

# HIBERNATE

ACCESO A DATOS

Alvaro Castro

LAS NAVES SALESIANOS 2DAM

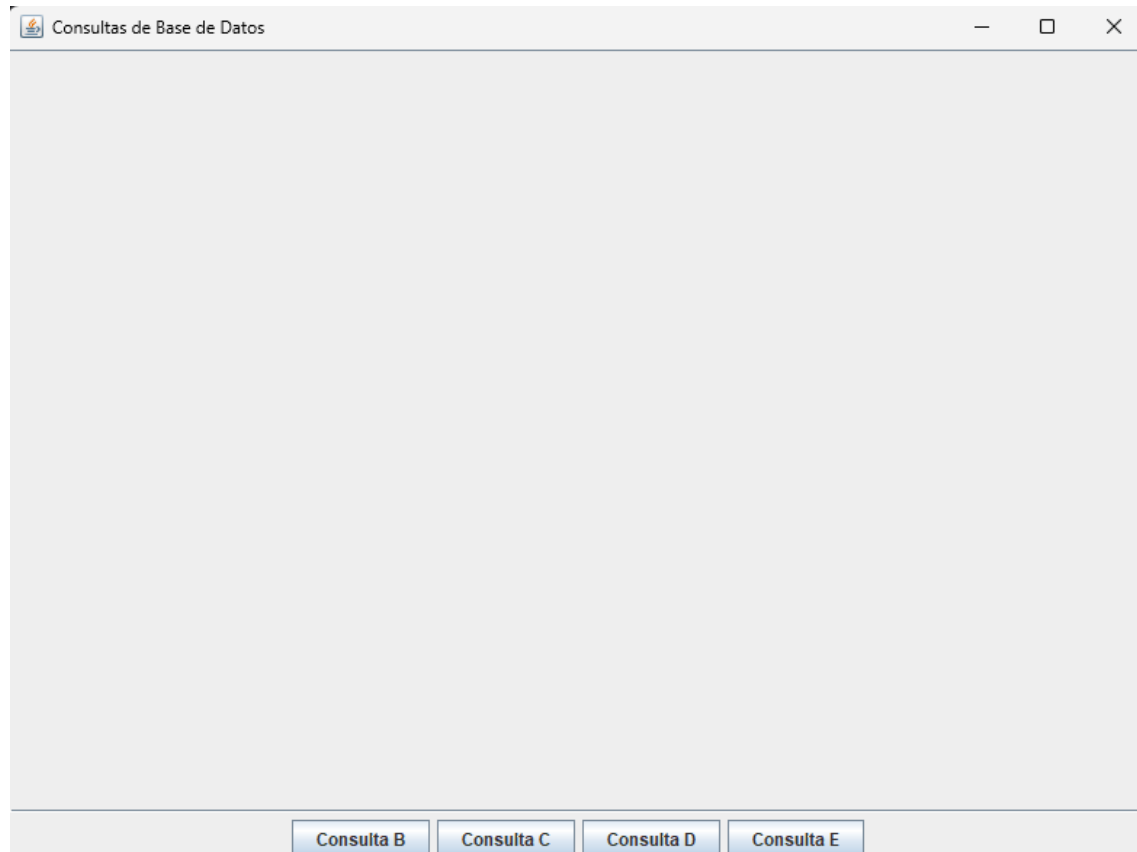
En este documento voy a explicar el funcionamiento de mi programa.

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.

```
> customers CREATE TABLE "customers" ( "customer_id" INTEGER, "customer_name" TEXT, "contact_email" TEXT, "contact_phone" TEXT, PRIMARY KEY("customer_id" AUTOINCREMENT) )
> departments CREATE TABLE "departments" ( "department_id" INTEGER, "department_name" TEXT, "manager_id" INTEGER, PRIMARY KEY("department_id" AUTOINCREMENT) )
> employee_projects CREATE TABLE "employee_projects" ( "employee_id" INTEGER, "project_id" INTEGER, "hours_worked" REAL, "employee_projects_id" INTEGER, PRIMARY KEY("employee_projects_id" AUTOINCREMENT) )
> employees_realistic CREATE TABLE "employees_realistic" ( "employee_id" INTEGER, "first_name" TEXT, "last_name" TEXT, "department_id" INTEGER, "hire_date" TEXT, "salary" REAL, "position" TEXT, "employees_realistic_id" INTEGER, PRIMARY KEY("employees_realistic_id" AUTOINCREMENT) )
> order_items CREATE TABLE "order_items" ( "order_item_id" INTEGER, "order_id" INTEGER, "product_name" TEXT, "quantity" INTEGER, "price" REAL, PRIMARY KEY("order_item_id" AUTOINCREMENT) )
> orders CREATE TABLE "orders" ( "order_id" INTEGER, "customer_id" INTEGER, "order_date" TEXT, "amount" REAL, PRIMARY KEY("order_id" AUTOINCREMENT) )
> projects CREATE TABLE "projects" ( "project_id" INTEGER, "project_name" TEXT, "department_id" INTEGER, "budget" REAL, "start_date" TEXT, "end_date" TEXT, PRIMARY KEY("project_id" AUTOINCREMENT) )
```

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:

Employee ID	Salary	Project ID
25	63059.43	10
92	62315.72	8
90	139659.29	9
50	55178.73	11
64	62059.38	11
52	122598.84	18
32	133670.22	3
19	146812.86	5
84	71305.55	5
89	71724.47	8
1	129459.86	13
97	74778.64	5
99	74816.7	7
14	113484.05	3
100	110099.91	14
55	59119.92	14
29	126225.4	11
23	78359.31	18
90	139659.29	15
67	97770.02	14
60	101944.43	2
7	95469.56	7
72	86756.17	14
32	133670.22	13
16	135010.62	19
59	109367.95	1
18	89771.59	19
60	101944.43	13
86	141989.41	16
68	72894.69	1
72	86756.17	12
77	444457.64	40

## Consulta C:

Project ID	Salary	Hours Worked
10	63059.43	174.7
8	62315.72	44.5
9	139659.29	59.4
11	55178.73	72.2
11	62059.38	96.4
18	122598.84	74.1
3	133670.22	177.1
5	146812.86	62.9
5	71305.55	190.7
8	71724.47	90.1
13	129459.86	168.6
5	74778.64	99.4
7	74816.7	152.3
3	113484.05	19.9
14	110099.91	190.9
14	59119.92	52.6
11	126225.4	23.0
18	78359.31	191.9
15	139659.29	17.6
14	97770.02	15.9
2	101944.43	57.0
7	95469.56	169.5
14	86756.17	128.1
13	133670.22	55.3
19	135010.62	100.0
1	109367.95	31.7
19	89771.59	190.1
13	101944.43	98.4
16	141989.41	58.7
1	72894.69	80.1
12	86756.17	125.1
40	444457.64	400.4

Consulta D:

Consultas de Base de Datos		
Project ID	Salary Costs	Budget
1	18602.57647105263	25597.9
2	3058.3329	73139.01
3	14113.668865789474	46416.0
5	27521.60791473684	87841.22
6	4734.739897894737	77580.34
7	14514.038857894735	44025.78
8	24007.472766842104	44742.77
9	6839.159714210527	52671.43
10	17400.166795789475	52192.03
11	10994.06910894737	43652.42
12	5712.20887736842	30183.02
13	46964.73970210526	53635.71
14	41936.218527894736	95229.09
15	6187.176054736842	74185.44
16	11343.227723157896	92224.44
18	23114.92123473684	44075.99
19	25386.919709473685	63834.98
20	14120.022584210528	20032.48

### Consulta E:

Project ID	Salary Costs Fraction
1	1860257.6471052633
2	305833.29
3	1411366.8865789475
5	2752160.791473684
6	473473.98978947365
7	1451403.8857894735
8	2400747.2766842106
9	683915.9714210527
10	1740016.6795789474
11	1099406.9108947369
12	571220.887736842
13	4696473.970210526
14	4193621.8527894737
15	618717.6054736843
16	1134322.7723157895
18	2311492.1234736843
19	2538691.9709473685
20	1412002.2584210527

Ahora voy a explicar un poco de código.

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

```
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("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("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:

```
@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	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
customer_id	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
order_date	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
amount	REAL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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 sysout 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 foreach, 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;
}
```