Documentación Técnica: Registro de Usuarios y Diseño de Tablas

Este documento explica el funcionamiento del registro de usuarios en el programa y detalla el diseño de las tablas de la base de datos, con énfasis en los conceptos clave de claves primarias y foráneas, integridad referencial, y buenas prácticas en la gestión de bases de datos.

1. Registro de Usuarios

Flujo del Registro

El proceso de registro de usuarios consta de los siguientes pasos:

1. **Validación de Datos**:
    a. Se asegura que los campos requeridos (usuario, correo, contraseña, aceptación de términos) estén completos.
    b. Se valida el formato del correo electrónico y que la contraseña cumpla con los criterios de seguridad.
2. **Verificación de Existencia**:
    a. Se consulta la tabla Usuario para verificar que el nombre de usuario y correo electrónico no existan previamente.
3. **Inserción en la Base de Datos**:
    a. Se utiliza una sentencia SQL parametrizada para insertar un nuevo registro en la tabla Usuario.

Claves Foráneas y Relaciones

El sistema utiliza claves foráneas para relacionar a los usuarios con otras entidades, como transacciones y activos. Esto permite mantener la integridad referencial y garantizar que no existan datos huérfanos.

2. Diseño de Tablas

Tabla Usuario

Define a los usuarios del sistema.

```sql
CREATE TABLE IF NOT EXISTS Usuario (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(50) NOT NULL,
    acepta_terminos BOOLEAN NOT NULL
);
```

- **Claves y Constraints**:
  - id: Clave primaria, generada automáticamente.
  - user y email: Deben ser únicos.

Tabla Activos

Representa los activos financieros asociados a cada usuario.

```sql
CREATE TABLE IF NOT EXISTS Activos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    sigla VARCHAR(3) NOT NULL,
    cantidad REAL NOT NULL,
    tipo VARCHAR(12) NOT NULL,
    usuario_id INTEGER NOT NULL,
    FOREIGN KEY (usuario_id) REFERENCES Usuario(id) ON DELETE CASCADE
);
```

- **Relaciones**:
  - La columna usuario_id es una clave foránea que referencia a Usuario(id).
  - Si un usuario es eliminado, sus activos también se eliminan gracias a la restricción ON DELETE CASCADE.

Tabla Transaccion

Almacena las transacciones realizadas por los usuarios.

```sql
CREATE TABLE IF NOT EXISTS Transaccion (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    resumen VARCHAR(100) NOT NULL,
    fecha_hora VARCHAR(79) NOT NULL,
    usuario_id INTEGER NOT NULL,
    FOREIGN KEY (usuario_id) REFERENCES Usuario(id) ON DELETE CASCADE
);
```

- **Relaciones**:
    - La columna usuario_id es una clave foránea que referencia a Usuario(id).
    - Se garantiza que las transacciones estén asociadas a un usuario válido.

Tabla Moneda

Contiene información sobre las monedas disponibles en el sistema.

```sql
CREATE TABLE IF NOT EXISTS Moneda (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre VARCHAR(15) NOT NULL,
    sigla VARCHAR(3) NOT NULL UNIQUE,
    valor REAL NOT NULL,
    tipo VARCHAR(12) NOT NULL,
    volatilidad REAL,
    stock REAL
);
```

- **Notas**:
    - La columna sigla debe ser única para evitar duplicados.

3. Relaciones entre Tablas y Buenas Prácticas

Relaciones Implementadas

- **Usuario → Activos**: Un usuario puede tener varios activos, pero cada activo pertenece a un único usuario.
- **Usuario → Transaccion**: Un usuario puede tener varias transacciones asociadas.

Integridad Referencial

El uso de claves foráneas garantiza:

- Que no se puedan insertar registros huérfanos en Activos o Transaccion.
- La eliminación en cascada asegura que los datos relacionados sean eliminados automáticamente cuando se elimina un usuario.

Constraints y Validaciones

- **Unicidad**:
  - En Usuario, los campos user y email son únicos.
  - En Moneda, la columna sigla debe ser única.
- **Restricciones de tipo**:
  - Campos como password y resumen son NOT NULL para evitar datos incompletos.

Buenas Prácticas

1. **Uso de Sentencias Parametrizadas**:
   a. Evita vulnerabilidades como inyección SQL.
   b. Ejemplo:

```
String query = "INSERT INTO Usuario (user, email, password, acepta_terminos) VALUES (?, ?, ?, ?)";
try (PreparedStatement stmt = connection.prepareStatement(query)) {
    stmt.setString(1, user);
    stmt.setString(2, email);
    stmt.setString(3, password);
    stmt.setBoolean(4, aceptaTerminos);
    stmt.executeUpdate();
}
```

2. **Constraints Bien Definidos**:
   a. Garantizan la consistencia de los datos y facilitan el mantenimiento del sistema.
3. **Eliminación en Cascada**:
   a. Simplifica el manejo de dependencias entre tablas.

4. Inserciones de Ejemplo

Insertar Usuario

```
INSERT INTO Usuario (user, email, password, acepta_terminos)
VALUES ('john_doe', 'john@example.com', 'securepassword', 1);
```

Insertar Activo Asociado a un Usuario

```
INSERT INTO Activos (sigla, cantidad, tipo, usuario_id)
VALUES ('BTC', 0.5, 'Criptomoneda', 1);
```

Insertar Transacción Asociada a un Usuario

```
INSERT INTO Transaccion (resumen, fecha_hora, usuario_id)
VALUES ('Compra de Bitcoin', '2024-12-16 10:30:00', 1);
```

5. Resumen

- **Claves primarias**: Garantizan la unicidad de los registros en cada tabla.
- **Claves foráneas**: Establecen relaciones entre tablas y aseguran la integridad referencial.
- **Constraints**: Refuerzan la validez y consistencia de los datos.
- **Eliminación en cascada**: Facilita la gestión de dependencias entre registros relacionados.

Este diseño modular y relacional asegura que el sistema sea escalable, seguro y fácil de mantener.

Flujo de Inicio de Sesión (LoginController)

El proceso de inicio de sesión en el sistema consta de las siguientes etapas:

1. **Validación de Credenciales**
    a. Se utiliza el método Usuario.obtenerUsuario(Connection con, String username) para recuperar los datos del usuario desde la base de datos.
    b. Se valida si el usuario existe y si la contraseña ingresada coincide con la almacenada.

2. **Redirección**
    a. Si las credenciales son correctas, se cierra la ventana de inicio de sesión y se redirige al usuario a la pantalla principal (PantallaPrincipal), pasando su nombre de usuario.
    b. En caso de credenciales incorrectas, se muestra un mensaje de error al usuario mediante JOptionPane.
3. **Manejo de Errores**
    a. Los errores más comunes, como *usuario no encontrado*, *contraseña incorrecta* o *fallos de conexión a la base de datos*, son gestionados mostrando mensajes claros y específicos al usuario.

Flujo de Registro de Usuario (RegistroController)

El flujo de registro incluye las siguientes etapas clave:

1. **Validación de Datos de Entrada**
    a. Se verifica que los campos obligatorios como correo electrónico, nombre de usuario y contraseña estén completos.
    b. El formato del correo electrónico se valida mediante expresiones regulares (email.matches(...)).
2. **Verificación de Existencia**
    a. Se utiliza el método Usuario.obtenerUsuario(Connection con, String username) para confirmar si el usuario ya está registrado.
    b. Además, una consulta SQL parametrizada verifica que el correo electrónico no esté asociado a otra cuenta.
3. **Inserción en la Base de Datos**
    a. Si las validaciones son exitosas, se utiliza el método Usuario.insertarUsuario para registrar al nuevo usuario en la base de datos.
4. **Redirección Automática**
    a. Después de un registro exitoso, se redirige al usuario al formulario de inicio de sesión (PantallaLogin) para continuar el flujo.
5. **Manejo de Errores y Excepciones**
    a. Los errores se gestionan a través de excepciones personalizadas, como RegistroException, para tratar problemas como datos inválidos o errores en la conexión con la base de datos.

Aspectos Técnicos Relacionados con la Materia

1. **Manejo de Eventos en GUI**

a. Los controladores emplean ActionListeners combinados con expresiones lambda para gestionar eventos de usuario.
b. Por ejemplo, el método pantallaLogin.agregarAccionLogin(temp -> iniciarSesion()) asocia el clic del botón de inicio de sesión con la validación de credenciales, separando la lógica de la interfaz gráfica y promoviendo un diseño modular.

2. **Manejo de Excepciones**
    a. El controlador de registro utiliza bloques try-catch para manejar errores, encapsulando la validación en excepciones personalizadas como RegistroException.
    b. Ejemplo: Si el correo ingresado ya existe, se lanza la excepción con un mensaje claro que se muestra al usuario.
3. **Interacción con la Base de Datos**
    a. Los controladores utilizan PreparedStatement y ResultSet para realizar consultas y gestionar flujos de entrada/salida.
    b. El uso de consultas parametrizadas asegura protección contra inyecciones SQL, mejorando la seguridad del sistema.

PantallaLogin

1. **Interfaz Gráfica y Estilo**
    a. Implementada con componentes Swing como JTextField, JPasswordField, JButton y JPanel.
    b. Los elementos se organizan mediante BoxLayout, logrando un diseño limpio y alineado verticalmente.
2. **Personalización Visual**
    a. Los botones (btnLogin y btnRegistro) y campos de texto se personalizan con bordes y colores dinámicos mediante MouseAdapter. Por ejemplo:

```
boton.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(java.awt.event.MouseEvent evt) {
        boton.setBackground(color.brighter());
    }
    public void mouseExited(java.awt.event.MouseEvent evt) {
        boton.setBackground(color);
    }
});
```

3. **Integración con el Controlador**

a. Los eventos de usuario, como el clic en el botón de inicio de sesión, están vinculados a métodos específicos en el controlador (LoginController) mediante ActionListeners.

PantallaRegistro

1. **Interfaz Gráfica Completa para Registro**
    a. Incluye campos para correo electrónico, usuario y contraseña, además de una casilla de verificación para aceptar los términos y condiciones (JCheckBox).
    b. El diseño sigue un esquema basado en BoxLayout, garantizando consistencia visual con la pantalla de inicio de sesión.
2. **Validación de Campos desde la Vista**
    a. Proporciona métodos como getEmail(), getUsuario() y isTerminosAceptados() para que el controlador acceda fácilmente a los datos ingresados.
3. **Integración con el Controlador**
    a. El constructor de la vista inicializa un objeto RegistroController, estableciendo una conexión directa entre la lógica del registro y la interfaz.
    b. Los botones (btnRegistrar y btnVolver) utilizan ActionListeners para manejar eventos y ejecutar las acciones correspondientes.

Conclusión: Aspectos Técnicos Clave

1. **Diseño Modular**
    a. La separación de lógica y presentación mediante controladores y vistas asegura la escalabilidad y el mantenimiento del sistema.
2. **Seguridad y Eficiencia**
    a. Las consultas SQL parametrizadas protegen contra inyecciones y optimizan la interacción con la base de datos.
3. **Experiencia de Usuario Mejorada**
    a. Los efectos visuales dinámicos y los mensajes claros proporcionan una experiencia interactiva y amigable para el usuario.