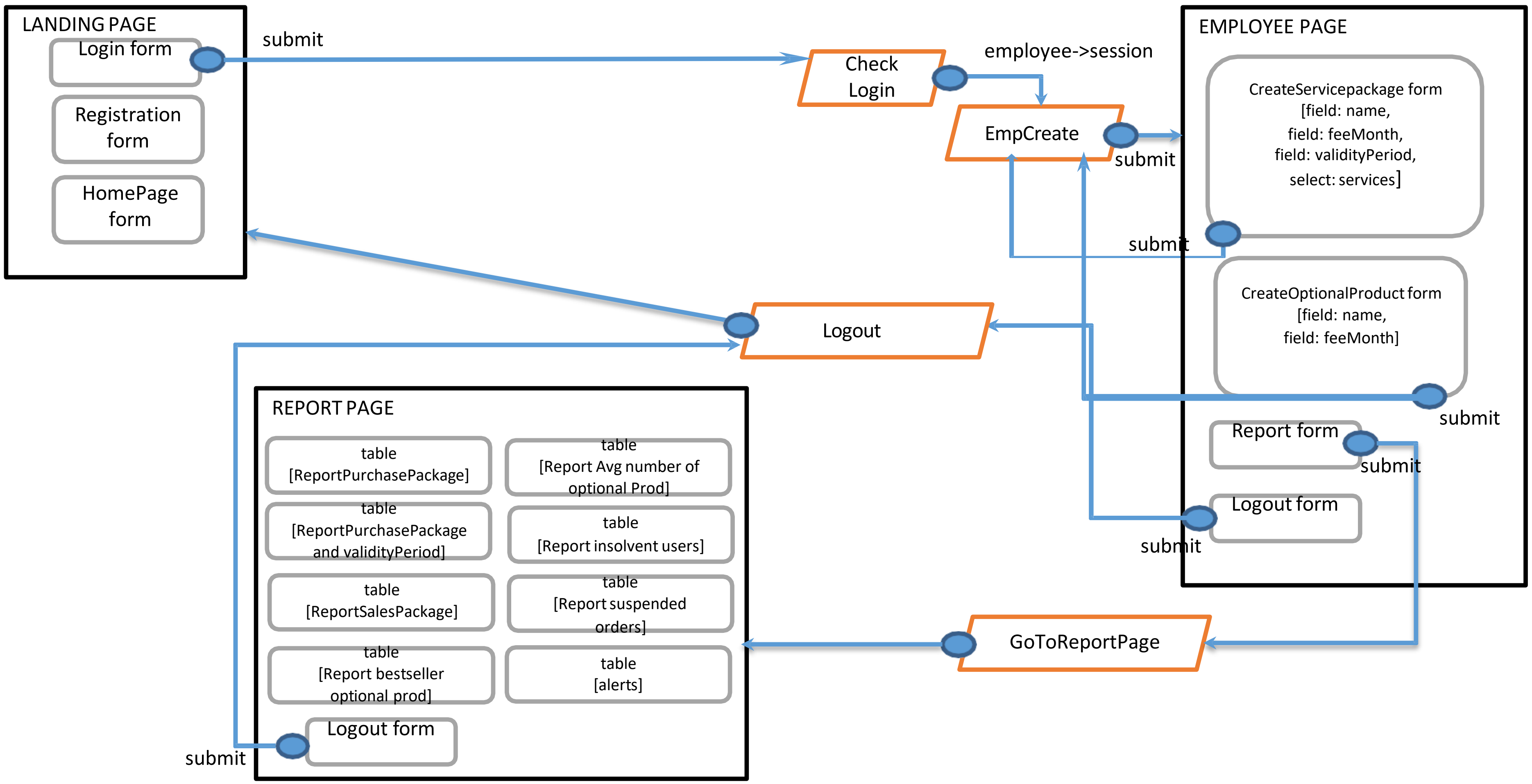
Client Components

- GoToHomePage: if the consumer is logged extracts all the packages bought and also his/her suspended orders. In any case (i.e., if the user is logged in or not) it shows all the available package of the Telecomunication company.
- GoToOrderPage: extracts the available package, optional product, services and validity period, it allows also to select the start date of subscription. All these data are needed to create the order.
- GoToReportPage: extracts all the reports for the employee.
- Logout: invalidates the session and go back to the Landing Page
- Register: fetches the data provided by the Consumer and calls the createConsumer, then it returns to the landingPage.

Functional analysis of the interaction: Consumer Application



Functional analysis of the interaction: Employee Application



LANDING PAGE
Login form
Registration form
HomePage form
submit
Check Login
employee->session
EmpCreate
submit
EMPLOYEE PAGE
CreateServicepackage form [field: name, field: feeMonth, field: validityPeriod, select: services]
submit
CreateOptionalProduct form [field: name, field: feeMonth]
submit
Report form
submit
Logout form
submit
Logout
GoToReportPage
REPORT PAGE
table [ReportPurchasePackage]
table [Report Avg number of optional Prod]
table [ReportPurchasePackage and validityPeriod]
table [Report insolvent users]
table [ReportSalesPackage]
table [Report suspended orders]
table [Report bestseller optional prod]
table [alerts]
Logout form
submit

Back end components

- Entities
  - Alert
  - Consumer
  - Employee
  - Optional
  - Ordertable
  - OrdertableOptional
  - Service
  - Servicepackage
  - ValidityPeriod

Back end components

- Views
  - ReportAverageOptionalPackage
  - ReportBestsellerOptional
  - ReportInsolventUser
  - ReportPackageSale
  - ReportSuspendedOrder
  - ReportPurchasepackageValidityPeriod
  - ReportPurchasepackageValidityPeriodPK
  - ReportPurchasePackage
  - ReportTotalSalesPackage

Business Components (EJBs)

- @Stateless ConsumerService
  - `Void CreateProfile(Consumer c)`
  - `Consumer checkCredentials(String usrn, String pwd)`
- @Stateless EmployeeService
  - `Telcoemployee getEmployee(int idEmployee)`
  - `Void createProfile(Employee e)`

Business Components (EJBs)

- @Stateless OptionalService
  - `List<Optional> findAllOptionals()`
  - `Optional findOptionaleById(int optionalId)`
  - `void createOptional(Optional optional)`
- @Stateless ServiceService
  - `Service findServiceById(int serviceId)`
  - `List<Service> findAllServices()`

Business Components (EJBs)

- @Stateless ReportService
  - `ReportService`
  - `List<ReportPurchasesPackage> findAllReportPurchasesPackage()`
  - `List<ReportPurchasePackageValidityPeriod> findAllReportPurchasePackageValidityPeriod()`
  - `List<ReportAvegareOptionalPackage> findAllReportAvegareOptionalPackage()`
  - `List<ReportTotalSalesPackage> findAllReportTotalSalesPackage()`
  - `List<ReportBestSellerOptional> findAllReportBestSellerOptional()`
  - `List<ReportInsolventUser> findAllReportInsolventUsers()`
  - `List<ReportSuspendedOrder> findAllReportSuspendedOrders()`
  - `List<Alert> findAllAlerts()`

Business Components (EJBs)

- @Stateless OrderService
  - `List<Ordertable> findOrdertablesByConsumer(int consumerId)`
  - `List<Ordertable> findOrdertablesByConsumerRefresh(int consumerId)`
  - `Ordertable findOrdertableById(int orderId)`
  - `Ordertable createOrdertable(ServicePackage servicePackage,Consumer consumer,Date dateOfCreation, int totalPrice, Date dateOfSubscription, int valid, int idValidityPeriod)`
  - `void createOrdertableOptional(Ordertable ordertable, List<Optional> optionals)`
  - `Ordertable updateOrdertable(Ordertable rejectOrdertable, int consumerId)`
  - `int casualValidBit()`

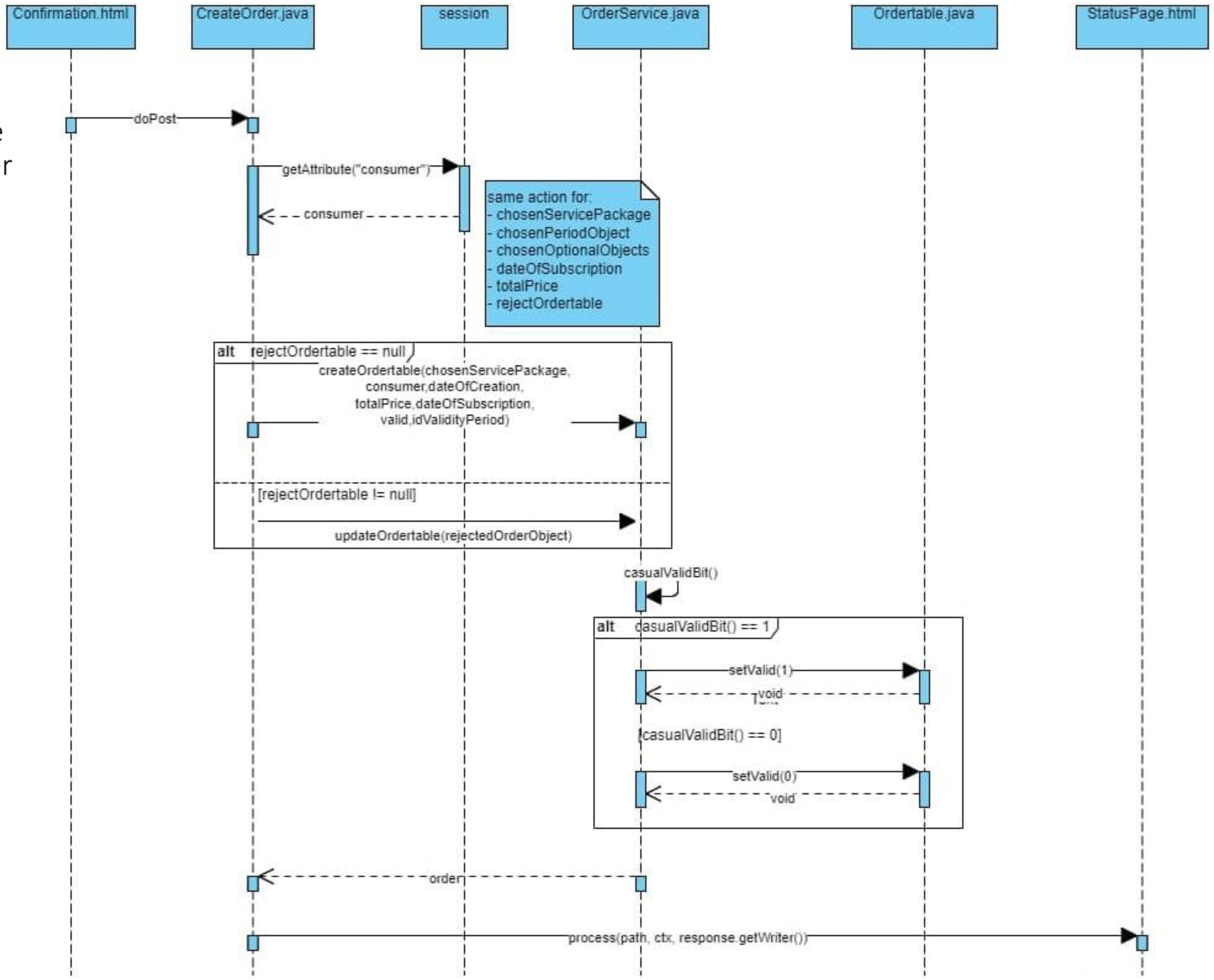
Business Components (EJBs)

- @Stateless ServicePackageService
  - `List<ServicePackage> findAllPackages()`
  - `ServicePackage findServicePackageById(int packageId)`
  - `ServicePackage findDefault()`
  - `void createServicePackage(int id, String packageName,List<Service> services, List<ValidityPeriod> validityPeriods)`
  - `void createServicePackageNoId(String packageName,List<Service> services, List<ValidityPeriod> validityPeriods)`
  - `void createServicePackageId(int id)`

Business Components (EJBs)

- @Stateless ValidityPeriodService
  - `List<ValidityPeriod> findByServicePackage(int packageId)`
  - `ValidityPeriod findValidityPeriodById(int periodId)`
  - `ValidityPeriod createValidityPeriod(ServicePackage servicePackage, int months, int price)`
  - `ValidityPeriod createValidityPeriodWId(int id, ServicePackage servicePackage, int months, int price)`

UML sequence diagram : Order Creation and Payment


Confirmation.html
CreateOrder.java
session
OrderService.java
Ordertable.java
StatusPage.html
doPost
getAttribute("consumer")
consumer
same action for:
- chosenServicePackage
- chosenPeriodObject
- chosenOptionalObjects
- dateOfSubscription
- totalPrice
- rejectOrdertable
alt rejectOrdertable == null
createOrdertable(chosenServicePackage, consumer,dateOfCreation, totalPrice,dateOfSubscription, valid,idValidityPeriod)
[rejectOrdertable != null]
updateOrdertable(rejectedOrderObject)
casualValidBit()
alt casualValidBit() == 1
setValid(1)
void
[casualValidBit() == 0]
setValid(0)
void
order
process(path, ctx, response.getWriter())