

Viac entít

UINF/PAZ_{1c}
5. prednáška



Zmena dát cez JdbcTemplate

```
String sql = "INSERT INTO user (chip_id, name, active)  
VALUES (?, ?, ?)";
```

```
jdbcTemplate.update(sql, user.getChipId(), user.getName(),  
user.getActive());
```

- Konverziu necháme na Spring
 - Escapovanie stringov, null sa nevloží ako "null", ...
 - Žiadne SQL injection

Zmena dát cez JDBCTemplate

```
String sql = "INSERT INTO user (chip_id, name, active)  
VALUES (?, ?, ?)";
```

```
jdbcTemplate.update(sql, user.getChipId(), user.getName(),  
user.getActive());
```

- Naplnenie id necháme na AUTOINCREMENT
 - Žiadne "SELECT max(id) FROM user" a potom "INSERT ... id+1..."

Ak chceme vedieť aké ID bolo pridelené riadku

```
var keyHolder = new GeneratedKeyHolder();
jdbc.update(connection -> {
    var ps = connection.prepareStatement(query,
        Statement.RETURN_GENERATED_KEYS);
    ps.setString(1, user.getChipId());
    ps.setString(2, user.getName());
    ps.setBoolean(3, user.isActive());
    return ps;
}, keyHolder);

var id = keyHolder.getKey().longValue();
user.setId(id);
```

Ak chceme vedieť aké ID bolo pridelené riadku

```
SimpleJdbcInsert simpleJdbcInsert = new  
    SimpleJdbcInsert(jdbcTemplate);  
simpleJdbcInsert.withTableName("user");
```

```
dataSource.setUrl("jdbc:mysql://localhost/users_management?  
serverTimezone=Europe/Bratislava&nullNamePatternMatchesAll=true");
```

... lebo stackoverflow.com
Ak nájdete niečo krajšie, dajte vedieť.

```
values.put("name", user.getName());  
values.put("active", user.isActive());
```

```
long id = simpleJdbcInsert.executeAndReturnKey(values)  
    .longValue();  
user.setId(id);
```

Vzťahy tried a entít

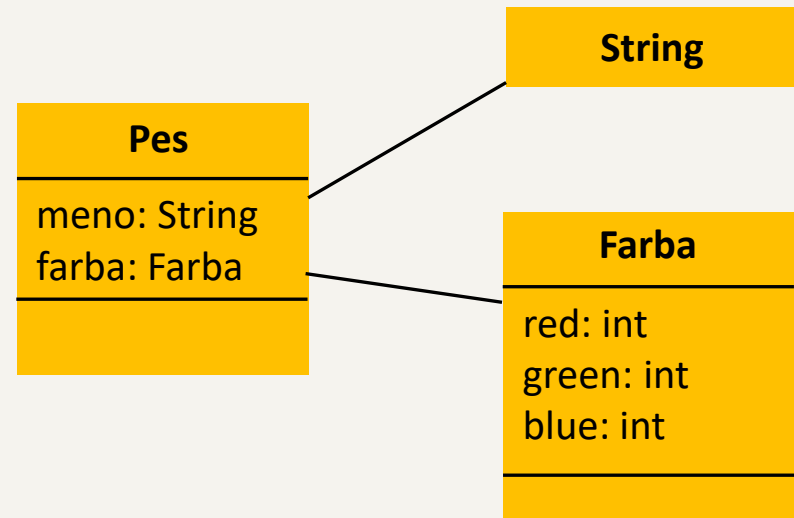


- Veľa druhov vzťahov
 - má vlastnosť, potrebuje, dedí (rozširuje), obsahuje / je časťou, súvisí, ...
- Veľa spôsobov modelovania (kreslenia)
 - UML, ontológie, ...
- Reprezentácia v rôznych jazykoch
 - OOP, SQL, OWL, ...

Vzťah „má vlastnosť“ / „má stav“

Pes MÁ meno (typ String)
Pes MÁ farbu (typ Color)

```
class Farba {  
    int red  
    int green  
    int blue  
}
```



```
class Pes {  
    String meno  
    Color farba  
}
```

```
class String {  
    ...  
}
```

Vzťah „závisí na“ (dependency)



- typické pre moduly – nie pre entity
- trieda **potrebuje** pre svoje **fungovanie** inú triedu
- **typicky**: metóda objektu nebude fungovať, ak nemá k dispozícii príslušný objekt
- **príklad**: kontrolér pre ukladanie objektu v používateľskom rozhraní potrebuje objekt DAO pre prístup k databáze



Vzťah „závisí na“ (dependency)

- realizácia = mám referenciu na závislosť
 - typicky v inštančnej premennej
- získanie referencie
 - aktívne
 - objekt si vytvorí závislosť - vytvorí potrebný objekt
 - opýta si závislosť od továrne
 - pasívne (dependency injection)
 - dostane závislosť cez konštruktor
 - dostane závislosť cez setter



Factory

- trieda závisí na inštancii továrne
- po vytvorení inštancie sa trieda hneď dá používať
- pre používateľa je trieda samostatnejšia
 - vytvoríme inštanciu a používame

Dependency injection

- trieda nezávisí na ničom
- po vytvorení inštancie môže byť trieda v „medzistave“
 - existuje, ale nemá závislosti
- pre vývojára je trieda flexibilnejšia
 - môžeme nastaviť závislosti zvonku



Oba prístupy sa dajú kombinovať

Vzťah „je časťou“ (agregácia)

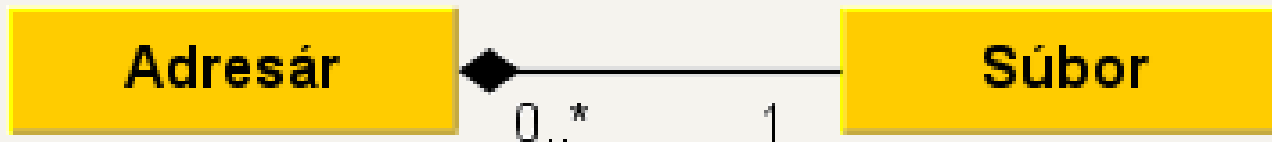
- v UML: agregácia a.k.a. zdieľaná kompozícia
- objekt je **súčasťou** iného objektu
 - dokáže však fungovať samostatne a nezávisle
 - môže byť súčasťou viacerých objektov naraz

```
public class Projekt {  
    private List<Výskumník> riešitelia = new ArrayList<>();  
    ...  
    public void pridajRiešiteľa(Výskumník výskumník) {  
    }  
}
```



Kompozícia

- **kompozícia** a.k.a. silná agregácia
- objekty majú spriahnuté životné cykly
- ak sa vymaže adresár, majú sa vymazať aj podsúbory
- súbor **musí** byť v **nejakom** adresári



Kompozícia 1:1

```
public class Minister {  
    ...  
}
```

```
public class Ministerstvo {  
    private Minister minister;  
  
    public void zrušit() {  
        ...  
        minister.odvolaj();  
    }  
}
```

Ak sa rozpustí ministerstvo, minister sa zruší tiež.

Kompozícia 1:M

```
public class Zamestnanec  
{  
    ...  
}
```

```
public class Firma {  
    private List<Zamestnanec>  
        zamestnanci;  
  
    public void zkrachuj() {  
        ...  
        for(var z: zamestnanci)  
            z.prepustiť();  
    }  
}
```

Ak firma zkrachuje, prepustí zamestnancov.

Kompozícia 1:M

```
public class Uzol {  
    private List<Uzol> uzly = new ArrayList<Uzol>();  
  
    public void zrušit() {  
        for(Uzol dieťa : uzly) {  
            dieťa.zrušit();  
        }  
    }  
}
```

Ak sa zruší uzol v strome, zrušia sa aj jeho potomkovia.

Agregácia alebo kompozícia?

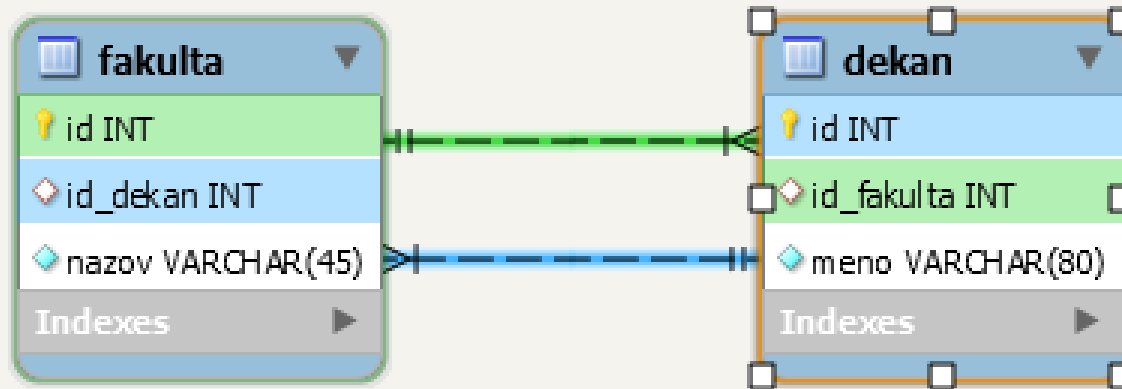
- vec aplikačnej domény
- rozdiel iba v integritných obmedzeniach
 - v databáze môžeme kompozíciu čiastočne ustrážiť cez NOT NULL cudzie kľúče
 - výrobok musí mať odkaz na svojho výrobcu
 - nevieme: výrobca musí mať aspoň 1 výrobok
 - v kóde si všetko musíme ustrážiť sami v metódach
- oba typy vzťahov reprezentované rovnako
 - zdroják: cez inštančné premenné
 - databáza: cez cudzie kľúče
- spoločný názov: **asociácie** (vzťahy)

V kóde: asociácia 1:1

```
public class Dekan {  
    private Fakulta fakulta;  
    /*...*/  
}
```

```
public class Fakulta {  
    private Dekan dekan;  
    /*...*/  
}
```

V databáze: asociácia 1:1

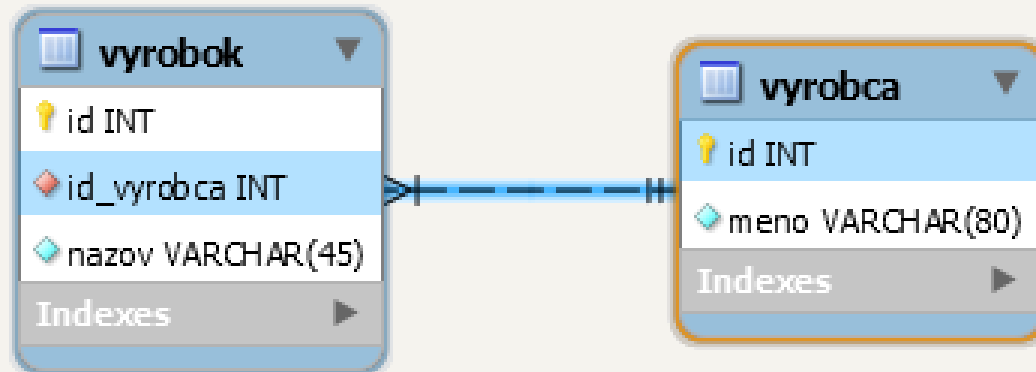


```
CREATE TABLE dekan (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  id_fakulta INT,  
  `meno` VARCHAR(80) NOT NULL,  
  PRIMARY KEY (`id`),  
  CONSTRAINT `fk_dekan_fakulta` FOREIGN KEY (id_fakulta)  
    REFERENCES fakulta (id)  
)
```

Asociácia 1:M

```
public class Výrobok {  
    private Výrobca výrobca;  
    /*...*/  
}
```

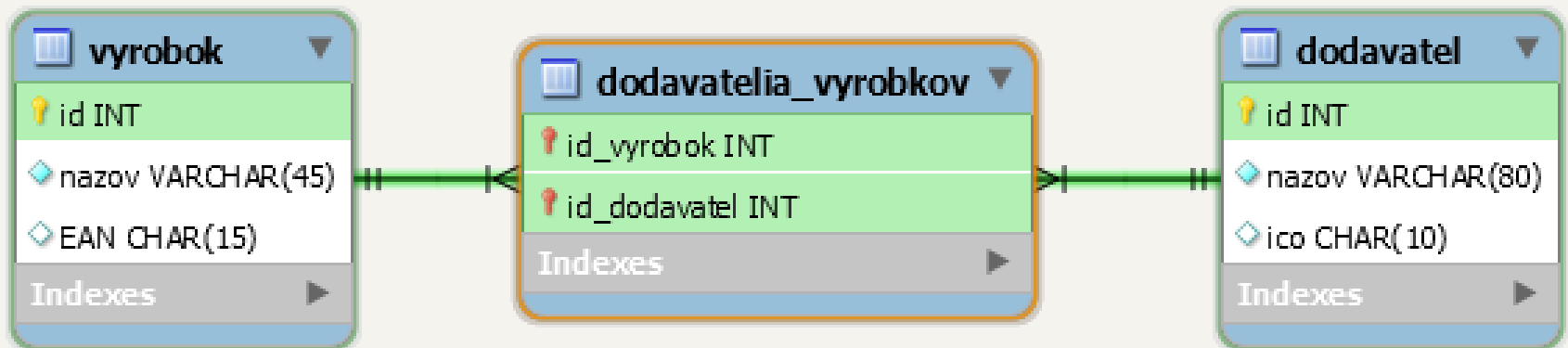
```
public class Výrobca {  
    private List<Výrobok> výrobky;  
    /*...*/  
}
```



Asociácia M:N

```
public class Výrobok {  
    private List< Dodávateľ > dodávatelia;  
    /* ... */  
}
```

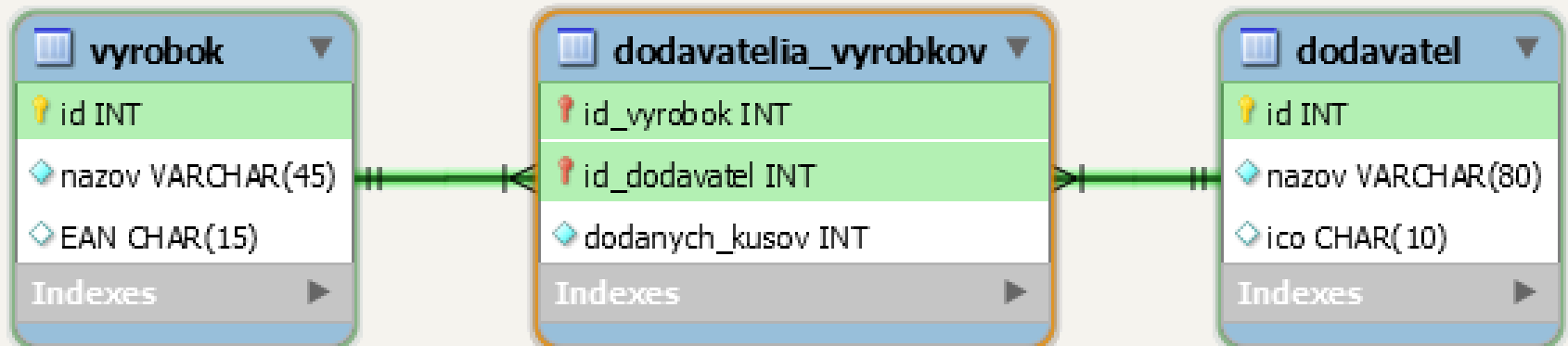
```
public class Dodávateľ {  
    private List< Výrobok > výrobky;  
    /* ... */  
}
```



Asociácia M:N s dodatočnými údajmi

```
public class Výrobok {  
    private List< Dodávateľ > dodávatelia;  
    private Map< Dodávateľ, Integer > dodávateliaPočty;  
    /* ... */  
}
```

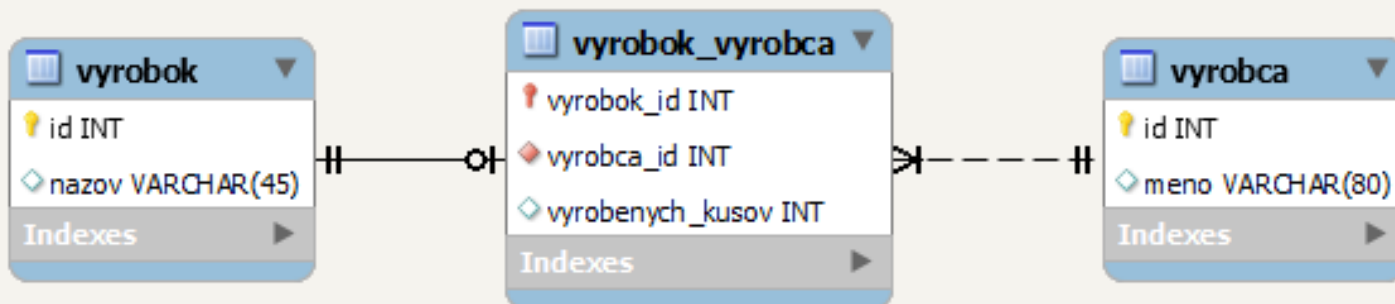
```
public class Dodávateľ {  
    private List< Výrobok > výrobky;  
    private Map< Výrobok, Integer > výrobkyPočty;  
    /* ... */  
}
```



Asociácia 1:M s dodatočnými údajmi

```
public class Výrobok {  
    private Výrobca výrobca;  
    private int vyrobenýchKusov;  
    /*...*/  
}
```

```
public class Výrobca {  
    private List<Výrobok> výrobky;  
    /*...*/  
}
```



Prístupy v celej budove (niekde áno, niekde nie)

- Administrátor

- pridáva, odoberá, (de)aktivuje používateľov

- mení používateľom karty

- pridáva, odoberá miesta s čítačkami - dvere

- prideluje používateľom práva na vstup do konkrétnych dverí

} CRUD

} CRUD

Používatelia vs. dvere

- Môžu jedny dvere pustiť viac ako jedného používateľa
 - Zjavne áno
- Môže jeden používateľ vojsť do viacerých dverí?
 - Ak nie
 - Máme vzťah 1:N
 - Používateľovi napíšeme ďalší parameter: dvere, do ktorých môže vojsť
 - Ak áno
 - Máme vzťah M:N
 - Reálnejšie (zvolíme radšej aj vtedy, keď zákazník tvrdí prvé)



Modelujme

CardReader je entita

- zaslúži si vlastné CardReaderDao
- vlastnú obrazovku na realizáciu CRUD operácií
- vlastnú továreň?
 - UserDaoFactory, CardReaderDaoFactory,...
 - koľko DAO toľko tovární? Čo keď ich bude 20?
 - veľa tovární = veľa miest, kde sa konfiguruje
 - Ak chcem prejsť v celom projekte na inú prezistentnú vrstvu, musím meniť všetky továrne

CardReader je entita

- zaslúži si vlastné CardReaderDao
- vlastnú obrazovku na realizáciu CRUD operácií
- vlastnú továreň?
 - UserDaoFactory, CardReaderDaoFactory,...
 - koľko DAO toľko tovární? Čo keď ich bude 20?
 - veľa tovární = veľa miest, kde sa konfiguruje
 - Ak chcem prejsť v celom projekte na inú prezistentnú vrstvu, musím meniť všetky továrne

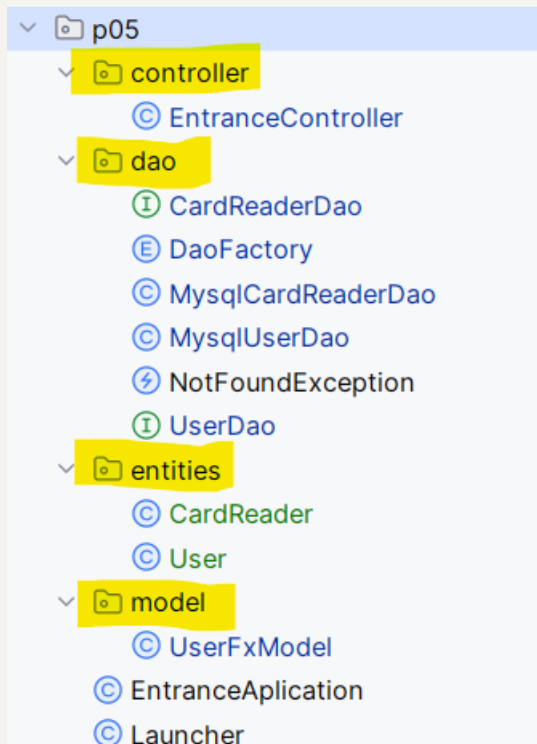
DaoFactory

Jedna továrneň na viac vecí

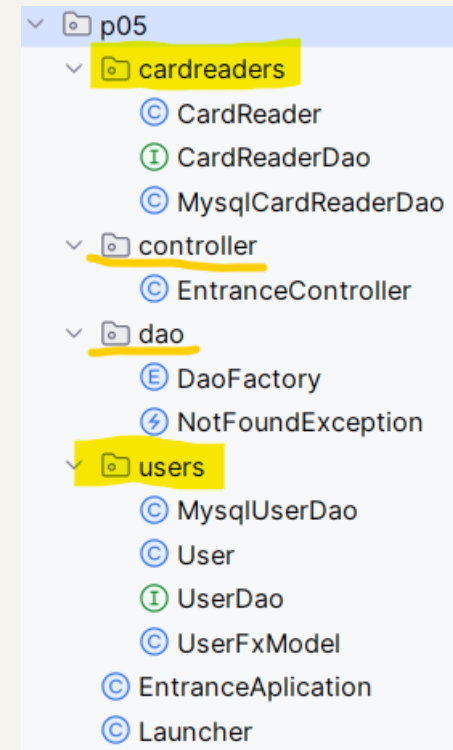
```
public enum DaoFactory {  
    INSTANCE;  
  
    public UserDao getUserDao() {  
        ...  
    }  
  
    public CardReaderDao getCardReaderDao() {  
    }  
}
```

Základná štruktúra kódu

Podľa vrstiev



Podľa entít



Do akých dverí môže vstúpiť používateľ?

- Asi častá otázka
- Naťahujem rovno s každým používateľom z DB

```
SELECT user.id AS idu, user.name, user.chipId, user.active,  
       user_card_reader.id_card_reader AS idcr FROM user  
JOIN user_card_reader ON user.id = user_card_reader.id_user
```

idu	name	chipId	active	idcr
1	Jano	cc1f86ab86	0	2
2	Fero	fe265d741a	1	1
2	Fero	fe265d741a	1	2

čítačky sa často opakujú, dotiahneme ďalším selektom

ResultSetExtractor

idu	name	chipId	active	idcr
1	Jano	cc1f86ab86	0	2
2	Fero	fe265d741a	1	1
2	Fero	fe265d741a	1	2

- RowMapper by vrátil pre každý riadok nového používateľa
- ResultSetExtractor vráti jeden objekt pre celý výsledok selektu

ResultSetExtractor

```
jdbcTemplate.query(sql, new
                    ResultSetExtractor<List<User>>() {
    public List<User> extractData(ResultSet rs) {
        List<User> users = new ArrayList<>();
        while(rs.next()) {
            /*spracovanie 1 riadku*/
        }
        return users;
    }
});
```


Akých používateľov má pridelená čítačka?

- Potrebujeme to zakaždým vedieť?
- Čo ak je používateľov niekoľko tisíc?
 - Chceme to zakaždým držať v pamäti?
- Lazy prístup: zistím, až keď to niekto bude chcieť
 - Urobím to cez getter čítačky, alebo popýtam priamo CardReaderDao?
 - Dobrá rada: Nikdy si nepamätajme inštanciu Dao v objektoch entít
 - Entity by mali byť ľahko serializovateľné
 - Intuícia:
 - getAll asi nemusí inicializovať čítačky
 - findById by mal inicializovať všetko

Serializovateľnosť entít

- Entity sa obvykle posielajú hore dole medzi službami, do externých GUI,...
- Entita má mať v inštančných premenných iba svoje atribúty a asociácie s inými entitami
 - nikdy nie objekty business logiky, objekty perzistentnej vrstvy, komponenty, ani modely
- Pozor na zacyklenie
 - napr. používateľ si pamätá svoje čítačky, čítačky svojich používateľov
 - mnohé automatické serializácie zlyhávajú, je potrebné ich štelovať, alebo to robiť ručne

OBJECT-RELATIONAL MAPPING

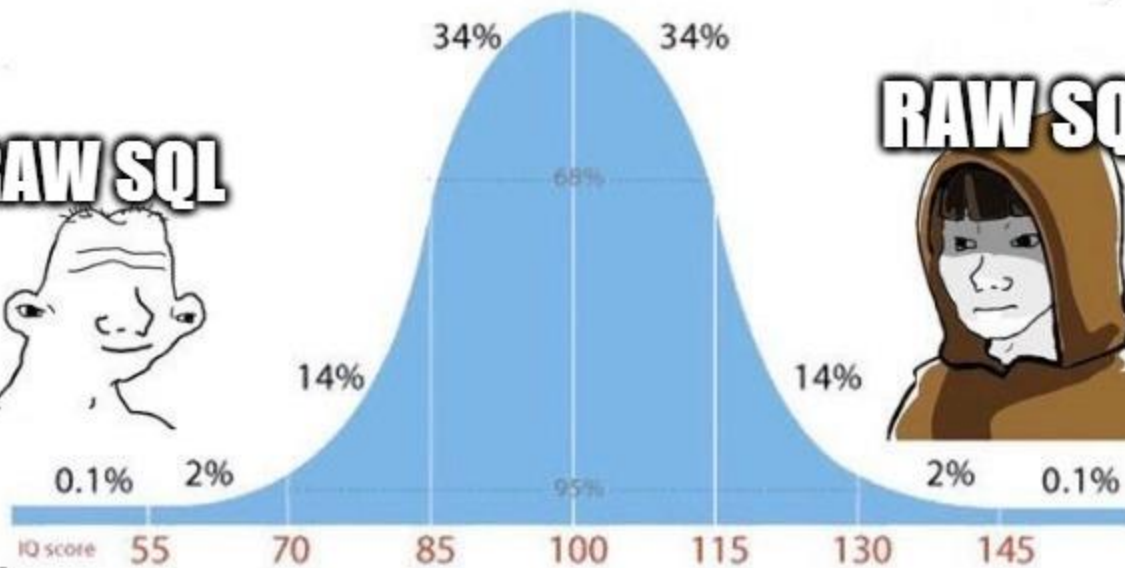


**SPRING
DATA JPA**

RAW SQL



RAW SQL



Otázky?