ORM v Jave

Object-Relational Mapping

Teória pre stredné školy

Práca s databázou pomocou objektov v Jave

Čo je ORM?

ORM = Object-Relational Mapping

- ORM je technika, ktorá umožňuje pracovať s databázou pomocou objektov v Jave namiesto priameho písania SQL príkazov.
- Automaticky premieňa riadky z databázových tabuliek na objekty a naopak.
- Programátor pracuje s triedami (Kniha, Študent, Objednávka) namiesto ResultSet a SQL reťazcov.

Prečo potrebujeme ORM?

Problémy s čistým JDBC pri väčších projektoch:

- Veľa opakovaného a nudného kódu (PreparedStatement, ResultSet, manuálne mapovanie)
- Objekty v Jave a tabuľky v databáze sú dva rôzne svety
- Ručné mapovanie stĺpcov na polia objektov je zdĺhavé a chybové
- Ťažko sa udržiava kód pri zmene štruktúry databázy

Čo presne robí ORM?

1. **Mapovanie**
   Automaticky premieňa triedy Javy na tabuľky v databáze

2. **CRUD operácie**
   Jednoduché ukladanie, načítanie, aktualizáciu a mazanie objektov

3. **Dotazy**
   Umožňuje písať dotazy v jazyku bližšom Jave (HQL / JPQL)

Výhody ORM

- Výrazne menej kódu (často 5–10× menej ako pri JDBC)
- Práca s objektmi – prirodzenejšie pre programátora
- Automatické mapovanie medzi objektmi a tabuľkami
- Jednoduchá podpora transakcií
- Lepšia čitateľnosť a údržba kódu
- Relatívne jednoduchá zmena databázy (napr. z SQLite na PostgreSQL)

Hibernate – najpoužívanejší ORM

- Najpopulárnejší a najpoužívanejší ORM framework pre Javu
- Existuje už viac ako 20 rokov a stále sa aktívne vyvíja
- Podporuje moderné anotácie (odporúčaný spôsob)
- Funguje ako implementácia štandardu Jakarta Persistence (JPA)
- Výborná podpora pre SQLite, MySQL, PostgreSQL, Oracle a ďalšie

Základné pojmy

Entita
Trieda, ktorá reprezentuje jeden riadok v databázovej tabuľke (napr. Kniha)

Session
"Pripojenie" k databáze – hlavný objekt na prácu s entitami

SessionFactory
Továreň na Session – vytvára sa raz na začiatku aplikácie

Transaction
Transakcia – zabezpečuje konzistenciu dát (commit / rollback)

Entita – základná stavebná jednotka

```
@Entity
@Table(name = "knihy")
public class Kniha {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nazov;

    private String autor;
    private Integer rokVydania;

    // gettery, settery, konštruktory...
}
```

Session a SessionFactory

SessionFactory
- Vytvára sa raz na začiatku aplikácie (ťažký objekt)
- Slúži ako továreň na Session

Session
- Vytvára sa často – pre každú logickú operáciu
- Ľahký objekt, zodpovedný za prácu s entitami
- Používa sa na persist, find, remove, query...

Transakcie

```
Transaction tx = session.beginTransaction();

session.persist(kniha1);
session.persist(kniha2);

tx.commit();      // uloží všetky zmeny naraz
// tx.rollback(); // pri chybe zruší všetky zmeny
```

Transakcia zabezpečuje, že databáza ostane vždy v konzistentnom stave.

Základné operácie (CRUD)

```
// Create (uloženie)
session.persist(kniha);

// Read (načítanie)
Kniha k = session.find(Kniha.class, 1L);

// Update (aktualizácia)
k.setNazov("Nový názov knihy");

// Delete (vymazanie)
session.remove(k);
```

Dotazy – HQL / JPQL

```
// Načítanie všetkých kníh
List<Kniha> knihy = session
    .createQuery("FROM Kniha", Kniha.class)
    .list();

// Filtrovanie podľa autora
List<Kniha> knihy = session
    .createQuery("FROM Kniha WHERE autor = :autor", Kniha.class)
    .setParameter("autor", "George Orwell")
    .list();
```

JDBC vs ORM – kedy čo použiť?

Použi JDBC, keď:
• Potrebuješ absolútne maximálny výkon
• Projekt je veľmi malý a jednoduchý

Použi ORM (Hibernate), keď:
• Pracuješ s väčším počtom entít a väčšími projektmi
• Chceš pracovať objektovo a čisto
• Chceš výrazne ušetriť čas pri vývoji a údržbe

Zhrnutie

- ORM umožňuje pracovať s databázou cez objekty
- Hibernate je najpoužívanejší ORM framework v Jave
- Základné prvky: Entita, Session, Transaction
- Pracujeme hlavne s anotáciami (@Entity, @Id, @Column)
- Namiesto SQL používame HQL/JPQL
- Väčšina reálnych projektov dnes používa ORM

Otázky na zamyslenie

1. Čo je hlavný rozdiel medzi JDBC a ORM?

2. Na čo slúži anotácia @Entity?

3. Prečo je dôležité používať Transaction?

4. Kedy by si radšej použil čisté JDBC a kedy ORM?

5. Čo sa stane, keď zavoláš session.persist(obj)?