ESCUELA POLITECNICA NACIONAL

PROGRAMACIÓN II

SEGUNDO BIMESTRE

¿Qué son las bases de datos?

Una base de datos es un sistema organizado para recopilar, almacenar y gestionar datos. Está diseñada para permitir la recuperación rápida y eficiente de información. En lugar de almacenar datos de forma aislada en archivos, la base de datos organiza la información en tablas interrelacionadas. Estas tablas pueden contener datos como nombres, números de teléfono, direcciones, fechas, y más.

USOS

1. Almacenamiento de Datos: Guardan grandes cantidades de información de manera estructurada.
2. Recuperación de Datos: Permiten recuperar y mostrar datos de manera eficiente.
3. Actualización de Datos: Facilitan la actualización y modificación de la información almacenada.
4. Relacionar Datos: Establecen relaciones entre conjuntos de datos para representar conexiones y asociaciones.
5. Integridad y Seguridad: Garantizan la integridad y seguridad de los datos mediante reglas y permisos.

Existen diferentes tipos de bases de datos, y se clasifican principalmente en dos categorías:

- Bases de Datos Relacionales (RDBMS): Utilizan un modelo relacional para organizar los datos en tablas con filas y columnas. Cada tabla tiene un identificador único llamado clave primaria y puede estar relacionada con otras tablas mediante claves foráneas.

- Bases de Datos No Relacionales (NoSQL): Utilizan modelos de datos diferentes al modelo relacional. Pueden ser basadas en documentos, clave-valor, columnares, etc. No siguen la estructura de tabla típica de las bases de datos relacionales.

SQL (Structured Query Language)

SQL (Lenguaje de Consulta Estructurado) es un lenguaje estándar para la gestión de bases de datos relacionales. Con SQL, puedes realizar diversas operaciones en una base de datos, como crear, leer, actualizar y eliminar datos. Algunos conceptos clave de SQL incluyen:

1. DDL (Data Definition Language): Incluye comandos para definir y gestionar la estructura de la base de datos, como `CREATE TABLE`, `ALTER TABLE`, y `DROP TABLE`.
2. DML (Data Manipulation Language): Comprende comandos para manipular los datos almacenados en la base de datos, como `SELECT`, `INSERT`, `UPDATE` y `DELETE`.
3. Cláusulas SQL: Estas son instrucciones que proporcionan funcionalidades específicas, como `WHERE` para filtrar datos, `ORDER BY` para ordenar resultados, y `JOIN` para combinar datos de múltiples tablas.

SQLite

SQLite es un sistema de gestión de bases de datos relacional que se implementa como una biblioteca en C. A diferencia de algunos sistemas de gestión de bases de datos que funcionan como procesos independientes, SQLite se integra directamente en la aplicación. Algunas características de SQLite incluyen:

- **Sin Servidor:** SQLite no requiere un servidor separado para funcionar. La base de datos se almacena en un solo archivo, lo que facilita la portabilidad.
- **Ligero:** Es una biblioteca pequeña y liviana, adecuada para aplicaciones de bajo a mediano tráfico.
- **Sin Configuración:** No hay una configuración de servidor compleja; simplemente, se accede a la base de datos mediante llamadas a funciones en la biblioteca.

Relacion con Java

Java Database Connectivity (JDBC): Java proporciona la API JDBC, que permite a los desarrolladores interactuar con bases de datos desde aplicaciones Java. Puedes usar JDBC para ejecutar sentencias SQL y gestionar conexiones con bases de datos.

Ejemplo de conexión a una base de datos SQLite en Java utilizando JDBC:

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

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

        try {
            Connection conexion = DriverManager.getConnection(url);
```

```
            System.out.println("Conexión exitosa a SQLite");
        } catch (SQLException e) {
            System.out.println("Error en la conexión: " + e.getMessage());
        }
    }
}
```

Aquí, nombre_base_de_datos.db es el nombre de tu base de datos SQLite.

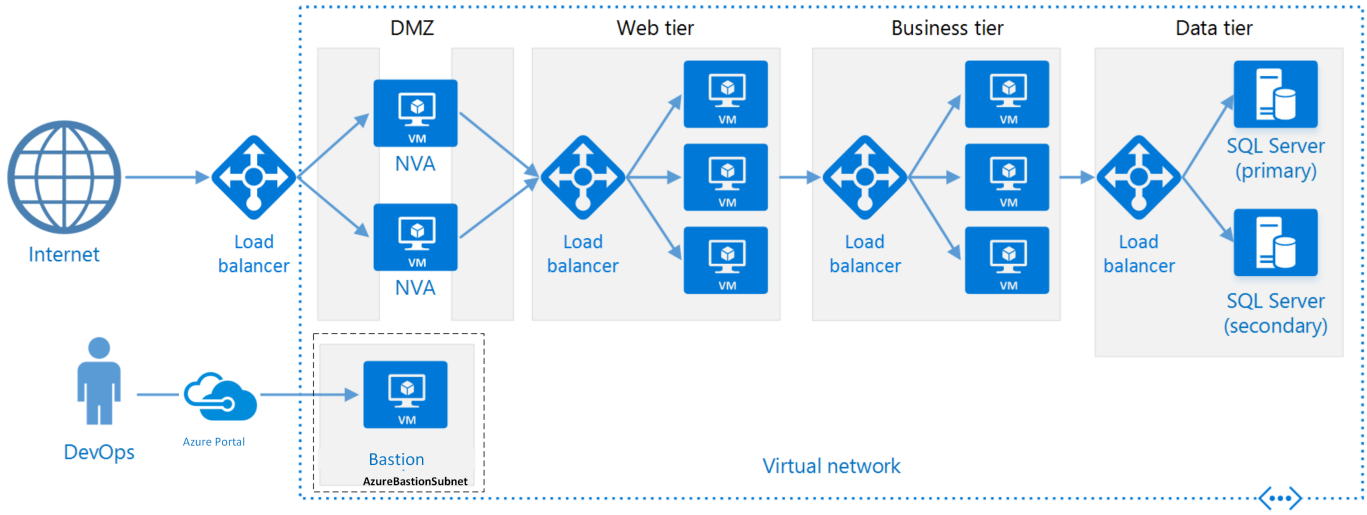
2. Integración con SQL: En tus programas Java, puedes utilizar sentencias SQL para realizar operaciones en la base de datos. JDBC facilita la ejecución de estas sentencias y la manipulación de datos.

Ejemplo de consulta simple con JDBC:

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

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

        try (Connection conexion = DriverManager.getConnection(url);
             Statement statement = conexion.createStatement();
             ResultSet resultSet = statement.executeQuery("SELECT * FROM tabla"))
{

            while (resultSet.next()) {
                // Procesar los resultados
                String columna1 = resultSet.getString("columna1");
                int columna2 = resultSet.getInt("columna2");
                System.out.println(columna1 + ", " + columna2);
            }

        } catch (SQLException e) {
            System.out.println("Error en la consulta: " + e.getMessage());
        }
    }
}
```

Aquí, nombre_base_de_datos.db es el nombre de tu base de datos SQLite, y `tabla` y `columna1`/`columna2` son nombres de la tabla y columnas, respectivamente.

Arquitectura N-Tier



Definición y Características

La arquitectura N-Tier es un enfoque en el desarrollo de aplicaciones que distribuye las responsabilidades en capas o niveles. Cada capa cumple una función específica, promoviendo la modularidad y la escalabilidad del sistema.

Ventajas Clave

Las ventajas clave de la arquitectura N-Tier incluyen una mayor modularidad, mantenibilidad y escalabilidad. La separación de las capas permite una fácil actualización y mejora de componentes individuales sin afectar otras partes del sistema.

Uso de DTO (Data Transfer Object)

En la arquitectura N-Tier, los DTO (Data Transfer Object) son utilizados para transferir datos entre las capas. Estos objetos contienen datos sin lógica de negocio y facilitan la comunicación eficiente entre los diferentes niveles de la arquitectura.

Arquitectura de Base de Datos


Arquitectura

Importancia

La arquitectura de base de datos es esencial en el diseño de sistemas de información, ya que define cómo se organiza y accede a la información. Una buena arquitectura garantiza un rendimiento eficiente, integridad de datos y facilidad de mantenimiento.

Componentes Clave

Los componentes clave de la arquitectura de base de datos incluyen el DBMS (Sistema de Gestión de Bases de Datos), el modelo de datos, el esquema de base de datos y el lenguaje de consulta. Cada componente desempeña un papel fundamental en la gestión y manipulación de datos.

Tipos de Arquitectura de Base de Datos

Existen varios tipos de arquitectura de base de datos, entre ellos la relacional, la NoSQL y la cliente-servidor. Cada tipo tiene sus propias características y se elige según los requisitos y la naturaleza de los datos que manejará el sistema.

Pasos para crear Interfaces graficas GUI

¡Por supuesto! En Java, puedes crear interfaces gráficas de usuario (GUI) utilizando la biblioteca `javax.swing`, que proporciona clases y componentes para construir ventanas, botones, campos de texto y otros elementos de la interfaz gráfica. Aquí hay una explicación básica de cómo crear una interfaz gráfica en Java:

Paso 1: Importar las Clases Necesarias

Primero, importa las clases necesarias de `javax.swing` y otras bibliotecas relacionadas. Puedes hacer esto al inicio de tu archivo Java:

```
import javax.swing.*;
import java.awt.*;
```

Paso 2: Crear un Marco (JFrame)

Un `JFrame` es una ventana en la que colocarás todos los componentes de tu interfaz. Puedes crear un marco utilizando el siguiente código:

```
public class MiVentana extends JFrame {
    public MiVentana() {
        // Configurar el título y el cierre del marco
        setTitle("Mi Interfaz");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Configurar el tamaño del marco
        setSize(400, 300);

        // Configurar el diseño (layout) del contenido del marco
        setLayout(new FlowLayout());

        // Agregar más componentes aquí

        // Hacer visible el marco
        setVisible(true);
    }

    public static void main(String[] args) {
        // Crear una instancia de la ventana
        SwingUtilities.invokeLater(() -> new MiVentana());
    }
}
```

Paso 3: Agregar Componentes

Dentro del constructor del marco, puedes agregar diversos componentes, como botones, etiquetas, campos de texto, etc. Aquí hay un ejemplo que agrega un botón:

```
public MiVentana() {
    // ... (código anterior)

    // Crear un botón
    JButton miBoton = new JButton("Haz clic");

    // Agregar el botón al marco
    add(miBoton);

    // ... (código posterior)
}
```

Paso 4: Responder a Eventos

Si deseas que algo suceda cuando el usuario interactúa con un componente (por ejemplo, hacer clic en un botón), necesitas agregar un listener (escuchador). Aquí hay un ejemplo:

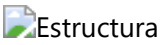
```
public MiVentana() {
    // ... (código anterior)

    // Crear un botón
    JButton miBoton = new JButton("Haz clic");

    // Agregar un ActionListener al botón
    miBoton.addActionListener(e -> {
        // Acciones a realizar cuando se hace clic en el botón
        JOptionPane.showMessageDialog(this, "¡Haz hecho clic!");
    });

    // Agregar el botón al marco
    add(miBoton);

    // ... (código posterior)
}
```

Paso 5: Ejecutar la Aplicación

Finalmente, ejecuta tu aplicación. En el método `main`, crea una instancia de tu ventana. Puedes hacerlo utilizando `SwingUtilities.invokeLater` para asegurarte de que las operaciones de interfaz gráfica se ejecuten en el hilo de eventos de la interfaz gráfica.

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new MiVentana());
}
```

Estructura de un proyecto

 Estructura

Proyecto

- **database:** Contiene el archivo DDL (Data Definition Language) para crear la estructura de la base de datos. El archivo `DDL_PoliBank` es esencial para establecer la arquitectura y definir las tablas, relaciones y restricciones necesarias para almacenar y organizar los datos de manera efectiva.
    - `DataBasePoliBank.sqlite`**:** Archivo de base de datos SQLite que almacena la información del proyecto. Este archivo se crea y se actualiza según las instrucciones en los scripts DDL y DML.

 Tabla

Carpeta "Script"

- **scripts:** Contiene los scripts DML (Data Manipulation Language) para poblar la base de datos con datos de prueba. Aquí se encuentra el archivo `DML_PoliBank`, que proporciona instrucciones SQL para la inserción de datos. Esto es crucial para realizar pruebas y verificar el funcionamiento correcto del sistema.
- `DDL_PoliBank`**:** Archivo que contiene el lenguaje de definición de datos (DDL) para la creación de la estructura de la base de datos. Especifica tablas, relaciones y restricciones. Este archivo puede contener declaraciones como `CREATE TABLE`, donde se definen los campos de las tablas, sus tipos de datos y las claves primarias y foráneas que conectan las tablas entre sí. Añade una capa de documentación valiosa para el diseño de la base de datos.

```
-- database: ../database/DataBasePoliBank.sqlite
-- DROP TABLE IF EXISTS Transferencia;
-- DROP TABLE IF EXISTS Usuario;
-- DROP TABLE IF EXISTS Sexo;

CREATE TABLE IF NOT EXISTS Sexo (
    IdSexo                  INTEGER PRIMARY KEY AUTOINCREMENT,
    Nombre                  TEXT NOT NULL UNIQUE,
    Estado                  VARCHAR(1) NOT NULL DEFAULT 'A',
    FechaCrea               DATETIME NOT NULL DEFAULT (DATETIME('NOW',
'LOCALTIME')),
    FechaModifica           DATE
);

CREATE TABLE IF NOT EXISTS Usuario (
    IdUsuario               INTEGER PRIMARY KEY AUTOINCREMENT,
    Nombre                  TEXT NOT NULL UNIQUE,
```

```
    Cedula                  VARCHAR(10) NOT NULL UNIQUE,
    Username                TEXT NOT NULL UNIQUE,
    Clave                   TEXT NOT NULL,
    Email                   TEXT NOT NULL UNIQUE,
    IdSexo                  INTEGER NOT NULL REFERENCES Sexo (IdSexo),
    Saldo                   REAL NOT NULL DEFAULT (0.0),
    Estado                  VARCHAR(1) NOT NULL DEFAULT 'A',
    FechaCrea               DATETIME NOT NULL DEFAULT (DATETIME('NOW',
'LOCALTIME')),
    FechaModifica           DATE
);

CREATE TABLE IF NOT EXISTS Transferencia (
    IdTransferencia         INTEGER PRIMARY KEY AUTOINCREMENT,
    IdUsuarioEnvia          INTEGER NOT NULL REFERENCES Usuario (IdUsuario),
    IdUsuarioRecibe         INTEGER NOT NULL REFERENCES Usuario (IdUsuario),
    Monto                   REAL NOT NULL,
    Fecha                   DATE NOT NULL,
    Estado                  VARCHAR(1) NOT NULL DEFAULT 'A',
    FechaCrea               DATETIME NOT NULL DEFAULT (DATETIME('NOW',
'LOCALTIME')),
    FechaModifica           DATE
);
```

- `DML_PoliBank`**:** Archivo que contiene instrucciones SQL del lenguaje de manipulación de datos (DML) para poblar la base de datos con datos de prueba. Dentro de este archivo, se encuentran comandos como `INSERT INTO` que añaden registros específicos a las tablas creadas con el DDL. La presencia de datos de prueba facilita el desarrollo y la validación del sistema antes de su implementación final.

```
-- database: ../database/DataBasePoliBank.sqlite
INSERT OR IGNORE INTO Sexo (Nombre)
VALUES  ('Masculino'),
        ('Femenino'),
        ('Híbrido');
```

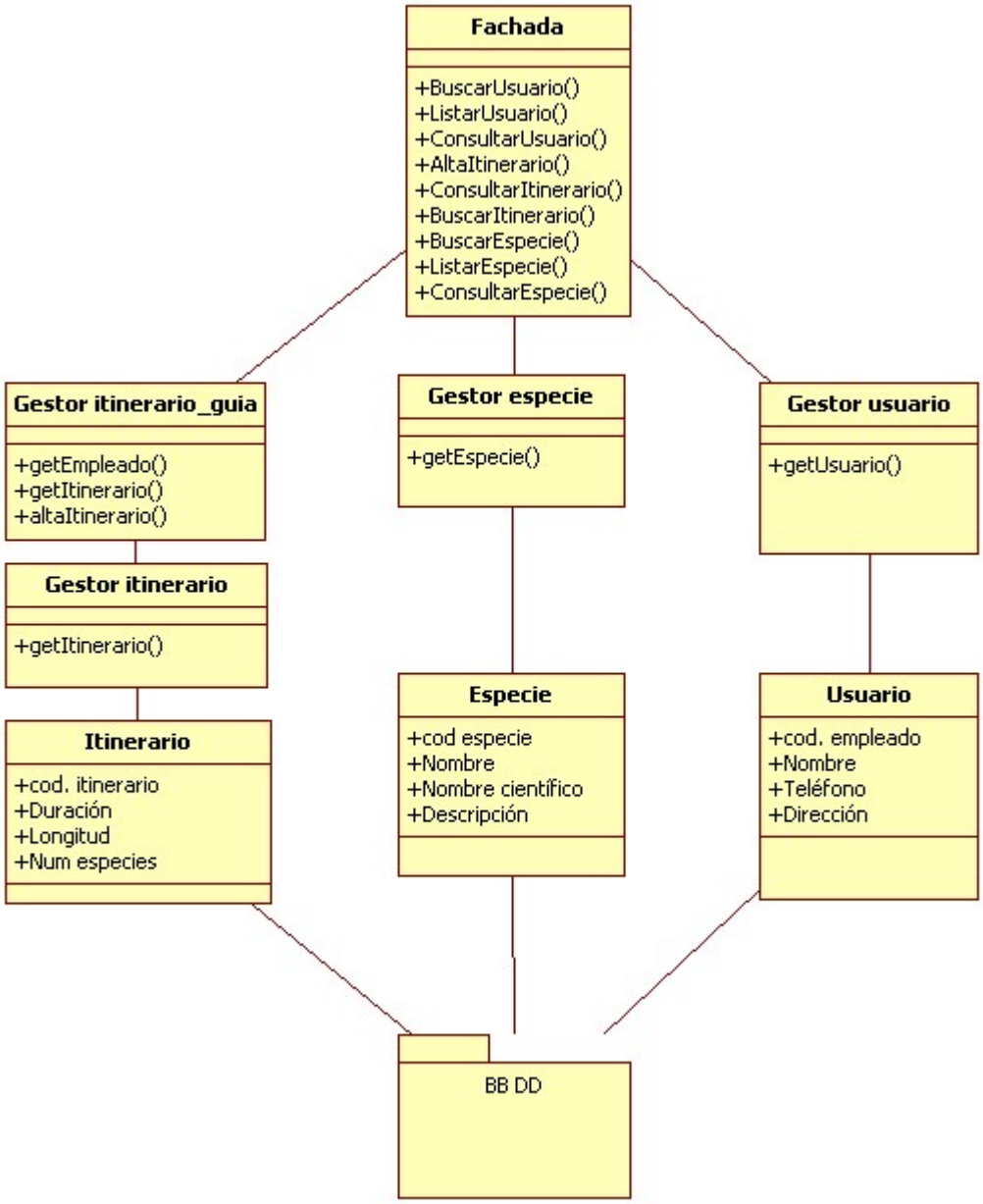
Carpeta "design"

Subcarpetas design

- **DiagramaArquitectura:** Alberga el diagrama de arquitectura del proyecto. Este diagrama visual proporciona una visión general de cómo interactúan los diversos componentes del sistema. Puede incluir capas como la interfaz de usuario, lógica de negocio y acceso a datos, destacando las dependencias y conexiones.
- **DiagramaCasoUso:** Contiene diagramas que detallan las interacciones entre los actores y el sistema. Cada caso de uso representa una funcionalidad específica desde la perspectiva del usuario. Estos diagramas son valiosos para comprender los requisitos funcionales del sistema.

 alt text

- **DiagramaClase:** Ofrece una representación visual de las clases en el proyecto y sus relaciones. Puede incluir atributos, métodos y asociaciones entre las clases. Este diagrama es esencial para la comprensión detallada de la estructura del código.



- **DiagramaEntidadRelacion:** Contiene el diagrama entidad-relación de la base de datos. Este diagrama visualiza las entidades, atributos y relaciones en la base de datos, brindando una representación gráfica de la estructura de los datos.

 Relacion

Carpeta "lib"

Subcarpetas de Lib

- **JDBC (Java Database Connectivity):** Aquí se encuentran las librerías relacionadas con JDBC, que permiten la conexión y la interacción con la base de datos SQLite desde Java. JDBC facilita la

comunicación entre la aplicación y la base de datos, permitiendo operaciones como consultas, inserciones y actualizaciones.

- **Herramientas de Interfaz Gráfica:** Contiene librerías y herramientas para mejorar la apariencia y la experiencia de usuario de la interfaz gráfica. Pueden incluir bibliotecas para la creación de componentes visuales atractivos y herramientas de diseño que simplifican el desarrollo de interfaces de usuario.

Carpeta "src"

Subcarpetas src

- **BusinessLogic:** Aquí se encuentran las clases que implementan la lógica de negocio del proyecto. Estas clases contienen las reglas y procesos esenciales para el funcionamiento del sistema. Proporcionan una capa de abstracción entre la interfaz de usuario y la capa de acceso a datos.
- **DataAccess:** Reside en esta carpeta las clases responsables del acceso a la base de datos. Estas clases interactúan con la capa de datos y gestionan las consultas y actualizaciones. El uso de JDBC, ubicado en la carpeta "lib", puede ser esencial en esta capa para establecer conexiones y ejecutar consultas SQL.
- **Framework:** Contiene las clases y componentes que proporcionan funcionalidades compartidas y utilidades. Puede incluir clases de utilidad genéricas, métodos de manejo de eventos y otras funciones que facilitan el desarrollo y la coherencia del código.
- **UserInterface:** Alberga las clases que implementan la interfaz de usuario del proyecto. Aquí se encuentran los elementos visuales y la lógica asociada. Las herramientas de interfaz gráfica, provenientes de la carpeta "lib", pueden ser esenciales para personalizar y mejorar la apariencia de la interfaz.

Otros archivos

- **.gitignore:** Un archivo crucial que especifica archivos y carpetas que no deben ser rastreados por Git. Esto evita la inclusión accidental de archivos sensibles o irrelevantes en el repositorio.
- **README.md:** Un documento informativo sobre el proyecto. Incluye detalles como el nombre, descripción, autores y requisitos. Sirve como una guía rápida y una referencia útil para cualquier persona que interactúe con el repositorio.