**1. ¿Por Qué Hablar con la Base de Datos? (Introducción - Página 2)**

* Tu programa hace cosas, pero necesita guardar información de forma permanente (que no se borre al cerrarlo). Las bases de datos son el sitio perfecto para guardar y organizar grandes cantidades de datos (clientes, productos, etc.).
* **El "Desfase" (Páginas 3-4):** Tu programa piensa en "Objetos" (un objeto Cliente, con sus datos dentro). La base de datos piensa en "Tablas" (filas y columnas). Son formas distintas de organizar la información. Necesitas una manera de traducir entre tu objeto y las filas de la tabla, y viceversa. Esto es el "desfase objeto-relacional", ¡una pequeña molestia que hay que solucionar!

**2. Las Herramientas para Hablar: JDBC y Drivers**

* **El Problema:** Hay muchas bases de datos diferentes (MySQL, Oracle, etc.), y cada una habla un dialecto un poco diferente. Tu programa Java necesita hablar con cualquiera.
* **La Solución (Páginas 5-9):**
  + **JDBC (Java Database Connectivity):** Es el **lenguaje estándar** que usa Java para hablar con bases de datos relacionales. Es como si Java supiera hablar un idioma universal de bases de datos.
  + **Driver:** Es el **traductor** específico para CADA base de datos. Si quieres hablar con MySQL, necesitas el Driver de MySQL. Si quieres hablar con Oracle, necesitas el Driver de Oracle. El Driver convierte el lenguaje JDBC estándar de Java al dialecto que entiende esa base de datos en concreto.
  + **Funcionamiento:** Tu Programa Java -> JDBC (Lenguaje Estándar) -> Driver Específico (Traductor) -> Base de Datos (MySQL, Oracle...).

**3. El Kit de Herramientas JDBC (Clases Clave - Página 14)**

Para hablar con la base de datos usando JDBC, Java te da varias "herramientas" (clases e interfaces):

* **DriverManager:** La herramienta para **conseguir la conexión** con la base de datos. Es como el directorio telefónico de drivers.
* **Connection:** Representa la **línea de teléfono abierta** con la base de datos. La pides a DriverManager. ¡Es tu enlace directo!
* **Statement:** Una herramienta para **mandar órdenes simples** (sentencias SQL) a la base de datos. La pides a la Connection.
* **PreparedStatement:** Una herramienta **más avanzada** para mandar órdenes con partes que cambian (¡y más segura!). La pides a la Connection. (¡Importante para el examen!)
* **ResultSet:** Es como la **hoja de resultados** que te devuelve la base de datos cuando le pides información (con SELECT).

**4. Los Pasos para Hablar con la Base de Datos (Páginas 15-21)**

Aquí está la secuencia que debes seguir en tu código:

1. **Decir a Java con Quién Vas a Hablar (Cargar Driver - Página 18):** Antes se hacía con Class.forName(), pero ahora los drivers modernos se cargan solos. ¡Menos trabajo para ti!
2. **Establecer la Comunicación (¡La Llamada! - Página 18):** Pedir la línea de teléfono (Connection). Usas DriverManager.getConnection(). Necesitas darle 3 datos clave:
   1. **URL:** La "dirección" de la base de datos (ej: jdbc:mysql://localhost:3306/nombre\_bd).
   2. **Usuario:** Tu nombre de usuario para entrar a la BD.
   3. **Contraseña:** Tu clave.
3. **Preparar la Orden (¡El Papel para la Orden! - Página 19):** Decidir si usas Statement (simple) o PreparedStatement (con partes variables, ¡mejor!). Le pides a la Connection que te prepare uno:
   1. Connection.createStatement() -> te da un Statement.
   2. Connection.prepareStatement("tu SQL con ?") -> te da un PreparedStatement.
4. **Enviar la Orden (¡A la BD! - Página 19):** Usas el Statement o PreparedStatement para mandar la sentencia SQL.
   1. Si es un SELECT (pedir datos): usas .executeQuery(). ¡Te devuelve un ResultSet!
   2. Si es un INSERT, UPDATE, DELETE (cambiar datos): usas .executeUpdate(). Te devuelve cuántas filas cambiaron (un número int).
5. **Leer los Resultados (Si es SELECT - Páginas 20, 28-30):** Si hiciste un SELECT, tienes un ResultSet. Es como una tabla.
   1. Usas .next() en un bucle while: while(rs.next()). next() va a la siguiente fila y es true si hay otra fila, false si ya no hay más.
   2. Dentro del bucle, usas .getXXX("nombre\_columna") (o .getXXX(numero\_columna)) para sacar los datos de cada columna de la fila actual. getXXX significa getString, getInt, getDouble, etc., según el tipo de dato.
6. **¡Limpiar Todo! (Liberar Recursos - Página 21):** **¡ESTO ES SÚPER IMPORTANTE PARA EL EXAMEN!** Cuando terminas, tienes que cerrar la "línea de teléfono" y las "órdenes" que preparaste. Si no, dejas cosas abiertas en el sistema.
   1. Debes cerrar: el ResultSet (si lo usaste), el Statement o PreparedStatement, y la Connection.
   2. **La Mejor Forma (¡Úsala en el examen!):** Usar **try-with-resources**. Pones la creación de la Connection, el Statement/PreparedStatement y el ResultSet dentro de los paréntesis del try. Java se encarga automáticamente de cerrarlos al terminar el bloque try, ¡incluso si hay un error!

**Ejemplo Básico de Conexión y Consulta (usando try-with-resources)**

Java

Plain Textimport java.sql.Connection;import java.sql.DriverManager;import java.sql.Statement;import java.sql.ResultSet;import java.sql.SQLException; // Importa para manejar los errores de BDpublic class EjemploConexionBasico {  
  
 // \*\*\* Configura estos datos para tu base de datos MySQL \*\*\*static final String URL\_BD = "jdbc:mysql://localhost:3306/nombre\_de\_tu\_base\_de\_datos";  
 static final String USUARIO\_BD = "tu\_usuario";  
 static final String CONTRASENA\_BD = "tu\_contrasena";  
  
 public static void main(String[] args) {  
  
 // Paso 2, 3, 4 y 5 (en este caso con Statement y SELECT)// Usamos try-with-resources: Connection, Statement y ResultSet se cerrarán solostry (Connection conexion = DriverManager.getConnection(URL\_BD, USUARIO\_BD, CONTRASENA\_BD); // Paso 2: Conectar Statement stmt = conexion.createStatement(); // Paso 3: Crear Statement ResultSet resultados = stmt.executeQuery("SELECT NIF, Nombre, Salario FROM Empleados")) // Paso 4: Ejecutar SELECT { System.out.println("Conexión exitosa. Datos de empleados:");  
  
 // Paso 5: Leer resultados (ResultSet)while (resultados.next()) { // rs.next() = ¿Hay otra fila?// Obtener datos de la fila actual String nif = resultados.getString("NIF"); // getString para texto String nombre = resultados.getString("Nombre");  
 double salario = resultados.getDouble("Salario"); // getDouble para números con decimales  
  
 System.out.printf("NIF: %s, Nombre: %s, Salario: %.2f%n", nif, nombre, salario);  
 }  
  
 } catch (SQLException e) { // Paso 6: Capturar errores de la base de datos  
 System.err.println("Error al conectar o consultar la base de datos:");  
 e.printStackTrace(); // Muestra detalles del error  
 }  
 // ¡Aquí, fuera del try, la conexión, el statement y el resultset ya se han cerrado!  
 }  
}

**5. El Duelo: Statement vs PreparedStatement (¡Pregunta Segura de Examen!)**

* **Statement (Páginas 22-23):** Para sentencias SQL que SIEMPRE son iguales.
  + **Peligro:** Si creas la sentencia juntando ("concatenando") texto que viene de fuera (por ejemplo, lo que escribe el usuario), ¡puedes sufrir un ataque de **Inyección SQL**! Alguien malintencionado podría escribir código SQL en una caja de texto y romper tu base de datos.
* **PreparedStatement (Páginas 24-27):** Para sentencias SQL que tienen partes que **cambian** (parámetros). Usas signos de interrogación ? donde van los valores variables.
  + **Cómo funciona:** Primero preparas la sentencia con los ?. Luego, *antes* de ejecutar, le dices a Java qué valor poner en cada ? usando métodos setXXX(numero\_del\_?, valor). ¡Los números de ? empiezan en **1**!
  + **Ventajas (¡Dí esto en el examen!):**
    - **¡Seguridad! Evita Inyección SQL:** Los valores se envían a la base de datos por separado de la sentencia, la base de datos sabe qué es sentencia y qué son datos.
    - **Eficiencia:** Si ejecutas la misma sentencia muchas veces con diferentes valores, la base de datos la "prepara" una vez y la ejecuta más rápido después.
* **Ejemplo: INSERT con PreparedStatement (¡Seguro y Recomendado!)**

Java

Plain Textimport java.sql.Connection;import java.sql.DriverManager;import java.sql.PreparedStatement; // ¡Importante!import java.sql.SQLException;  
public class EjemploInsertPreparedStatement {  
  
 // Configura tus datos de conexiónstatic final String URL\_BD = "jdbc:mysql://localhost:3306/nombre\_de\_tu\_base\_de\_datos";  
 static final String USUARIO\_BD = "tu\_usuario";  
 static final String CONTRASENA\_BD = "tu\_contrasena";  
  
 public static void main(String[] args) {  
  
 // Datos que queremos insertar (¡ahora NO los concatenamos en la SQL!) String nifNuevo = "98765432Z";  
 String nombreNuevo = "Pedro";  
 double salarioNuevo = 2100.00;  
  
 // Sentencia SQL con ? para los valores variables String sqlInsert = "INSERT INTO Empleados (NIF, Nombre, Salario) VALUES (?, ?, ?)"; // 3 interrogantes// Usamos try-with-resourcestry (Connection conexion = DriverManager.getConnection(URL\_BD, USUARIO\_BD, CONTRASENA\_BD);  
 PreparedStatement pstmt = conexion.prepareStatement(sqlInsert)) // Preparamos la sentencia { System.out.println("Conexión exitosa. Preparando INSERT...");  
  
 // Asignar valores a los interrogantes (¡Empieza en 1!) pstmt.setString(1, nifNuevo); // El primer ? (NIF) será el valor de nifNuevo pstmt.setString(2, nombreNuevo); // El segundo ? (Nombre) será el valor de nombreNuevo  
 pstmt.setDouble(3, salarioNuevo); // El tercer ? (Salario) será el valor de salarioNuevo  
  
 System.out.println("Ejecutando INSERT con PreparedStatement...");  
 int filasAfectadas = pstmt.executeUpdate(); // Ejecutamos ¡SIN pasar la SQL de nuevo!  
  
 System.out.println("INSERT completado. Filas afectadas: " + filasAfectadas);  
  
 } catch (SQLException e) {  
 System.err.println("Error al insertar datos:");  
 e.printStackTrace();  
 }  
 }  
}

Plain Text\* \*\*Nota:\*\* `executeUpdate()` se usa para `INSERT`, `UPDATE`, `DELETE` y también para sentencias DDL (`CREATE`, `ALTER`, `DROP` - Páginas 31-34).

**6. Reportes Bonitos: JasperReports (Páginas 40-50)**

* **Qué es:** Una herramienta para crear **informes con formato** a partir de los datos de tu base de datos. No es solo mostrar datos en consola, sino hacer facturas, listados con encabezados, etc.
* **Cómo funciona (Idea Sencilla):**
  + **Diseñas el Informe:** Usas un programa especial llamado **Jasper Studio** (Páginas 42-48). Es un editor visual donde dibujas cómo quieres que se vea el informe, pones los campos donde irán los datos, y le dices qué consulta SQL necesita para sacar esos datos. Esto crea un archivo con extensión .jrxml (es un XML).
  + **Lo Llenas con Datos desde Java:** En tu programa Java, usas las librerías de JasperReports. Tomas el archivo .jrxml, le das tu Connection JDBC (¡lo que aprendiste antes!) y le dices "llena este informe con los datos de esta conexión". JasperReports ejecuta la consulta SQL que definiste en el .jrxml, pone los datos en el diseño, y crea el informe (Página 49).
  + **Lo Exportas:** Una vez "lleno", puedes exportar el informe a diferentes formatos (PDF, Excel, etc.).
* **Para el Examen:** Define JasperReports como una herramienta para hacer informes. Explica que tiene dos partes: Jasper Studio (para diseñar el *formato* y la *consulta*) y las librerías de Java (para *ejecutar* la consulta, *llenar* el diseño con los datos, y *exportar*).

**7. ¡No Olvides Cerrar y Capturar Errores!**

* **Cerrar:** Ya lo dijimos, pero es CRUCIAL. Usa try-with-resources. Garantiza que la conexión, el statement y el resultset se cierran siempre. Si no, ¡mala nota y problemas en tus programas!
* **Errores:** Cualquier fallo al hablar con la base de datos (mal usuario, mala contraseña, SQL incorrecto, la BD no está encendida) lanzará una **SQLException**. Tienes que capturarla con catch(SQLException e).

**Resumen para Triunfar en el Examen:**

1. **JDBC:** El estándar de Java para bases de datos SQL. Necesita un **Driver** específico.
2. **Herramientas Clave:**
   1. DriverManager: Consigue la Connection.
   2. Connection: La conexión activa.
   3. Statement: Para SQL simple y fijo (evita si hay variables).
   4. **PreparedStatement:** ¡Para SQL con ?! **Seguro (evita Inyección SQL)** y eficiente. Usa setXXX().
   5. ResultSet: Los resultados de un SELECT. Recórrelo con while(rs.next()) y lee con getXXX().
3. **Pasos:** Conectar -> Crear Statement/PreparedStatement -> Ejecutar (executeQuery o executeUpdate) -> (Si SELECT) Leer ResultSet -> **¡Cerrar Todo con try-with-resources!**
4. **Errores:** Captura SQLException.
5. **JasperReports:** Para hacer informes bonitos. Diseñas en Studio, ejecutas en Java.

¡Estudia estos puntos, entiende los ejemplos de código (especialmente la diferencia entre Statement y PreparedStatement y cómo usar try-with-resources), y estarás listo! ¡Mucha suerte!