ORM-FRAMEWORK

Introducing a Java-based Object Relational Mapping (ORM) framework framework based on Java 18 that is specifically designed for seamless integration with MySQL Database Management System. This framework simplifies database interactions, allowing developers to easily map Java objects to MySQL tables, making it easier to manage and manipulate data within your applications

Benefit of using this framework.

1. No need to write SQL statements for insertion,deletion,retrival and updation.

2. No need to apply validations related to data.

3. No need to create pojo's for the database tables, all the pojos will be created in the specified folder structure and jar file will be created of compiled pojo's.

4. User only need to create a file named as conf.json in the working directory that contains required information to establish connection with the database.

Annotations:-

Package : com.thinking.machines.orm.annotations

If the user solely relies on the provided tools, there is no need to use or worry about annotations. However, if the user manually creates the POJOs, the following annotations must be used:

Annotation	**Description**
Table	This annotation is used with the class. The value of this annotation is the table name that the user wants to map to the class
View	This annotation is applied to the class, where its value represents the name of the view the user intends to map to that class. To simplify performing joins or retrieving data from multiple tables, create a view in the database for easier access and use.
Column	This annotation is applied to the fields of the class. Its value specifies the column name in the table that the user intends to map with that field.
PrimaryKey	This Annotation used with the field representing the primary key of the table.
AutoIncrement	This annotation is used with a field mapped to an auto-incremented column.
ForeignKey	This Annotation used with the field that is mapped to the foreign key of the table.

DataManager Class :-

Package : com.thinking.machines.orm

The connection is established and stored in the DataManager object, equipped with the necessary data structure for conducting operations and validations.

NOTE :- Users can only perform retrieval operations on fields annotated with @View. Operations such as insertion, updating, and deletion are not allowed on fields annotated with @View

Method	Return Type	Description
getDataManager()	DataManager	A static method returns the instance of the DataManager. The Framework is uses singleton pattern.
begin()	void	To perform the below operations user need to call begin() method first, it stablish the connection with database.
init()	void	The init method should be called after begin because it manages the in-memory database.
save(Object)	int	It maps the received object relationally and saves the information into the database. A non-zero return value confirms successful saved of the data.
update(Object)	int	Updates all the details of the given object into the database. A non-zero return value confirms successful updated of the data.
delete(Class,Object)	int	Delete the row from the databasae by looking for the field mapped to the primary key in the object. A non-zero return value confirms successful deletion of the data.
query(Class)	Query	Return the list of data retrival from the database
end()	void	Closes the connection with database.

Query :-

Here are the necessary query statements for retrieval operations:

Method	Return Type	Description
where(Object)	DataManager	Add a **where** clause with parameter to the query for filtering results.
eq(Object)	DataManager	Add '**==**'(equal to) followed by the parameter of the query.
ne(Object)	DataManager	Adds '!=' (not equal to) followed by the parameter of the query.
lt(Object)	DataManager	Adds '<' (less then) followed by the parameter of the query.
gt(Object)	DataManager	Adds '>' (greater then) followed by the parameter of the query.
ge(Object)	DataManager	Adds '>=' (greater then equal to) followed by the parameter of the query.
le(Object)	DataManager	Adds '<=' (greater then) followed by the parameter of the query.
and(Object)	DataManager	Add a **and** clause with parameter to the query for filtering results.
or(Object)	DataManager	Add a **or** clause with parameter to the query for filtering results.

orderBy(Object)	DataManager	Add a **orderBy** clause with parameter to the query for filtering results.
between(Object,Object)	DataManager	Add a **between** clause with to the query for filtering results.
ascending()	DataManager	Add a **ascending** clause with to the query for filtering results.
descending()	DataManager	Add a **descending** clause with to the query for filtering results.
fire()	Object(List<Object>)	To execute the query.

Exception :-DataException

Package : com.thinking.machines.orm.exceptions

An exception that provide information related to accessing or setting up the database,validation of data arrived to perform some operation.

How to Effectively Use This Framework:- (tutorial)

Set Up Your Environment–

Download the all jar files from the jar folder and put it into the working directory or in the lib folder.

Note : Then the user need to create a file named as conf.json in the working directory with the information related to database,packaging structure and jar file.(compulsory)

conf.json file should be created as shown below :-

{

"jdbc_driver":**className,**

"connection_url":**ConnectionString**.,

"database":**database**,

"username":**username**,

"password":**password**

"packagName": **The package name will be used to generate the POJO class**,

"jarFile": **jarFile name**

}

Given Example :-

```
{
"jdbc_driver":"com.mysql.cj.jdbc.Driver",
"connection_url":"jdbc:mysql://localhost:3306/hrdb",
"database":"hrdb",
"username":"hruser",
"password":"hrpass",
"packageName":"com.thinking.machines.hr",
"jarFile":"hr"
}
```

Creating POJO's :-

To generate Plain Old Java Objects (POJOs), please use the GeneratePojoClasses method. Here's how you can call it:

java -classpath path_to_orm_framework *;.

com.thinking.machines.orm.tools.GeneratePojoClasses

Perform Insertion Operation :-

In the hrdb database, there are two tables: course and student. Additionally, a database engineer has created a view for more efficient data management and retrieval.

```
import com.thinking.machines.orm.*;
import com.thinking.machines.orm.exceptions.*;
import java.text.*;
import java.math.*;
import java.sql.*;
import java.util.*;
class eg1psp
{
public static void main(String gg[])
{
try
{
DataManager dm=DataManager.getDataManager();
dm.begin();
dm.init();
Course c=new Course();
c.setTitle("React Native");
int code=dm.save(c);
System.out.println("Code : "+code);

Student s=new Student();
s.setRollNumber(100);
s.setFirstName("Deepak");
s.setLastName("Porwal");
s.setAadharCardNumber("ABCDE1234");
s.setCourseCode(10);
s.setGender("M");
SimpleDateFormat simpleDateFormat=new SimpleDateFormat("yyyy-MM-dd");
String dateString = "2002-07-07";
java.util.Date utilDate=simpleDateFormat.parse(dateString);
java.sql.Date date=new java.sql.Date(utilDate.getTime());
s.setDateOfBirth(date);
```

```
dm.save(s);

dm.end();
}catch(DataException dataException)
{
System.out.println("Exception : "+dataException);
}catch(Exception exception)
{
System.out.println("Do nothing"+exception);
}
}
}
```

Compile: To compile your code, use the following command

javac -classpath path_to_orm_framework/*;path_to_pojo_jar_file;. YourFileName.java

Replace path_to_orm_framework with the actual path to your ORM framework JAR files, path_to_pojo_jar_file with the path to your POJO JAR file, and YourFileName.java with the name of your Java file.

In my Example

javac -classpath path__to_orm_framework /*;com/thinking/machines/hr/dist/hr.jar;. YourFileName.java

Run the JAR: Execute your JAR file:

java -classpath path_to_orm_framework/*;path_to_pojo_jar_file;. YourFileName

In my Example

java -classpath path_to_orm_framework*;com\thinking\machines\hr\dist\hr.jar;. eg1psp

Perform Updation Operation :-

```
import com.thinking.machines.orm.*;
import com.thinking.machines.orm.exceptions.*;
import java.text.*;
import java.math.*;
import java.sql.*;
import java.util.*;
class eg2psp
{
public static void main(String gg[])
{
try
{
DataManager dm=DataManager.getDataManager();
dm.begin();
dm.init();
Course c=new Course();
c.setCode(10);
c.setTitle("C++(Data Structures)");
dm.update(c);

Student s=new Student();
s.setRollNumber(100);
s.setFirstName("Krishna");
s.setLastName("Sharma");
s.setAadharCardNumber("123456ABCD");
s.setCourseCode(11);
s.setGender("M");

SimpleDateFormat simpleDateFormat=new SimpleDateFormat("yyyy-MM-dd");
String dateString = "2002-06-04";
java.util.Date utilDate=simpleDateFormat.parse(dateString);
java.sql.Date date=new java.sql.Date(utilDate.getTime());
s.setDateOfBirth(date);
dm.update(s);


dm.end();
}catch(DataException dataException)
{
System.out.println("Exception : "+dataException);
}catch(Exception exception)
{
System.out.println("Do nothing"+exception);
}
```

```
}
}
```

Perform Deletion Operation :-

```
import com.thinking.machines.orm.*;
import com.thinking.machines.orm.exceptions.*;
import java.text.*;
import java.math.*;
import java.sql.*;
import java.util.*;
class eg2psp
{
public static void main(String gg[])
{
try
{
DataManager dm=DataManager.getDataManager();
dm.begin();
dm.init();
dm.delete(Course.class,90);//Primary key
dm.delete(Student.class,100);
dm.end();
}catch(DataException dataException)
{
System.out.println("Exception : "+dataException);
}catch(Exception exception)
{
System.out.println("Do nothing"+exception);
}
}
}
```

Perform Retrival Operation :-

View Annotation :- To simplify performing joins or retrieving data from multiple tables, create a view in the database for easier access and use.

In the given example, the StudentView POJO (generated by the GeneratePojoClasses tool) represents the view. This allows for easy usage and access to the data. Just make sure to create the view in your database.

```
import com.thinking.machines.orm.*;
import com.thinking.machines.orm.exceptions.*;
import java.text.*;
import java.math.*;
import java.sql.*;
import java.util.*;
class eg3psp
{
public static void main(String gg[])
{
try
{
DataManager dm=DataManager.getDataManager();
dm.begin();
dm.init();
List<Course> courses=(List<Course>)dm.query(Course.class).fire();
for(Course c:courses)
{
System.out.println("Code : "+c.getCode()+"  Title : "+c.getTitle());
}

List<Course>
c=(List<Course>)dm.query(Course.class).where("code").between(90,92).fire();
for(Course clz:c)
{
System.out.println("Code : "+clz.getCode()+" Title : "+clz.getTitle());
}

// View created by the database designer StudentView is a View
List<StudentView>
students=(List<StudentView>)dm.query(StudentView.class).orderBy("roll_number"
).fire();
for(StudentView s:students)
{
```

```
System.out.println("First Name : "+s.getFirstName()+" Last Name :
"+s.getLastName());
System.out.println("Course title : "+s.getCourseTitle());

}

dm.end();
}catch(DataException dataException)
{
System.out.println("Exception : "+dataException);
}catch(Exception exception)
{
System.out.println("Do nothing"+exception);
}
}
}
```

----------End----------