Úloha: CP-5 Základ JPA aplikace
Téma: Informační systém pro incomingovou cestovní kancelář
Student: Vladimir Zubkov, 2. sem. SIT
Předmět: B0B36DBS – Databázové systémy

Na databází CP-3 vystavte základ javové aplikace využívající JPA, který obsahuje zejména:

- datový model odpovídající celé databázi zahrnující:
  - „many-to-many“ vazbu; ✓
  - dědičnost. ✓
- DAO vrstvu zajišťující nutný nízkoúrovňový přístup k datům; ✓
- servisní vrstvu volající DAO vrstvu obsahující 5 vybraných užití vašich dat, specializovanou zejména na zapisovací operace a pokrývající transakci z CP-4. ✗

Odevzdává se formou odevzdání relevantní části Java-projektu, běh aplikace v rámci odevzdání je nutné předvést cvičícímu.

Výsledek:

Video-prezentace aplikace:

- https://drive.google.com/file/d/1GTyZu_Lji6aGHfAXgS9irqGQ-DEOJhDR/view?usp=sharing (29 min, ukázka aplikace ve 3. třetině)

GitHub:

- https://github.com/zubkovla/DBS-CP5

Dílčí implementace

Detail modelu dat (datových entit):

Několik poznámek ke tvorbě entit: datový model je tvořený entitami, třídami odpovídajícími tabulkám v databázi. Tyto třídy jsou tzv. „beany“ – veřejnými třídami, ve kterých jsou veřejné metody, gettery a settery (tzv. accesory), a privátní vlastnosti (fieldy)[1]. Anotují se vlastnosti (fieldy) nebo gettery entitních tříd; častěji se anotuje přes vlastnosti s tím odůvodněním, že se objekt při načtení z databáze nemá měnit, viz. odkaz na diskusi ve zdrojích. Konstruktor se v entitách obvyklé používá výchozí (tj. žádný definovaný uživatelem), ale pro aplikační logiku je rozumné vytvořit konstruktor s poli pro cizí klíč, případně druh entity – údaji, jež jsou známé při vytváření entit; zbytek se nastaví přes příslušné settery (rovněž diskuse ve zdrojích).

⌂ Anotace přes vlastnosti a přes přístupové metody, anotace „@JoinColumn“

- *Anotace vlastností:*

```
@Column(name = "SOME_STRING")
private String someString;
```

- *Anotace přes přístupové metody:*

```
@Column(name = "SOME_STRING")
public String getSomeString() {
    return someString;
}
public void setSomeString (String id) {
    this.someString = someString;
}
```

Anotace „@JoinColumn“ se používá pro mapování cizích klíčů:

```
@Entity
public class OwningEntity {
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "OTHER_ENTITY_ID")
private OtherEntity otherEntity;
}

@Entity
public class OtherEntity {
@OneToMany(mappedBy = “otherEntity”)
private Set<OwningEntity> owningEntities = new LinkedHashSet<>();
}
```

(zdroj)

[1] JavaBeans, dle striktní definice, beany mají být i serializovatelné (serializable): https://stackoverflow.com/a/3295517

⌂ Atribut „@Enumerated“ na základě datového typu „Enum“

Pozor, datový typ Enum podporuje pouze celá slova a podtržítko, a v případě potřeby konzistence s databázi, je možné použit podobnou konstrukci:

```
public enum FacilityType {
   HOTEL("Hotel"), SPA("Spa"), GUEST_HOUSE("Guest house");

   private final String toString;
   private FacilityType(String toString) {
      this.toString = toString;
   }
   public String toString() {
      return toString;
   }
}
```

Je ale lepé se tomu vyvarovat rovnou při definici hodnot v db, případně změnit přípustné hodnoty v db na přípustná celá slova. (zdroj)

NB: JPA poskytovatel podporuje konverzi datového typu Enum do hodnot odpovídajícího řetězce anebo hodnoty pořadí ve seznamu:

- *@Enumerated(EnumType.STRING)*
- *@Enumerated(EnumType.ORDINAL)*

Ovšem pozor, použití kovenze vracející pořadové číslo může vést k nekonzstenci dat, bude-li seznam někdy aktualizován, a uprostřed něj budou přidané nové položky – pořadí se změní. (zdroj)

Třída odpovídající entity:

```
@Entity
@Table(name = "facility")
public class Facility {
   @Id  @Column(name = "facilityid", nullable = false) private Integer id;

   @Enumerated(EnumType.STRING)
   @Column(name = "type", nullable = false, length = 15)
   private FacilityType type;

   public FacilityType getType() {
      return type;
   }
   public void setType(FacilityType type) {
      this.type = type;
   }
}
```

⌂ Složený atribut „@Embedded“ pro několik tabulek se zákazem jednoho z polí pro jednu z entit/tabulek

Třída, která nebude přímo mapována do tabulek, a bude použitá na více místech („@Embeddable“):

```
@Embeddable
@Access(AccessType.FIELD)
public class Address {
   @Column(name = "city", nullable = false, length = 50)
   private String city;
   @Column(name = "street", nullable = false, length = 150)
   private String street;
   @Column(name = "house", nullable = false)
   private Integer house;
   @Column(name = "postcode", length = 10)
   private String postcode;
// getters and setters
}
```

Třída „společnost“ s adresou („@Embedded“):

```
@Entity
@Table(name = "company")
public class Company {
...
@Embedded
private Address address;
...
}
```

Třída „ubytovací zařízení“ s adresou nepoužívá pole „postcode“:

```
@Entity
@Table(name = "facility")
public class Facility {
...
@Embedded
@AttributeOverrides({
      @AttributeOverride(name = "postcode", column = @Column(nullable = false, insertable = false))
})
private Address address;
...
}
```

(zdroj)

⌂ Obousměrná rekurzivní vazba „Many-To-Many“ (společnosti v partnerství):

Tabulka v databázi:

```
CREATE TABLE IF NOT EXISTS companyPartnership (
   companyId INTEGER, -- FK
   companyPartnerId INTEGER, -- FK
   type VARCHAR(20) NOT NULL DEFAULT 'Seasonal',
   PRIMARY KEY (companyId, companyPartnerId),
   CONSTRAINT companyPartnership_ck_type CHECK (type = 'Seasonal' OR
type = 'Long-term'),
   CONSTRAINT companyPartnership_fk_companyId
       FOREIGN KEY (companyId)
       REFERENCES company(companyId)
       ON UPDATE CASCADE
       ON DELETE CASCADE,
   CONSTRAINT companyPartnership_fk_companyPartnerId
       FOREIGN KEY (companyPartnerId)
       REFERENCES company(companyId)
       ON UPDATE CASCADE
       ON DELETE CASCADE
);
```

Odpovídající tabulce vztah v entitě „Company“:

```
@Entity
@Table(name = "company")
public class Company {
   @Id
   @Column(name = "companyid", nullable = false)
   private Integer id;

   @ManyToMany(cascade = {
CascadeType.PERSIST, CascadeType.MERGE })
   @JoinTable(name="companypartnership",
         joinColumns=@JoinColumn(name="companyId"),
         inverseJoinColumns=@JoinColumn(name="companyId")
   )
   private List<Company> company;

  @ManyToMany(cascade = {
CascadeType.PERSIST, CascadeType.MERGE })
   @JoinTable(name="companypartnership",
         joinColumns=@JoinColumn(name="companyId"),
         inverseJoinColumns=@JoinColumn(name="companyId")
   )
   private List<Company> companyPartner;
...
```

(Zdroje:
- implementace vztahu Many-to-Many,
- použití kaskádového typu,
- přehled kaskádových typů)

⌂ Dědičnost

„Atrakce“ a „Doprava“ jsou kompletními exkluzívnímu podtypy entity „Služba“. Pro každou entitu se používá zvláštní tabulka, proto aplikujeme strategii „jednotlivé tabulky pro jednotlivé třídy“ – *„@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)“:*

Mapování je následující:

```
@Entity
@Table(name = "service")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Service {
    @Id
    @Column(name = "serviceid", nullable = false)
    private Integer id;
    @Column(name = "name", nullable = false, length = 150)
    private String name;
... }
```

```
@Entity
@Table(name = "transport")
public class Transport extends Service{
    @Id
    @Column(name = "transportplate", nullable = false, length = 20)
    private String plate;

    @ManyToOne
    @JoinColumn(name = "serviceid")
    private Service serviceid;
...
    public String getPlate() { return plate; }
    public void setPlate(String plate) { this.plate = plate; }
    public Service getServiceid() { return serviceid; }
    public void setServiceid(Service serviceid) { this.serviceid = serviceid; }
...
}
```

Všimněte si toho, že id v podřazených třídách musí mít jiný název, aby se zamezilo kolizím nebo přepsání s id třídy „service“ , to platí i pro stejně pojmenovavaná pole (viz. @Column(name = "name"...):

```
@Entity
@Table(name = "attraction")
public class Attraction extends Service {
    @Id
    @Column(name = "attractionid", nullable = false)
    private Integer attractionid;

    @ManyToOne
    @JoinColumn(name = "serviceid")
    private Service serviceid;

    @Column(name = "name", nullable = false, length = 100)
    private String attraction;
...
    public Integer getAttractionid() { return attractionid; }
    public void setAttractionid(Integer attractionid) { this.attractionid = attractionid; }
    public Service getServiceid() { return serviceid; }
    public void setServiceid(Service serviceid) { this.serviceid = serviceid; }
    public String getAtraction() { return attraction; }
    public void setAtraction(String atraction) { this.attraction = atraction; }
... }
```

Problematika dědičnosti je širší, průvodcem mohou sloužit odkazy v tomto zdroji.

Některé chyby, které mohou nastat:

– „Multiple writable mappings exist for the field“

Výpis trasování:

Exception Description: Multiple writable mappings exist for the field [passenger.passengerid].
Only one may be defined as writable, all others must be specified read-only.
Mapping: org.eclipse.persistence.mappings.ManyToOneMapping[passengerid]
Descriptor: RelationalDescriptor(jpa.modelEntities.Itinerary --> [DatabaseTable(passenger)])

Řešení:

Chyba je výsledkem toho, že několik mapování je schopno zapisování do vlastnosti (fieldu). Řešením je přidání *„insertable = false, updatable = false“* do definice *@JoinColumn* takto v jedné z vlastnících tabulek, které vyvolávají konflikt:

místo

```
@JoinColumn(name = "someEntityId")
```

použij

```
@JoinColumn(name = "someEntityId", insertable = false, updatable = false)
```

Zdroj: https://www.eclipse.org/forums/index.php/t/485946/

Zdroje pro definici modelu dat (datových entit):

- Anotace „@JoinColumn“: https://www.baeldung.com/jpa-join-column
- Dobrý úvod do JPA: http://voho.eu/wiki/java-jpa/
- Příklad anotování v EclipseLink: https://wiki.eclipse.org/EclipseLink/Examples/JPA/JSF_Tutorial
- Specifikace JPA 2.2: https://jcp.org/en/jsr/detail?id=338
- Je lepší mapovat přímo přes vlastnosti (pole – fieldy) nežli přes přístupy k vlastnostem (tzv. accesory – gettery):
  - Co je lepší, anotace vlastností či přístupových metod? – vlastnosti: https://stackoverflow.com/a/6084701
    – přístupové metody jsou jedinou možnosti při anotací vlastností z abstraktní nadřazené třídy: https://stackoverflow.com/a/3184011
  - Access Strategies in JPA and Hibernate – Which is better, field or property access?
    field: https://thorben-janssen.com/access-strategies-in-jpa-and-hibernate/
- „Tvorba perfektních JPA entit“ – Create the perfect JPA entity: https://stackoverflow.com/a/14822709
- Automatická tvorba tříd na základě existující databáze v IntelliJ IDEA pomocí pluginu JPA Buddy: https://www.jpa-buddy.com/documentation/reverse-engineering/
- Objektově-relační mapování v EclipseLink (ORM): https://www.eclipse.org/eclipselink/documentation/2.7/concepts/blocks002.htm
- Oficiální Java EE 6 Persistence API tutoriál: https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html
- Specifikace balíčku „Jakarta Persistence API“ (JPA):
  https://jakarta.ee/specifications/persistence/2.2/apidocs/javax/persistence/package-summary.html
- Tvorba a konfigurování JPA entit v EclipseLink:
  https://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Entities/Creating_and_Configuring_Entities
- Nastavení persistence.xml: https://docs.oracle.com/cd/E16439_01/doc.1013/e13981/cfgdepds005.htm
- Podrobný příklad nastavení persistence.xml: https://thorben-janssen.com/jpa-persistence-xml/
- Složené atributy „@Embeddable“ a „@Embedded“ v JPA: https://www.baeldung.com/jpa-embedded-embeddable
- Mapování s použitím datových typů „kolekce, list, set, map“: https://www.javatpoint.com/jpa-collection-mapping
- Diskuse o použití datových typů ve vztazích OneToMany a ManyToMany (kolekce, set, list): https://stackoverflow.com/a/4655417
- Diskuse o výkonu a alokace paměti pro datové typy „set“ a „list“: https://stackoverflow.com/a/10799483

Detail uspořádání přístupu k datům (DAO):

Poznámka: při definici vrstvy zajišťující nutný nízkoúrovňový přístup k datům, jako je „tvorba, čtení, aktualizace, mazání“ (CRUD), je vhodné použití generických DAO tříd na základě implementace abstraktních tříd a rozhraní. Toto je ale mimo rozsah tohoto zadání.

Δ Nastavení EntityManager, EntityManagerFactory v podobě Singletonu (* otázka je-li to správně):

```
public class EntityManagerUtility {
   // Name of persistence context from persistence.xml
   private static final String PERSISTENCE_UNIT_NAME = "TOUR_AGENCY_PU_JPA_TEST";
   // Static variable reference of type EntityManagerFactory
   private static EntityManagerFactory emf;
   // Static variable reference of type EntityManager
   private static EntityManager em;
   // Constructor of the class, which prevents object creation outside the class
   private EntityManagerUtility() {
   }
   // Instantiation of EntityManager class
   public static EntityManager getEm() {
      if (em == null) {
         // same as in persistence.xml
         emf = Persistence.createEntityManagerFactory("TOUR_AGENCY_PU_JPA_TEST");
         em = emf.createEntityManager();
         return em;
      }
      return em;
   }
   // Close EntityManager and EntityManagerFactory
   public void close() {
      this.em.clear(); // this may be obsolete, but is used for consistency
      this.em.close();
      this.emf.close();
   }
}
```

Δ Použití EntityManageru a rozhraní EntityTransaction v DAO:

```
// instance of EntityManager from EntityManagerUtility
private EntityManager em = EntityManagerUtility.getEm();

// a shortcut to EntityTransaction interface
private EntityTransaction et = em.getTransaction();
...
public x metoda() {
et.begin();
   // a query or some action with persisted entity
et.commit();
em.clear();
}

Atd…
```

⌂ Kontrola připojení k databázi *(em.find)* – R

- Metoda:

```
public Passenger findPassenger(Integer id) {
    Session session = em.unwrap(Session.class); // get session object from EntityManager
    System.out.println("Sesson: " + session.toString());
    et.begin();
    java.sql.Connection connection = em.unwrap(java.sql.Connection.class); // check database connection
    et.commit();
    System.out.println("Connected to: " + connection.toString());
    Passenger passenger = em.find(Passenger.class, id); // find a passenger by id by native EntityManager's 'find' method
    if (passenger != null) {
        System.out.println("Passenger found: " + passenger.toString());
    } else {
        System.out.println("Passenger with id " + id + " was not found in the database, return " + passenger);
    }
    return passenger;
}
```

- Volání:

```
PassengerRepository passengerRepository = new PassengerRepository();
passengerRepository.findPassenger(1);
passengerRepository.findPassenger(36001); // out of datarange
```

- Výsledek:

```
Sesson: ServerSession(
    DatabaseAccessor(connected)
    PostgreSQLPlatform)
Connected to: org.postgresql.jdbc.PgConnection@61078690
Passenger found: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@2c715e84, name='Damian', surname='McCarl', dateofbirth=1951-12-06, arrivalticketno='y
--------------------
Passenger with id 36001 was not found in the database, return null
```

⌂ Aktualizace dat *(em.update)* – U

- Metoda JPQL:

```
public Passenger updatePassengerFirstNameByIdJPQL(String name, Integer id) {
    System.out.println("This passenger will be updated: " + em.find(Passenger.class, id).toString());
    et.begin();
    // prevent SQL injection by avoiding concatenating with user-controlled variables, using parametrised queries instead
    Query query = em.createQuery("update Passenger set name = :name where id = :id");
    query.setParameter("name", name);
    query.setParameter("id", id);
    query.executeUpdate();
    et.commit(); // enter new data to the database
    Passenger passengerUpdated = em.find(Passenger.class, id); // add an entity to persistence context, however it is loaded from repository
    em.refresh(passengerUpdated); // refresh entity manager object from database - this is needed to return actual db-data
    em.clear(); // clear the persistence context
    System.out.println("Return updated passenger from database: " + passengerUpdated.toString());
    return passengerUpdated;
}
```

- Metoda entitního manažera:

```
public Passenger updatePassengerFirstNameByIdEntityManager(String name, Integer id) {
    Passenger passengerUpdated = em.find(Passenger.class, id); // add an entity to persistence context
    System.out.println("This passenger will be updated: " + passengerUpdated.toString());
    et.begin();
    passengerUpdated.setName(name);
    et.commit(); // enter new data to the database
    em.clear(); // clear the persistence context
    System.out.println("Return updated passenger from database: " + passengerUpdated.toString());
    return passengerUpdated;
}
```

– Volání:

```
PassengerRepository passengerRepository = new PassengerRepository();
passengerRepository.updatePassengerFirstNameByIdEntityManager("Stinky", 1);
passengerRepository.updatePassengerFirstNameByIdJPQL("Damian", 1);
```

– Výsledek:

```
This passenger will be updated: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@12cd9150, name='Damian', surname='McCarl', dateofbirth=
Return updated passenger from database: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@12cd9150, name='Stinky', surname='McCarl', date
This passenger will be updated: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@2e77b8cf, name='Stinky', surname='McCarl', dateofbirth=
Return updated passenger from database: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@2e77b8cf, name='Damian', surname='McCarl', date
```

Je vidět, že metoda používající přímo JPQL je pracnější – je nutno ošetřit parametry a po provedení dotazu uvést instanci entity do kontextu a obnovit ji z databáze. Výsledek nakonec je stejný.

⌂ Čtení seznamu *(em.list)* – L

– Metoda:

```
public List<Passenger> listPassengerByIdAndLimit(Integer limit) {
   // the following query is a 'SELECT *' in JPQL, use 'order by p.id' to retrieve fixed list, as it will otherwise change its order after update
   Query query = em.createQuery("Select p from Passenger p order by p.id asc");
   List<Passenger> passengerListLimit = query.setFirstResult(0).setMaxResults(limit).getResultList();
   System.out.println("< List starts from 1 to " + limit + ":");
   for (Passenger passenger : passengerListLimit) {
      System.out.println(passenger.toString());
   }
   System.out.println("> List ends.");
   return passengerListLimit;
}
```

– Volání:

```
PassengerRepository passengerRepository = new PassengerRepository();
passengerRepository.listPassengerByIdAndLimit(3);
```

– Výsledek:

```
[EL Info]: 2022-05-29 14:38:44.416--ServerSession(775386112)--EclipseLink, version: Eclipse Persistence Services - 2.7.10.v20211216-f
< List starts from 1 to 3:
Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@62923ee6, name='Damian', surname='McCarl', dateofbirth=1951-12-06, arrivalticke
Passenger{id=2, grouppaxid=jpa.modelEntities.Grouppax@372ea2bc, name='Damian', surname='Ferrolli', dateofbirth=1955-04-14, arrivaltic
Passenger{id=3, grouppaxid=jpa.modelEntities.Grouppax@5dcbb60, name='Harris', surname='Cherrison', dateofbirth=1934-11-12, arrivaltic
> List ends.
```

⌂ Mazání dat *(em.remove)* – D

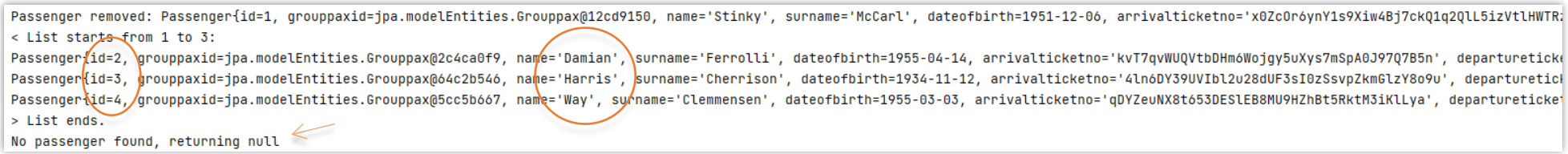
– Metoda:

```
// Remove passenger by native EntityManager 'remove' method. Note: it is easier to operate by id on complex entities, rather than address them directly
public Passenger removePassenger(Integer id) {
    try {
        Passenger passengerToDelete = em.find(Passenger.class, id);
        et.begin();
        em.remove(passengerToDelete);
        et.commit();
        em.clear();
        System.out.println("Passenger removed: " + passengerToDelete.toString());
        return passengerToDelete;
    } catch (Exception ex) {
        System.out.println("No passenger found, returning " + em.find(Passenger.class, id));
        et.rollback();
    }
    return null;
}
```

– Volání:

```
PassengerRepository passengerRepository = new PassengerRepository();
Passenger deletedPassenger = passengerRepository.removePassenger(1);
passengerRepository.listPassengerByIdAndLimit(3);
passengerRepository.removePassenger(0);
```

– Výsledek:



⌂ Přidání řádku do databáze *(em.add)* – C

– Metoda:

```
public void add(Passenger passenger) {
    et.begin();
    try {
        em.persist(passenger);
        et.commit();
        System.out.println("Passenger added: " + passenger.toString());
    } catch (Exception ex) {
        et.rollback();
    }
}
```

– Volání:

```
PassengerRepository passengerRepository = new PassengerRepository();
Passenger deletedPassenger = passengerRepository.removePassenger(1);
passengerRepository.listPassengerByIdAndLimit(3);

passengerRepository.add(deletedPassenger);
passengerRepository.listPassengerByIdAndLimit(3);
passengerRepository.close();
```

– Výsledek:

```
Passenger removed: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@1e0f9063, name='Damian', surname='McCarl', dateofbirth=1951-1
< List starts from 1 to 3:
Passenger{id=2, grouppaxid=jpa.modelEntities.Grouppax@363f6148, name='Damian', surname='Ferrolli', dateofbirth=1955-04-14, arrivaltick
Passenger{id=3, grouppaxid=jpa.modelEntities.Grouppax@16423501, name='Harris', surname='Cherrison', dateofbirth=1934-11-12, arrivaltic
Passenger{id=4, grouppaxid=jpa.modelEntities.Grouppax@459f7aa3, name='Way', surname='Clemmensen', dateofbirth=1955-03-03, arrivalticke
> List ends.
Passenger added: Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@1e0f9063, name='Damian', surname='McCarl', dateofbirth=1951-12-
< List starts from 1 to 3:
Passenger{id=1, grouppaxid=jpa.modelEntities.Grouppax@1e0f9063, name='Damian', surname='McCarl', dateofbirth=1951-12-06, arrivaltickett
Passenger{id=2, grouppaxid=jpa.modelEntities.Grouppax@363f6148, name='Damian', surname='Ferrolli', dateofbirth=1955-04-14, arrivaltick
Passenger{id=3, grouppaxid=jpa.modelEntities.Grouppax@16423501, name='Harris', surname='Cherrison', dateofbirth=1934-11-12, arrivaltic
> List ends.
```

⌂ Příklad jmenovaného dotazu JPQL s parametrem:

– Entita „Passenger“:

```
@Entity
@NamedQueries({
    @NamedQuery(name = "Passenger.listByExcludingCountrySortSurnameDesc",
        query = "SELECT p FROM Passenger p WHERE p.passportcountry.id <> :country ORDER BY p.passportcountry.id, p.surname DESC"),

    @NamedQuery(name = "Passenger.listByCountryAndExcludingPartialSurnameSortByNameDesc",
        query = "SELECT p FROM Passenger p WHERE p.passportcountry.id = :country AND p.surname NOT LIKE :surname " +
            "ORDER BY p.surname DESC"),

    @NamedQuery(name = "Passenger.countPaxPerMothByYearOfBirth",
        query = "SELECT COUNT(p) FROM Passenger p " +
            "WHERE func('DATE_PART', 'YEAR', p.dateofbirth) = :year " +
            "GROUP BY func('DATE_PART', 'MONTH', p.dateofbirth)" +
            "HAVING COUNT(p) >= :amount " +
            "ORDER BY avg(func('DATE_PART', 'MONTH', p.dateofbirth)) DESC")
})
@Table(name = "passenger", indexes = {
    @Index(name = "passenger_idx_year_of_birth", columnList = "date_part('year'::text, dateofbirth)")
})
public class Passenger {
...
```

Pozn.: *pole „passportcoutry“ odkazuje na entitní třídu „country“, a abychom dosáhli na název země (řetězec), který je současně id tabulky „country“, je potřeba na něj takto odkázat: „p.passportcountry.id“*

Ve třetím případě volaná funkce PostgreSQL „DATE_PART“ je náhradou funkce EXTRACT, která nefunguje ve formě EclipseLink „Extract(Year, fromDate)“, a pro Postgre její konstrukce nedovoluje volání z JPQL: „Extract(Year from Date)“

– Metody „PassengerRepository“:

```
public List<Passenger> listPassengerByExcludingCountrySortBySurnameDesc(String excludingCountry) {
    Query query = em.createNamedQuery("Passenger.listByExcludingCountrySortSurnameDesc");
    query.setParameter("country", excludingCountry);
    System.out.println(query.getResultList().toString());
    System.out.println("Total: " + query.getResultList().stream().count() + " pax.");
    return query.getResultList();
}

public List<Passenger> listPassengerByCountryAndExcludingPartialSurnameSortByNameDesc(String country, String excludingSurnamePart) {
    Query query = em.createNamedQuery("Passenger.listByCountryAndExcludingPartialSurnameSortByNameDesc");
    query.setParameter("country", country);
    query.setParameter("surname", '%' + excludingSurnamePart + '%');
    System.out.println(query.getResultList().toString());
    System.out.println("Total: " + query.getResultList().stream().count() + " pax.");
    return query.getResultList();
}

public List<Integer> countPassengerPaxPerMothByYearOfBirth(Integer yearSelected, Integer hurdleAmount) {
    Query query = em.createNamedQuery("Passenger.countPaxPerMothByYearOfBirth");
    query.setParameter("year", yearSelected);
    query.setParameter("amount", hurdleAmount);
    System.out.println(query.getResultList());
    System.out.println("Total: " + query.getResultList().stream().count() + " values.");
    return query.getResultList();
}
```

– Volání:

passengerRepository.listPassengerByExcludingCountrySortBySurnameDesc("China");

– Výsledek:

```
 grouppaxid=jpa.modelEntities.Grouppax@3a7b503d, name='Land', surname='Aaronson', dateof
departureticketno='NKu4PwhU0ZeEKM0NFTqDIcIYnZdfyHu5yAxcuA7H', passportcountry=jpa.modelE
insuranceno='hKuAEEEólXX3YVjX1yROcócFbsaLKWd2Elr4TVXJ', contacts={IndirectSet: not insta
Total: 10712 pax.
```

– Volání:

passengerRepository.listPassengerByCountryAndExcludingPartialSurnameSortByNameDesc("China", "A");

– Výsledek:

```
Passenger{id=19857, grouppaxid=jpa.modelEntities.Grouppax@36a7abe1, name='Ardis', surname='Baal',
departureticketno='KShihDGSQkOZjAl464AXi1WLsVgNiMD8HEfWu1vw', passportcountry=jpa.modelEntities.Co
insuranceno='K3tQIuFf5zF5D2E9spP2lQwBhS7bNFArHjRrlyQx', contacts={IndirectSet: not instantiated},
Total: 24671 pax.
```

– Volání:

passengerRepository.countPassengerPaxPerMothByYearOfBirthWithHurdleAmount(1951, 300);

– Výsledek:

```
[340, 306, 303, 300]
Total: 4 values.
```

Pozn.: bohužel, nevím, jak uvést čísla měsíců u tohoto výsledku

Něco ke čtení o dotazech v JPQL:

- https://www.objectdb.com/java/jpa/query/named
- https://www.codejava.net/java-ee/jpa/jpa-named-query-examples
- https://www.baeldung.com/jpa-query-parameters
- https://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/JPQL
- https://docs.oracle.com/middleware/1213/toplink/concepts/queries.htm#OTLCG94352
- https://www.freecodecamp.org/news/sql-group-by-clauses-explained/
- https://stackoverflow.com/questions/4362876/how-to-view-the-sql-queries-issued-by-jpa
- https://stackoverflow.com/questions/3441193/are-sql-injection-attacks-possible-in-jpa
- https://www.baeldung.com/sql-injection

– Vlastní nastavení konstruktéru v entitě:

```
...
public Employee(String id, String name, String surname) {
   this.id = id; // this is an employee's phone number
   this.name = name;
   this.surname = surname;
}
public Employee() { }
...
```

– Zavedení repositáře zaměstnanců:

```
public class EmployeeRepository {
   private EntityManager em = EntityManagerUtility.getEm();
   private EntityTransaction et = em.getTransaction();

public void close() {
   this.em.close();
}
...
```

– Zavedení nastavení testu repositáře zaměstnanců:

```
class EmployeeRepositoryTest {
   private static EmployeeRepository repository;

   @BeforeAll
   public static void beforeClass() throws Exception{
      repository = new EmployeeRepository();
   }

   @AfterAll
   public static void afterClass() {
      repository.close();
   }

   @BeforeEach
   public void setUp() { }

   @AfterEach
   public void tearDown() { }
...
```

– Metody CRUD repositáře zaměstnanců:

```
public void add(Employee employee) {
    et.begin();
    try {
        em.persist(employee);
        et.commit();
        System.out.println("Employee added: " + employee.toString());
    } catch (Exception ex) {
        et.rollback();
    }
}
```

```
public Employee find(String id) {
    Employee employee = em.find(Employee.class, id);
    if (employee != null) {
        System.out.println("Employee found: " + employee.toString());
    }
    return employee;
}
```

– Testy jednotlivých metod:

```
@Test
void add() {
    Employee employee = new Employee("777-777-55551", "Jack",
"Nickolson");
    employee.setAddress("Bishop house 15");
    employee.setEmail("Nico@gogo.com");
    employee.setType("Driver");

    repository.add(employee);

    assertNotNull(employee);
    assertNotNull(employee.getId());
    assertEquals("Jack", employee.getName());
}
```

```
@Test
void find() {
    Employee employee = new Employee("777-777-55552", "Jack",
"Nickolson");
    employee.setAddress("Bishop house 15");
    employee.setEmail("Nico1@gogo.com"); // email have to be unique
    employee.setType("Driver");

    repository.add(employee);
    employee = repository.find(employee.getId());

    assertNotNull(employee);
    assertNotNull(employee.getId());
    assertEquals("Jack", employee.getName());
}
```

```
public Employee update(Employee employee) {
    Employee employeeToUpdate = find(employee.getId());
    et.begin();
    employeeToUpdate.setName(employee.getName());
    employeeToUpdate.setSurname(employee.getSurname());
    et.commit();
    em.clear();
    System.out.println("Employee updated: " +
employeeToUpdate.toString());
    return employeeToUpdate;
}
```

```
public void remove(Employee employee) {
    try {
        et.begin();
        em.remove(employee);
        et.commit();
        em.clear();
        System.out.println("Employee removed: " + employee.toString());
    } catch (Exception ex) {
    }
}

// clear db:
// delete from employee where address like '%Bishop%';
```

```
@Test
void update() {
    Employee employee = new Employee("777-777-55553", "Jack",
"Nickolson");
    employee.setAddress("Bishop house 15");
    employee.setEmail("Nico3@gogo.com");
    employee.setType("Driver");

    repository.add(employee);
    employee.setName("John");
    employee = repository.update(employee);

    assertNotNull(employee);
    assertNotNull(employee.getId());
    assertEquals("John", employee.getName());
    assertEquals("Nickolson", employee.getSurname());
}
```

```
@Test
void remove() {
    Employee employee = new Employee("777-777-55554", "Jack",
"Nickolson");
    employee.setAddress("Bishop house 15");
    employee.setEmail("Nico4@gogo.com");
    employee.setType("Driver");

    repository.add(employee);
    repository.remove(employee);
    employee = repository.find(employee.getId());

    assertNull(employee);
}
```

Zdroje pro definici přístupu k datům:

- EclipseLink: https://www.eclipse.org/eclipselink/
- Co je EclipseLink: https://www.eclipse.org/eclipselink/documentation/2.4/concepts/general001.htm#CHDIJJGA
- Rozdíl mezi poskytovateli implementaci JPA, EclipseLink a Hibernate:
  - https://thorben-janssen.com/difference-jpa-hibernate-eclipselink/
  - https://www.baeldung.com/jpa-hibernate-difference
- Příklad implementace JPA: https://riptutorial.com/jpa
- Java Persistence/Persisting: https://en.wikibooks.org/wiki/Java_Persistence/Persisting
- JPA EntityManager – kdy použit metodu getTransaction(): https://stackoverflow.com/a/8464432
- JPA EntityManager – proč použit metodu persist() místo merge(): https://stackoverflow.com/a/1070629
- JPQL – jak definovat dotazy v JPA: https://thorben-janssen.com/jpql/
- Výhody použití jmenovaných dotazů v JPQL: https://stackoverflow.com/a/29609618
- Interface EntityManager: https://www.eclipse.org/eclipselink/api/2.6/javax/persistence/EntityManager.html
- JavaTM Persistence 2.1 Final Release for Evaluation: https://download.oracle.com/otndocs/jcp/persistence-2_1-fr-eval-spec/index.html
- Generické DAO:
  - Diskuse: https://stackoverflow.com/a/14541894
  - Příklad s použitím Spring: https://www.codeproject.com/Articles/251166/The-Generic-DAO-Pattern-in-Java-with-Spring-and-JP
- Používání reflexe v Javě (Java Reflection):
  - https://www.oracle.com/technical-resources/articles/java/javareflection.html
  - Průvodce reflexi v Javě: https://www.baeldung.com/java-reflection

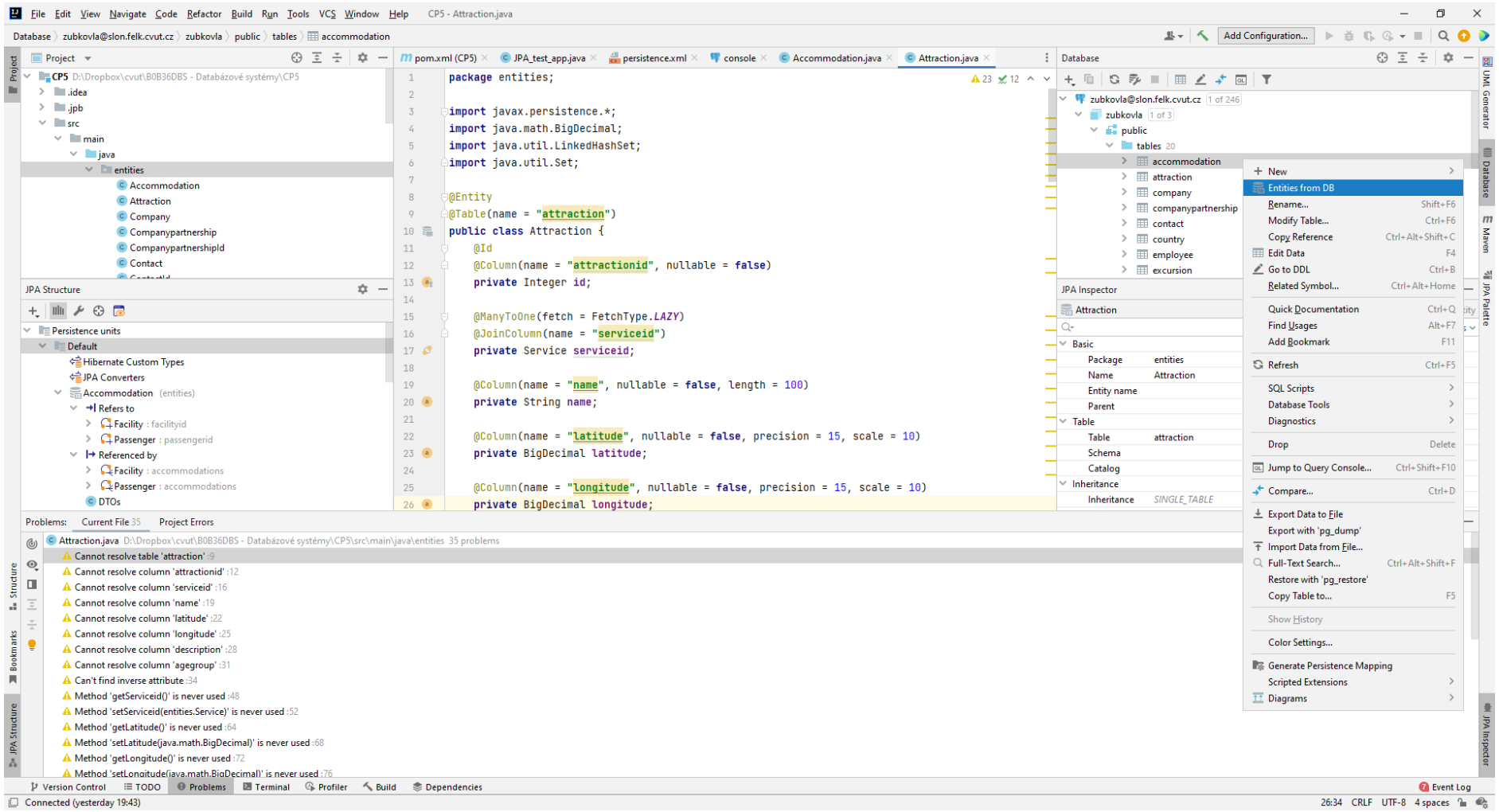
Detail servisního modelu, užití transakcí v JPA:
[NOT IMPEMENTED]

Zdroje pro nastavení EclipseLink JPA, transakce:

- Introduction to EclipseLink transactions (ELUG): https://wiki.eclipse.org/Introduction_to_EclipseLink_Transactions_(ELUG)
- Pessimistic locking in JPA: https://www.baeldung.com/jpa-pessimistic-locking
- Optimistic locking in JPA: https://www.baeldung.com/jpa-optimistic-locking
- Set default level of isolation in EclipseLink: https://stackoverflow.com/a/32685637
- Change default isolation level in EclipseLink by another instance of EntityManagerFactory: https://java.tutorialink.com/cleaning-up-after-changing-the-isolation-level-in-jpa-eclipslink-entitymanager/
- EclipseLink performance features: https://www.eclipse.org/eclipselink/documentation/2.7/solutions/performance001.htm
- Performance tuning in JPA: https://docs.oracle.com/middleware/1212/core/ASPER/toplink.htm#ASPER99173
- Transakce řízené aplikaci (Java SE) a kontejnerem (Java EE):
  - https://stackoverflow.com/questions/8464370/jpa-when-to-use-gettransaction-when-persisting-objects
  - https://stackoverflow.com/questions/13489057/differences-between-container-managed-and-application-managed-entitymanager
  - https://docs.oracle.com/cd/E19798-01/821-1841/bnbqz/index.html

Příloha

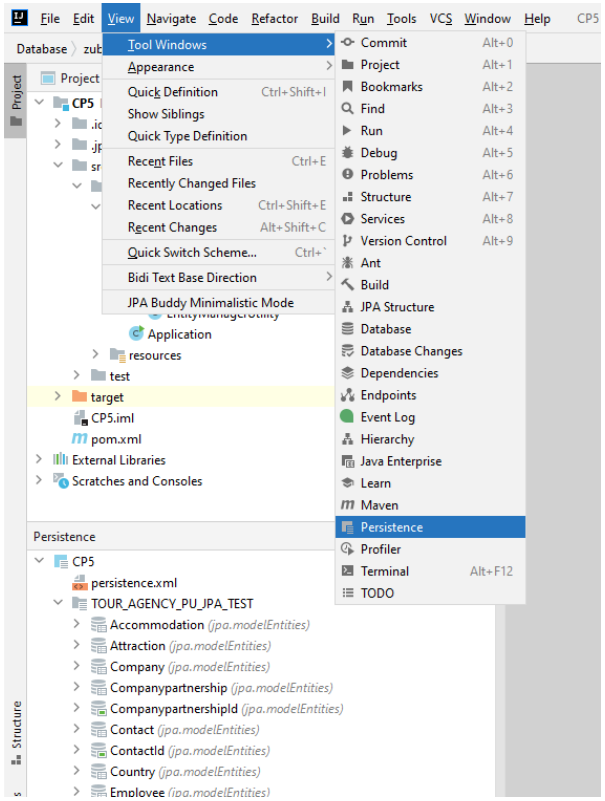
Tvorba tříd na základě databáze v IntelliJ IDEA pomocí pluginu JPA Buddy (předem je nutné se přihlásit do databáze):



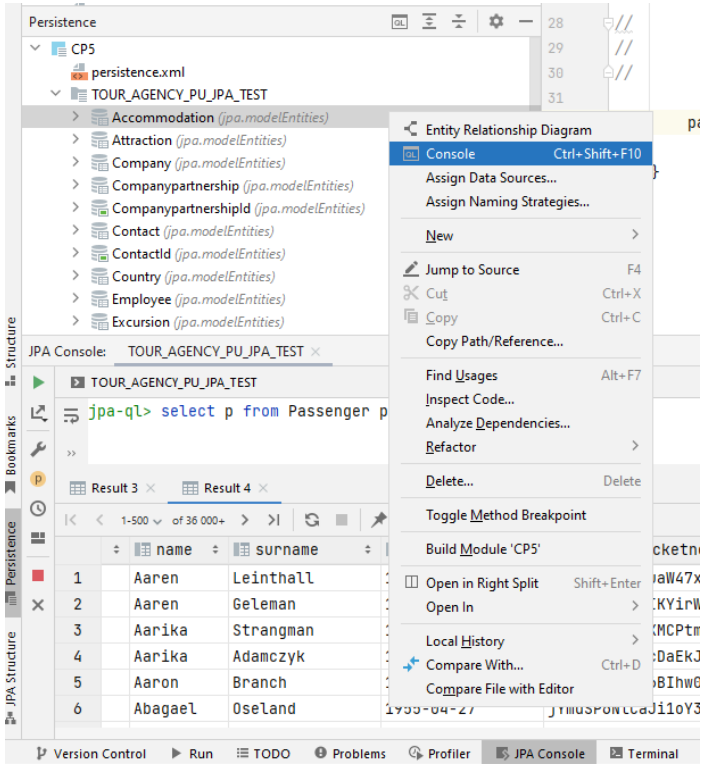
Jiný způsob je tvorba a anotace POJO tříd. Ovšem třídy generované JPA Buddy jsou správně anotované a mohou jen potřebovat změnit definici vztahů dle gusta vývojáře.

Tvorba entitních tříd v IntelliJ IDEA pomocí záložky „Persistence“ nebo „JPA Structure“ pluginu JPA Buddy:

Zobrazení záložky „Persistence“ v IntelliJ IDEA:



Otevření konzole dotazů JPQL v IntelliJ IDEA a spuštění dotazu (předem entity musí být naplněné z databáze spuštěním aplikace):



Entity lze tvořit jak v záložce „Persistence“, tak i v záložce „JPA Structure“ pluginu JPA Buddy s pomocí záložek „JPA Palette“ a „JPA Inspector“.

- Persistence tool window: https://www.jetbrains.com/help/idea/persistence-tool-window.html
- Entity Designer: https://www.jpa-buddy.com/documentation/entity-designer/

Ukázka „persistence.xml“ pro poskytovatele JPA EclipseLink

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd"
        version="2.2">
   <persistence-unit name="TOUR_AGENCY_PU_JPA_TEST">
      <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <!-- or use names of the entities instead-->
      <!-- <class>% name of entity 1%</class> -->
      <!-- <class>% name of entity 2% etc.</class> -->
      <!-- note, sometimes <exclude-unlisted-classes> does not work -->
      <properties>
         <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
         <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://%database host%:5432/%database%"/>
         <property name="javax.persistence.jdbc.user" value="%username%"/>
         <property name="javax.persistence.jdbc.password" value="%password%"/>
         <!-- create database schema from entities-->
         <property name="javax.persistence.schema-generation.database.action" value="create"/>
      </properties>
   </persistence-unit>
</persistence>

      <!-- source: -->
      <!-- https://pastebin.com/hLEMBXHp -->
```

Ukázka Maven-závislostí v pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>Lab</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>42.3.4</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.eclipse.persistence/org.eclipse.persistence.jpa -->
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>org.eclipse.persistence.jpa</artifactId>
            <version>2.7.4</version>
        </dependency>
    </dependencies>
</project>
```