**Proyecto II Bimestre**

**Base de Datos Distribuidas**

**Gestión de Terminales Terrestres**

**ISWD - 553**

**GR1CC**

**Integrantes:** Pasquel Johann, Torres Jorge

**FRAGMENTACIÓN**

**Grafo relacional**

1. Campo y condición de fragmentación

Campo de fragmentación

Condición de fragmentación

*cod\_terminal 1 se refiere a Quito, mientras que cod\_terminal 2 se refiere a Ibarra*

1. Esquema de fragmentación
   1. Fragmentaciones verticales
      1. Pasajeros
      2. Viaje
   2. Fragmentaciones horizontales derivadas
      1. Viaje respecto a terminal
      2. Pasajero respecto a Viaje Información
      3. Vehículo respecto a Terminal

Donde

* + 1. Conductor respecto a Viaje Información

Donde

**REPLICACIÓN**

1. Esquema de replicación

Dentro del diseño del sistema distribuido, la tabla Terminal ha sido replicada en ambas sedes ***(Quito e Ibarra).*** Esta decisión se fundamenta en el hecho de que la tabla no está sujeta a operaciones de escritura o actualización frecuentes, sino que su uso es principalmente de consulta.

* **Tipo de replicación:** Se ha implementado una replicación unidireccional.
* **Nodo de gestión:** El nodo de gestión es Quito, lo que significa que todas las modificaciones o cargas iniciales sobre la tabla Terminal se realizan desde esta sede, y posteriormente se distribuyen a Ibarra.

1. Esquema de Ubicación

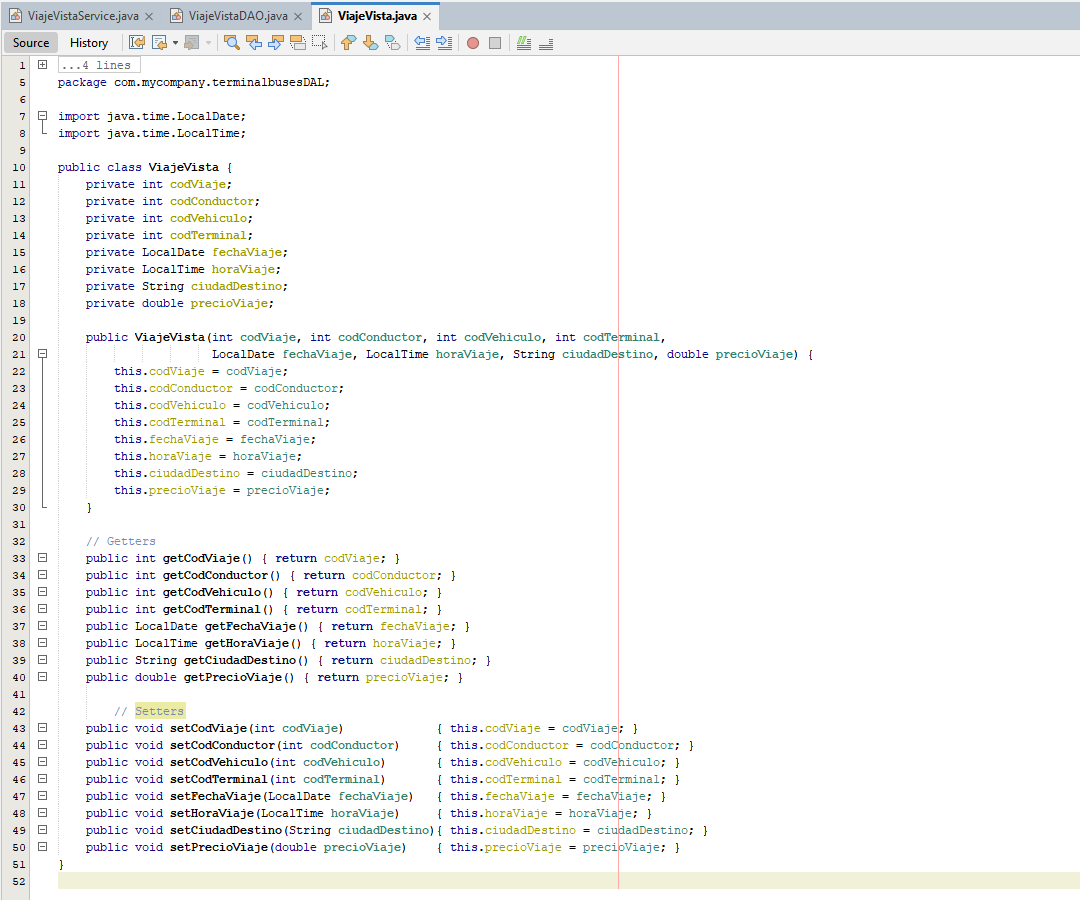
|  |  |  |
| --- | --- | --- |
|  | **QUITO** | **IBARRA** |
| **TERMINAL** | TERMINAL | TERMINAL |
| **VIAJE** | ViajeInformación\_Quito  ViajeTiempo | ViajeInformación\_Ibarra |
| **PASAJERO** | PasajerosInformación\_Quito  PasajerosContato | PasajerosInformación\_Ibarra |
| **CONDUCTOR** | Conductor\_Quito | Conductor\_Ibarra |
| **VEHÍCULO** | Vehículo\_Quito | Vehículo\_Ibarra |

**PROCESO DE NEGOCIO**

El funcionamiento general del aplicativo se basa en la interacción coordinada de tres componentes principales para gestionar las operaciones sobre la base de datos distribuida. Tomando como ejemplo la pestaña Viaje, el proceso inicia con la interfaz gráfica desarrollada en JavaFX, donde el usuario realiza acciones como inserciones, actualizaciones o borrados. Estas acciones son capturadas por el controlador, que se encarga de manejar los eventos y coordinar la lógica de negocio.

El controlador llama a las clases DAO (Data Access Object), que actúan como puente entre el aplicativo y la base de datos, accediendo a las vistas que unifican los fragmentos distribuidos y ejecutando los procedimientos almacenados que realizan las operaciones CRUD en los fragmentos correspondientes. De esta forma, el sistema garantiza que las modificaciones se apliquen correctamente en la base de datos distribuida, manteniendo la integridad y sincronización de los datos entre las sedes.

**ViajeVista**



El código define una clase llamada ViajeVista que se utiliza para representar los datos de un viaje en el sistema. Esta clase contiene atributos privados para almacenar el código del viaje, el código del conductor, el código del vehículo, el código del terminal, la fecha y la hora del viaje, la ciudad de destino y el precio del viaje. Además, incluye un constructor para inicializar todos estos atributos al crear un objeto de tipo ViajeVista.

La clase también proporciona métodos llamados getters y setters, que permiten obtener y modificar los valores de cada uno de los atributos de manera controlada. En resumen, esta clase sirve como un modelo o estructura de datos que facilita el manejo y transporte de la información de los viajes dentro de la aplicación.

**ViajeProcedimientoDAO**

Interfaz de usuario gráfica

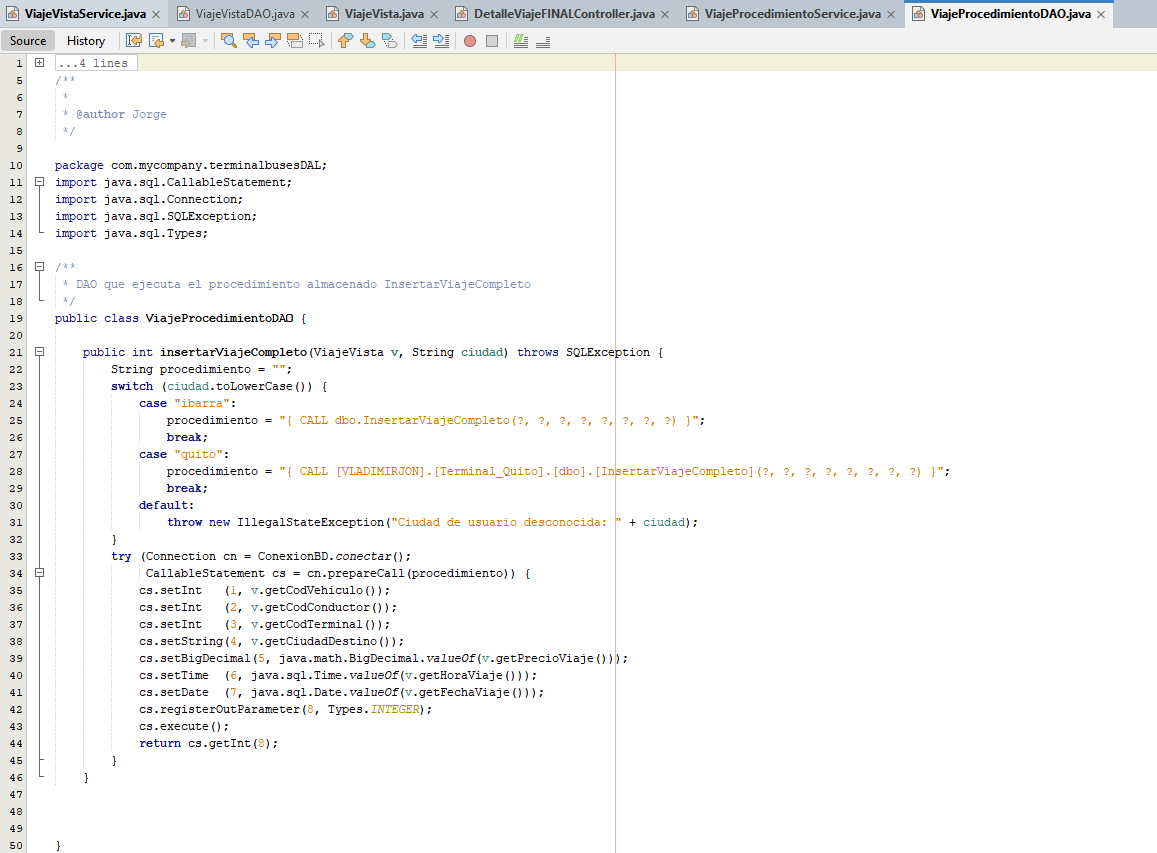
El contenido generado por IA puede ser incorrecto.

El código define la clase ViajeProcedimientoDAO, la cual se encarga de ejecutar el procedimiento almacenado que inserta un viaje completo en la base de datos. La clase contiene un método llamado insertarViajeCompleto, que recibe un objeto de tipo ViajeVista y una cadena que indica la ciudad (por ejemplo, "ibarra" o "quito").

Dependiendo de la ciudad indicada, el método selecciona el procedimiento almacenado correspondiente, ya sea para Ibarra o para Quito. Posteriormente, se conecta a la base de datos y prepara la llamada al procedimiento almacenado, asignando cada uno de los parámetros requeridos usando los datos obtenidos del objeto ViajeVista.

Por último, ejecuta el procedimiento y retorna el resultado obtenido, que generalmente corresponde al identificador generado por la inserción del viaje. Esta clase facilita la comunicación entre la aplicación Java y los procedimientos almacenados en la base de datos, permitiendo que la lógica de inserción sea centralizada y reutilizable.

**ViajeProcedimientoService**



El código define la clase ViajeProcedimientoDAO, cuya función principal es ejecutar el procedimiento almacenado encargado de insertar un viaje completo en la base de datos. El método insertarViajeCompleto recibe como parámetros un objeto ViajeVista con los datos del viaje y una cadena que representa la ciudad.

Dependiendo de la ciudad indicada ("ibarra" o "quito"), el método selecciona la llamada al procedimiento almacenado correspondiente. Si la ciudad no es reconocida, lanza una excepción para notificar el error.

Luego, el método establece la conexión con la base de datos y prepara la llamada al procedimiento almacenado, asignando cada uno de los parámetros necesarios utilizando los valores extraídos del objeto ViajeVista. Finalmente, ejecuta la llamada y retorna el identificador generado por el procedimiento.

En resumen, esta clase permite a la aplicación interactuar fácilmente con los procedimientos almacenados en la base de datos para registrar viajes de manera centralizada y estructurada.

**ViajeProcedimientoService**

SET XACT\_ABORT ON;

GO

CREATE OR ALTER PROCEDURE dbo.InsertarViajeCompleto

@codVehiculo INT,

@codConductor INT,

@codTerminal INT,

@ciudadDestino NVARCHAR(100),

@precioViaje DECIMAL(10,2),

@horaViaje TIME,

@fechaViaje DATE,

@nuevoCodViaje INT OUTPUT

AS

BEGIN

SET NOCOUNT ON;

-- 1) Inserción local en Ibarra

BEGIN TRANSACTION;

INSERT INTO dbo.ViajeInformacion\_Ibarra

(cod\_vehiculo, cod\_conductor, cod\_terminal, ciudad\_destino, precio\_viaje)

VALUES

(@codVehiculo, @codConductor, @codTerminal, @ciudadDestino, @precioViaje);

-- Capturamos el ID

SET @nuevoCodViaje = SCOPE\_IDENTITY();

COMMIT TRANSACTION;

-- 2) Inserción remota en Quito

INSERT INTO [VLADIMIRJON].Terminal\_Quito.dbo.ViajeTiempo

(cod\_viaje, hora\_viaje, fecha\_viaje)

VALUES

(@nuevoCodViaje, @horaViaje, @fechaViaje);

END;

GO

USE Terminal\_Ibarra;

GO

DECLARE @nuevoID INT;

EXEC dbo.InsertarViajeCompleto

@codVehiculo = 7,

@codConductor = 3,

@codTerminal = 2,

@ciudadDestino = N'Manta',

@precioViaje = 55.00,

@horaViaje = '16:45',

@fechaViaje = '2025-08-02',

@nuevoCodViaje = @nuevoID OUTPUT;

PRINT 'ID generado localmente: ' + CAST(@nuevoID AS VARCHAR(10));

El procedimiento almacenado InsertarViajeCompleto permite registrar un nuevo viaje asegurando la consistencia de la información entre dos bases de datos: la local (Ibarra) y la remota (Quito).

Primero, el procedimiento inserta los datos principales del viaje (código de vehículo, código de conductor, código de terminal, ciudad de destino y precio del viaje) en la tabla ViajeInformacion\_Ibarra de la base de datos local. Una vez insertados los datos, se obtiene el identificador único generado para el viaje utilizando la función SCOPE\_IDENTITY() y se almacena en el parámetro de salida @nuevoCodViaje.

Luego, utilizando este identificador único, el procedimiento realiza una segunda inserción en la base de datos remota de Quito, específicamente en la tabla ViajeTiempo. En esta tabla se registra el código del viaje junto con la hora y la fecha correspondiente.

De esta manera, se logra que ambas bases de datos mantengan una relación consistente y sincronizada sobre los viajes registrados, utilizando el mismo identificador para facilitar su seguimiento y administración en ambas sedes.

**Controlador Donde se van a obtener los datos del Viaje**

