POLITECNICO
MILANO 1863

DATA BASES 2 Project

Natalia Bagnoli (994176)
Filippo Caliò (994184)

Academic Year: 2021-22
Prof. Sara Comai

INDEX

- SPECIFICATIONS
- ENTITY RELATIONSHIP
- RELATIONAL MODEL
- TRIGGERS
- ORM design
- ENTITIES
- COMPONENTS
- UML sequence diagram

SPECIFICATIONS

TELCO SERVICE APPLICATIONS

A telco company offers pre-paid online services to web users. Two client applications using the same database need to be developed.

CONSUMER APPLICATION

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username, 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 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.

SPECIFICATIONS

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.

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.

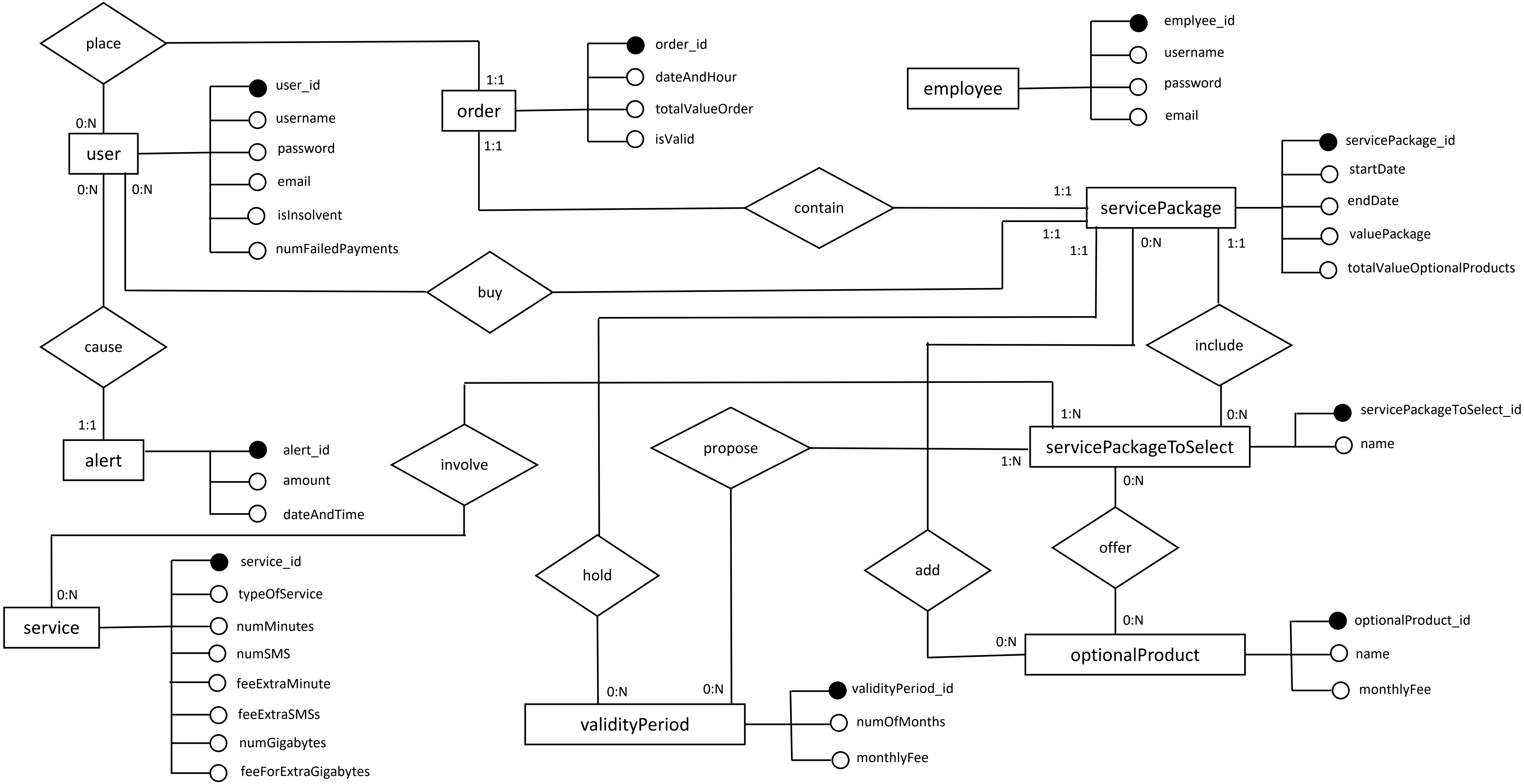
A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- 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.

IMPLEMENTATION NOTES

- The call to the external service must be simulated with a function that returns true or falsepseudo-randomly. For testing purposes, the demonstration should be able to show at least one case in which the service call fails and one case in which the service call succeeds.
- The aggregate data of the sales report must be computed by triggers that populate materialized view tables. The documentation must describe the SQL code of the view that would compute the aggregate data, the logical schema of the materialized view table(s) that store the aggregate data and the triggers that populate the content of the materialized view table(s).

ENTITY RELATIONSHIP



RELATIONAL MODEL

```
create table user
(
    user_id           int auto_increment primary key,
    username          varchar(64)       not null,
    password          varchar(64)       not null,
    email             varchar(64)       not null,
    isInsolvent       tinyint default 0  NOT NULL,
    numFailedPayments int     default 0 NOT NULL,
    constraint email unique (email),
    constraint user_id unique (user_id),
    constraint username unique (username)
);

create table employee
(
    employee_id int auto_increment primary key,
    username    varchar(64) not null,
    password    varchar(64) not null,
    email       varchar(64) not null,
    constraint email unique (email),
    constraint username unique (username)
);
```

RELATIONAL MODEL

```
create table alert
(
    alert_id    int auto_increment primary key,
    amount      int      not null,
    dateAndTime datetime not null,
    userOwner   int      not null,
    constraint alert_fk0 foreign key (userOwner) references user (user_id)
);

create table order
(
    order_id                 int auto_increment primary key,
    dateAndHour              datetime not null,
    userOwner                int      not null,
    servicePackageAssociated int      not null,
    totalValueOrder          float    not null,
    isValid                  tinyint  null,
    constraint order_fk0 foreign key (userOwner) references user (user_id),
    constraint order_fk1 foreign key (servicePackageAssociated) references servicepackage
(servicePackage_id)
);
```

RELATIONAL MODEL

```
create table servicepackage
(
    servicePackage_id           int auto_increment primary key,
    packageSelected             int            not null,
    validityPeriod              int            not null,
    userOwner                   int            not null,
    startDate                   date           not null,
    endDate                     date           not null,
    valuePackage                float          not null,
    totalValueOptionalProducts float default 0 not null,
    constraint servicePackage_fk foreign key (packageSelected) references servicepackagetoselect
(servicePackageToSelect_id),
    constraint servicePackage_fk1 foreign key (validityPeriod) references validityperiod
(validityPeriod_Id),
    constraint servicePackage_fk2 foreign key (userOwner) references user (user_id)
);

create table validityperiod
(
    validityPeriod_Id int auto_increment primary key,
    numOfMonths       int not null,
    monthlyFee        int not null
);
```

RELATIONAL MODEL

```
create table optionalproduct
(
    optionalProduct_Id int auto_increment primary key,
    name               varchar(64) not null,
    monthlyFee         float       not null,
    constraint name unique (name)
);

create table servicepackagetoselect
(
    servicePackageToSelect_id int auto_increment primary key,
    name                      varchar(64) not null,
    constraint name unique (name)
);
create table service
(
    service_id             int auto_increment primary key,
    typeOfService          varchar(64) not null,
    numMinutes             int         null,
    numSMS                 int         null,
    feeExtraMinute         float       null,
    feeExtraSMSs           float       null,
    numGigabytes           int         null,
    feeForExtraGigabytes float         null
);
```

MOTIVATIONS

A particular structure is certainly the one that links the ServicePackageToSelect to the ServicePackage. We have decided to make this distinction because in this way it is possible to distinguish the packages that the employee creates (with the possible optional products that can be selected and the validity periods) from those selected and subsequently purchased by the user that contain defined optional products and a single validity period.

By designing the schema in this way, each ServicePackage is associated with a single ServicePackageToSelect created by employee and each ServicePackageToSelect can be selected by multiple users and therefore can be contained in multiple ServicePackages.

TRIGGERS – first query

```
CREATE TRIGGER addPurchaseToPackage
    AFTER INSERT ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        UPDATE totalpurchasesperpackage SET totalPurchases = totalPurchases + 1
        WHERE package_id IN (SELECT s.packageSelected
                             FROM servicepackage s
                             WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
END;


CREATE TRIGGER updatePurchaseToPackage
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        UPDATE totalpurchasesperpackage SET totalPurchases = totalPurchases + 1
        WHERE package_id IN (SELECT s.packageSelected
                             FROM servicepackage s
                             WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
END;
```

TRIGGERS – first query

```
CREATE TRIGGER createServicePackageToSelect1
    AFTER INSERT ON servicepackagetoselect FOR EACH ROW BEGIN
        INSERT INTO totalpurchasesperpackage(package_id)
        VALUES(NEW.servicePackageToSelect_id);
END;


create table totalpurchasesperpackage
(
    package_id int not null primary key,
    totalPurchases int default 0 not null,
    constraint numberOfTotalPurchasesPerPackage_fk0
        foreign key (package_id) references servicepackagetoselect (servicePackageToSelect_id)
);
```

MOTIVATIONS

Each time an employee creates a service package, it is added to the table "totalpurchasesperpackage" that contains the servicePackageID and the default value of total purchases equal to zero.

Every time a customer buys a service package (or updates an order, passing from an invalid order to a valid one), the value of total purchases of the package purchased, changes and it is incremented by one, in the table "totalpurchasesperpackage".

TRIGGERS – second query

```
CREATE TRIGGER createServPackageAndValPeriod
    AFTER INSERT ON proposal FOR EACH ROW BEGIN
    INSERT INTO totalpurchasesperpackageandvalperiod(package_id, valPeriod_id)
    VALUES(NEW.servicePackageToSelect_id, NEW.validityPeriod_id);

END;

CREATE TRIGGER addPurchaseToPackageAndValPeriod
    AFTER INSERT ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        UPDATE totalpurchasesperpackageandvalperiod SET totalPurchases = totalPurchases + 1
        WHERE (package_id, valPeriod_id) IN (SELECT s.packageSelected, s.validityPeriod
                                             FROM servicepackage s
                                             WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
end;

CREATE TRIGGER updatePurchaseToPackageAndValPeriod
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        UPDATE totalpurchasesperpackageandvalperiod SET totalPurchases = totalPurchases + 1
        WHERE (package_id, valPeriod_id) IN (SELECT s.packageSelected, s.validityPeriod
                                             FROM servicepackage s
                                             WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
end;
```

TRIGGERS – second query

```
create table totalpurchasesperpackageandvalperiod
(
    package_id     int not null,
    valPeriod_id   int not null,
    totalPurchases int DEFAULT 0 not null,
    constraint totalpurchasesperpackageandvalperiod_fk0
        foreign key (package_id) references servicepackagetoselect (servicePackageToSelect_id),
    constraint totalpurchasesperpackageandvalperiod_fk1
        foreign key (valPeriod_id) references validityperiod (validityPeriod_Id)
);

create index totalpurchasesperpackageandvalperiod_fk0_idx
    on totalpurchasesperpackageandvalperiod (package_id);

create index totalpurchasesperpackageandvalperiod_fk1_idx
    on totalpurchasesperpackageandvalperiod (valPeriod_id);
```

MOTIVATIONS

Each time an employee creates a service package with its validity periods, it is added to the table "totalpurchasesperpackageandvalperiod" that contains the servicePackageID, the validityPeriodID associated and the default value of total purchases equal to zero.

Every time a customer buys a service package (or updates an order, passing from an invalid order to a valid one), the value of total purchases of the package purchased, with its validity periods, changes and it is incremented by one, in the table "totalpurchasesperpackageandvalperiod".

TRIGGERS – third query

```
CREATE TRIGGER addSales
    AFTER INSERT ON order FOR EACH ROW BEGIN
    DECLARE x,y float;
    IF NEW.isValid = true THEN
        SELECT sp.valuePackage, sp.totalValueOptionalProducts INTO x,y
        FROM servicepackage sp
        WHERE sp.servicePackage_id = NEW.servicePackageAssociated;

        UPDATE salespackage s
        SET s.totalSalesWithOptProduct = s.totalSalesWithOptProduct + x + y,
            s.totalSalesWithoutOptProduct = s.totalSalesWithoutOptProduct + x
        WHERE s.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated );
    END IF;
END;
```

TRIGGERS – third query

```
CREATE TRIGGER updateSales
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    DECLARE x,y float;
    IF NEW.isValid = true THEN
        SELECT sp.valuePackage, sp.totalValueOptionalProducts INTO x,y
        FROM servicepackage sp
        WHERE sp.servicePackage_id = NEW.servicePackageAssociated;

        UPDATE salespackage s
        SET s.totalSalesWithOptProduct = s.totalSalesWithOptProduct + x + y,
            s.totalSalesWithoutOptProduct = s.totalSalesWithoutOptProduct + x
        WHERE s.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated );
    END IF;
END;
```

```
CREATE TRIGGER createServicePackageToSelect2
    AFTER INSERT ON servicepackagetoselect FOR EACH ROW BEGIN
    INSERT INTO salespackage(package_id)
    VALUES(NEW.servicePackageToSelect_id);
END;
```

TRIGGERS – third query

```
create table salespackage
(
    package_id int not null primary key,
    totalSalesWithOptProduct int not null DEFAULT 0,
    totalSalesWithoutOptProduct int not null DEFAULT 0,
    constraint salespackage_fk0
        foreign key (package_id) references servicepackagetoselect
(servicePackageToSelect_id)
);
```

MOTIVATIONS

Each time an employee creates a service package, it is added to the table "salespackage" that contains the servicePackageID and the default value of total value of sales of the package with and without optional products equal to zero.
Every time a customer buys a service package (or updates an order, passing from an invalid order to a valid one), the value of the total sales of the service package purchases changes and to its old value of sales (with and without optional products), it is added the value of sales of the new service package, in the table "salespackage".

TRIGGERS – forth query

```
CREATE TRIGGER insertOptProduct
    AFTER INSERT ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        DELETE FROM totalnumberofoptionalproduct t
        WHERE t.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated );


        INSERT INTO totalnumberofoptionalproduct
        SELECT ss.servicePackageToSelect_id, count(*)
        FROM order as o
                 JOIN servicepackage as s on o.servicePackageAssociated = s.servicePackage_id
                 JOIN servicepackagetoselect as ss on s.packageSelected = ss.servicePackageToSelect_id
                 JOIN addedproduct as a on a.servicePackage_id = o.servicePackageAssociated
        WHERE o.isValid = true AND ss.servicePackageToSelect_id IN (SELECT s.packageSelected
                                               FROM servicepackage s
                                               WHERE s.servicePackage_id = NEW.servicePackageAssociated)
        GROUP BY ss.servicePackageToSelect_id;


        DELETE FROM avgnumofoptproductssoldperpackage
        WHERE package_id IN (   SELECT s.packageSelected
                                FROM servicepackage s
                                WHERE s.servicePackage_id = NEW.servicePackageAssociated);

        INSERT INTO avgnumofoptproductssoldperpackage
        SELECT t.package_id, IFNULL((top.total / t.totalPurchases), 0.0)
        FROM totalpurchasesperpackage as t
                 LEFT OUTER JOIN totalnumberofoptionalproduct as top on t.package_id = top.package_id

        WHERE t.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
end;
```

TRIGGERS – forth query

```
CREATE TRIGGER updateOptProduct
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    IF NEW.isValid = true THEN
        DELETE FROM totalnumberofoptionalproduct t
        WHERE t.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated );


        INSERT INTO totalnumberofoptionalproduct
        SELECT ss.servicePackageToSelect_id, count(*)
        FROM order as o
                 JOIN servicepackage as s on o.servicePackageAssociated = s.servicePackage_id
                 JOIN servicepackagetoselect as ss on s.packageSelected = ss.servicePackageToSelect_id
                 JOIN addedproduct as a on a.servicePackage_id = o.servicePackageAssociated
        WHERE o.isValid = true AND ss.servicePackageToSelect_id IN (SELECT s.packageSelected
                                               FROM servicepackage s
                                               WHERE s.servicePackage_id = NEW.servicePackageAssociated)
        GROUP BY ss.servicePackageToSelect_id;


        DELETE FROM avgnumofoptproductssoldperpackage
        WHERE package_id IN (   SELECT s.packageSelected
                                FROM servicepackage s
                                WHERE s.servicePackage_id = NEW.servicePackageAssociated);

        INSERT INTO avgnumofoptproductssoldperpackage
        SELECT t.package_id, IFNULL((top.total / t.totalPurchases), 0.0)
        FROM totalpurchasesperpackage as t
                 LEFT OUTER JOIN totalnumberofoptionalproduct as top on t.package_id = top.package_id

        WHERE t.package_id IN (SELECT s.packageSelected
                               FROM servicepackage s
                               WHERE s.servicePackage_id = NEW.servicePackageAssociated);
    END IF;
end;
```

TRIGGERS – forth query

```
CREATE TRIGGER createServicePackageToSelect3
    AFTER INSERT ON servicepackagetoselect FOR EACH ROW BEGIN
    INSERT INTO avgnumofoptproductssoldperpackage(PACKAGE_ID)
    VALUES(NEW.servicePackageToSelect_id);
END;

create table totalnumberofoptionalproduct
(
    package_id int not null primary key,
    total      int not null DEFAULT 0,
    constraint totalnumberofoptionalproduct_fk0
        foreign key (package_id) references servicepackagetoselect (servicePackageToSelect_id)
);


create table avgnumofoptproductssoldperpackage
(
    package_id int              not null primary key,
    average    float default -1 not null,
    constraint avgnumofoptproductssoldperpackage_fk0
        foreign key (package_id) references servicepackagetoselect (servicePackageToSelect_id)
);
```

MOTIVATIONS

Each time an employee creates a service package, it is added to the table "avgnumofoptproductssoldperpackage" that contains the servicePackageID and the default average number of optional products sold together equal to -1.
Every time a customer buys a service package (or updates an order, passing from an invalid order to a valid one), we insert into the table "totalnumberofoptionalproduct" the total number of optional products sold with each package and then we compute the average number of optional products sold together for each service package, inserting the value in the table "avgnumofoptproductssoldperpackage". In this table, the average is computed with the following formula: total of optional products sold with each package / how many times a service package is purchased.

TRIGGERS – fifth query

```sql
CREATE TRIGGER addAlert
    AFTER INSERT ON alert FOR EACH ROW BEGIN
    INSERT INTO alerts
    VALUES (NEW.alert_id);
end;

create table alerts
(
    alert_id int not null,
    constraint alerts_fk0 foreign key (alert_id) references alert (alert_id)
);

create index alerts_fk0_idx
    on alerts (alert_id);


CREATE TRIGGER updateInsolventUser
    AFTER UPDATE ON user FOR EACH ROW BEGIN
     IF NEW.isInsolvent = true THEN
        IF(NEW.user_id NOT IN (SELECT user_id FROM insolventusers)) THEN
        INSERT INTO insolventusers
        VALUES (NEW.user_id);
    ELSE
        DELETE FROM insolventusers i
        WHERE i.user_id = NEW.user_id;
    END IF;
end;
```

TRIGGERS – fifth query

```
create table insolventusers
(
    user_id int not null,
    constraint insolventUsers_fk0 foreign key (user_id) references user (user_id)
);

create index insolventUsers_fk0_idx
    on insolventusers (user_id);

CREATE TRIGGER addSuspendedOrder
    AFTER INSERT ON order FOR EACH ROW BEGIN
    IF(NEW.isValid = false) THEN
        IF(NEW.servicePackageAssociated NOT IN (SELECT order_id FROM suspendedorders)) THEN
            INSERT INTO suspendedorders
            SELECT s.servicePackage_id
            FROM servicepackage s
            WHERE NEW.servicePackageAssociated = s.servicePackage_id;
        END IF;
    ELSE
        IF(NEW.servicePackageAssociated IN (SELECT order_id FROM suspendedorders)) THEN
            DELETE FROM suspendedorders s
            WHERE s.order_id = NEW.servicePackageAssociated;
        END IF;
    END IF;
end;
create table suspendedorders
(
    order_id int not null,
    constraint suspendedOrders_fk0 foreign key (order_id) references order (order_id)
);
```

TRIGGERS – fifth query

```
create index suspendedOrders_fk0_idx
    on suspendedorders (order_id);

CREATE TRIGGER updateSuspendedOrder
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    IF(NEW.isValid = false) THEN
        IF(NEW.servicePackageAssociated NOT IN (SELECT order_id FROM suspendedorders)) THEN
            INSERT INTO suspendedorders
            SELECT s.servicePackage_id
            FROM servicepackage s
            WHERE NEW.servicePackageAssociated = s.servicePackage_id;
        END IF;
    ELSE
        IF(NEW.servicePackageAssociated IN (SELECT order_id FROM suspendedorders)) THEN
            DELETE FROM suspendedorders s
            WHERE s.order_id = NEW.servicePackageAssociated;
        END IF;
    END IF;
end;
```

MOTIVATIONS

Each time an alert is created, it is added in the table "alerts" that contains only the alert id.
Each time the attribute "isInsovent" of an user is updated, if he is insolvent, he is added in the table "insolventusers" (if not present already), otherwise, if he is not insolvent anymore, he is deleted from that table.
Every time a customer buys a service package (or updates an order), if that order is not valid, it is added in the table "suspendedorders" (if not already present). If the order is valid, it is deleted from that table.

TRIGGERS – sixth query

```
CREATE TRIGGER addSalesOptProduct
    AFTER INSERT ON order FOR EACH ROW BEGIN
    DECLARE id int;
    DECLARE amount int;
    IF NEW.isValid = true THEN
        DELETE FROM optProductsOrder;

        INSERT INTO optProductsOrder
        SELECT op.optionalProduct_id, (op.monthlyFee * v.numOfMonths)
        FROM order o
                 JOIN servicepackage s on s.servicePackage_id = o.servicePackageAssociated
                 JOIN addedproduct a on a.servicePackage_id = s.servicePackage_id
                 JOIN validityperiod v on v.validityPeriod_Id = s.validityPeriod
                 JOIN optionalproduct op on op.optionalProduct_Id = a.optionalProduct_id
        WHERE s.servicePackage_id = NEW.servicePackageAssociated;

        UPDATE salesperoptproduct s, optProductsOrder op
        SET s.salesOfOptProd = s.salesOfOptProd + op.salesOfOptProd
        WHERE s.optionalProduct_id = op.optionalProduct_id;

        DELETE FROM bestoptionalproduct;
        INSERT INTO bestoptionalproduct
        SELECT s1.optionalProduct_id, s1.salesOfOptProd
        FROM salesperoptproduct s1
        WHERE s1.optionalProduct_id is not null and s1.salesOfOptProd IN (SELECT MAX(s2.salesOfOptProd)
FROM salesperoptproduct s2);
    end if;
end;
```

TRIGGERS – sixth query

```
CREATE TRIGGER updateSalesOptProduct
    AFTER UPDATE ON order FOR EACH ROW BEGIN
    DECLARE id int;
    DECLARE amount int;
    IF NEW.isValid = true THEN
        DELETE FROM optProductsOrder;

        INSERT INTO optProductsOrder
        SELECT op.optionalProduct_id, (op.monthlyFee * v.numOfMonths)
        FROM order o
                 JOIN servicepackage s on s.servicePackage_id = o.servicePackageAssociated
                 JOIN addedproduct a on a.servicePackage_id = s.servicePackage_id
                 JOIN validityperiod v on v.validityPeriod_Id = s.validityPeriod
                 JOIN optionalproduct op on op.optionalProduct_Id = a.optionalProduct_id
        WHERE s.servicePackage_id = NEW.servicePackageAssociated;

        UPDATE salesperoptproduct s, optProductsOrder op
        SET s.salesOfOptProd = s.salesOfOptProd + op.salesOfOptProd
        WHERE s.optionalProduct_id = op.optionalProduct_id;

        DELETE FROM bestoptionalproduct;
        INSERT INTO bestoptionalproduct
        SELECT s1.optionalProduct_id, s1.salesOfOptProd
        FROM salesperoptproduct s1
        WHERE s1.optionalProduct_id is not null and s1.salesOfOptProd IN (SELECT MAX(s2.salesOfOptProd)
FROM salesperoptproduct s2);
    end if;
end;
```

TRIGGERS – sixth query

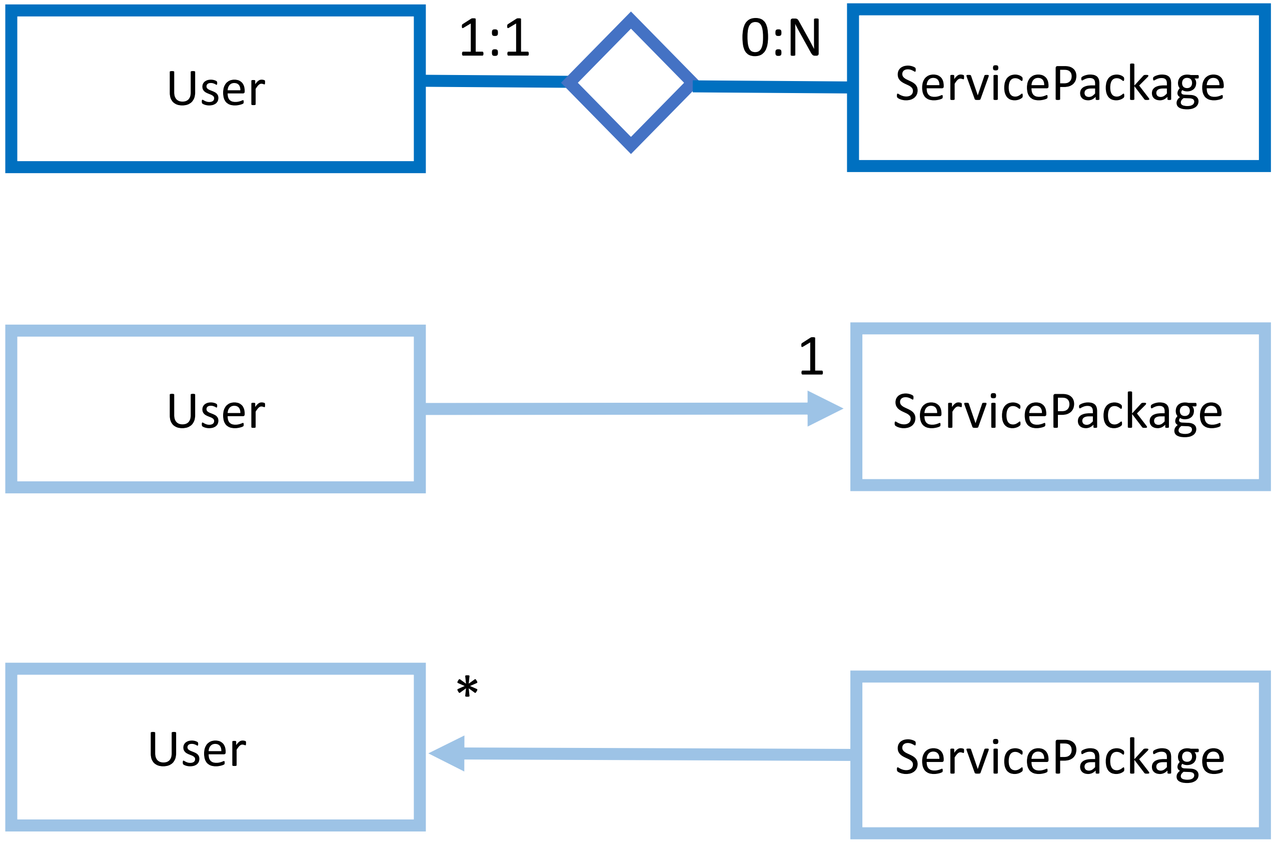
```
CREATE TRIGGER insertOptProduct
    AFTER INSERT ON optionalproduct FOR EACH ROW BEGIN
    INSERT INTO salesperoptproduct(optionalProduct_id)
    VALUES(NEW.optionalProduct_Id);
END;

create table bestoptionalproduct
(
    optionalProduct_id int   not null primary key,
    sales              float not null,
    constraint bestoptionalproduct_fk0 foreign key (optionalProduct_id) references optionalproduct (optionalProduct_Id)
);

create table optProductsOrder
(
    optionalProduct_id int   not null primary key,
    salesOfOptProd     float default 0 not null
);

create table salesperoptproduct
(
    optionalProduct_id int             not null,
    salesOfOptProd     float default 0 not null
);
```

MOTIVATIONS

Each time an employee creates an optional product, it is added to the table "salesperoptproduct" that contains the optProductID and the default value of sales of that optional product equal to 0.

Every time a customer buys a service package (or updates an order, passing from an invalid order to a valid one), we save in the temporary table "optProductsOrder", all the optional products just sold with the service package with their value sales (number of months * monthly fee). After that, we update the table "salesperoptproduct", adding to the optional products just sold, the new value of their sales. At least, we insert in the table "bestoptionalproduct", the optional product with the maximum value of sales, taken from the table "salesperoptproduct".

RELATIONSHIP



User → ServicePackage

@OneToMany

→ Eager Fetch

→ Cascade in All cases

→ Orphan Removal

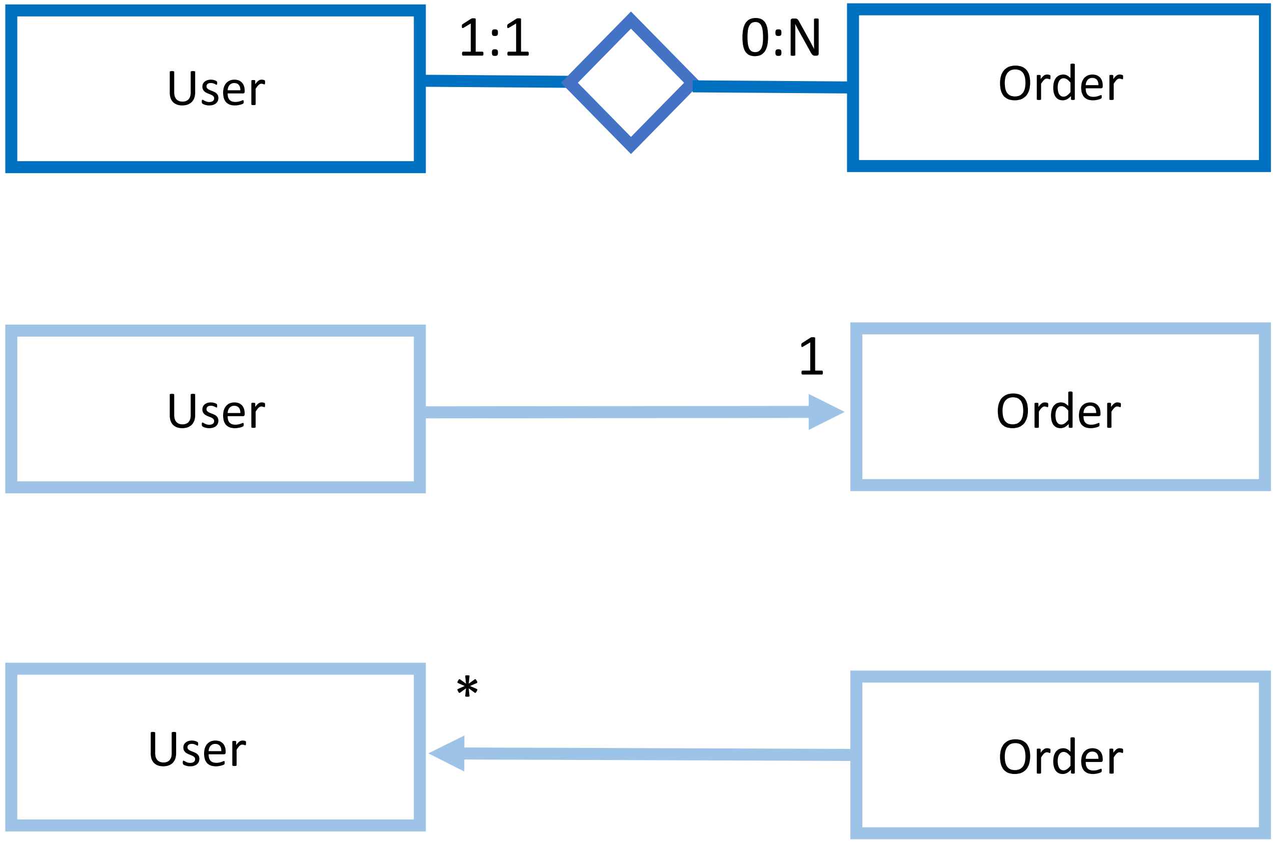
ServicePackage → User

@ManyToOne

→ Eager Fetch

→ Cascade in All cases apart from the delete

RELATIONSHIP



User → Order

@OneToMany

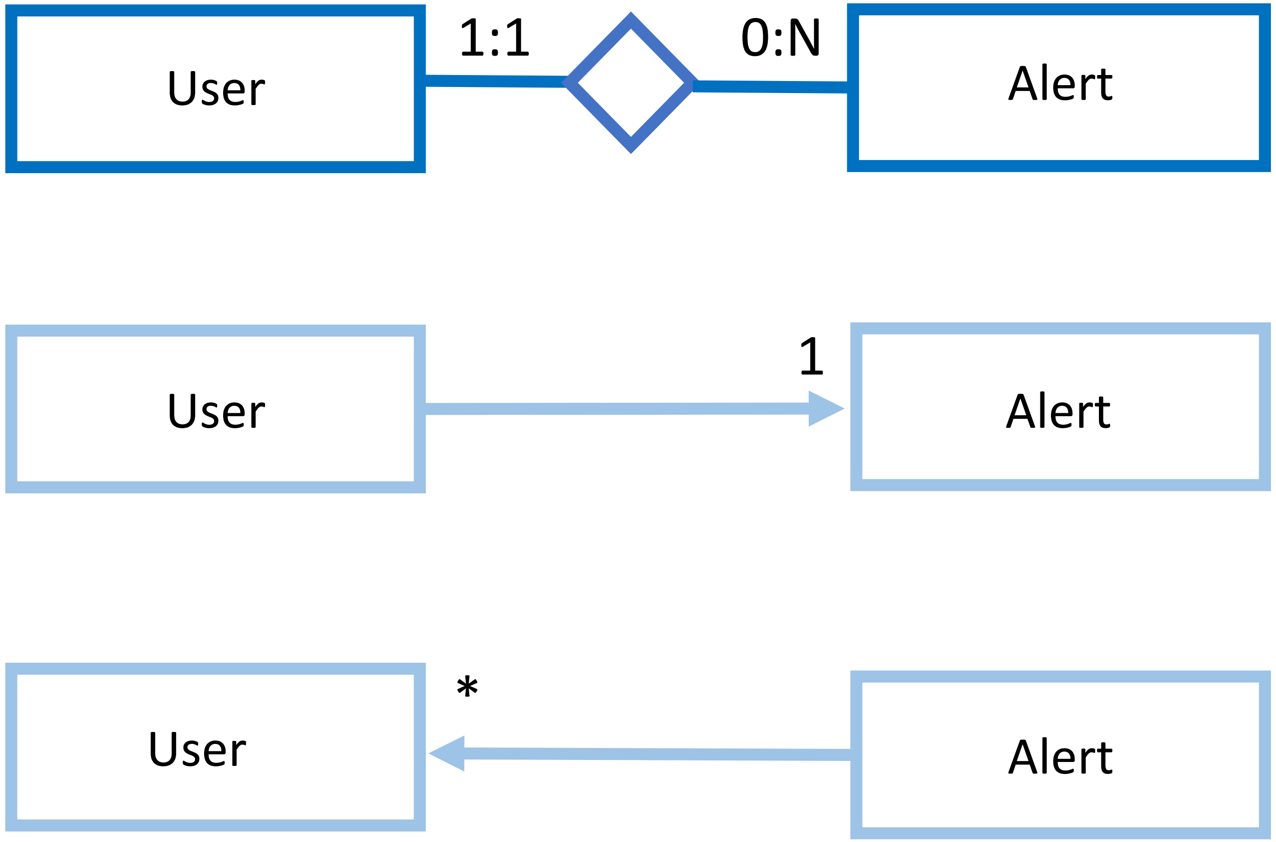
→ Lazy Fetch
→ Cascade in All cases
→ Orphan Removal

Order → User

@ManyToOne

→ Eager Fetch
→ Cascade in All cases apart from the delete

RELATIONSHIP



User → Alert

@OneToMany

→ Eager Fetch

→ Cascade in All cases

→ Orphan Removal

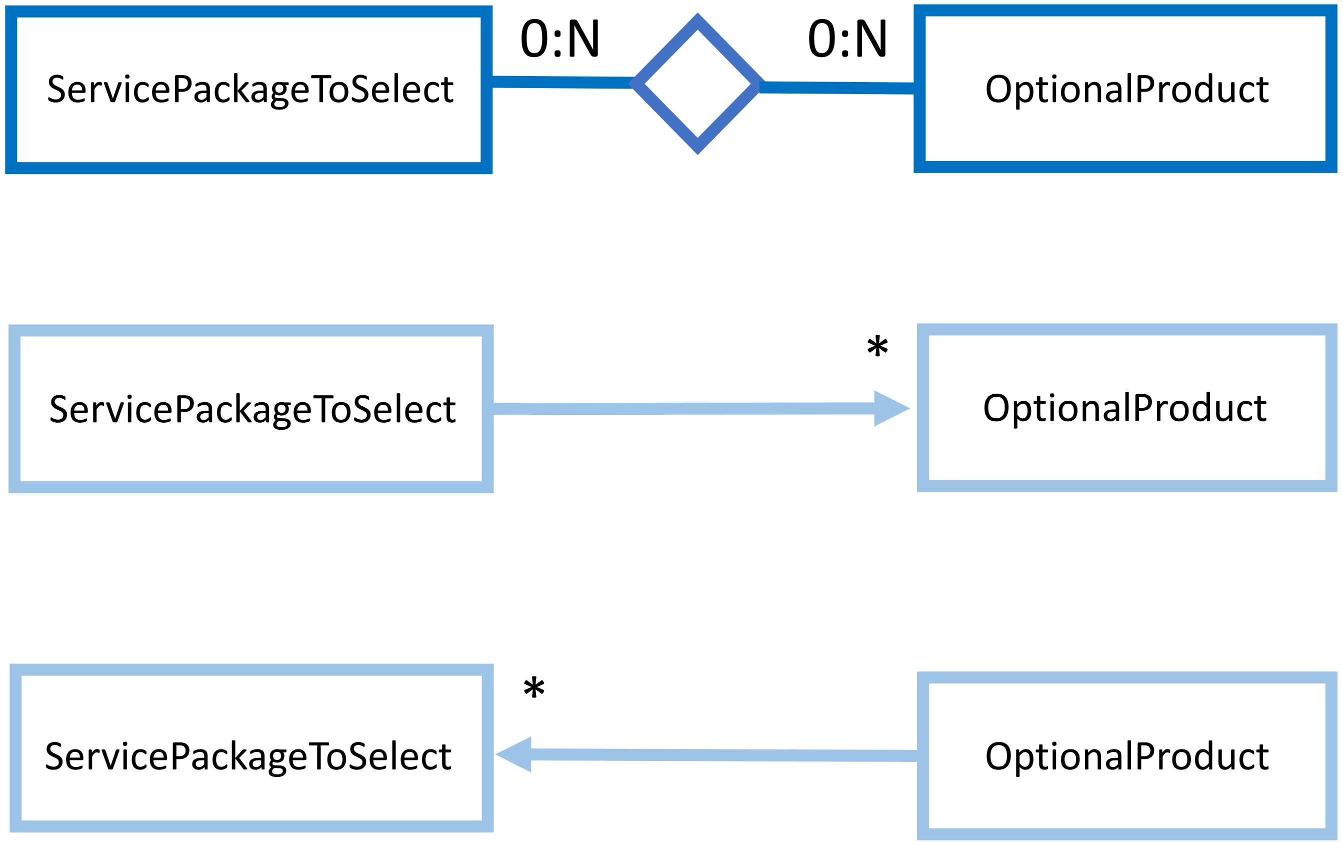
Alert→ User

@ManyToOne

→ Eager Fetch

→ Cascade in All cases apart from the delete

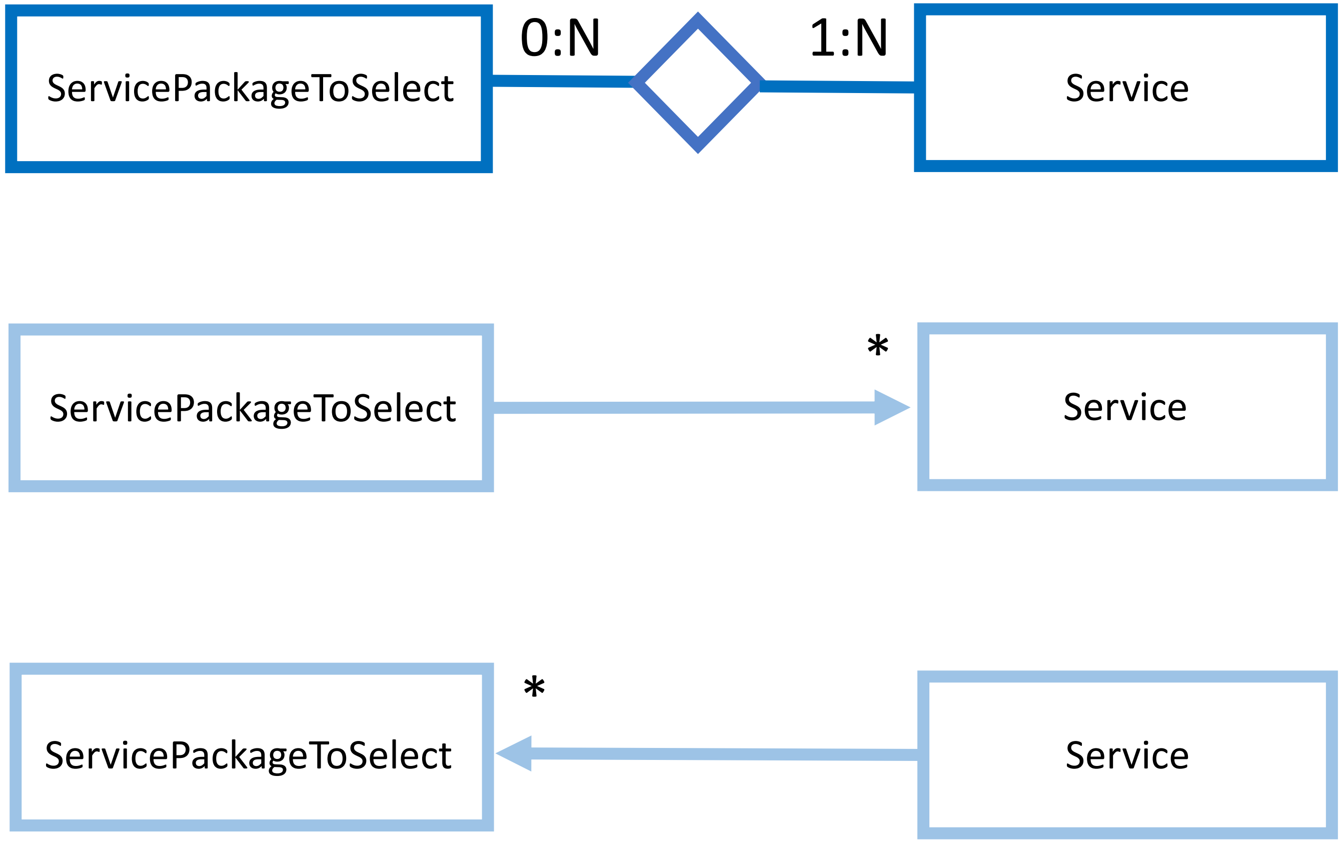
RELATIONSHIP



ServicePackageToSelect
→ OptionalProduct
@ManyToMany needed to show all the optional product of a service package to select
→ Eager Fetch
→ Cascade in All cases apart from the delete

Service → OptionalProduct
@ManyToMany not necessary can be mapped for consistency
→ Lazy Fetch
→ Cascade in All cases apart from the delete

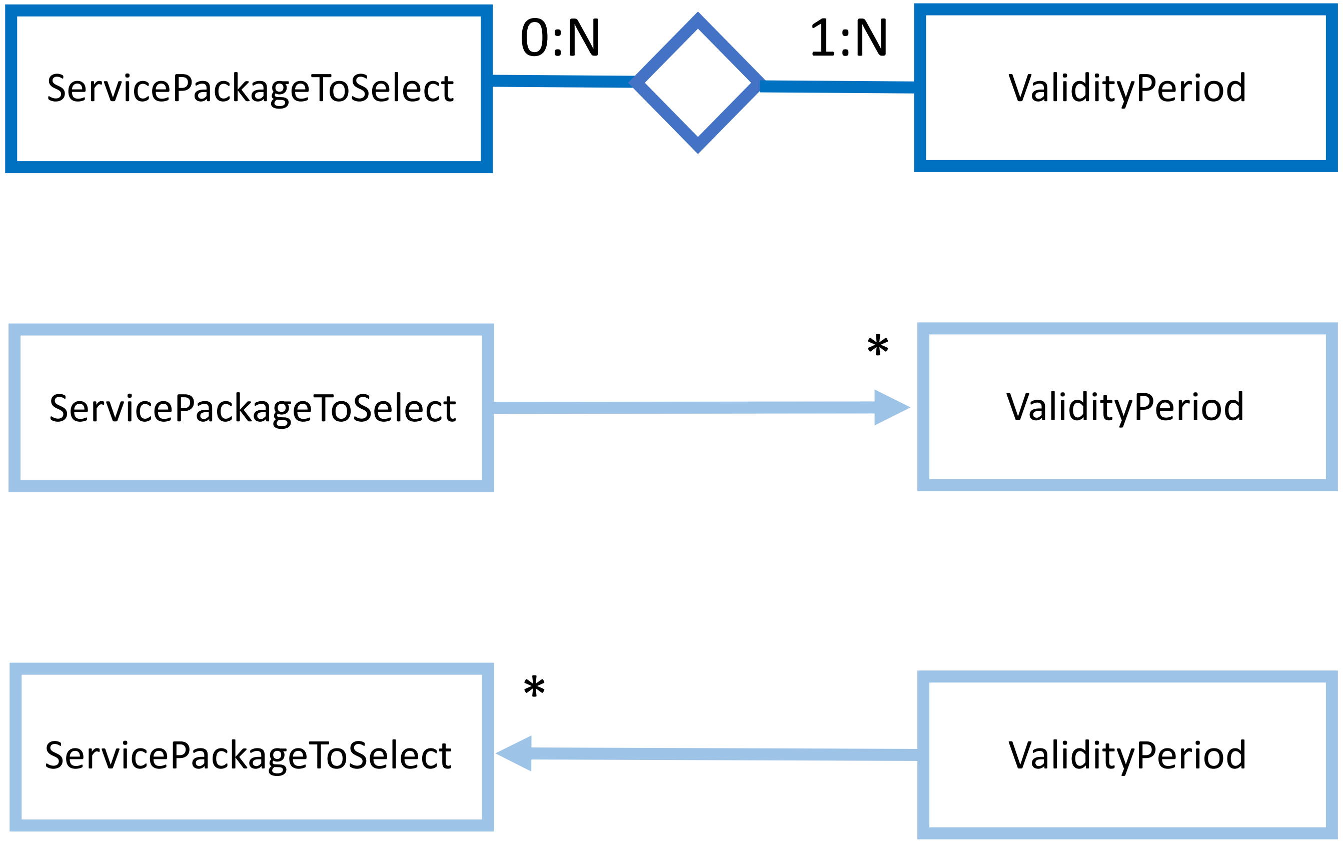
RELATIONSHIP



ServicePackageToSelect → Service
@ManyToMany needed to show all the services of a service package to select
→ Eager Fetch
→ Cascade in All cases apart from the delete

Service → ServicePackageToSelect
@ManyToMany not necessary can be mapped for consistency
→ Lazy Fetch
→ Cascade in All cases

RELATIONSHIP



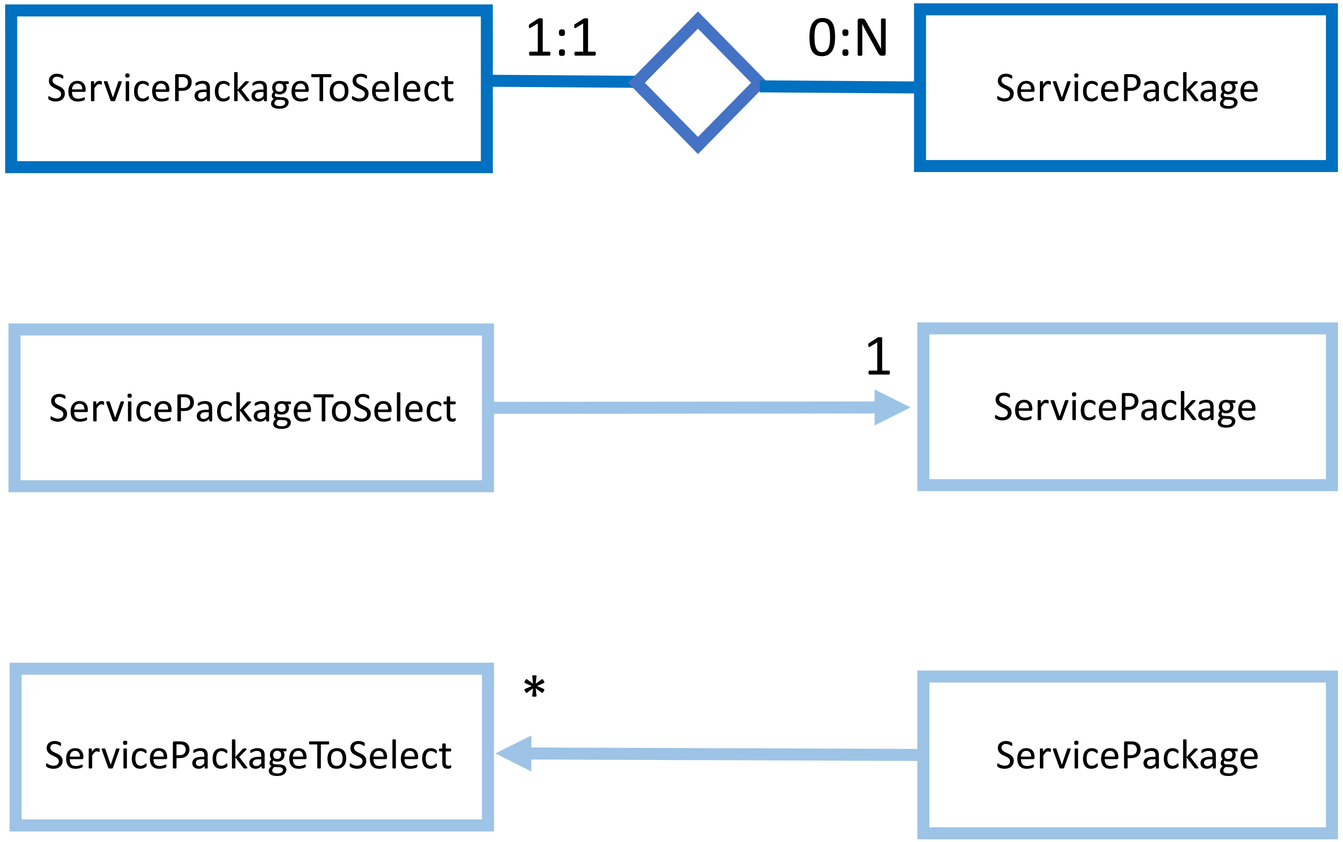
ServicePackageToSelect
→ ValidityPeriod

@ManyToMany needed to show all the validity periods of a service package to select
→ Eager Fetch
→ Cascade in All cases apart from the delete

ValidityPeriod
→ ServicePackageToSelect

@ManyToMany not necessary can be mapped for consistency
→ Lazy Fetch
→ Cascade in All cases

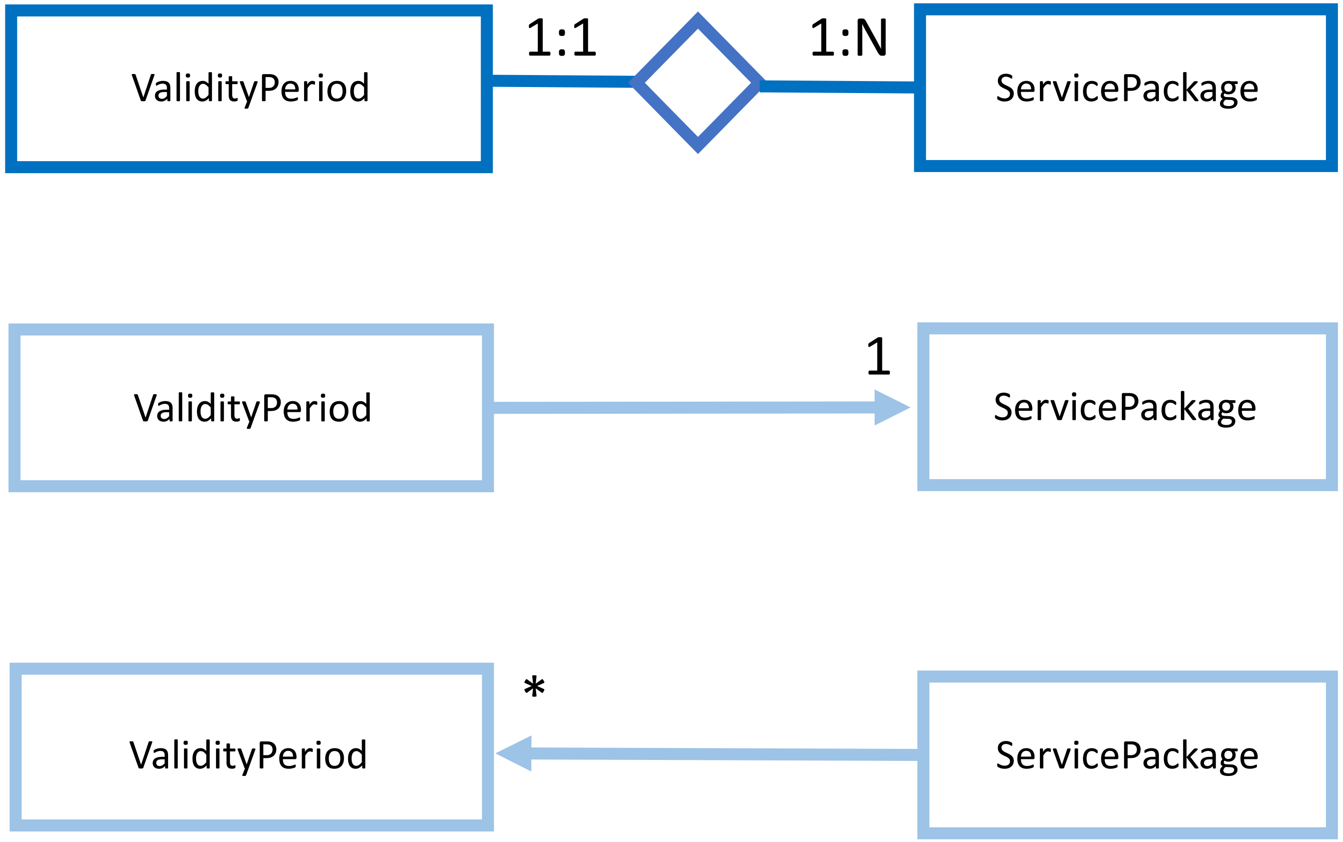
RELATIONSHIP



ServicePackageToSelect
→ServicePackage
@OneToMany
→ Lazy Fetch
→ Cascade in All cases
→ Orphan Removal

ServicePackage
→ ServicePackageToSelect
@ManyToOne
→ Lazy Fetch
→ Cascade in All cases apart from the delete

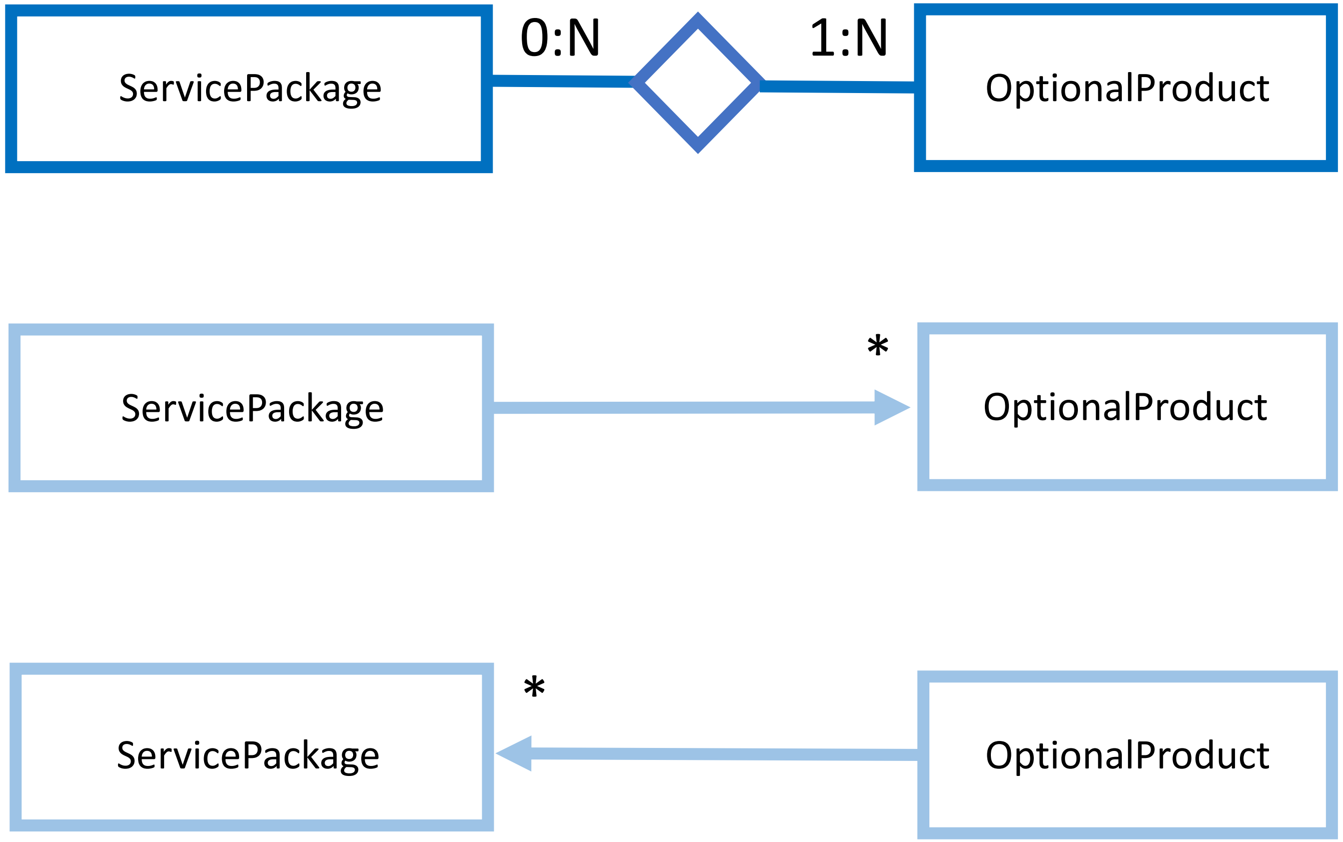
RELATIONSHIP



ValidityPeriod→ServicePackage
@OneToMany
→ Lazy Fetch
→ Cascade in All cases
→ Orphan Removal

ServicePackage → ValidityPeriod
@ManyToOne
→ Eager Fetch
→ Cascade in All cases apart from the delete

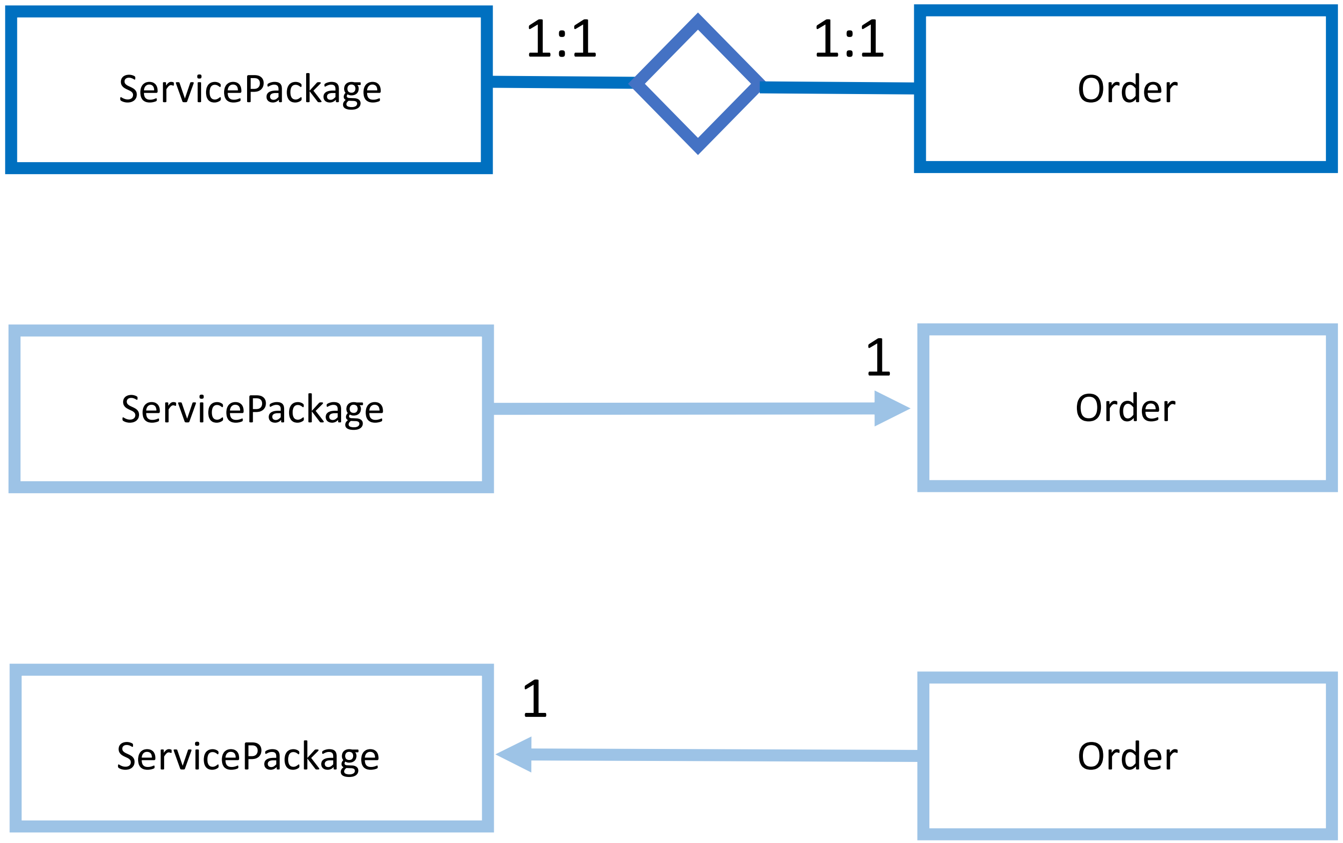
RELATIONSHIP



ServicePackage→ OptionalProduct
@ManyToMany needed to show all the optional products of a service package
→ Eager Fetch
→ Cascade in All cases apart from the delete

OptionalProduct→ ServicePackage
@ManyToMany not necessary can be mapped for consistency
→ Lazy Fetch
→ Cascade in All cases apart from the delete

RELATIONSHIP



ServicePackage→ Order

@OneToOne

→ Cascade in All cases

→ Orphan Removal

Order→ ServicePackage

@OneToOne

→ Cascade in All cases apart from the delete

ENTITY User

```
@Entity

@NamedQuery(
   name = " User.checkCredentials ",
   query = "SELECT r FROM UserEntity r " +
      "WHERE r.username = ?1 and r.password = ?2"
)
@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"
)
@NamedQuery(
   name = "User.findByID",
   query = "SELECT u FROM UserEntity u " +
      "WHERE u.user_id = : user_id"
)
@Table(name = "user", schema = "dbtelco")
public class UserEntity implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "user_id", nullable=false)
   private Long user_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 = "isInsolvent")
 private Boolean isInsolvent;

 @Column(name = "numFailedPayments")
 private int numFailedPayments;

 @OneToMany(fetch= FetchType.EAGER, mappedBy="userOwner", cascade = Cascade
     Type.ALL, orphanRemoval = true)
 private List<ServicePackageEntity> servicePackages;

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

 @OneToMany(fetch= FetchType.EAGER, mappedBy="userOwner", cascade = Cascade
    Type.ALL, orphanRemoval = true)
 private List<AlertEntity> alerts;
 …
}
```

ENTITY Employee

```
@Entity

@NamedQuery(
   name = "Employee.findByUsername",
   query = "SELECT e FROM  EmployeeEntity  e " +
      "WHERE e. username = :username"
)
@NamedQuery(
   name = "Employee. findByEmail ",
   query = "SELECT e FROM  EmployeeEntity  e " +
      "WHERE e. email = : email "
)
@NamedQuery(
   name = "Employee. checkCredentials ",
   query = "SELECT e FROM  EmployeeEntity  e " +
      "WHERE  e.username = ?1 and e.password = ?2"
)
```

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

   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "employee_id", nullable = false)
   private Long employee_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;
   …
}
```

ENTITY Service

```
@Entity

@NamedQuery(
      name = "Service.findByID",
      query = "SELECT s FROM ServiceEntity s " +
            "WHERE s.service_id = :service_id"
)
@NamedQuery(
   name = "Service.findAll",
   query = "SELECT o FROM ServiceEntity o "
)

@Table(name = "service", schema = "dbtelco")
public class ServiceEntity implements Serializable {
   private static final long serialVersionUID = 1L;

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

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

   @Column(name = "numMinutes")
   private int numMinutes;
```

```
   @Column(name = "numSMS")
   private int numSMS;

   @Column(name = "feeExtraMinute")
   private float feeExtraMinute;

   @Column(name = "feeExtraSMSs")
   private float feeExtraSMSs;

   @Column(name = "numGigabytes")
   private int numGigabytes;

   @Column(name = "feeForExtraGigabytes")
   private float feeForExtraGigabytes;

   @ManyToMany(mappedBy = "services", fetch = FetchType.LAZY, cascade =
           CascadeType.ALL)
   private List<ServicePackageToSelectEntity> servicePackagesToSelect;
   …
}
```

ENTITY ServicePackageToSelect

```
@Entity

@NamedQuery(
        name = "ServicePackageToSelect.findByID",
        query = "SELECT s FROM ServicePackageToSelectEntity s " +
                "WHERE s.servicePackageToSelect_id = :servicePackageToSelect_id"
)
@NamedQuery(
        name = "ServicePackageToSelect.findByName",
        query = "SELECT s FROM ServicePackageToSelectEntity s " +
                "WHERE s.name = :name"
)

@NamedQuery(
    name = "ServicePackageToSelect.findAll",
    query = "SELECT stp FROM ServicePackageToSelectEntity stp "
)

@Table(name = "servicepackagetoselect", schema = "dbtelco")
public class ServicePackageToSelectEntity implements Serializable {
    private static final long serialVersionUID = 1L;

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

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

```
    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
        name="offer",
        joinColumns={@JoinColumn(name="servicePackageToSelect_id")},
        inverseJoinColumns={@JoinColumn(name="service_id")}
    )
    private List<ServiceEntity> services;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
        name="offerProduct",
        joinColumns={@JoinColumn(name="servicePackageToSelect_id")},
        inverseJoinColumns={@JoinColumn(name="optionalProduct_id")}
    )
    private List<OptionalProductEntity> optionalProducts;

    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(
        name="proposal",
        joinColumns={@JoinColumn(name="servicePackageToSelect_id")},
        inverseJoinColumns={@JoinColumn(name="validityPeriod_id")}
    )
    private List<ValidityPeriodEntity> validityPeriods;

    @OneToMany(mappedBy="packageSelected", fetch
        = FetchType.EAGER, cascade = CascadeType.ALL)
    private List<ServicePackageEntity> servicePackages;
    …
}
```

ENTITY ServicePackage

```
@Entity

@NamedQuery(
   name = "ServicePackage.findServicePackageThatContainServicePackageToSelect",
   query = "SELECT s FROM ServicePackageEntity s " +
      "WHERE s.packageSelected = : servicePackageToSelect_id ")

@NamedQuery(
   name =
"ServicePackage.findServicePackageThatContainServicePackageToSelectAndValPeriod",
   query = "SELECT s FROM ServicePackageEntity s " +
      "WHERE s.packageSelected = : servicePackageToSelect_id AND s.validityPeriod =:
validityPeriod_id ")

@NamedQuery(
      name = "ServicePackage.findServicePackageOfUser",
      query = "SELECT s FROM ServicePackageEntity s " +
            "WHERE s.userOwner = : user"
)

@Table(name = "servicepackage", schema = "dbtelco")
public class ServicePackageEntity implements Serializable {
   private static final long serialVersionUID = 1L;

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

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

  @Temporal(TemporalType.DATE)
   @Column(name = "startDate")
   private Date startDate;
```

```
@Temporal(TemporalType.DATE)
@Column(name = "endDate")
private Date endDate;

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

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

@ManyToOne (fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST,
CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn(name = "packageSelected")
private ServicePackageToSelectEntity packageSelected;

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

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(
   name="addedProduct",
   joinColumns={@JoinColumn(name="servicePackage_id")},
   inverseJoinColumns={@JoinColumn(name="optionalProduct_id")})
private List<OptionalProductEntity> optionalProducts;

@ManyToOne (fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST,
CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
@JoinColumn(name = "userOwner")
private UserEntity userOwner;

@OneToOne(mappedBy = "servicePackageAssociated", cascade = CascadeType.ALL)
private OrderEntity order;
… }
```

ENTITY ValidityPeriod

```
@Entity

@NamedQuery(
      name = "ValidityPeriod.findByID",
      query = "SELECT v FROM ValidityPeriodEntity v " +
            "WHERE v.validityPeriod_id = :validityPeriod_id"
)

@NamedQuery(
      name = "ValidityPeriod. findAll ",
      query = "SELECT v FROM ValidityPeriodEntity v "
)

@NamedQuery(
      name = "ValidityPeriod. findValPeriodsByServPackage ",
      query = "SELECT v FROM ValidityPeriodEntity v "+
                  "JOIN v.servicePackagesToSelect s " +
                  "WHERE s.servicePackageToSelect_id :servicePackageToSelect_id "
)
```

```
@Table(name = "validityperiod", schema = "dbtelco")
public class ValidityPeriodEntity implements Serializable {
@Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "validityPeriod_id", nullable = false)
   private Long validityPeriod_id;

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

   @Column(name = "monthlyFee")
   private float monthlyFee;

   @ManyToMany(mappedBy = "validityPeriods", fetch = FetchType.LAZY,
      cascade=CascadeType.ALL)
   private List<ServicePackageToSelectEntity> servicePackagesToSelect;


   @OneToMany(mappedBy="validityPeriod",
       fetch=FetchType.LAZY, cascade=CascadeType.ALL)
   private List<ServicePackageEntity> servicePackages;
   …
}
```

ENTITY OptionalProduct

```
@Entity

@NamedQuery(
      name = " OptionalProduct.findByID",
      query = "SELECT o FROM OptionalProductEntity o " +
            "WHERE o.optionalProduct _id = :optionalProduct_id"
)

@NamedQuery(
      name = " OptionalProduct. findByName ",
      query = "SELECT o FROM OptionalProductEntity o " +
            "WHERE o.name = :name"
)

@NamedQuery(
      name = " OptionalProduct. findAll ",
      query = "SELECT o FROM OptionalProductEntity o "
)

@NamedQuery(
      name = " OptionalProduct. findOptProdOfServicePackageToSelect ",
      query = "SELECT o FROM OptionalProductEntity o "+
            "JOIN o.servicePackagesToSelect s " +
            "WHERE s.servicePackageToSelect_id = :servicePackageToSelect_id "
)
```

```
@NamedQuery(
      name = " OptionalProduct. findOptProdOfServicePackage ",
      query = "SELECT o FROM OptionalProductEntity o "+
            "JOIN o.servicePackages s " +
            "WHERE s.servicePackage_id = :servicePackage_id "
)
```

```
@Table(name = "optionalproduct", schema = "dbtelco")
public class OptionalProductEntity implements Serializable {

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

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

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

   @ManyToMany(mappedBy = "optionalProducts", fetch = FetchType.EAGER)
   private List<ServicePackageEntity> servicePackages;

   @ManyToMany(mappedBy = "optionalProducts", fetch = FetchType.EAGER)
   private List<ServicePackageToSelectEntity> servicePackagesToSelect;
   …
}
```

ENTITY Order

```
@Entity

@NamedQuery(
        name = "Order.findByID",
        query = "SELECT o FROM OrderEntity o " +
                "WHERE o.order_id = :order_id"
)
@NamedQuery(
        name = "Order.findAllOrderByUser",
        query = "SELECT o FROM OrderEntity o " +
                "WHERE o.userOwner = :user "
)
@NamedQuery(
        name = "Order.findRejectedOrdersOfUser",
        query = "SELECT o FROM OrderEntity o " +
                "WHERE o.userOwner = :user AND " +
                "o.isValid=false"
)
@NamedQuery(
        name = "Order.findOrdersToActivate",
        query = "SELECT DISTINCT o FROM OrderEntity o " +
                "JOIN o.servicePackageAssociated s " +
                "WHERE o.userOwner = :user AND " +
                "o.isValid=true AND " +
                "s.startDate > CURRENT_TIMESTAMP "
)
```

```
@Table(name = "order", schema = "dbtelco")
public class OrderEntity implements Serializable {
    private static final long serialVersionUID = 1L;

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

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

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

    @Column(name = "isValid")
    private boolean isValid;

    @ManyToOne ((fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST,
    CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH}) , optional = false)
    @JoinColumn(name = "userOwner")
    private UserEntity userOwner;

    @OneToOne (cascade = {CascadeType.PERSIST, CascadeType.MERGE
          CascadeType.REFRESH, CascadeType.DETACH}), optional = false)
    @JoinColumn(name = "servicePackageAssociated")
    private ServicePackageEntity servicePackageAssociated;
…
}
```

ENTITY Alert

```
@Entity

@NamedQuery(
    name = "Alert.findByUser",
    query = "SELECT a " +
        "FROM AlertEntity a " +
        "WHERE a.userOwner = : user"
)

@Table(name = "alert", schema = "dbtelco")
public class AlertEntity implements Serializable {
  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "alert_id", nullable = false)
  private Long alertId;
```

```
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "alert_id", nullable = false)
  private Long alertId;

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

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

  @ManyToOne (fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST,
  CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
  @JoinColumn(name = "userOwner")
  private UserEntity userOwner;
  …
}
```

COMPONENTS

CLIENT TIER

- HTML page generated from *index.jsp*
- HTML page generated from *homeCustomer.jsp*
- HTML page generated from *homeEmployee.jsp*
- HTML page generated from *buyPage.jsp*
- HTML page generated from *confirmationPage.jsp*
- HTML page generated from *salesReportPage.jsp*
- HTML page generated from *serviceActivationSchedule.jsp*

COMPONENTS

WEB TIER

- LoginServlet
- LogoutServlet
- SignupServlet
- ServicePackageToSelectServlet
- ServiceActivationScheduleServlet
- OptionalProductServlet
- HomePageCustomerServlet
- HomePageEmployeeServlet
- BuyServlet
- ConfirmationServlet
- SalesReportPageServlet

COMPONENTS

BUSINESS TIER

- **UserService** → STATELESS
  - checkCredentials(String usrn, String pwd)
  - createUser(String username, String email, String password)
  - createAlert(AlertEntity alert)
  - findByUsername(String usrn)
  - findByEmail(String email)
  - createServicePackage(ServicePackageEntity servicePackage, UserEntity userOwner)
  - findAllServicePackageToSelect()
  - findAllOrdersByUser(Long user_id)
  - findByUserID(Long user_id)
  - findByServicePackageToSelectID(Long servicePackageToSelect_id)
  - findByOptProdID(Long optionalProduct_id)

COMPONENTS

BUSINESS TIER

- **UserService** → STATELESS
  - findByValPeriodID(Long validityPeriod_id)
  - findOrderByID(Long order_id)
  - findValPeriodsOfService(Long servicePackageToSelect_id)
  - findOptProdOfService(Long servicePackageToSelect_id)
  - findAllService()
  - createOrder(Timestamp now, UserEntity userOwner, ServicePackageEntity servicePackage)
  - updateOrder (OrderEntity order, boolean isValid)
  - setUserInsolvent(UserEntity user, boolean isInsolvent)
  - incrementsFailedPayments(UserEntity user)
  - setNumFailedPayments(UserEntity user)
  - findRejectedOrdersByUser(Long user_id)
  - findOrdersToActivate(Long user_id)
  - randomPayment()

COMPONENTS

BUSINESS TIER

- **EmployeeService**→ STATELESS

  - findByNameServicePackage(String namePackage)
  - findByNameOptProd(String nameOptProd)
  - findByUsername(String usrn)
  - findByEmail(String email)
  - checkCredentials(String usrn, String pwd)
  - createEmployee(String username, String email, String password)
  - createOptionalProduct(String name, float monthlyFee)
  - createServicePackageToSelect(String name, ArrayList<ServiceEntity> services, List<OptionalProductEntity> optionalProducts, List<ValidityPeriodEntity> validityPeriods)
  - findAllOptProd()
  - findAllServices()
  - findAllValidityPeriods()
  - findAllAlerts()

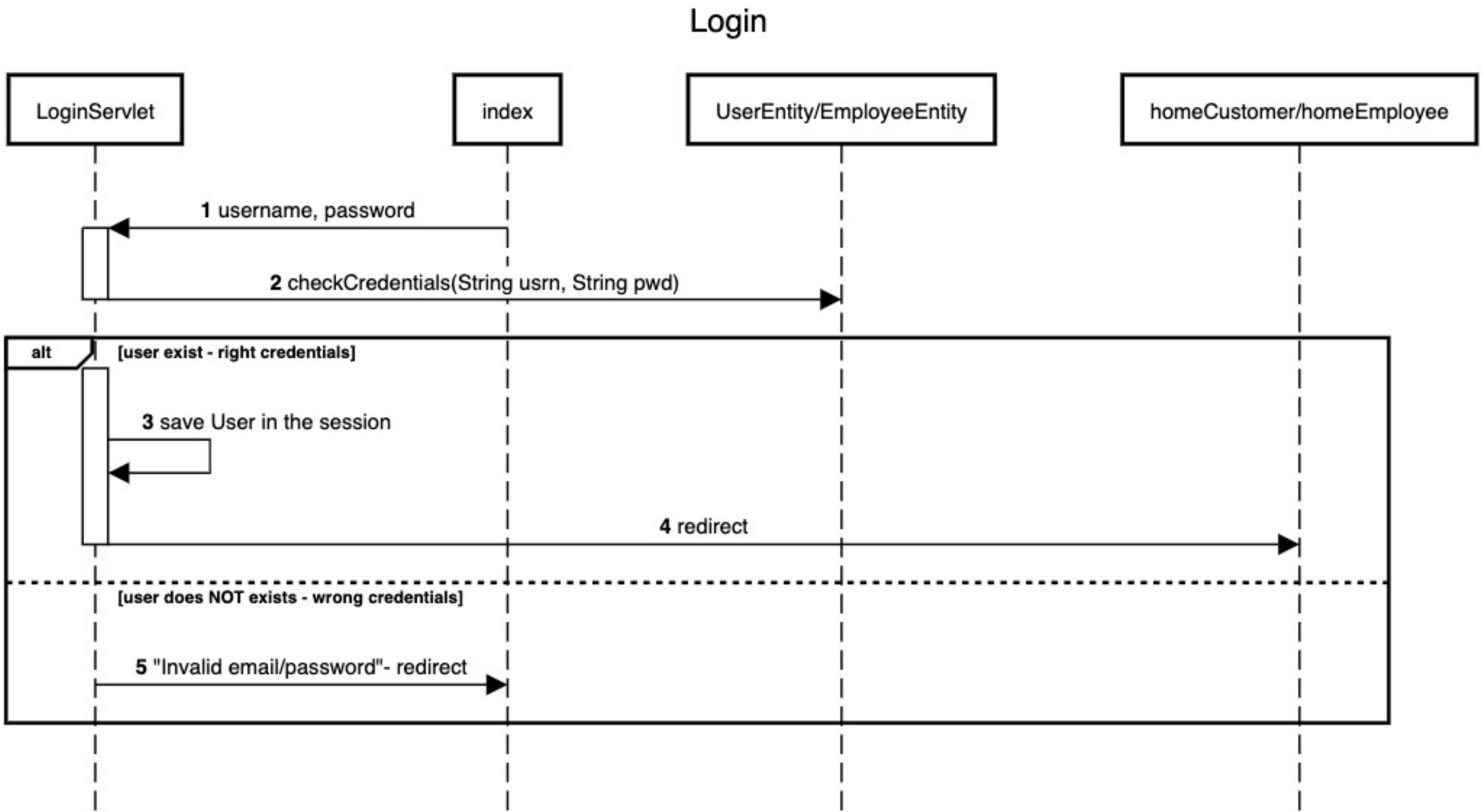
COMPONENTS

BUSINESS TIER

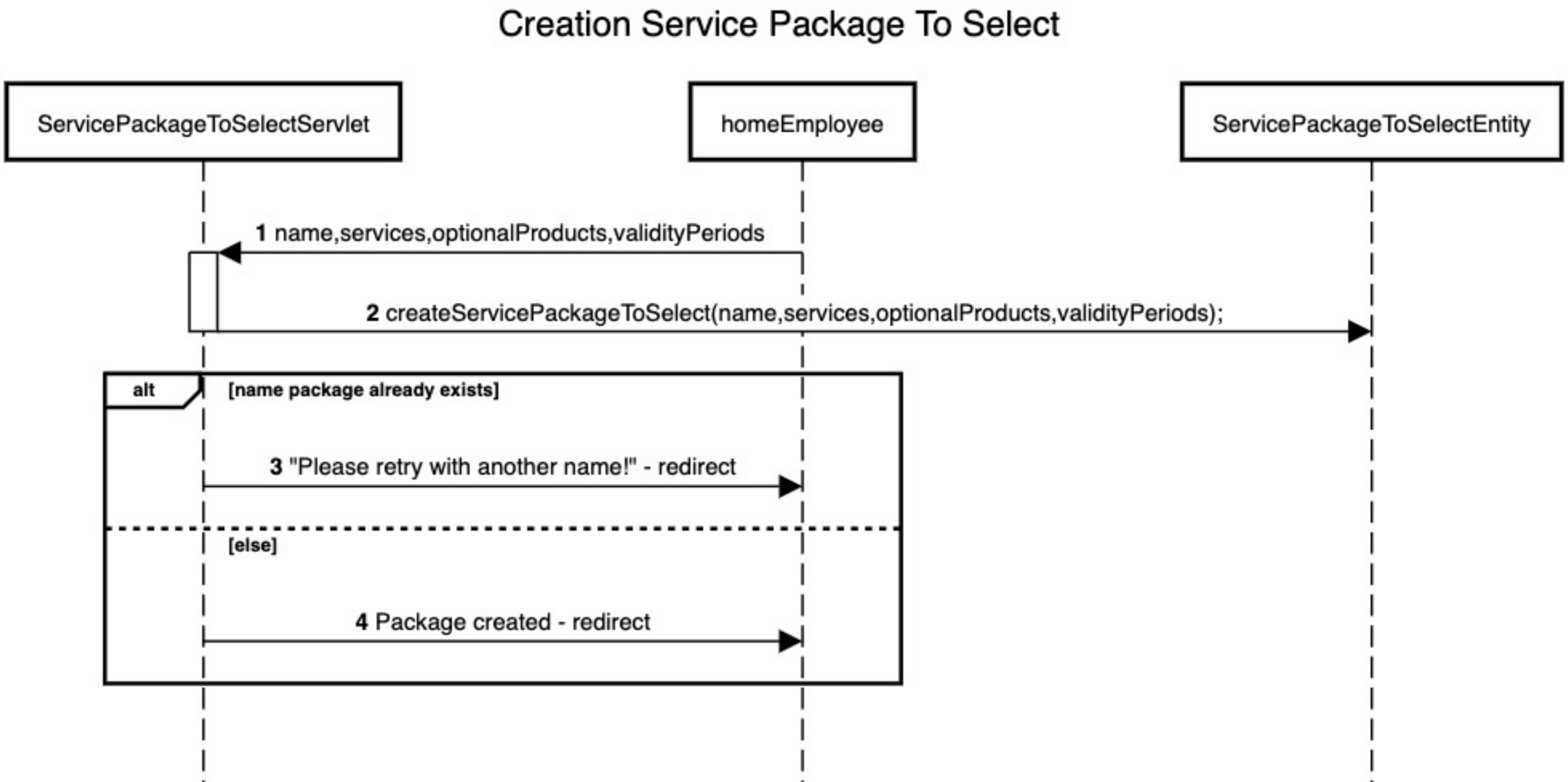
- **EmployeeService**→ STATELESS

  - findAllServicePackageToSelect()
  - findByServiceID(Long service_id)
  - findByOptProdID(Long optionalProduct_id)
  - findByValPeriodID(Long validityPeriod_id)
  - findValPeriodsOfServicePackage(Long servicePackageToSelect_id)
  - findServicePackageToSelectByID(Long servicePackageToSelect_id)
  - purchasesPerPackage(Long package_id)
  - purchasesPerPackageAndValPeriod(Long package_id, Long valPeriod_id)
  - valueOfSales (Long package_id)
  - avgNumOfOptProductsSoldPerPackage(Long package_id)
  - findAllInsolventUsers()
  - findAllSuspendedOrders()
  - findBest()

UML sequence diagram


Login
LoginServlet
index
UserEntity/EmployeeEntity
homeCustomer/homeEmployee
1 username, password
2 checkCredentials(String usrn, String pwd)
alt
[user exist - right credentials]
3 save User in the session
4 redirect
[user does NOT exists - wrong credentials]
5 "Invalid email/password"- redirect

UML sequence diagram



Creation Service Package To Select
ServicePackageToSelectServlet
homeEmployee
ServicePackageToSelectEntity
1 name,services,optionalProducts,validityPeriods
2 createServicePackageToSelect(name,services,optionalProducts,validityPeriods);
alt
[name package already exists]
3 "Please retry with another name!" - redirect
[else]
4 Package created - redirect