Implementare un ORM di base per PostgreSQL in Java

In questo articolo vedremo come creare un ORM (Object-Relational Mapping) di base per PostgreSQL utilizzando Java. Un ORM consente di mappare oggetti Java su tabelle di un database relazionale, semplificando l'accesso ai dati.

1. Configurazione del progetto

Creiamo un progetto Java con il driver PostgreSQL nel classpath. Se usi Maven, aggiungi questa dipendenza nel file `pom.xml`:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.1</version>
</dependency>
```

2. Definizione di una classe Entity

Definiamo una semplice classe `User` come entità da salvare nel database.

```
public class User {
    private int id;
    private String name;
    private String email;

    // Costruttori, getter e setter
}
```

3. Creazione della classe ORM

Scriviamo una classe `EntityManager` con metodi per salvare e caricare entità nel database.

```
import java.lang.reflect.Field;
import java.sql.*;

public class EntityManager {

    private Connection connection;

    public EntityManager(Connection connection) {
        this.connection = connection;
    }

    public void save(Object entity) throws Exception {
        Class<?> clazz = entity.getClass();
        String tableName =
clazz.getSimpleName().toLowerCase();
        Field[] fields = clazz.getDeclaredFields();

        StringBuilder columns = new StringBuilder();
        StringBuilder values = new StringBuilder();
        for (Field field : fields) {
            field.setAccessible(true);
            if (!field.getName().equals("id")) {
                columns.append(field.getName()).append(",");

values.append("'").append(field.get(entity)).append("',");
            }
        }

        String sql = String.format(
            "INSERT INTO %s (%s) VALUES (%s)",
            tableName,
            columns.substring(0, columns.length() - 1),
            values.substring(0, values.length() - 1)
        );
```

```
        try (Statement stmt = connection.createStatement()) {
            stmt.executeUpdate(sql);
        }
    }

    public <T> T find(Class<T> clazz, int id) throws
Exception {
        String tableName =
clazz.getSimpleName().toLowerCase();
        String sql = "SELECT * FROM " + tableName + " WHERE
id = " + id;

        try (Statement stmt = connection.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            if (rs.next()) {
                T entity =
clazz.getDeclaredConstructor().newInstance();
                for (Field field : clazz.getDeclaredFields())
{
                    field.setAccessible(true);
                    field.set(entity,
rs.getObject(field.getName()));
                }
                return entity;
            } else {
                return null;
            }
        }
    }
}
```

4. Esempio di utilizzo

```
public class Main {
    public static void main(String[] args) throws Exception {
        String url =
"jdbc:postgresql://localhost:5432/testdb";
        String user = "postgres";
```

```
        String password = "password";

        try (Connection conn =
DriverManager.getConnection(url, user, password)) {
            EntityManager em = new EntityManager(conn);

            User newUser = new User();
            newUser.setName("Mario Rossi");
            newUser.setEmail("mario@example.com");

            em.save(newUser);

            User loadedUser = em.find(User.class, 1);
            System.out.println("User: " +
loadedUser.getName());
        }
    }
}
```

Conclusione

Questa è una semplice implementazione di un ORM in Java. Per progetti reali, si consiglia l’uso di framework consolidati come Hibernate o JPA, ma costruire un ORM da zero è utile per comprendere i meccanismi di base dietro il mapping tra oggetti e tabelle relazionali.