ORMFramework

Description

An object relational mappings i.e ORM is a framework that gives facilities related to SQL on backend to ease data layer work. The framework is developed in Java and having usage of Java's Reflection API & Collections Framework. User who is using the ORM Framework doesn't need to bother about writing hundred's of thousand's of SQL code to push & pull data from database. Along with this , user doesn't need to worry about writing POJO (Entity) classes with appropriate annotatins. Our framework will take care of creating POJO's all from scratch just by analyzing Tables & will do proper structuring along with class level & field level annotations.

Features

- POJO Generator
- Custom Annotations
- Overcomes the headache of writing SQL
- Single JAR availability so you just have to include it in classpath & do magic

Installing

- Install JDK 1.8 on your local machine
- Download the ORM.jar from master branch & put it in your libs of working directory
- Install MYSQL version 8 to overcome some minor bugs & errors
- Include ORM.jar at compile time & runtime
- Create conf.json file in current working directory having JSON formatting similar to conf.json of master branch

Test Cases

- Save

```
DataManager dataManager=new DataManager();
dataManager.begin();
Vehicle vehicle=new Vehicle();
vehicle.setCode(101);
```

```
vehicle.setName("Sonet");
vehicle.setBrandName("Kia");
vehicle.setColor("Red");
dataManager.save(vehicle);
dataManager.end();
```

- Update

```
DataManager dataManager=new DataManager();
dataManager.begin();
Animal animal=new Animal();
animal.setName("Turtle Hanky");
animal.setType("Reptile");
animal.setAge(53);
dataManager.update(animal);
dataManager.end();
```

- Delete

```
DataManager dataManager=new DataManager();
dataManager.begin();
Vehicle vehicle=new Vehicle();
vehicle.setCode(201);                    //the record having primary key column as
code & value as 201
dataManager.update(animal);
dataManager.end();
```

- Select
  - Select Using Statement

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> animals=dataManager.select(new Animal()).fire();
for(Object object:animals)
{
Animal animal=(Animal) object;
System.out.println("Name : "+animal.getName());
System.out.println("Type : "+animal.getType());
System.out.println("Age : "+animal.getAge());
}
dataManager.end();
```

- Select Using Prepared Statement

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehicleCode").eq(101).fire();    //eq for equals
dataManager.end();
```

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehicleCode").ne(101).fire();    //ne for not equals
dataManager.end();
```

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehiclePrice").lt(1500000).fire();    //lt for less than
dataManager.end();
```

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehiclePrice").gt(2000000).fire();    //gt for greater than
dataManager.end();
```

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehiclePrice").le(7000000).fire();    //le for less than equal
to
dataManager.end();
```

```
DataManager dataManager=new DataManager();
dataManager.begin();
List<Object> vehicles=dataManager.select(new
Vehicle()).where("vehiclePrice").ge(7000000).fire();    //ge for greater than
equal to
dataManager.end();
```

Version History

- 0.2 - Upcoming
  - Logger facility
- 0.1
  - Initial Release

Acknowledgments

- https://github.com/google/gson
- https://docs.oracle.com