Tema 2. Acceso a bases de datos relacionales.

Contenido

Introducción

Actualmente, las bases de datos relacionales siguen siendo uno de los sistemas de almacenamiento más extendidos, aunque otros sistemas se estén abriendo paso poco a poco.

Una base de datos relacional se puede definir, de una manera simple, como aquella que presenta la información en tablas con filas y columnas.

Una tabla o relación es una colección de objetos del mismo tipo (filas o tuplas).

En cada tabla de una base de datos se elige una clave primaria para identificar de manera unívoca a cada fila.

El sistema gestor de bases de datos Database Management System (DBMS) gestiona el modo en que los datos se almacenan, mantienen y recuperan.

JDBC (Java Database Connectivity), permite simplificar el acceso a bases de datos relacionales, proporcionando un lenguaje mediante el cual las aplicaciones pueden comunicarse con motores de bases de datos. Sun desarrolló este API para el acceso a bases de datos, con tres objetivos principales en mente:

- Ser un API con soporte de SQL: poder construir sentencias SQL e insertarlas dentro de llamadas al API de Java.
- Aprovechar la experiencia de los API's de bases de datos existentes.
- Ser lo más sencillo posible.

Conceptos claves:

Conector: API que permite a los programas de aplicación trabajar con bases de datos.

Desfase objeto-relacional: Los problemas que supone almacenar objetos en bases de datos relacionales, con estructuras basadas en tablas.

Driver JDBC: Biblioteca de software que proporciona una implementación para una base de datos particular de las interfaces definidas en la especificación JDBC, de forma que la permite interactuar con la base de datos.

Iterador: Mecanismo para acceder a los resultados de una consulta, con operaciones para navegar por los resultados de la consulta y recuperar los datos uno a uno.

Pool de conexiones: Conjunto de conexiones de una base de datos que se mantienen abiertas y a disposición de los procesos que puedan necesitarlas, lo que evita el retraso y la sobrecarga que supone la apertura de una nueva conexión cada vez que un proceso la necesita.

Desfase objeto-relacional

Las bases de datos relacionales no están preparadas para almacenar objetos.

Existe un desfase entre las construcciones usadas en las bbdd relacionales con los proporcionados en programación. Al guardar Objetos en una bd relacional se incrementa la complejidad del código para salvar la diferencia entre objetos y tablas.

El desfase objeto-relacional consiste en la diferencia de aspectos que existen entre la programación orientada a objetos y la base de datos. Estos aspectos se pueden presentar en cuestiones como:

- Lenguaje de programación: el programador debe conocer:
  - El lenguaje de programación orientado a objetos (POO).
  - El lenguaje de acceso a datos.
- Tipos de datos:
  - En las bases de datos relacionales siempre hay restricciones en el uso de tipos.
  - Mientras que la programación orientada a objetos utiliza tipos de datos más complejos.
- Paradigma de programación: en el proceso de diseño y construcción del software se tiene que hacer una traducción del modelo orientado a objetos de clases al modelo Entidad-Relación (E/R) puesto que el primero maneja objetos y el segundo maneja tablas y tuplas (o filas), lo que implica que se tengan que diseñar dos diagramas diferentes para el diseño de la aplicación.

Definición:

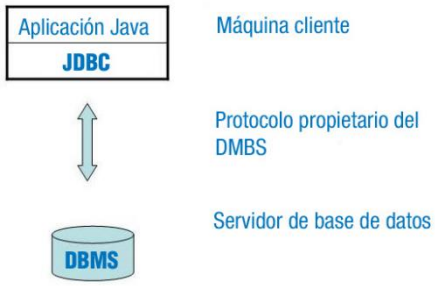
Definición: problemas que ocurren debido a las diferencias entre el modelo de datos en la BD y el lenguaje de POO.

A pesar de esto, ambos paradigmas pueden ser compatibles. Cada vez que se deben extraer o almacenar en una bd relacional algún objeto, se necesita realizar un mapeo. Esto lo veremos más adelante.

Arquitectura JDBC.

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

Modelo de dos capas:



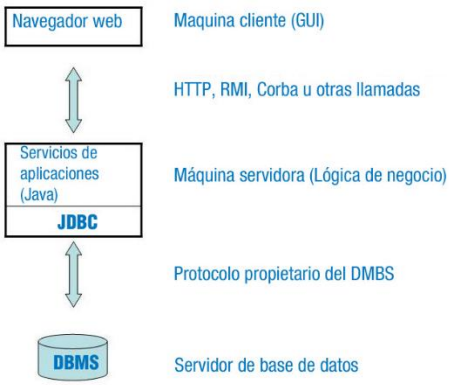
La aplicación se comunica directamente a la fuente de datos.

Necesita un conector JDBC que pueda comunicar con la fuente de datos específica a la que acceder.

Los comandos o instrucciones del usuario se envían a la base de datos y los resultados se devuelven al usuario.

La fuente de datos puede estar ubicada en otra máquina a la que el usuario se conecte por red. A esto se denomina configuración cliente/servidor, con la máquina del usuario como cliente y la máquina que aloja los datos como servidor.

Modelo de tres capas:



Los comandos se envían a una capa intermedia de servicios, la cual envía los comandos a la fuente de datos.

La fuente de datos procesa los comandos y envía los resultados de vuelta la capa intermedia, desde la que luego se le envían al usuario.

El API JDBC viene distribuido en dos paquetes:

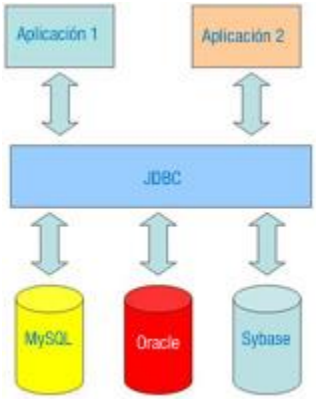
- java.sql, dentro de J2SE.
- javax.sql, extensión dentro de J2EE.

Conectores o drivers

Un conector o driver es un conjunto de clases encargadas de implementar los interfaces del API y acceder a la base de datos.

Un conector suele ser un fichero .jar que contiene una implementación de todas las interfaces del API JDBC.

El conector lo proporciona el fabricante de la base de datos o bien un tercero.



El código de nuestra aplicación no depende del driver, puesto que trabajamos mediante los paquetes java.sql y javax.sql.

JDBC ofrece las clases e interfaces para:

- Establecer una conexión a una base de datos.
- Ejecutar una consulta.
- Procesar los resultados.

Hay cuatro tipos de drivers JDBC: Tipo 1, Tipo 2, Tipo 3 y Tipo 4

ACTIVIDAD DE BÚSQUEDA: Busca en Internet información sobre los 4 tipos de conectores JDBC.

Operaciones básicas JDBC

Los sublenguajes SQL son:

- DML: para manipular los datos
- DDL: Para la definición de los datos.

Si pensamos en DML, sabemos que podemos diferenciar entre operaciones de:

- Consulta: SELECT, que se ejecutan con executeQuery() y nos devuleve una lista de filas en un ResultSet, sobre el que *iteraremos* para obtener los resultados de uno en uno.
- Modificación de datos: UPDATE, INSERT, DELETE, se ejecutan sobre executeUpdate, que devuelve el número de filas afectadas por la operación.

Consultas DDL: se ejecutan con execute()

Crear una conexión	Connection con = DriverManager.getConnection()
Crear una sentencia	Statement sentencia = con.createStatement()

Ejecutar sentencia SELECT	ResultSet rs = sentencia.executeQuery(sql)
	while (rs.next()){
	....
	}
	rs.close

Ejecutar sentencia	ResultSet rs = s.execute()

UPDATE, INSERT, DELETE	
sentencia. Close con.close	Cerrar sentencia Cerrar conexión

Actividad práctica 1: Crea una conexión jdbc con la base de datos.

Paso 1: Busca en el sitio oficial el conector JDBC adecuado para la conexión con MySQL.

Paso 2: Crea una conexión con la base de datos ies.sql

Conectar con la base de datos.

Actividad de búsqueda y reflexión:

1. Parámetros necesarios
2. Clases
3. Métodos
4. Sintaxis

Acceder a los resultados de una consulta.

No podemos perder de vista que la consulta devuelve registros y el lenguaje maneja objetos.

Los conectores nos permiten acceder a una base de datos relacional para hacer operaciones sobre ella.

Sabemos que una consulta SQL devuelve un conjunto de filas o registros. Los conectores nos permiten recuperar esos resultados fila a fila mediante un objeto que actúa como iterador o cursos.

- **ResultSet**
  - Encapsula el conjunto de resultados
  - Para obtener el valor de cada campo hay que usar el método getX("campo") correspondiente al tipo del valor SQL:
    - getInt ← INTEGER
    - getLong ← BIG INT
    - getFloat ← REAL
    - getDouble ← FLOAT
    - getBignum ← DECIMAL
    - getBoolean ← BIT
    - getString ← VARCHAR
    - getString ← CHAR
    - getDate ← DATE
    - getTime ← TIME
    - getTimesstamp ← TIME STAMP
    - getObject ← cualquier otro tipo
  - Para pasar al siguiente registro se usa el método **next**()
    - Devuelve false cuando no hay más registros

Si hacemos consultas Statement que permitan introducir variables o datos, podemos tener serios problemas.

Por ejemplo, imagino en la base de datos hospital que quiero obtener los datos del médico cuya especialidad pasamos por teclado.

String cod= “especialidad x”;

Statement sentencia = conexion.executeQuery(“Select * from medico where Especialidad = ‘”+cod+”’”);

Sustituyendo en la sentencia la variable de esta forma, podemos tener problemas:

- Seguridad: Una sentencia sql construida en tiempo de ejecución usando variables está expuesta a los ataques mediante inyección de SQL. De forma que un atacante podría modificar la consulta para obtener y modificar datos.
- Rendimiento: La consulta que se envía al gestor de bases de datos se compila antes de ejecutarse, luego se analiza y se crea un plan de ejecución para ella. Si entre una consulta y otra solo cambia el valor de algunas variables, se compilará cada consulta, para obtener siempre el mismo plan de ejecución.

Estos problemas podemos evitarlos usando consultas preparadas.

Consultas preparadas PreparedStatement

En las consultas preparadas es normal que intervengan variables, por ejemplo, recogidas de campos de un formulario o que vienen de una consulta previa o un fichero de datos.

Las sentencias preparadas permiten incluir marcadores en determinados puntos de la sentencia donde van los valores que se proporcionarán en el momento en que vaya a ejecutarse la sentencia.

Actividad de búsqueda: Busca información sobre los métodos de PreparedStatement.