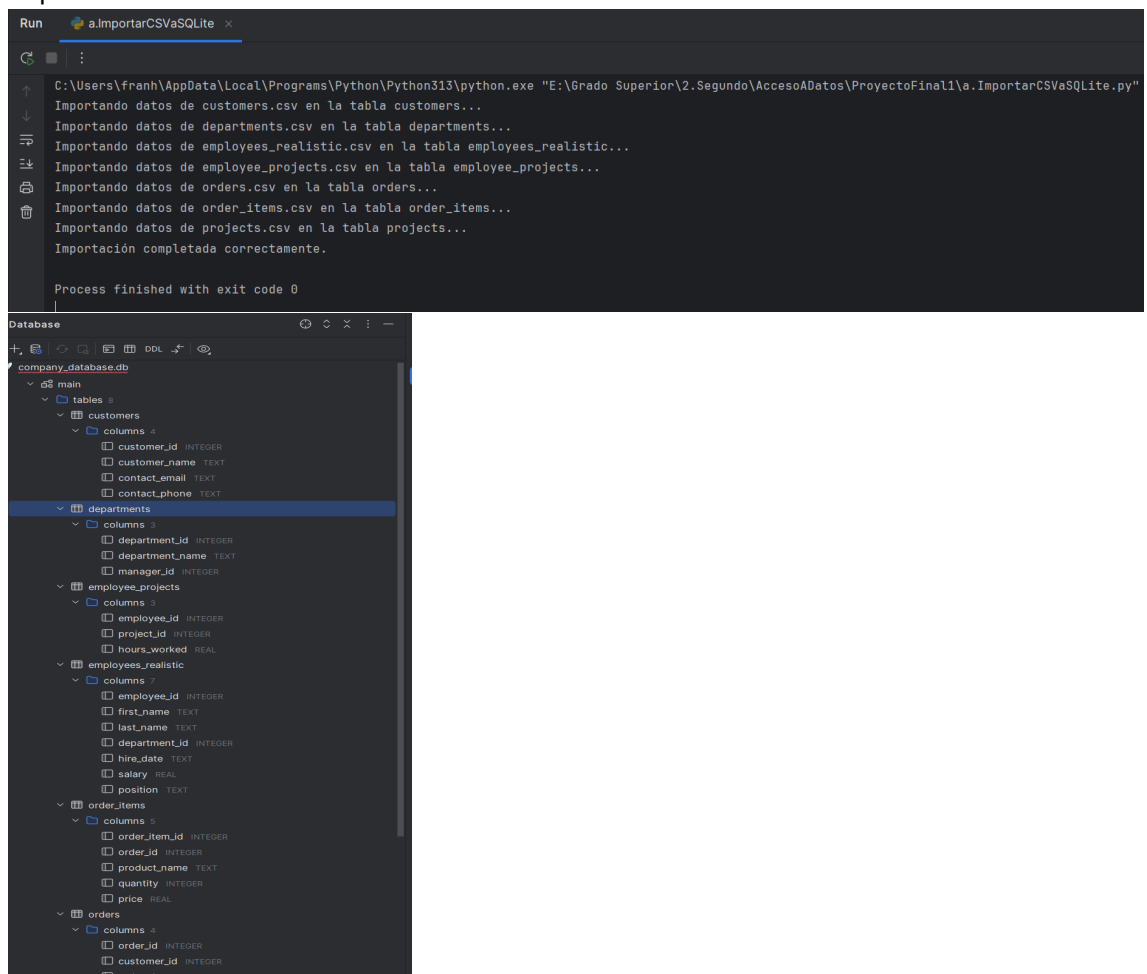


## Funcionalidad:

0. Asumimos que tenemos instalado Python (y está añadido a PATH) y un IDE de Java como IntelliJ IDEA o NetBeans



1. Ejecutamos el archivo "a.ImportarCSVaSQLite.py" ubicado en src/main/csv para crear la base de datos SQLite (company\_database.db, se creará en la ruta del proyecto) e importar los archivos .csv a la base de datos. En caso de que no estén todos los archivos ("customers.csv", "departments.csv", "employees\_realistic.csv", "employee\_projects.csv", "orders.csv", "order\_items.csv", "projects.csv") la importación no se efectuará.



En caso de fallo porque Python no funcione, podemos ejecutar el archivo "ImportCSVtoSQLite.java" y ocurrirá lo mismo.

2. Las consultas pedidas se han realizado en archivos .sql ("b.ConsultarInformacionProyecto.sql", "CalcularCostosSalarialesPorProyecto.sql", "CombinarCostosConPresupuesto.sql" y "e.CalcularFraccionPresupuestoCostosSalariales.sql").
3. En el archivo "SQLiteJDBCFullQueries.java" se encuentran las consultas mencionadas previamente que se ejecutan utilizando JDBC.  
Lo ejecutamos y obtenemos lo siguiente:



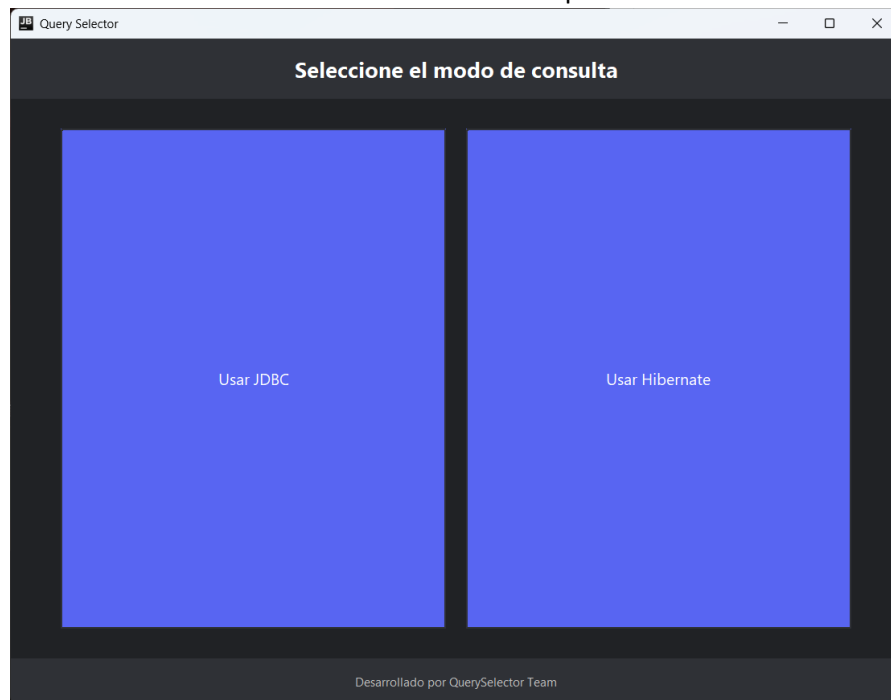
ResultadoSQLiteJDBCFullQueries.java.txt

4. En el archivo "HibernateSQLiteFullQueries.java" se encuentran las consultas mencionadas previamente que se ejecutan utilizando Hibernate.  
Lo ejecutamos y obtenemos lo siguiente:

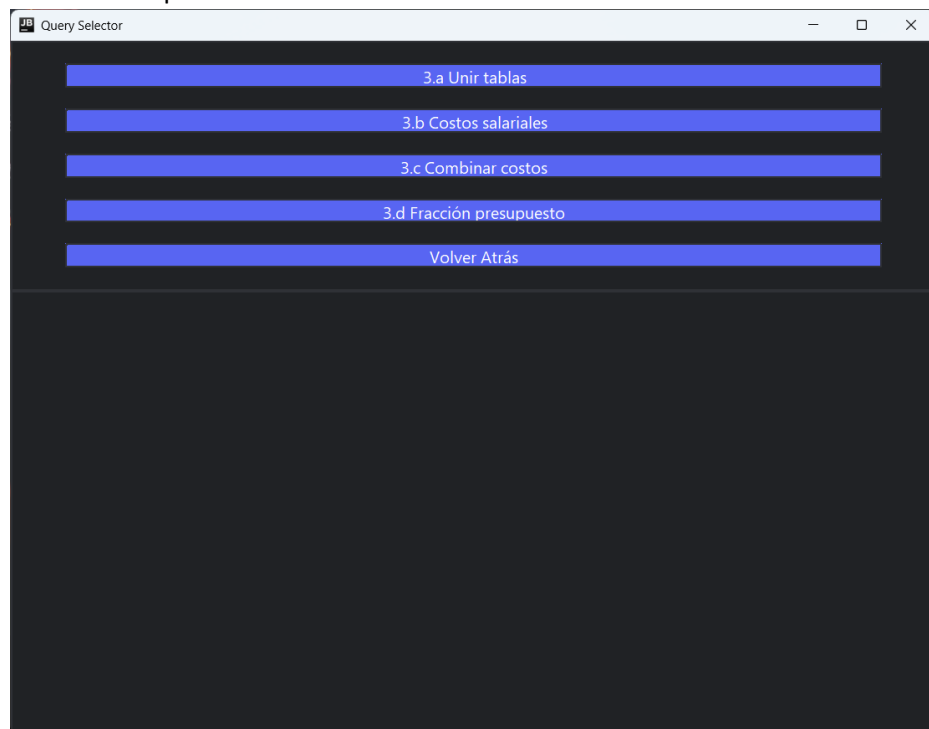


ResultadoHibernateSQLiteFullQueries.java.txt

5. Para mejorar la visualización de los datos, contamos con la clase "QuerySelectorGUI.java", que, al ejecutarla, nos ofrece una GUI de Java Swing donde:
  - a. Podemos seleccionar si usar JDBC o Hibernate para la consulta a realizar:



- b. Independientemente de cuál seleccionemos, nos mostrará la siguiente lista de consultas disponibles:




- c. Seleccionaremos la 3.d: Primero en JDBC, y luego en Hibernate  
En JDBC se ven los nombres de las columnas:

The screenshot shows the same "Query Selector" window, but now it displays a table of data below the buttons. The table has four columns: "project\_id", "budget", "project\_salary\_costs", and "cost\_fraction". The data is as follows:

project_id	budget	project_salary_costs	cost_fraction
1	25597.9	247.46903157894738	0.9667552087434803
2	73139.01	53.654963157894734	0.07336025351983126
3	46416.0	167.6296789473684	0.3611463265842994
5	87841.22	237.29714736842104	0.27014327370273433
6	77580.34	70.35274736842105	0.09068373168823578
7	44025.78	89.62434736842104	0.2035724236309295
8	44742.77	173.89177368421053	0.3886477607090722
9	52671.43	103.9227157894737	0.19730376750635722

En Hibernate no se ven los nombres de las columnas, puesto que hibernate trabaja con objetos y no tiene un método de acceder al nombre de las columnas:



The screenshot shows a window titled 'Query Selector'. It has a menu with five items: '3.a Unir tablas', '3.b Costos salariales', '3.c Combinar costos', '3.d Fracción presupuesto', and 'Volver Atrás'. Below the menu is a table with four columns labeled 'Column 1', 'Column 2', 'Column 3', and 'Column 4'. The table contains 9 rows of numerical data.

Column 1	Column 2	Column 3	Column 4
1	25597.9	247.46903157894738	0.9667552087434803
2	73139.01	53.654963157894734	0.07336025351983126
3	46416.0	167.6296789473684	0.3611463265842994
5	87841.22	237.29714736842104	0.27014327370273433
6	77580.34	70.35274736842105	0.09068373168823578
7	44025.78	89.62434736842104	0.2035724236309295
8	44742.77	173.89177368421053	0.3886477607090722
9	52671.43	103.9227157894737	0.19730376750635722

## Documentación del código

### Importación de Datos

#### a. ImportarCSVaSQLite.py

Este script en Python utiliza pandas para leer archivos CSV y sqlite3 para insertar los datos en una base de datos SQLite.

```
# Python script para importar datos CSV a SQLite
import sqlite3
import pandas as pd
import os

# Ruta relativa para la base de datos
db_path = os.path.join(".", "..", "..", "..", "company_database.db")

# Verificar si los archivos CSV existen en el directorio actual
csv_files = ["customers.csv", "departments.csv", "employees_realistic.csv", "employee_projects.csv", "orders.csv", "order_items.csv", "projects.csv"]
missing_files = [f for f in csv_files if not os.path.exists(f)]

if missing_files:
    print(f"Faltan los siguientes archivos CSV: {', '.join(missing_files)}")
    exit(1)

# Crear la base de datos SQLite en la ruta relativa especificada
connection = sqlite3.connect(db_path)
cursor = connection.cursor()

# Diccionario de archivos y nombres de tablas
tables = {
    "customers.csv": "customers",
    "departments.csv": "departments",
    "employees_realistic.csv": "employees_realistic",
    "employee_projects.csv": "employee_projects",
    "orders.csv": "orders",
    "order_items.csv": "order_items",
    "projects.csv": "projects",
}

# Importar datos a SQLite
for file, table in tables.items():
    print(f"Importando datos de {file} en la tabla {table}...")
    df = pd.read_csv(file)
    df.to_sql(table, connection, if_exists="replace", index=False)

print("Importación completada correctamente.")
connection.close()
```

## b. ImportCSVtoSQLite.java

Esta clase Java lee archivos CSV desde un directorio específico y los inserta en la base de datos SQLite utilizando JDBC.

Captura del main:

```
public class ImportCSVtoSQLite {
    public static void main(String[] args) {
        // Usar rutas relativas
        String baseDir = "src/main"; // Ruta base para el proyecto (relativa al directorio principal)
        String csvDir = "csv"; // Subdirectorio donde están los archivos CSV
        String dbDir = "ProyectoFinal"; // Carpeta donde se encuentra la base de datos

        // Ruta relativa de la base de datos
        Path dbPath = Paths.get(baseDir, dbDir, "company_database.db");

        // Archivos CSV y sus tablas correspondientes
        Map<String, String> csvFiles = new HashMap<>();
        csvFiles.put("customers.csv", "customers");
        csvFiles.put("departments.csv", "departments");
        csvFiles.put("employees_realistic.csv", "employees_realistic");
        csvFiles.put("employee_projects.csv", "employee_projects");
        csvFiles.put("orders.csv", "orders");
        csvFiles.put("order_items.csv", "order_items");
        csvFiles.put("projects.csv", "projects");

        // Conexión a la base de datos SQLite
        try (Connection conn = DriverManager.getConnection("jdbc:sqlite:" + dbPath.toString())) {
            Statement stmt = conn.createStatement();
```

Captura de la sentencia SQL para inserción:

```
            try (PreparedStatement pstmt = conn.prepareStatement(sql.toString())) {
                // Leer cada línea de datos y preparar para inserción
                while ((line = br.readLine()) != null) {
                    String[] values = line.split(",");
                    for (int i = 0; i < values.length; i++) {
                        pstmt.setString(i + 1, values[i]);
                    }
                    pstmt.executeUpdate(); // Ejecutar la inserción
                }
            } catch (IOException e) {
                System.err.println("Error leyendo el archivo CSV " + csvFile + ": " + e.getMessage());
            }
        }

        System.out.println("Importación completada correctamente.");
    } catch (SQLException e) {
        System.err.println("Error con la base de datos: " + e.getMessage());
    }
}
```

## Configuración de Maven

### pom.xml

Este archivo contiene las configuraciones y dependencias necesarias para el proyecto, incluyendo Hibernate y SQLite JDBC Driver.

Captura del bloque <dependencies> de pom.xml:

```
<dependencies>
  <!-- Hibernate Core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.4.1.Final</version>
  </dependency>

  <!-- SQLite JDBC Driver -->
  <dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.47.1.0</version>
  </dependency>

  <!-- Jakarta Persistence (para anotaciones) -->
  <dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.1.0</version>
  </dependency>

  <!-- Log4j para logs (opcional) -->
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
  </dependency>

  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
  </dependency>

  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-csv</artifactId>
    <version>1.8</version>
  </dependency>
</dependencies>
```

Captura del bloque <repositories> de pom.xml:

```
<repositories>
  <repository>
    <id>central</id>
    <url>https://repo.maven.apache.org/maven2</url>
  </repository>
  <repository>
    <id>jboss</id>
    <url>https://repository.jboss.org/nexus/content/repositories/releases</url>
  </repository>
</repositories>
```

## Configuración de Hibernate:

### hibernate.cfg.xml

Este archivo XML configura Hibernate para conectar con la base de datos SQLite y mapear las clases en entidades/objetos.

Captura del bloque <session-factory> de hibernate.cfg.xml: y captura de las etiquetas

<mapping class="..." />:

```
<session-factory>
  <property name="hibernate.hbm2ddl.auto">update</property> <!-- Cambié 'create' a 'update' para no eliminar los datos -->
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property> <!-- Dialecto de SQLite (usamos el de MySQL) -->
  <property name="hibernate.connection.driver_class">org.sqlite.JDBC</property> <!-- Driver de SQLite -->
  <property name="hibernate.connection.url">jdbc:sqlite:company_database.db</property> <!-- Ruta a la base de datos SQLite -->
  <property name="hibernate.connection.username"></property> <!-- SQLite no necesita usuario -->
  <property name="hibernate.connection.password"></property> <!-- SQLite no necesita contraseña -->
  <property name="hibernate.show_sql">>true</property> <!-- Mostrar las consultas SQL generadas -->
  <property name="hibernate.format_sql">>true</property> <!-- Formatear las consultas SQL para mejor lectura -->
  <mapping class="org.example.EmployeeProject"/>
  <mapping class="org.example.EmployeeRealistic"/>
  <mapping class="org.example.Project"/>
</session-factory>
```

## Clases de Hibernate

### SessionFactoryProvider.java

Esta clase proporciona una instancia única de SessionFactory.

Captura del método provideSessionFactory() de SessionFactoryProvider.java:

```
public class SessionFactoryProvider { 2 usages
    public static SessionFactory provideSessionFactory() 1 usage
    {
        Configuration config = new Configuration();
        config.configure();
        return config.buildSessionFactory();
    }
}
```

## Consultas con JDBC y Hibernate

### SQLiteJDBCFullQueries.java

Este script Java contiene consultas SQL directas a la base de datos SQLite para unir tablas, calcular costos salariales, combinar costos con presupuesto y calcular fracciones del presupuesto.

Captura del bloque try donde se ejecutan las consultas (de una consulta específica joinQuery):

```
public class SQLiteJDBCFullQueries {

    public static void main(String[] args) {
        String url = "jdbc:sqlite:company_database.db";

        try (Connection conn = DriverManager.getConnection(url)) {
            System.out.println("Conexión establecida con SQLite.");
            Statement stmt = conn.createStatement();

            // 3.a Unir tablas
            String joinQuery = ""
                + "SELECT ep.project_id, ep.employee_id, er.salary"
                + "FROM employee_projects AS ep"
                + "JOIN employees_realistic AS er"
                + "ON ep.employee_id = er.employee_id;"
                + "";

            ResultSet rsJoin = stmt.executeQuery(joinQuery);
            System.out.println("3.a Resultados de unir tablas:");
            while (rsJoin.next()) {
                System.out.printf("Project ID: %d, Employee ID: %d, Salary: %.2f%n",
                    rsJoin.getInt( "columnLabel: \"project_id\"", rsJoin.getInt( "columnLabel: \"employee_id\"", rsJoin.getDouble( "columnLabel: \"salary\""));
            }
        }
    }
}
```



### HibernateSQLiteFullQueries.java

Esta clase Java utiliza Hibernate para realizar consultas similares a las realizadas con JDBC.

Captura de la definición de las entidades (EmployeeProject, EmployeeRealistic, Project):

```
@Entity
@Table(name = "employee_projects")
class EmployeeProject {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne 6 usages
    @JoinColumn(name = "project_id", referencedColumnName = "project_id", insertable = false, updatable = false)
    private Project project;

    @ManyToOne 7 usages
    @JoinColumn(name = "employee_id", referencedColumnName = "employee_id", insertable = false, updatable = false)
    private EmployeeRealistic employeeRealistic;
```

```
@Entity
@Table(name = "employees_realistic")
class EmployeeRealistic {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "employee_id") 2 usages
    private int employeeId;

    @Column(name = "salary") 7 usages
    private double salary;

    @OneToMany(mappedBy = "employeeRealistic") 2 usages
    private List<EmployeeProject> employeeProjects;
```

```
@Entity
@Table(name = "projects")
class Project {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "project_id")
    private int projectId;

    @Column(name = "budget") 7 usages
    private double budget;

    @OneToMany(mappedBy = "project") 4 usages
    private List<EmployeeProject> employeeProjects;
```

Captura del método unirTablas(), que realiza la primera consulta (3.a) utilizando Hibernate:

```
public class HibernateSQLiteFullQueries {

    private final SessionFactory factory; 5 usages

    public HibernateSQLiteFullQueries() { 2 usages
        this.factory = new SessionFactoryProvider().provideSessionFactory();
    }

    public List<Object[]> unirTablas() { 2 usages
        try (Session session = factory.openSession()) {
            String hql = ""
                SELECT ep.project.projectId, er.salary
                FROM EmployeeProject ep
                JOIN ep.employeeRealistic er
                "";
            return session.createQuery(hql, Object[].class).getResultList();
        }
    }
}
```



## Interfaz Gráfica (GUI)

### QuerySelectorGUI.java

Esta clase Java crea una interfaz gráfica utilizando Java Swing que permite al usuario seleccionar el modo de consulta (JDBC o Hibernate) y ejecutar diferentes consultas.

Captura del método createAndShowGUI():

```
public class QuerySelectorGUI {  
    private static final String JDBC_URL = "jdbc:sqlite:company_database.db"; // usage  
  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(() -> new QuerySelectorGUI().createAndShowGUI());  
    }  
  
    private void createAndShowGUI() { // 2 usages  
        JFrame frame = new JFrame( "Query Selector");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setSize( 900, 700);  
  
        JPanel mainPanel = new JPanel();  
        mainPanel.setLayout(new BorderLayout());  
        mainPanel.setBackground(new Color( r: 32, g: 34, b: 37));  
  
        // Header Panel  
        JPanel headerPanel = new JPanel();  
        headerPanel.setBackground(new Color( r: 47, g: 49, b: 54));  
        headerPanel.setBorder(BorderFactory.createEmptyBorder( top: 10, left: 10, bottom: 10, right: 10));  
  
        JLabel titleLabel = new JLabel( text: "Seleccione el modo de consulta");  
        titleLabel.setFont(new Font( name: "Segoe UI", Font.BOLD, size: 22));  
        titleLabel.setForeground(Color.WHITE);  
        headerPanel.add(titleLabel);  
  
        // Button Panel  
        JPanel buttonPanel = new JPanel();  
        buttonPanel.setLayout(new GridLayout( rows: 1, cols: 2, hgap: 20, vgap: 0));  
        buttonPanel.setBackground(new Color( r: 32, g: 34, b: 37));  
        buttonPanel.setBorder(BorderFactory.createEmptyBorder( top: 30, left: 50, bottom: 30, right: 50));  
  
        JButton jdbcButton = new JButton( text: "Usar JDBC");  
        JButton hibernateButton = new JButton( text: "Usar Hibernate");  
  
        styleButton(jdbcButton);  
        styleButton(hibernateButton);  
  
        buttonPanel.add(jdbcButton);  
        buttonPanel.add(hibernateButton);  
  
        // Footer Panel  
        JPanel footerPanel = new JPanel();  
        footerPanel.setBackground(new Color( r: 47, g: 49, b: 54));  
        footerPanel.setBorder(BorderFactory.createEmptyBorder( top: 10, left: 10, bottom: 10, right: 10));  
  
        JLabel footerLabel = new JLabel( text: "Desarrollado por QuerySelector Team");  
        footerLabel.setFont(new Font( name: "Segoe UI", Font.PLAIN, size: 12));  
        footerLabel.setForeground(Color.LIGHT_GRAY);  
        footerPanel.add(footerLabel);  
  
        mainPanel.add(headerPanel, BorderLayout.NORTH);  
        mainPanel.add(buttonPanel, BorderLayout.CENTER);  
        mainPanel.add(footerPanel, BorderLayout.SOUTH);  
  
        frame.add(mainPanel);  
  
        jdbcButton.addActionListener( ActionEvent e -> showQueryPanel(frame, mode: "JDBC"));  
        hibernateButton.addActionListener( ActionEvent e -> showQueryPanel(frame, mode: "Hibernate"));  
  
        frame.setVisible(true);  
    }  
}
```

Captura del método showQueryPanel(JFrame frame, String mode):

```
public class QuerySelectorGUI {  
    private void showQueryPanel(JFrame frame, String mode) { 2 usages  
        frame.getContentPane().removeAll();  
  
        JPanel queryPanel = new JPanel();  
        queryPanel.setLayout(new GridLayout( rows: 5, cols: 1, hgap: 10, vgap: 20)); // Mayor separación entre filas  
        queryPanel.setBackground(new Color( r: 32, g: 34, b: 37));  
        queryPanel.setBorder(BorderFactory.createCompoundBorder(  
            BorderFactory.createLineBorder(new Color( r: 47, g: 49, b: 54), thickness: 2),  
            BorderFactory.createEmptyBorder( top: 20, left: 50, bottom: 20, right: 50)));  
  
        JButton joinButton = new JButton( text: "3.a Unir tablas");  
        JButton salaryCostsButton = new JButton( text: "3.b Costos salariales");  
        JButton combineCostsButton = new JButton( text: "3.c Combinar costos");  
        JButton fractionButton = new JButton( text: "3.d Fracción presupuesto");  
        JButton backButton = new JButton( text: "Volver Atrás");  
  
        styleButton(joinButton);  
        styleButton(salaryCostsButton);  
        styleButton(combineCostsButton);  
        styleButton(fractionButton);  
        styleButton(backButton);  
  
        queryPanel.add(joinButton);  
        queryPanel.add(salaryCostsButton);  
        queryPanel.add(combineCostsButton);  
        queryPanel.add(fractionButton);  
        queryPanel.add(backButton);  
  
        JPanel tablePanel = new JPanel();  
        tablePanel.setLayout(new BorderLayout());  
        tablePanel.setBackground(new Color( r: 32, g: 34, b: 37));  
        tablePanel.setBorder(BorderFactory.createCompoundBorder(  
            BorderFactory.createLineBorder(new Color( r: 47, g: 49, b: 54), thickness: 2),  
            BorderFactory.createEmptyBorder( top: 10, left: 10, bottom: 10, right: 10)));  
        tablePanel.revalidate();  
        tablePanel.repaint();  
  
        DefaultTableModel tableModel = new DefaultTableModel();  
        JTable resultTable = new JTable(tableModel);  
        resultTable.setFont(new Font( name: "Segoe UI", Font.PLAIN, size: 14));  
        resultTable.setForeground(Color.WHITE);  
        resultTable.setBackground(new Color( r: 47, g: 49, b: 54)); // Fondo de la tabla  
        resultTable.setGridColor(new Color( r: 88, g: 101, b: 242));  
        resultTable.setRowHeight(50); // Aumento del tamaño de las filas  
  
        // Cambiar fondo del JScrollPane para que coincida  
        JScrollPane scrollPane = new JScrollPane(resultTable);  
        scrollPane.setBackground(new Color( r: 32, g: 34, b: 37)); // Fondo del scrollPane  
        scrollPane.getViewport().setBackground(new Color( r: 32, g: 34, b: 37)); // Fondo del área visible  
        scrollPane.setBorder(BorderFactory.createEmptyBorder());  
        tablePanel.add(scrollPane, BorderLayout.CENTER);  
  
        frame.add(queryPanel, BorderLayout.CENTER);  
        frame.add(tablePanel, BorderLayout.SOUTH);  
  
        joinButton.addActionListener( ActionEvent e -> executeQuery(mode, queryType: "join", tableModel));  
        salaryCostsButton.addActionListener( ActionEvent e -> executeQuery(mode, queryType: "salaryCosts", tableModel));  
        combineCostsButton.addActionListener( ActionEvent e -> executeQuery(mode, queryType: "combineCosts", tableModel));  
        fractionButton.addActionListener( ActionEvent e -> executeQuery(mode, queryType: "fraction", tableModel));  
        backButton.addActionListener( ActionEvent e -> {  
            createAndShowGUI();  
            frame.dispose();  
        });  
    }  
}
```

## Captura de los métodos para ejecutar Query (con JDBC o Hibernate):

```
private void executeQuery(String mode, String queryType, DefaultTableModel tableModel) { // 4 usages
    if (mode.equals("JDBC")) {
        executeJDBCQuery(queryType, tableModel);
    } else if (mode.equals("Hibernate")) {
        executeHibernateQuery(queryType, tableModel);
    }
}

private void executeJDBCQuery(String queryType, DefaultTableModel tableModel) { // 1 usage
    String query = switch (queryType) {
        case "join" -> "SELECT ep.project-id, ep.employee-id, er.salary FROM employee-projects AS ep JOIN employees-realistic AS er ON ep.employee-id = er.employee-id";
        case "salaryCosts" -> "SELECT ep.project-id, SUM(er.salary / 1200) AS project-salary-costs FROM employee-projects AS ep JOIN employees-realistic AS er ON ep.employee-id = er.employee-id GROUP BY ep.project-id";
        case "combineCosts" -> "SELECT p.project-id, p.budget, ps.project-salary-costs FROM projects AS p JOIN ( SELECT ep.project-id, SUM(er.salary / 1200) AS project-salary-costs FROM employee-projects AS ep JOIN employees-realistic AS er ON ep.employee-id = er.employee-id GROUP BY ep.project-id ) AS ps ON p.project-id = ps.project-id";
        case "fraction" -> "SELECT p.project-id, p.budget, ps.project-salary-costs, (ps.project-salary-costs / p.budget) * 100 AS cost-fraction FROM projects AS p JOIN ( SELECT ep.project-id, SUM(er.salary / 1200) AS project-salary-costs FROM employee-projects AS ep JOIN employees-realistic AS er ON ep.employee-id = er.employee-id GROUP BY ep.project-id ) AS ps ON p.project-id = ps.project-id";
        default -> "";
    };

    try {
        Connection conn = DriverManager.getConnection(JDBC_URL);
        Statement stat = conn.createStatement();
        ResultSet rs = stat.executeQuery(query);
        updateTableModel(tableModel, rs);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void executeHibernateQuery(String queryType, DefaultTableModel tableModel) { // 1 usage
    try {
        SessionFactory factory = new Configuration().configure("resources/hibernate.cfg.xml").buildSessionFactory();
        Session session = factory.openSession();
        HibernateSQLiteDatabase queries = new HibernateSQLiteDatabase();
        List<Object[]> results = switch (queryType) {
            case "join" -> queries.unirTablas();
            case "salaryCosts" -> queries.calcularCostos();
            case "combineCosts" -> queries.combinarCostosConPresupuesto();
            case "fraction" -> queries.calcularFraccionPresupuesto();
            default -> List.of();
        };

        updateTableModelHibernate(tableModel, results);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```