Data bases 2

Giuseppe Tortorelli - 10582962
Juri Sacchetta - 10618120

Index

Specifications

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

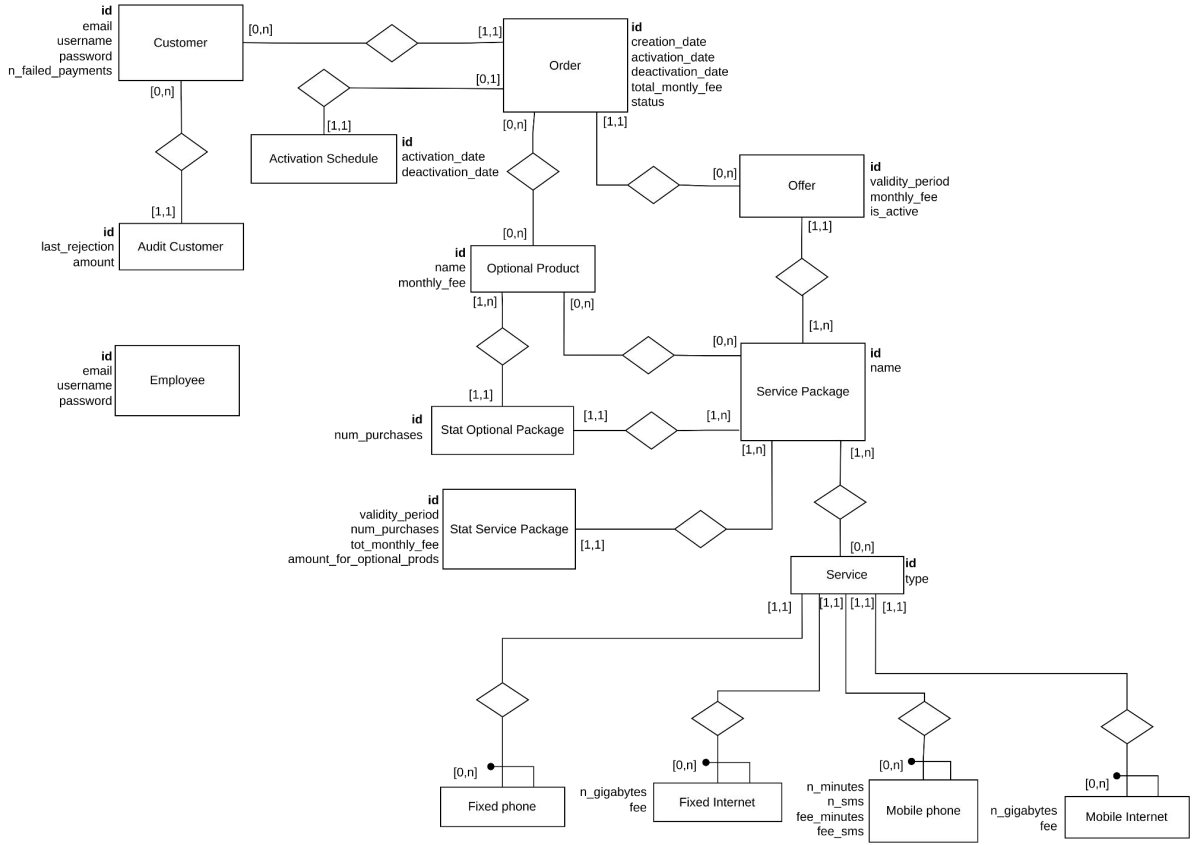
- Number of total purchases per package.
- Number of total purchases per package and validity period.
- Total value of sales per package with and without the optional products.
- Average number of optional products sold together with each service package.
- List of insolvent users, suspended orders and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

Specification interpretation

we made the following assumptions:

- An **optional product** can be sell only associated with a service package
- The **total monthly fee** is fixed when an order is created and will never change also if the fee of the associated offer will change
- A customer is marked as **insolvent** after one failed payment but he is inserted in **audit table** after 3 failed payment

Entity Relationship



Motivation of the ER design

- The **Offer** is defined as a combination of validity period, monthly fee and service package. Only the active offers will be shown to the user, the others are used to keep the info associated with the order
- The **Order** table is associated with the optional products that the user has selected from those available in the service package. The status field can take 2 values: accepted, rejected.
- In the **Service** table, the type field is used to select the table in which to search for the tuple. The id of the tuple is the same as that of the service
- The **Activation Schedule** table is populated by triggers and contains only orders for which payment has been successful (when order status is successful)
- **Fixed Phone**, **Mobile Phone**, **Fixed Internet**, **Mobile Internet** are downward collapsed hierarchies of Service

Relational model

Employee(id, email, username, password)

Customer(**id**, email, username, password, num_failed_payments)

AuditCustomer(id, id_customer, amount, last_rejection)

ServicePackage(**id**, name)

Offer(**id**, id_package, validity_period, monthly_fee, is_active)

Order(**id**, id_user, id_offer, creation_date, activation_date, deactivation_date, total_monthly_fee, status)

ActivationSchedule(id, id_order, activation_date, deactivation_date)

OptionalProduct(**id**, name, monthly_fee)

Service(**id**, type)

Relational model

FixedInternet(id, n_gigabytes, fee)

MobilePhone(id, n_minutes, fee_minutes, n_gigabytes, fee_gigabytes)

MobileInternet(id, n_gigabytes, fee)

AuditCustomer(id, id_service_package, id_offer, id_order)

OrderToOptionalProduct(id_order, id_optional_product)

ServicePackageToOptionalProduct(id_service_package, id_optional_product)

ServicePackageToService(id_service_package, id_service)

StatOptionalPackage(id, id_optional_product, id_service_package, num_purchases)

StatServicePackage(id, id_service_package, validity_period, num_purchases, tot_monthly_fee, amount_for_optional_prods)

Motivations of the logical design

- **Fixed Phone** is not present in the logical scheme since he has no attributes and can be recognized by the type field of service
- **StatOptionalPackage** and **StatServicePackage** are conceptually materialized views but basically they are tables since MySQL does not support Materialized Views. Nonetheless, they are populated with triggers.

Trigger design and code

populate_activation_schedule_insert

- **Event**: when a new order is inserted
- **Condition**: if the attribute status is equal to 1 (successful payment)
- **Action**: insert a new tuple in activaton_schedule table
- **SQL code**

```
CREATE TRIGGER populate_activation_schedule_insert AFTER INSERT ON order FOR EACH ROW
BEGIN
       IF new.status = 1 THEN
              INSERT INTO activation_schedule(id_order, activation_date, deactivation_date)
              VALUES (new.id, new.activation_date, new.deactivation_date);
       END IF;
END
```

Trigger design and code

populate_activation_schedule_update

- **Event**: when a order is updated
- **Condition**: if the attribute status is setted to 1 (successful payment)
- **Action**: insert a new tuple in activaton_schedule table
- **SQL code**

```
CREATE TRIGGER populate_activation_schedule_update AFTER UPDATE ON order FOR EACH ROW
BEGIN
	IF old.status = 0 AND new.status = 1 THEN
		INSERT INTO activation_schedule(id_order, activation_date, deactivation_date)
		VALUES (new.id, new.activation_date, new.deactivation_date);
	END IF;
END
```

- **Motivation**: this trigger is used when the payment fails on the first attempt and it’s retried later.

Trigger design and code

populate_purchases_stat_insert

- **Event**: when a order is inserted
- **Condition**: always performs the action
- **Action**: if exists in the stat_service_package table the reference on inserted package with corresponding validity_period , update values otherwise insert a new entry
- **SQL Code**:

```
CREATE TRIGGER populate_purchases_stat_insert AFTER INSERT ON order FOR EACH ROW
BEGIN
    DECLARE idPackage INT;
    DECLARE validityPeriod INT;
    DECLARE monthlyFee DOUBLE;
    SELECT id_package, validity_period, monthly_fee INTO idPackage, validityPeriod, monthlyFee FROM offer
       WHENE id = new.id_offer;

    IF EXISTS(SELECT * FROM stat_service_package WHERE id_package = idPackage AND validity_period = validityPeriod) THEN
        UPDATE stat_service_package SET num_purchases = num_purchases + 1,
             tot_monthly_fee = stat_service_package.tot_monthly_fee + monthlyFee;
    ELSE
        INSERT INTO stat_service_package(id_package, validity_period, num_purchases, tot_monthly_fee)
             VALUES (idPackage, validityPeriod, 1, monthlyFee);
    END IF;
END;
```

- **Motivation**: every time an order is created, service package statistics are updated

Trigger design and code

update_amount_optional_stat_insert

- **Event**: every time an order with optional product associated is inserted
- **Condition**: always performs the action
- **Action**: update the amount of the optional product that has been purchased
- **SQL code:**

```
CREATE TRIGGER update_amount_optional_stat_insert AFTER INSERT ON order_to_optional_produt FOR EACH ROW
BEGIN
    UPDATE stat_service_package SET amount_for_optional_prods = amount_for_optional_prods +
       (SELECT monthly_fee FROM optional_product WHERE id = new.id_optional_product);
END;
```

Trigger design and code

populate_purchases_optional_strat_insert

- **Event**: every time an order with optional product associated is inserted
- **Condition**: always perform the action
- **Action**: if exists in the stat_service_optional table the reference on inserted optional product with corresponding service package, update values otherwise insert a new entry
- **SQL code**:

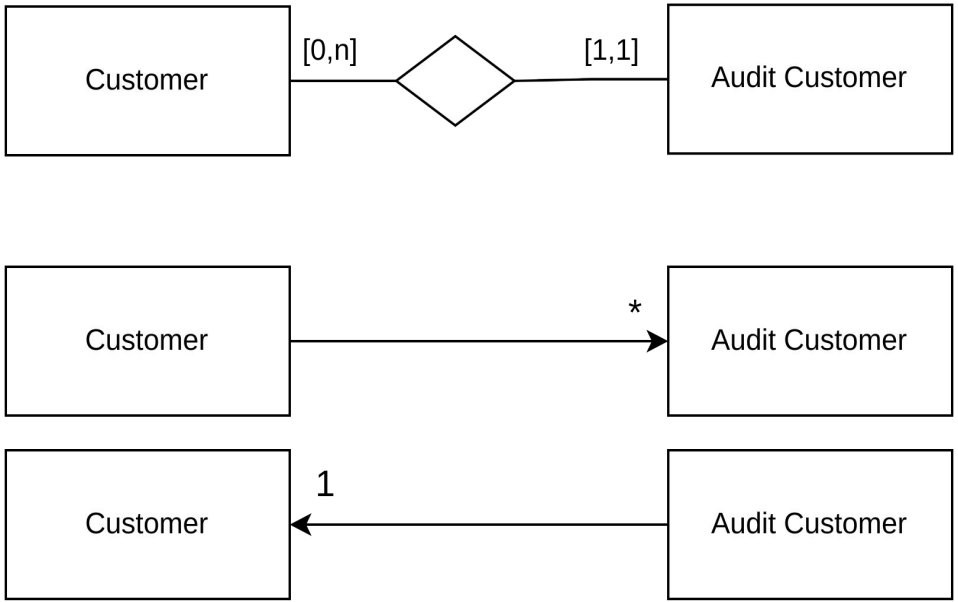
```
CREATE TRIGGER populate_purchases_optional_strat_insert AFTER INSERT ON order_to_optional_product FOR EACH ROW
BEGIN
    DECLARE idServicePackage INT;
    SELECT id_package INTO idServicePackage FROM offer WHERE id IN (SELECT id_offer FROM order WHERE id = new.id_order);

    IF EXISTS(SELECT * FROM stat_optional_package WHERE
                   id_optional = new.id_optional_product AND id_package = idServicePackage) THEN
       UPDATE stat_optional_package SET num_purchases = num_purchases + 1 WHERE
             id_optional = new.id_optional_product AND id_package = idServicePackage;
    ELSE
        INSERT INTO stat_optional_package VALUES (idServicePackage, new.id_optional_product, 1);
    END IF;
END;
```

- **Motivation**: every time optional products are bought, statistics have to be updated

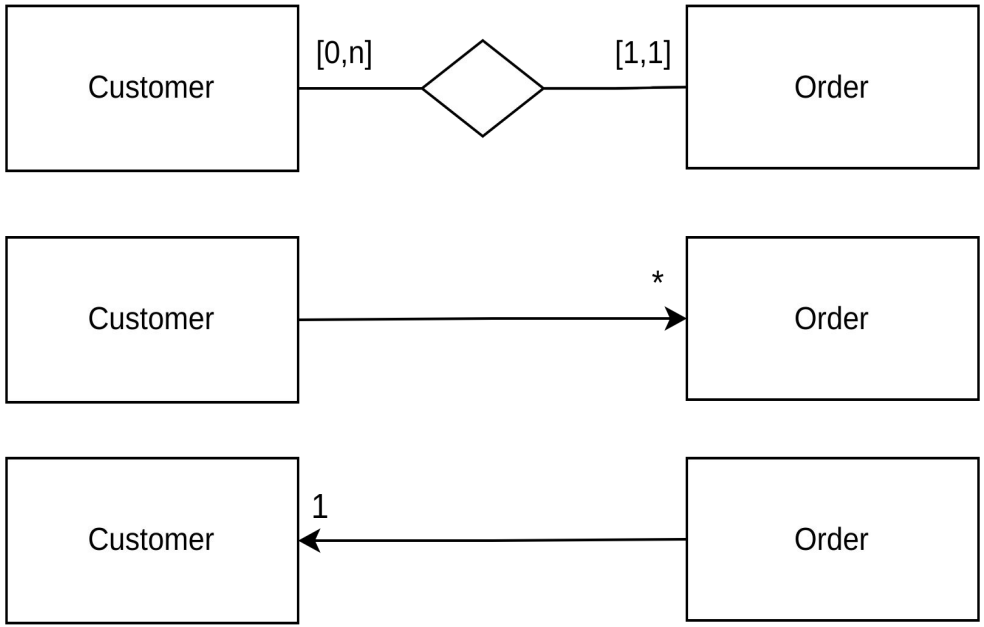
ORM design

Relationship “customer-audit_customer”



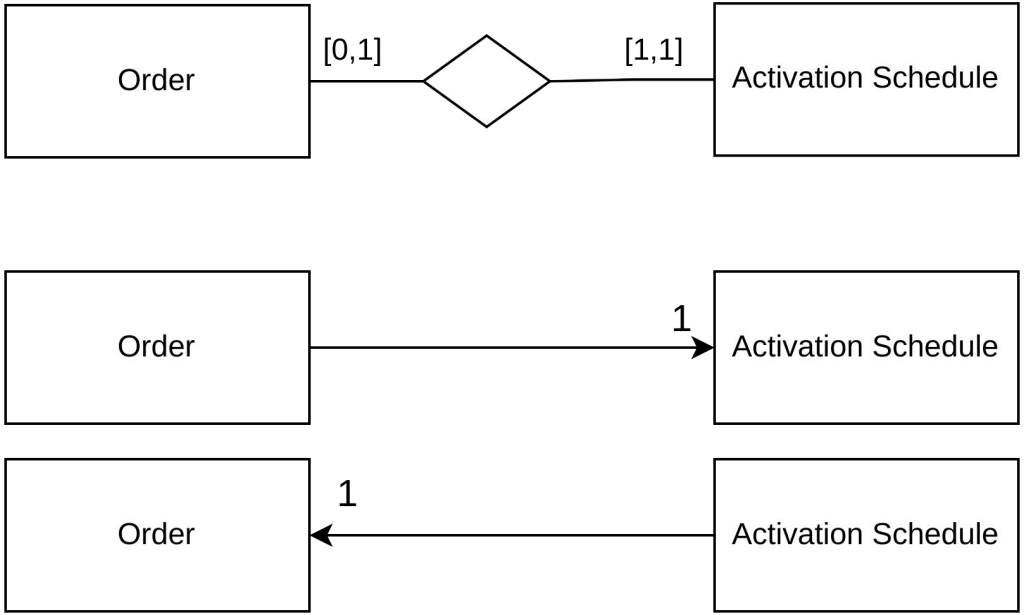
- **Customer -> Audit Customer**
  - `@OneToMany` (not needed)
- **Audit Customer -> Customer**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_customer”, nullable = false, updatable = false)`

Relationship “customer-order”



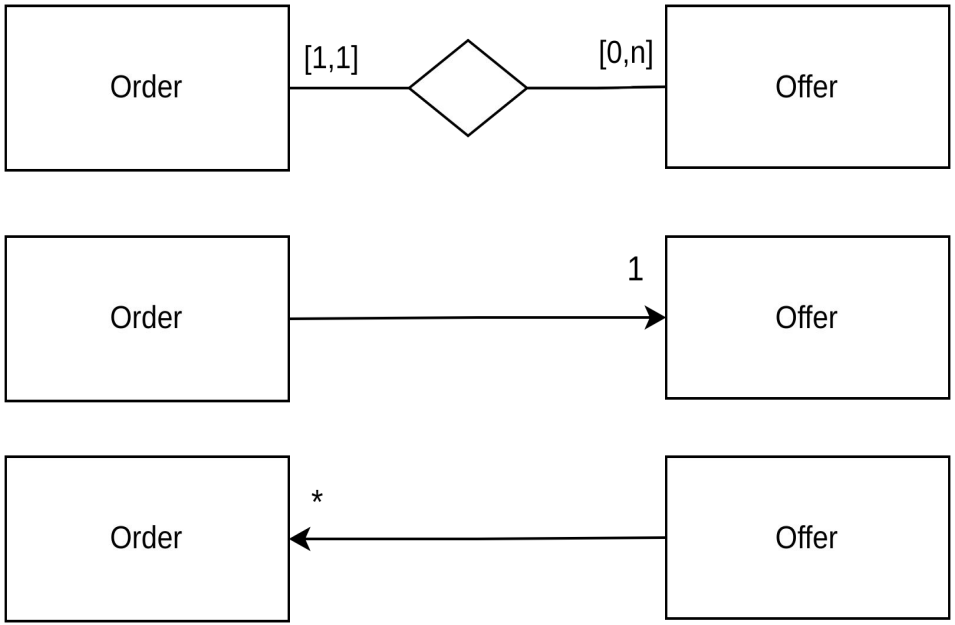
- **Customer -> Order**
  - `@OneToMany` (not needed)
- **Order -> Customer**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_user”, nullable = false, updatable = false)`

Relationship “order-activation_schedule”



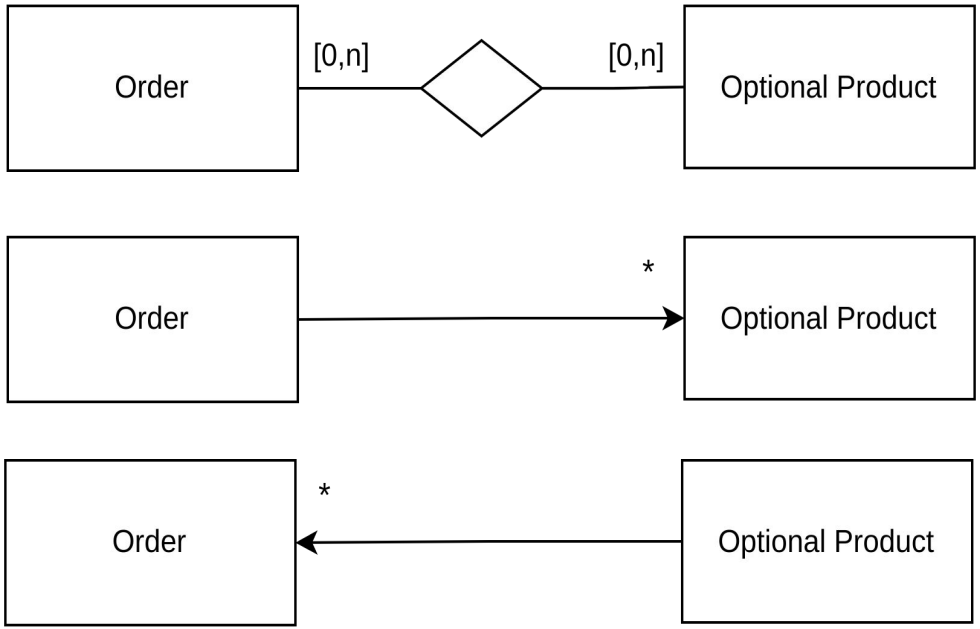
- **Order** -> **Activation Schedule**
  - `@OneToOne` (not needed)
- **Activation Schedule** -> **Order**
  - `@OneToOne` (not needed)

Relationship “order_offer”



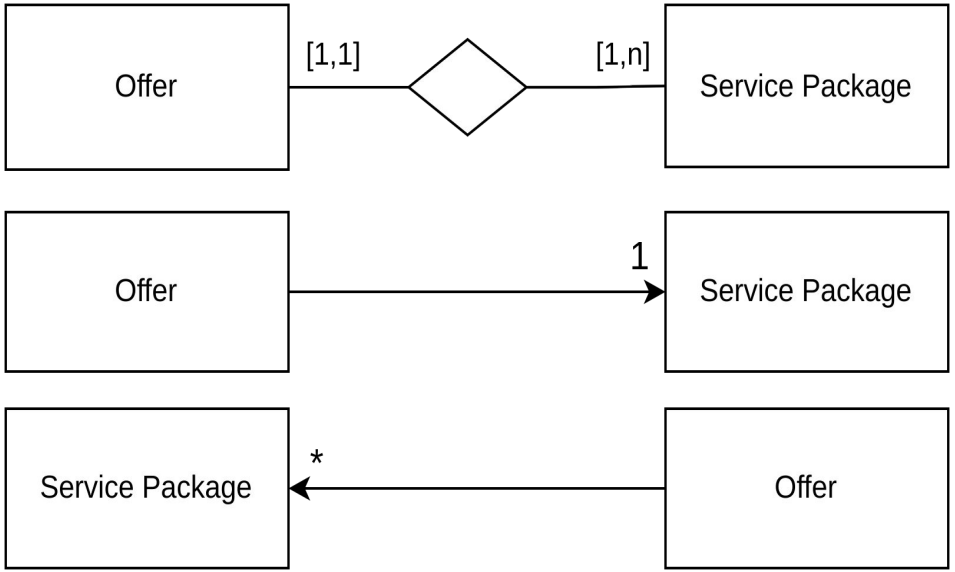
- **Order -> Offer**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_offer”, nullable = false, updatable = false)`
- **Offer -> Order**
  - `@OneToMany` (not needed)

Relationship “order-optional_product”



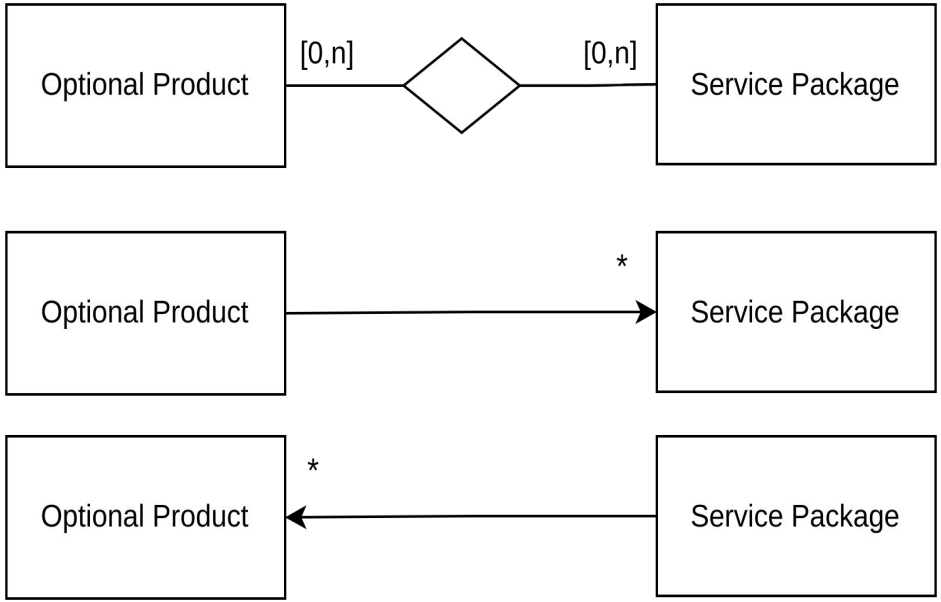
- **Order -> Optional Product**
  - `@ManyToMany`
  - `@JoinTable(name = "order_to_optional_product", joinColumns = @JoinColumn(name = "id_order"), inverseJoinColumns = @JoinColumn(name = "id_optional_product"))`
- **Optional Product -> Order**
  - `@ManyToMany` (not needed)

Relationship “offer-service_package”



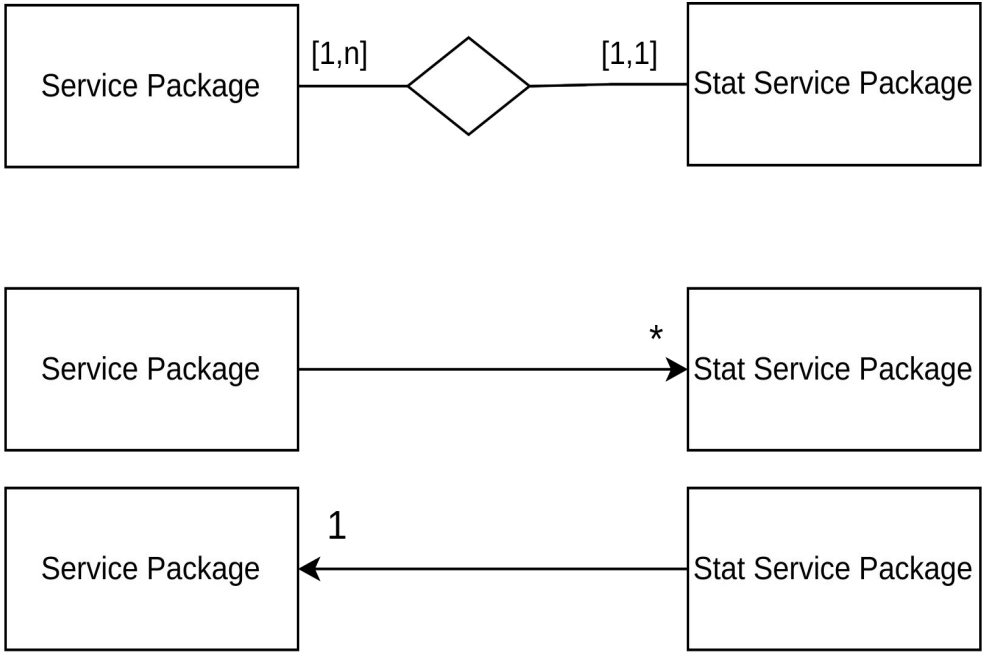
- **Offer -> Service Package**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_package”, nullable = false, updatable = false)`
- **Service Package -> Offer**
  - `@OneToMany(fetch = FetchType.LAZY, mappedBy = "servicePackage", cascade = CascadeType.ALL)`

Relationship “optional_product-service_pacakge”



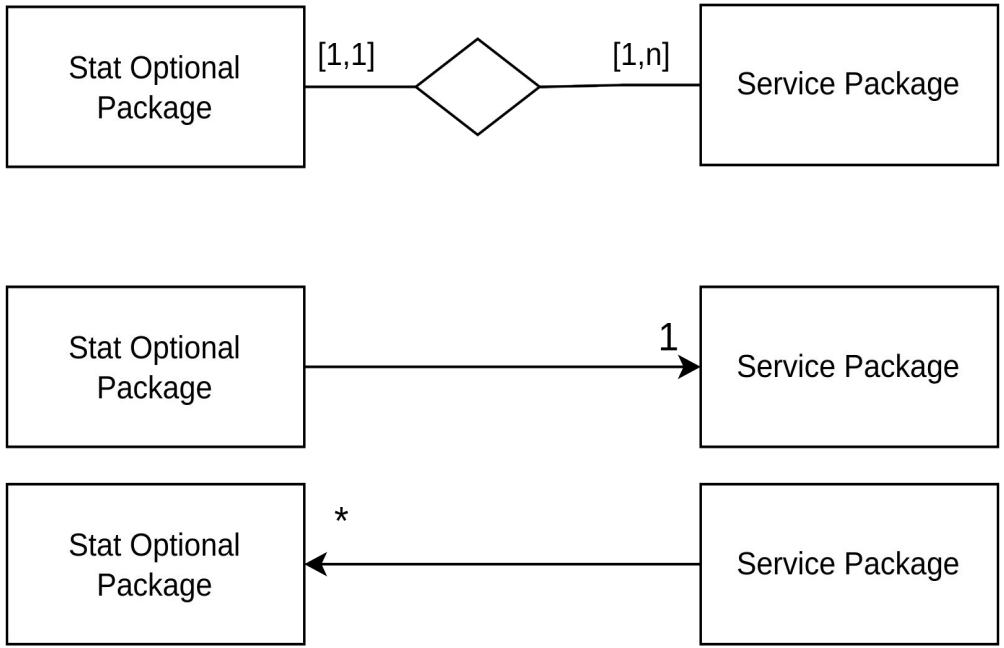
- **Optional Product -> Service Package**
  - `@ManyToMany` (not needed)
- **Service Package -> Optional Product**
  - `@ManyToMany`
  - `@JoinTable(name = "service_package_to_optional_product", joinColumns = @JoinColumn(name = "id_service_package"), inverseJoinColumns = @JoinColumn(name = "id_optional_product"))`

Relationship “service_package-stat_service_package”



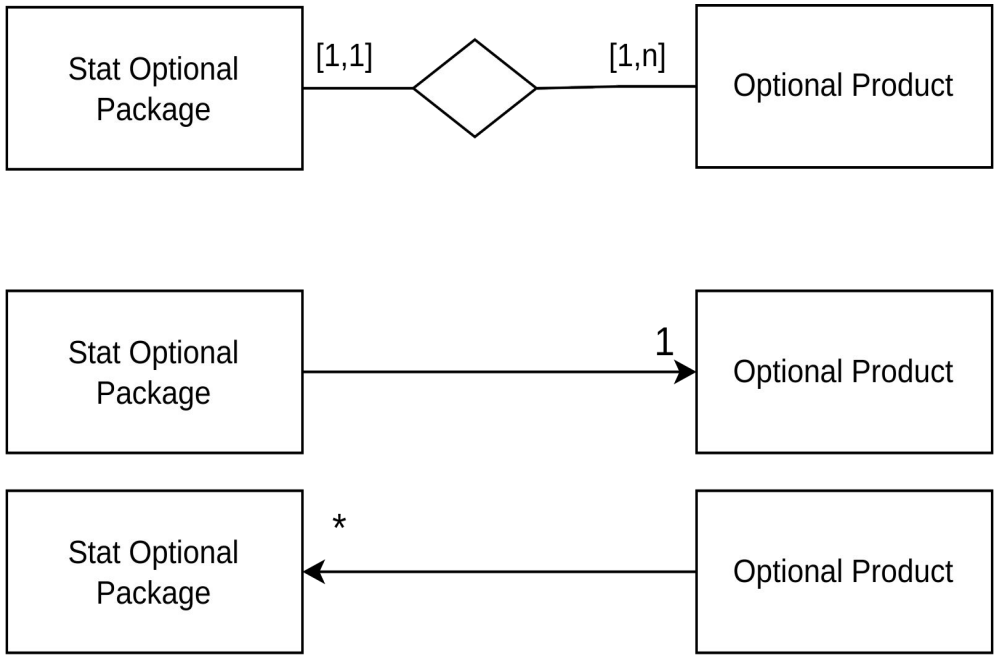
- **Service Package -> Stat Service Package**
  - `@OneToMany` (not needed)
- **Stat Service Package -> Service Package**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_package”, nullable = false, updatable = false)`

Relationship “stat_optional_package-service_package”



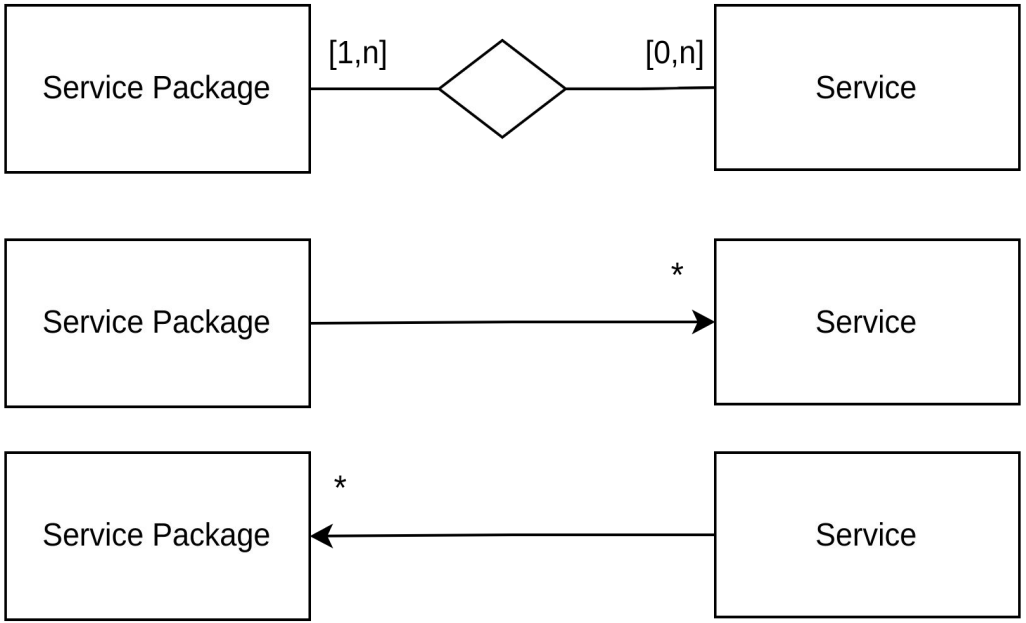
- **Stat Optional Package -> Service Package**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_optional”, nullable = false, updatable = false)`
- **Service Package -> Stat Optional Package**
  - `@OneToMany` (not needed)

Relationship “stat_optional_package-optional_product”



- **Stat Optional Package -> Optional Product**
  - `@ManyToOne`
  - `@JoinColumn(name = “id_optional”, nullable = false, updatable = false)`
- **Optional Product -> Stat Optional Package**
  - `@OneToMany` (not needed)

Relationship “service_package-service”



- **Service Package -> Service**
  - `@ManyToMany`
  - `@JoinTable(name = "service_package_to_service", joinColumns = @JoinColumn(name = "id_package"), inverseJoinColumns = @JoinColumn(name = "id_service"))`
- **Service -> Service Package**
  - `@ManyToMany` (not needed)

ORM design motivations

- The mapping between **Service** and **Fixed Phone**, **Mobile Phone**, **Fixed Internet**, **Mobile Internet** was done by extending the Service class and using the Joined strategy on a discriminant based on the type attribute of the service table

Entity Customer

```
@Entity
@Table(name = "customer", schema = "db2_project")
@NamedQuery(name = "Customer.checkCredentials", query = "SELECT r FROM Customer r  WHERE r.email =
?1 and r.password = ?2")
public class Customer implements User, Serializable, Comparable<Customer> {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    protected Long id;
    @Column(name = "email")
    protected String email;
    @Column(name = "username")
    protected String username;
    @Column(name = "password")
    protected String password;
    @Column(name = "num_failed_payments")
    private int numFailedPayments;
```

Entity AuditCustomer

```java
@Entity
@Table(name = "audit_customer", schema = "db2_project")
public class AuditCustomer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
    @Column(name = "amount")
    private double amount;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "date")
    private Date date;
    @ManyToOne
    @JoinColumn(name = "id_customer", updatable = false, nullable = false)
    private Customer customer;
```

Entity Employee

```
@Entity
@Table(name = "employee", schema = "db2_project")
@NamedQuery(name = "Employee.checkCredentials", query = "SELECT r FROM Employee r  WHERE r.email
= ?1 and r.password = ?2 ")
public class Employee implements User, Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    protected Long id;
    @Column(name = "email")
    protected String email;
    @Column(name = "username")
    protected String username;
    @Column(name = "password")
    protected String password;
```

Entity Order

```
@Entity
@Table(name = "order", schema = "db2_project")
@NamedQuery(name = "Order.rejectedOrders", query = "SELECT r FROM Order r  WHERE r.customer.id = ?1
and r.status = ?2")
@NamedQuery(name = "Order.rejectedOrdersByID", query = "SELECT r FROM Order r  WHERE r.id = ?1 and
r.customer.id = ?2 and r.status = ?3")
public class Order implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "creation_date")
    private Date creationDate;
    @Temporal(TemporalType.DATE)
    @Column(name = "activation_date")
    private Date activationDate;
    @Temporal(TemporalType.DATE)
    @Column(name = "deactivation_date")
    private Date deactivationDate;
```

Entity Order

```
    @Column(name = "total_monthly_fee")
    private double totalMonthlyFee;
    @Column(name = "status")
    private State status;
    @ManyToOne
    @JoinColumn(name = "id_user", nullable = false)
    private Customer customer;
    @ManyToOne
    @JoinColumn(name = "id_offer", nullable = false)
    private Offer offer;
    @ManyToMany
    @JoinTable(name = "order_to_optional_product", joinColumns = @JoinColumn(name = "id_order"),
inverseJoinColumns = @JoinColumn(name = "id_optional_product"))
    private Set<OptionalProduct> optionalProductSet;
```

Entity Offer

```
@Entity
@Table(name = "offer", schema = "db2 project")
public class Offer implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;
    @Column(name = "validity period")
    private int validityPeriod;
    @Column(name = "monthly_fee")
    private double monthlyFee;
    @Column(name = "is active")
    private boolean active;
    @ManyToOne
    @JoinColumn(name = "id_package", nullable = false, updatable = false)
    private ServicePackage servicePackage;
```

Entity ServicePackage

```
@Entity
@Table(name = "service_package", schema = "db2_project")
public class ServicePackage implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
    @Column(name = "name")
    private String name;
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "servicePackage", cascade = CascadeType.ALL)
    private List<Offer> offers;
    @ManyToMany
    @JoinTable(name = "service_package_to_optional_product", joinColumns = @JoinColumn(name =
"id_service_package"), inverseJoinColumns = @JoinColumn(name = "id_optional_product"))
    private List<OptionalProduct> optionalProductList;
    @ManyToMany
    @JoinTable(name = "service_package_to_service", joinColumns = @JoinColumn(name = "id_package"),
inverseJoinColumns = @JoinColumn(name = "id_service"))
    private List<Service> serviceList;
```

Entity ActivationSchedule

```java
@Entity
@Table(name = "activation schedule", schema = "db2 project")
public class ActivationSchedule implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
    @Temporal(TemporalType.DATE)
    @Column(name = "activation date", nullable = false)
    private Date activationDate;
    @Temporal(TemporalType.DATE)
    @Column(name = "deactivation date", nullable = false)
    private Date deactivationDate;
}
```

Entity OptionalProduct

```java
@Entity
@Table(name = "optional_product", schema = "db2_project")
public class OptionalProduct implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;
    @Column(name = "name")
    private String name;
    @Column(name = "monthly_fee")
    private double monthlyFee;
```

Entity PackageOptionalStatistics

```
public static class PackageOptionalKey implements Serializable {
    private Long idPackage;
    private Long idOptional;

    public PackageOptionalKey() {
    }

    public PackageOptionalKey(Long idPackage, Long idOptional) {
        this.idPackage = idPackage;
        this.idOptional = idOptional;
    }
}
```

Entity PackageOptionalStatistics

```java
@Entity
@IdClass(PackageOptionalStatistics.PackageOptionalKey.class)
@Table(name = "stat_optional_package", schema = "db2_project")
public class PackageOptionalStatistics {

    @Id
    @Column(name = "id_package", insertable = false, updatable = false)
    private Long idPackage;
    @Id
    @Column(name = "id_optional", insertable = false, updatable = false)
    private Long idOptional;
    @Column(name = "num_purchases")
    private int numPurchases;
    @ManyToOne
    @JoinColumn(name = "id_package", nullable = false, updatable = false)
    private ServicePackage servicePackage;
    @ManyToOne
    @JoinColumn(name = "id_optional", nullable = false, updatable = false)
    private OptionalProduct optionalProduct;
```

Entity PackagePurchasesStatistics

```
public static class PackageValidityPeriod implements Serializable {
   private Long idPackage;
   private int validityPeriod;

   public PackageValidityPeriod() {
   }

   public PackageValidityPeriod(Long idPackage, int validityPeriod) {
       this.idPackage = idPackage;
       this.validityPeriod = validityPeriod;
   }
}
```

Entity PackagePurchasesStatistics

```
@Entity
@IdClass(PackagePurchasesStatistics.PackageValidityPeriod.class)
@Table(name = "stat_service_package", schema = "db2_project")
public class PackagePurchasesStatistics {

    @Id
    @Column(name = "id_package", insertable = false, updatable = false)
    private Long idPackage;
    @Id
    @Column(name = "validity_period", insertable = false)
    private int validityPeriod;
    @Column(name = "num_purchases")
    private int numPurchases;
    @Column(name = "tot_monthly_fee")
    private double totalMonthlyFee;
    @Column(name = "amount_for_optional_prods")
    private double amountForOptionalProds;
    @ManyToOne
    @JoinColumn(name = "id_package", nullable = false, updatable = false)
    private ServicePackage servicePackage;
```

Entity Service

```
@Entity
@Table(name = "service", schema = "db2_project")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "type", discriminatorType = DiscriminatorType.INTEGER, columnDefinition =
"TINYINT(1)")
public abstract class Service implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;
    @Column(name = "type")
    private int type;
```

Entity FixedPhone

```
@Entity
@Table(name = "service", schema = "db2_project")
@DiscriminatorValue ("2")
public class FixedPhone extends Service {
}
```

Entity FixedInternet

```
@Entity
@Table(name = "fixed_internet", schema = "db2_project")
@DiscriminatorValue("1")
public class FixedInternet extends Service {
    @Column(name = "n_gigabytes")
    private int nGigabytes;
    @Column(name = "fee")
    private double feeGigabytes;
```

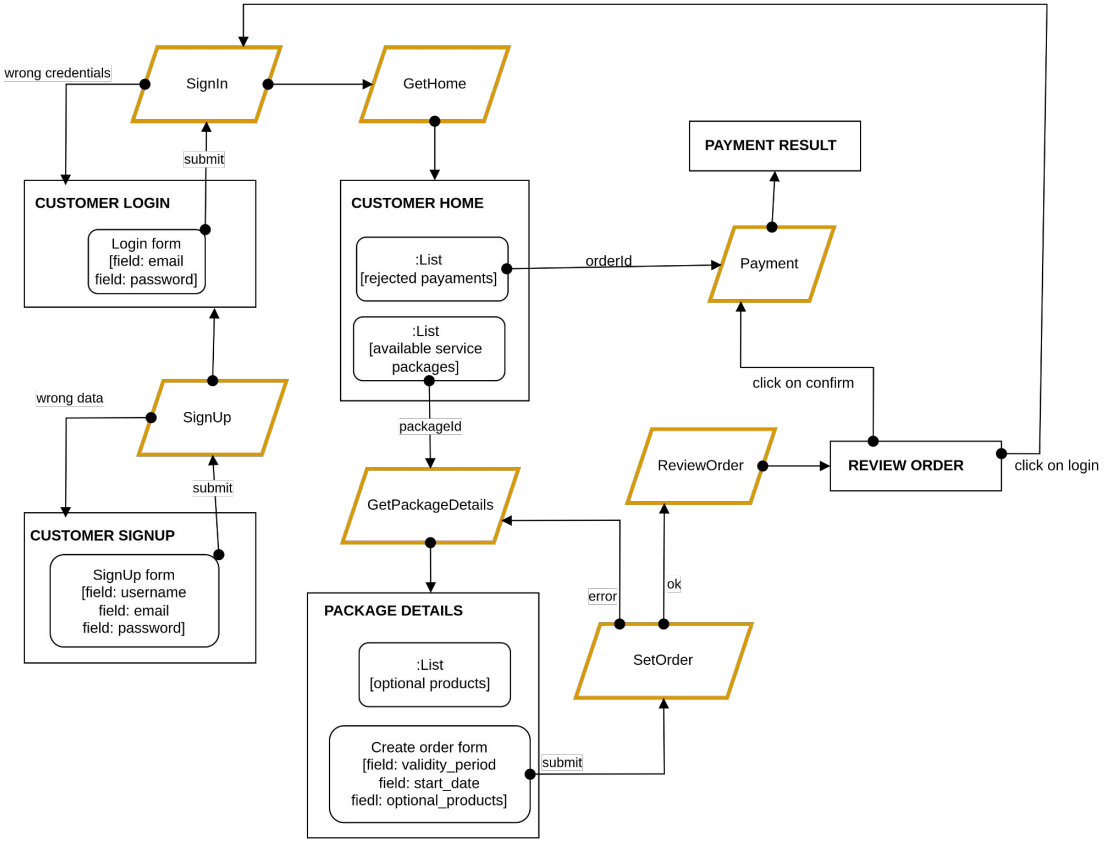
Entity MobilePhone

```
@Entity
@Table(name = "mobile_phone", schema = "db2_project")
@DiscriminatorValue("4")
public class MobilePhone extends Service {
    @Column(name = "n_minutes")
    private int nMinutes;
    @Column(name = "fee_minutes")
    private double feeMinutes;
    @Column(name = "n_sms")
    private int nSms;
    @Column(name = "fee_sms")
    private double feeSms;
```

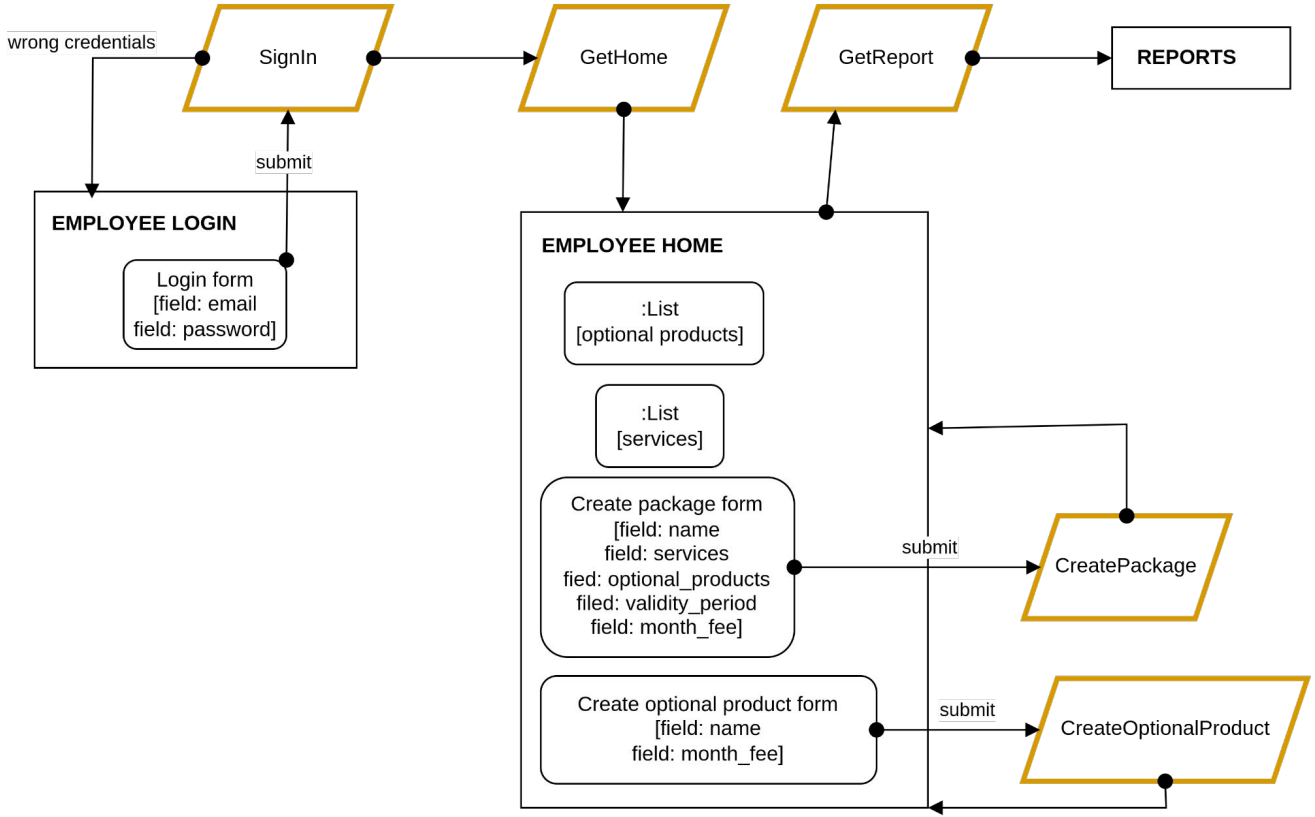
Entity MobileInternet

```
@Entity
@Table(name = "mobile internet", schema = "db2_project")
@DiscriminatorValue("3")
public class MobileInternet extends Service {
    @Column(name = "n_gigabytes")
    private int nGigabytes;
    @Column(name = "fee")
    private double feeGigabytes;
    public int getnGigabytes() {
        return nGigabytes;
    }
```

Functional analysis of the interaction



Functional analysis of the interaction



List of Components (Client)

- **Servlets (customer)**
  - SignIn
  - SignUp
  - Logout
  - GetHome: gets list of available packages and list of rejected orders (if user is logged in) and provide them to the home page
  - GetPackageDetails: a package id is provided. It gets the details of specified service package and provide it to the package details page. It's also the phase in which the buy process start.
  - SetOrder: check all details selected for the order, insert them in the state and call ReviewOrder servlet
  - ReviewOrder: get the order details from the state and provide to the user a summary of the order
  - Payment: get order from the state and save it into the database. Checks if the payment was successful

List of Components (Client)

- **Views (customer)**
  - SignInPage
  - SignUpPage
  - HomePage
  - PackageDetailsPage: displays details of selected package and contains a form to buy selected package
  - ReviewOrderPage: displays a summary of submitted order and offers the possibility to finalize the order by paying it
  - PaymentResultPage: displays the outcome of the revision of the payment

List of Components (Client)

- **Servlets (employee)**
  - SignIn
  - Logout
  - GetHome: get all services and all optional products in order to provide them to the employee home.
  - GetReport: get all stats from the database
  - CreatePackage: takes all informations in order to create a new service package
  - CreateOptionalProduct: takes all informations in order to create a new optional product
- **Views (employee)**
  - SignInPage
  - HomePage
  - ReportPage

List of Components (Back end)

- **Entities**
  - Customer
  - AuditCustomer
  - Employee
  - Order
  - ActivationSchedule
  - Offer
  - ServicePackage
  - OptionalProduct
  - ActivationSchedule
  - Service
  - FixedInternet
  - FixedPhone
  - MobileInteret
  - MobilePhone
  - PackageOptionalStatistics
  - PackagePurchasesStatistics

List of Components (Back end)

- **Business Components**
  - `@Statless` UserService
    - `Customer checkCredentialCustomer(String email, String password)`
    - `Employee checkCredentialEmployee(String email, String password)`
    - `Customer registerNewUser(String username, String email, String password)`
  - `@Statless` PackageService
    - `ServicePackage findById(Long id)`
    - `List<ServicePackage> getAvailableServicePackages()`
    - `List<Service> getAllService()`
    - `Service getServiceById(Long id)`
    - `Long createNewServicePackage(String name, List<Long> servicesIds, Listr<Long> optionalProductIds)`

List of Components (Back end)

- **Business Components**
  - `@Statless` OrderService
    - `List<Order> getRejectedOrdersByCustomer(Long customerId)`
    - `boolean updateCustomerOrderPayment(int onrderId, Long userId)`
  - `@Statless` OptionalProductService
    - `void createNewOptionalProduct(String name, Double monthlyFee)`
    - `OptionalProduct getOptionalProductById(Long id)`
    - `List<OptionalProduct> getAllOptionalProducts()`
  - `@Statless` OfferService
    - `void createNewOffer(Long packageId, int validityPeriod, double monthlyFee)`

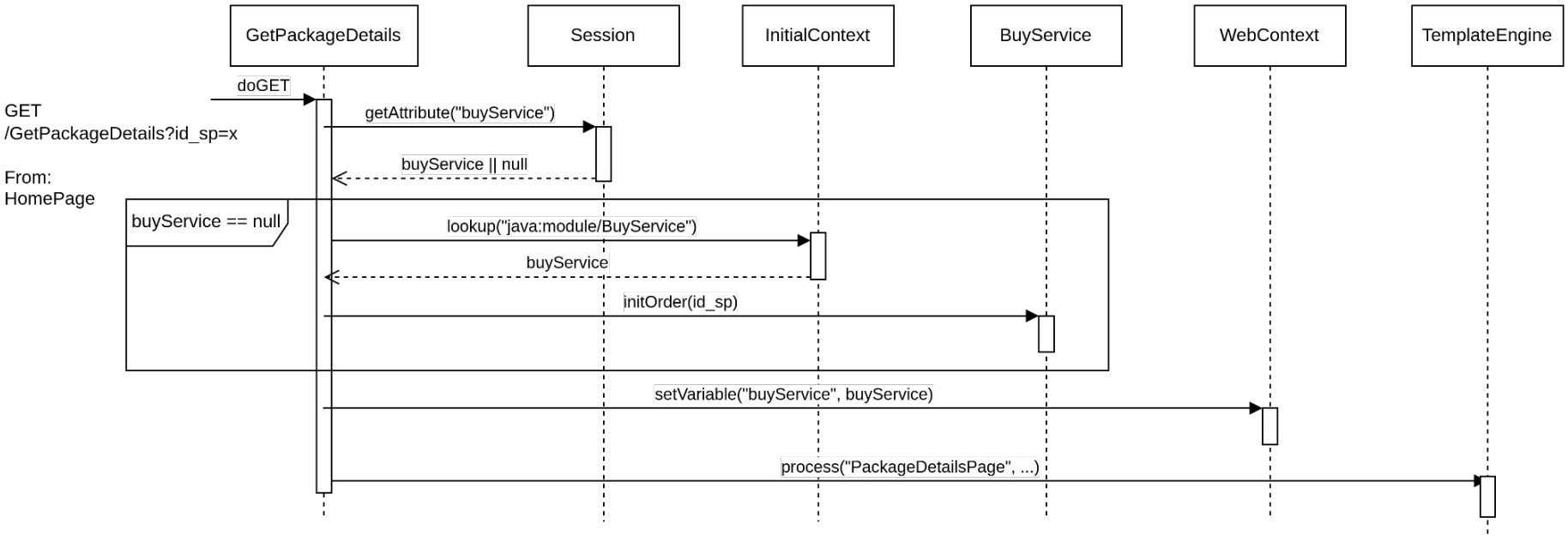
List of Components (Back end)

- **Business Components**
  - `@Statless` ReportService
    - `List<PackagePurchasesStatistics> getAllStatPackagePurchases()`
    - `List<PackageOptionalStatistics> getAllStatPackageOptional()`
    - `List<AuditCustomer> getAllAuditCustomer()`
    - `List<Order> getSuspendedOrder()`

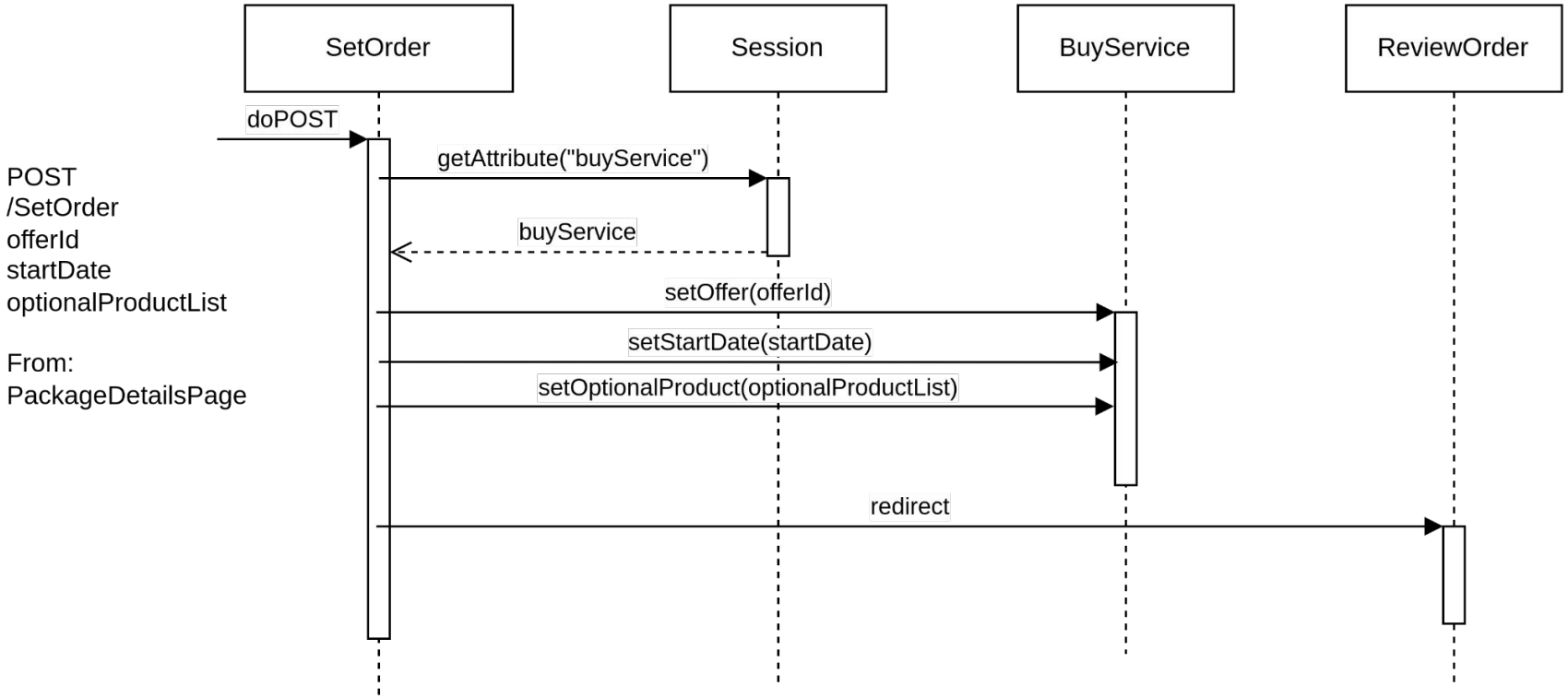
List of Components (Back end)

- **Business Components**
  - `@Statefull` BuyService
    - `void initOrder(Long serviceId)`
    - `void setOffer(Long offerId)`
    - `void setOptionalProducts(List<Integer> optionalProductIds)`
    - `void setStartDate(Date date)`
    - `Order getOrder()`
    - `ServicePackage getServicePackage()`
    - `Map<OptionalProduct, Boolean> getOptionalProducts()`
    - `@Remove boolean executePayment(Customer customer)`
    - `@Remove void stopProcess()`

UML sequence diagrams



UML sequence diagrams



UML sequence diagrams