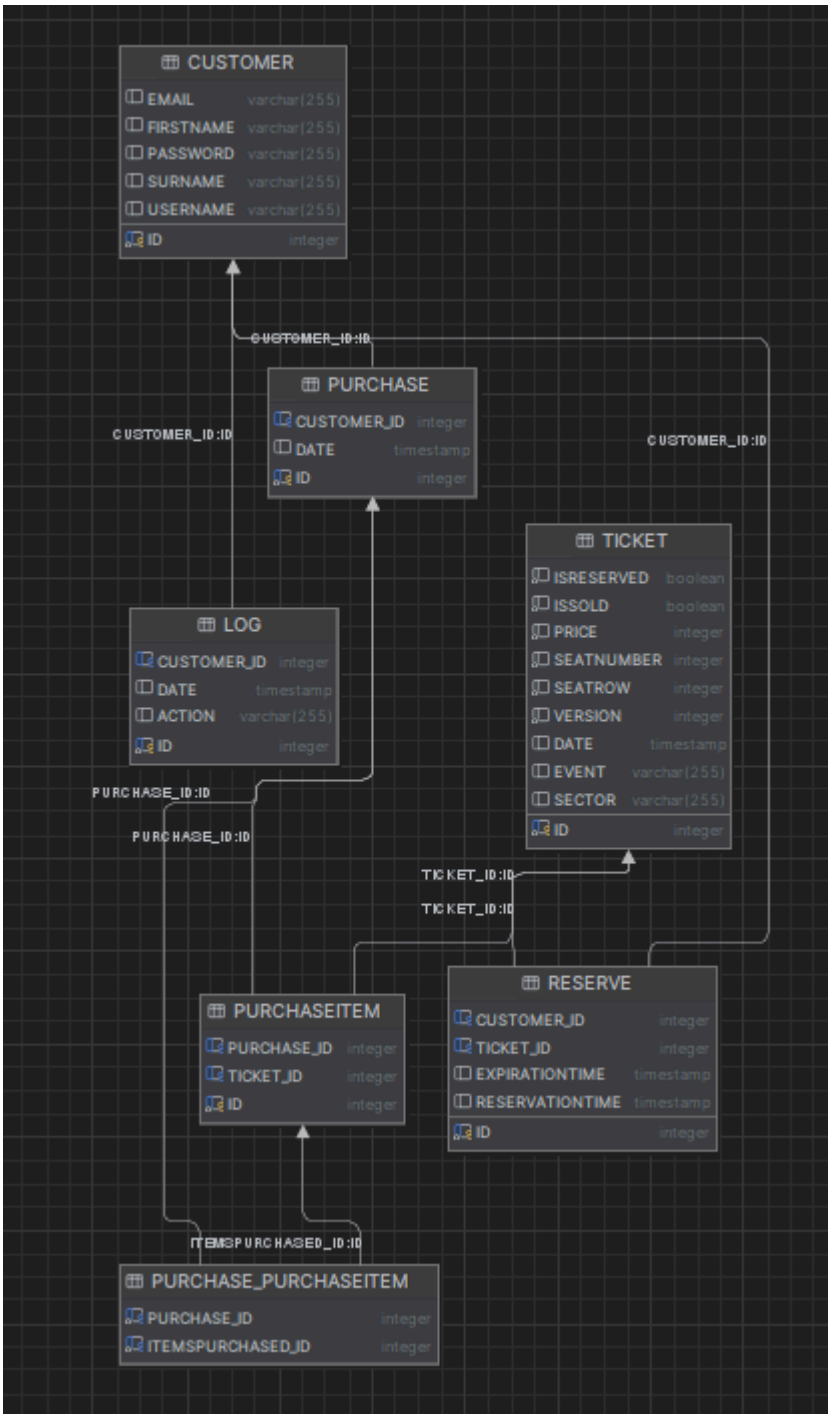
Projekt Bazy Danych

Java Hibernate + SQLLITE

Paweł Gałczyński

Diagram Bazy:



Tabele:

Customers:

```
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String username;
    private String firstname;
    private String surname;
    private String email;
    private String password;

    public Customer() {}

    public Customer(String username, String firstname, String
surname, String email, String password) {
        this.username = username;
        this.firstname = firstname;
        this.surname = surname;
        this.email = email;
        this.password = password;
    }

    public void printCustomer() {
        System.out.printf(
                "ID: %d, Username: %s, Firstname: %s, Surname: %s,
Email: %s%n",
                id, username, firstname, surname, email
        );
    }

    public static void printCustomers(List<Customer> customers) {
        for (Customer c : customers) {
            c.printCustomer();
        }
    }

    public int getId() {
        return id;
    }
}
```

Tickets:

```
package org.example.tables;

import jakarta.persistence.*;

import java.util.Date;
import java.util.List;

@Entity
public class Ticket {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   protected int id;
   private Date date;
   private String event;
   private int price;
   private String sector;
   private int seatRow;
   private int seatNumber;
   private boolean isReserved;
   private boolean isSold;
   @Version
   private int version;
   public Ticket(){}

   public Ticket(Date date, String event, int price, String
sector, int seatRow, int seatNumber) {
       this.date = date;
       this.event = event;
       this.price = price;
       this.sector = sector;
       this.seatRow = seatRow;
       this.seatNumber = seatNumber;
       this.isReserved = false;
       this.isSold = false;

   }

   public static void printTickets(List<Ticket> tickets) {
       for (Ticket t : tickets) {
           System.out.printf("ID: %d, Event: %s, Date: %s, Sector:
%s, Row: %d, Seat: %d, Price: %d, Sold: %b, Reserved: %b%n",
                   t.getId(),
                   t.getEvent(),
                   t.getDate(),
```

```
                t.getSector(),
                t.getSeatRow(),
                t.getSeatNumber(),
                t.getPrice(),
                t.isSold(),
                t.isReserved()
        );
    }
}



public String getEvent() {
    return event;
}

public int getId() {
    return id;
}
public boolean isReserved() {
    return isReserved;
}

public boolean isSold() {
    return isSold;
}

public void setReserved(boolean reserved) {
    isReserved = reserved;
}

public void setSold(boolean sold) {
    isSold = sold;
}

public int getSeatRow() {
    return seatRow;
}

public void setSeatRow(int seatRow) {
    this.seatRow = seatRow;
}

public int getSeatNumber() {
    return seatNumber;
}

public void setSeatNumber(int seatNumber) {
```

```
        this.seatNumber = seatNumber;
    }

    public String getSector() {
        return sector;
    }

    public void setSector(String sector) {
        this.sector = sector;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

Logs:

```
@Entity
public class Log {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private Date date;
    @ManyToOne
    private Customer customer;
    private String action;

    public Log() {}
    public Log(Date date, String action, Customer customer) {
        this.date = date;
        this.action = action;
        this.customer = customer;
    }
}
```

Purchase:

```
package org.example.tables;

import jakarta.persistence.*;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

@Entity
public class Purchase {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    @ManyToOne
    private Customer customer;
    @OneToMany
    private List<PurchaseItem> itemsPurchased = new
ArrayList<PurchaseItem>();
    private Date date;

    public Purchase() {}
    public Purchase(Customer customer) {
        this.customer = customer;
        this.date = new Date();
    }

    public void addItem(PurchaseItem item) {
        itemsPurchased.add(item);
    }
}
```

PurchaseItem:

```
@Entity
public class PurchaseItem {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    @ManyToOne
    private Purchase purchase;
    @ManyToOne
    private Ticket ticket;

    public PurchaseItem() {}
    public PurchaseItem(Purchase purchase, Ticket ticket) {
```

```
        this.purchase = purchase;
        this.ticket = ticket;
    }
}
```

Reserve:

```
package org.example.tables;

import jakarta.persistence.*;

import java.time.LocalDateTime;

@Entity
public class Reserve {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @ManyToOne
    private Customer customer;

    @ManyToOne
    private Ticket ticket;

    private LocalDateTime reservationTime;
    private LocalDateTime expirationTime;

    public Reserve() {}

    public Reserve(Customer customer, Ticket ticket) {
        this.customer = customer;
        this.ticket = ticket;
        this.reservationTime = LocalDateTime.now();
        this.expirationTime = LocalDateTime.now().plusMinutes(20);
    }

    public int getId() {
        return id;
    }

    public LocalDateTime getExpirationTime() {
        return expirationTime;
    }

}
```

Transakcje:
BuyTicket:

```
package org.example.transactions;

import jakarta.persistence.OptimisticLockException;
import org.example.tables.*;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.time.LocalDateTime;
import java.util.Date;
import java.util.List;

public class BuyTicket {
   public static void buy(Session session, List<Ticket>
ticketList, Customer customer) {
       Transaction tx = session.beginTransaction();
       try {
           Purchase purchase = new Purchase(customer);

           for (Ticket t : ticketList) {
               Ticket ticket = session.get(Ticket.class,
t.getId());

               if (ticket == null) throw new
RuntimeException("Ticket does not exist");
               if (ticket.isSold()) throw new
RuntimeException("Ticket is already sold");

               String hql = "FROM Reserve r WHERE r.ticket =
:ticket AND r.customer = :customer ORDER BY r.expirationTime
DESC";
               Reserve reservation = session.createQuery(hql,
Reserve.class)
                       .setParameter("ticket", ticket)
                       .setParameter("customer", customer)
                       .setMaxResults(1)
                       .uniqueResult();

               if (reservation == null) throw new
RuntimeException("Reservation not found");
               if
(reservation.getExpirationTime().isBefore(LocalDateTime.now())) {
                   throw new RuntimeException("Reservation has
expired");
               }
```

```
                ticket.setSold(true);
                PurchaseItem purchaseItem = new
PurchaseItem(purchase, ticket);
                session.persist(purchaseItem);
                purchase.addItem(purchaseItem);
            }
            Log log = new Log(new Date(),"BUY",customer);
            session.persist(log);
            session.persist(purchase);
            tx.commit();
        }
        catch (OptimisticLockException e) {
            tx.rollback();
            e.printStackTrace();
            System.out.println("racing condition");
        }

        catch (Exception e) {
            tx.rollback();
            e.printStackTrace();
            System.out.println("Nie udało się kupić biletów");
        }
    }
}
```

ReserveTickets:

```
package org.example.transactions;

import jakarta.persistence.OptimisticLockException;
import org.example.tables.Customer;
import org.example.tables.Log;
import org.example.tables.Reserve;
import org.example.tables.Ticket;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.time.LocalDateTime;
import java.util.Date;

// rezerwacja jest dokonywana wtedy gdy uzytkownik dodaje bilet do
koszyka i jest aktywna przez 20 min
public class ReserveTicket {
    public static void reserve(Session session, Customer customer,
Ticket t) {
            Transaction tx = session.beginTransaction();

            try{
```

```
                Ticket ticket =
session.get(Ticket.class,t.getId());

                if(ticket==null){
                    throw new RuntimeException("Ticket does not
exist");
                }
                else if (ticket.isSold()){
                    throw new RuntimeException("Ticket is Sold");
                }
                else if(ticket.isReserved()){
                    String HQL = "FROM Reserve r WHERE r.ticket.id
= :ticketId ORDER BY r.expirationTime DESC";
                    Reserve reservation = (Reserve)
session.createQuery(HQL,Reserve.class)

.setParameter("ticketId",ticket.getId())
                            .setMaxResults(1)
                            .uniqueResult();
                    LocalDateTime expiration = LocalDateTime.now();
                    if
(reservation.getExpirationTime().isBefore(expiration)) {
                        throw new RuntimeException("Ticket is
currently reserved");
                    }
                    else {
                        System.out.println("Ticket is available
last reservation is expired");
                    }
                }
                ticket.setReserved(true);
                Reserve reserve = new Reserve(customer, t);
                session.persist(reserve);
                Log log = new Log(new Date(),"RESERVE",customer);
                session.persist(log);
                tx.commit();
                System.out.println("Bilet zarezerwowany
pomyślnie");
            } catch (OptimisticLockException e) {
                tx.rollback();
                System.out.println("Nie udało się zarezerwować
biletu (ktoś go już zarezerwował)");
            } catch (Exception e) {
                tx.rollback();
                e.printStackTrace();
            }

            }
```

```
    }
```

Operacje CRUD:
CrudCustomer:

```
package org.example.CRUD;

import org.example.tables.Customer;
import org.hibernate.Session;

import java.util.List;

public class CrudCustomer {

    public static List<Customer> allCustomers(Session session) {
        String hql = "FROM Customer";
        return session.createQuery(hql,
Customer.class).getResultList();
    }

    public static Customer getCustomerById(Session session, int id)
{
        return session.get(Customer.class, id);
    }

    public static Customer getCustomerByUsername(Session session,
String username) {
        String hql = "FROM Customer c WHERE c.username =
:username";
        return session.createQuery(hql, Customer.class)
                .setParameter("username", username)
                .uniqueResult();
    }

    public static List<Customer> getCustomersBySurname(Session
session, String surname) {
        String hql = "FROM Customer c WHERE c.surname = :surname";
        return session.createQuery(hql, Customer.class)
                .setParameter("surname", surname)
                .getResultList();
    }
}
```

CrudTickets:

```
package org.example.CRUD;

import org.example.tables.Customer;
import org.example.tables.Ticket;
```

```java
import org.hibernate.Session;

import java.util.Date;
import java.util.List;

public class CrudTickets {

   public static List<Ticket> allAvailableTickets(Session session)
{
       String hql = "FROM Ticket t WHERE t.isSold = false";
       return session.createQuery(hql,
Ticket.class).getResultList();
   }

   public static List<Ticket> ticketsForConreteEvent(Session
session,String event,String sector) {

       String hql = "FROM Ticket t WHERE t.event = :event AND
t.sector = :sector";

       return session.createQuery(hql, Ticket.class)
               .setParameter("event", event)
               .setParameter("sector", sector)
               .getResultList();
   }

   public static List<Ticket> futureEvents(Session session) {
       String hql = "FROM Ticket t WHERE t.date > :now";
       return session.createQuery(hql, Ticket.class)
               .setParameter("now", new Date())
               .getResultList();
   }

   public static List<Ticket> priceRange(Session session,int
from,int to) {
       String hql = "FROM Ticket t WHERE t.price BETWEEN :min AND
:max";
       return session.createQuery(hql, Ticket.class)
               .setParameter("min", from)
               .setParameter("max", to)
               .getResultList();
   }

   public static List<Ticket> allReserved(Session session) {
       String hql = "FROM Ticket t WHERE t.isSold = true OR
t.isReserved = true";
       return session.createQuery(hql,
Ticket.class).getResultList();
```

```
    }

    public static List<Ticket> getPurchasedTickets(Session session,
Customer customer) {
        String hql = """
            SELECT pi.ticket
            FROM PurchaseItem pi
            WHERE pi.purchase.customer = :customer
        """;

        return session.createQuery(hql, Ticket.class)
                .setParameter("customer", customer)
                .getResultList();
    }

}
```

CrudReports:

```
package org.example.CRUD;

import org.hibernate.Session;

import java.time.LocalDate;
import java.time.LocalDateTime;

public class CrudReports {
    public static Long amountOfSoldTicketsLast7Days(Session
session) {
        String hqlCount = """
                        SELECT COUNT(pi)
                        FROM PurchaseItem pi
                        WHERE pi.purchase.date >= :lastWeek
                       """;
        LocalDateTime lastWeek = LocalDateTime.now().minusDays(7);

        return session.createQuery(hqlCount, Long.class)
                .setParameter("lastWeek", lastWeek)
                .getSingleResult();
    }

    public static Long totalRevenueLast7Days(Session session) {
        String hqlSum = """
            SELECT SUM(pi.ticket.price)
            FROM PurchaseItem pi
            WHERE pi.purchase.date >= :lastWeek
        """;

        LocalDateTime lastWeek = LocalDateTime.now().minusDays(7);
```

```
        Long totalRevenue = session.createQuery(hqlSum, Long.class)
                .setParameter("lastWeek", lastWeek)
                .getSingleResult();

        if (totalRevenue == null) totalRevenue = 0L;

        return totalRevenue;
    }

}
```

DataSeeder:

```
public class DataSeed {
    public static void seedCustomers(Session session) {
        List<Customer> customers = List.of(
                new Customer("jnowak", "Jan", "Nowak",
"jan.nowak@example.com", "haslo123"),
                new Customer("akowalska", "Anna", "Kowalska",
"anna.kowalska@example.com", "qwerty"),
                new Customer("tmalinowski", "Tomasz", "Malinowski",
"t.malinowski@example.com", "tajne123"),
                new Customer("mszpak", "Magda", "Szpak",
"magda.szpak@example.com", "magda2025"),
                new Customer("pzajac", "Piotr", "Zając",
"piotr.zajac@example.com", "piotrz!"),
                new Customer("klewandowski", "Kasia",
"Lewandowska", "kasia.lewandowska@example.com", "lewkas"),
                new Customer("dolszewski", "Dawid", "Olszewski",
"dawid.olszewski@example.com", "abc12345"),
                new Customer("akaczmarek", "Agnieszka",
"Kaczmarek", "agnieszka.kaczmarek@example.com", "agn2024"),
                new Customer("rmazur", "Robert", "Mazur",
"robert.mazur@example.com", "robmaz"),
                new Customer("nwróbel", "Natalia", "Wróbel",
"natalia.wrobel@example.com", "haslonatalia")
        );

        for (Customer customer : customers) {
            session.persist(customer);
        }
    }

    public static void seedTickets(Session session) {
        List<Ticket> tickets = List.of(
```

```
                new Ticket(Date.valueOf("2025-07-10"), "Koncert A",
100, "A", 1, 1),
                new Ticket(Date.valueOf("2025-07-10"), "Koncert A",
100, "A", 1, 2),
                new Ticket(Date.valueOf("2025-07-10"), "Koncert A",
100, "A", 1, 3),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert B",
150, "B", 2, 1),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert B",
150, "B", 2, 2),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert B",
150, "B", 2, 3),
                new Ticket(Date.valueOf("2025-07-12"), "Teatr C",
200, "C", 3, 1),
                new Ticket(Date.valueOf("2025-07-12"), "Teatr C",
200, "C", 3, 2),
                new Ticket(Date.valueOf("2025-07-12"), "Teatr C",
200, "C", 3, 3),
                new Ticket(Date.valueOf("2025-07-13"), "Koncert A",
100, "A", 4, 1),
                new Ticket(Date.valueOf("2025-07-13"), "Koncert A",
100, "A", 4, 2),
                new Ticket(Date.valueOf("2025-07-13"), "Koncert A",
100, "A", 4, 3),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert B",
150, "B", 5, 1),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert B",
150, "B", 5, 2),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert B",
150, "B", 5, 3),
                new Ticket(Date.valueOf("2025-07-10"), "Teatr C",
200, "C", 6, 1),
                new Ticket(Date.valueOf("2025-07-10"), "Teatr C",
200, "C", 6, 2),
                new Ticket(Date.valueOf("2025-07-10"), "Teatr C",
200, "C", 6, 3),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert A",
100, "A", 7, 1),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert A",
100, "A", 7, 2),
                new Ticket(Date.valueOf("2025-07-11"), "Koncert A",
100, "A", 7, 3),
                new Ticket(Date.valueOf("2025-07-12"), "Koncert B",
150, "B", 8, 1),
                new Ticket(Date.valueOf("2025-07-12"), "Koncert B",
150, "B", 8, 2),
                new Ticket(Date.valueOf("2025-07-12"), "Koncert B",
150, "B", 8, 3),
```

```
                new Ticket(Date.valueOf("2025-07-13"), "Teatr C",
200, "C", 9, 1),
                new Ticket(Date.valueOf("2025-07-13"), "Teatr C",
200, "C", 9, 2),
                new Ticket(Date.valueOf("2025-07-13"), "Teatr C",
200, "C", 9, 3),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert A",
100, "A", 10, 1),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert A",
100, "A", 10, 2),
                new Ticket(Date.valueOf("2025-07-14"), "Koncert A",
100, "A", 10, 3)
        );

        for (Ticket t : tickets) {
            session.persist(t);
        }
    }

}
```

Testy:
TicketTest:

```
class TicketTest {

    @Test
    public void testConstructorAndGetters() {
        Date date = new Date();
        Ticket ticket = new Ticket(date, "Koncert", 150, "A", 5,
12);

        assertEquals(date, ticket.getDate());
        assertEquals("A", ticket.getSector());
        assertEquals("Koncert", ticket.getEvent());
        assertEquals(150, ticket.getPrice());
        assertEquals(5, ticket.getSeatRow());
        assertEquals(12, ticket.getSeatNumber());
        assertFalse(ticket.isReserved());
        assertFalse(ticket.isSold());
    }

    @Test
    public void testSetters() {
        Ticket ticket = new Ticket();
        Date now = new Date();

        ticket.setPrice(200);
```

```
        ticket.setSeatNumber(3);
        ticket.setSeatRow(1);
        ticket.setSector("VIP");
        ticket.setReserved(true);
        ticket.setSold(true);
        ticket.setDate(now);

        assertEquals(200, ticket.getPrice());
        assertEquals(3, ticket.getSeatNumber());
        assertEquals(1, ticket.getSeatRow());
        assertEquals("VIP", ticket.getSector());
        assertTrue(ticket.isReserved());
        assertTrue(ticket.isSold());
        assertEquals(now, ticket.getDate());
    }

}
```