
Newton
Quem se prepara, não para.

Programação Orienta a Objetos

3º período

Professora: Michelle Hanne

Sumário

- ✓ Comando de Persistência de Dados
- ✓ ORM

Create Table

Newton
Quem se prepara, não para.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class CreateTableExample {

private final String url =
"jdbc:postgresql://localhost/BDlivrariaUniversitaria";
private final String user = "postgres";
private final String password = "b0aofmd0";

    private static final String createTableSQL = "CREATE TABLE
users " +
        "(ID INT PRIMARY KEY ," +
        " NAME TEXT, " +
        " EMAIL VARCHAR(50), " +
        " COUNTRY VARCHAR(50), " +
        " PASSWORD VARCHAR(50))";

    public static void main(String[] argv) throws SQLException {
        CreateTableExample createTableExample = new
CreateTableExample();
        createTableExample.createTable();
    }
```

```
 public void createTable() throws SQLException {

        System.out.println(createTableSQL);
        // Step 1: Establishing a Connection
        try (Connection connection = DriverManager.getConnection(url, user,
password);

            // Step 2:Create a statement using connection object
            Statement statement = connection.createStatement();) {

            // Step 3: Execute the query or update query
            statement.execute(createTableSQL);
        } catch (SQLException e) {

            // print SQL exception information
            printSQLException(e);
        }
    }
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-insert-example.html

CRUD no Banco de Dados

Newton
Quem se prepara, não para.

- create (criação), read (leitura), update (atualização) e delete (exclusão)

Fonte: https://www.devmedia.com.br/jdbc-tutorial/6638

CRUD - INSERT

Newton
Quem se prepara, não para.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;


public class InsertRecordExample {
private final String url = "jdbc:postgresql://localhost/BDlivrariaUniversitaria";
private final String user = "postgres";
private final String password = "b0aofmd0";

    private static final String INSERT_USERS_SQL = "INSERT INTO users" +
        "  (id, name, email, country, password) VALUES " +
        " (?, ?, ?, ?, ?);";

    public static void main(String[] argv) throws SQLException {
        InsertRecordExample createTableExample = new InsertRecordExample();
        createTableExample.insertRecord();
    }
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-insert-example.html

CRUD - INSERT

Newton
Quem se prepara, não para.

```
public void insertRecord() throws SQLException {
     System.out.println(INSERT_USERS_SQL);
     // Step 1: Establishing a Connection
     try (Connection connection = DriverManager.getConnection(url, user, password);

         // Step 2:Create a statement using connection object
         PreparedStatement preparedStatement = connection.prepareStatement(INSERT_USERS_SQL)) {
         preparedStatement.setInt(1, 1);
         preparedStatement.setString(2, "Tony");
         preparedStatement.setString(3, "tony@gmail.com");
         preparedStatement.setString(4, "US");
         preparedStatement.setString(5, "secret");

         System.out.println(preparedStatement);
         // Step 3: Execute the query or update query
         preparedStatement.executeUpdate();
     } catch (SQLException e) {

         // print SQL exception information
         printSQLException(e);
     }

     // Step 4: try-with-resource statement will auto close the connection.
 }
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-insert-example.html

CRUD - INSERT

Newton
Quem se prepara, não para.

```
public static void printSQLException(SQLException ex) {
        for (Throwable e: ex) {
            if (e instanceof SQLException) {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " + ((SQLException) e).getSQLState());
                System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
                System.err.println("Message: " + e.getMessage());
                Throwable t = ex.getCause();
                while (t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }
        }
    }
}
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-insert-example.html

CRUD - UPDATE

Newton
Quem se prepara, não para.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;


public class UpdateRecordExample {

private final String url =
"jdbc:postgresql://localhost/BDlivrariaUniversitaria";
private final String user = "postgres";
private final String password = "b0aofmd0";

    private static final String UPDATE_USERS_SQL = "update users set name = ?
where id = ?;";

    public static void main(String[] argv) throws SQLException {
        UpdateRecordExample updateStatementExample = new
UpdateRecordExample();
        updateStatementExample.updateRecord();
    }
```

```
 public void updateRecord() throws SQLException {
        System.out.println(UPDATE_USERS_SQL);
        // Step 1: Establishing a Connection
        try (Connection connection = DriverManager.getConnection(url, user,
password);

            // Step 2:Create a statement using connection object
            PreparedStatement preparedStatement =
connection.prepareStatement(UPDATE_USERS_SQL)) {
            preparedStatement.setString(1, "Ram");
            preparedStatement.setInt(2, 1);

            // Step 3: Execute the query or update query
            preparedStatement.executeUpdate();
        } catch (SQLException e) {

            // print SQL exception information
            printSQLException(e);
        }

        // Step 4: try-with-resource statement will auto close the connection.
    }
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-update-example.html

CRUD - DELETE

Newton
Quem se prepara, não para.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;


public class DeleteRecordExample {

    private static final String DELETE_USERS_SQL = "delete from
users where id = ?;";

private final String url =
"jdbc:postgresql://localhost/BDlivrariaUniversitaria";
private final String user = "postgres";
private final String password = "b0aofmd0";

    public static void main(String[] argv) throws SQLException {
        DeleteRecordExample deleteStatementExample = new
DeleteRecordExample();
        deleteStatementExample.deleteRecord();
    }
```

```
public void deleteRecord() throws SQLException {
        System.out.println(DELETE_USERS_SQL);

        // Step 1: Establishing a Connection
        try (Connection connection = DriverManager.getConnection(url, user,
password);

            // Step 2:Create a statement using connection object
            PreparedStatement preparedStatement =
connection.prepareStatement(DELETE_USERS_SQL);) {
            preparedStatement.setInt(1, 1);

            // Step 3: Execute the query or update query
            int result = preparedStatement.executeUpdate();
            System.out.println("Number of records affected :: " + result);
        } catch (SQLException e) {

            // print SQL exception information
            printSQLException(e);
        }

        // Step 4: try-with-resource statement will auto close the connection.
    }
```

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-delete-example.html

ORM

Newton
Quem se prepara, não para.

Object-relational mapping(ORM) é o nome dado para tecnologias, ferramentas e técnicas usadas para ligar os objetos aos banco de dados relacional. Isso significa que estaremos construindo uma camada extra para persistir os objetos no repositório, ou seja, criaremos uma camada de persistência.

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-delete-example.html

ORM

Newton
Quem se prepara, não para.

Hibernate é um framework ORM(Object-Relational Mapping) e foi, inclusive, a primeira tecnologia para esse propósito em Java. Portanto, sua popularização fez com que a Oracle convidasse seus criadores para determinarem uma interface comum para **frameworks de persistência de dados** que quisessem disponibilizar seus serviços, e essa interface comum é hoje o que conhecemos como JPA.

Ou seja, Hibernate é um framework que implementa o JPA, não se esqueçam disso.

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-delete-example.html

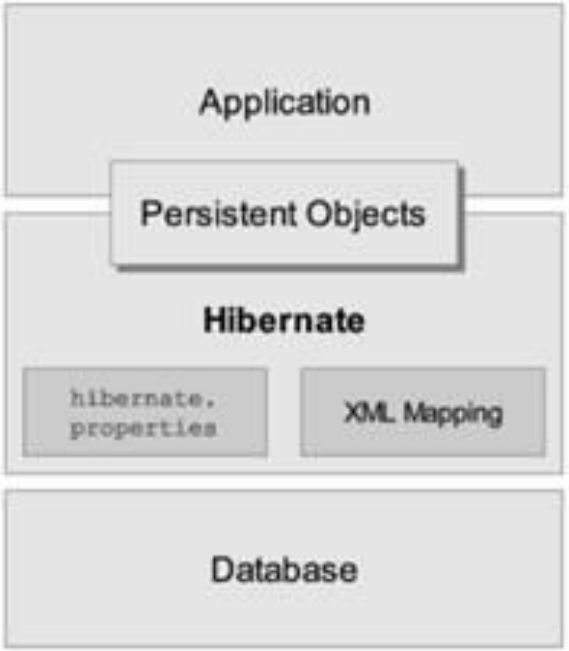
Hibernate

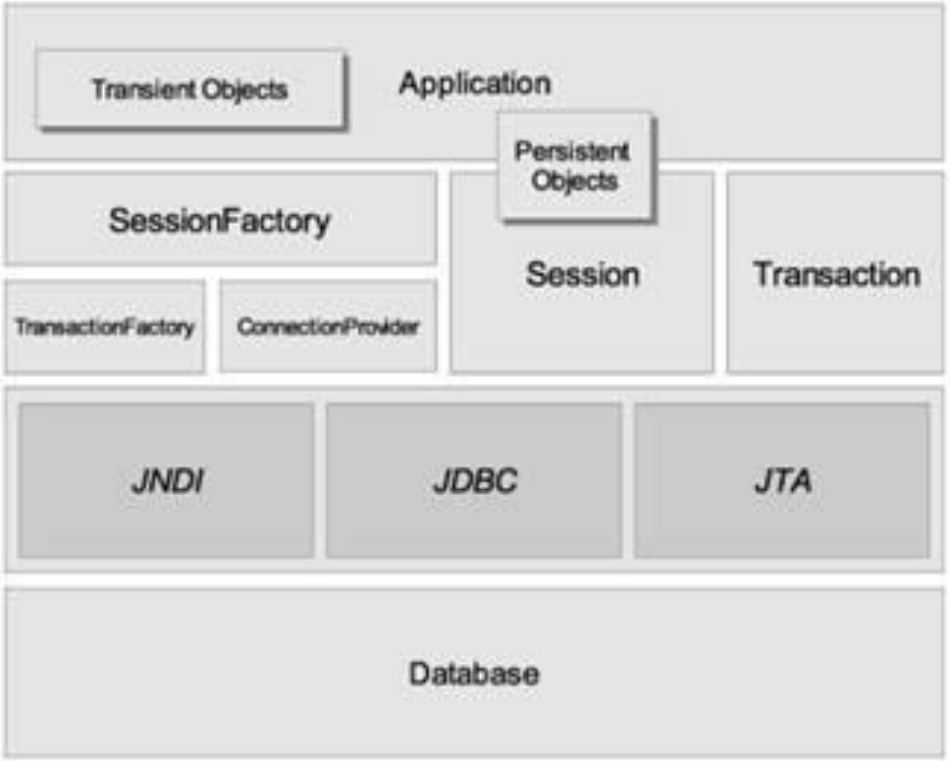
Newton
Quem se prepara, não para.

No Hibernate, o mapeamento entre os objetos e as tabelas pode ser implementado através de arquivos XML, código Java ou via JSR-220 Persistence Annotation. O sucesso do Hibernate está centralizado na simplicidade, onde o coração de toda a interação entre o código e o banco de dados utiliza o Hibernate Session.

Fonte: https://www.javaguides.net/2020/02/java-jdbc-postgresql-delete-example.html

Hibernate





Fonte: https://www.devmedia.com.br/persistencia-de-dados-com-a-tecnologia-orm-hibernate/32840

Hibernate

Newton
Quem se prepara, não para.

É importante que se conheça o conceito das interfaces presentes na camada de persistência do diagrama:

•**Session**: Principal interface usada pelos aplicativos do Hibernate. É onde o Hibernate é iniciado para a realização de operações CRUD, execução de consultas, controle de transações etc. Essa interface é responsável pelo gerenciamento de estados dos objetos;
•**SessionFactory**: Fornece ao aplicativo instâncias da interface Session. Realiza ainda o armazenamento de instruções SQL e metadados de mapeamento utilizados pelo Hibernate em tempo de execução;

Fonte: https://www.devmedia.com.br/persistencia-de-dados-com-a-tecnologia-orm-hibernate/32840

Hibernate

Newton
Quem se prepara, não para.

- **Transaction**: Realiza a tarefa de abstrair o código do aplicativo do usuário que implementaria a transação subjacente. Tal abstração permite o controle dos limites de transações pelo aplicativo, mantendo a portabilidade dos aplicativos do Hibernate mesmo entre ambientes de execução distintos. A interface Transaction é opcional;
- **TransactionFactory**: Fornece instâncias da interface Transaction. A interface é opcional;
- **ConnectionProvider**:Consiste numa fábrica de conexões JDBC, abstraindo DriverManager adjacentes da aplicação. Também é opcional.

Fonte: https://www.devmedia.com.br/persistencia-de-dados-com-a-tecnologia-orm-hibernate/32840

Referências

https://www.javaguides.net/p/java-postgresql-tutorial.html
https://blog.cod3r.com.br/crud-com-jpa-e-hibernate/

https://www.devmedia.com.br/persistindo-objetos-com-java-hibernate-e-postgresql/4149
https://www.devmedia.com.br/persistencia-de-dados-com-a-tecnologia-orm-hibernate/32840