Universidad Técnica Nacional

Curso: Base de Datos

Documentación del Laboratorio

Laboratorio: Reservas de Vuelo

Profesor: Andrés Joseph Jiménez Leandro - IIC-2025

Estudiantes:

Kristel Fiorella Barquero Arias

Kiany Daney Granados Díaz

Cristian Gdo Acuña Rodríguez

Tabla de contenidos

Introducción

El presente laboratorio consiste en el desarrollo de un sistema de gestión de reservas de vuelos, diseñado para aplicar los conocimientos adquiridos en bases de datos y programación orientada a objetos. Para la creación y administración de la base de datos se utilizó como motor SQL Server 2022, donde se implementó la normalización con el objetivo de garantizar la integridad y consistencia de los datos. En cuanto al desarrollo del frontend, se empleó Java Swing para la interfaz gráfica, esto permite a los usuarios realizar operaciones básicas como consultar vuelos, gestionar reservaciones y procesar pagos en una vista más amigable y entendible.

El trabajo se distribuyó entre los miembros del equipo, asignando responsabilidades específicas: algunos integrantes se enfocaron en el desarrollo de consultas SQL complejas, mientras que otros trabajaron en la implementación de la interfaz gráfica. Esta división de tareas permitió integrar satisfactoriamente ambos componentes. Como resultado se obtuvo un sistema funcional que cumple con los requisitos académicos establecidos. Así mismo, el laboratorio demostró la importancia de un buen diseño de base de datos y una correcta integración con la interfaz de usuario.

Implementación de las tablas normalizadas

Imagen #1: Tablas independientes

```sql
CREATE TABLE Aerolinea (
    Id_Aerolinea INT PRIMARY KEY,
    Nombre VARCHAR(50) NOT NULL,
    Siglas_Aero CHAR(3) NULL,
    Pais VARCHAR(30) NOT NULL
);

CREATE TABLE Pasajero (
    Id_Pasajero INT PRIMARY KEY,
    Nombre VARCHAR(50) NOT NULL,
    Apellido VARCHAR(50) NOT NULL,
    Pasaporte VARCHAR(20) NOT NULL,
    Email VARCHAR(100) NULL,
    Telefono VARCHAR(15) NULL
);
```

Fuente: Elaboración propia

Primero se crearon las tablas independientes, ya que estas tablas sirven como base para luego establecer relaciones con las demás entidades.

Imagen #2: Estructura de la Tabla Vuelo

```sql
CREATE TABLE Vuelo (
    Id_Vuelo INT PRIMARY KEY,
    Origen VARCHAR(3) NOT NULL,
    Destino VARCHAR(3) NOT NULL,
    Fecha_Salida DATE NOT NULL,
    Hora_Salida TIME NOT NULL,
    Id_Aerolinea INT NOT NULL,
    FOREIGN KEY (Id_Aerolinea) REFERENCES Aerolinea(Id_Aerolinea)
);
```

Fuente: Elaboración propia

La tabla Vuelo fue establecida para brindar de manera concisa los datos espaciales y temporales del Vuelo que se desea reservar además de relacionarse con la entidad Aerolínea. En este sentido la tabla vuelo muestra detalles específicos de cada uno de los vuelos disponibles.

Imagen #3: Tablas en las que se basa el sistema de reservas

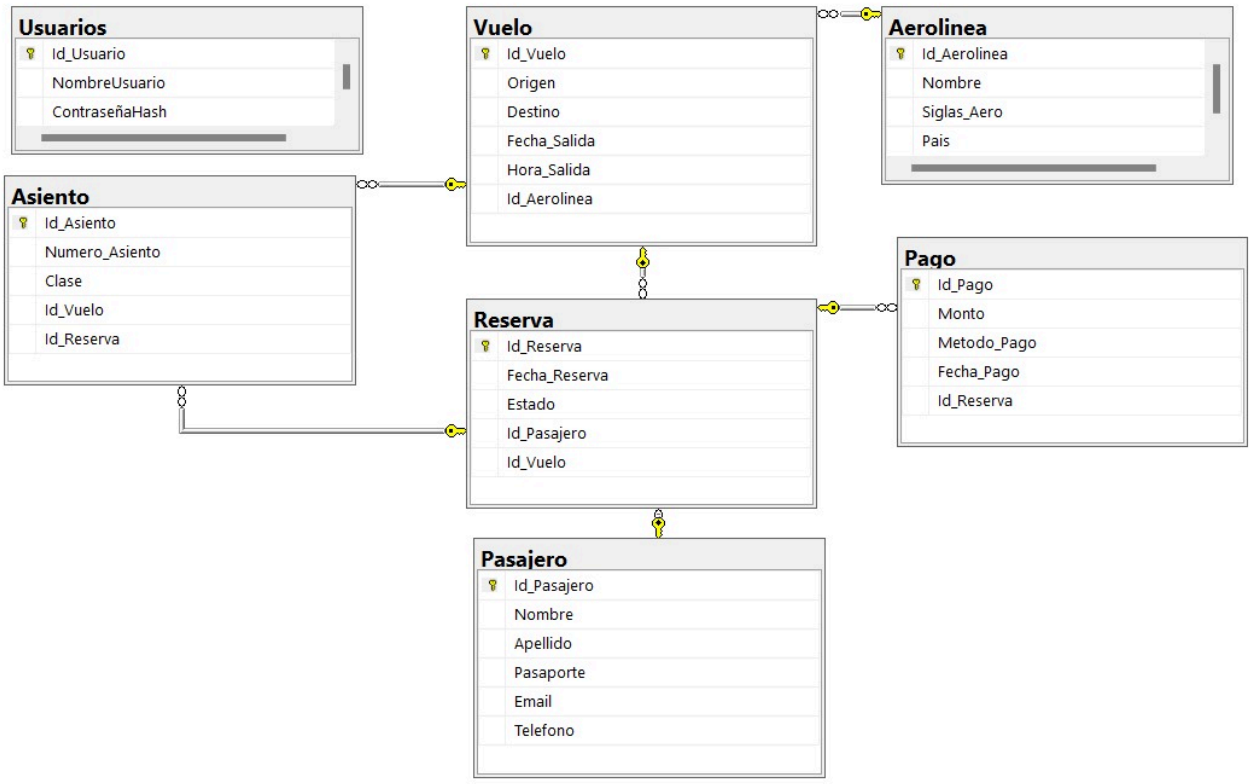
```sql
CREATE TABLE Reserva (
    Id_Reserva INT PRIMARY KEY,
    Fecha_Reserva DATE NOT NULL,
    Estado VARCHAR(20) NOT NULL,
    Id_Pasajero INT NOT NULL,
    Id_Vuelo INT NOT NULL,
    FOREIGN KEY (Id_Pasajero) REFERENCES Pasajero(Id_Pasajero),
    FOREIGN KEY (Id_Vuelo) REFERENCES Vuelo(Id_Vuelo)
);

CREATE TABLE Asiento (
    Id_Asiento INT PRIMARY KEY,
    Numero_Asiento VARCHAR(5) NOT NULL,
    Clase VARCHAR(20) NOT NULL,
    Id_Vuelo INT NOT NULL,
    Id_Reserva INT NULL,
    FOREIGN KEY (Id_Vuelo) REFERENCES Vuelo(Id_Vuelo),
    FOREIGN KEY (Id_Reserva) REFERENCES Reserva(Id_Reserva)
);

CREATE TABLE Pago (
    Id_Pago INT PRIMARY KEY,
    Monto DECIMAL(10, 2) NOT NULL,
    Metodo_Pago VARCHAR(20) NOT NULL,
    Fecha_Pago DATE NOT NULL,
    Id_Reserva INT NOT NULL,
    FOREIGN KEY (Id_Reserva) REFERENCES Reserva(Id_Reserva)
);
```

Fuente: Elaboración propia

Estas tres tablas trabajan juntas para gestionar todo el proceso de la reserva como tal. En un primer orden, la tabla “Reserva” es la tabla principal en la lógica del sistema, ya que vincula al pasajero con su vuelo. En un segundo orden, por cada reserva creada la tabla “Asiento” permite asignar el puesto del pasajero, manteniendo la libertad de poder manejar el estado de los asientos entre disponibles o reservados. Finalmente, la tabla “Pago” registra la transacción de la compra, esto asegura que cada reserva tenga un comprobante de pago vinculado, generando mayor organización a la hora de gestionar el tema finanzas de la aerolínea y comprobante del pasajero. La unión de estas tablas garantizan que desde la selección del vuelo hasta el pago final, la información quede registrada en el sistema.

Diagrama de relación UML de Bases de Datos

A continuación se muestra el diagrama UML de la base de datos



Administración de la Base de Datos

Inserts de tablas:

Imagen 1: Tabla Aerolínea

```
--Inserts
INSERT INTO Aerolinea VALUES
(1, 'Aerolíneas Argentinas', 'ARG', 'Argentina'),
(2, 'Latam Airlines', 'LAT', 'Chile'),
(3, 'Avianca', 'AVI', 'Colombia'),
(4, 'Copa Airlines', 'COP', 'Panamá'),
(5, 'Iberia', 'IBE', 'España'),
(6, 'American Airlines', 'AAL', 'Estados Unidos'),
(7, 'Air France', 'AFR', 'Francia'),
(8, 'Lufthansa', 'DLH', 'Alemania'),
(9, 'Emirates', 'EMI', 'Emiratos Árabes'),
(10, 'Qatar Airways', 'QTR', 'Catar'),
(11, 'Turkish Airlines', 'THY', 'Turquía'),
(12, 'KLM', 'KLM', 'Países Bajos'),
(13, 'United Airlines', 'UAL', 'Estados Unidos'),
(14, 'British Airways', 'BAW', 'Reino Unido'),
(15, 'Aeroméxico', 'AMX', 'México');
```

Imagen 2: Tabla Pasajero

```
INSERT INTO Pasajero VALUES
(1, 'Carlos', 'Gómez', 'P123456', 'carlosg@example.com', '+549111111111'),
(2, 'María', 'Pérez', 'P223456', 'mariap@example.com', '+569222222222'),
(3, 'Luis', 'Martínez', 'P323456', 'luism@example.com', '+573333333333'),
(4, 'Ana', 'López', 'P423456', 'anal@example.com', '+574444444444'),
(5, 'Pedro', 'Sánchez', 'P523456', 'pedros@example.com', '+525555555555'),
(6, 'Lucía', 'Torres', 'P623456', 'luciat@example.com', '+346666666666'),
(7, 'Javier', 'Morales', 'P723456', 'javierm@example.com', '+336777777777'),
(8, 'Sofía', 'Hernández', 'P823456', 'sofiah@example.com', '+971888888888'),
(9, 'Martín', 'Ramírez', 'P923456', 'martinr@example.com', '+974999999999'),
(10, 'Elena', 'Fernández', 'P103456', 'elenaf@example.com', '+905000000000'),
(11, 'Diego', 'Castro', 'P113456', 'diegoc@example.com', '+315111111111'),
(12, 'Valentina', 'Rojas', 'P123457', 'valentinar@example.com', '+447222222222'),
(13, 'Andrés', 'García', 'P133456', 'andresg@example.com', '+525333333333'),
(14, 'Paula', 'Ortiz', 'P143456', 'paulao@example.com', '+528444444444'),
(15, 'Fernando', 'Mendoza', 'P153456', 'fernandom@example.com', '+529555555555');
```

Imagen 3: Tabla Vuelo

```
INSERT INTO Vuelo VALUES
(1, 'EZE', 'SCL', '2025-09-01', '08:00:00', 1),
(2, 'SCL', 'BOG', '2025-09-02', '10:30:00', 2),
(3, 'BOG', 'PTY', '2025-09-03', '14:45:00', 3),
(4, 'PTY', 'MAD', '2025-09-04', '18:20:00', 4),
(5, 'MAD', 'JFK', '2025-09-05', '12:15:00', 5),
(6, 'JFK', 'CDG', '2025-09-06', '09:00:00', 6),
(7, 'CDG', 'FRA', '2025-09-07', '07:30:00', 7),
(8, 'FRA', 'DXB', '2025-09-08', '22:00:00', 8),
(9, 'DXB', 'DOH', '2025-09-09', '06:45:00', 9),
(10, 'DOH', 'IST', '2025-09-10', '15:00:00', 10),
(11, 'IST', 'AMS', '2025-09-11', '13:30:00', 11),
(12, 'AMS', 'ORD', '2025-09-12', '08:15:00', 12),
(13, 'ORD', 'LHR', '2025-09-13', '20:45:00', 13),
(14, 'LHR', 'MEX', '2025-09-14', '16:25:00', 14),
(15, 'MEX', 'EZE', '2025-09-15', '11:50:00', 15);
```

Imagen 4: Tabla Reserva

```
INSERT INTO Reserva VALUES
(1, '2025-08-01', 'Confirmada', 1, 1),
(2, '2025-08-02', 'Pendiente', 2, 2),
(3, '2025-08-03', 'Cancelada', 3, 3),
(4, '2025-08-04', 'Confirmada', 4, 4),
(5, '2025-08-05', 'Confirmada', 5, 5),
(6, '2025-08-06', 'Pendiente', 6, 6),
(7, '2025-08-07', 'Confirmada', 7, 7),
(8, '2025-08-08', 'Cancelada', 8, 8),
(9, '2025-08-09', 'Confirmada', 9, 9),
(10, '2025-08-10', 'Pendiente', 10, 10),
(11, '2025-08-11', 'Confirmada', 11, 11),
(12, '2025-08-12', 'Confirmada', 12, 12),
(13, '2025-08-13', 'Cancelada', 13, 13),
(14, '2025-08-14', 'Pendiente', 14, 14),
(15, '2025-08-15', 'Confirmada', 15, 15);
```

Imagen 5: Tabla Asiento

```
INSERT INTO Asiento VALUES
(1, '1A', 'Primera', 1, 1),
(2, '2B', 'Económica', 1, 2),
(3, '3C', 'Económica', 2, 3),
(4, '4D', 'Ejecutiva', 2, 4),
(5, '5E', 'Económica', 3, 5),
(6, '6F', 'Económica', 3, 6),
(7, '7A', 'Primera', 4, 7),
(8, '8B', 'Ejecutiva', 4, 8),
(9, '9C', 'Económica', 5, 9),
(10, '10D', 'Económica', 5, 10),
(11, '11E', 'Ejecutiva', 6, 11),
(12, '12F', 'Primera', 6, 12),
(13, '13A', 'Económica', 7, 13),
(14, '14B', 'Económica', 7, 14),
(15, '15C', 'Primera', 8, 15);
```

Imagen 6: Tabla Pago

```
INSERT INTO Pago VALUES
(1, 150.00, 'Tarjeta Crédito', '2025-08-02', 1),
(2, 200.00, 'Efectivo', '2025-08-03', 2),
(3, 300.00, 'Tarjeta Débito', '2025-08-04', 3),
(4, 450.00, 'Tarjeta Crédito', '2025-08-05', 4),
(5, 500.00, 'Transferencia', '2025-08-06', 5),
(6, 600.00, 'Tarjeta Crédito', '2025-08-07', 6),
(7, 700.00, 'Efectivo', '2025-08-08', 7),
(8, 800.00, 'Tarjeta Débito', '2025-08-09', 8),
(9, 900.00, 'Transferencia', '2025-08-10', 9),
(10, 1000.00, 'Tarjeta Crédito', '2025-08-11', 10),
(11, 1100.00, 'Efectivo', '2025-08-12', 11),
(12, 1200.00, 'Tarjeta Débito', '2025-08-13', 12),
(13, 1300.00, 'Tarjeta Crédito', '2025-08-14', 13),
(14, 1400.00, 'Transferencia', '2025-08-15', 14),
(15, 1500.00, 'Efectivo', '2025-08-16', 15);
```

Usuario y rol:

En la imagen se observa cómo se establece un login a nivel de servidor con una contraseña y políticas de seguridad específicas, luego se crea un usuario en la base de datos vinculado a ese login, y finalmente se configura un rol denominado Rol_Administrador que facilitará la organización y asignación de permisos a los usuarios.

```
-- 1. Crear un Login
CREATE LOGIN MiUsuarioLogin WITH PASSWORD = 'MiContraseñaSegura', CHECK_POLICY = ON, CHECK_EXPIRATION = OFF;
--2.Crear un Usuario
CREATE USER MiUsuario FOR LOGIN MiUsuarioLogin;
--3.Crear un Rol
CREATE ROLE Rol_Administrador;
```

En la imagen se muestra cómo asignamos los permisos necesarios al rol creado y como se asigna el rol a nuestro usuario

```
-- 4. Asignar permisos al Rol
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Aerolinea TO Rol_Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Pasajero TO Rol_Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Vuelo TO Rol_Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Reserva TO Rol_Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Asiento TO Rol_Administrador;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Pago TO Rol_Administrador;
-- 5. Asignar el Rol al Usuario
ALTER ROLE Rol_Administrador ADD MEMBER MiUsuario;
```

Encriptación

En la imagen se crea una tabla de usuarios con contraseñas encriptadas, se inserta un usuario administrador con su clave cifrada mediante sha2_256 y se realiza una consulta para validar las credenciales contra los datos almacenados.

```
--Encriptacion
-- Crear una tabla para usuarios con contraseña
CREATE TABLE Usuarios (
    Id_Usuario INT PRIMARY KEY,
    NombreUsuario VARCHAR(50) NOT NULL,
    ContraseñaHash VARBINARY(MAX) NOT NULL
);
--Insertar usuario con contraseña encriptada
INSERT INTO Usuarios (Id_Usuario, NombreUsuario, ContraseñaHash)
VALUES (1, 'administrador', HASHBYTES('SHA2_256', 'MiContraseñaSuperSecreta'));
--Verificar la contraseña
SELECT *
FROM Usuarios
WHERE NombreUsuario = 'administrador'
AND ContraseñaHash = HASHBYTES('SHA2_256', 'MiContraseñaSuperSecreta');
```

Funcion

En la imagen se muestra un ejemplo que implementa una función para verificar la contraseña de un usuario y comprobar si la autenticación es exitosa o fallida.

```
--Funcion
--Verificar contraseña de un usuario
CREATE FUNCTION dbo.VerificarContraseñaUsuario (
    @NombreUsuario VARCHAR(50),
    @ContraseñaPlana VARCHAR(MAX)
)
RETURNS BIT
AS
BEGIN
    DECLARE @Resultado BIT = 0;
    DECLARE @HashAlmacenado VARBINARY(MAX);
    SELECT @HashAlmacenado = ContraseñaHash
    FROM Usuarios
    WHERE NombreUsuario = @NombreUsuario;
    IF @HashAlmacenado IS NOT NULL AND @HashAlmacenado = HASHBYTES('SHA2_256', @ContraseñaPlana)
    BEGIN
        SET @Resultado = 1;
    END
    RETURN @Resultado;
END;
--Función con contraseña correcta
SELECT dbo.VerificarContraseñaUsuario('administrador', 'MiContraseñaSuperSecreta') AS AutenticacionExitosa;
--Función con contraseña incorrecta
SELECT dbo.VerificarContraseñaUsuario('administrador', 'ContraseñaIncorrecta') AS AutenticacionFallida;
```

Row_Number y Partition by

La imagen muestra una consulta que utiliza la función de ventana row_number con partition by, en la consulta selecciona información de reservas, pasajeros y vuelos, y le asigna un número secuencial a cada reserva para cada pasajero. El número se basa en la fecha de la reserva.

```
--Uso de Row_Number y Partition by
SELECT
    p.Nombre + ' ' + p.Apellido AS Pasajero,
    r.Id_Reserva,
    r.Fecha_Reserva,
    r.Estado,
    v.Origen + ' → ' + v.Destino AS Ruta,
    ROW_NUMBER() OVER (PARTITION BY p.Id_Pasajero ORDER BY r.Fecha_Reserva) AS Num_Reserva_Pasajero
FROM Reserva r
INNER JOIN Pasajero p ON r.Id_Pasajero = p.Id_Pasajero
INNER JOIN Vuelo v ON r.Id_Vuelo = v.Id_Vuelo
ORDER BY p.Apellido, r.Fecha_Reserva;
```

Over:

La imagen final presenta una consulta que utiliza la cláusula over para llevar a cabo cálculos agregados (suma, promedio y conteo) así como cálculos acumulados sobre la información de aerolíneas, vuelos y pagos.

```
--Uso de Over
SELECT
    a.Nombre AS Aerolinea,
    a.Pais,
    v.Fecha_Salida,
    pg.Monto,
    SUM(pg.Monto) OVER (PARTITION BY a.Id_Aerolinea) AS Total_Ingresos_Aerolinea,
    AVG(pg.Monto) OVER (PARTITION BY a.Id_Aerolinea) AS Promedio_Ingresos_Aerolinea,
    COUNT(*) OVER (PARTITION BY a.Id_Aerolinea) AS Total_Reservas_Aerolinea,
    SUM(pg.Monto) OVER (PARTITION BY a.Id_Aerolinea ORDER BY v.Fecha_Salida) AS Ingresos_Acumulados
FROM Aerolinea a
INNER JOIN Vuelo v ON a.Id_Aerolinea = v.Id_Aerolinea
INNER JOIN Reserva r ON v.Id_Vuelo = r.Id_Vuelo
INNER JOIN Pago pg ON r.Id_Reserva = pg.Id_Reserva
WHERE r.Estado = 'Confirmada'
ORDER BY a.Nombre, v.Fecha_Salida;
```

Join:

La ilustración presenta una consulta que utiliza inner join para determinar estadísticas de ocupación por vuelo y clase, incluyendo la cantidad de asientos ocupados, libres y el porcentaje de ocupación.

```
--Uso de Inner joins
SELECT
    v.Id_Vuelo,
    a.Nombre AS Aerolinea,
    v.Origen + ' → ' + v.Destino AS Ruta,
    v.Fecha_Salida,
    ast.Clase,
    COUNT(ast.Id_Asiento) AS Asientos_Clase,
    COUNT(ast.Id_Reserva) AS Asientos_Ocupados,
    COUNT(ast.Id_Asiento) - COUNT(ast.Id_Reserva) AS Asientos_Disponibles,
    CAST(COUNT(ast.Id_Reserva) * 100.0 / COUNT(ast.Id_Asiento) AS DECIMAL(5,2)) AS Porcentaje_Ocupacion,
    AVG(CAST(COUNT(ast.Id_Reserva) * 100.0 / COUNT(ast.Id_Asiento) AS DECIMAL(5,2)))
        OVER (PARTITION BY ast.Clase) AS Promedio_Ocupacion_Clase
FROM Vuelo v
INNER JOIN Aerolinea a ON v.Id_Aerolinea = a.Id_Aerolinea
INNER JOIN Asiento ast ON v.Id_Vuelo = ast.Id_Vuelo
GROUP BY v.Id_Vuelo, a.Nombre, v.Origen, v.Destino, v.Fecha_Salida, ast.Clase
ORDER BY v.Fecha_Salida, ast.Clase;
```

Vista:

Se presenta una vista que elige información específica de las reservas con estado Confirmada. La vista integra información de las tablas Reserva, Pasajero, Vuelo, Aerolinea y Asiento para presentar el ID de reserva, el nombre del pasajero, el origen y destino del vuelo, la fecha de salida, el nombre de la aerolínea y el número de asiento

```sql
--Vista
CREATE VIEW VistaReservasConfirmadas AS
SELECT
    r.Id_Reserva,
    p.Nombre + ' ' + p.Apellido AS NombrePasajero,
    v.Origen,
    v.Destino,
    v.Fecha_Salida,
    a.Nombre AS NombreAerolinea,
    ast.Numero_Asiento
FROM Reserva r
INNER JOIN Pasajero p ON r.Id_Pasajero = p.Id_Pasajero
INNER JOIN Vuelo v ON r.Id_Vuelo = v.Id_Vuelo
INNER JOIN Aerolinea a ON v.Id_Aerolinea = a.Id_Aerolinea
INNER JOIN Asiento ast ON r.Id_Reserva = ast.Id_Reserva
WHERE r.Estado = 'Confirmada';
```

Trigger:

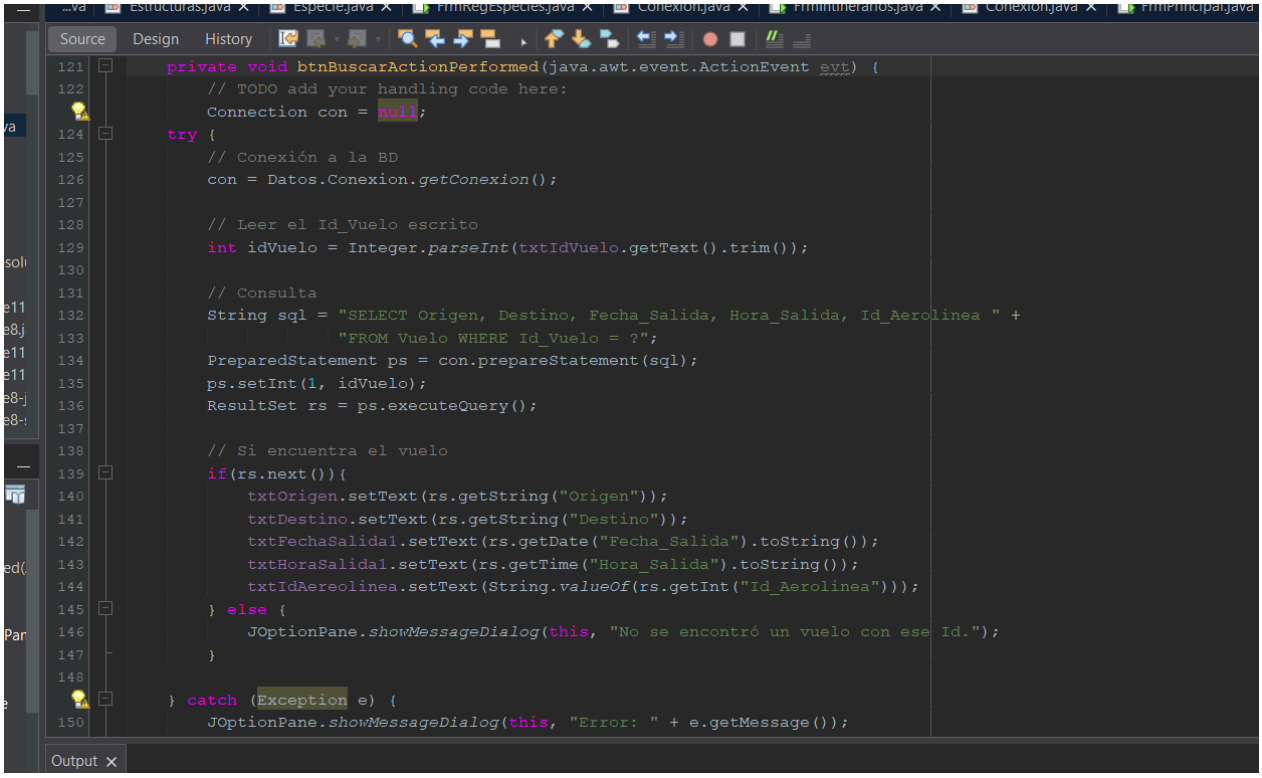
En la imagen se ve como un trigger se ejecuta de forma automática tras la inserción de una nueva fila en la tabla Pago. Su tarea es modificar el estado de la reserva a Confirmada si el estado inicial era Pendiente. La modificación se efectúa para la reserva vinculada al pago recién ingresado.

```sql
--Trigger
CREATE TRIGGER ActualizarEstadoReserva
ON Pago
AFTER INSERT
AS
BEGIN
    UPDATE Reserva
    SET Estado = 'Confirmada'
    FROM Reserva r
    INNER JOIN inserted i ON r.Id_Reserva = i.Id_Reserva
    WHERE r.Estado = 'Pendiente';
END;
--La reserva con id=2 tiene el estado pendiente.
INSERT INTO Pago VALUES (16, 250.00, 'Tarjeta Débito', '2025-08-17', 2);
--Mirar estado de la reserva 2
SELECT * FROM Reserva WHERE Id_Reserva = 2;
```

Interfaz de Java

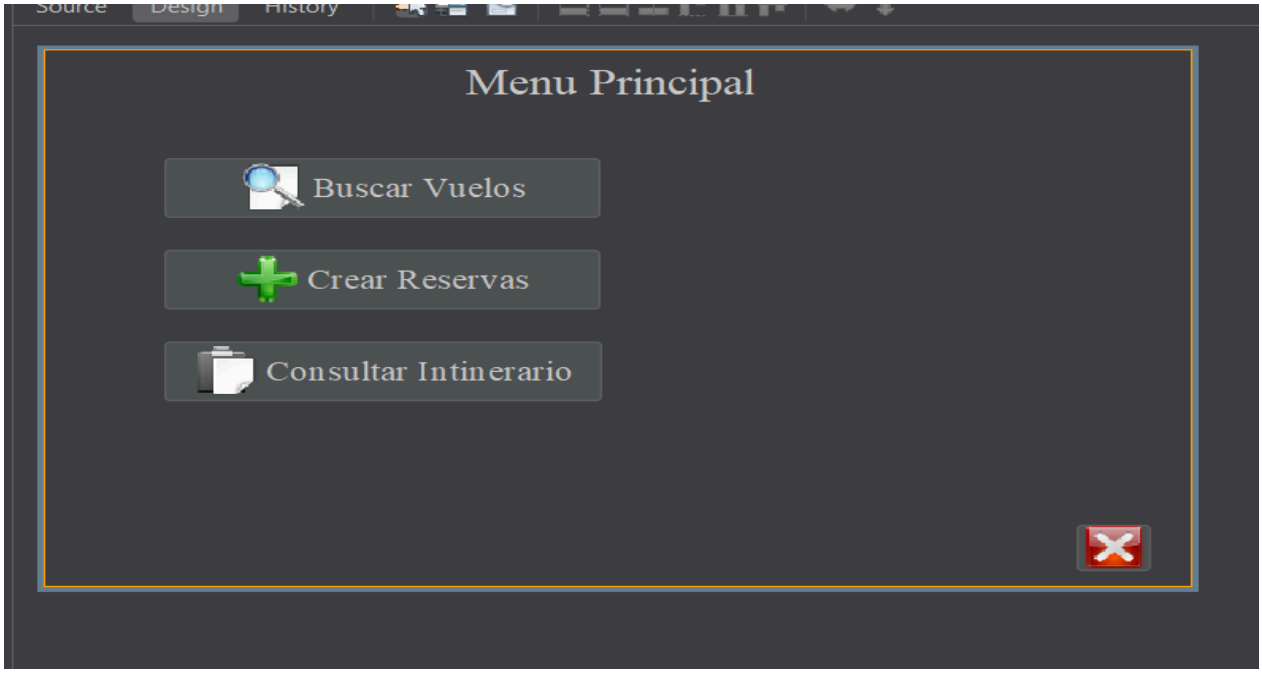
Para la interfaz en Java se mantuvo todo de forma sencilla, los datos se cargan desde la base de datos creada y con una conexión exitosa logrando la carga de todos los datos.

De manera que se cumple la función de buscar vuelos,crear reservas y consultar itinerarios, para esta ultima se modificó la base de datos.



```
    private void btnBuscarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        Connection con = null;
    try {
        // Conexión a la BD
        con = Datos.Conexion.getConexion();

        // Leer el Id_Vuelo escrito
        int idVuelo = Integer.parseInt(txtIdVuelo.getText().trim());

        // Consulta
        String sql = "SELECT Origen, Destino, Fecha_Salida, Hora_Salida, Id_Aerolinea " +
                     "FROM Vuelo WHERE Id_Vuelo = ?";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setInt(1, idVuelo);
        ResultSet rs = ps.executeQuery();

        // Si encuentra el vuelo
        if(rs.next()){
            txtOrigen.setText(rs.getString("Origen"));
            txtDestino.setText(rs.getString("Destino"));
            txtFechaSalida1.setText(rs.getDate("Fecha_Salida").toString());
            txtHoraSalida1.setText(rs.getTime("Hora_Salida").toString());
            txtIdAereolinea.setText(String.valueOf(rs.getInt("Id_Aerolinea")));
        } else {
            JOptionPane.showMessageDialog(this, "No se encontró un vuelo con ese Id.");
        }

    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
```

Este es el código utilizado el cual hace una consulta en sql para llamar los datos que queremos traer.



Este es el diseño utilizado para la interfaz gráfica del menú principal, cuenta con las tres opciones requeridas para el proyecto, buscar vuelos, crear reservas y consultar itinerarios.

```
    }

    private void btnVuelosActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
         FrmVuelos rgVuelos = new FrmVuelos();
        rgVuelos.setLocationRelativeTo(null);
        rgVuelos.setVisible(true);
    }

    private void btnReservasActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        FrmReservas rgReservas = new FrmReservas();
        rgReservas.setLocationRelativeTo(null);
        rgReservas.setVisible(true);
    }

    private void btnIntinerarioActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        FrmIntinerarios rgIntinerarios = new FrmIntinerarios();
        rgIntinerarios.setLocationRelativeTo(null);
        rgIntinerarios.setVisible(true);
    }
```

Algo así se ve el código dentro del menú principal, lo que se está haciendo es llamar a todos los Jframe Form donde están las funcionalidades de cada uno de ellos que permiten la carga de datos desde la base de datos.

```
 * @author Unknown
 */
public class Conexion {

    public Conexion() {
    }

    public static Connection getConexion() throws SQLException
    {
        //Parametros para establecer la connexion
        String cadenaConexion = "jdbc:sqlserver://localhost:1433;" + //controlador jdbc, servidor y su puerto
                                "database=reservasvuelo;" + //Base de datos a usar
                                "user=sa;" + //usuario de conexion a sql
                                "password=191809;" + //cave de conexion a sql
                                "encrypt=true;" + //conexion encriptada
                                "trustServerCertificate=true;" ; //certificado de confianza
        try {
            // establecer una conexion con los parametros dados
            Connection con = DriverManager.getConnection(cadenaConexion);
            return con;
        }catch (SQLException ex){
            JOptionPane.showMessageDialog(null, ex.toString());
            return null;

        }
    }

```

Por último este es el código utilizado en una Java class llamada conexion, aquí es donde se realizó la conexión del java con sql y la que permite la carga de datos mediante las funcionalidades realizadas.

Recomendaciones Finales

Se recomienda mantener una buena estructura de normalización para asegurar la integridad de los datos y evitar la redundancia de los mismos. También se recomienda seguir documentando cada uno de los cambios que se realicen en la base de datos, ya que es la manera en la que se puede entender la estructura para dar el seguimiento correspondiente y actuar a partir de la evidencia suministrada en los documentos.

Se recomienda mantener un manejo ordenado de las diferentes clases o formularios utilizados para crear la interfaz de Java en cualquier caso lo ideal es la realización de formularios divididos y simplemente llamados en los diferentes lugares.