ORM

**Bevezetés**

Az ORM, az angol Object-Relation Mapping kifejezés mozaik szava, magyar jelentése az Objektum-Relációs Leképzés. Ez egy, a programozás módszertanában gyakran használt technika, amikor is egyfajta hidat képzünk az objektumorientált programozási nyelvekben létrehozott objektumok és a relációs adatbázisok táblái között. Másszóval, az MVC rétegszemléletét követve ez a réteg kapcsolja össze az Entitást az Adatbázissal. Ezt meg lehet oldani SQL kódokkal és ORM eszközökkel.

Példa:

DB

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

**Perzisztencia**

A perzisztencia általában adatok hosszú távú tárolását jelenti, azzal a céllal, hogy az adatok túléljék azokat a folyamatokat, amelyek létrehozták őket. A modell lényege, hogy az adatokat tartsuk a memóriában ahelyett, hogy relációs adatbázisba képeznénk le őket, hogy a memória állapota visszaállítható legyen egy esetleges rendszerleállás/áramszünet, stb. után. A memória alapú adatbázis rendszer minden műveletet naplóz, valamint bizonyos időközönként egy mentést készít a memória állapotáról. Ha a rendszert újraindítjuk, a mentést visszatöltve, és a műveleteket a log alapján végrehajtva visszakapjuk a memória eredeti állapotát.

**Entitás**

Az "entitás" fogalma alapvető jelentőséggel bír az informatikában és több kontextusban is használatos. Általában egy entitás egy jól meghatározott, egyedi azonosítóval rendelkező objektumot vagy "dolog"-ot jelent, valamilyen kontextusban vagy rendszerben létezik, és azzal kapcsolatos adatokat tárol.

Objektumorientált programozásban egy entitás egy objektumot vagy egy osztály példányát jelölheti. Itt az entitás jellemzőit a példány változói és metódusai adják, amelyek az osztály definícióiból erednek. Például.: egy Autó osztály esetében az entitás lehet egy konkrét autó példány, amely rendelkezik saját színnel, márkával és rendszámmal.

**Adatbázissal való kapcsolat**

Az adatbázissal való kapcsolatot minden nyelven, minden keretrendszerben máshogy kell megoldani. Én a Java Springboot-ban fogom ezt bemutatni.

A Springboot új projekt létrehozásakor generál egy .properties fájlt. Ebbe a fájlba kell beleírni a különböző adatokat, (pl.: ip cím, adatbázis név, felhasználó név, jelszó)

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírásMint az ebből a kódrészletből látható, számunkra a legfontosabb sorban a JDBC, azaz Java Database Connectivity könyvtár biztosítja nekünk ezt a bizonyos kapcsolatot a Hibernate-tel.

**CRUD**

Amikor egy OOP nyelv segítségével szeretnénk egy adatbázissal kommunikálni, akkor mindenképpen fogunk létrehozni, olvasni, frissíteni és törölni adatokat, ezt hívják CRUD-nak, ezekhez általában SQL parancsokat használunk. Habár az alap ötlet, hogy ehhez SQL parancsokat használjunk, nem rossz, de nem is a legjobb. Ekkor jönnek be a képbe az ORM eszközök, amik segítségével az objektum orientált nyelvben tudjuk megírni a lekérdezéseinket. A legtöbb objektum-orientált nyelv rendelkezik ORM eszközzel (Java – Hibernate, C# - Entity Framework). Én a következő példámban a Java springframework.data.repository csomag CrudRepository-ját fogom használni

**Feladat**

Erre egy rövid pl.:

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Először is szükségünk lesz egy entitásra, azaz egy @Entity annotációval ellátott osztályra. Az entitás a @Table annotációval kapja meg, hogy melyik tábla leképzése. Ennek az osztálynak minden változója egy @Column annotáció, ami megadja, hogy az adott változó melyik oszlopnak felel meg a táblában.

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

Ezt Java-ban írt, natív SQL lekérdezéssel írtam meg visszaadja a users tábla azon objektumát, ahol a username „Pistike02” vagy tetszőleges felhasználónévre. Az adatbázishoz való kapcsolódást ekkor nem a .properties fájlba kell írni, hanem a kódba kell beleírni, majd midig ellenőrizni, hogy van-e kapcsolat. Ez a módszer nem valami moduláris, illetve hosszú időbe telik megírni. Egy ORM eszköz segítségével viszont elég ennyit írni:

A képen képernyőkép, Betűtípus, szöveg, sor látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Látható, hogy az ORM eszköz segítségével nemcsak könnyebb volt megírni a lekérést, de egy sokkal átláthatóbb kódot kaptunk, ami rögtön a user osztályunk egy példányát adta vissza, már ha létezik. Ha egy kis változtatással általánosabbá tesszük a kódot, akkor nem „Pistike02” az egyetlen felhasználó, akit lekérhetünk az adatbázisból, hanem akárkit. Ehhez mindenképpen szükségünk van egy a CrudRepository-t öröklő interfészre. Ez az interfész tartalmaz egy API-t, ami a beírt függvénynevet lefordítja SQL lekéréssé.

**Összehasonlítás:**

|  |  |
| --- | --- |
| SQL | ORM |
| INSERT INTO | save(példány) |
| UPDATE | save() //Ha már létezik az objektum a DB-ben |
| SELECT \* FROM {táblanév} WHERE … | findBy…() vagy findAllBy…() |
| SELECT \* FROM {táblanév} | findAll() |
| DELETE FROM | delete(példány) |

**Előnyök:**

* Felgyorsítja a fejlesztést.
* Lecsökkenti a fejlesztés költségét.
* Leegyszerűsíti az adatbázissal való kommunikációt.
* Biztonságosabbak, mert általában ezek az eszközök rendelkeznek védelemmel SQL-injection-ök (SQL-injekciók) ellen.
* Kevesebb kódot kell írni.

**Hátrányok:**

* Hosszú idő lehet megtanulni.
* Komplex lekérdezéseknél lehet, hogy nem teljesítenek olyan jól, mint egy SQL lekérdezés.
* Lassabb, mint egy SQL lekérdezés.