HIBERNATE

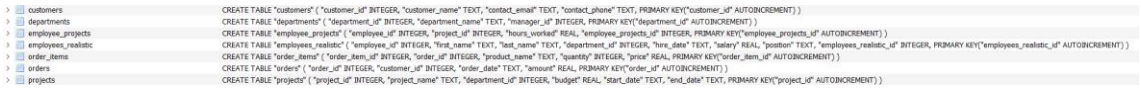
ACCESO A DATOS

Alvaro Castro
LAS NAVES SALESIANOS 2DAM

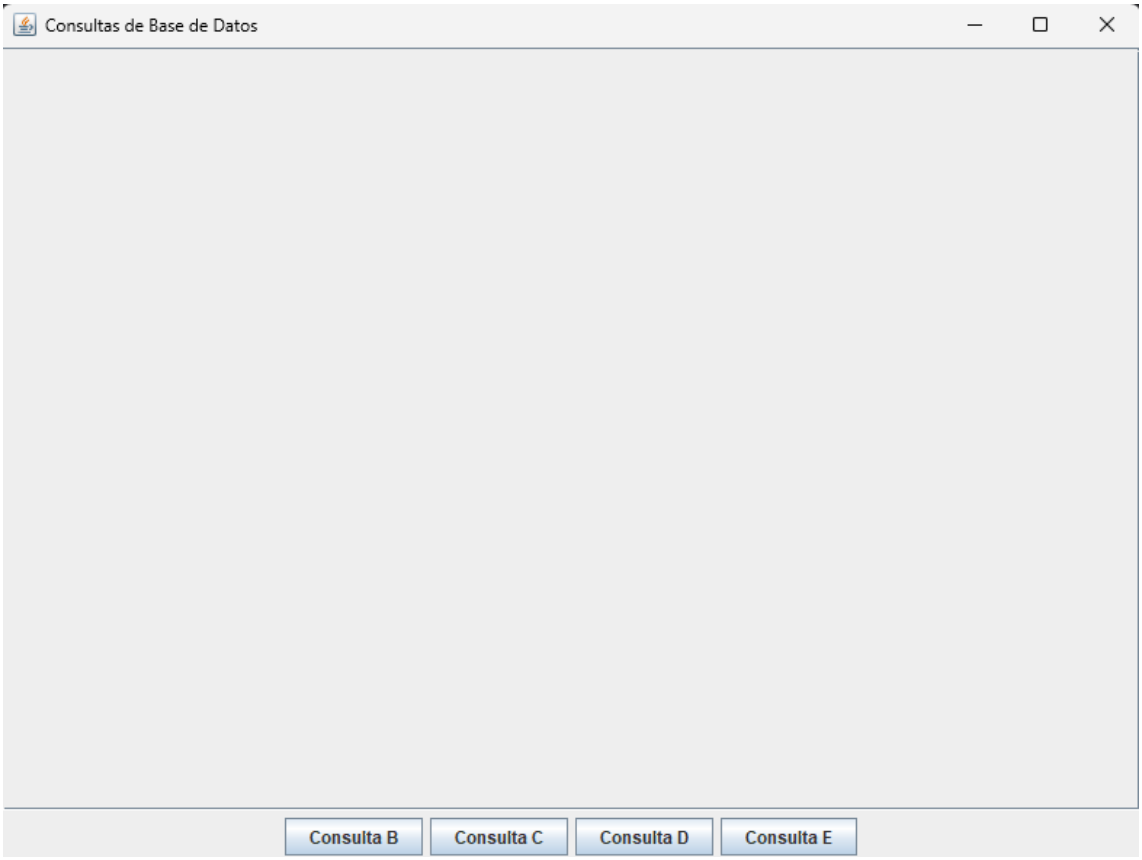
En este documento voy a explicar el funcionamiento de mi programa.

Lo primero es avisar que he modificado las tablas que nos has pasado, para a la hora de pasar los datos del archivo SQLite, nos sea más fácil usarlos.

Las tablas se quedarían tal que así.



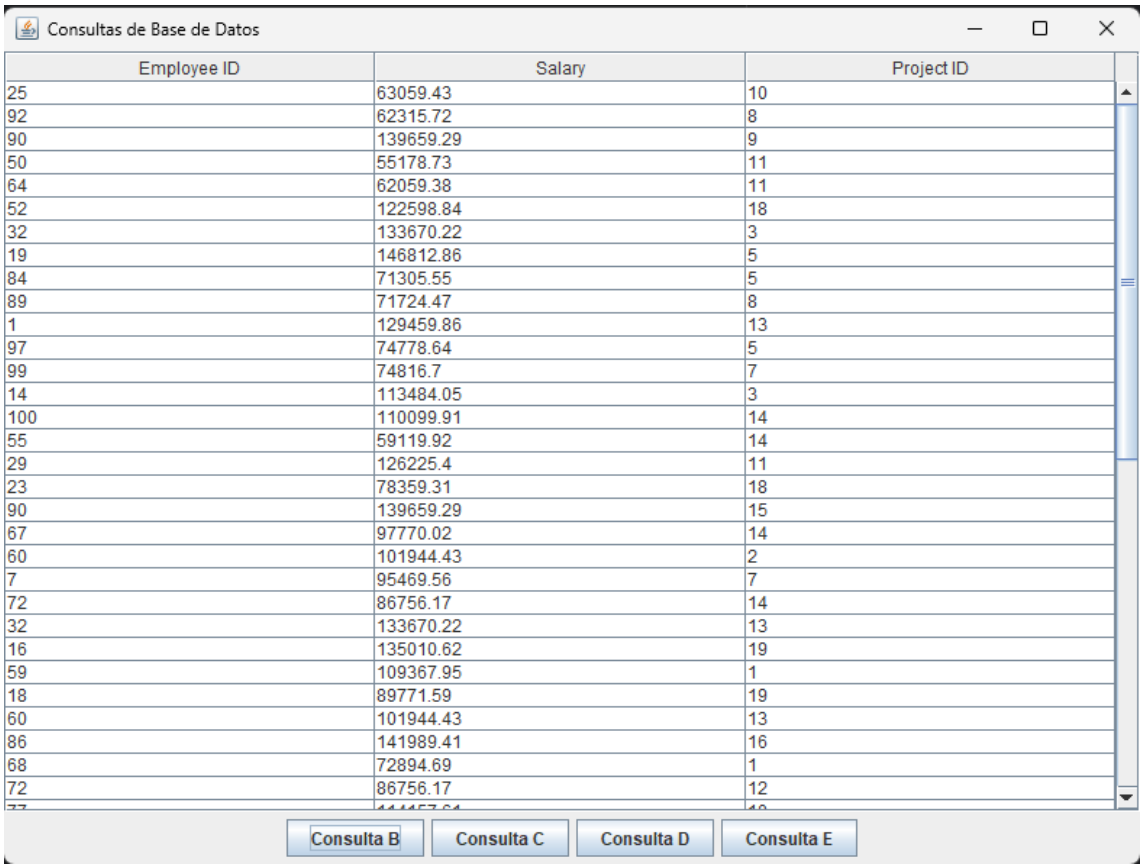
Ahora mostraré la interfaz que he hecho para mostrar los datos.



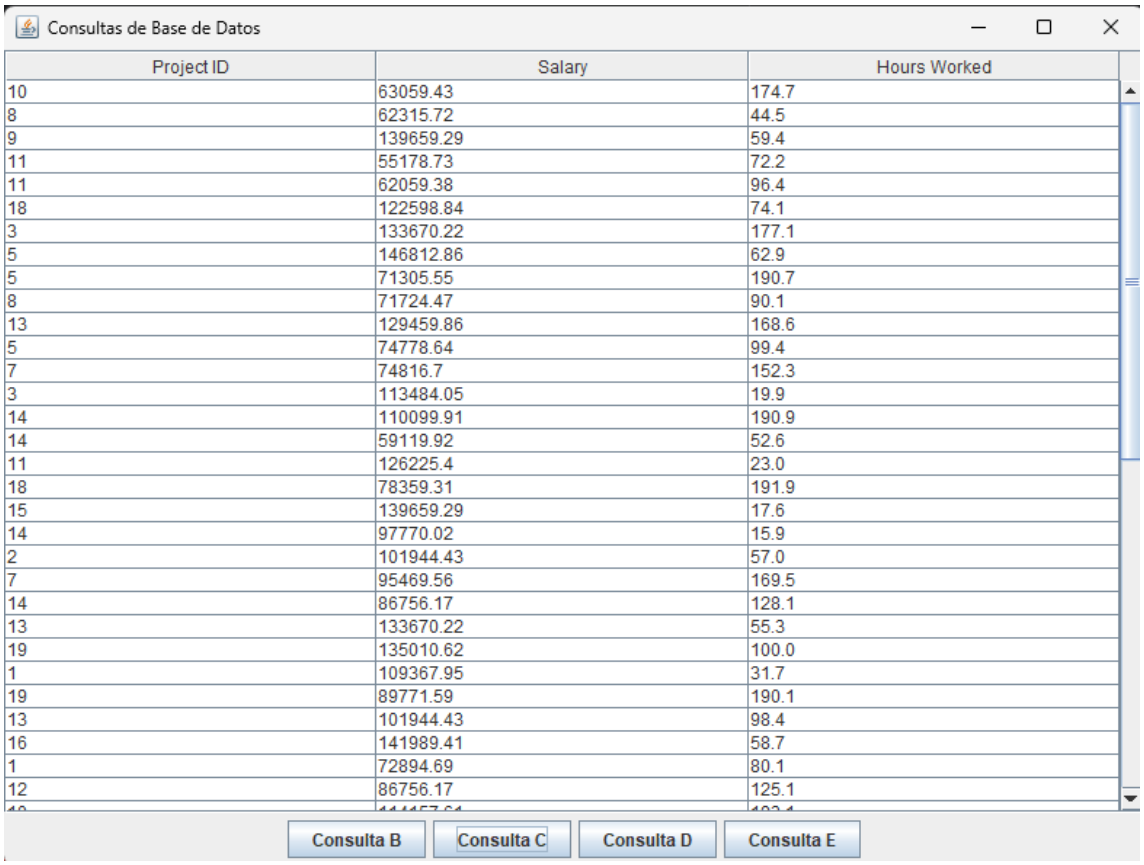
Es una interfaz muy simple. No me he centrado en el diseño, si no en la funcionalidad.

Cuando le damos a los botones de consulta. Se conectará al archivo SQLite y mostrará los datos de cada consulta en una tabla.

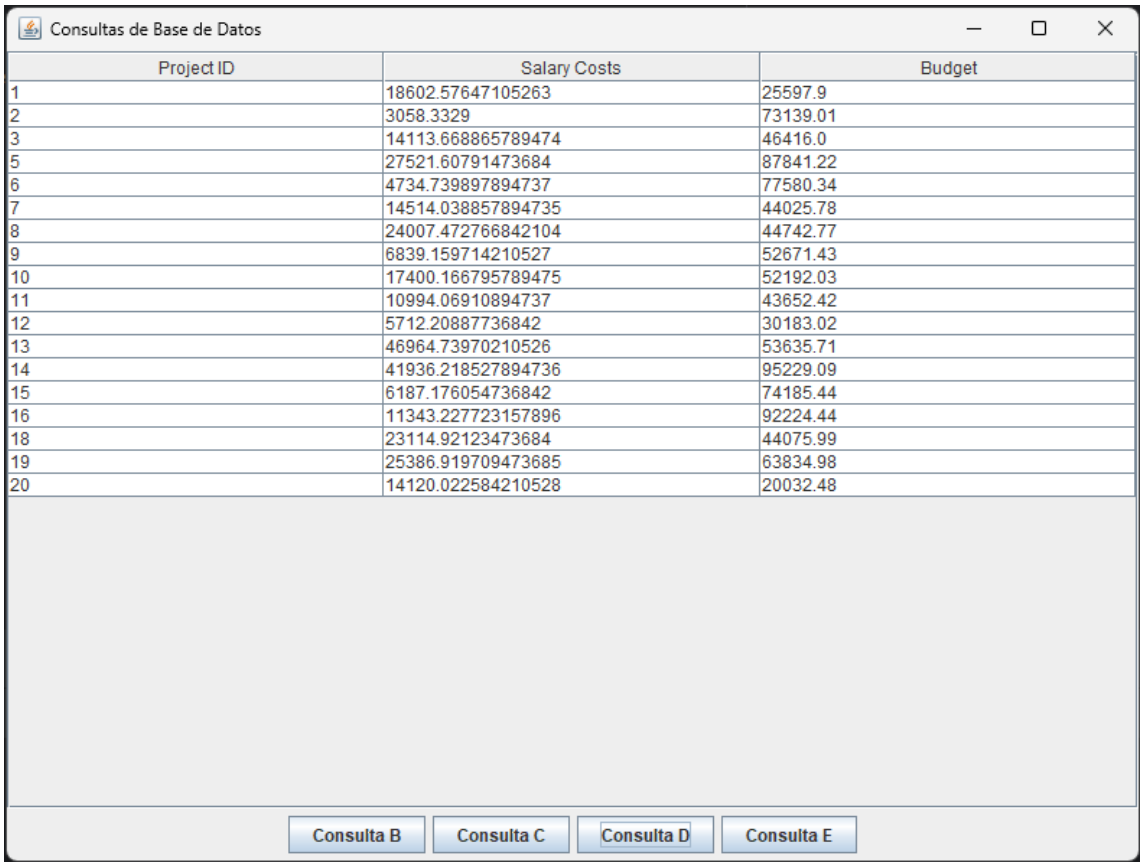
Consulta B:



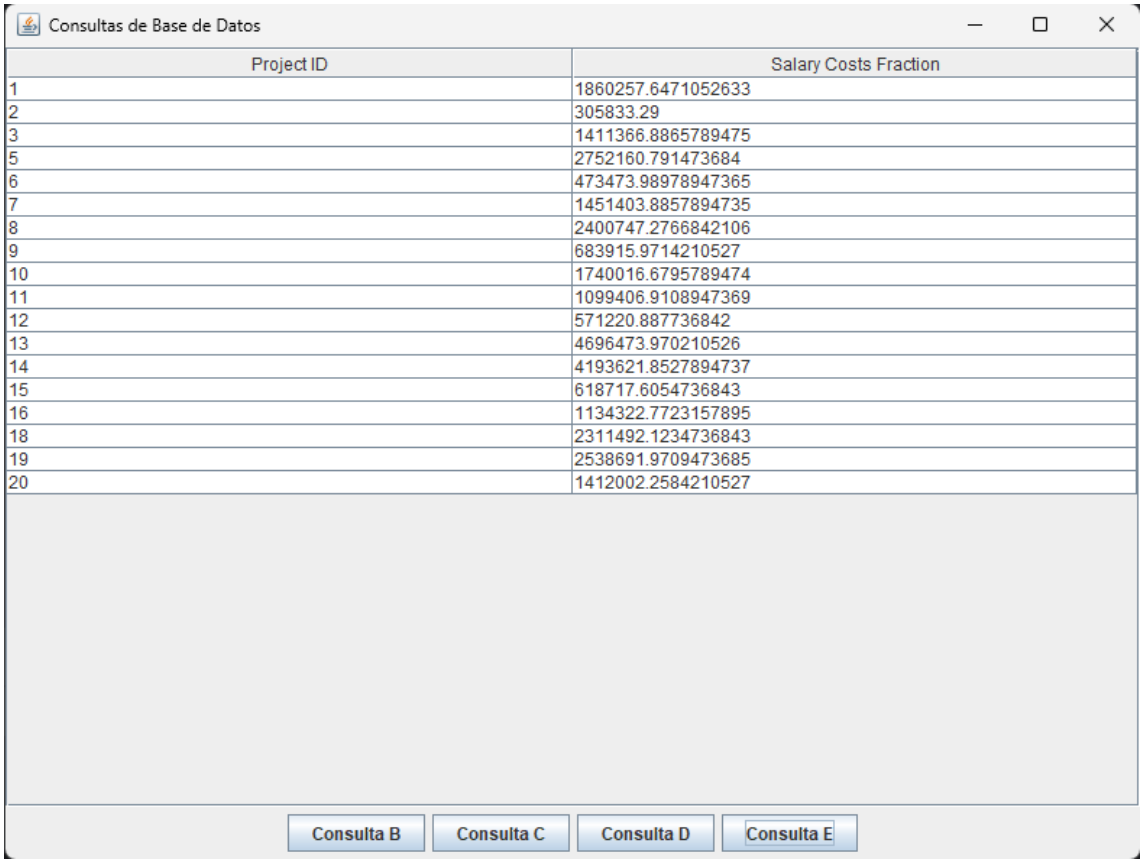
Consulta C:



Consulta D:



Consulta E:



Ahora voy a explicar un poco de codigo.

Primero he creado una clase que solo sirve para hacer la conexión al fichero SQLite.

```java
package org.example;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

3 usages
public class ConexionBBDD {
    2 usages
    private static final SessionFactory sessionFactory;

    static {
        try {
            // Crear la SessionFactory desde la configuración de hibernate.cfg.xml
            sessionFactory = new Configuration().configure( resource: "hibernate.cfg.xml").addAnnotatedClass(EmployeeProject.class)
                    .addAnnotatedClass(Project.class)
                    .addAnnotatedClass(OrderItem.class)
                    .addAnnotatedClass(EmployeeRealistic.class)
                    .addAnnotatedClass(Department.class)
                    .addAnnotatedClass(Customer.class)
                    .addAnnotatedClass(Order.class).buildSessionFactory();
        } catch (Exception e) {
            throw new ExceptionInInitializerError( s: "Initial SessionFactory creation failed." + e);
        }
    }

    4 usages
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    no usages
    public static void shutdown() {
        // Cerrar caché de Hibernate y conexiones a la base de datos
        getSessionFactory().close();
    }
```

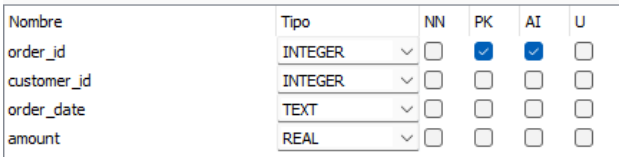
Y le añado las clases que hacen referencia a tablas para aplicar hibernate.

Luego tengo 7 clases distintas que hacen referencia a las tablas, como he explicado antes.

Cad clase contiene una variable por columna, con el tipo de variable equivalente.

Un ejemplo:

```java
@Entity
@Table(name = "orders")
public class Order {

    2 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "order_id")
    private Long orderId;
    2 usages
    @Column(name = "customer_id")
    private int customerId;
    2 usages
    @Column(name = "order_date")
    private String orderDate;
    2 usages
    @Column(name = "amount")
    private Double amount;

    // Getters and Setters
    no usages
    public Long getOrderId() { return orderId; }

    no usages
    public void setOrderId(Long orderId) { this.orderId = orderId; }

    no usages
    public int getCustomerId() { return customerId; }

    no usages
    public void setCustomerId(int customerId) { this.customerId = customerId; }

    no usages
    public String getOrderDate() { return orderDate; }

    no usages
    public void setOrderDate(String orderDate) { this.orderDate = orderDate; }

    no usages
    public Double getAmount() { return amount; }

    no usages
    public void setAmount(Double amount) { this.amount = amount; }
}
```



Nombre	Tipo	NN	PK	AI	U
order_id	INTEGER	☐	☑	☑	☐
customer_id	INTEGER	☐	☐	☐	☐
order_date	TEXT	☐	☐	☐	☐
amount	REAL	☐	☐	☐	☐

He hecho otra clase llamada Consultas, en la que hay un metodo por apartado, y cada metodo devuelve la lista de objetos, y cada objeto contiene el resultado de la consulta fila por fila para poder usarlo en el JFrame de la interfaz.

Consulta B:

```
public List<Object[]> consultab() {
    Session session = ConexionBBDD.getSessionFactory().openSession();
    session.beginTransaction();

    String hql = "SELECT ep.employeeId, er.salary, ep.projectId "
            + "FROM EmployeeProject ep "
            + "JOIN EmployeeRealistic er ON ep.employeeId = er.employeeId";

    Query<Object[]> query = session.createQuery(hql, Object[].class);
    List<Object[]> results = query.getResultList();

    for (Object[] row : results) {
        int employeeId = (int) row[0];
        Double salary = (Double) row[1];
        int projectId = (int) row[2];
        System.out.println("Employee ID: " + employeeId + ", Salary: " + salary + ", Project ID: " + projectId);
    }
    session.close();
    return results;
}
```

Podemos observar que creamos una sesion, usando la clase creada anteriormente para la conexión a la bbdd.

Luego escribimos la consulta en un string, y ese string se lo pasamos a una query, que ejecutamos junto a la sesion creada anteriormente y luego los resultados los añadimos a una lista, que mas adelante devolveremos para poder usarlo en el JFrame.

He hecho un syso con el resultado, para comprobar que la consulta está bien hecha.

La estructura es similar en las demas consultas.

La forma de manejar la lista del resultado, es un foreach en la que asignas cada posicion de la lista a un objeto que contiene cada columna de cada fila. Dentro del foeach, hay que hacer una variable por columna, y asignarle el valor del array de objetos correspondiente. Una vez estén asignadas las variables, ya puedes manejarlas como necesites para mostrarlas donde quieras.

No te adjunto capturas de los demas metodos porque es lo mismo, lo unico que cambian son las consultas. El unico que podria cambiar un poco sería la consulta c, pero es solo una variable que se le añade el coste a si misma.

```
double totalCost = 0.0;
for (Object[] row : results) {
    Double salary = (Double) row[1];
    Double hoursWorked = (Double) row[2];

    double hourlySalary = salary / 1900; // Salario por hora
    double cost = hourlySalary * hoursWorked;
    totalCost += cost;
}
```