ORMLite

Sadržaj

Objektno-relaciono mapiranje i ORMLite

- Objektno-relaciono mapiranje je tehnika konvertovanja podataka koji se čuvaju kao objekti u memoriji u podatke koji se mogu poslati u bazu podataka, takođe se mogu i podaci iz baze konvertovati u objekte odgovarajućeg tipa u memoriji
- ORMLite je alat za objektno-relaciono mapiranje
- ORMLite nudi osnovi skup funkcionalnosti za mapiranje Java klasa na odgovarajuće tabele u bazi
- Sakriva pozive JDBC API i omogućava komunikaciju sa bazom podataka koristeći koncepte objekto-orjentisanog programiranja instanciranja objekata i poziva odgovarajućih metoda za slanje odgovarajućih SQL izraza bazi
- Sajt ORMLite alata: `http://ormlite.com/`

Uspostavljanje konekcije

- Uspostavimo konekciju sa bazom preko koje možemo da pošaljemo naredbe bazi

```
ConnectionSource connectionSource = new
    JdbcConnectionSource("jdbc:sqlite:restoran.db");
    //naredbe koje saljemo bazi
    connectionSource.close();
```

- Dok je konekcija uspostavljena možemo da šaljemo naredbe bazi
- Nakon što pošaljemo sve naredbe potrebno je pozvati metodu `close()` kako bi omogućili drugim programima da se konektuju na bazu

DatabaseTable anotacija

- Java klase treba mapirati na odgovarajuće tabele u bazi
- Anotacija DatabaseTable se koristi da odredi koja tabela odgovara kojoj Java klasi

```
@DatabaseTable(tableName = "jelo")
public class Jelo {
public Jelo(){
//konstruktor bez parametara obavezan za ORMLite
}
}
```

- tableName je atribut anotacije kojim se određuje naziv tabele u bazi kojoj odgovara klasa koja se definiše
- Ako se ne navede tableName koristiće se naziv klase kao naziv tabele u bazi
- Obavezno se mora navesti konstruktor bez parametara, jer ORMLite preko njega instancira nove objekte kad vraća vrednosti iz baze

DatabaseField anotacija

- Atributi klase se mapiraju na odgovarajuće kolone u bazi
- Anotacija DatabaseField se koristi da odredi dodatna podešavanja za kolone u bazi

```
@DatabaseField(columnName = POLJE_NAZIV, canBeNull = false , unique=false)
private String naziv;
```

- columnName je atribut anotacije kojim se određuje naziv kolone u bazi, ako se ne navede koristiće se naziv atributa klase kao naziv kolone
- canBeNull atribut određuje da li kolona mora da sadrži sačuvanu vrednost ili može da se upiše vrednost NULL u koloni, ako se ne navede podrazumevana vrednost je true
- unique atribut određuje da li vrednosti u koloni moraju biti jedinstvene, podrazumevana vrednost je false

Primarni ključ

- Da bi se odredilo koji atribut klase će čuvati vrednosti primarnog ključa određuje se na osnovu atributa id anotacije DatabaseField
- ```
  @DatabaseField(id=true)
  private int id;
  ```
- Primarni ključ može biti vrednost koja se automatski generiše prilikom upisa novih redova u tabelu
- Ako hoćemo da se vrednosti automatski generišu postavljamo atribut generateId umesto atributa id anotacije DatabaseField

  ```
  @DatabaseField(generatedId = true)
  private int id;
  ```

- Atribut generateId određuje da atribut klase čuva vrednosti primarnog ključa i da se automatski generiše prilikom upisa u bazu
  ```

Tipovi vrednosti

- Na osnovu tipa atributa klase određuje se koji će tip biti korišćen za kolone u bazi prilikom kreiranja tabela
- ORMLite podržava primitivne tipove, wrapper klase, String i Date tipove podataka
- Za veću kontrolu koji se tip koristi može se postaviti vrednost atributa dataType anotacije DatabaseField

Tipovi vrednosti

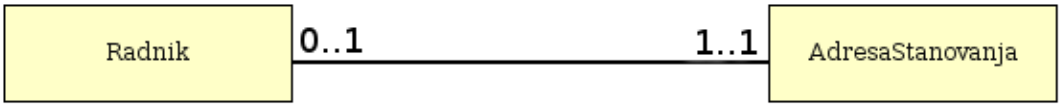
Java tip	dataType vrednost	SQL tip
String	DataType.STRING	VARCHAR
	DataType.LONG_STRING	LONGVARCHAR
	DataType.STRING_BYTES	VARBINARY
boolean or Boolean	DataType.BOOLEAN or DataType.BOOLEAN_OBJ	BOOLEAN
java.util.Date	DataType.DATE	VARCHAR
	DataType.DATE_LONG	LONG
byte or Byte	DataType.BYTE or DataType.BYTE_OBJ	TINYINT
byte array	DataType.BYTE_ARRAY	VARBINARY
char or Character	DataType.CHAR or DataType.CHAR_OBJ	CHAR
short or Short	DataType.SHORT or DataType.SHORT_OBJ	SMALLINT

Tipovi vrednosti

Java tip	dataType vrednost	SQL tip
int or Integer	DataType.INTEGER or DataType.INTEGER_OBJ	INTEGER
long or Long	DataType.LONG or DataType.LONG_OBJ	BIGINT
float or Float	DataType.FLOAT or DataType.FLOAT_OBJ	FLOAT
double or Double	DataType.DOUBLE or DataType.DOUBLE_OBJ	DOUBLE
enum or Enum	DataType.ENUM_STRING	VARCHAR
	DataType.ENUM_INTEGER	INTEGER
BigInteger	DataType.BIG_INTEGER	VARCHAR
BigDecimal	DataType.BIG_DECIMAL	VARCHAR
	DataType. BIG_DECIMAL_NUMERIC	NUMERIC

1:1 veza

- Veza jedan prema jedan postoji između entiteta Radnik i AdresaStanovanja



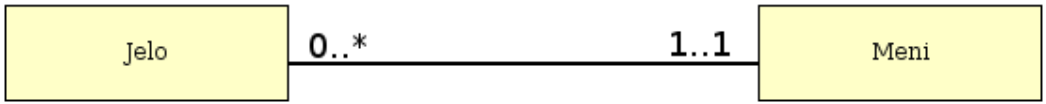
- Veza između Radnika i AdresaStanovanja se definiše kao poseban atribut adresa, klase Radnik, koji je tipa AdresaStanovanja
- Za atribut adresa dodaje se anotacija DatabaseField

```
@DatabaseField(foreign = true, foreignAutoRefresh = true, canBeNull =
    false)
private AdresaStanovanja adresa;
```

- Atribut foreign anotacije DatabaseField određuje da je ovo strani ključ iz tabele AdresaStanovanja
- Atribut foreignAutoRefresh anotacije DatabaseField određuje da li se prilikom učitavanja radnika iz baze učitavaju i sve vrednosti za adrese stanovanja
- Ako je atribut foreignAutoRefresh postavljen na false prilikom instanciranja objekta tipa Radnik instancira se i objekat tipa AdresaStanovanja ali samo sa postavljenom vrednošću za primarni ključ
- ORMLite ne podržava način da se definiše drugi kraj veze na strani klase AdresaStanovanja

1:N veza

- Veza jedan prema više postoji između entiteta Jelo i Meni



- Za kraj veze više dodaje se atribut meni za klasu Jelo koji je tip Meni
- Za atribut meni dodaje se anotacija DatabaseField

```
@DatabaseField(foreign = true, foreignAutoRefresh = true, canBeNull =
    false)
private Meni meni;
```

- Atribut foreign anotacije DatabaseField određuje da je ovo strani ključ iz tabele Meni
- Atribut foreignAutoRefresh anotacije DatabaseField određuje da li se prilikom učitavanja jela iz baze učitaju i sve vrednosti za menije
- Ako je atribut foreignAutoRefresh postavljen na false prilikom instanciranja objekta tipa Jelo instancira se i objekat tipa Meni ali samo sa postavljenom vrednošću za primarni ključ

1:N veza

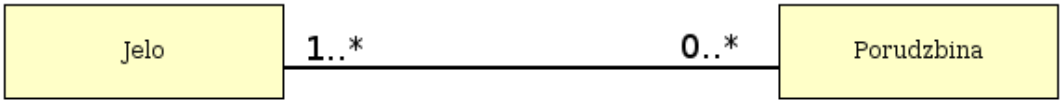
- Za jedinični kraj veze dodaje se atribut jela u klasi Meni koji je tipa ForeignCollection<Jelo>
- Za atribut jela dodaje se anotacija ForeignCollectionField

```
@ForeignCollectionField(foreignFieldName =
      "meni",eager=false,maxEagerLevel = 1)
private ForeignCollection<Jelo> jela;
```

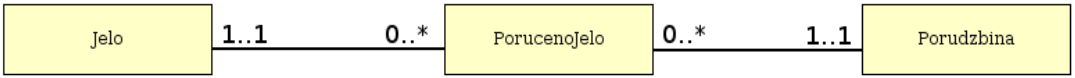
- Atribut foreignFieldName anotacije ForeignCollectionField određuje naziv atributa koji predstavlja kraj veze sa jedinične strane
- Atribut eager određuje da li se prilikom učitavanja menija iz baze učitavaju i sva jela
- Predefinisana vrednost za eager je false i određuje da se jela neće odmah učitati nego tek kad se bude iteriralo kroz kolekciju
- Atribut maxEagerLevel određuje da li prilikom učitavanja jela treba da učitava i kolekcije koje mogu biti definisane u klasi Jelo i tako rekurzivno do određenog nivoa
- Predefinisana vrednost atributa maxEagerLevel je 1 tako da se učitavaju samo jela ali kolekcije u klasi Jelo neće biti učitane
- Svako učitavanje kolekcije zahteva poseban pristup bazi i prilikom postavljanja vrednosti atributa eager i maxEagerLevel treba uzeti u obzir vreme koje će biti potrebno da se sve te vrednosti dobiju iz baze

M:N veza

- Veza više prema više postoji između entiteta Jelo i Porudzbina



- ORMLite ne podržava mogućnost definisanja ovakvog tipa veze
- Veza više prema više se može razdvojiti na dve posebne veze jedan prema više
- Veza jedan prema više između Jelo i PorucenoJelo i veza jedan prema više između Porudzbina i PorucenoJelo



- Ovako dobijene veze jedan prema više se mogu napisati kao što je već objašnjeno u prethodnim slajdovima za vezu jedan prema više

Kreiranje i brisanje tabela

- Za rad sa tabelama definisane su pomoćne metode u klasi TableUtils
- Ovim metodama se direktno prosleđuju klase u kojima su definisane potrebne anotacije i na osnovu anotacija se kreiraju odgovarajuće SQL naredbe koje se šalju bazi
- Za kreiranje tabele poziva se metoda createTable

```
TableUtils.createTable(connectionSource, Jelo.class);
```

- Za brisanje tabela poziva se metoda dropTable

```
TableUtils.dropTable(connectionSource, Jelo.class, true);
```

- Za brisanje vrednosti iz tabela poziva se metoda clearTable

```
TableUtils.clearTable(connectionSource, Jelo.class);
```

Data Access Object (DAO)

- Za rad sa vrednostima upisanim u tabelama koriste se metode definisane u interfejsu Dao
- Za svaki poseban entitet potrebno je instancirati poseban Dao objekat

```
Dao<Jelo, Integer> jeloDao = DaoManager.createDao(connectionSource,
    Jelo.class);
```

- Dao interfejs je definisan kao generički interfejs kojem se određuju dve vrednosti
  - Tip entiteta koji odgovara tabeli u bazi i za koji će se koristiti pomoćne metode
  - Tip primarnog ključa koji je definisan u entitetu za koji se kreira Dao objekat

Dodavanje, izmena i brisanje vrednosti

- Za dodavanje, izmenu i brisanje vrednosti koriste se sledeće metode:
  - create - za upisivanje vrednosti u bazu prosleđivanjem instanciranog objekta odgovarajućeg tipa

```
Jelo j1 = new Jelo("Spagete", "Testo sa mesom", 400, m1);
jeloDao.create(j1);
```

  - update - izmena vrednost, prosleđivanjem objekta koji je učitan iz baze ili koji je bio kreiran, a kojem je promenjena vrednost atributa. Mora imati postavljenu vrednost za atribut primarnog ključa

```
Jelo jeloZaIzmenu = jeloDao.queryForId(j1.getId());
jeloZaIzmenu.setOpis("Rezanci sa mesom");
jeloDao.update(jeloZaIzmenu);
```

  - delete - brisanje vrednosti, prosleđivanjem objekta koji je učitan iz baze ili koji je bio kreiran. Mora imati postavljenu vrednost za atribut primarnog ključa

```
Jelo jeloZaBrisanje = jeloDao.queryForId(j6.getId());
jeloDao.delete(jeloZaBrisanje);
```

Pretraga i vraćanja vrednosti entiteta

- Dao interfejs definiše i pomoćne metode za pretragu i vraćanje vrednosti entiteta za koji je instanciran:
  - queryForAll - vraća sve vrednosti odgovarajućeg entiteta iz tabele

```
List<Jelo> jela = jeloDao.queryForAll();
for (Jelo j : jela)
     System.out.println(j);
```

  - queryForId - za pronalaženje objekta odgovarajućeg entiteta prosleđivanjem vrednosti primarnog ključa

```
Jelo jeloZaIzmenu = jeloDao.queryForId(j1.getId());
```

  - queryForEq - pretraga vrednosti odgovarajućeg entiteta, upoređivanjem vrednosti jednog od atributa

```
List<Jelo> jela=jeloDao.queryForEq(Jelo.POLJE_CENA,200);
for (Jelo j : jela)
     System.out.println(j);
```

Prolazak kroz kolekciju ForeignCollection

- Prolazak kroz kolekciju ForeignCollection se vrši dobijanje CloseableIterator objekta nad kojim su definisane metode:
  - hasNext - kojom se proverava da li postoji element koji se može preuzeti iz kolekcije, ako je kolekcija prazna vrati će false
  - next - preuzima se trenutni element i prelazi se na sledeći ako postoji u kolekciji

```
ForeignCollection<Jelo> jelaMenija=m.getJela();

CloseableIterator<Jelo> iterator=jelaMenija.closeableIterator() ;

try {

while (iterator.hasNext()) {
    Jelo j = iterator.next();
    System.out.println("j = " + j);
}
} catch(Exception e)
{
    System.out.println("Greska prilikom iteracije");
}finally {
    iterator.close();
}
```

- Nakon prolaska kroz celu kolekciju moraju se osloboditi resursi koje koristi CloseableIterator pozivom metode close

Select izraz

- Za kreiranje select izraza koristi se posebna metoda queryBuilder definisana u Dao interfejsu
- Metoda queryBuilder vraća objekat tipa QueryBuilder nad kojim se može dodati izraz za where klauzulu

```
QueryBuilder<Jelo,Integer> jeloCenaNazivQuery=jeloDao.queryBuilder();
Where<Jelo,Integer> where=jeloCenaNazivQuery.where();
where.eq(Jelo.POLJE_CENA,200).and().like(Jelo.POLJE_NAZIV,"%e%");
PreparedQuery<Jelo> jeloCenaPripremljen=jeloCenaNazivQuery.prepare();
List<Jelo> jela=jeloDao.query(jeloCenaPripremljen);
for(Jelo j:jela)
    System.out.println("j = " + j);
```

- Objekat tipa Where služi za generisanje where klauzule, pozivom metoda eq, and i like koje odgovaraju operatorima koji postoje u SQL-u
- Svaka posebna metoda vraća isti objekat tipa Where tako da se u jednoj naredbi može kreirati ceo izraz

Select izraz

SQL operator	metoda
OR	or
AND	and
NOT	not
IS NULL	isNull
IS NOT NULL	isNotNull
=	eq
<>	ne
<	lt
>	gt
>=	ge
<=	le
LIKE	like
BETWEEN	between
IN	in
NOT IN	notIn

SelectArg

- Kada je potrebno pozvati isti select izraz, ali sa drugim vrednostima za poređenje koristi se SelectArg objekat
- SelectArg se postavlja prilikom poziva metoda, koje odgovaraju SQL operatorima, umesto vrednosti sa kojima se poredi

```
SelectArg selectZaOpis=new SelectArg();
PreparedQuery<Jelo> jeloQueryPripremljen=
                jeloDao.queryBuilder()
                .where().like(Jelo.POLJE_OPIS,selectZaOpis).prepare();
selectZaOpis.setValue("%mesom%");
jela=jeloDao.query(jeloQueryPripremljen);
for(Jelo j:jela)
System.out.println("j = " + j);

selectZaOpis.setValue("%sa%");
jela=jeloDao.query(jeloQueryPripremljen);
for(Jelo j:jela)
    System.out.println("j = " + j);
```

- Svaka posebna vrednost koja se poredi treba da se zameni sa posebnim SelectArg objektom
- Vrednosti prosleđene SelectArg objektu se u where klauzulu dodaju sa posebnim karakterima kojima se sprečava mogućnost SQL injection napada

Spajanje dve tabele JOIN

- Kada je potrebno spojiti vrednosti iz dve tabele može se definisati select izraz sa JOIN operatorm
- Objekat QueryBuilder definiše metodu join koja odgovara SQL JOIN operatoru
- Za svaku tabelu koja se spaja JOIN operatorom mora se definisati poseban QueryBuilder objekat

```
QueryBuilder<Jelo, Integer> jeloQuery=jeloDao.queryBuilder();

QueryBuilder<PorucenoJelo, Integer>
    porucenoJeloQuery=porucenoJeloDao.queryBuilder();

porucenoJeloQuery.where().eq(PorucenoJelo.POLJE_KOLICINA,2);

List<Jelo> jela=jeloQuery.join(porucenoJeloQuery).query();
for(Jelo j:jela)
    System.out.println("j = " + j);
```

- Prilikom korišćenja join metoda mogu se povezati samo entiteti koji su međusobno povezani atributom kojem je postavljena anotacija DatabaseField i vrednost foreign na true
- Prilikom spajanja tabela za vrednosti koje se vraćaju mogu se dobiti samo vrednosti jednog tipa entiteta

Transakcije

- Transakcije omogućavaju da se definiše skup naredbi koje je potrebno da budu izrvšene odjednom
- Ako jedna naredba ne uspe da se izvrši i sve ostale naredbe u transakciji neće biti izvršene
- Ako hoćemo da obrišemo meni potrebno je da obrišemo jela koja su određena za taj meni
- Naredbe za brisanje pozivamo unutar transakcije i ako ne uspeju da se obrišu sva jela ili meni sve naredbe za brisanje će biti poništene

```
TransactionManager.callInTransaction(connectionSource,
new Callable<Void>() {
    public Void call() throws Exception {

        jeloDao.delete(zaBrisanje);

        meniDao.delete(m2);

        return null;
    }
});
```

- Za transakcije koristi se klasa TransactionManager koja ima definisanu metodu callInTransaction
- Metodi callInTransaction se prosleđuje vrednost objekat tipa Callable, koji je generički tip i kojem se određuje tip objekta koji se vraća kao povratna vrednost metode call
- Koristeći wrapper klasu za Java ključnu reč void postavljeno je da Callable objekat ne mora da vraća vrednost