DATA BASES 2

TELCOSERVICE WEB PAGE

Vittorio Andreotti

SPECIFICATIONS

SPECIFICATION

Consumer Application

- View Packages.
- Create Orders.
- Pay for rejected Orders.
- View Alerts.

Employee Application

- Create new Packages.
- Create new Optional Products.
- View statistics in SalesReport page related to:
  - 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.

SPECIFICATION INTERPRETATION

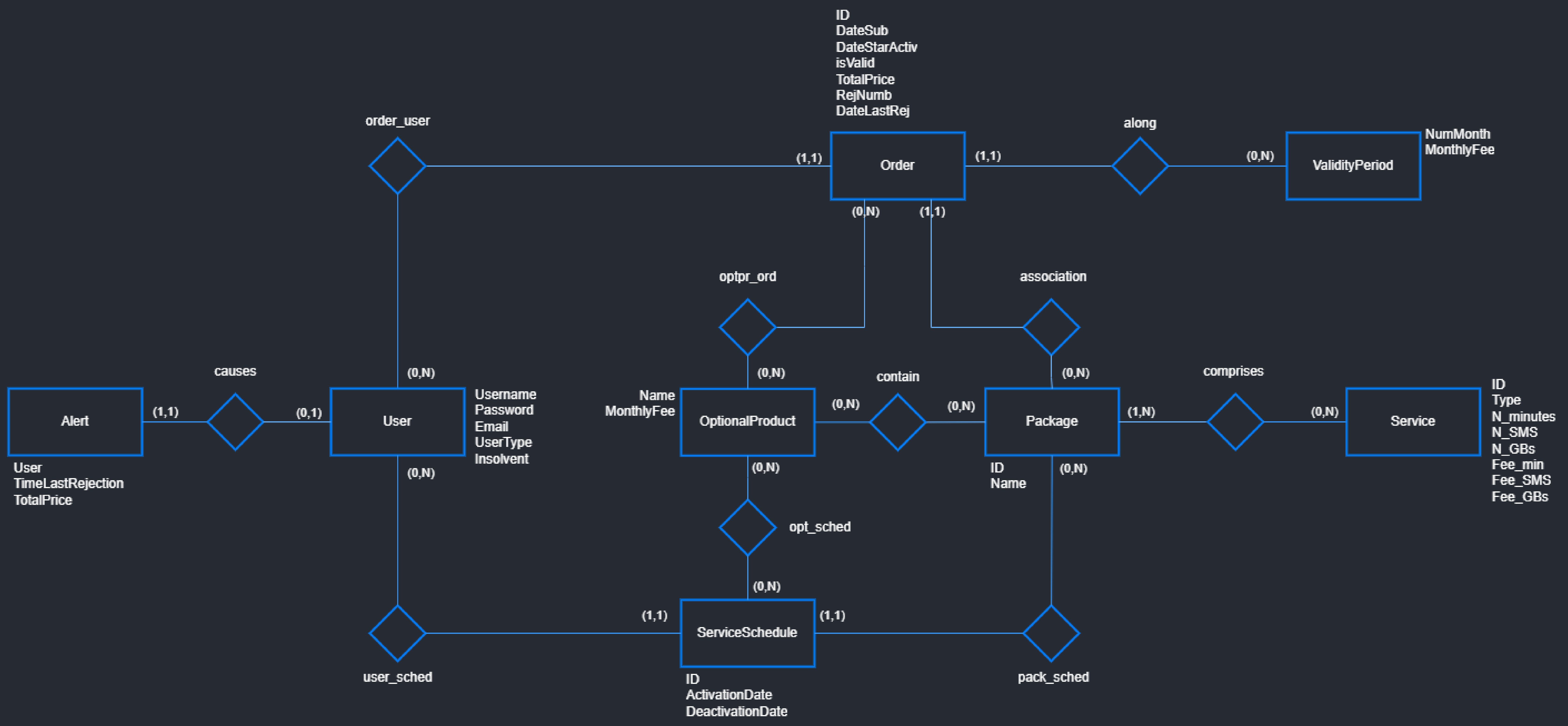
- Instead of creating two different applications, one for the users and one for the employees, the application is the same and the routing to the different pages is based on the value of is_valid field in the user’s table due to usability purposes.

- Instead of using a function to return to a random boolean value, two different buttons are displayed in the Confirmation page, one to mark the order as valid and one to mark it as invalid.

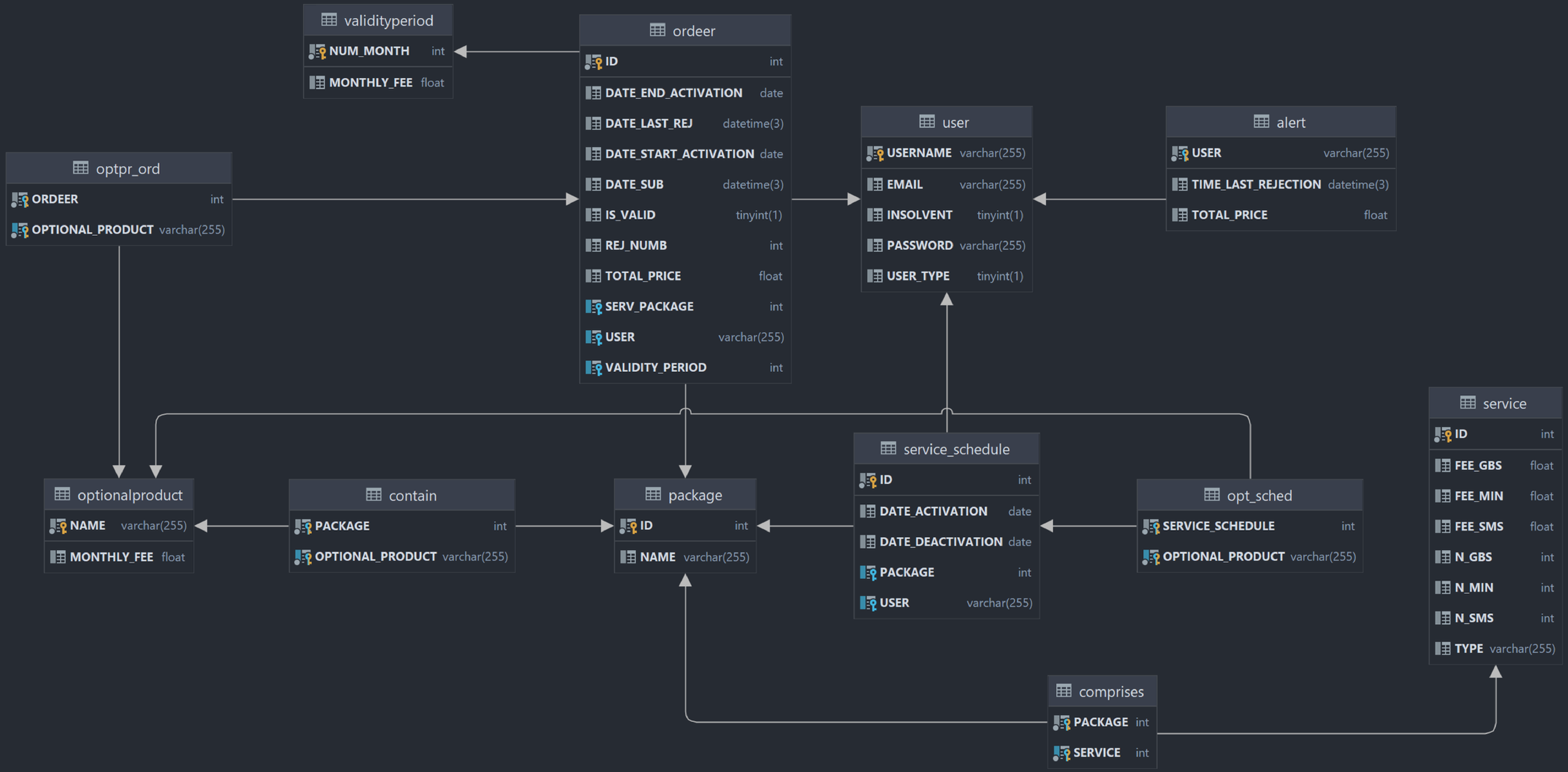
- Regarding the link between the Validity Periods and the Service Packages, an User can choose for each Service Package whichever ValidityPeriod.

E-R SCHEME AND LOGIC MODEL

ENTITY RELATIONSHIP



RELATIONAL MODEL



validityperiod	
NUM_MONTH	int
MONTHLY_FEE	float

ordeer	
ID	int
DATE_END_ACTIVATION	date
DATE_LAST_REJ	datetime(3)
DATE_START_ACTIVATION	date
DATE_SUB	datetime(3)
IS_VALID	tinyint(1)
REJ_NUMB	int
TOTAL_PRICE	float
SERV_PACKAGE	int
USER	varchar(255)
VALIDITY_PERIOD	int

user	
USERNAME	varchar(255)
EMAIL	varchar(255)
INSOLVENT	tinyint(1)
PASSWORD	varchar(255)
USER_TYPE	tinyint(1)

alert	
USER	varchar(255)
TIME_LAST_REJECTION	datetime(3)
TOTAL_PRICE	float

optpr_ord	
ORDEER	int
OPTIONAL_PRODUCT	varchar(255)

optionalproduct	
NAME	varchar(255)
MONTHLY_FEE	float

contain	
PACKAGE	int
OPTIONAL_PRODUCT	varchar(255)

package	
ID	int
NAME	varchar(255)

service_schedule	
ID	int
DATE_ACTIVATION	date
DATE_DEACTIVATION	date
PACKAGE	int
USER	varchar(255)

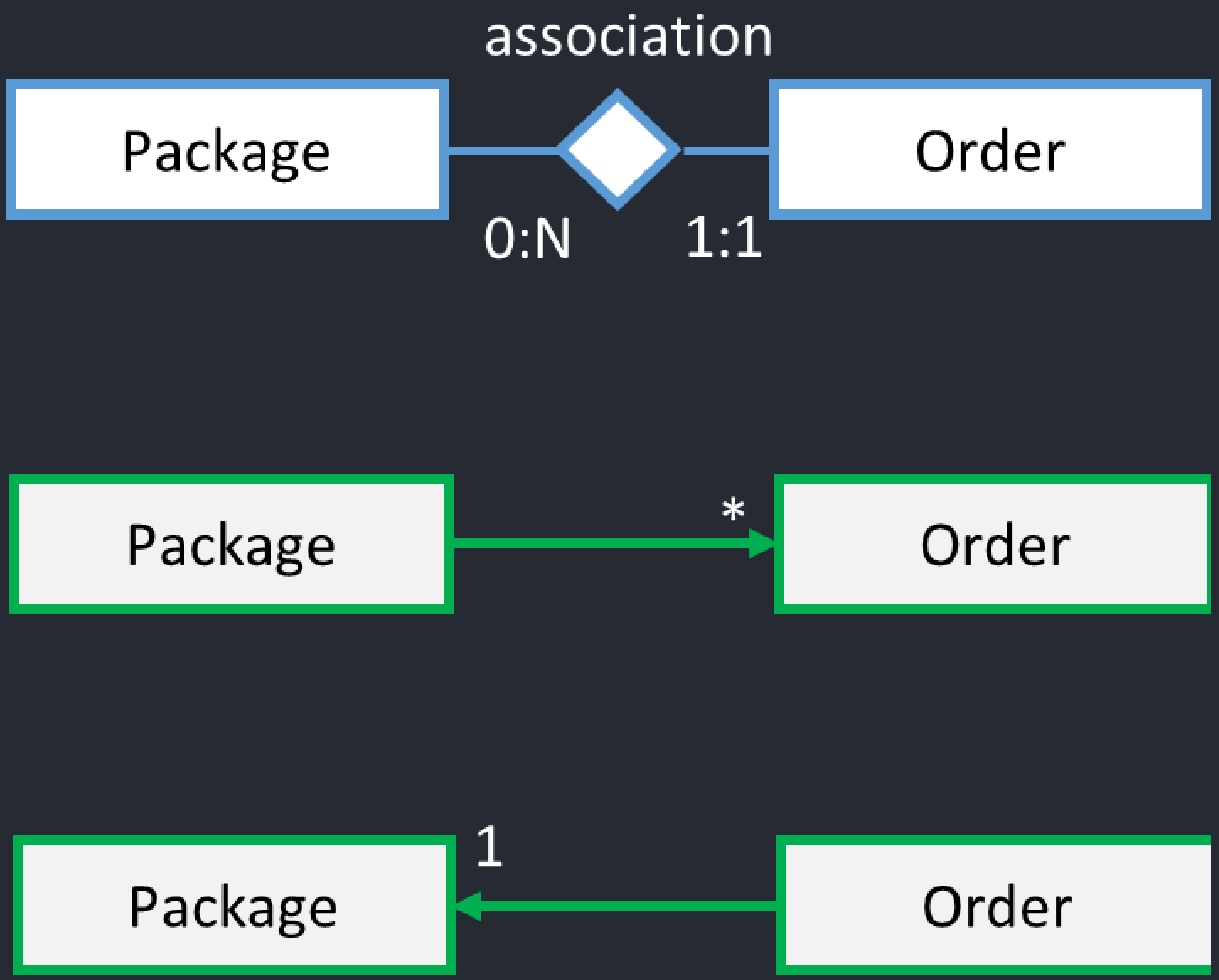
opt_sched	
SERVICE_SCHEDULE	int
OPTIONAL_PRODUCT	varchar(255)

service	
ID	int
FEE_GBS	float
FEE_MIN	float
FEE_SMS	float
N_GBS	int
N_MIN	int
N_SMS	int
TYPE	varchar(255)

comprises	
PACKAGE	int
SERVICE	int

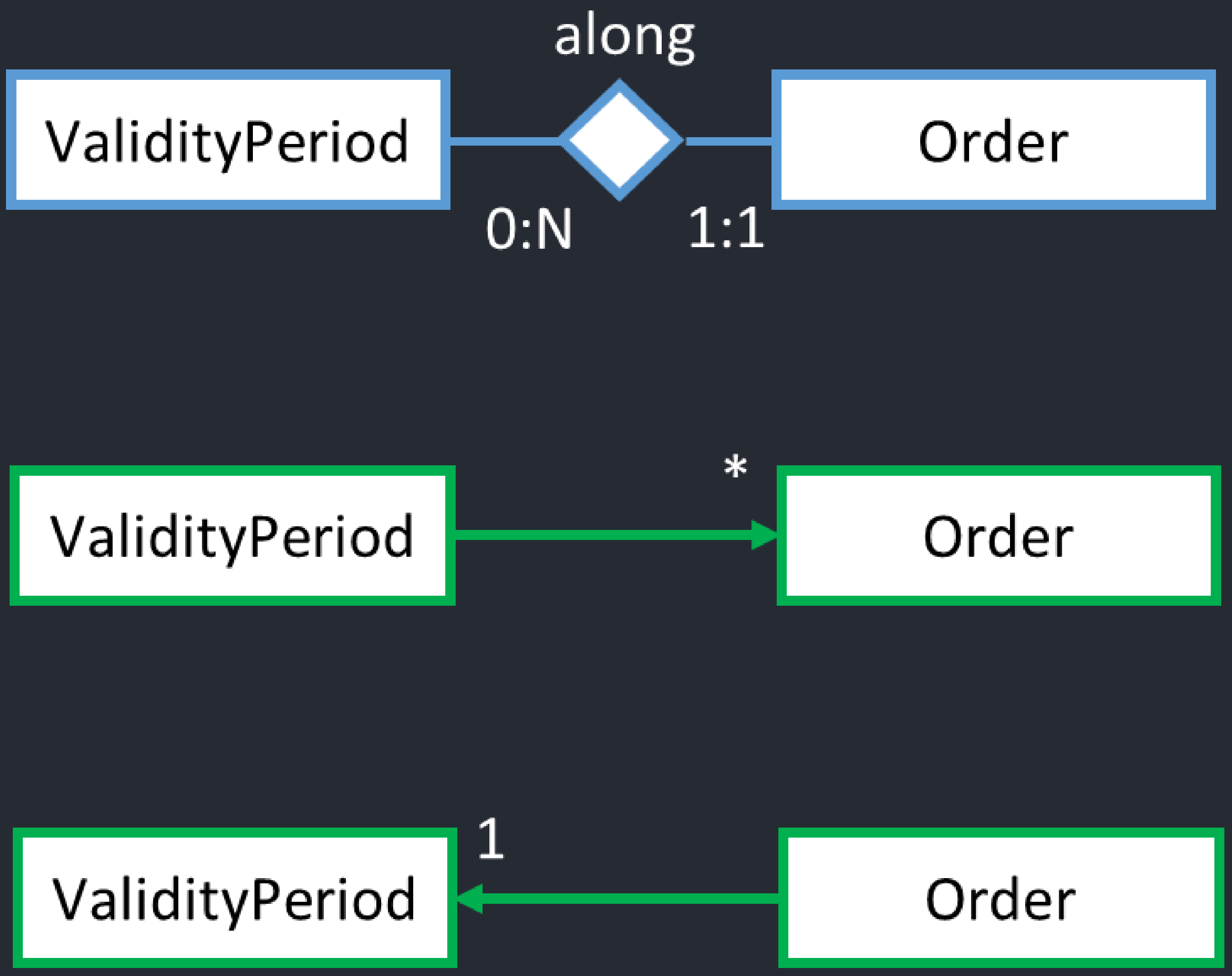
ORM DESIGN

RELATIONSHIP "ASSOCIATION"



- **Package → Order @OneToMany** is not necessary, we can map it for consistency
  - FetchType is LAZY
- **Order → Package @ManyToOne** is necessary to get the package chosen by the user
  - Owner = Order

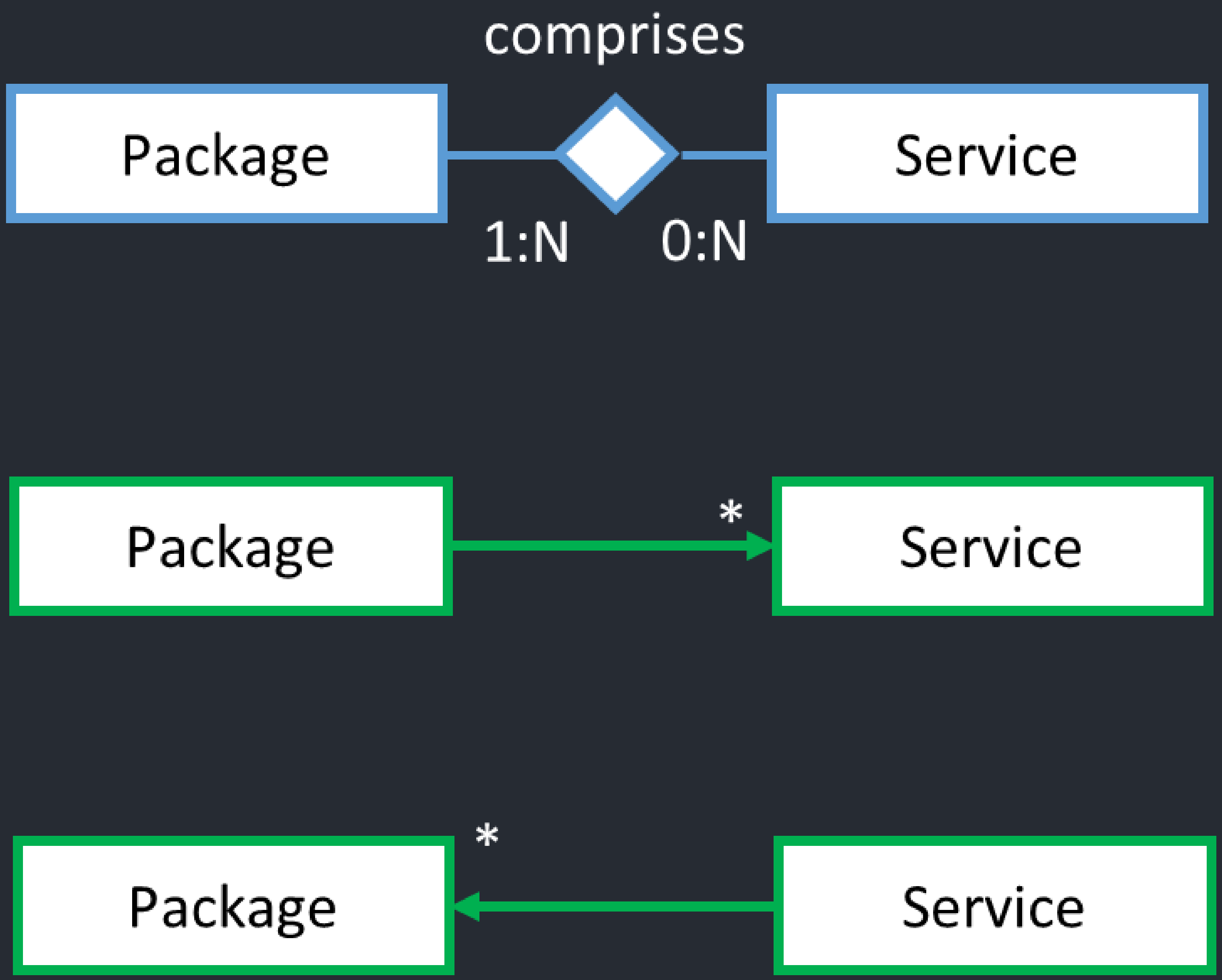
RELATIONSHIP "ALONG"



- **ValidityPeriod → Order**
  @OneToMany is not necessary

- **Order → ValidityPeriod**
  @ManyToOne is necessary to get the period chosen by the user for products and services
  - Owner = Order

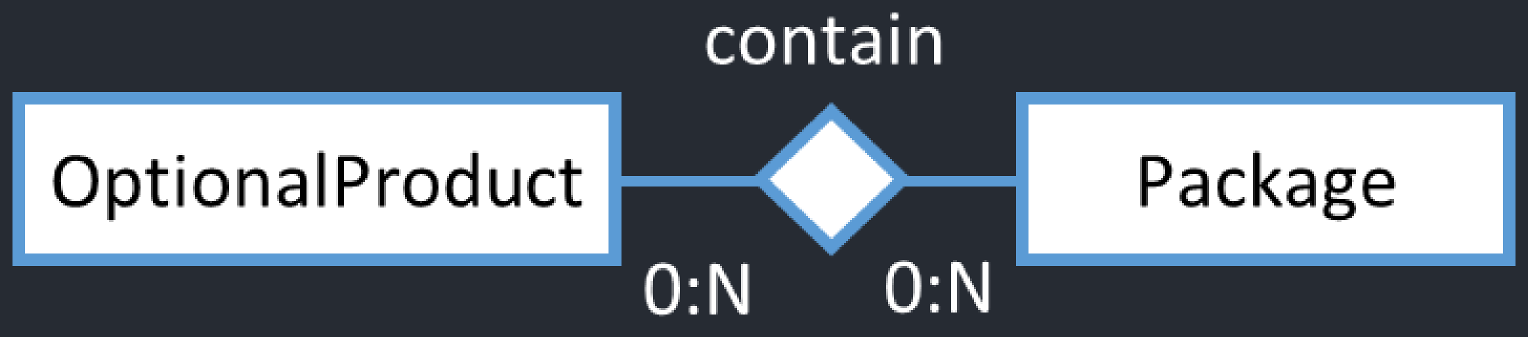
RELATIONSHIP "COMPRISES"

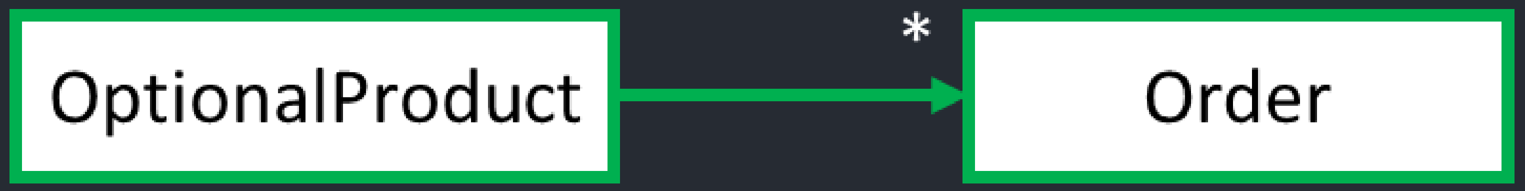


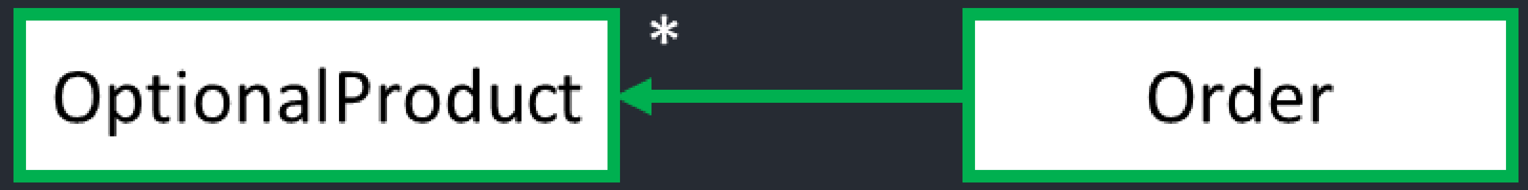
- **Package → Service**
  @ManyToMany is necessary to show the services related to the Package
  - Owner = Package
  - FetchType is EAGER to show to the User the Services offered in a Package when on Home Page.

- **Service → Package**
  @ManyToMany is not necessary

RELATIONSHIP "CONTAIN"



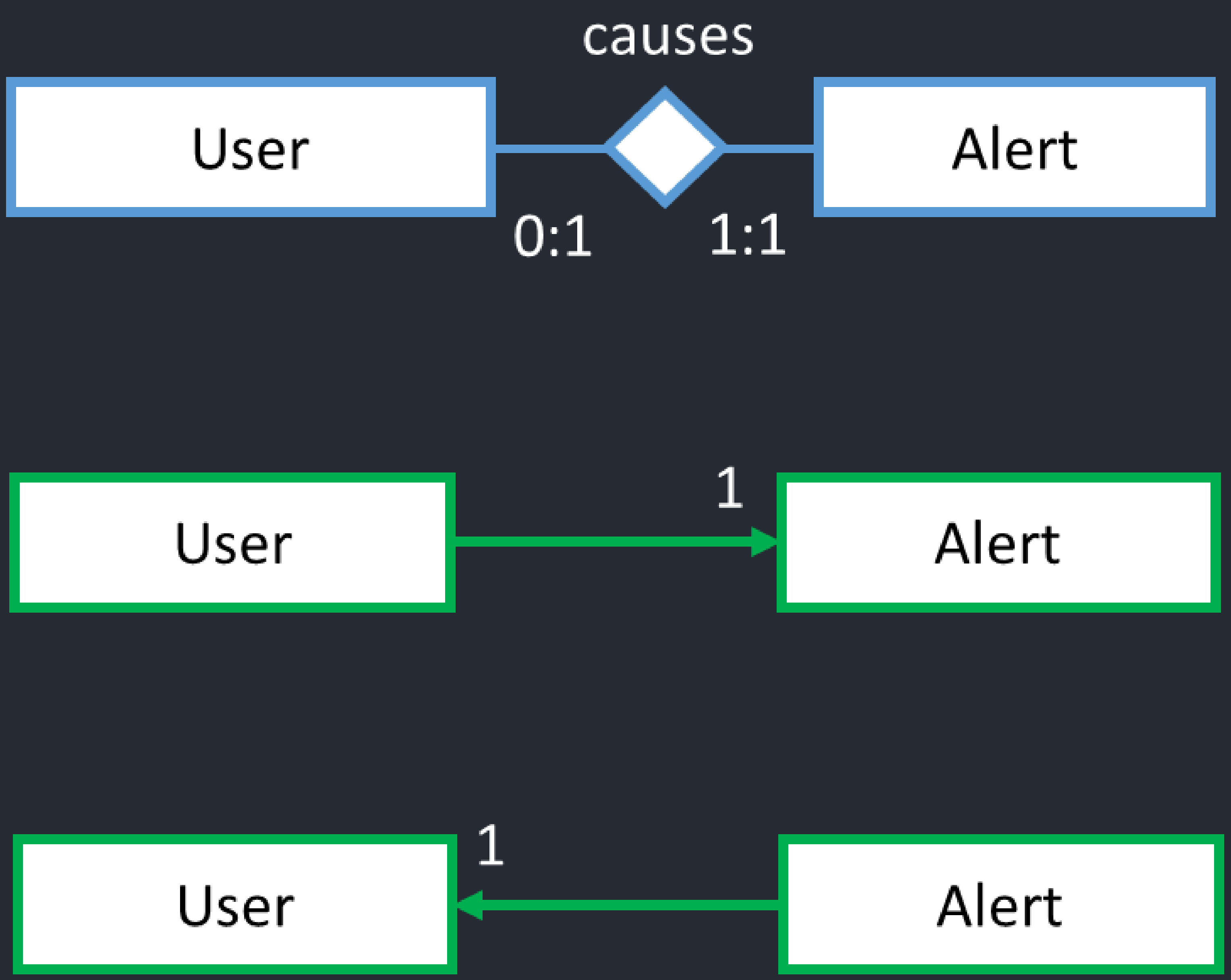




- **Package → OptionalProduct**
  @ManyToMany is necessary to get the optional products chosen by the user.
  - Owner can be any side (OptionalProduct or Package). In this case the Owner is the Package.
  - FetchType is EAGER because is necessary to retrieve the OptionalProducts in BuyService.

- **OptionalProduct → Order**
  @ManyToMany is not necessary.

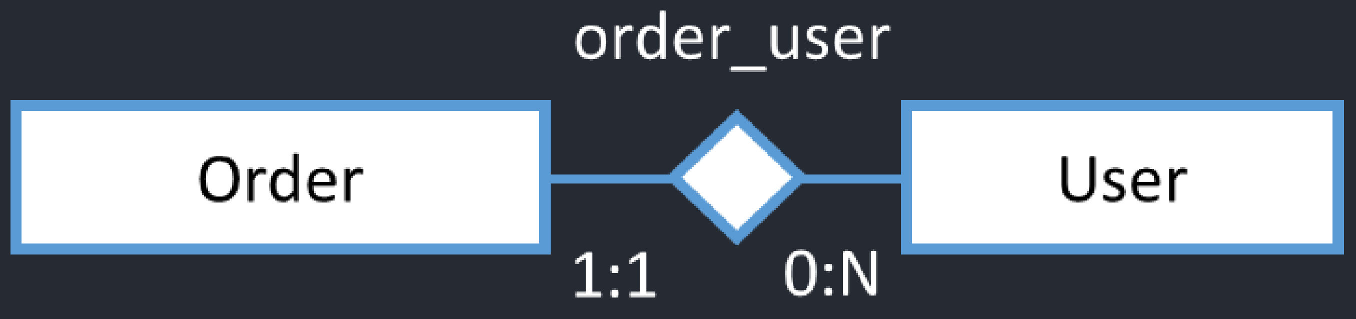
RELATIONSHIP "CAUSES"

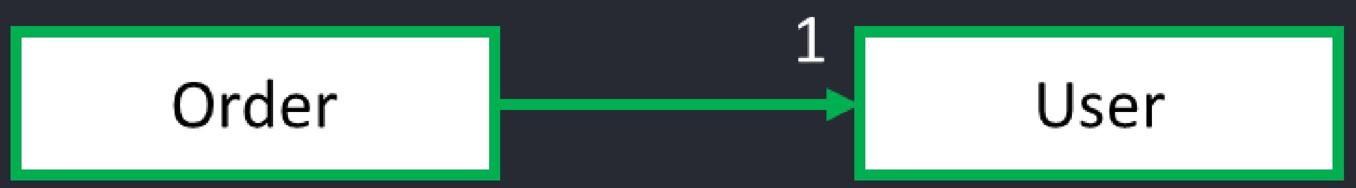


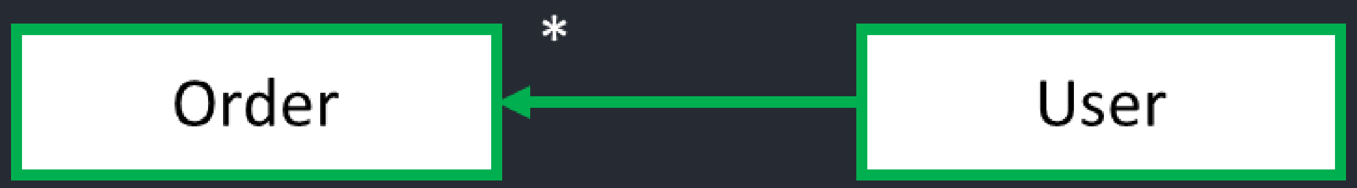
- **Alert → User @OneToOne** is necessary to keep track of the Alert for each User
  - Alert is a weak entity, so its pk is also a fk. It has a referential integrity constaint on Username on User.
  - Owner = Alert
  - FetchType is EAGER to show info related to the User.

- **User → Alert @OneToOne** is not necessary

RELATIONSHIP "ORDER_USER"

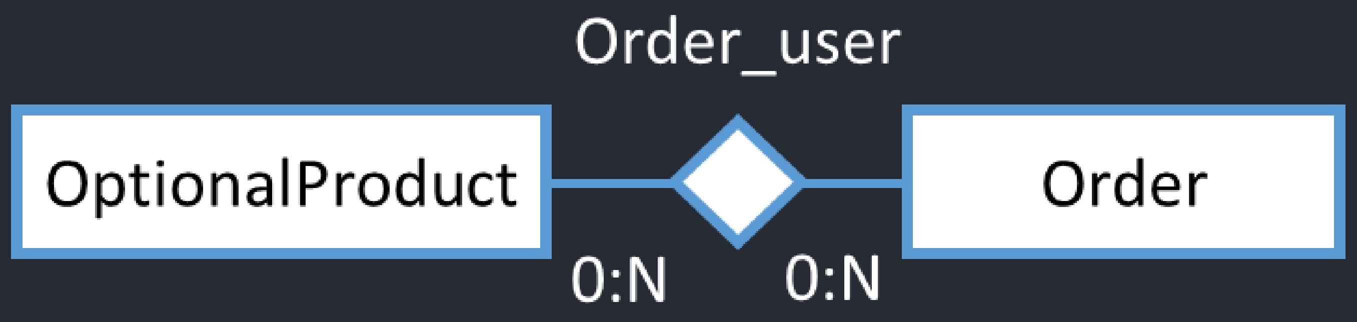


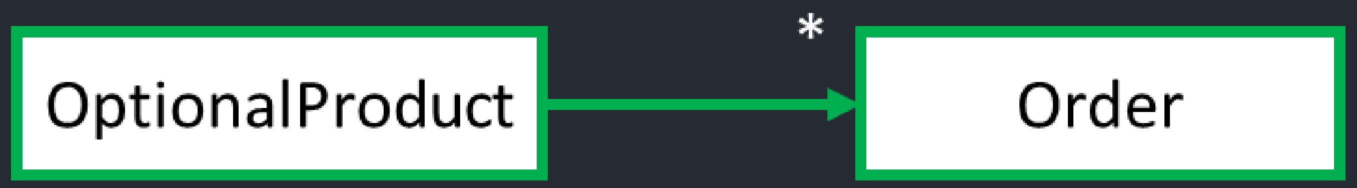


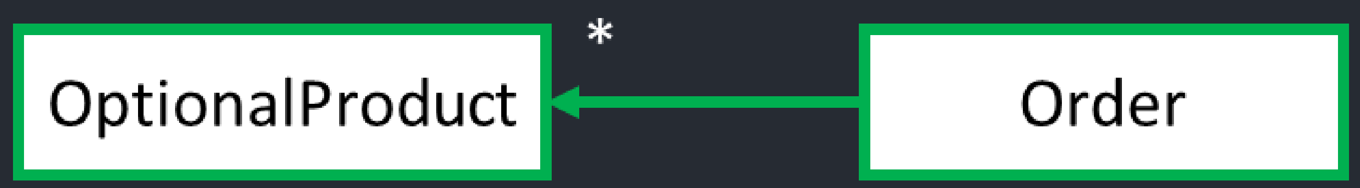


- **Order → User @ManyToOne** is necessary to get the User.
  - Owner = Order
- **User → Order @OneToMany** is not necessary because is necessary to show only the orders with is_valid flag set to false so with a named query is simpler, but it is mapped for consistency.

RELATIONSHIP "OPTPR_ORD"



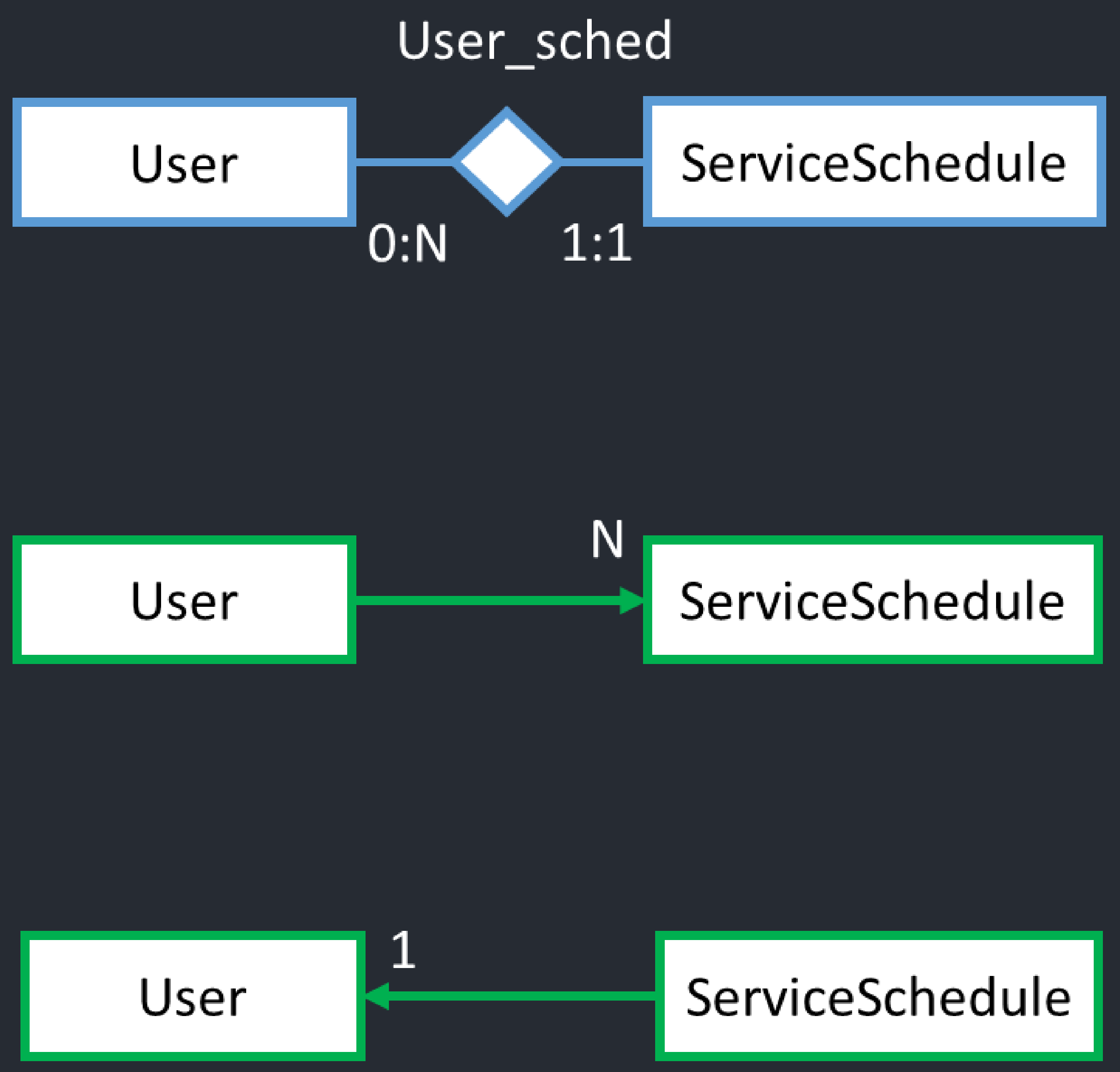




- **Order → OptionalProduct**
  @ManyToMany is necessary to get OptionalProducts related to the Order.
  - Owner = Order

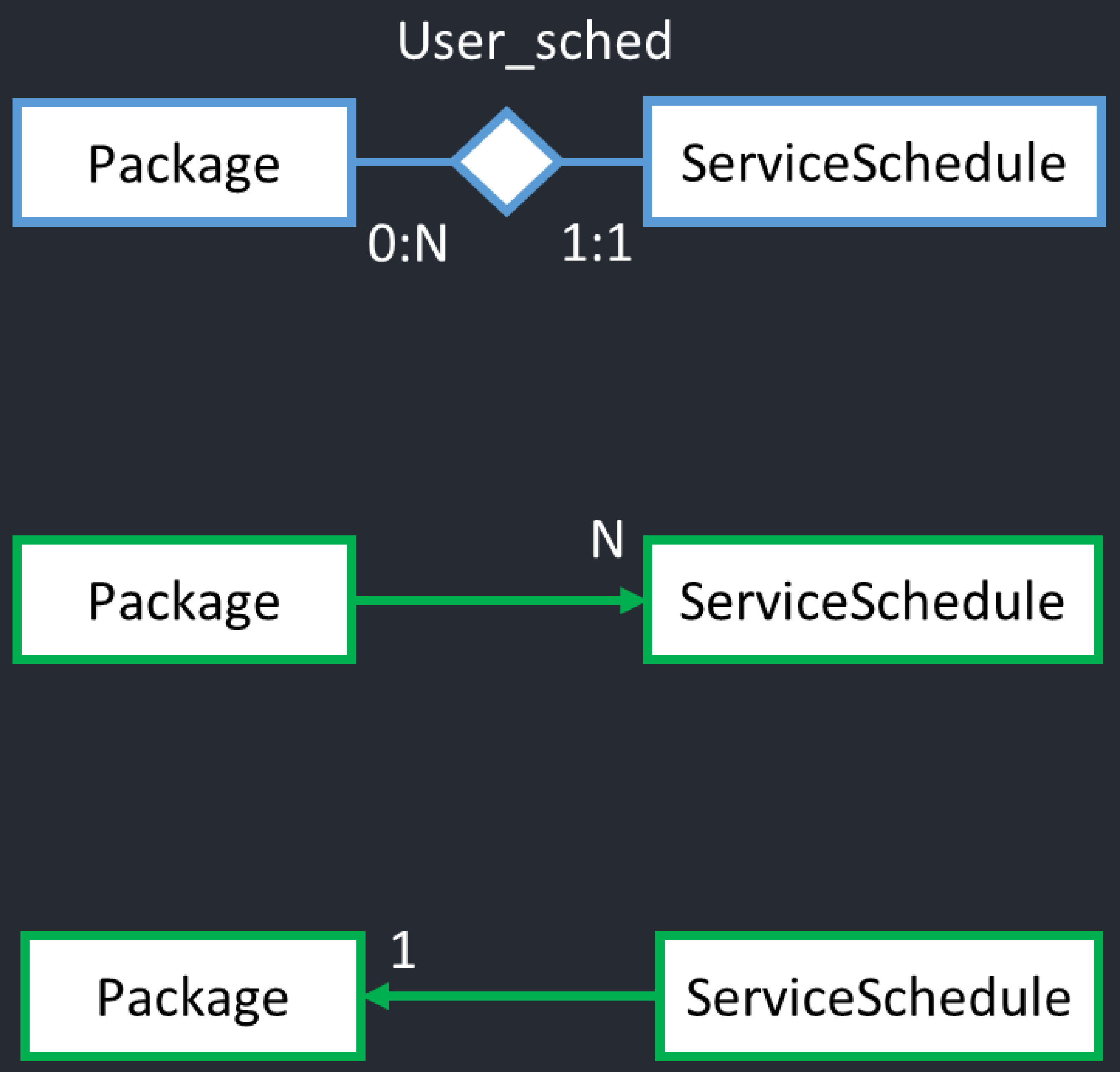
- **OptionalProduct → Order**
  @ManyToMany is not necessary.

RELATIONSHIP "USER_SCHED"



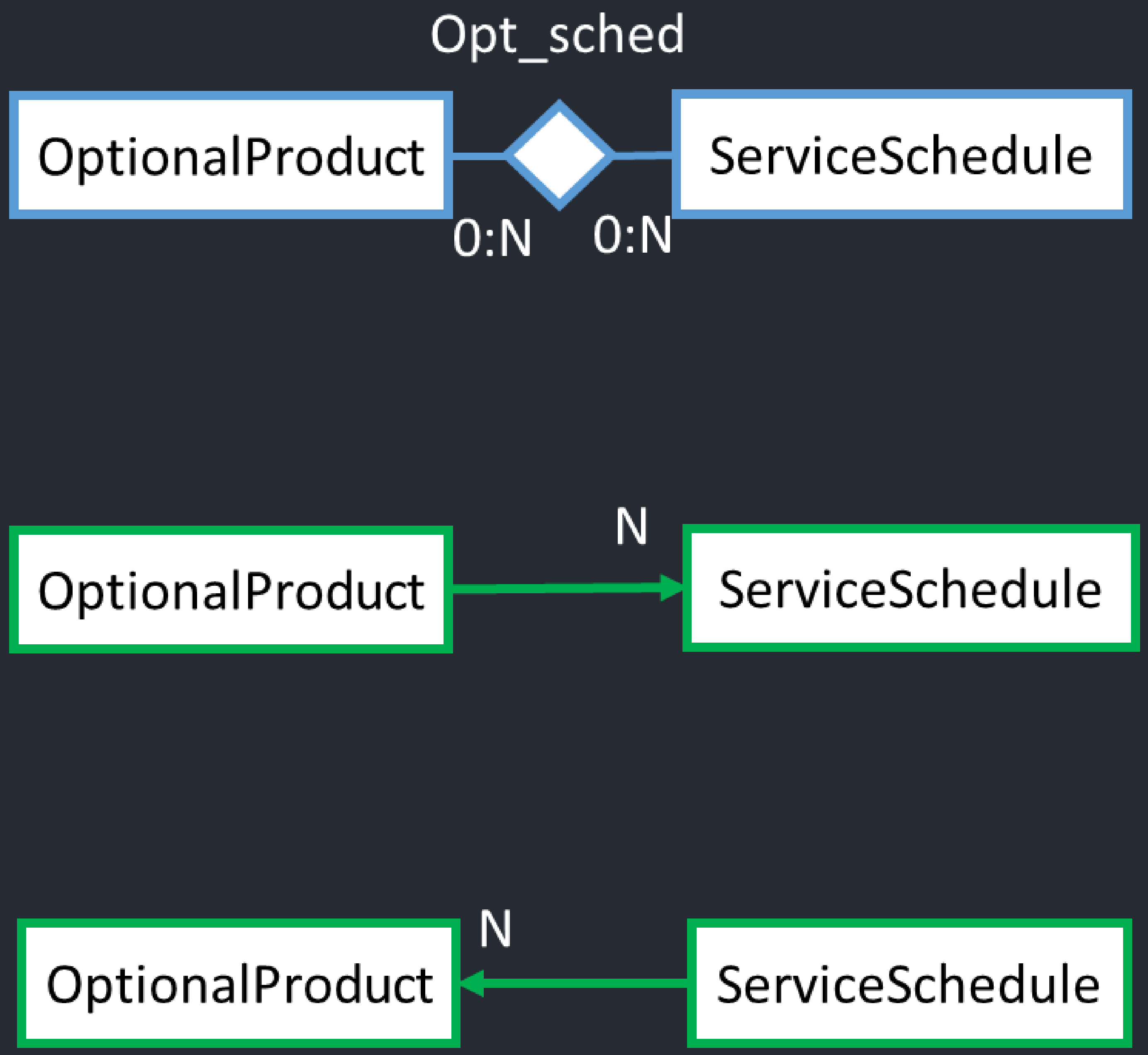
- **ServiceSchedule → User @ManyToOne.**
  - Owner = ServiceSchedule
  - FetchType is LAZY
- **User → ServiceSchedule @OneToMany** is not necessary.

RELATIONSHIP "PACK_SCHED"



- **ServiceSchedule → Package @ManyToOne**.
  - Owner = ServiceSchedule
  - FetchType is LAZY
- **Package → ServiceSchedule @OneToMany** is not necessary

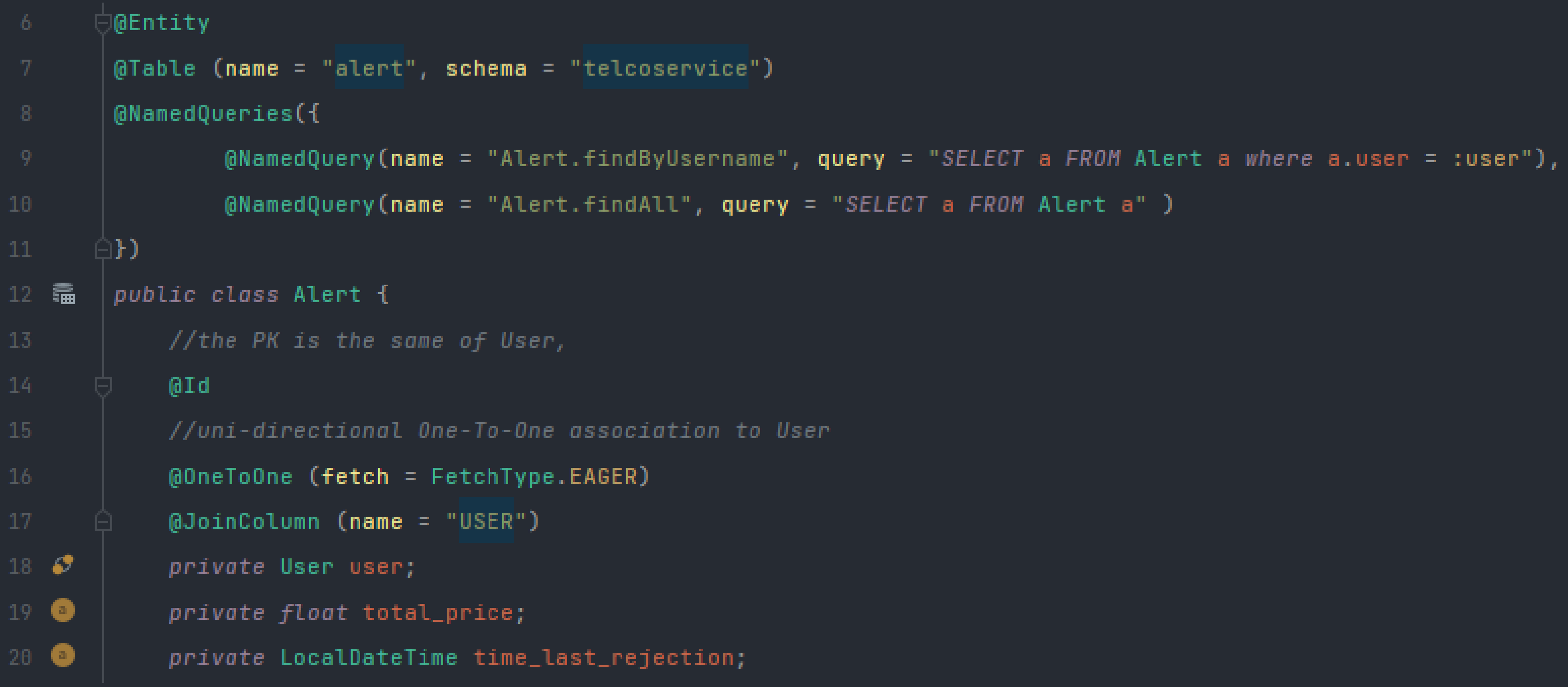
RELATIONSHIP "OPT_SCHED"



- **ServiceSchedule → OptionalProduct @ManyToOne**.
  - Owner = ServiceSchedule
  - FetchType is LAZY
- **OptionalProduct → ServiceSchedule @OneToMany** is not necessary.

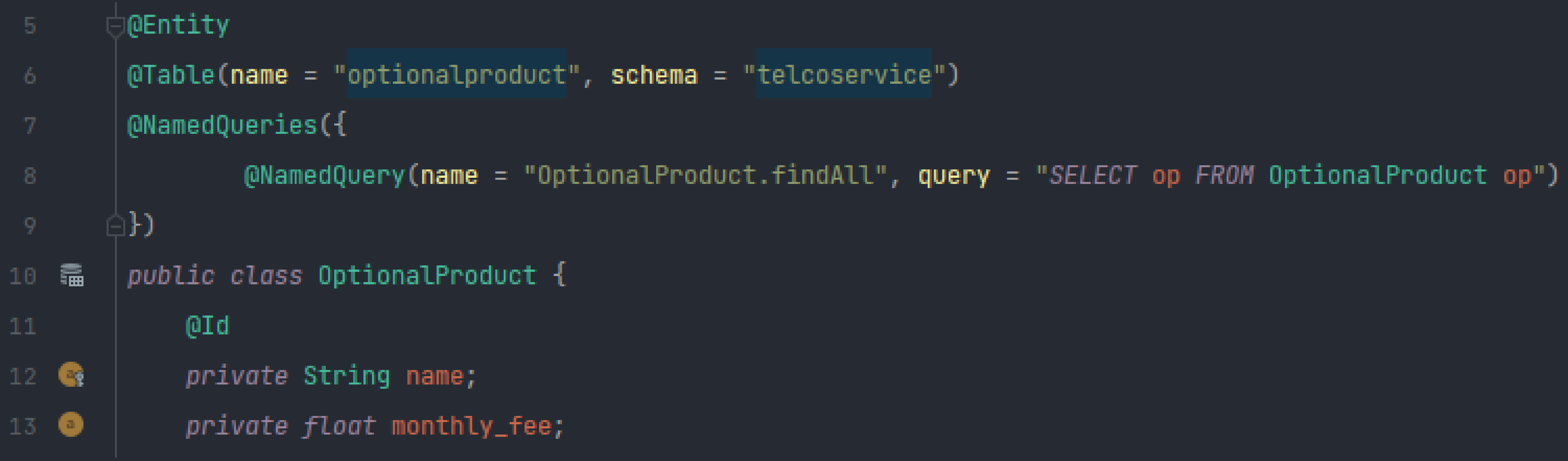
ENTITIES CODE

ENTITY ALERT



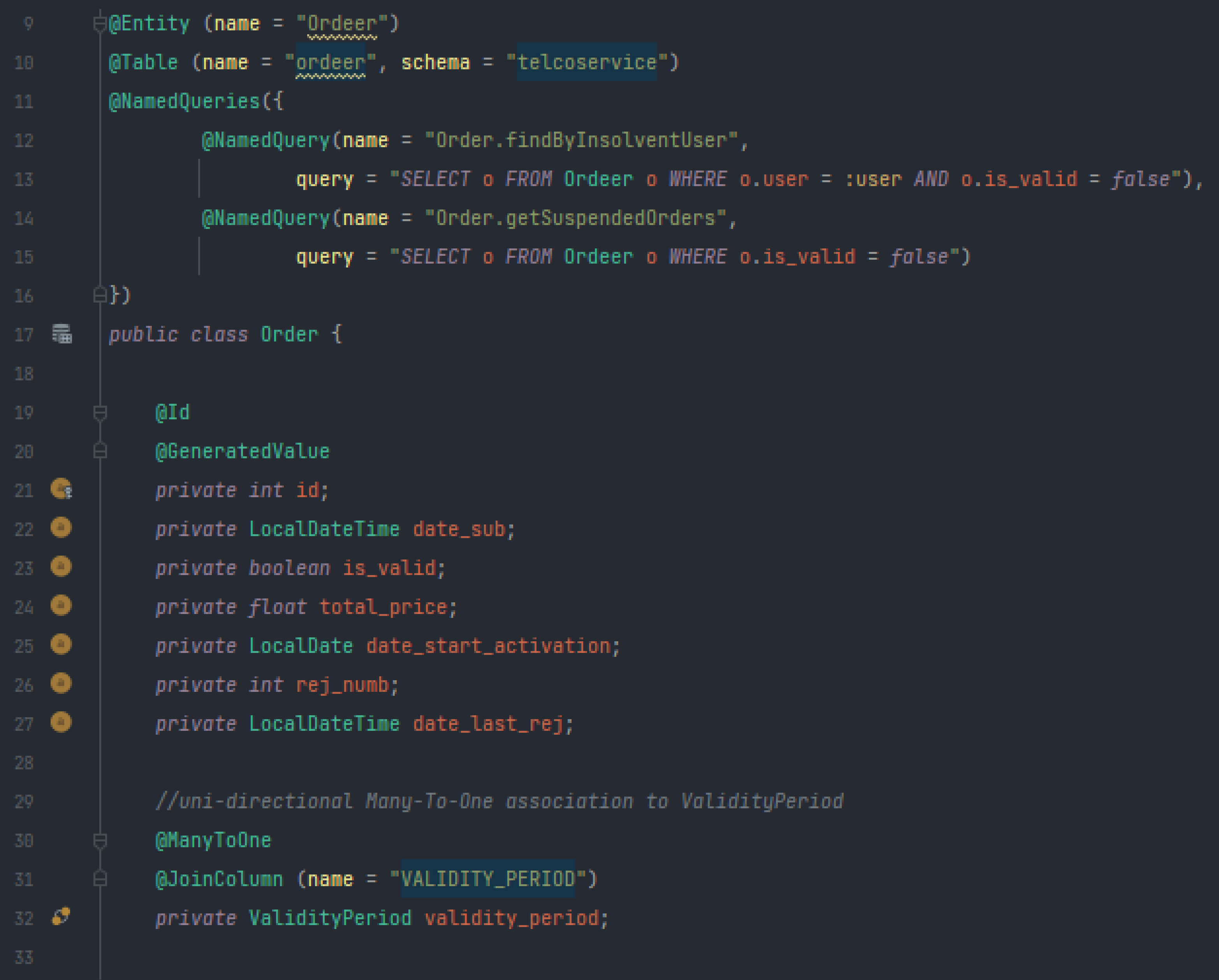
```java
@Entity
@Table (name = "alert", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "Alert.findByUsername", query = "SELECT a FROM Alert a where a.user = :user"),
        @NamedQuery(name = "Alert.findAll", query = "SELECT a FROM Alert a" )
})
public class Alert {
    //the PK is the same of User,
    @Id
    //uni-directional One-To-One association to User
    @OneToOne (fetch = FetchType.EAGER)
    @JoinColumn (name = "USER")
    private User user;
    private float total_price;
    private LocalDateTime time_last_rejection;
```

ENTITY OPTIONAL PRODUCT

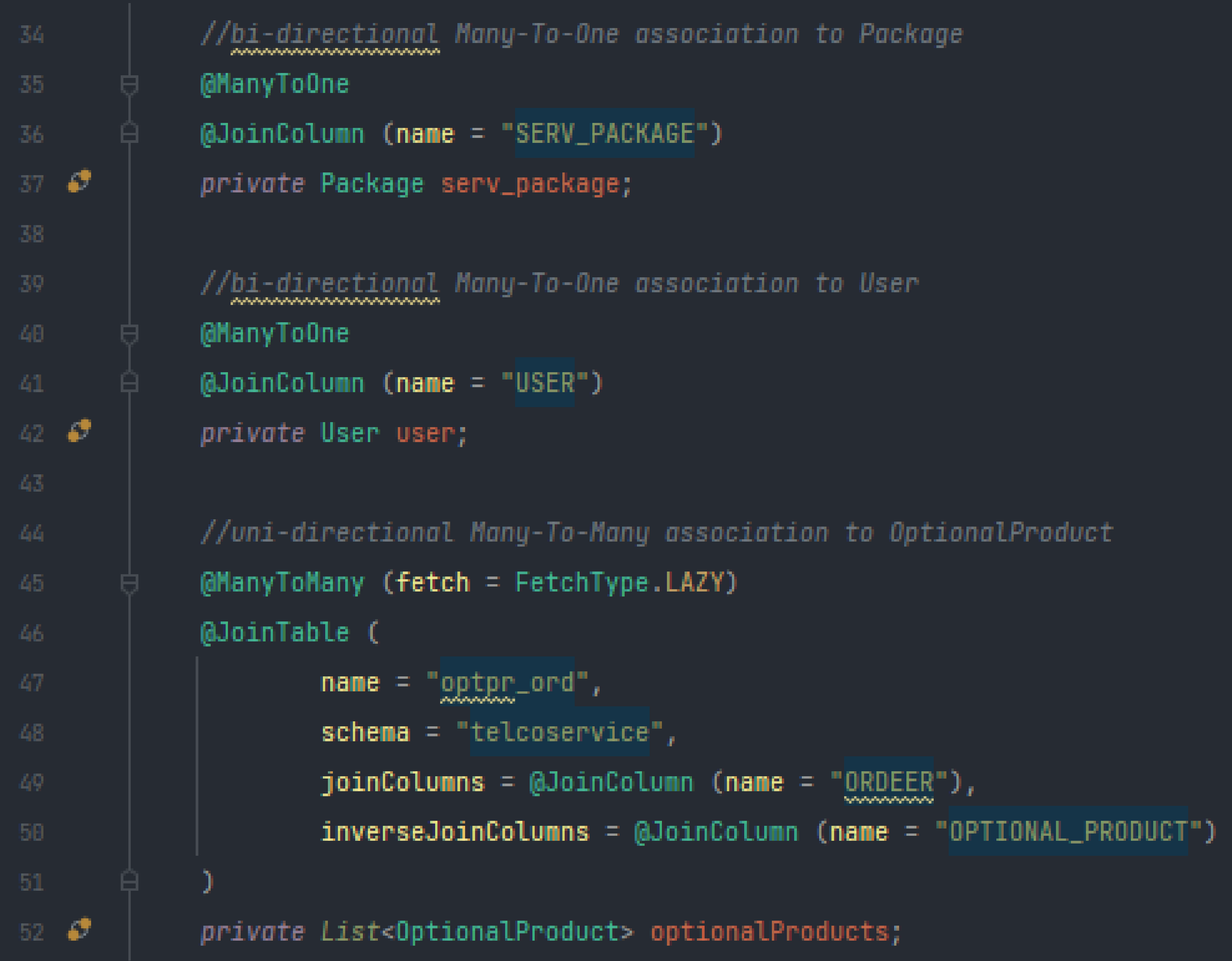


```java
@Entity
@Table(name = "optionalproduct", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "OptionalProduct.findAll", query = "SELECT op FROM OptionalProduct op")
})
public class OptionalProduct {
    @Id
    private String name;
    private float monthly_fee;
```

ENTITY ORDER

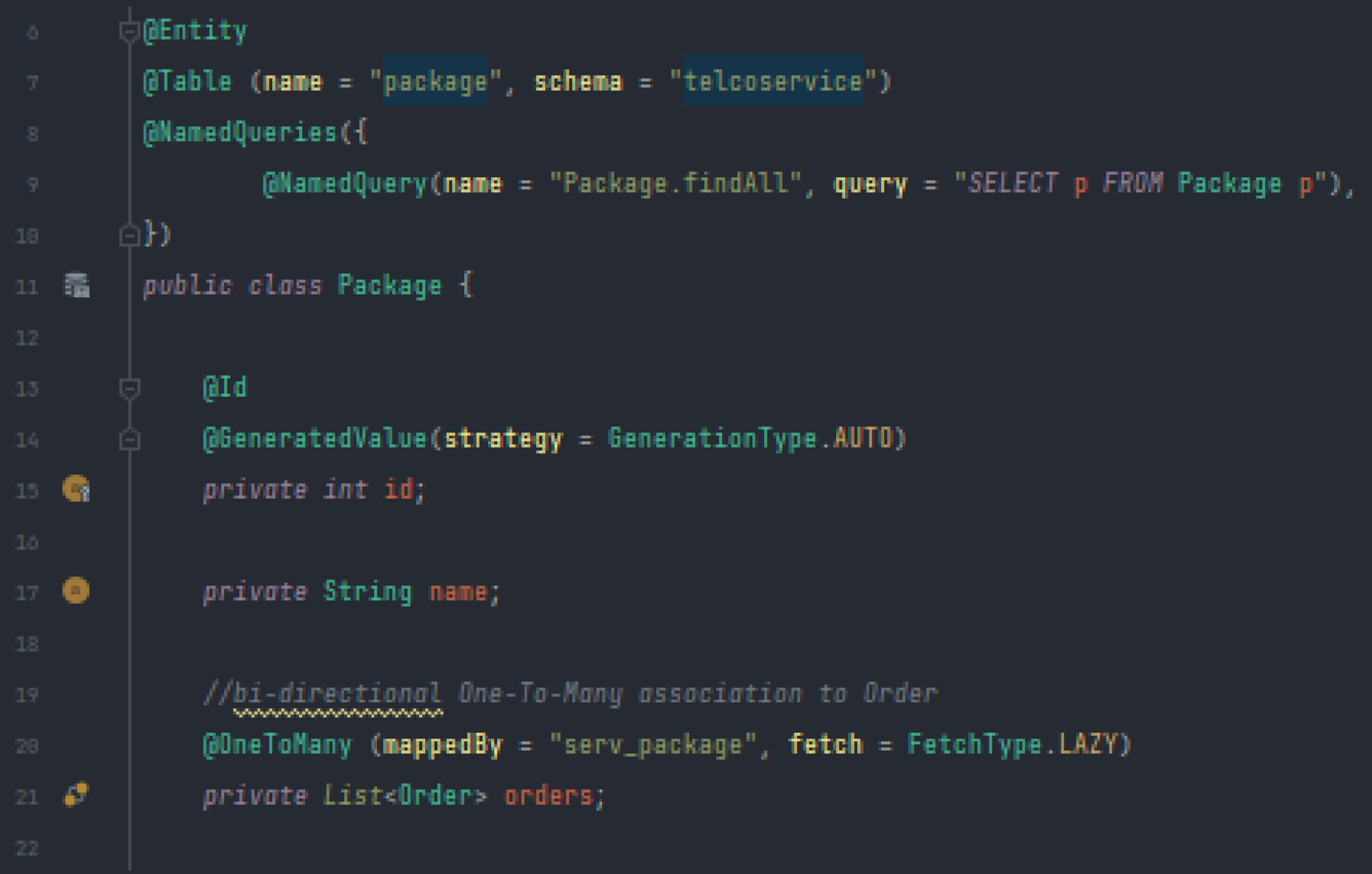


```java
@Entity (name = "Ordeer")
@Table (name = "ordeer", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "Order.findByInsolventUser",
                query = "SELECT o FROM Ordeer o WHERE o.user = :user AND o.is_valid = false"),
        @NamedQuery(name = "Order.getSuspendedOrders",
                query = "SELECT o FROM Ordeer o WHERE o.is_valid = false")
})
public class Order {

    @Id
    @GeneratedValue
    private int id;
    private LocalDateTime date_sub;
    private boolean is_valid;
    private float total_price;
    private LocalDate date_start_activation;
    private int rej_numb;
    private LocalDateTime date_last_rej;

    //uni-directional Many-To-One association to ValidityPeriod
    @ManyToOne
    @JoinColumn (name = "VALIDITY_PERIOD")
    private ValidityPeriod validity_period;

```

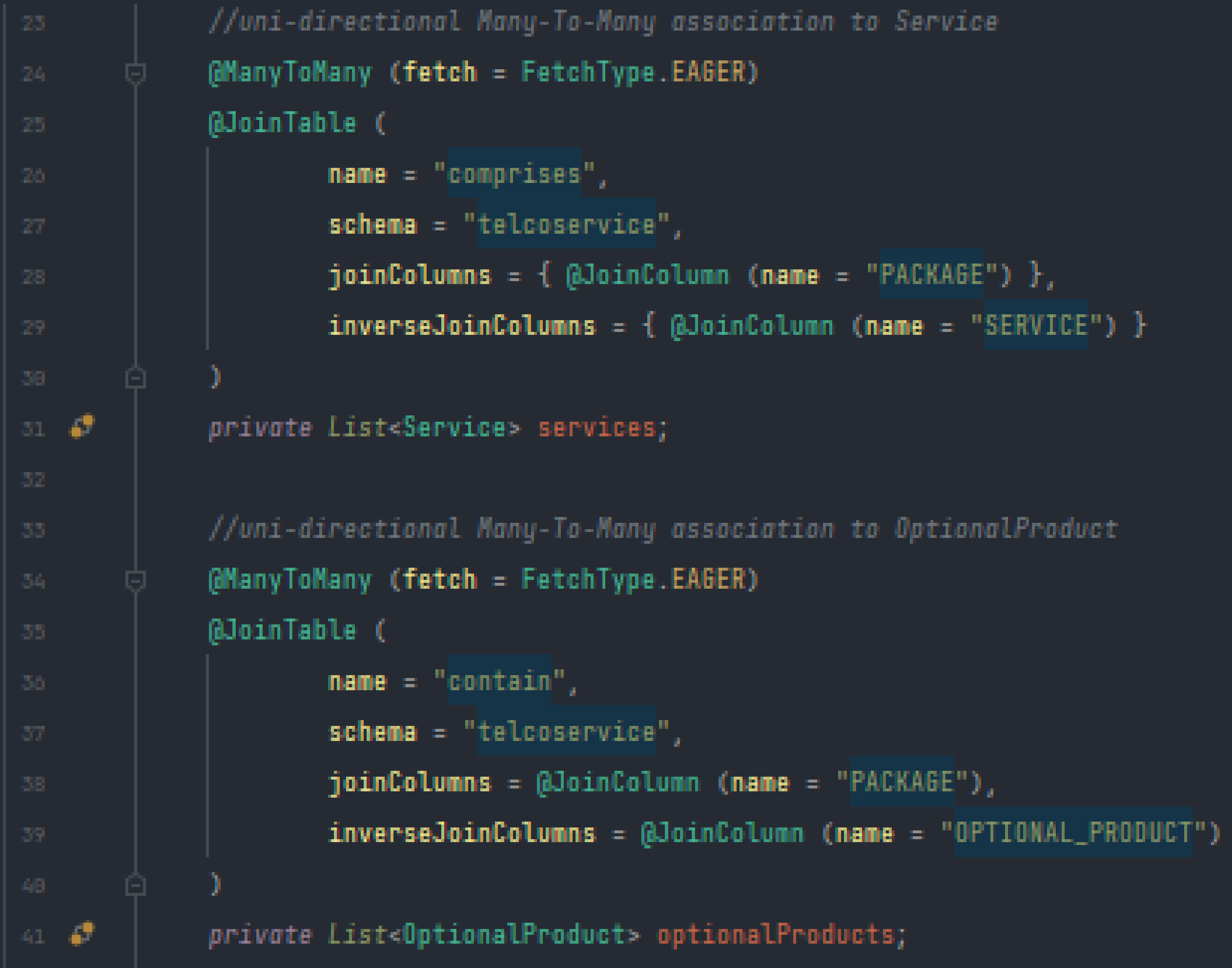


```java
    //bi-directional Many-To-One association to Package
    @ManyToOne
    @JoinColumn (name = "SERV_PACKAGE")
    private Package serv_package;

    //bi-directional Many-To-One association to User
    @ManyToOne
    @JoinColumn (name = "USER")
    private User user;

    //uni-directional Many-To-Many association to OptionalProduct
    @ManyToMany (fetch = FetchType.LAZY)
    @JoinTable (
            name = "optpr_ord",
            schema = "telcoservice",
            joinColumns = @JoinColumn (name = "ORDEER"),
            inverseJoinColumns = @JoinColumn (name = "OPTIONAL_PRODUCT")
    )
    private List<OptionalProduct> optionalProducts;
```

ENTITY PACKAGE

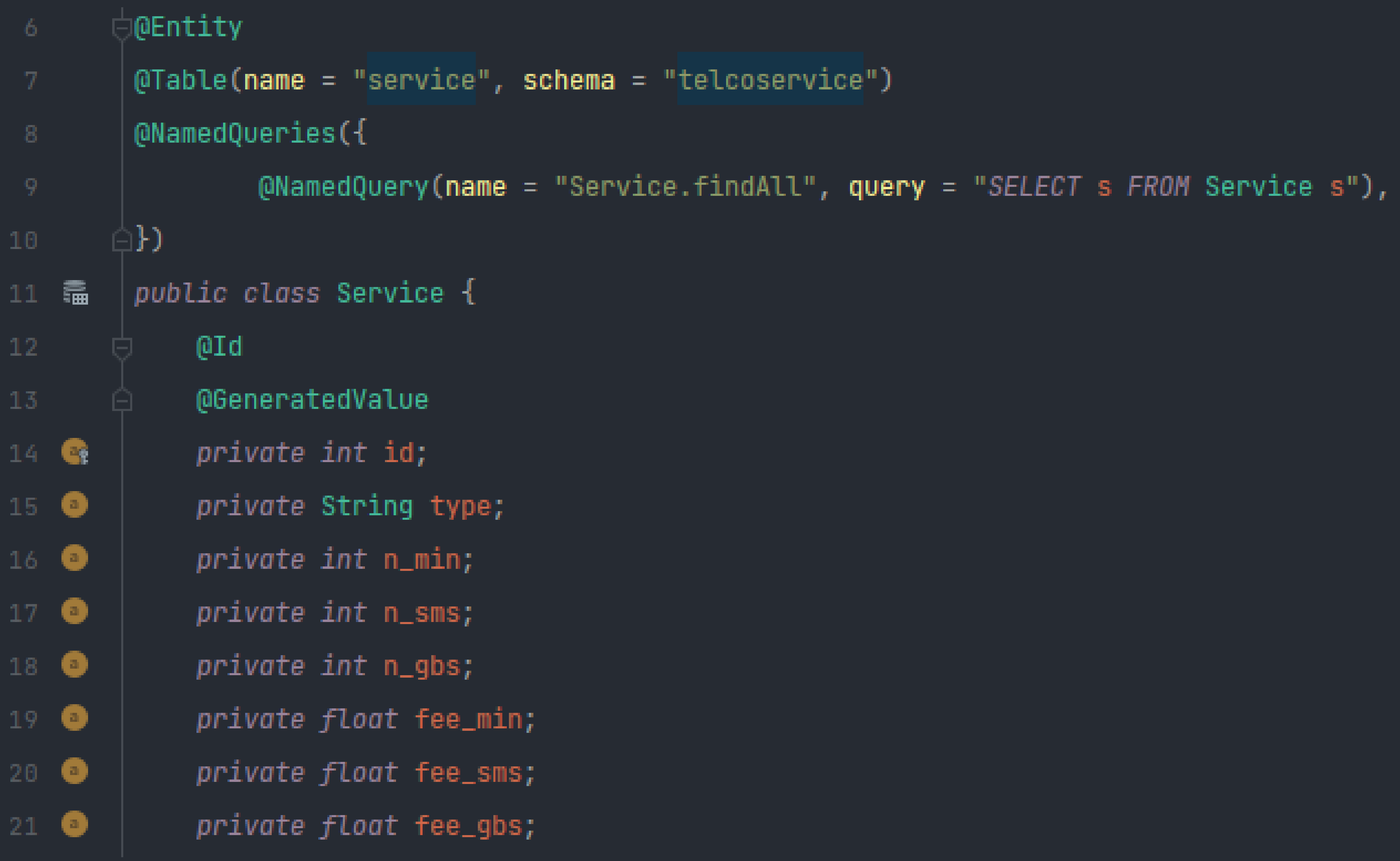


```java
@Entity
@Table (name = "package", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "Package.findAll", query = "SELECT p FROM Package p"),
})
public class Package {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    private String name;

    //bi-directional One-To-Many association to Order
    @OneToMany (mappedBy = "serv_package", fetch = FetchType.LAZY)
    private List<Order> orders;

```

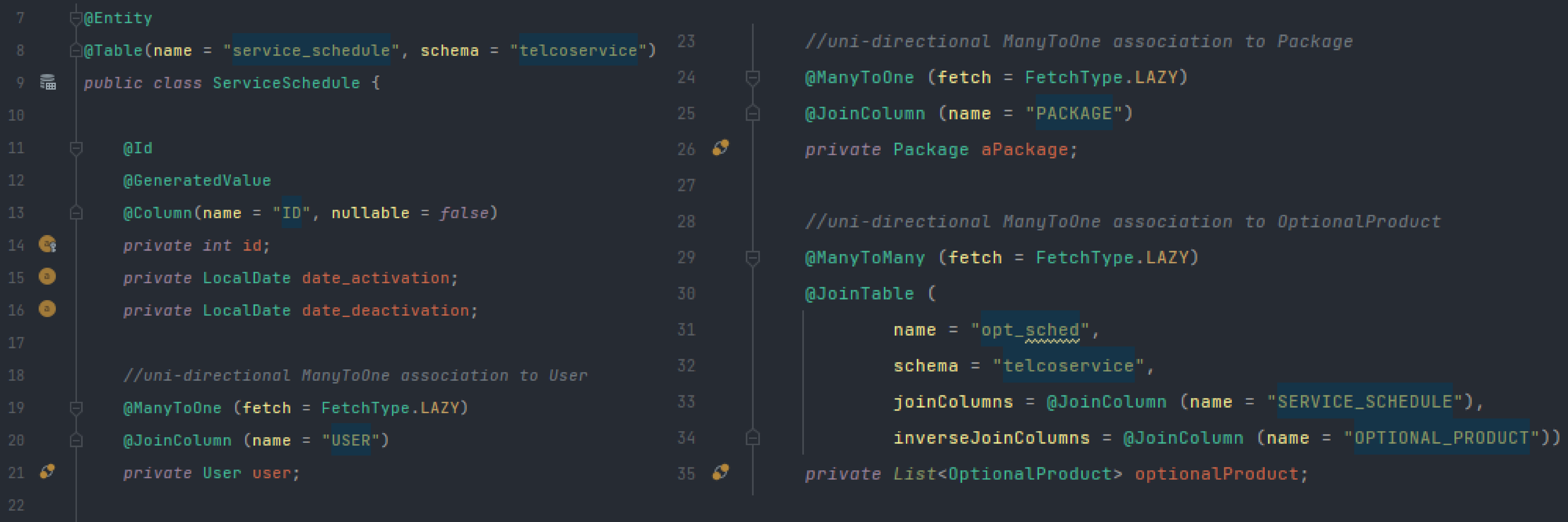


```java
    //uni-directional Many-To-Many association to Service
    @ManyToMany (fetch = FetchType.EAGER)
    @JoinTable (
            name = "comprises",
            schema = "telcoservice",
            joinColumns = { @JoinColumn (name = "PACKAGE") },
            inverseJoinColumns = { @JoinColumn (name = "SERVICE") }
    )
    private List<Service> services;

    //uni-directional Many-To-Many association to OptionalProduct
    @ManyToMany (fetch = FetchType.EAGER)
    @JoinTable (
            name = "contain",
            schema = "telcoservice",
            joinColumns = @JoinColumn (name = "PACKAGE"),
            inverseJoinColumns = @JoinColumn (name = "OPTIONAL_PRODUCT")
    )
    private List<OptionalProduct> optionalProducts;
```

ENTITY SERVICE

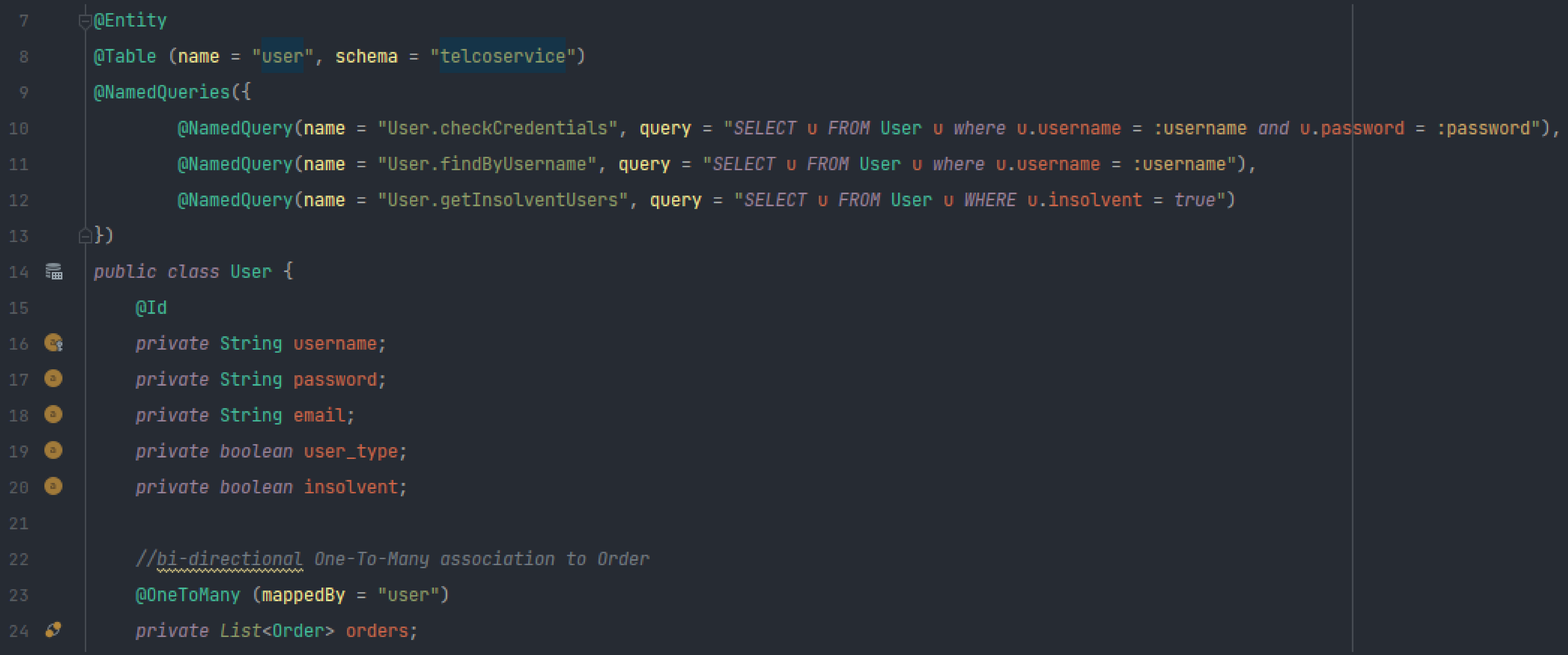

```java
@Entity
@Table(name = "service", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "Service.findAll", query = "SELECT s FROM Service s"),
})
public class Service {
    @Id
    @GeneratedValue
    private int id;
    private String type;
    private int n_min;
    private int n_sms;
    private int n_gbs;
    private float fee_min;
    private float fee_sms;
    private float fee_gbs;
```

ENTITY SERVICE SCHEDULE



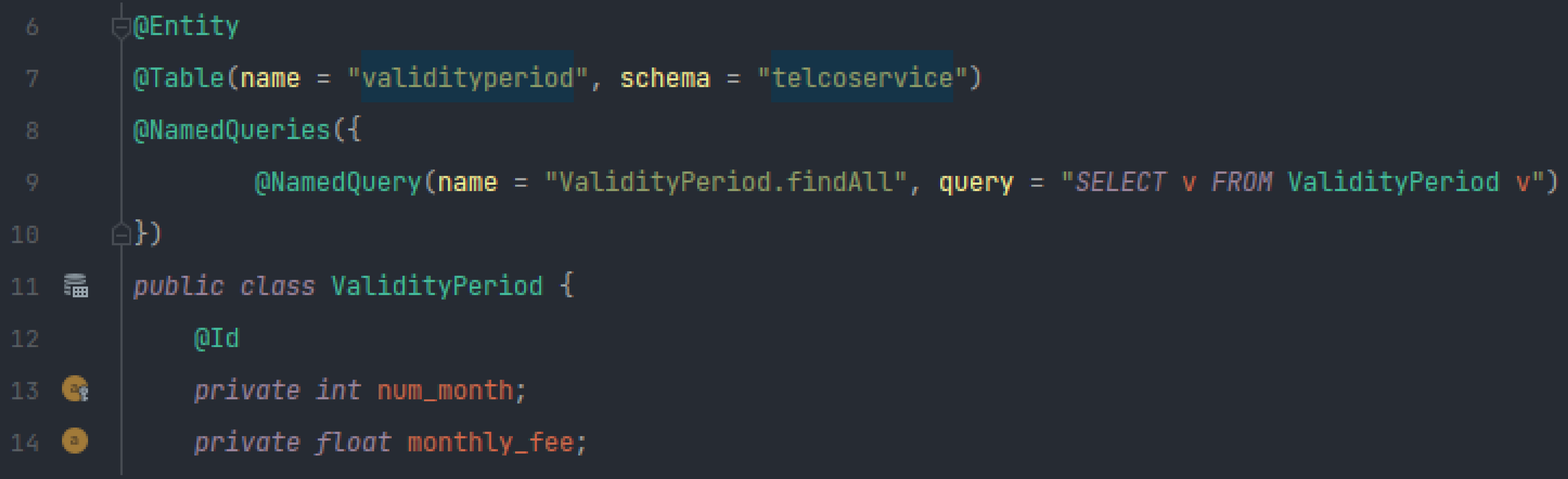
```java
@Entity
@Table(name = "service_schedule", schema = "telcoservice")
public class ServiceSchedule {

    @Id
    @GeneratedValue
    @Column(name = "ID", nullable = false)
    private int id;
    private LocalDate date_activation;
    private LocalDate date_deactivation;

    //uni-directional ManyToOne association to User
    @ManyToOne (fetch = FetchType.LAZY)
    @JoinColumn (name = "USER")
    private User user;

    //uni-directional ManyToOne association to Package
    @ManyToOne (fetch = FetchType.LAZY)
    @JoinColumn (name = "PACKAGE")
    private Package aPackage;

    //uni-directional ManyToOne association to OptionalProduct
    @ManyToMany (fetch = FetchType.LAZY)
    @JoinTable (
            name = "opt_sched",
            schema = "telcoservice",
            joinColumns = @JoinColumn (name = "SERVICE_SCHEDULE"),
            inverseJoinColumns = @JoinColumn (name = "OPTIONAL_PRODUCT"))
    private List<OptionalProduct> optionalProduct;
```

ENTITY USER



```java
@Entity
@Table (name = "user", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "User.checkCredentials", query = "SELECT u FROM User u where u.username = :username and u.password = :password"),
        @NamedQuery(name = "User.findByUsername", query = "SELECT u FROM User u where u.username = :username"),
        @NamedQuery(name = "User.getInsolventUsers", query = "SELECT u FROM User u WHERE u.insolvent = true")
})
public class User {
    @Id
    private String username;
    private String password;
    private String email;
    private boolean user_type;
    private boolean insolvent;

    //bi-directional One-To-Many association to Order
    @OneToMany (mappedBy = "user")
    private List<Order> orders;
```

ENTITY VALIDITYPERIOD



```java
@Entity
@Table(name = "validityperiod", schema = "telcoservice")
@NamedQueries({
        @NamedQuery(name = "ValidityPeriod.findAll", query = "SELECT v FROM ValidityPeriod v")
})
public class ValidityPeriod {
    @Id
    private int num_month;
    private float monthly_fee;
```

TRIGGER
DESIGN AND CODE

TRIGGER MOTIVATION

- order_AFTER_INSERT:

  after the insertion of new Order, the trigger
  - creates a new record in Alert table if it doesn't exist an alert for the current user and it has more than 3 pending payments, otherwise it updates the alert.
  - creates a new record in the report table related to Package&ValidityPeriod if it doesn't exist yet, otherwise it updates "numb_purchase" field adding 1
  - creates a new record in the report table related to Package if it doesn't exist yet, otherwise it updates the row. "total_value_sales_without_op" field is set, both for the packages sold with optional products and for those sold without them.

- order_AFTER_UPDATE:

  after the update of an Order, the trigger:
  - creates a new record in Alert table if it doesn't exist an alert for the current user and it has more than 3 pending payments, otherwise it updates the alert if the user still has 3 pending payments or delete the row if it hasn't.
  - updates the "insolvent" flag to false if the user has no longer pending payments.

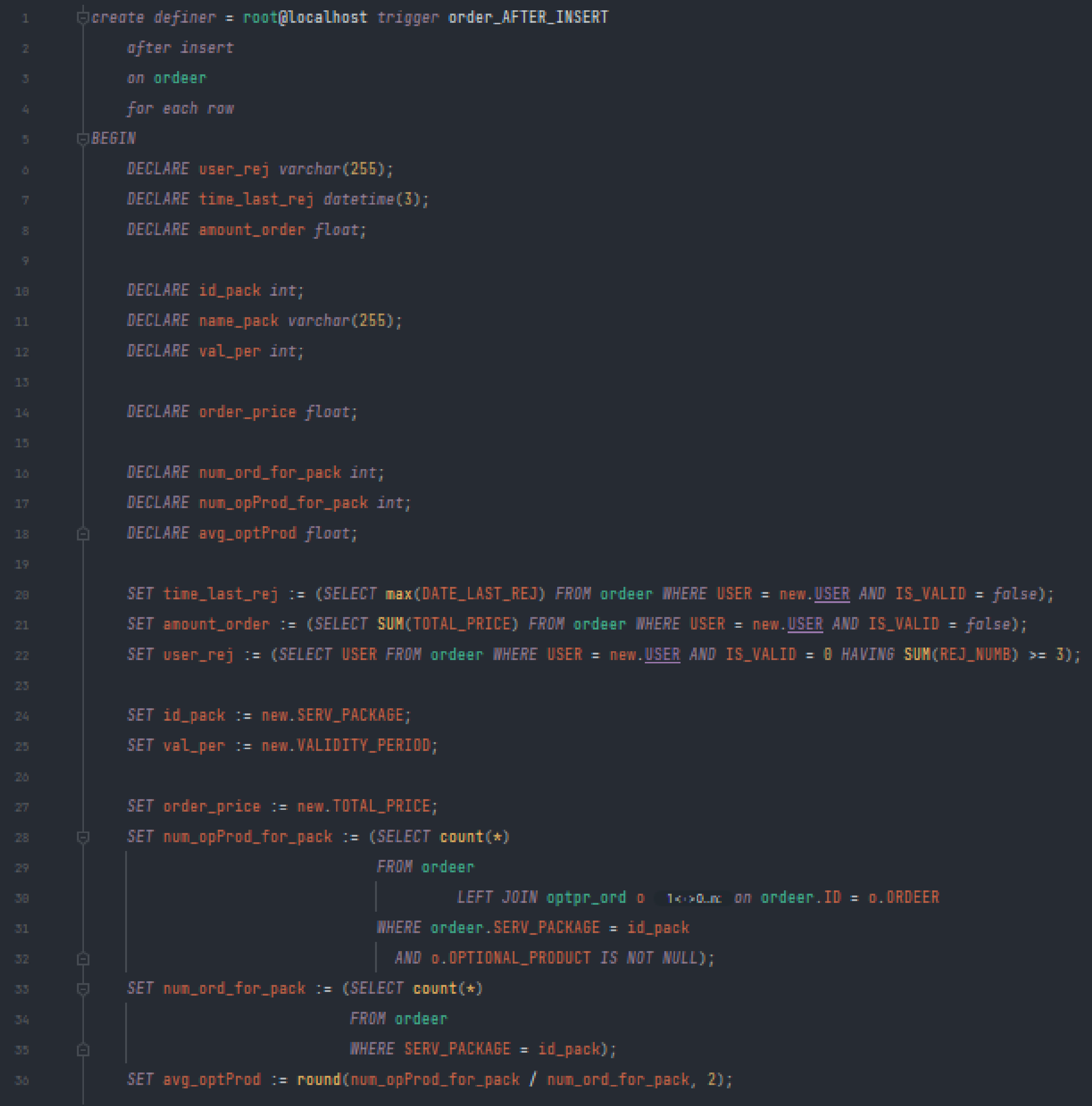
- optpr_ord_AFTER_INSERT:

  after an insertion of a tuple in "optpr_ord" table, the trigger:
  - creates a new record in the report table related to Optional Products if it doesn't exists yet, otherwise it updates "total_values_sales" field adding the current price.
  - if the row exists in the report table related to Packages, it updates it subtracting the value of the order from "total_value_sales_without_op" and adding it to "total_value_with_op" if it is the first insertion, otherwise it updates only the "avg_optionalproduct" field.

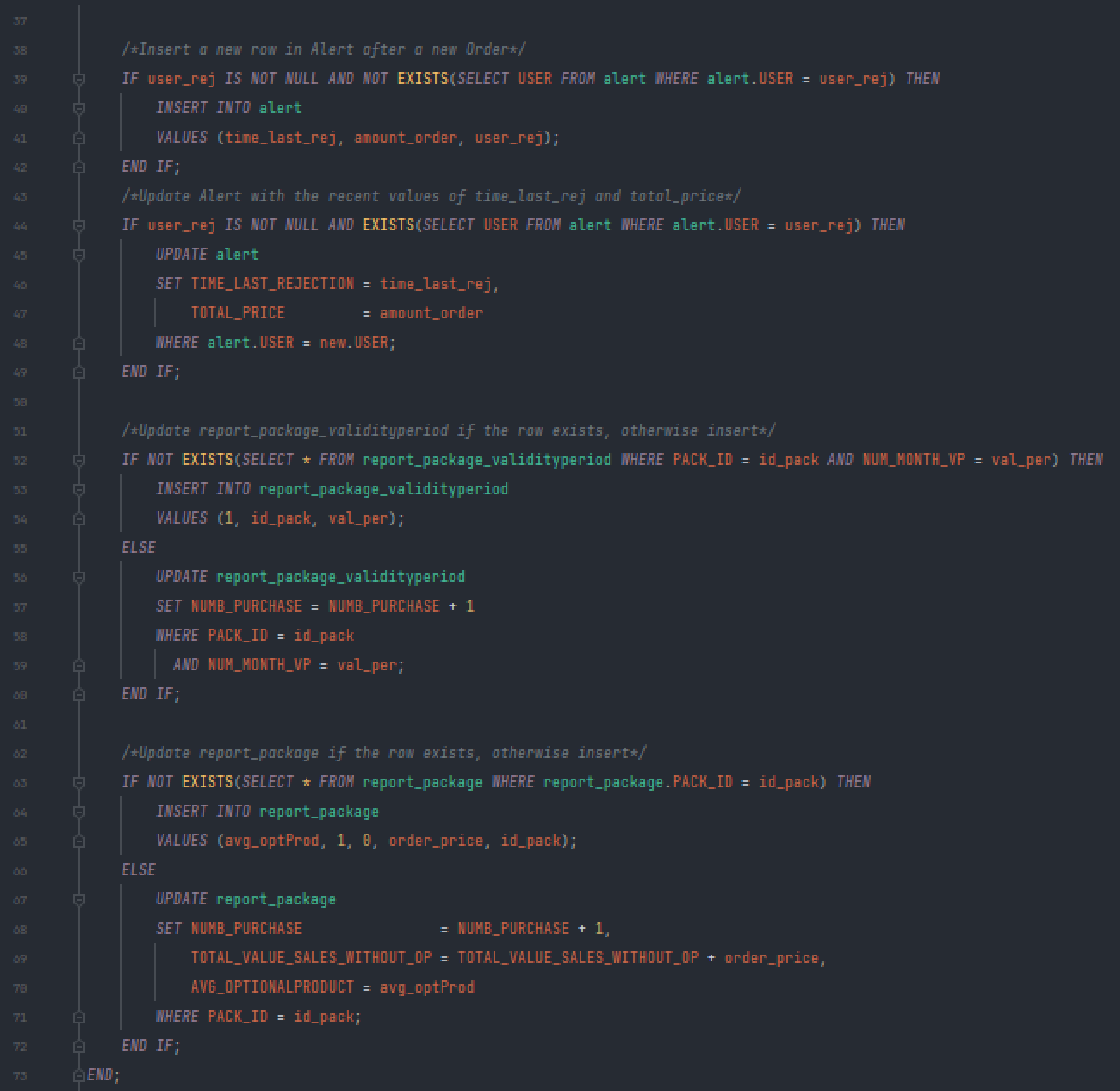
- package_AFTER_INSERT:

  after the insertion of new Package, the trigger creates a new record in the report table.

order_AFTER_INSERT

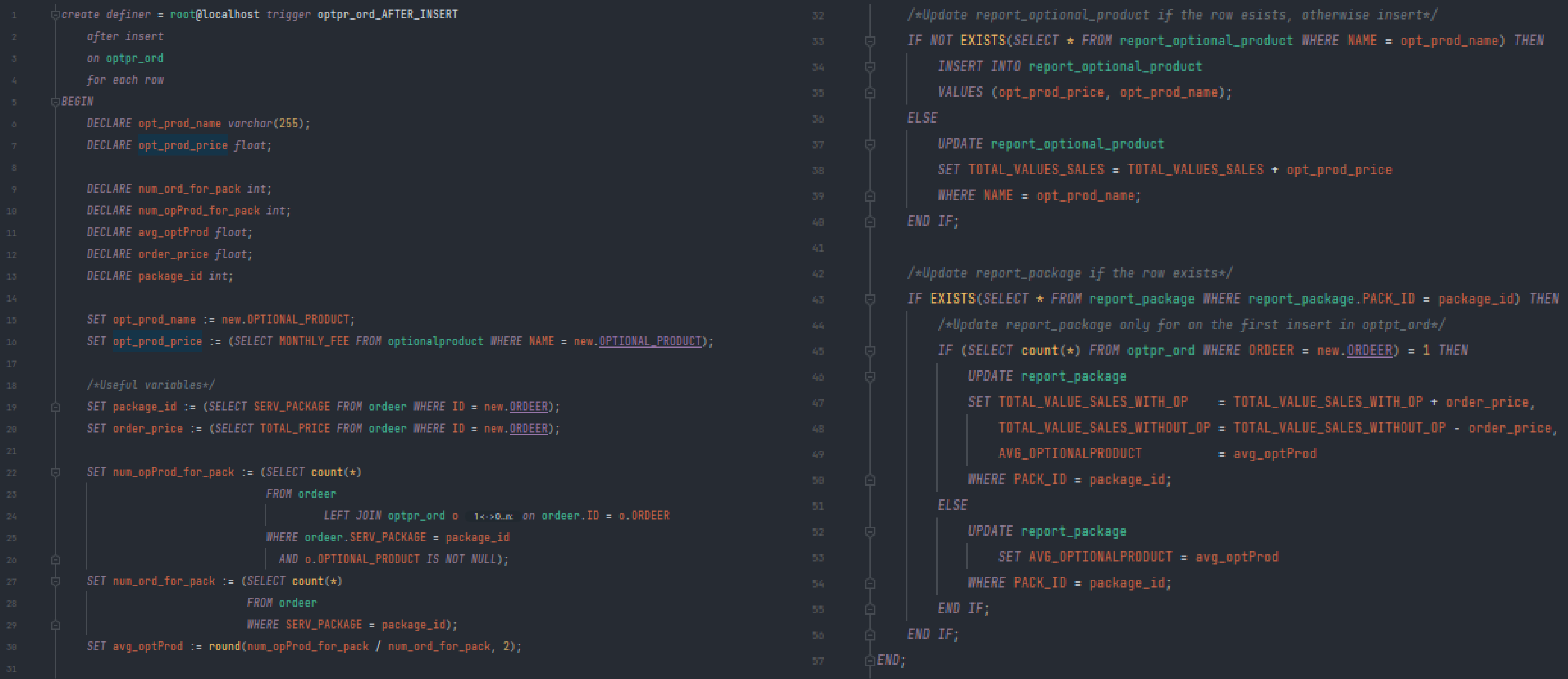


```sql
create definer = root@localhost trigger order_AFTER_INSERT
    after insert
    on ordeer
    for each row
BEGIN
    DECLARE user_rej varchar(255);
    DECLARE time_last_rej datetime(3);
    DECLARE amount_order float;

    DECLARE id_pack int;
    DECLARE name_pack varchar(255);
    DECLARE val_per int;

    DECLARE order_price float;

    DECLARE num_ord_for_pack int;
    DECLARE num_opProd_for_pack int;
    DECLARE avg_optProd float;

    SET time_last_rej := (SELECT max(DATE_LAST_REJ) FROM ordeer WHERE USER = new.USER AND IS_VALID = false);
    SET amount_order := (SELECT SUM(TOTAL_PRICE) FROM ordeer WHERE USER = new.USER AND IS_VALID = false);
    SET user_rej := (SELECT USER FROM ordeer WHERE USER = new.USER AND IS_VALID = 0 HAVING SUM(REJ_NUMB) >= 3);

    SET id_pack := new.SERV_PACKAGE;
    SET val_per := new.VALIDITY_PERIOD;

    SET order_price := new.TOTAL_PRICE;
    SET num_opProd_for_pack := (SELECT count(*)
                                FROM ordeer
                                         LEFT JOIN optpr_ord o  1<->0..n on ordeer.ID = o.ORDEER
                                WHERE ordeer.SERV_PACKAGE = id_pack
                                  AND o.OPTIONAL_PRODUCT IS NOT NULL);
    SET num_ord_for_pack := (SELECT count(*)
                             FROM ordeer
                             WHERE SERV_PACKAGE = id_pack);
    SET avg_optProd := round(num_opProd_for_pack / num_ord_for_pack, 2);
```



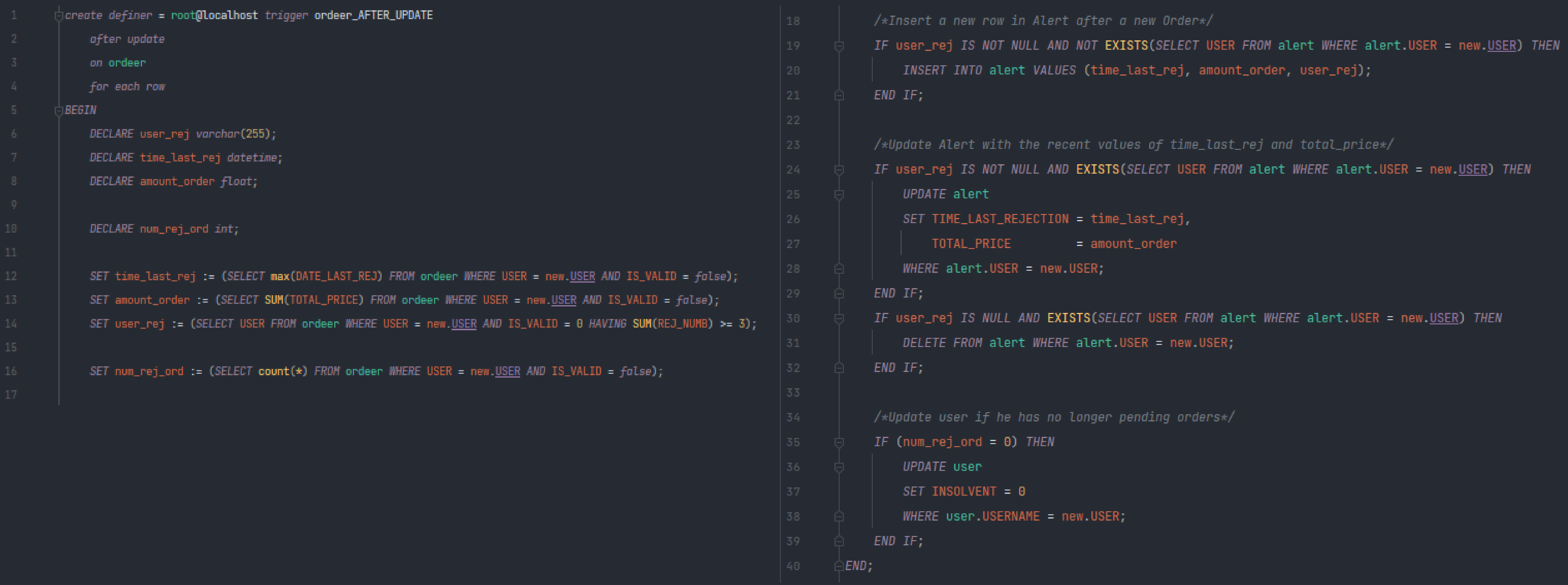
```sql

    /*Insert a new row in Alert after a new Order*/
    IF user_rej IS NOT NULL AND NOT EXISTS(SELECT USER FROM alert WHERE alert.USER = user_rej) THEN
        INSERT INTO alert
        VALUES (time_last_rej, amount_order, user_rej);
    END IF;
    /*Update Alert with the recent values of time_last_rej and total_price*/
    IF user_rej IS NOT NULL AND EXISTS(SELECT USER FROM alert WHERE alert.USER = user_rej) THEN
        UPDATE alert
        SET TIME_LAST_REJECTION = time_last_rej,
            TOTAL_PRICE         = amount_order
        WHERE alert.USER = new.USER;
    END IF;

    /*Update report_package_validityperiod if the row exists, otherwise insert*/
    IF NOT EXISTS(SELECT * FROM report_package_validityperiod WHERE PACK_ID = id_pack AND NUM_MONTH_VP = val_per) THEN
        INSERT INTO report_package_validityperiod
        VALUES (1, id_pack, val_per);
    ELSE
        UPDATE report_package_validityperiod
        SET NUMB_PURCHASE = NUMB_PURCHASE + 1
        WHERE PACK_ID = id_pack
          AND NUM_MONTH_VP = val_per;
    END IF;

    /*Update report_package if the row exists, otherwise insert*/
    IF NOT EXISTS(SELECT * FROM report_package WHERE report_package.PACK_ID = id_pack) THEN
        INSERT INTO report_package
        VALUES (avg_optProd, 1, 0, order_price, id_pack);
    ELSE
        UPDATE report_package
        SET NUMB_PURCHASE                 = NUMB_PURCHASE + 1,
            TOTAL_VALUE_SALES_WITHOUT_OP = TOTAL_VALUE_SALES_WITHOUT_OP + order_price,
            AVG_OPTIONALPRODUCT = avg_optProd
        WHERE PACK_ID = id_pack;
    END IF;
END;
```

optpr_ord_AFTER_INSERT



```sql
create definer = root@localhost trigger optpr_ord_AFTER_INSERT
    after insert
    on optpr_ord
    for each row
BEGIN
    DECLARE opt_prod_name varchar(255);
    DECLARE opt_prod_price float;

    DECLARE num_ord_for_pack int;
    DECLARE num_opProd_for_pack int;
    DECLARE avg_optProd float;
    DECLARE order_price float;
    DECLARE package_id int;

    SET opt_prod_name := new.OPTIONAL_PRODUCT;
    SET opt_prod_price := (SELECT MONTHLY_FEE FROM optionalproduct WHERE NAME = new.OPTIONAL_PRODUCT);

    /*Useful variables*/
    SET package_id := (SELECT SERV_PACKAGE FROM ordeer WHERE ID = new.ORDEER);
    SET order_price := (SELECT TOTAL_PRICE FROM ordeer WHERE ID = new.ORDEER);

    SET num_opProd_for_pack := (SELECT count(*)
                                FROM ordeer
                                         LEFT JOIN optpr_ord o 1<->0..n on ordeer.ID = o.ORDEER
                                WHERE ordeer.SERV_PACKAGE = package_id
                                  AND o.OPTIONAL_PRODUCT IS NOT NULL);
    SET num_ord_for_pack := (SELECT count(*)
                             FROM ordeer
                             WHERE SERV_PACKAGE = package_id);
    SET avg_optProd := round(num_opProd_for_pack / num_ord_for_pack, 2);

    /*Update report_optional_product if the row esists, otherwise insert*/
    IF NOT EXISTS(SELECT * FROM report_optional_product WHERE NAME = opt_prod_name) THEN
        INSERT INTO report_optional_product
        VALUES (opt_prod_price, opt_prod_name);
    ELSE
        UPDATE report_optional_product
        SET TOTAL_VALUES_SALES = TOTAL_VALUES_SALES + opt_prod_price
        WHERE NAME = opt_prod_name;
    END IF;

    /*Update report_package if the row exists*/
    IF EXISTS(SELECT * FROM report_package WHERE report_package.PACK_ID = package_id) THEN
        /*Update report_package only for on the first insert in optpt_ord*/
        IF (SELECT count(*) FROM optpr_ord WHERE ORDEER = new.ORDEER) = 1 THEN
            UPDATE report_package
            SET TOTAL_VALUE_SALES_WITH_OP    = TOTAL_VALUE_SALES_WITH_OP + order_price,
                TOTAL_VALUE_SALES_WITHOUT_OP = TOTAL_VALUE_SALES_WITHOUT_OP - order_price,
                AVG_OPTIONALPRODUCT          = avg_optProd
            WHERE PACK_ID = package_id;
        ELSE
            UPDATE report_package
                SET AVG_OPTIONALPRODUCT = avg_optProd
            WHERE PACK_ID = package_id;
        END IF;
    END IF;
END;
```

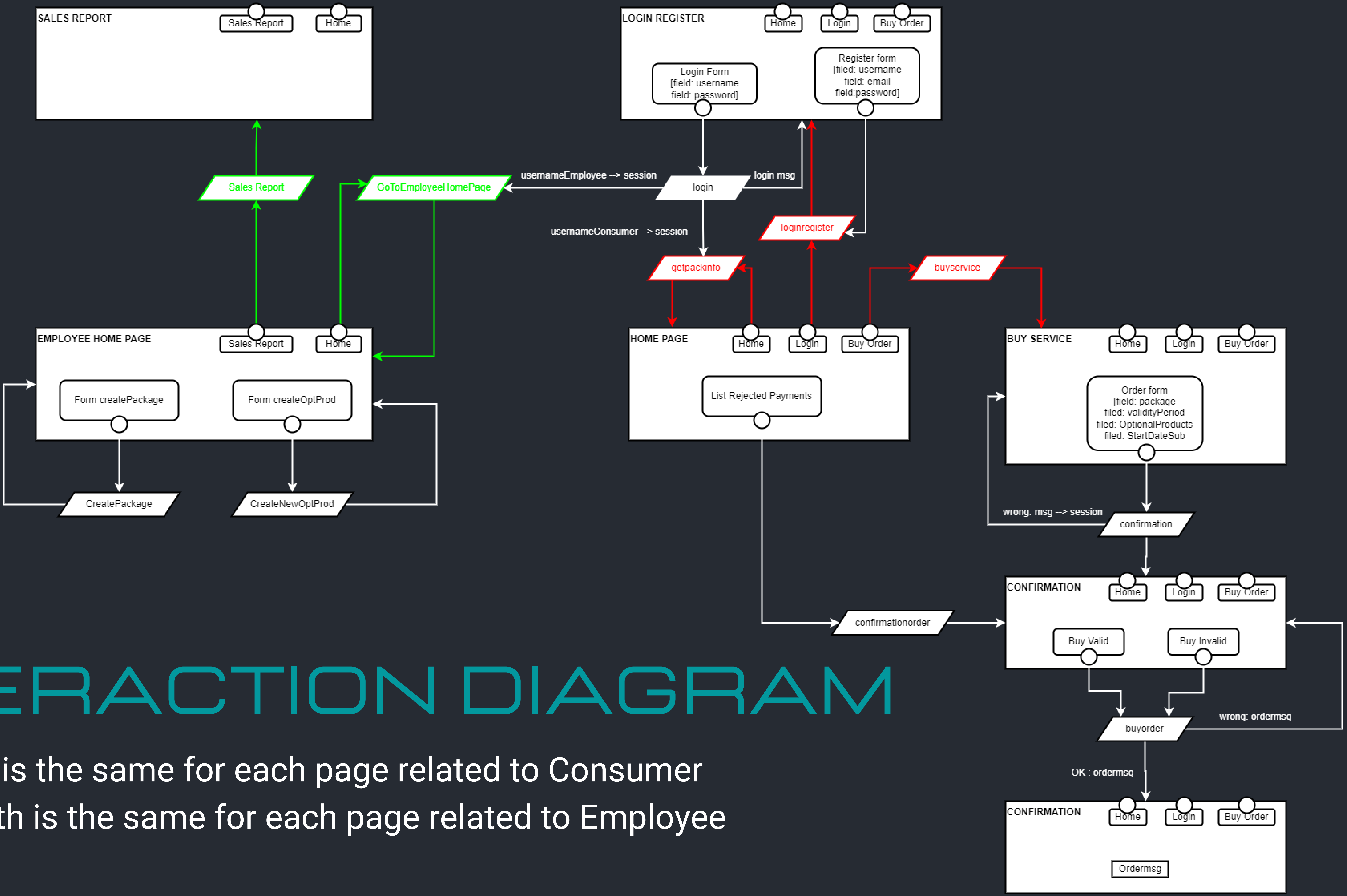
order_AFTER_UPDATE



```sql
create definer = root@localhost trigger ordeer_AFTER_UPDATE
    after update
    on ordeer
    for each row
BEGIN
    DECLARE user_rej varchar(255);
    DECLARE time_last_rej datetime;
    DECLARE amount_order float;

    DECLARE num_rej_ord int;

    SET time_last_rej := (SELECT max(DATE_LAST_REJ) FROM ordeer WHERE USER = new.USER AND IS_VALID = false);
    SET amount_order := (SELECT SUM(TOTAL_PRICE) FROM ordeer WHERE USER = new.USER AND IS_VALID = false);
    SET user_rej := (SELECT USER FROM ordeer WHERE USER = new.USER AND IS_VALID = 0 HAVING SUM(REJ_NUMB) >= 3);

    SET num_rej_ord := (SELECT count(*) FROM ordeer WHERE USER = new.USER AND IS_VALID = false);

    /*Insert a new row in Alert after a new Order*/
    IF user_rej IS NOT NULL AND NOT EXISTS(SELECT USER FROM alert WHERE alert.USER = new.USER) THEN
        INSERT INTO alert VALUES (time_last_rej, amount_order, user_rej);
    END IF;

    /*Update Alert with the recent values of time_last_rej and total_price*/
    IF user_rej IS NOT NULL AND EXISTS(SELECT USER FROM alert WHERE alert.USER = new.USER) THEN
        UPDATE alert
        SET TIME_LAST_REJECTION = time_last_rej,
            TOTAL_PRICE         = amount_order
        WHERE alert.USER = new.USER;
    END IF;
    IF user_rej IS NULL AND EXISTS(SELECT USER FROM alert WHERE alert.USER = new.USER) THEN
        DELETE FROM alert WHERE alert.USER = new.USER;
    END IF;

    /*Update user if he has no longer pending orders*/
    IF (num_rej_ord = 0) THEN
        UPDATE user
        SET INSOLVENT = 0
        WHERE user.USERNAME = new.USER;
    END IF;
END;
```

package_AFTER_INSERT

```sql
create definer = root@localhost trigger package_AFTER_INSERT
    after insert
    on package
    for each row
BEGIN
    INSERT INTO report_package VALUES (0, 0, 0, 0, new.ID);
END;
```

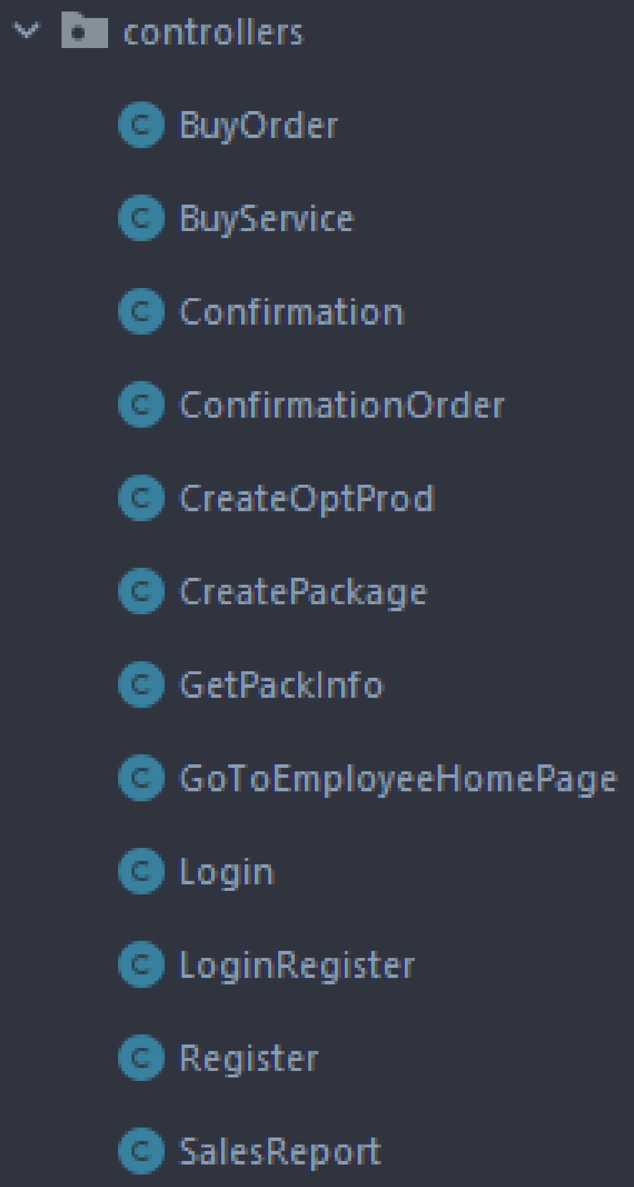


INTERACTION DIAGRAM

- Red path is the same for each page related to Consumer
- Green path is the same for each page related to Employee

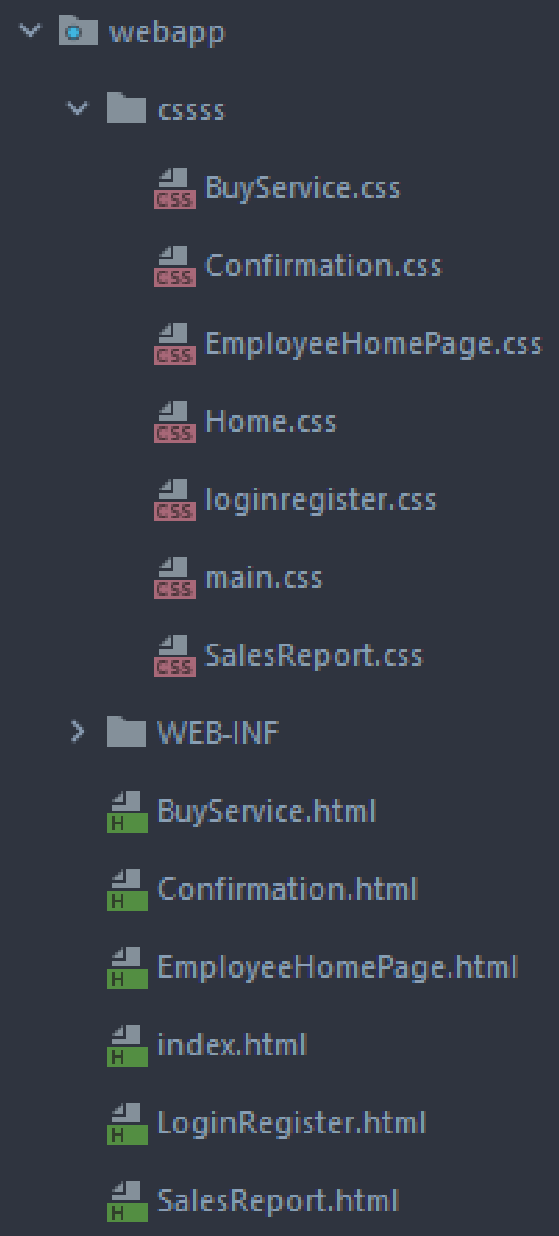
COMPONENTS

- Client Components
  - Servlets



controllers
- BuyOrder
- BuyService
- Confirmation
- ConfirmationOrder
- CreateOptProd
- CreatePackage
- GetPackInfo
- GoToEmployeeHomePage
- Login
- LoginRegister
- Register
- SalesReport

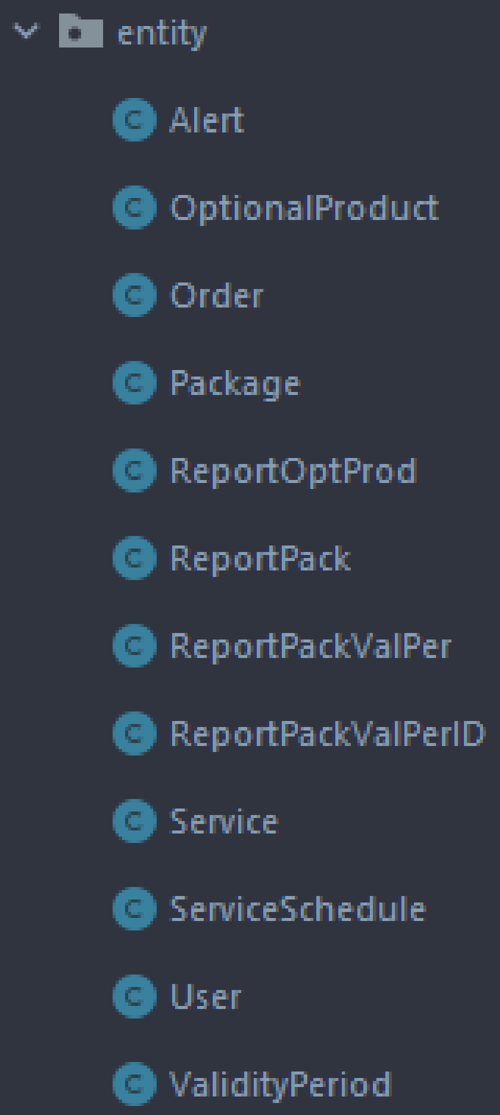
  - Views



webapp
- cssss
  - BuyService.css
  - Confirmation.css
  - EmployeeHomePage.css
  - Home.css
  - loginregister.css
  - main.css
  - SalesReport.css
- WEB-INF
- BuyService.html
- Confirmation.html
- EmployeeHomePage.html
- index.html
- LoginRegister.html
- SalesReport.html

COMPONENTS

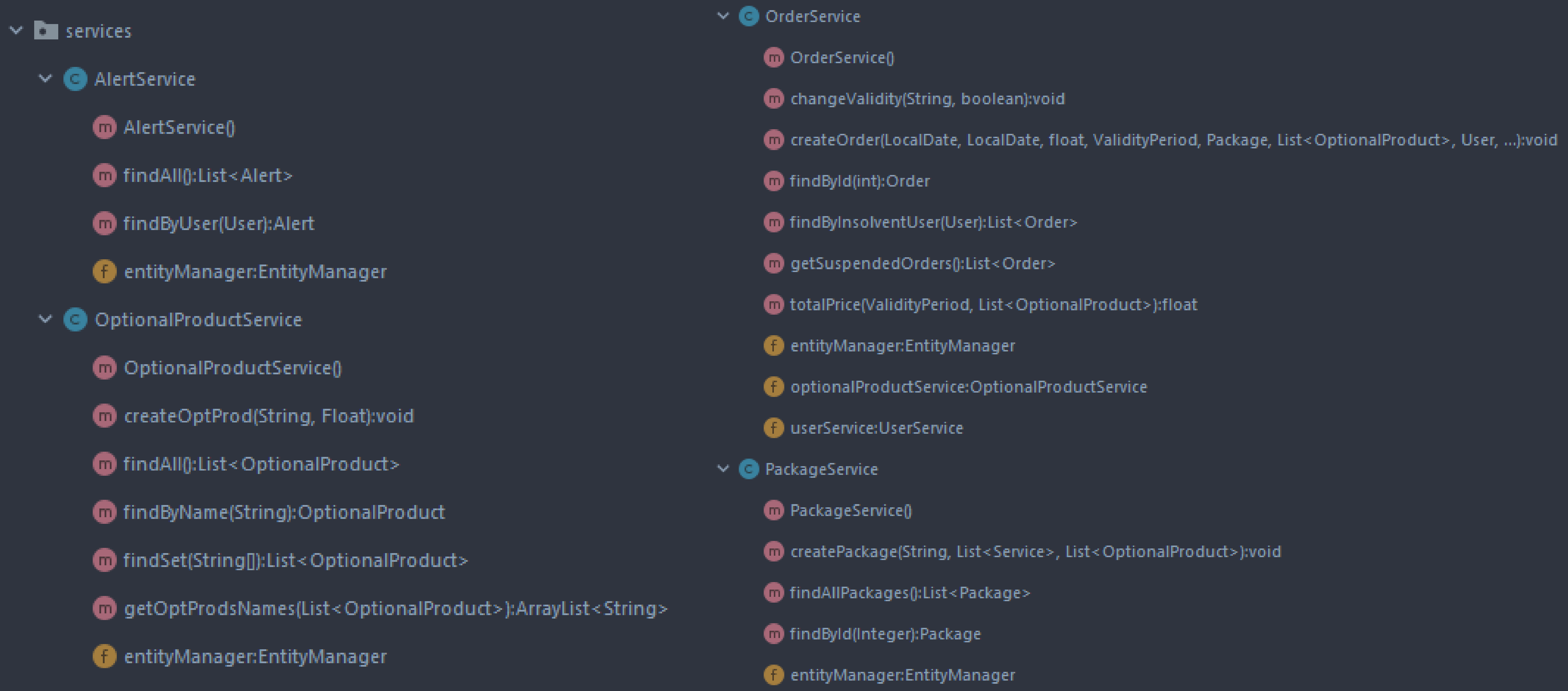
- Back-end Components
  - Entities



entity
- Alert
- OptionalProduct
- Order
- Package
- ReportOptProd
- ReportPack
- ReportPackValPer
- ReportPackValPerID
- Service
- ServiceSchedule
- User
- ValidityPeriod

COMPONENTS

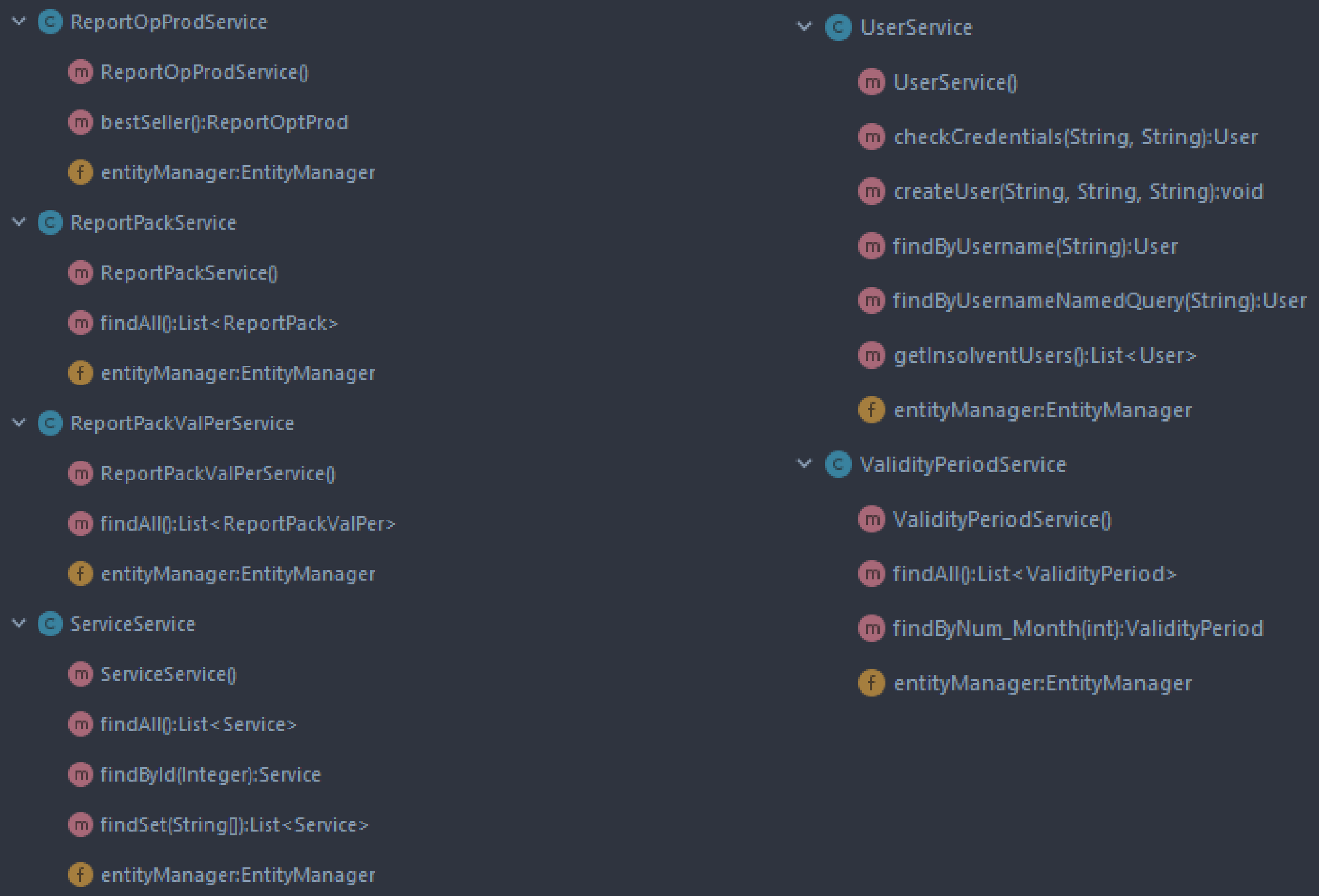
- Back-end Components
  - Business components (EJBs)



```
services
  AlertService
    AlertService()
    findAll():List<Alert>
    findByUser(User):Alert
    entityManager:EntityManager
  OptionalProductService
    OptionalProductService()
    createOptProd(String, Float):void
    findAll():List<OptionalProduct>
    findByName(String):OptionalProduct
    findSet(String[]):List<OptionalProduct>
    getOptProdsNames(List<OptionalProduct>):ArrayList<String>
    entityManager:EntityManager
  OrderService
    OrderService()
    changeValidity(String, boolean):void
    createOrder(LocalDate, LocalDate, float, ValidityPeriod, Package, List<OptionalProduct>, User, ...):void
    findById(int):Order
    findByInsolventUser(User):List<Order>
    getSuspendedOrders():List<Order>
    totalPrice(ValidityPeriod, List<OptionalProduct>):float
    entityManager:EntityManager
    optionalProductService:OptionalProductService
    userService:UserService
  PackageService
    PackageService()
    createPackage(String, List<Service>, List<OptionalProduct>):void
    findAllPackages():List<Package>
    findById(Integer):Package
    entityManager:EntityManager
```

COMPONENTS

- Back-end Components
    - Business components (EJBs)



```
ReportOpProdService
    ReportOpProdService()
    bestSeller():ReportOptProd
    entityManager:EntityManager
ReportPackService
    ReportPackService()
    findAll():List<ReportPack>
    entityManager:EntityManager
ReportPackValPerService
    ReportPackValPerService()
    findAll():List<ReportPackValPer>
    entityManager:EntityManager
ServiceService
    ServiceService()
    findAll():List<Service>
    findById(Integer):Service
    findSet(String[]):List<Service>
    entityManager:EntityManager
```

```
UserService
    UserService()
    checkCredentials(String, String):User
    createUser(String, String, String):void
    findByUsername(String):User
    findByUsernameNamedQuery(String):User
    getInsolventUsers():List<User>
    entityManager:EntityManager
ValidityPeriodService
    ValidityPeriodService()
    findAll():List<ValidityPeriod>
    findByNum_Month(int):ValidityPeriod
    entityManager:EntityManager
```