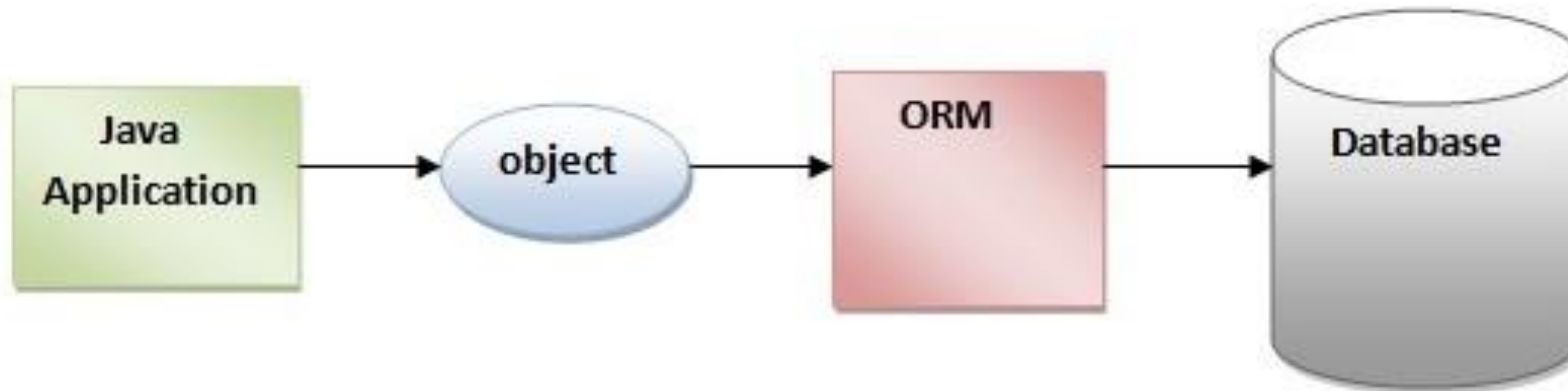
Spring Data JPA

Projekt 1

ORM – Objektovo-relačné mapovanie

- Prepojenie objektov v prog. jazyku a SQL databázy
- Entitná tabuľka v SQL = Entitná trieda v Java
- Relačná tabuľka v SQL ~ inštančné premenné typu List/Set/... v entitných triedach



Problémy s ORM

- **Problém cyklických referencií**
  - Užívatelia majú čipy. Čítačka čipov potrebuje vedieť akého užívateľa môže pustiť.
  - V SQL -> relačná tabuľka users_chipreaders
  - V Jave -> trieda User má field List<ChipReader> a/alebo trieda ChipReader má field List<User>
    - Zacyklenie

Problémy s ORM

- **N+1 query problém**
  - Entita má relácie s N ďalšími entitami.
  - SELECT-nem entity a jej relácie na-JOIN-ujem v jednom über dopyte?
    - SELECT mi vráti milión riadkov ani neviem ako
    - Potrebujem stránkovanie (OFFSET + LIMIT vo vnorenom dopyte)
  - Alebo SELECT-nem iba entity a relácie budem SELECT-ovať samostatne pre každý riadok?
    - Každý dopyt je IO operácia. IO je drahé.
- Na Paz1c sme ORM robili ručne v DAO triedach
  - Trieda ResultSet predstavovala výsledok príkazu SELECT
  - Každý riadok z ResultSet sme previedli na náš objekt (User, ChipReader)

JPA – Jakarta Persistence

- Pred 2019 známe ako Java Persistence API
- Vysokoúrovňová práca s SQL v Jave
- Nadstavba nad JDBC
- Má vlastný dopytovací jazyk JPQL
    - SQL, ale nad Java triedami namiesto tabuľkami
- Tiež môžeme SQL dopyty robiť programovo a objektovo
    - Criteria Query API
- Generovanie SQL tabuliek, primárnych a cudzích kľúčov z Java entitných tried
- ...

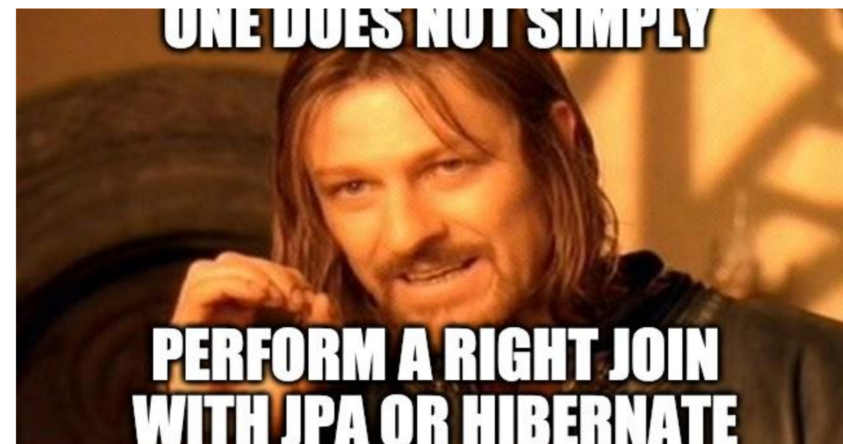
JPQL – Java Persistence Query Language

- JPQL dopyt ...

```
SELECT u FROM User u WHERE u.chipReaders IS NOT EMPTY
```

  - User je Java trieda, nie tabuľka
- ... sa preloží na SQL dopyt

```
SELECT u.* FROM users u JOIN user_chip_readers ucr ON u.id = ucr.user_id
```



JPA je rozhranie

- JPA je špecifikácia a banda rozhraní
- Potrebuje implementáciu aby fungovalo
- Má reálne viac implementácií

- Hibernate ORM – asi najznámejšia implementácia JPA
  - Hibernate je paradoxne starší ako JPA
  - Jeho vlastné API vyzerá úplne inak ako JPA
  - Asi najpopulárnejšie Java ORM pred JPA (a nepriamo aj po JPA)

- Rôzne frameworky používajú rôzne implementácie JPA
  - Ale to vás nemusí v súčasnosti veľmi zaujímať
  - V nových projektoch vládne JPA alebo JOOQ

Spring Data JPA

- Použitie **Repository** vzoru namiesto DAO
- Vytvoríme si interface UserRepository extends JpaRepository<User, Long>
- Automaticky dostaneme, **findById**, **findAll**, aj so stránkovaním a ORDER BY
- Spring automaticky implementuje hlavičky našich metód
  - Podľa názvu/návratového typu/vstupných parametrov metódy
  - Pr.: `List<User> findByName(String name);`
    - `SELECT * FROM users WHERE name = ?;`
  - Pr.: `User findByUsernameOrEmail(String username, String email);`
    - `SELECT * FROM users WHERE username = ? OR email = ? LIMIT 1;`
  - https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html
- Zložitejšie dopyty cez anotáciu @Query
  - implicitne JPQL, alebo voliteľne SQL
  - Alebo implementujeme ručne cez Criteria Query API, alebo cez surové JDBC

Poďme si vytvoriť JPA demo

- V IntelliJ IDEA Ultimate, alebo cez https://start.spring.io/ si vytvorme nový *Spring Boot Project*
- ->File/New/Project/Spring Initializr
  - Language: Java
  - Type: Maven
  - JDK: cesta k JDK 21, alebo nechajte IDE stiahnuť
  - Java: 21
  - Packaging: Jar
- Klikneme na *Next*

Komponenty

- V okne *Dependencies* si vyklikáme
    - Lombok
    - Spring Web
    - Spring Data JPA
    - MySQL Driver

- Klikneme dole na *Create*

Spustíme si MySQL v Dockeri

docker-compose.yaml

```yaml
services:
  mysql:
    image: 'mysql:latest‘
    environment:
      - 'MYSQL_DATABASE=entrance'
      - 'MYSQL_PASSWORD=secret'
      - 'MYSQL_ROOT_PASSWORD=verysecret'
      - 'MYSQL_USER=myuser'
    ports:
      - '3306:3306‘
```

Spustíme príkazom docker compose up

- docker compose stop – zastaví kontajnery
- docker compose down – zastaví a zmaže kontajnery

Nakonfigurujeme Spring

src/main/resources/application.properties

```
# url databázy, tiež nastaviteľná cez premennú prostredia
spring.datasource.url=${DB_JDBC:jdbc:mysql://localhost:3306/entrance}

# konto v databáze, tiež nastaviteľné cez premennú prostredia
spring.datasource.username=${DB_JDBC:myuser}

# heslo k databáze, tiež nastaviteľné cez premennú prostredia
spring.datasource.password=${DB_JDBC:secret}

# z entitných tried generuj tabuľky v databáze
spring.jpa.generate-ddl=true

# zapni JPA repozitáre (namiesto DAO)
spring.data.jpa.repositories.enabled=true
```

Modelujme entity

```
@Data
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(unique=true)
    private String email;

    private boolean active;

    private int sex;

    @ManyToMany
    private List<ChipReader> chipReaders;
}
```

- @Data nám vygeneruje gettre a settre
- @Entity táto trieda bude tabuľka v SQL
- @Id stĺpec s primárnym kľúčom
- @GeneratedValue(...) bude automaticky generovať hodnoty ID. Na vstup vieme určiť spôsob generovanie. *IDENTITY* znamená použite AUTO_INCREMENT v MySQL
- Inštančné premenné bez anotácie budú normálne stĺpce
- @Column(unique=true) nám zabezpečí unikátnosť
- Pohlavie ľudí v DB sa môže riadiť štandardom https://en.wikipedia.org/wiki/ISO/IEC_5218
- @ManyToMany predstavuje vzťah M:N s entitou ChipReader. Pomocná relačná tabuľka v SQL sa vygeneruje automaticky na pozadí a v Jave ju neriešime (ale môžeme ak chceme)

Modelujme entity

```
@Data
@Entity
public class ChipReader {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(unique=true)
   private String position;

   @ManyToMany(mappedBy = “cardReaders”)
   @JsonIgnore
   private List<User> users;
}
```

- @ManyToMany predstavuje vzťah M:N s entitou User.

  Vstupný parameter určuje, že inštančná premenná User.chipReaders je „vlastníkom“ vzťahu.

  V praxi je premenná ChipReader.users "iba na čítanie". Modifikácia listu a následné uloženie entity **nebude fungovať**.

- @JsonIgnore nám nebude serializovať tento stĺpec aby sme sa vyhli nekonečnej rekurzii
  - User má ChipReader, ChipReader má User, ...
  - V praxi sa to rieši ináč
    - Napr. na tejto strane relácie nebudeme mať inštančnú premennú users
    - Napr. použijeme tzv. DTO objekty
      - UserDTO, ChipReaderDTO, ...

~~DAO~~ Repozitáre

```
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByEmail(String email);

    List<User> findByActiveTrue();
    Page<User> findByActiveTrue(Pageable pageable);

    @Query("SELECT u FROM User u WHERE u.chipReaders IS NOT EMPTY")
//    @Query(nativeQuery = true, value = "SELECT u.* FROM user u JOIN user_chip_readers ucr ON u.id = ucr.users_id")
    List<User> findWithChipReaderAccess();

    List<User> findByChipReaders(ChipReader chipReader);

}
```

Ďalšie možnosti vzťahov

- Ešte máme relačné anotácie
  - @OneToMany, @ManyToOne a @OneToOne
- V inštančnej premennej, ktorá je vlastníkom vzťahu (t.j. nemá v relačnej anotácií atribút mappedBy) môžeme
  - definovať kaskádové operácie
  - definovať načítavanie vzťahov (N+1 problém)
    - okamžité načítavanie – FetchType.EAGER
    - načítavanie podľa potreby – FetchType.LAZY (defaultne)

Lenivé načítavanie vzťahov

Príklad: Robíme JPQL SELECT na entitu (User),

- FetchType.LAZY nechá List<CardReader> „prázdny“
  - Čítanie z „prázdneho“ zoznamu ho bude „magicky“ napĺňať
  - **Pozor**, „magické“ (*lenivé*) napĺňanie funguje iba kým je otvorené spojenie k DB
  - Ak používate SpringBoot + Spring Web + JPA, tak spojenie je aktívne počas celej HTTP požiadavky a lenivé napĺňanie funguje bez problémov

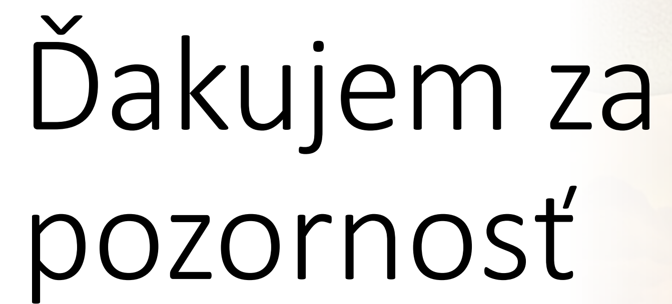
Časté chyby pri lenivom načítavaní

- Mnohé Spring ~~Boot~~ projekty majú nastavenú inú politiku spojenia k DB
  - Spojenie začína a končí v Repository
  - Po vrátení entity z Repository, dostanete pri čítaní z „prázdneho“ zoznamu LazyInitializationException
  - Ak vám nejaká metóda hádže túto výnimku:
    - V dopyte použiť JOIN FETCH
      - Pozor na stránkovanie
    - Anotovať metódu s @Transactional
      - To vám (okrem iného) bude udržiavať spojenie s DB počas volania metódy

S veľkou mocou...

JDBC (pre absolventov PAZ1c)

JPA



Ďakujem za pozornosť

Nech vás sprevádza Sila