Database 2 Project

2021/2022

Giulia Forasassi - 10770107
Alessandro Barbiero - 10692413

Index

SPECIFICATIONS

Telco company offers pre-paid online services.
Two client application are developed using the same db.

CONSUMER can access:

- **Landing page:** registration and login forms; it also permits to enter as a guest
- **Home page:** displays service packages offered by Telco Company
- **Buy service page:** form for purchasing a service package creating a subscription
- **Confirmation page:** summarizes all the details of the order made

EMPLOYEE can access:

- **Login page:** login form
- **Home page:** with a form for creating service packages and optional products
- **Sales report page:** allows the employee to inspect the essential data about the sales and the users

Other details

- 4 types of **Services**
  - Mobile Phone
  - Mobile Internet
  - Fixed Phone
  - Fixed Internet
- the validity period of the optional product is the same of the service package associated
- if the payment for the order **is not accepted**
  - order status = rejected
  - user = insolvent
- if the payment for the order is **accepted**
  - order status = valid
  - creation of the Activation Schedule
- If an user causes 3 **failed payments** an alert is created in the dedicated table

Sales report page displays:

- Number of total purchases per package
- Number of total purchase per package and validity period
- Total value of sales per package with and without 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

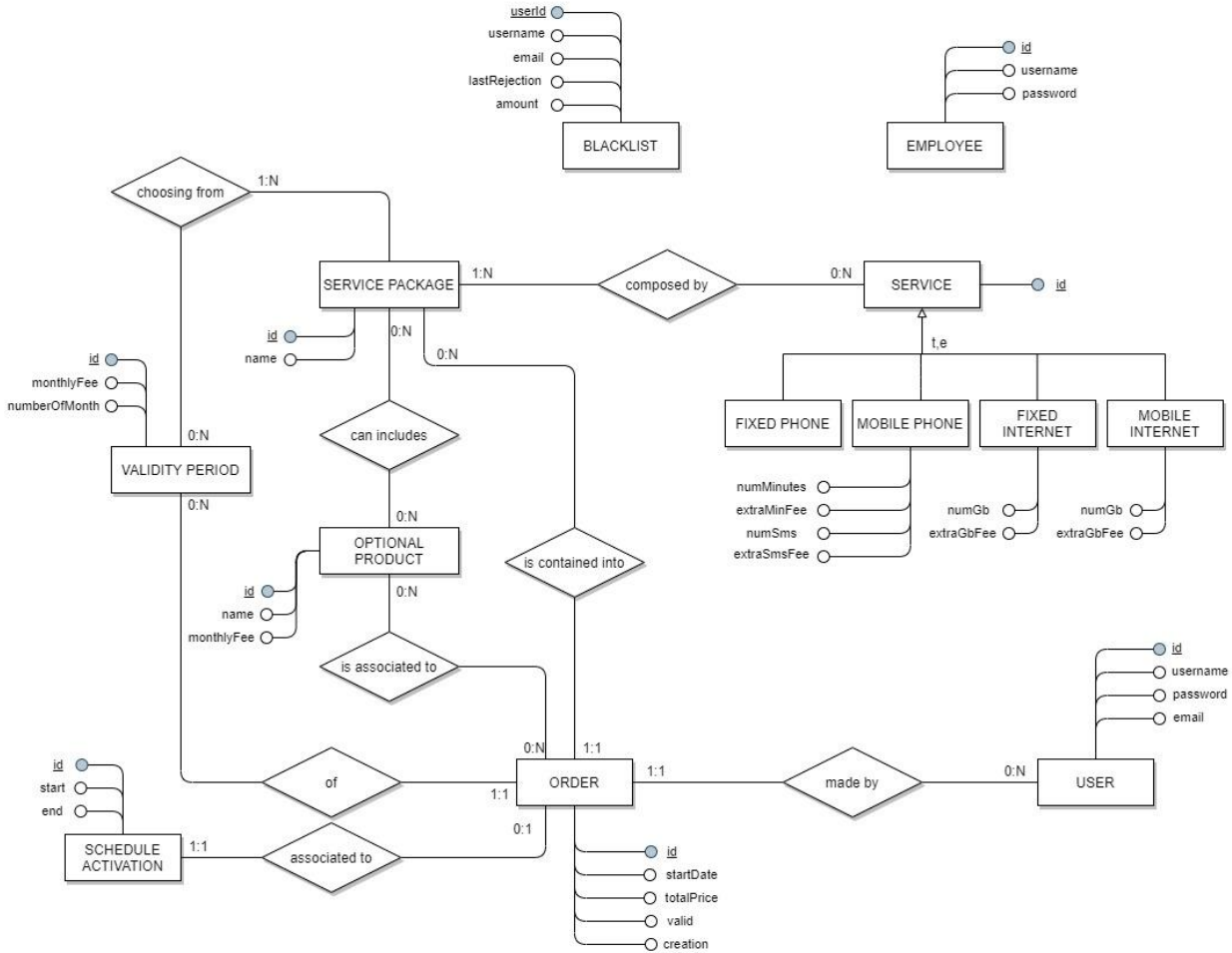
The aggregated data of sales report must be computed with Triggers.

Assumptions

- Validity periods and Services
  - are given by the company
  - the employee cannot create new ones: he can combine the elements given by the company in order to create service packages

- The Employee does not register himself on the platform
  - credentials are given by the company

ER DIAGRAM


userId
username
email
lastRejection
amount
BLACKLIST
id
username
password
EMPLOYEE
choosing from
1:N
SERVICE PACKAGE
1:N
composed by
0:N
SERVICE
id
t,e
FIXED PHONE
MOBILE PHONE
FIXED INTERNET
MOBILE INTERNET
numMinutes
extraMinFee
numSms
extraSmsFee
numGb
extraGbFee
numGb
extraGbFee
id
name
0:N
0:N
id
monthlyFee
numberOfMonth
0:N
VALIDITY PERIOD
0:N
can includes
0:N
OPTIONAL PRODUCT
0:N
id
name
monthlyFee
is contained into
is associated to
id
start
end
SCHEDULE ACTIVATION
1:1
of
associated to
0:N
1:1
ORDER
1:1
0:1
1:1
made by
0:N
USER
id
username
password
email
id
startDate
totalPrice
valid
creation

Assumptions

- Employee is not connected to any other entity
  - the company does not care about which employee creates a specific service package
- Blacklist is not connected to any other entity is only a recap table

RELATIONAL MODEL

User and Employee Tables

```
CREATE TABLE `user` (
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(255) NOT NULL UNIQUE,
`password` varchar(255) NOT NULL,
`email` varchar(255) NOT NULL UNIQUE,
PRIMARY KEY (`id`)
);
```

```
CREATE TABLE `employee` (
`id` int NOT NULL AUTO_INCREMENT,
`username` varchar(255) NOT NULL UNIQUE,
`password` varchar(255) NOT NULL,
PRIMARY KEY (`id`)
);
```

Service and Service Package Tables

```
CREATE TABLE `service` (

`id` int NOT NULL AUTO_INCREMENT,

`type` enum('FIXED PHONE', 'MOBILE PHONE',
'FIXED_INTERNET', 'MOBILE_INTERNET') NOT NULL,

`numberOfSms` int,

`numberOfMinutes` int,

`extraMinutesFee` FLOAT,

`extraSmsFee` FLOAT,

`numberOfGb` int,

`extraGbFee` FLOAT,

PRIMARY KEY (`id`)

);
```

```
CREATE TABLE `service_package` (

`id` int NOT NULL AUTO_INCREMENT ,

`name` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

);
```

Validity Period and Optional Product Tables

```
CREATE TABLE `validity_period` (
`id` int NOT NULL AUTO_INCREMENT,
`numberOfMonths` int NOT NULL,
`monthlyFee` FLOAT NOT NULL,
PRIMARY KEY (`id`)
);
```

```
CREATE TABLE `optional_product` (
`id` int NOT NULL AUTO_INCREMENT,
`name` varchar(255) NOT NULL,
`monthlyFee` FLOAT NOT NULL,
PRIMARY KEY (`id`)
);
```

Order and ServiceComposition Tables

```
CREATE TABLE `order` (
`id` int NOT NULL AUTO_INCREMENT,
`valid` bool NOT NULL,
`startDate` DATE NOT NULL,
`creation` TIMESTAMP NOT NULL,
`totalPrice` float NOT NULL,
`servicePackageId` int NOT NULL,
`userId` int NOT NULL,
`validityPeriodId` int NOT NULL,
PRIMARY KEY (`id`)
);
```

```
CREATE TABLE `service_composition` (
`serviceId` int NOT NULL,
`servicePackageId` int NOT NULL,
PRIMARY KEY
(`serviceId`,`servicePackageId`)
);
```

Constraints

ORDER

```
ALTER TABLE `order` ADD CONSTRAINT `order_fk0` FOREIGN KEY (`servicePackageId`) REFERENCES `service_package` (`id`);

ALTER TABLE `order` ADD CONSTRAINT `order_fk1` FOREIGN KEY (`userId`) REFERENCES `user` (`id`);

ALTER TABLE `order` ADD CONSTRAINT `order_fk2` FOREIGN KEY (`validityPeriodId`) REFERENCES `validity_period` (`id`);
```

SERVICE COMPOSITION

```
ALTER TABLE `service composition` ADD CONSTRAINT `service_composition_fk0` FOREIGN KEY (`serviceId`) REFERENCES `service` (`id`);

ALTER TABLE `service composition` ADD CONSTRAINT `service_composition_fk1` FOREIGN KEY (`servicePackageId`) REFERENCES `service_package` (`id`);
```

Join Tables

```
CREATE TABLE `optional_product_choice` (

`optionalProductId` int NOT NULL,

`orderId` int NOT NULL,

PRIMARY KEY (`optionalProductId`,`orderId`)

);

CREATE TABLE `possible_validity_period` (

`validityPeriodId` int NOT NULL,

`servicePackageId` int NOT NULL,

PRIMARY KEY
(`validityPeriodId`,`servicePackageId`)

);
```

```
CREATE TABLE `possible_extensions` (

`optionalProductId` int NOT NULL,

`servicePackageId` int NOT NULL,

PRIMARY KEY
(`optionalProductId`,`servicePackageId`)

);
```

Constraints

OPTIONAL PRODUCT CHOICE

```
ALTER TABLE `optional product choice`ADD CONSTRAINT `optional_product_choice_fk0`FOREIGN KEY (`optionalProductId`)
REFERENCES `optional_product`(`id`);

ALTER TABLE `optional product_choice`ADD CONSTRAINT `optional_product_choice_fk1`FOREIGN KEY (`orderId`)
REFERENCES `order`(`id`);
```

POSSIBLE VALIDITY PERIOD

```
ALTER TABLE `possible validity period`ADD CONSTRAINT `possible_validity_period_fk0`FOREIGN KEY
(`validityPeriodId`) REFERENCES `validity_period`(`id`);

ALTER TABLE `possible validity period`ADD CONSTRAINT `possible_validity_period_fk1`FOREIGN KEY
(`servicePackageId`) REFERENCES `service_package`(`id`);
```

POSSIBLE EXTENSIONS

```
ALTER TABLE `possible_extensions`ADD CONSTRAINT `possible_extensions_fk0`FOREIGN KEY (`optionalProductId`)
REFERENCES `optional_product`(`id`);

ALTER TABLE `possible extensions`ADD CONSTRAINT `possible_extensions_fk1`FOREIGN KEY (`servicePackageId`)
REFERENCES `service_package`(`id`);
```

Blacklist and Schedule Activation Tables

```
CREATE TABLE `blacklist` (
`userId` int NOT NULL,
`username` varchar(255) NOT NULL,
`email` varchar(255) NOT NULL,
`lastRejection` TIMESTAMP NOT NULL,
`amount` FLOAT NOT NULL,
PRIMARY KEY (`userId`)
);
```

```
CREATE TABLE `schedule_activation` (
`id` int NOT NULL AUTO_INCREMENT,
`orderId` int NOT NULL,
`startDate` DATE NOT NULL,
`endDate` DATE NOT NULL,
PRIMARY KEY (`id`)
);
```

Constraints

BLACKLIST

```
ALTER TABLE `blacklist` ADD CONSTRAINT `blacklist_fk0` FOREIGN KEY (`userId`)
REFERENCES `user`(`id`);
```

SCHEDULE ACTIVATION

```
ALTER TABLE `schedule_activation` ADD CONSTRAINT `schedule_activation_fk0` FOREIGN
KEY (`orderId`) REFERENCES `order`(`id`);
```

Materialized view tables

```
CREATE TABLE `purchases_per_package` (
`servicePackageId` int NOT NULL,
`totalPurchases` int NOT NULL
);
```

```
CREATE TABLE
`purchases_per_package_and_vp` (
`servicePackageId` int NOT NULL,
`validityPeriodId` int NOT NULL,
`totalPurchases` int NOT NULL
);
```

Materialized view tables

```
CREATE TABLE `sales_per_package` (
`servicePackageId` int NOT NULL,
`revenueWithOpProd` int NOT NULL,
`revenueWithoutOpProd` int NOT NULL
);
```

```
CREATE TABLE
`average_number_opt_products_per_package
` (
`servicePackageId` int NOT NULL,
`optProductsSold` int NOT NULL,
`totOrders` int NOT NULL,
`avg` float NOT NULL
);
```

Materialized view tables

```
CREATE TABLE `insolvent_users` (
`id` int NOT NULL UNIQUE,
`username` varchar(255) NOT NULL UNIQUE,
`email` varchar(255) NOT NULL UNIQUE
);
```

```
CREATE TABLE `best_seller_opt_prod` (
`optionalProductId` int NOT NULL,
`revenue` int NOT NULL
);
```

Choices

- The different types of service are collapsed in the parent class inserting an attribute “type” to distinguish them

TRIGGER

Total purchases per package - 1

```
CREATE TRIGGER total_purchases_after_insert_sp
    AFTER INSERT ON service_package
    FOR EACH ROW
        INSERT INTO purchases_per_package
            values(new.id, 0)
```

- **EVENT**: Insert in Service Package Table
- **CONDITION** : none
- **ACTION**: Insert of the new service Package in Purchases per Package Table

Total purchases per package - 2

```
CREATE TRIGGER total_purchases_after_insert_order
    AFTER INSERT ON `order`
    FOR EACH ROW
        begin
            IF  new.valid = true
            THEN
                UPDATE purchases_per_package
                    SET totalPurchases = totalPurchases + 1
                    WHERE new.servicePackageId = purchases_per_package.servicePackageId;
            END IF;
        END
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is valid
- **ACTION**: The number of total Purchases in Purchases Per Package Table for the relative package is incremented by 1

Total purchases per package - 3

```
CREATE TRIGGER total_purchases_after_update_order
    AFTER UPDATE ON `order`
    FOR EACH ROW
        begin
            IF  old.valid = false AND new.valid = true
            THEN
                UPDATE purchases_per_package
                SET totalPurchases = totalPurchases + 1
                WHERE new.servicePackageId = purchases_per_package.servicePackageId;
            END IF;
        END
```

- **EVENT**: Update in Order Table
- **CONDITION**: order changes from not valid to valid
- **ACTION**: The number of total Purchases in Purchases Per Package Table for the relative package is incremented by 1

Total purchases per package View

```
CREATE VIEW `purchases_per_package_view` (servicePackageId, servicePackageName, totalPurchases) as
    SELECT s.id, s.name, count(*)
    FROM service_package s, `order` o
    WHERE s.id = o.servicePackageId AND o.valid = true
    GROUP BY s.id;
```

Total purchases per package and val.period - 1

```
CREATE TRIGGER total_purchases_vp_after_insert_sp
    AFTER INSERT ON possible_validity_period
    FOR EACH ROW
        INSERT INTO purchases_per_package_and_vp
            values (new.servicePackageId, new.validityPeriodId, 0);
```

- **EVENT**: Insert in Possible Validity Period Table (Join table for VP associated with a Service Package)
- **CONDITION**: none
- **ACTION**: Insert a row for each VP in Purchases Per Package and VP table

Total purchases per package and val.period - 2

```
CREATE TRIGGER total_purchases_vp_after_insert_order
    AFTER INSERT ON `order`
    FOR EACH ROW
        begin
            IF  new.valid = true
            THEN
                UPDATE purchases_per_package_and_vp
                SET totalPurchases = totalPurchases + 1
                WHERE new.servicePackageId = purchases_per_package_and_vp.servicePackageId
                  AND new.validityPeriodId = purchases_per_package_and_vp.validityPeriodId;
            END IF;
        END
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is valid
- **ACTION**: The number of total Purchases in Purchases Per Package and VP Table for the relative package and VP is incremented by 1

Total purchases per package and val.period - 3

```
CREATE TRIGGER total_purchases_vp_after_update_order
    AFTER UPDATE ON `order`
    FOR EACH ROW
        begin
            IF  old.valid = false AND new.valid = true
            THEN
                UPDATE purchases_per_package_and_vp
                SET totalPurchases = totalPurchases + 1
                WHERE new.servicePackageId = purchases_per_package_and_vp.servicePackageId
                  AND new.validityPeriodId = purchases_per_package_and_vp.validityPeriodId;
            END IF;
        END
```

- **EVENT**: Update in Order Table
- **CONDITION**: order changes from not valid to valid
- **ACTION**: The number of total Purchases in Purchases Per Package and VP Table for the relative package and VP is incremented by 1

Total purchases per package and val.period View

```
CREATE VIEW `purchases_per_package_and_vp_view` (servicePackageName, numberOfMonths, monthlyFee, totalPurchases) as
    SELECT s.name, v.numberOfMonths, v.monthlyFee, count(*)
    FROM service_package s, validity_period v, `order` o
    WHERE s.id = o.servicePackageId AND v.id = o.validityPeriodId AND o.valid = true
    GROUP BY s.id, v.id;
```

Tot. sales per package - opt.products - 1

```
CREATE TRIGGER sales_per_package_init
AFTER INSERT ON service_package
FOR EACH ROW
        INSERT INTO sales_per_package
            values(new.id, 0, 0);
```

- **EVENT**: Insert in Service Package Table
- **CONDITION**: none
- **ACTION**: Insert into Sales Per Package

Tot. sales per package - opt.products - 2

```
CREATE TRIGGER all_insert
AFTER INSERT ON `order`
FOR EACH ROW
    IF new.valid THEN
    UPDATE sales_per_package
        SET revenueWithoutOpProd = revenueWithoutOpProd +
                                   (SELECT monthlyFee * numberOfMonths
                                    FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                    WHERE o.id = new.id),
            revenueWithOpProd = revenueWithOpProd +
                                (SELECT monthlyFee * numberOfMonths
                                 FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                 WHERE o.id = new.id)
        WHERE servicePackageId = new.servicePackageId;
    END IF;
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is valid
- **ACTION**: Update of revenueWithoutOpProd and revenueWithOpProd in Sales Per Package Table.
  The revenue from the sale of the main services is added to both

Tot. sales per package - opt.products - 3

```
CREATE TRIGGER update_revenue_after_opt_prod_choice_insert
AFTER INSERT ON optional_product_choice
FOR EACH ROW
    IF (SELECT o.valid FROM `order` o WHERE o.id = new.orderId) THEN
    UPDATE sales_per_package
        SET revenueWithOpProd = revenueWithOpProd +
                               (SELECT op.monthlyFee * numberOfMonths
                                FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                       JOIN optional_product_choice opc ON o.id = opc.orderId
                                           JOIN optional_product op ON opc.optionalProductId = op.id
                                WHERE o.id = new.orderId AND op.id = new.optionalProductId)
        WHERE servicePackageId = (SELECT o.servicePackageId FROM `order` o WHERE o.id = new.orderId);
    END IF;
```

- **EVENT**: Insert in Optional Product Choice Table (Join table for optional products added to an order)
- **CONDITION**: order is valid
- **ACTION**: Update of revenueWithOpProd in Sales Per Package Table adding the revenue from the sale of the optional products

Tot. sales per package - opt.products - 4

```
CREATE TRIGGER revenue_update
AFTER UPDATE ON `order`
FOR EACH ROW
    IF old.valid = false AND new.valid = true THEN
        UPDATE sales_per_package
        SET revenueWithoutOpProd = revenueWithoutOpProd +
                                   (SELECT monthlyFee * numberOfMonths
                                   FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                   WHERE o.id = new.id),
            revenueWithOpProd = revenueWithOpProd +
                               (SELECT monthlyFee * numberOfMonths
                                FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                WHERE o.id = new.id) +
                               (SELECT COALESCE(SUM(op.monthlyFee * numberOfMonths), 0)
                                FROM `order` o JOIN validity_period v ON o.validityPeriodId = v.id
                                               JOIN optional_product_choice opc ON o.id = opc.orderId
                                               JOIN optional_product op ON opc.optionalProductId = op.id
                                WHERE o.id = new.id)
        WHERE servicePackageId = new.servicePackageId;
    END IF;
```

- **EVENT**: Update in Order Table
- **CONDITION**: order become valid
- **ACTION**: Update of revenueWithoutOpProd and revenueWithOpProd in Sales Per Package Table.
  The revenue from the sale of the main services is added to both and then the revenue for the op. products is added to the second for each op. product related to that order.

Tot. sales per package - opt.products View

```
CREATE VIEW `sales_per_package_view` (servicePackageId, revenueWithOpProd, revenueWithoutOpProd) as
SELECT
       o1.servicePackageId,
       COALESCE(SUM(vp1.monthlyFee * vp1.numberOfMonths), 0) + (SELECT COALESCE(SUM(op.monthlyFee * numberOfMonths), 0)
                                                                FROM `order` o2 JOIN validity_period v2 ON o2.validityPeriodId = v2.id
                                                                             JOIN optional_product_choice opc ON o2.id = opc.orderId
                                                                            JOIN optional_product op ON opc.optionalProductId = op.id
                                                                WHERE o2.valid = true AND o2.servicePackageId = o1.servicePackageId),
       COALESCE(SUM(vp1.monthlyFee * vp1.numberOfMonths), 0)
FROM `order` o1 JOIN validity_period vp1 on o1.validityPeriodId = vp1.id
WHERE o1.valid = true
GROUP BY o1.servicePackageId;
```

Avg of opt. prod. sold with each package - 1

```
CREATE TRIGGER avg_opt_prod_init
AFTER INSERT ON service_package
FOR EACH ROW
        INSERT INTO average_number_opt_products_per_package
            values(new.id, 0, 0, 0);
```

- **EVENTO**: Insert in Service Package Table
- **CONDIZIONE**: none
- **AZIONE**: Insert in Average Number Of Opt Products Per Package Table

Avg of opt. prod. sold with each package - 2

```
CREATE TRIGGER opt_prod_insert
AFTER INSERT ON optional_product_choice
FOR EACH ROW
    IF (SELECT o.valid FROM `order` o WHERE o.id = new.orderId) THEN
    UPDATE average_number_opt_products_per_package
        SET optProductsSold = optProductsSold + 1,
            `avg` = (optProductsSold / totOrders)
        WHERE servicePackageId = (SELECT servicePackageId
                                  FROM `order` O
                                  WHERE O.id = new.orderId);
    END IF;
```

- **EVENT**: Insert in Optional Product Choice (Join table for optional products added to an order)
- **CONDITION**: order is valid
- **ACTION**: Update of OptProductSold (+1 for each opt product) and avg in Average Number Of Opt Products Per Package Table

Avg of opt. prod. sold with each package - 3

```
CREATE TRIGGER avg_opt_prod_insert
AFTER INSERT ON `order`
FOR EACH ROW
    IF new.valid THEN
    UPDATE average_number_opt_products_per_package
        SET totOrders = totOrders + 1,
            `avg` = (optProductsSold / totOrders)
        WHERE servicePackageId = new.servicePackageId;
    END IF;
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is valid
- **ACTION**: Update of totOrders (+1) and avg in Average Number Of Opt Products Per Package Table

Avg of opt. prod. sold with each package - 4

```
CREATE TRIGGER opt_prod_update
AFTER UPDATE ON `order`
FOR EACH ROW
    IF old.valid = false AND new.valid = true THEN
        UPDATE average_number_opt_products_per_package
        SET totOrders = totOrders + 1,
            optProductsSold = optProductsSold +
                              (SELECT COUNT(opc.optionalProductId)
                              FROM optional_product_choice opc
                              WHERE opc.orderId = new.id),
            `avg` = (optProductsSold / totOrders)
        WHERE servicePackageId = new.servicePackageId;
    END IF;
```

- **EVENT**: Update in Order Table
- **CONDITION**: order become valid
- **ACTION**: Update of totOrders (+1), optProductSold (added the number of related optional products) and avg in Average Number Of Opt Products Per Package Table

Avg of opt. prod. sold with each package View

```
CREATE VIEW `average_number_opt_products_per_package` as
SELECT servicePackageId, COUNT(opc.optionalProductId, opc.orderId), COUNT( distinct o.id),
       (COUNT(opc.optionalProductId, opc.orderId)/COUNT( distinct o.id))
FROM `order` o LEFT JOIN optional_product_choice opc on o.id = opc.orderId
WHERE o.valid
GROUP BY servicePackageId;
```

List of insolvent users - 1

```
CREATE TRIGGER add_insolvent_user
    AFTER INSERT ON `order`
    FOR EACH ROW
begin
    IF
        new.valid = false AND
        NOT EXISTS(
            SELECT *
            FROM insolvent_users
            WHERE id=new.userId
            )
    THEN
        INSERT INTO insolvent_users
            (SELECT u.id, u.username, u.email
             FROM user u
             WHERE u.id = new.userId);

    END IF;
end;
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is not valid and user is not present in Insolvent Users Table
- **ACTION**: Insert in Insolvent Users Table

List of insolvent users - 2

```
CREATE TRIGGER remove_insolvent_user
    AFTER UPDATE ON `order`
    FOR EACH ROW
begin
    IF  old.valid = false AND
        new.valid = true AND
        NOT EXISTS(
            SELECT *
            FROM `order` o
            WHERE o.valid = false AND
                    new.userId = o.userId
            )
    THEN
        DELETE FROM insolvent_users
            WHERE insolvent_users.id = new.userId;
    END IF;
END
```

- **EVENT**: Update in Order Table
- **CONDITION**: order become valid and not exists an invalid order for that user
- **ACTION**: Delete of that user from Insolvent Users Table

List of insolvent users View

```
CREATE VIEW `insolvent_users_view` as
SELECT distinct u.username, u.email
FROM `order` o, user u
WHERE u.id = o.userId AND o.valid = false;
```

Suspended orders - 1

```
CREATE TRIGGER add_suspended_order
    AFTER INSERT ON `order`
    FOR EACH ROW
begin
    IF  new.valid = false
    THEN
        INSERT INTO suspended_orders
            value (new.id);

    END IF;
end;
```

- **EVENT**: Insert in Order Table
- **CONDITION**: order is not valid
- **ACTION**: Insert of order in Suspended Orders Table

Suspended orders - 2

```
CREATE TRIGGER remove_suspended_order
    AFTER UPDATE ON `order`
    FOR EACH ROW
begin
    IF  old.valid = false AND
        new.valid = true
    THEN
        DELETE FROM suspended_orders
        WHERE suspended_orders.id = new.id;
    END IF;
END
```

- **EVENT**: Update in Order Table
- **CONDITION**: order become valid
- **ACTION**: Delete of order from Suspended Orders Table

Suspended orders View

```
CREATE VIEW `suspended_orders_view` as
    SELECT o.id, u.username, s.name, v.monthlyFee, v.numberOfMonths, o.creation, o.totalPrice
    FROM `order` o, user u, service_package s, validity_period v
    WHERE u.id = o.userId AND o.validityPeriodId = v.id AND o.servicePackageId = s.id AND o.valid = false;
```

Alert

```
CREATE TRIGGER blacklist_population
AFTER INSERT ON `order`
FOR EACH ROW
BEGIN
    IF new.valid = false AND
        (SELECT count(*)
        FROM `order` o
        WHERE o.valid = false AND
            new.userId = o.userId) >= 3
    THEN
        IF exists
            (SELECT *
            FROM blacklist
            WHERE new.userId = blacklist.userId)
        THEN
            UPDATE blacklist
            SET blacklist.lastRejection = new.creation, blacklist.amount = new.totalPrice
            WHERE blacklist.userId = new.userId;
        ELSE
            INSERT INTO blacklist
                (SELECT u.id, u.username, u.email, new.creation, new.totalPrice
                FROM user u
                WHERE new.userId = u.id);
        END IF;
    END IF;
END
```

- **EVENT**: Insert in Order Table
- **CONDITION**: number of rejected Order ≥ 3
- **ACTION**:
  if the user isn't already in the Blacklist Table
  -> Insert of user and last order data in Blacklist Table
  if the user is present in the Blacklist Table
  -> update the alert with the data of the new rejected order

Best seller optional product - 1

```
CREATE TRIGGER best_seller_opt_prod_init
AFTER INSERT ON optional_product
FOR EACH ROW
        INSERT INTO best_seller_opt_prod
            values(new.id, 0);
```

- **EVENT**: Insert in Optional Product Table
- **CONDITION**: none
- **ACTION**: Insert in Best Seller Opt Product Table

Best seller optional product - 2

```
CREATE TRIGGER revenue_opt_prod_insert
AFTER INSERT ON optional_product_choice
FOR EACH ROW
    IF (SELECT o.valid FROM `order` o WHERE o.id = new.orderId) THEN
    UPDATE best_seller_opt_prod
        SET revenue = revenue + (SELECT op.monthlyFee * numberOfMonths
                                 FROM `order` o JOIN validity_period v ON  o.validityPeriodId = v.id
                                     JOIN optional_product_choice opc ON o.id = opc.orderId
                                     JOIN optional_product op ON opc.optionalProductId = op.id
                                 WHERE o.id = new.orderId AND op.id = new.optionalProductId)
        WHERE optionalProductId = new.optionalProductId;
    END IF;
```

- **EVENT**: Insert in Optional Product Choice
- **CONDITION**: order is valid
- **ACTION**: Update of revenue in Best Seller Optional Product Table adding the new revenue generated by the sale of each OP

Best seller optional product - 3

```
CREATE TRIGGER revenue_opt_prod_update
    AFTER UPDATE ON `order`
    FOR EACH ROW
    IF old.valid = false AND new.valid = true THEN
        UPDATE best_seller_opt_prod bsop
        SET revenue = revenue +
                    (SELECT op.monthlyFee * numberOfMonths
                    FROM `order` o JOIN validity_period v ON  o.validityPeriodId = v.id
                         JOIN optional_product_choice opc ON o.id = opc.orderId
                         JOIN optional_product op ON opc.optionalProductId = op.id
                    WHERE o.id = new.id AND op.id = bsop.optionalProductId)
        WHERE optionalProductId IN (SELECT op2.optionalProductId
                                FROM optional_product_choice op2
                                WHERE op2.orderId = new.id);
    END IF;
```

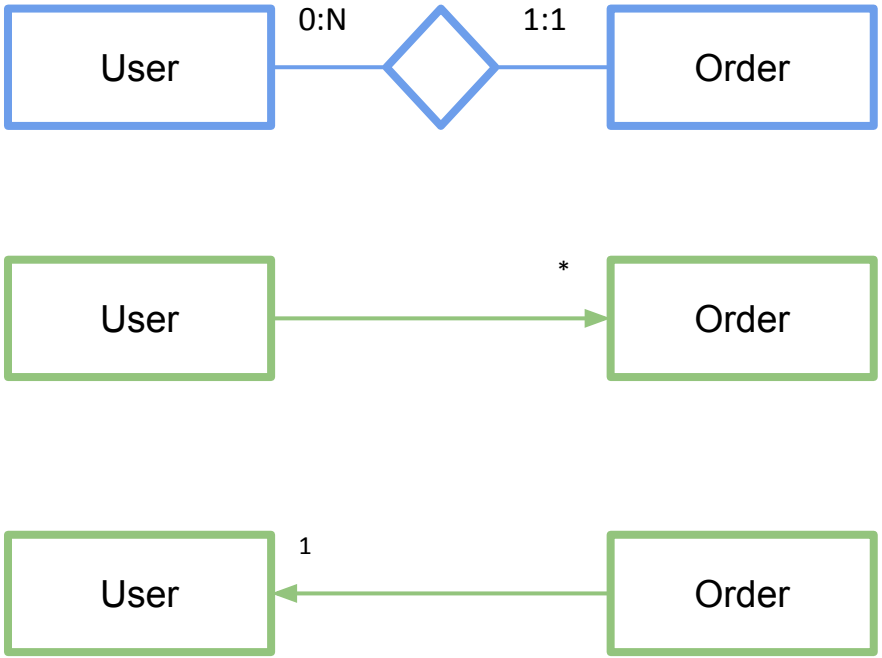
- **EVENT**: Update in Order Table
- **CONDITION**: order become valid
- **ACTION**: Update of revenue in Best Seller Optional Product Table adding the revenue coming from that order for all the OP related

Best seller optional product View

```
CREATE VIEW `best_seller_opt_prod` as
SELECT optionalProductId, COALESCE(SUM(op.monthlyFee * vp.numberOfMonths),0) as revenue
FROM optional_product op JOIN optional_product_choice opc on op.id = opc.optionalProductId
    JOIN `order` o on opc.orderId = o.id JOIN validity_period vp on vp.id = o.validityPeriodId
WHERE o.valid
GROUP BY optionalProductId
ORDER BY revenue desc
LIMIT 1;
```

ORM DESIGN

Relationship



User ➔ Order

@OneToMany

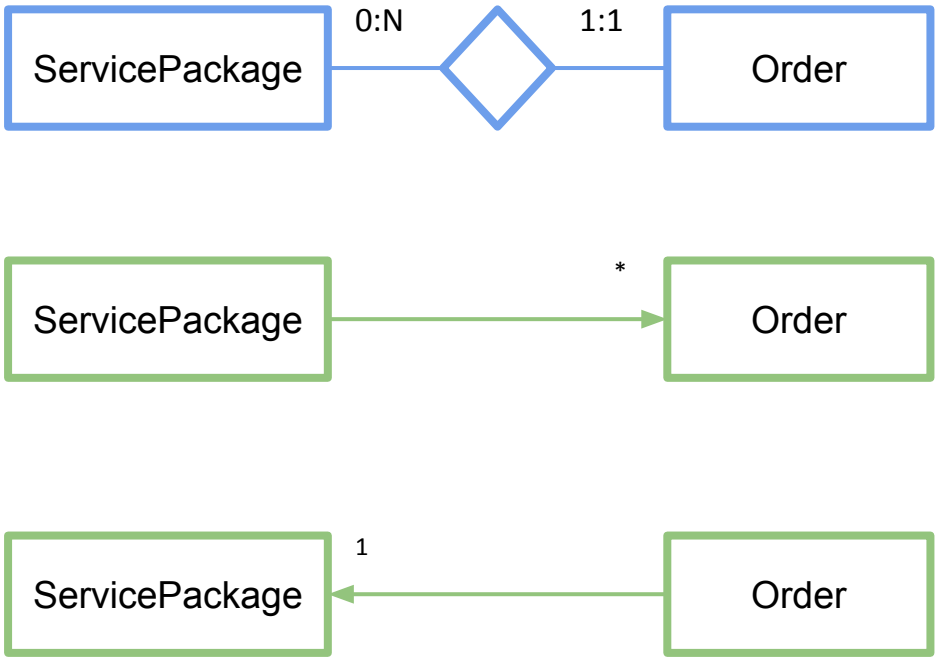
- *Eager Fetch*
- *Cascade in all cases*
- *Orphan removal*

Order ➔ User

@ManyToOne

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

Relationship



ServicePackage ➔ Order

@OneToMany

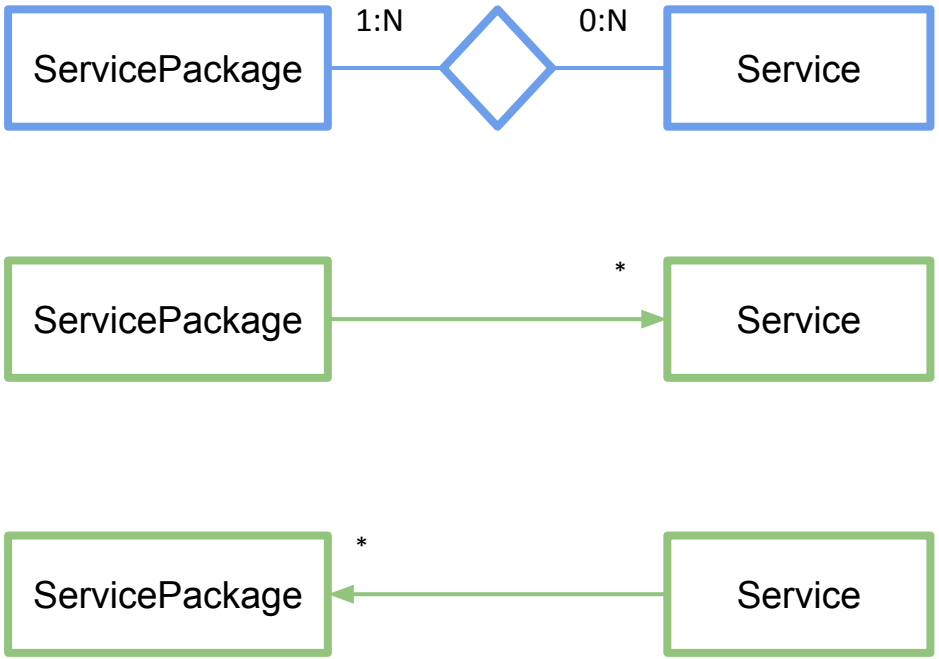
- *Lazy Fetch*
- *Cascade in all cases*
- *Orphan removal*

Order ➔ ServicePackage

@ManyToOne

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

Relationship



ServicePackage ➔ Service

@ManyToMany

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

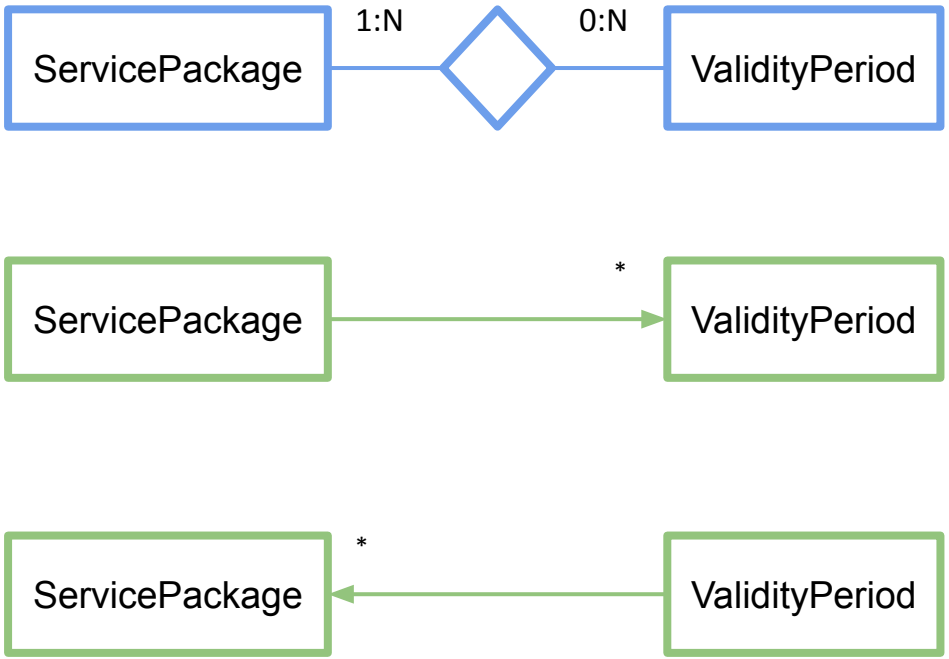
Service ➔ ServicePackage

@ManyToMany

(not necessary but mapped for consistency)

- *Lazy Fetch*
- *Cascade in all cases*

Relationship



ServicePackage ➔ ValidityPeriod

@ManyToMany

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

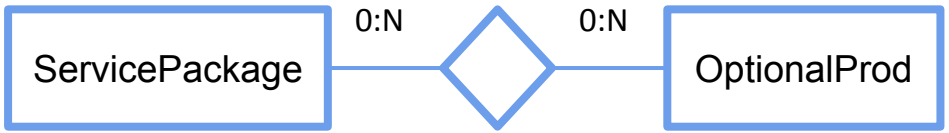
ValidityPeriod ➔ ServicePackage

@ManyToMany

(not necessary but mapped for consistency)

- *Lazy Fetch*
- *Cascade in all cases*

Relationship



ServicePackage ➔ OptionalProd

@ManyToMany

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

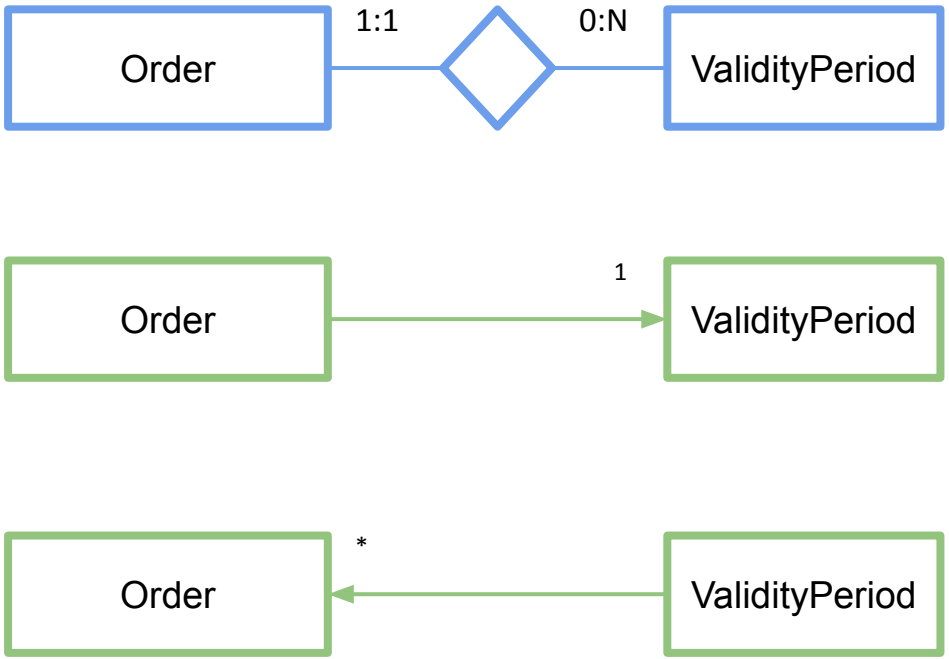
OptionalProd ➔ ServicePackage

@ManyToMany

(not necessary but mapped for consistency)

- *Lazy Fetch*
- *Cascade in all cases apart from the delete*

Relationship



Order ➔ ValidityPeriod

@ManyToOne

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

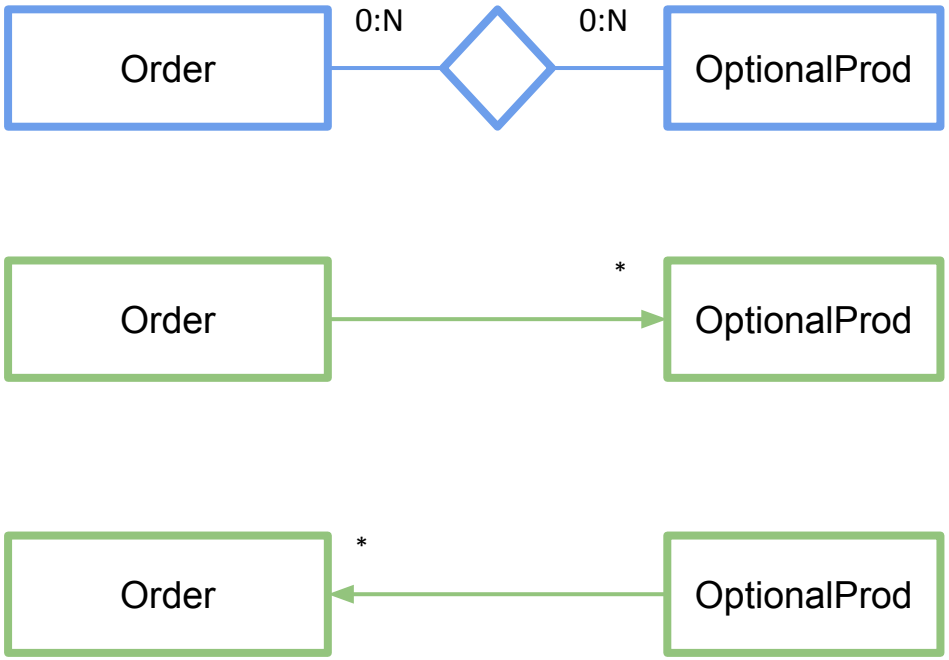
ValidityPeriod ➔ Order

@OneToMany

(not necessary but mapped for consistency)

- *Lazy Fetch*
- *Cascade in all cases*
- *Orphan removal*

Relationship



Order ➔ OptionalProduct

@ManyToMany

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

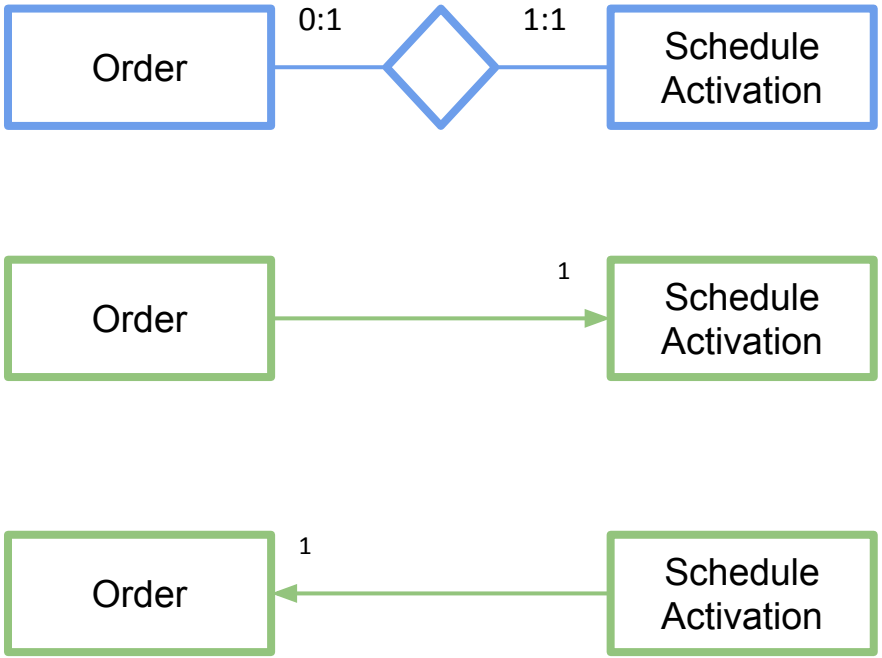
OptionalProduct ➔ Order

@ManyToMany

(not necessary but mapped for consistency)

- *Lazy Fetch*
- *Cascade in all cases apart from the delete*

Relationship



Order ➔ ScheduleActivation

@OneToOne

- *Eager Fetch*
- *Cascade in all cases*

ScheduleActivation ➔ Order

@OneToOne

- *Eager Fetch*
- *Cascade in all cases apart from the delete*

ENTITY CODE

User Entity

```
@Entity
@NamedQueries(
        {
                @NamedQuery(
                        name = "User.findByUsername",
                        query = "SELECT u " +
                                "FROM UserEntity u " +
                                "WHERE u.username = :username"
                ),
                @NamedQuery(
                        name = "User.findByEmail",
                        query = "SELECT u " +
                                "FROM UserEntity u " +
                                "WHERE u.email = :email"
                )
        }
)
```

User Entity

```java
@Table(name = "user", schema = "db2_database")
public class UserEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @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;

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

Employee Entity

```
@Entity
@NamedQueries(
        {
                @NamedQuery(
                        name = "Employee.findByUsername",
                        query = "SELECT e " +
                                "FROM EmployeeEntity e " +
                                "WHERE e.username = :username"
                )

        }
)
```

```
@Table(name = "employee", schema = "db2_database")
public class EmployeeEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @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;
```

Validity Period Entity

```
public class ValidityPeriodEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

    @Column(name = "numberOfMonths", nullable=false)

    @ManyToMany(mappedBy = "possibleValidityPeriods", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<ServicePackageEntity> servicePackages;

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

Service Entity

```
@Entity
@Table(name = "service", schema = "db2_database")
@NamedQueries({
        @NamedQuery(name = "Service.findAll", query =
                "select s from ServiceEntity s " +
                        "order by s.type"),
        @NamedQuery(name = "Service.findById", query =
                "select s from ServiceEntity s " +
                        "where s.id = :id")
})
```

Service Entity

```
public class ServiceEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

    @Column(name = "type", nullable=false)
    @Enumerated(EnumType.STRING)
    private ServiceType type;
```

```
@Column(name = "numberOfSms", nullable=true)
private int numberOfSms;

@Column(name = "numberOfMinutes", nullable=true)
private int numberOfMinutes;

@Column(name = "extraMinutesFee", nullable=true)
private float extraMinutesFee;

@Column(name = "extraSmsFee", nullable=true)
private float extraSmsFee;

@Column(name = "numberOfGb", nullable=true)
private int numberOfGb;

@Column(name = "extraGbFee", nullable=true)
private float extraGbFee;
```

```
    @ManyToMany(mappedBy = "services", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<ServicePackageEntity> servicePackages;
```

Validity Period Entity

```
@Entity
@Table(name = "validity_period", schema = "db2_database")
@NamedQueries({
        @NamedQuery(name = "ValidityPeriod.findAll", query =
                "select v from ValidityPeriodEntity v " +
                        "order by v.numberOfMonths"),
        @NamedQuery(name = "ValidityPeriod.findByPackage", query =
                "select v from ValidityPeriodEntity v join v.servicePackages sp " +
                        "where sp.id = :packId"),
        @NamedQuery(name = "ValidityPeriod.findById", query =
                "select v from ValidityPeriodEntity v " +
                        "where v.id = :id")
})
```

Optional Product Entity

```
@Entity
@Table(name = "optional_product", schema = "db2_database")
@NamedQueries({
        @NamedQuery(name = "OptionalProduct.findAll", query = "select o from OptionalProductEntity o"),
        @NamedQuery(name = "OptionalProduct.findByName", query =
                "select o from OptionalProductEntity o " +
                        "where o.name = :name"),
        @NamedQuery(name = "OptionalProduct.findByPackage", query =
                "select op from ServicePackageEntity sp join sp.possibleOptionalProducts op " +
                        "where sp.id = :packId"),
        @NamedQuery(name = "OptionalProduct.findById", query =
                "select o from OptionalProductEntity o " +
                        "where o.id = :id")
})
```

Optional Product Entity

```
public class OptionalProductEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

    @Column(name = "monthlyFee", nullable=false)
    private float monthlyFee;
```

Optional Product Entity

```
@ManyToMany(mappedBy = "possibleOptionalProducts", fetch = FetchType.LAZY,
        cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
private List<ServicePackageEntity> relatedServicePackages;

@ManyToMany(mappedBy = "optionalProducts", fetch = FetchType.LAZY,
        cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
private List<OrderEntity> relatedOrders;
```

Order Entity

```
@Entity
@Table(name = "order", schema = "db2_database")
@NamedQueries({
        @NamedQuery(name = "Order.findRejected", query = "select o from OrderEntity o " +
                "where o.valid = false and o.user.id = :userId"),
        @NamedQuery(name = "Order.findById", query = "select o from OrderEntity o where o.id = :orderId"),
        @NamedQuery(name = "Order.findAmong", query = "select o from OrderEntity o where o.id in :id")
})
```

Order Entity

```java
public class OrderEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

    @Column(name = "startDate", nullable=false)
    private Date startDate;

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

    @Column(name = "totalPrice", nullable=false)
    private float totalPrice;
```

Order Entity

```
@OneToOne(mappedBy = "order")
private ScheduleActivationEntity scheduleActivation;

@ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn (name = "userId")
private UserEntity user;

@ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn (name = "validityPeriodId")
private ValidityPeriodEntity validityPeriod;

@ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn (name = "servicePackageId")
private ServicePackageEntity servicePackage;

@ManyToMany(fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinTable (name="optional_product_choice",
        joinColumns = @JoinColumn(name="orderId"),
        inverseJoinColumns= @JoinColumn (name="optionalProductId"))
private List<OptionalProductEntity> optionalProducts;
```

Blacklist Entity

```
@Entity
@Table(name = "blacklist", schema = "db2_database")
@NamedQueries({
        @NamedQuery(name = "Blacklist.findAll",
                query = "select b from BlacklistEntity b")
})
```

```
public class BlacklistEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "userId", nullable=false)
    private int userId;

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

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

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

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

Schedule Activation Entity

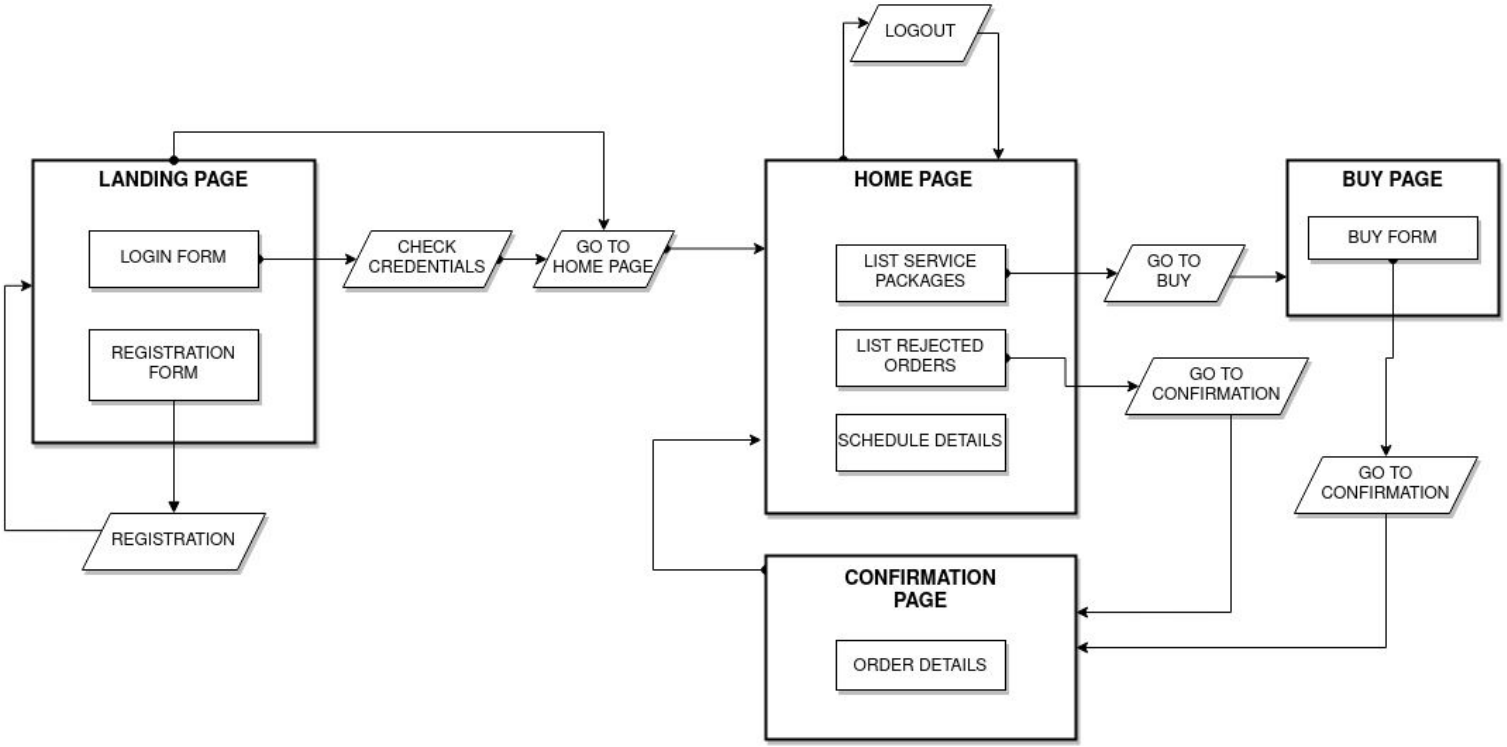
```
@Entity
@Table(name = "schedule_activation", schema = "db2_database")
@NamedQueries({ @NamedQuery(name = "Schedule.findValid",
        query = "select sa from ScheduleActivationEntity sa " +
                "where sa.order.user.id = :userId")})
```

```
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn (name = "orderId", referencedColumnName = "id")
    private OrderEntity order;
```

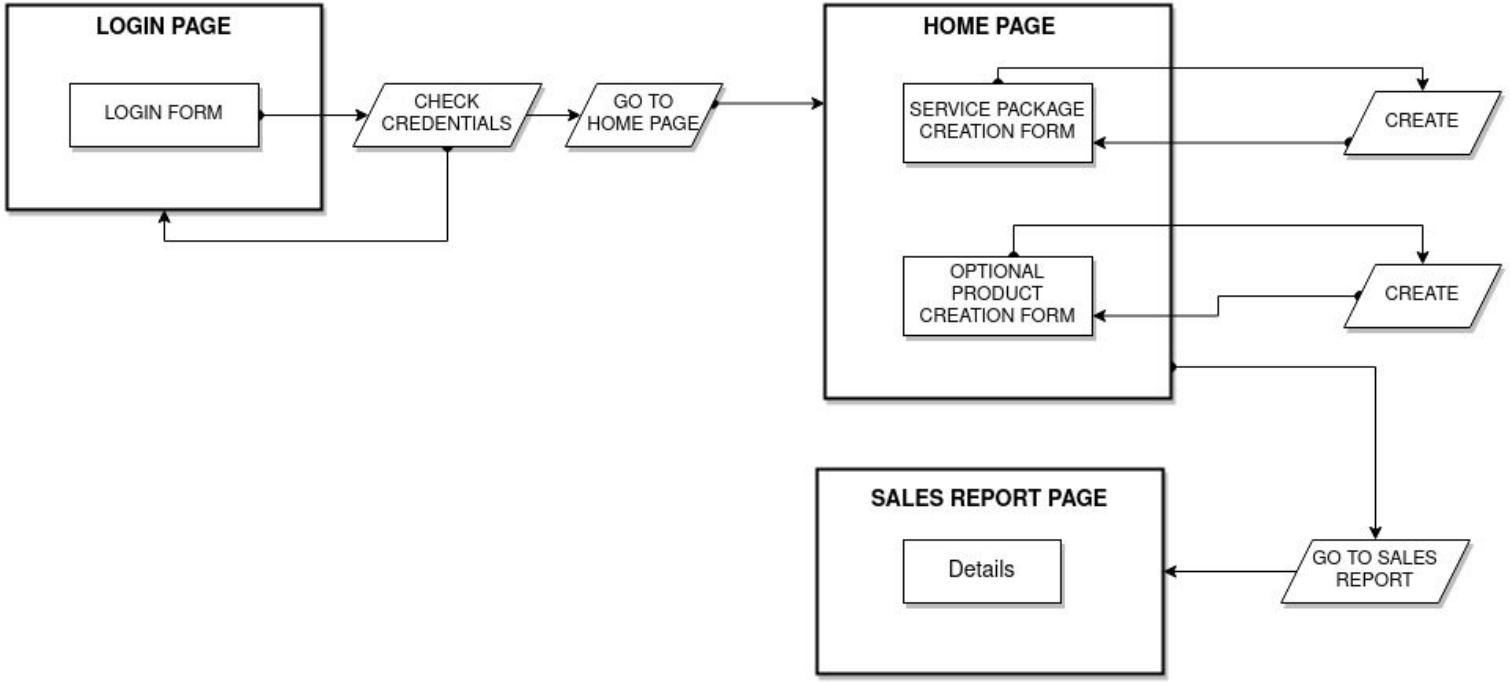
```
public class ScheduleActivationEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

    @Column(name = "startDate", nullable=false)
    private Date start;

    @Column(name = "endDate", nullable=false)
    private Date end;
```

INTERACTION DIAGRAM

User


LOGOUT
LANDING PAGE
LOGIN FORM
REGISTRATION FORM
REGISTRATION
CHECK CREDENTIALS
GO TO HOME PAGE
HOME PAGE
LIST SERVICE PACKAGES
LIST REJECTED ORDERS
SCHEDULE DETAILS
GO TO BUY
GO TO CONFIRMATION
BUY PAGE
BUY FORM
GO TO CONFIRMATION
CONFIRMATION PAGE
ORDER DETAILS

Employee


LOGIN PAGE
LOGIN FORM
CHECK CREDENTIALS
GO TO HOME PAGE
HOME PAGE
SERVICE PACKAGE CREATION FORM
CREATE
OPTIONAL PRODUCT CREATION FORM
CREATE
SALES REPORT PAGE
Details
GO TO SALES REPORT

COMPONENTS

Client tier

- **USER**
  - HTML page generated from UserPages/landing-page.jsp
  - HTML page generated from UserPages/home-page.jsp
  - HTML page generated from UserPages/buy-page.jsp
  - HTML page generated from UserPages/confirmation-page.jsp

- **EMPLOYEE**
  - HTML page generated from EmployeePages/login-page.jsp
  - HTML page generated from EmployeePages/home-page.jsp
  - HTML page generated from EmployeePages/sales-report-page.jsp

Web tier

USER

- LoginServlet
- RegistrationServlet
- UserHomePageServlet
- BuyPageServlet
- OrderConfirmationServlet
- RetrieveOrderServlet

EMPLOYEE

- EmployeeLoginServlet
- EmployeeHomePageServlet
- OptionalProductCreationServlet
- ServicePackageCreationServlet
- SalesReportServlet

Business tier (stateless EJBs)

- **UserService**
  - createUser(username, password, email)
  - findUserByUsername(username)
  - findUserByEmail(email)
  - findInsolventUsers()
  - checkCredentials(username, password)
- **EmployeeService**
  - findEmployeeByUsername(username)
  - checkCredentials(name, password)
- **ServiceService**
  - findServiceById(id)
  - findAllServices()
- **ServicePackageService**
  - createServicePackage(name, services, possibleValidityPeriods, possibleOptionalProducts)
  - findServicePackageById(id)
  - findServicePackageByName(name)
  - findAllServicePackages()

Business tier (stateless EJBs)

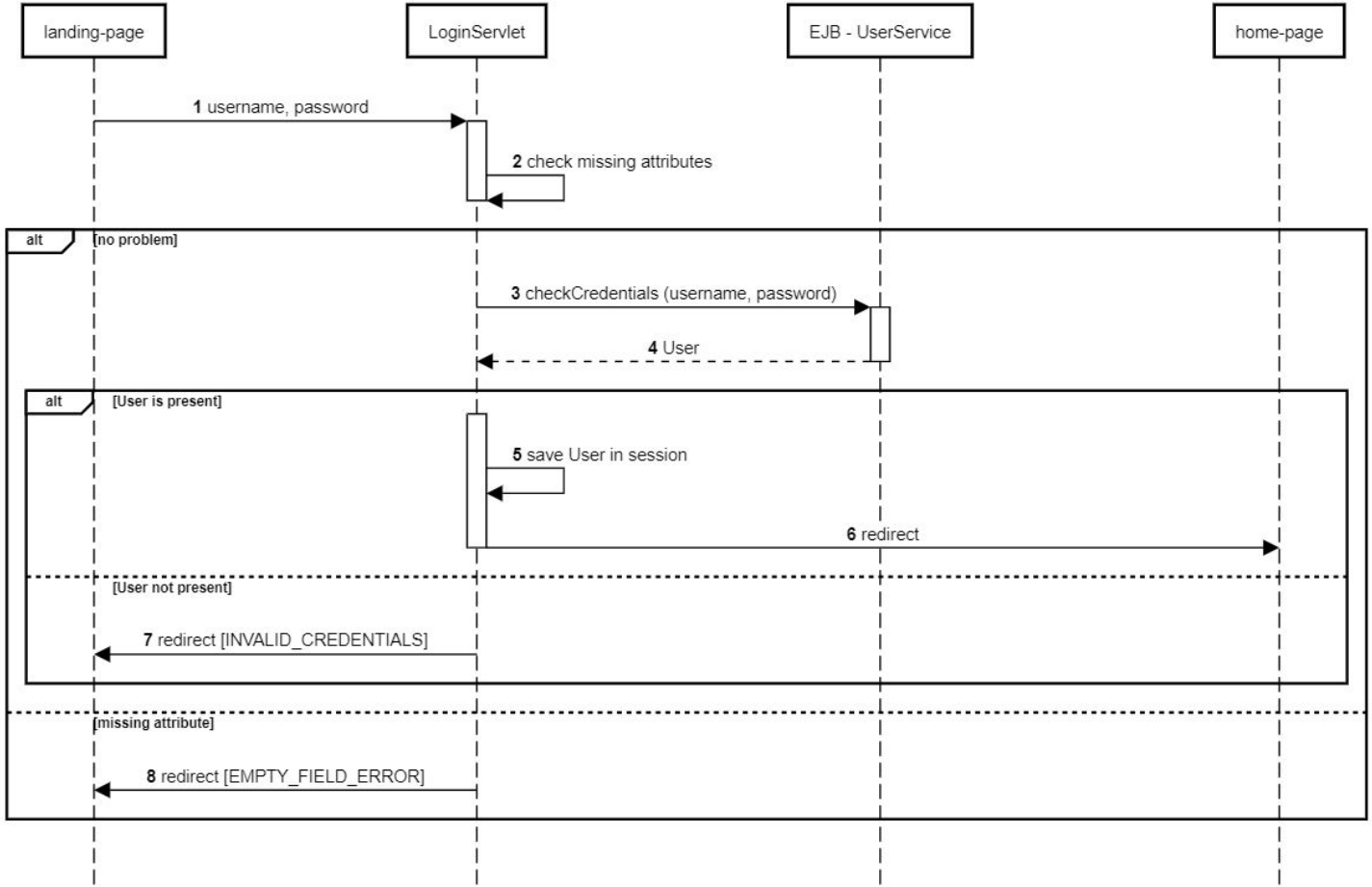
- **ValidityPeriodService**
  - findValidityPeriodById(id)
  - findValidityPeriodsOfPackage(chosen)
  - findAllValidityPeriods()
- **OptionalProductService**
  - createOptionalProduct(name, montlyFee)
  - findOptionalProductById(id)
  - findOptionalProductByName(name)
  - findOptionalProductsOfPackage(chosen)
  - findAllOptionalProducts()
- **OrderService**
  - createOrder(order)
  - findOrderById(id)
  - findRejectedOrders(userId)
  - findSuspendedOrders()
- **BlackListService**
  - findAllAlerts()

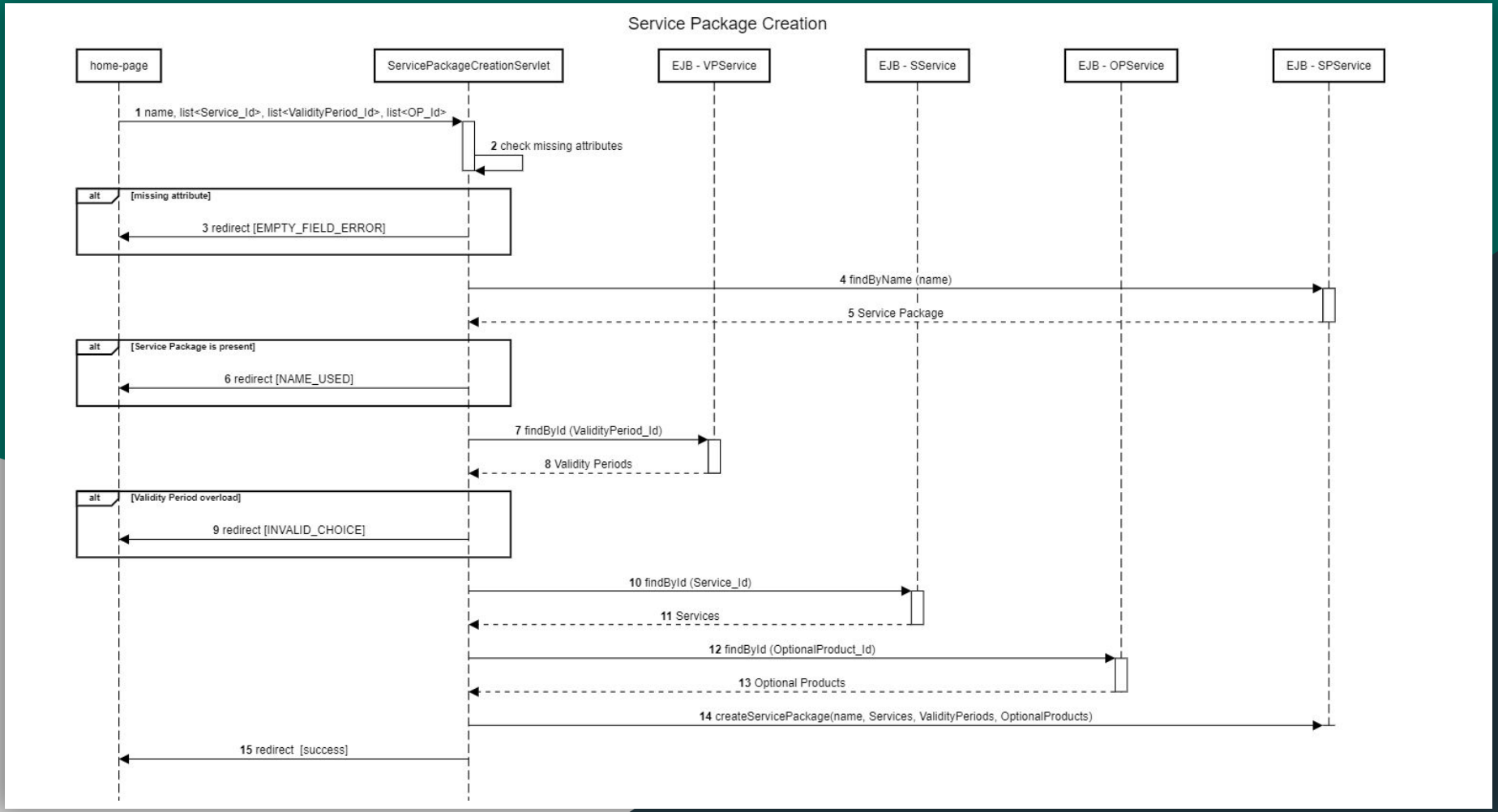
Business tier (stateless EJBs)

- **ViewService**
  - totalPurchasesPerPackage()
  - totalPurchasesPerPackageAndVP()
  - avgOptProdPerPackage()
  - bestSellerOptProduct()
  - salesPerPackage()
- **ScheduleActivationService**
  - createScheduleActivation(scheduleActivation)
  - findValidOrders(userId)

UML SEQUENCE DIAGRAMS

Login


landing-page
LoginServlet
EJB - UserService
home-page
1 username, password
2 check missing attributes
alt
[no problem]
3 checkCredentials (username, password)
4 User
alt
[User is present]
5 save User in session
6 redirect
[User not present]
7 redirect [INVALID_CREDENTIALS]
[missing attribute]
8 redirect [EMPTY_FIELD_ERROR]


Service Package Creation
home-page
ServicePackageCreationServlet
EJB - VPService
EJB - SService
EJB - OPService
EJB - SPService
1 name, list<Service_Id>, list<ValidityPeriod_Id>, list<OP_Id>
2 check missing attributes
alt
[missing attribute]
3 redirect [EMPTY_FIELD_ERROR]
4 findByName (name)
5 Service Package
alt
[Service Package is present]
6 redirect [NAME_USED]
7 findById (ValidityPeriod_Id)
8 Validity Periods
alt
[Validity Period overload]
9 redirect [INVALID_CHOICE]
10 findById (Service_Id)
11 Services
12 findById (OptionalProduct_Id)
13 Optional Products
14 createServicePackage(name, Services, ValidityPeriods, OptionalProducts)
15 redirect [success]