RESÚMEN TEMA 2: MANEJO DE CONECTORES

Java, a través de JDBC simplifica la interacción con bases de datos relacionales tiene tres objetivos clave:

1. **Soporte de SQL:**
    - Facilita la construcción de sentencias SQL dentro de llamadas al API de Java.
2. **Experiencia de API's Existentes:**
    - Aprovecha la experiencia de los API's de bases de datos ya existentes.
3. **Simplicidad:**
    - Diseñado para ser lo más simple posible en la comunicación entre aplicaciones y motores de bases de datos.

Impedancia Objeto-Relacional en Resumen:

El **desfase objeto-relacional** representa la diferencia entre la programación orientada a objetos y las bases de datos.

Trata de los problemas como; la divergencia en lenguajes, tipos de datos y paradigmas de programación.

El desfase tiene que ver con el paradigma de orientación a objetos que no posee naturaleza matemática mientras que el modelo relacional si.

JDBC y ODBC no son lo mismo, son APIs distintas.

El API JDBC admite dos modelos de procesamiento para acceder a bases de datos: de dos y tres capas.

- **Modelo de Dos Capas:**
    - La aplicación se comunica directamente con la fuente de datos, requiriendo un conector JDBC específico.
    - Los comandos del usuario se envían y los resultados se devuelven directamente desde la base de datos.
    - Puede ser una configuración cliente/servidor, donde la máquina del usuario es el cliente y la máquina que alberga los datos es el servidor.
- **Modelo de Tres Capas:**
    - Los comandos se envían a una capa intermedia de servicios, la cual interactúa con la fuente de datos.
    - La fuente de datos procesa los comandos y envía los resultados de vuelta a la capa intermedia, que luego los devuelve al usuario.

Cómo conectarse a una base de datos:

```
// Establece la conexion
      Connection con = DriverManager.getConnection
                ("jdbc:odbc:miNombreDeBD",
                 "miLogin",
                 "miPassword" );

      // Ejecuta la consulta
      Statement stmt = (Statement) con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT nombre, edad FROM Empleados");

      // Procesa los resultados
      while (rs.next()) {
             String nombre = rs.getString("nombre");
             int edad = rs.getInt("edad");
      }
```

Hay 4 tipos de conectores JDBC:

TIPO 1: Un único driver para todas las BD. No abarca todas las plataformas.Incluido en el JDK.

TIPO 2: Conocido como API nativa.Convierte llamadas JDBC directamente a BD.Mejor rendimiento que tipo 1.

TIPO 3: Enfoque de 3 capas. Peticiones del JDBC van al servidor intermedio.Estructura de capas: APP-INTERMEDIARIA DE DATOS(no es un nombre oficial)-BD

TIPO 4: Protocolo nativo. Convierte las llamadas al protocolo de red del sistema gestor de la BD. Se usa para acceso intranet. No hay capa intermediaria. No necesita software adicional. Requiere un driver por cada BD.

POOL DE CONEXIONES

En entornos web, se recomienda mantener conexiones abiertas en lugar de abrir y cerrar individualmente

La versión 3.0 de JDBC ofrece un pool de conexiones que gestiona esto de manera transparente.

Al iniciar un servidor Java EE, el pool crea conexiones iniciales, y cuando se necesita una conexión, la fuente de datos javax.sql.DataSource solicita al pool, que entrega una conexión lógica java.sql.Connection. Para cerrar la conexión, la fuente de datos vuelve a hablar con el pool.

Este enfoque mejora el rendimiento al evitar el costoso proceso de creación y destrucción de conexiones. Además, el pool ajusta dinámicamente el número de conexiones físicas según la demanda, creando o eliminando conexiones de forma automática.

Para conectarse a una base de datos con un pool de conexiones transparente a través de JNDI, el código básico sería:

```
// Inicialización del Contexto
javax.naming.Context ctx = new InitialContext();
dataSource = (DataSource) ctx.lookup("java:comp/env/jdbc/Basededatos");

// Obtención de una conexión lógica
connection = dataSource.getConnection();

// Cierre de la conexión lógica al finalizar la operación
connection.close();
```

OPERACIONES: EJECUCIÓN DE CONSULTAS

Para operar con una base de datos, la aplicación debe seguir estos pasos:

1. Cargar el driver necesario para entender el protocolo de la base de datos.
2. Establecer una conexión con la base de datos.
3. Enviar consultas SQL y procesar los resultados.
4. Liberar los recursos al finalizar.
5. Gestionar posibles errores.

Se pueden emplear diferentes tipos de sentencias en JDBC:

- Statement: Para sentencias sencillas en SQL.
- PreparedStatement: Para consultas preparadas con parámetros.
- CallableStatement: Para ejecutar procedimientos almacenados.

El API JDBC clasifica las consultas en dos tipos:

- Consultas: SELECT.
- Actualizaciones: INSERT, UPDATE, DELETE, sentencias DDL.

Tipos de Consultas:

La clase **Statement** tiene los métodos **executeQuery** y **executeUpdate** para realizar consultas y actualizaciones en SQL

El método **next** se utiliza para avanzar el cursor, mientras que los métodos **get** se emplean para obtener valores de las columnas de un registro. Existe un método **get** específico para cada tipo básico de Java y para las cadenas.

Las **PreparedStatement** son consultas **precompiladas** que se usan para optimizar consultas que se usan muy a menudo.

Por ejemplo, para realizar una consulta de un medicamento que tenga un código determinado, haríamos la consulta siguiente:

```
PreparedStatement pstmt = con.preparedStatement("SELECT * from medicamentos WHERE codigo
= ? ");
```

Procedimientos almacenados:
Un procedimiento almacenado es un subprograma almacenado en la base de datos, compatible con sistemas como **MySQL y Oracle.** Pueden ser procedimientos o funciones, con un nombre, una lista de parámetros y sentencias SQL.

Transacciones:

Tienen la característica de poder deshacer los cambios efectuados en las tablas si no se han podido realizar todas las operaciones.

Las BD que soportar transacciones son mucho más seguras y fáciles de recuperar.

Al ejecutar una **transaccion**, el motor de la BD garantiza:
ATOMICIDAD,CONSISTENCIA,AISLAMIENTO Y DURABILIDAD (**ACID**)

Una transacción puede concluir de dos maneras: mediante **COMMIT**, si se ejecuta correctamente, lo que aplica los cambios a la base de datos, o mediante **ROLLBACK** en caso de fallo, deshaciendo todos los cambios realizados hasta ese punto.

Por defecto, al menos en MySQL en una conexión trabajamos en modo **autocommit** con valor **true**. Eso significa que cada consulta es una transacción en la base de datos.

Por tanto, si queremos definir una transacción de varias operaciones, estableceremos el modo **autocommit** a **false** con el método **setAutoCommit** de la clase Connection.
Las BD consumen muchos recursos por lo que conviene cerrarlas con el método "**close**"

Tambien es conveniente cerrar las consultas **Statement** y **PreparedStatement**.

```
Connection con = null ;
try {

    con = DriverManager.getConnection("jdbc:odbc:admdb");
    System.out.println("Conexión realizada con éxito.");
    // Aquí hacemos lo que necesitemos
}
catch( SQLException e ) {

    // Aquí haríamos lo necesario para gestionar la excepción SQL
}

finally {
      // Si hay conexión la cerramos
    if( con != null ) {
        try { con.close(); }
    catch( SQLException e ) {
         System.out.println(e.getMessage());
     }
    }
}
```