POLITECNICO
MILANO 1863

M.Sc. Computer Science and Engineering
Data Bases 2 Project

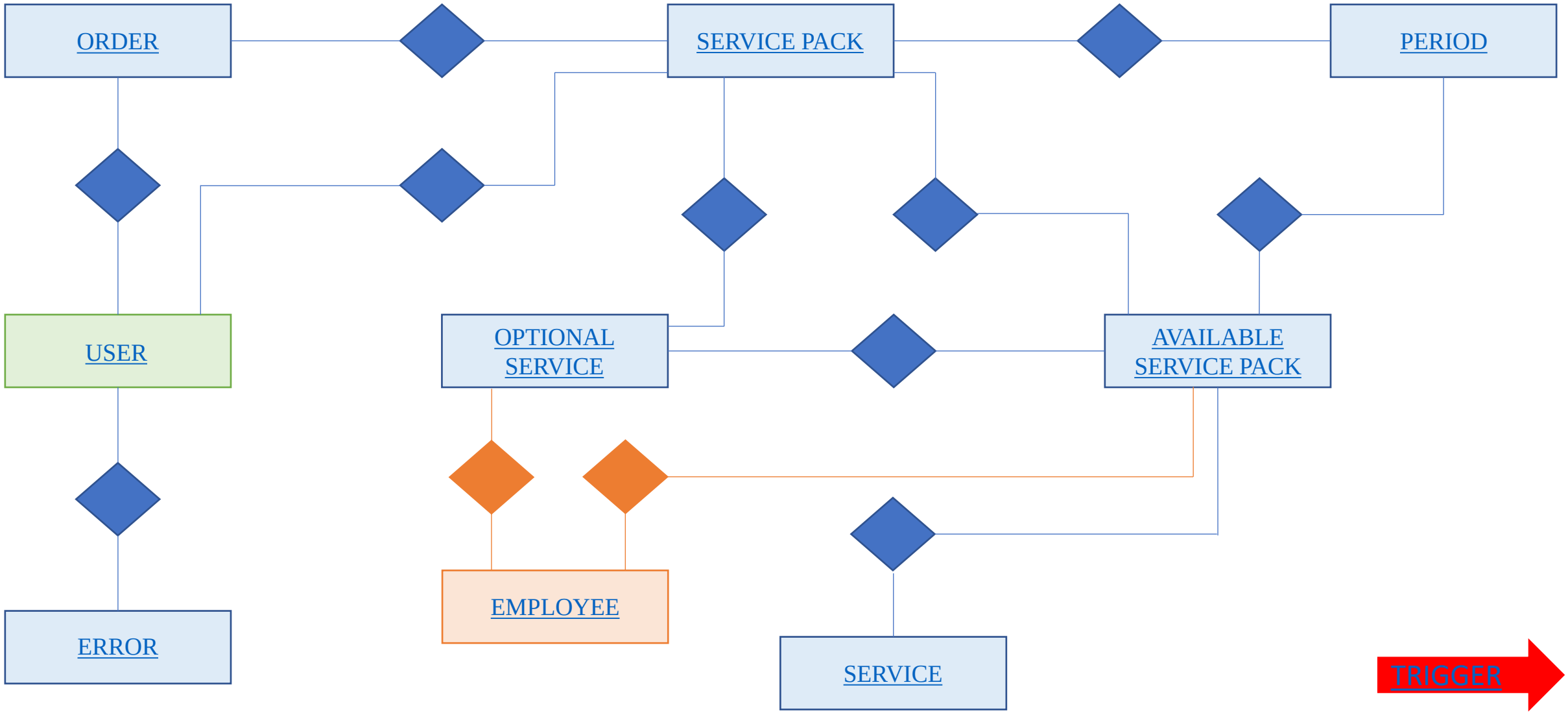
TELCO COMPANY

Censuales Simone (971685)
De Santis Gabriele (975789)

24th March 2022

GitHub Repository - https://github.com/titaniumwhite/DB2-project

ENTITY RELATIONSHIP DIAGRAM



USER RELATIONAL MODEL

```
create table user(
user_id            int auto_increment         primary key,
username           varchar(50)           not null,
name               varchar(50)           not null,
surname            varchar(50)           not null,
email              varchar(100)          not null,
password           varchar(50)           not null,
isInsolvent        tinyint(1) default 0 not null,
totFailedAttempts int         default 0 not null,
constraint user_email_uindex         unique (email),
constraint user_user_id_uindex         unique (user_id),
constraint user_username_uindex         unique (username)
);
```

JPA Implementation

EMPLOYEE RELATIONAL MODEL

```
create table employee(
employee_id       int auto_increment         primary key,
email             varchar(100)          not null,
password          varchar(50)           not null,
constraint employee_email_uindex      unique (email),
constraint employee_employee_id_uindex        unique (employee_id),
);
```

JPA Implementation

PERIOD RELATIONAL MODEL

```
create table period(
period_id   int auto_increment        primary key,
duration    int not null,
monthly_fee int not null
);
```

JPA Implementation

OPTIONAL SERVICE RELATIONAL MODEL

```
create table optional_service(
optional_service_id int auto_increment        primary key,
name                varchar(100) not null,
monthly_fee         int          not null
);
```

JPA Implementation

SERVICE RELATIONAL MODEL

```
create table service(
service_id         int auto_increment        primary key,
type               varchar(50) not null,
num_of_minutes     int         null,
num_of_SMS         int         null,
num_of_giga        int         null,
fee_extra_minutes  int         null,
fee_extra_sms      int         null,
fee_extra_giga     int         null
);
```

JPA Implementation

AVAILABLE SERVICE PACK RELATIONAL MODEL

```
create table available_service_package(
available_service_pack_id int auto_increment        primary key,
name                      varchar(50) not null,
constraint available_service_package_name_uindex        unique (name)
);
```

JPA Implementation

SERVICE PACK RELATIONAL MODEL

```
create table service_pack(
service_pack_id              int auto_increment        primary key,
start_date                   date          not null,
end_date                     date          not null,
cost                         int           not null,
total_cost_optional_services int default 0 not null,
available_package            int           not null,
period_service_pack          int           not null,
user_service_package         int           not null,
constraint service_pack_available_service_package__fk        foreign key
(available_package) references available_service_package
(available_service_pack_id),
constraint service_pack_period__fk        foreign key (period_service_pack)
references period (period_id),
constraint service_pack_user__fk        foreign key (user_service_package)
references user (user_id)
);
```

JPA Implementation

ERROR RELATIONAL MODEL

```
create table error(
error_id   int auto_increment        primary key,
tot_number int      not null,
timestamp  datetime not null,
user_error int      not null,
constraint error_user__fk        foreign key (user_error) references user
(user_id)
);
```

JPA Implementation

ORDER RELATIONAL MODEL

```
create table `order`(
order_id              int auto_increment        primary key,
timestamp_creation    datetime   not null,
total_cost            int        not null,
isPlaceable           tinyint(1) null,
`owner`             int        not null,
service_package_order int        not null,
constraint order_service_pack__fk        foreign key (service_package_order)
references service_pack (service_pack_id),
constraint order_user__fk        foreign key (`owner`) references user (user_id)
);
```

JPA Implementation

USER ENTITY

```
@Entity
@NamedQuery(
        name = "User.retrieveUserThroughID",
        query = "SELECT u FROM UserEntity u " +
                "WHERE u.id = :id"
)

@NamedQuery(
        name = "User.retrieveInsolventUser",
        query = "SELECT u FROM UserEntity u " +
                "WHERE u.isInsolvent = true"
)

@NamedQuery(
        name = "User.loginUser",
        query = "SELECT u FROM UserEntity u " +
                "WHERE u.username = :usn and u.password = :psw")

@Table(name = "user", schema = "dbproject2022")
public class UserEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

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

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

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

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

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

@Column(name = "isInsolvent")
private Boolean isInsolvent;

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

@OneToMany(targetEntity = ServicePackEntity.class, fetch =
FetchType.LAZY, cascade= {CascadeType.PERSIST, CascadeType.MERGE,
CascadeType.REFRESH, CascadeType.DETACH})
private List<ServicePackEntity> servicePackages;

@OneToMany(targetEntity = ErrorEntity.class, fetch = FetchType.EAGER,
mappedBy="owner", cascade = CascadeType.ALL, orphanRemoval = true)
private List<ErrorEntity> errors;
```

Return ER

EMPLOYEE ENTITY

```
@Entity
@NamedQuery(
        name = "Employee.loginEmployee",
        query = "SELECT e FROM EmployeeEntity e " +
                "WHERE e.username = :usn and e.password = :psw"
)

@Table(name = "employee", schema = "dbproject2022")
public class EmployeeEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

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

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

Return ER

PERIOD ENTITY

```
@Entity
@NamedQuery(
        name = "Period.findAll",
        query = "SELECT p FROM PeriodEntity p"
)

@NamedQuery(
        name = "Period.findPeriodThroughPackage",
        query = "SELECT p FROM PeriodEntity p " +
                "JOIN p.availableServicePackages s " +
                "WHERE s.availableServicePackId = :availableServicePackId "
)

@NamedQuery(
        name = "Period.findPeriodThroughID",
        query = "SELECT p FROM PeriodEntity p " +
                "WHERE p.periodId = :periodId"
)

@Table(name = "period", schema = "dbproject2022")
public class PeriodEntity implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "period_id", unique = true, nullable = false)
    private int periodId;

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

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

    @ManyToMany(mappedBy = "periods", fetch = FetchType.LAZY, cascade = CascadeType.MERGE)
    private List<AvailableServicePackEntity> availableServicePackages;
```

Return ER

OPTIONAL SERVICE ENTITY

```
@Entity
@NamedQuery(
        name = "OptionalService.findAll",
        query = "SELECT os FROM OptionalServiceEntity os " +
               "ORDER BY os.optionalService_id"
)
@NamedQuery(
        name = "OptionalService.findOptionalThroughPackage",
        query = "SELECT os FROM OptionalServiceEntity os " +
                "JOIN os.availableServicePackages a " +
                "WHERE a.availableServicePackId = :availableServicePackId "
)
@NamedQuery(
        name = "OptionalService.findServiceThroughID",
        query = "SELECT os FROM OptionalServiceEntity os " +
                "WHERE os.optionalService_id = :optionalService_id"
)
```

```
@Table(name = "optional_service", schema = "dbproject2022")
public class OptionalServiceEntity implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "optional_service_id", unique = true, nullable = false)
    private int optionalService_id;

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

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

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

@ManyToMany(mappedBy = "selectedOptionalServices", fetch = FetchType.EAGER)
private List<ServicePackEntity> servicePackages;
```

Return ER

SERVICE ENTITY

```
@NamedQuery(
        name = "Service.retrieveAllAvailableServicePackages",
        query = "SELECT s FROM ServiceEntity s"
)

@Entity
@Table(name = "service", schema = "dbproject2022")
public class ServiceEntity implements Serializable{

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

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

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

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

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

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

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

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

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

Return ER

AVAILABLE SERVICE PACK ENTITY

Return ER

```
@Entity
@NamedQuery(
        name = "AvailableServicePackage.findAll",
        query = "SELECT asp FROM AvailableServicePackEntity asp"
)
@NamedQuery(
        name = "AvailableServicePackage.findByID",
        query = "SELECT asp " +
                "FROM AvailableServicePackEntity asp " +
                "WHERE asp.availableServicePackId
= :availableServicePackId"
)
```

```
@Table(name = "available_service_package", schema = "dbproject2022")
public class AvailableServicePackEntity implements Serializable {

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

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

    @OneToMany(mappedBy="availablePackages", fetch =
           FetchType.LAZY,cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE,
            CascadeType.REFRESH,
            CascadeType.DETACH})
    private List<ServicePackEntity> offeredToPackage;

    @OneToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST,
CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
    private List<ServiceEntity> offeredServices;
```

```
@ManyToMany(fetch=FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(
        name="services_to_offer",
        joinColumns={@JoinColumn(name="available_service_pack_id")},
        inverseJoinColumns={@JoinColumn(name="service_id")}
)
private List<ServiceEntity> services;

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(
        name="period_to_offer",
        joinColumns={@JoinColumn(name="available_service_pack_id")},
        inverseJoinColumns={@JoinColumn(name="period_id")}
)
private List<PeriodEntity> periods;

@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(
        name="optional_services_to_offer",
        joinColumns={@JoinColumn(name="available_service_pack_id")},
        inverseJoinColumns={@JoinColumn(name="optional_service_id")}
)
private List<OptionalServiceEntity> optionalServices;
```

SERVICE PACK ENTITY

```
@Entity
@NamedQuery(
        name = "ServicePack.retrievePackageThroughOrderID",
        query = " SELECT s FROM ServicePackEntity s" +
                " WHERE s.order.orderId = :orderId"
)

@NamedQuery(
        name = "ServicePack.retrievePackageThroughID",
        query = " SELECT s FROM ServicePackEntity s" +
                " WHERE s.servicePackId = :servicePackId"
)

@Table(name = "service_pack", schema = "dbproject2022")
public class ServicePackEntity implements Serializable{

    private static final long serialVersionUID = 1L;

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

    @Column(name = "start_date", nullable = false)
    private java.sql.Date startDate;

    @Column(name = "end_date", nullable = false)
    private java.sql.Date endDate;

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

    @Column(name = "total_cost_optional_services", nullable = false)
    private int totalCostOptionalService;
```

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

    @ManyToMany (fetch = FetchType.EAGER, cascade = CascadeType.MERGE)
    @JoinTable(
            name="optional_services_selected",
            joinColumns={@JoinColumn(name="service_pack_id")},
            inverseJoinColumns={@JoinColumn(name="optional_service_id")}
    )
    private List<OptionalServiceEntity> selectedOptionalServices;


    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name ="period_service_pack")
    private PeriodEntity chosenPeriod;

    @ManyToOne (fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinColumn(name = "user_service_package")
    private UserEntity user_service_package;

    @OneToOne(mappedBy = "servicePackageOrder", cascade =
CascadeType.MERGE,          orphanRemoval = true)
    private OrderEntity order;
```

Return ER

ERROR ENTITY

```
@Entity
@NamedQuery(
        name = "Error.findAll",
        query = "SELECT e FROM ErrorEntity e " +
                "WHERE e.owner = :user "
)

@Table(name = "error", schema = "dbproject2022")
public class ErrorEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "error_id", unique = true, nullable = false)
    private int errorId;

    @Column(name = "tot_number", unique = true, nullable = false)
    private int totNumber;

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

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

Return ER

ORDER ENTITY

```
@Entity
@NamedQuery(
        name = "Order.retrieveAllUserOrderThroughID",
        query = " SELECT o FROM OrderEntity o" +
                " WHERE o.owner = :user and o.isPlaceable = true "
)
@NamedQuery(
        name = "Order.retrieveThroughID",
        query = " SELECT o FROM OrderEntity o " +
                " WHERE o.orderId = :orderId "
)
@NamedQuery(
        name = "Order.retrieveFailedUserOrder",
        query = " SELECT o FROM OrderEntity o " +
                " WHERE o.owner =: user AND o.isPlaceable = false"
)
@NamedQuery(
        name = "Order.retrievePendingOrder",
        query = " SELECT distinct o FROM OrderEntity o "+
                " JOIN o.servicePackageOrder s " +
                " WHERE o.owner = :user AND o.isPlaceable = false"
)
@NamedQuery(
        name = "Order.retrieveAllPendingOrder",
        query = " SELECT distinct o FROM OrderEntity o " +
                " WHERE o.isPlaceable = false "
)
@Table(name = "order", schema = "dbproject2022")
public class OrderEntity implements Serializable {

    private static final long serialVersionUID = 1L;

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

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

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

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

    @ManyToOne (targetEntity = UserEntity.class, fetch =
FetchType.EAGER,                    cascade = CascadeType.ALL, optional =
false)
    @JoinColumn(name = "owner")
    private UserEntity owner;

    @OneToOne (fetch = FetchType.EAGER, cascade = {
            CascadeType.PERSIST,
            CascadeType.MERGE,
            CascadeType.REFRESH,
            CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "service_package_order")
    private ServicePackEntity servicePackageOrder;
```

Return ER

TRIGGER
TOTAL PURCHASES PER PACKAGE

```
CREATE TABLE purchases_per_package(
    availableServicePack_id int not null primary key,
    totalOrder int default 0 not null,
    constraint purchases_per_package_fk
        foreign key (availableServicePack_id)
        references available_service_package (available_service_pack_id)
);

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertNewOrderOfPack AFTER INSERT ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        UPDATE purchases_per_package SET totalOrder = totalOrder + 1
        WHERE availableServicePack_id in (  SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = New.service_package_order);
    end if;
end //
delimiter ;
```

TRIGGER
TOTAL PURCHASES PER PACKAGE

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER updateServiceOrder AFTER UPDATE ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        UPDATE purchases_per_package SET totalOrder = totalOrder + 1
        WHERE availableServicePack_id IN (  SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = New.service_package_order);
    end if;
end //
delimiter ;

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER createNewAvailableServicePack_P1 AFTER INSERT ON available_service_package FOR EACH ROW
BEGIN
    INSERT INTO dbproject2022.purchases_per_package(availableServicePack_id) VALUES (NEW.available_service_pack_id);
end//
delimiter ;
```

TRIGGER
TOTAL PURCHASES PER PACKAGE AND PERIOD

```
CREATE TABLE purchases_per_package_and_period(
    availableServicePack_id int not null,
    period_id int not null,
    totalNumber int default 0 not null,
    constraint purchases_per_package_and_period_fk0
        foreign key (availableServicePack_id)
        references available_service_package (available_service_pack_id),
    constraint purchases_per_package_and_period_fk1
        foreign key (period_id)
        references period(period_id)
);
CREATE INDEX purchases_per_package_and_period_fk0_idx
    ON purchases_per_package_and_period(availableServicePack_id);

CREATE INDEX purchases_per_package_and_period_fk1_idx
    ON purchases_per_package_and_period(period_id);

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER createNewServPackWithPeriod AFTER INSERT ON period_to_offer FOR EACH ROW
BEGIN
    INSERT INTO purchases_per_package_and_period(availableServicePack_id, period_id)
        VALUES (NEW.available_service_pack_id, NEW.period_id);
end //
delimiter ;
```

TRIGGER
TOTAL PURCHASES PER PACKAGE AND PERIOD

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertNewPackageWithPeriod AFTER INSERT ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        UPDATE dbproject2022.purchases_per_package_and_period SET totalNumber = totalNumber + 1
        WHERE (availableServicePack_id, period_id) IN ( SELECT s.available_package, s.period_service_pack
                                                        FROM dbproject2022.service_pack s
                                                        WHERE s.service_pack_id = NEW.service_package_order);
    end if;
end//
delimiter ;

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER updateNewPackageWithPeriod AFTER UPDATE ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        UPDATE dbproject2022.purchases_per_package_and_period SET totalNumber = totalNumber + 1
        WHERE (availableServicePack_id, period_id) IN ( SELECT s.available_package, s.period_service_pack
                                                        FROM dbproject2022.service_pack s
                                                        WHERE s.service_pack_id = NEW.service_package_order);
    end if;
end//
delimiter ;
```

TRIGGER
SALES PER PACKAGE WITH AND WITHOUT OPTIONAL SERVICE

```
CREATE TABLE sales_per_package(
    availableServicePack_id int not null primary key,
    total_sales_with_optional int not null default 0,
    total_sales_no_optional int not null default 0,
    constraint sales_per_package_fk
        foreign key (availableServicePack_id)
        references available_service_package (available_service_pack_id)
);

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER createNewAvailableServicePack_P2 AFTER INSERT ON available_service_package FOR EACH ROW
BEGIN
    INSERT INTO sales_per_package(availableServicePack_id) VALUES (New.available_service_pack_id);
end//
delimiter ;
```

TRIGGER
SALES PER PACKAGE WITH AND WITHOUT OPTIONAL SERVICE

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertSale AFTER INSERT ON `order` FOR EACH ROW
BEGIN
    DECLARE a, b int;
    IF NEW.isPlaceable = true THEN
        SELECT s.cost, s.total_cost_optional_services INTO a, b
        FROM service_pack s
        WHERE s.service_pack_id = NEW.service_package_order;

        UPDATE sales_per_package spp
        SET spp.total_sales_with_optional = spp.total_sales_with_optional + a + b,
            spp.total_sales_no_optional = spp.total_sales_no_optional + a
        WHERE spp.availableServicePack_id IN (  SELECT s1.available_package
                                                FROM  service_pack s1
                                                WHERE s1.service_pack_id = NEW.service_package_order);
    end if;
end //
delimiter ;

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER updateSale AFTER UPDATE ON `order` FOR EACH ROW
BEGIN
    DECLARE a, b int;
    IF NEW.isPlaceable = true THEN
        SELECT s.cost, s.total_cost_optional_services INTO a, b
        FROM service_pack s
        WHERE s.service_pack_id = NEW.service_package_order;

        UPDATE sales_per_package spp
        SET spp.total_sales_with_optional = spp.total_sales_with_optional + a + b,
            spp.total_sales_no_optional = spp.total_sales_no_optional + a
        WHERE spp.availableServicePack_id IN (  SELECT s1.available_package
                                                FROM  service_pack s1
                                                WHERE s1.service_pack_id = NEW.service_package_order);
    end if;
end //
```

TRIGGER
AVERAGE NUMBER OF SERVICE PACK SOLD WITH EACH SERVICE PACKAGE

```
CREATE TABLE number_of_optionals(
    availableServicePack_id int not null primary key,
    total int default 0 not null,
    constraint number_of_optionals_fk
        foreign key (availableServicePack_id)
        references available_service_package (available_service_pack_id)
);

CREATE TABLE avg_numoptionservperservpack(
    availableServicePack_id int not null primary key,
    avgNum float default 0,
    constraint avg_numoptionservperservpack_fk
        foreign key (availableServicePack_id)
        references available_service_package (available_service_pack_id)
);

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER createNewAvailableServicePack_P3 AFTER INSERT ON available_service_package FOR EACH ROW
BEGIN
    INSERT INTO avg_numoptionservperservpack(availableServicePack_id)
        VALUES (NEW.available_service_pack_id);
end //
delimiter ;
```

TRIGGER
AVERAGE NUMBER OF SERVICE PACK SOLD WITH EACH SERVICE PACKAGE

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertOptionalService AFTER INSERT ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        DELETE FROM number_of_optionals t
        WHERE t.availableServicePack_id IN (SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = NEW.service_package_order);

        INSERT INTO number_of_optionals
        SELECT a.available_service_pack_id, count(*)
        FROM `order` AS o
                 JOIN dbproject2022.service_pack AS s ON o.service_package_order = s.service_pack_id
                 JOIN dbproject2022.available_service_package AS a ON s.available_package = a.available_service_pack_id
                 JOIN dbproject2022.optional_services_selected AS os ON os.service_pack_id = o.service_package_order
        WHERE o.isPlaceable = true and a.available_service_pack_id IN ( SELECT s.available_package
                                                                        FROM service_pack s
                                                                        WHERE s.service_pack_id = NEW.service_package_order)
        GROUP BY a.available_service_pack_id;

        DELETE FROM AVG_numOptionServPerServPack
        WHERE availableServicePack_id IN (  SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = NEW.service_package_order);

        INSERT INTO AVG_numOptionServPerServPack
        SELECT t.availableServicePack_id, IFNULL((o2.total / t.totalOrder), 0.0)
        FROM dbproject2022.purchases_per_package AS t
                 LEFT OUTER JOIN dbproject2022.number_of_optionals AS o2 ON t.availableServicePack_id = o2.availableServicePack_id
        WHERE t.availableServicePack_id IN (SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = NEW.service_package_order);
    end if;
end //
delimiter ;
```

TRIGGER
AVERAGE NUMBER OF SERVICE PACK SOLD WITH EACH SERVICE PACKAGE

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER updateOptionalService AFTER UPDATE ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN
        DELETE FROM number_of_optionals t
        WHERE t.availableServicePack_id IN ( SELECT s.available_package
                                             FROM service_pack s
                                             WHERE s.service_pack_id = NEW.service_package_order);

        INSERT INTO number_of_optionals
        SELECT a.available_service_pack_id, count(*)
        FROM `order` AS o
                 JOIN dbproject2022.service_pack AS s ON o.service_package_order = s.service_pack_id
                 JOIN dbproject2022.available_service_package AS a ON s.available_package = a.available_service_pack_id
                 JOIN dbproject2022.optional_services_selected AS os ON os.service_pack_id = o.service_package_order
        WHERE o.isPlaceable = true and a.available_service_pack_id IN ( SELECT s.available_package
                                                                        FROM service_pack s
                                                                        WHERE s.service_pack_id = NEW.service_package_order)
        GROUP BY a.available_service_pack_id;

        DELETE FROM AVG_numOptionServPerServPack
        WHERE availableServicePack_id IN (  SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = NEW.service_package_order);

        INSERT INTO AVG_numOptionServPerServPack
        SELECT t.availableServicePack_id, IFNULL((o2.total / t.totalOrder), 0.0)
        FROM dbproject2022.purchases_per_package AS t
                 LEFT OUTER JOIN dbproject2022.number_of_optionals AS o2 ON t.availableServicePack_id = o2.availableServicePack_id
        WHERE t.availableServicePack_id IN (SELECT s.available_package
                                            FROM service_pack s
                                            WHERE s.service_pack_id = NEW.service_package_order);
    end if;
end //
delimiter ;
```

TRIGGER
OPTIONAL SERVICE MOST SOLD

```
CREATE TABLE best_optional_service(
    optional_service_id int not null primary key,
    sales int not null,
    constraint best_optional_service_fk
        foreign key (optional_service_id) references optional_service (optional_service_id)
);

CREATE TABLE optional_service_order(
    optional_service_id int not null primary key,
    optional_service_sales int not null default 0
);

CREATE TABLE sales_per_optional_service(
    optional_service_id int not null,
    optional_service_sales int not null default 0
);

delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertOptionalServiceToBeSold AFTER INSERT ON optional_service FOR EACH ROW
BEGIN
    INSERT INTO sales_per_optional_service(optional_service_id)
    VALUES(NEW.optional_service_id);
end //
delimiter ;
```

TRIGGER
OPTIONAL SERVICE MOST SOLD

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER insertSaleOptionalService AFTER INSERT ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN DELETE FROM optional_service_order;
    INSERT INTO optional_service_order
    SELECT os.optional_service_id, (os.monthly_fee * p.duration)
    FROM `order` o
        JOIN service_pack s ON s.service_pack_id = o.service_package_order
        JOIN optional_services_selected oss ON oss.service_pack_id = s.service_pack_id
        JOIN period p ON p.period_id = s.period_service_pack
        JOIN optional_service os ON os.optional_service_id = oss.optional_service_id
    WHERE s.service_pack_id = NEW.service_package_order;

    UPDATE sales_per_optional_service s, optional_service_order ops
    SET s.optional_service_sales = s.optional_service_sales + ops.optional_service_sales
    WHERE s.optional_service_id = ops.optional_service_id;

    DELETE FROM best_optional_service;
    INSERT INTO best_optional_service
    SELECT s1.optional_service_id, s1.optional_service_sales
    FROM sales_per_optional_service s1
    WHERE s1.optional_service_id is not null
        AND s1.optional_service_sales IN (  SELECT MAX(s2.optional_service_sales)
                                            FROM sales_per_optional_service s2);
    END IF;
end //
delimiter ;
```

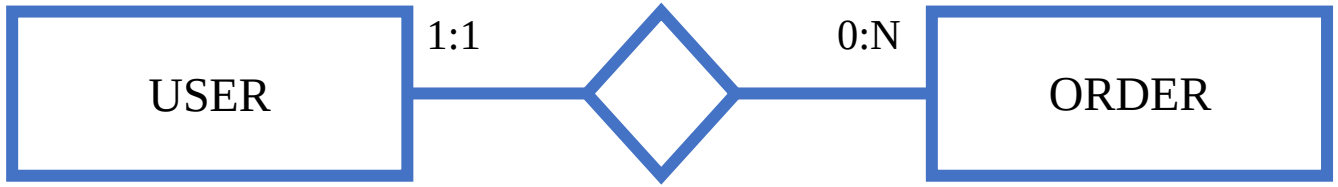
TRIGGER
OPTIONAL SERVICE MOST SOLD

```
delimiter //
CREATE DEFINER = CURRENT_USER TRIGGER updateSaleOptionalService AFTER UPDATE ON `order` FOR EACH ROW
BEGIN
    IF NEW.isPlaceable = true THEN DELETE FROM optional_service_order;
    INSERT INTO optional_service_order
    SELECT os.optional_service_id, (os.monthly_fee * p.duration)
    FROM `order` o
        JOIN service_pack s ON s.service_pack_id = o.service_package_order
        JOIN optional_services_selected oss ON oss.service_pack_id = s.service_pack_id
        JOIN period p ON p.period_id = s.period_service_pack
        JOIN optional_service os ON os.optional_service_id = oss.optional_service_id
    WHERE s.service_pack_id = NEW.service_package_order;

    UPDATE sales_per_optional_service s, optional_service_order ops
    SET s.optional_service_sales = s.optional_service_sales + ops.optional_service_sales
    WHERE s.optional_service_id = ops.optional_service_id;

    DELETE FROM best_optional_service;
    INSERT INTO best_optional_service
    SELECT s1.optional_service_id, s1.optional_service_sales
    FROM sales_per_optional_service s1
    WHERE s1.optional_service_id is not null
        AND s1.optional_service_sales IN (  SELECT MAX(s2.optional_service_sales)
                                            FROM sales_per_optional_service s2);
    END IF;
end //
delimiter ;
```

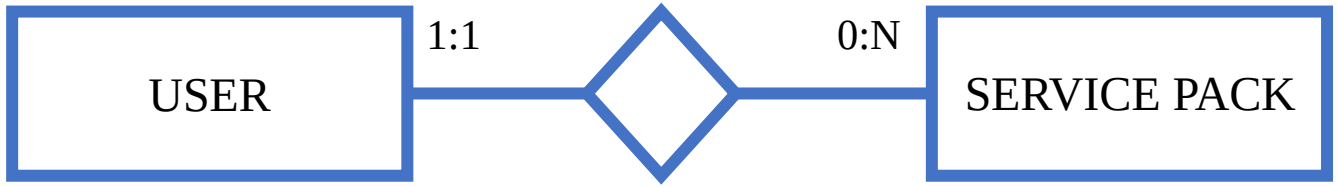
ORM RELATIONSHIP



User → Order
@OneToMany
→Lazy Fetch
→Cascade All
→Orphan Removal

Order → User
@ManyToOne
→Eager Fetch
→Cascade All

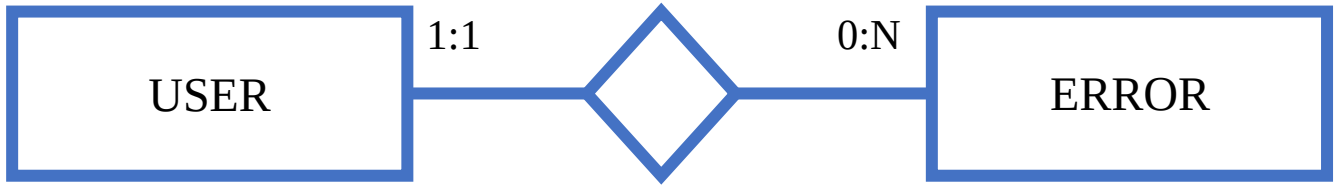
ORM RELATIONSHIP



User → ServicePack
@OneToMany
→Lazy Fetch
→Cascade All but Remove

ServicePack → User
@ManyToOne
→Eager Fetch
→Cascade All

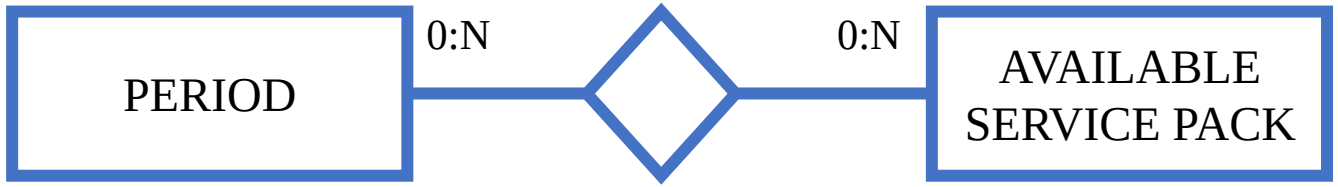
ORM RELATIONSHIP



User → Error
@OneToMany
→Eager Fetch
→Cascade All
→Orphan Removal

Error → User
@ManyToOne
→Eager Fetch
→Cascade All but Remove

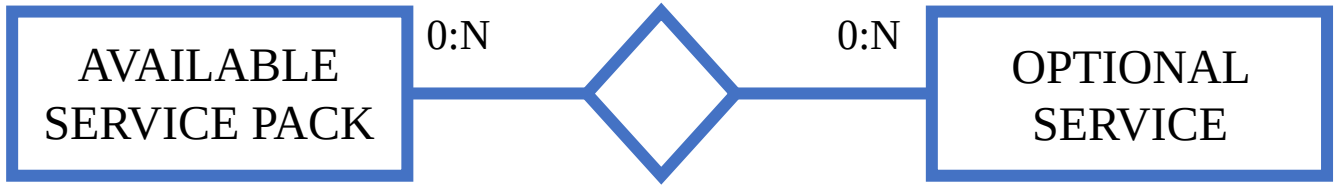
ORM RELATIONSHIP



Period → AvailableServicePack
@ManyToMany
→Lazy Fetch
→Cascade Merge Only
→Orphan Removal

AvailableServicePack → Period
@ManyToMany
→Eager Fetch
→Cascade All but Remove

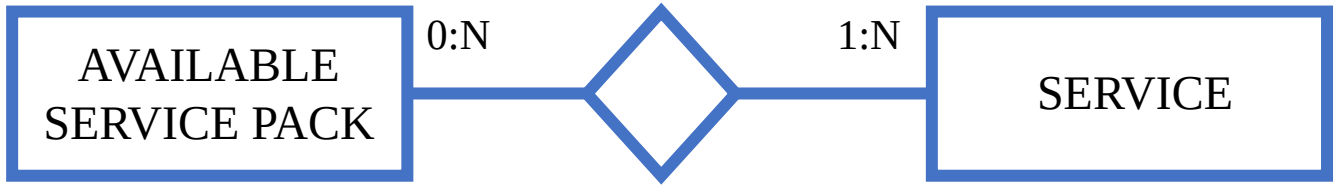
ORM RELATIONSHIP



AvailableServicePack → OptionalService
@ManyToMany
→Eager Fetch
→Orphan Removal

OptionalService → AvailableServicePack
@ManyToMany
→Lazy Fetch
→Cascade All but Remove

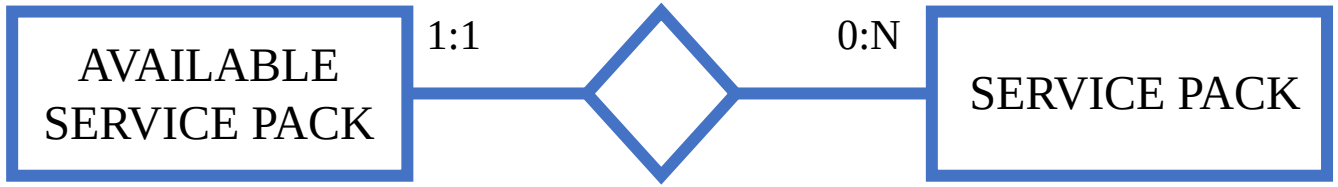
ORM RELATIONSHIP



AvailableServicePack → Service
@ManyToMany
→Eager Fetch
→Cascade All

Service → AvailableServicePack
@ManyToMany
→Lazy Fetch
→Cascade All

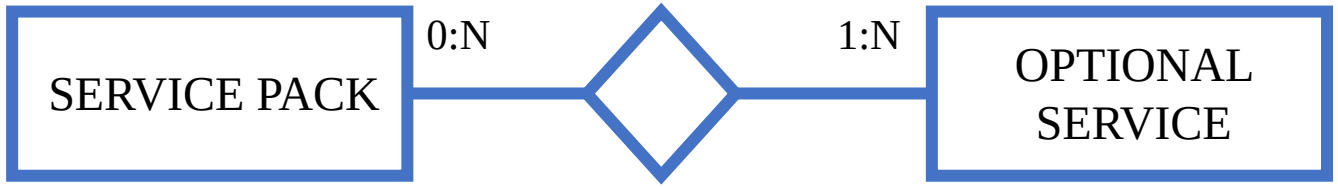
ORM RELATIONSHIP



AvailableServicePack → ServicePack
@OneToMany
→Lazy Fetch
→Cascade All but remove
→Orphan Removal

ServicePack → AvailableServicePack
@ManyToOne
→Eager Fetch
→Cascade All but Remove

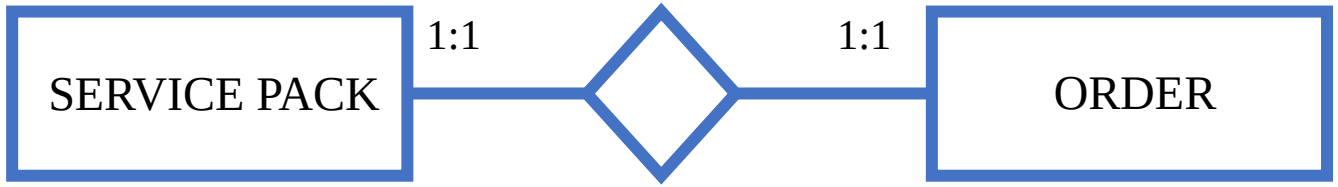
ORM RELATIONSHIP



ServicePack → OptionalService
@ManyToMany
→Eager Fetch
→Cascade Merge
→Orphan Removal

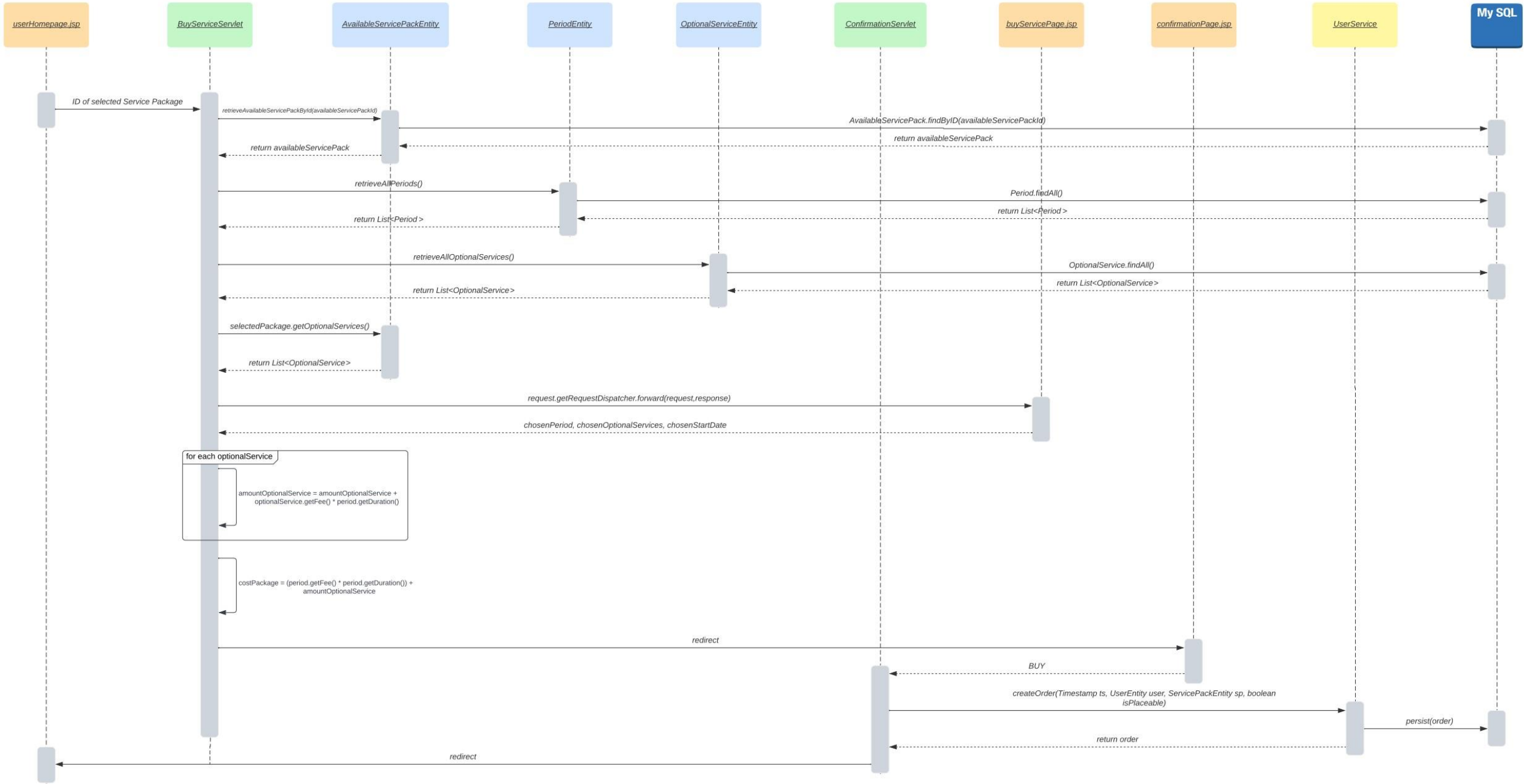
OptionalService → ServicePack
@ManyToMany
→Eager Fetch

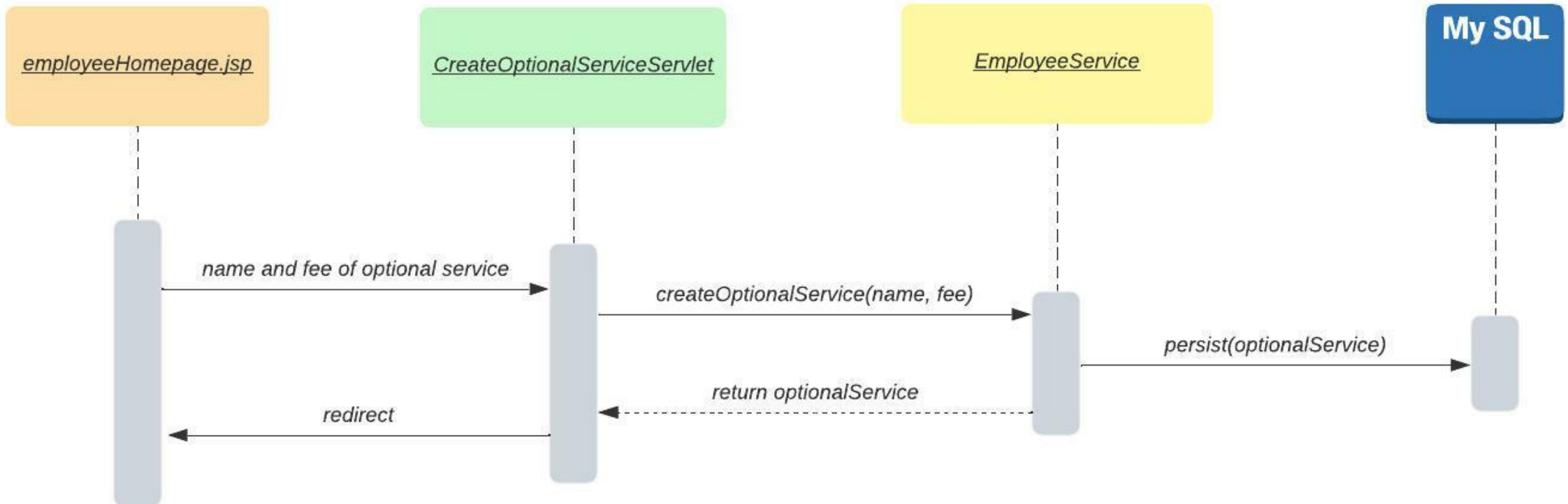
ORM RELATIONSHIP



ServicePack → Order
@OneToOne
→Cascade Merge
→Orphan Removal

Order → ServicePack
@ OneToOne
→Cascade All but Remove


userHomepage.jsp
BuyServiceServlet
AvailableServicePackEntity
PeriodEntity
OptionalServiceEntity
ConfirmationServlet
buyServicePage.jsp
confirmationPage.jsp
UserService
My SQL
ID of selected Service Package
retrieveAvailableServicePackById(availableServicePackId)
AvailableServicePack.findByID(availableServicePackId)
return availableServicePack
return availableServicePack
retrieveAllPeriods()
Period.findAll()
return List<Period>
return List<Period>
retrieveAllOptionalServices()
OptionalService.findAll()
return List<OptionalService>
return List<OptionalService>
selectedPackage.getOptionalServices()
return List<OptionalService>
request.getRequestDispatcher.forward(request,response)
chosenPeriod, chosenOptionalServices, chosenStartDate
for each optionalService
amountOptionalService = amountOptionalService + optionalService.getFee() * period.getDuration()
costPackage = (period.getFee() * period.getDuration()) + amountOptionalService
redirect
BUY
createOrder(Timestamp ts, UserEntity user, ServicePackEntity sp, boolean isPlaceable)
persist(order)
return order
redirect


employeeHomepage.jsp
CreateOptionalServiceServlet
EmployeeService
My SQL
name and fee of optional service
createOptionalService(name, fee)
persist(optionalService)
return optionalService
redirect

COMPONENTS

CLIENT TIER

- index.jsp
- userHomepage.jsp
- employeeHomepage.jsp
- confirmationPage.jsp
- buyServicePage.jsp
- salesReportPage.jsp

WEB TIER

- BuyServiceServlet
- ConfirmationServlet
- CreateOptionalServiceServlet
- CreatePackageServlet
- EmployeeHomepageServlet
- LoginServlet
- RegistrationServlet
- SalesReportServlet
- userHomepageServlet

COMPONENTS

Business Tier

UserService (Stateless)

- CheckUser(String username, String password)
- CreateUser(String username, String first_name, String last_name, String email, String password)
- GetAllServicePackages()
- RetrieveAvailableServicePackByID(int availableServicePackId)
- RetrieveServicePeriodID(int availableServicePackId)
- RetrieveOptionalOfAvailablePackage(int availableServicePackId)
- RetrieveAllPeriods()
- RetrievePeriodID(int periodId)
- RetrieveOptionalServicePackByID(int optionalService_id)
- RetrieveAllOptionalServices()
- RetrieveAllAvailableService()
- RetrieveAllOrdersOfUser(int userId)

COMPONENTS

Business Tier

UserService (Stateless)

- `RetrieveUserThroughID(int userId)`
- `RetrieveAllErrorsOfUser(int userId)`
- `CreateOrder(Timestamp ts, UserEntity user, ServicePackEntity sp, boolean isPlaceable)`
- `retrieveOrderThroughID(int orderId)`
- `orderUpdate(OrderEntity order, boolean isPlaceable)`
- `addFailedPayment(UserEntity user)`
- `setFailedPayments(UserEntity user)`
- `retrieveFailedOrderthroughUser(int userId)`
- `retrieveServicePackThroughId(int servicePackId)`
- `userIsInsolvent(UserEntity user, boolean isInsolvent)`
- `retrievePendingOrder(int userId)`
- `createServicePack(ServicePackEntity servicePack, UserEntity user)`
- `updateOrder(OrderEntity order, boolean isPlaceable)`

COMPONENTS

Business Tier

EmployeeService (Stateless)

- checkEmployee(String username, String password)
- createAvailableServicePack(String name, List<ServiceEntity> services, List<PeriodEntity> periods, List<OptionalServiceEntity> optionalServices)
- createOptionalService(String name, int fee)
- getAllOptionalServices()
- getAllPeriods()
- retrieveAllAvailableServicePackages()
- retrieveAllInsolventUsers()
- retrieveAllPendingOrders()
- retrieveAllErrors()
- retrieveBestOptionalProduct()
- retrieveAllPeriods()
- retrieveAvailablePackageThroughID(int servicePackId)
- retrieveAverageOptionalProductsPerPackage (int package_id)
- retrieveSalesPerPackage(int package_id)
- retrievePeriodById(int period_id)
- retrievePurchasesPerPackageAndPeriod(int package_id, int period_id)
- purchasesPerPackage(int package_id)