ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 Materia: **Java Web**

TEMA: JDBC

Agenda

- Introducción a JDBC
- Crear una Conexión JDBC
- Crear Consultas con JDBC
- Leer un ResultSet

Introducción a JDBC (1)

- JDBC (***Java DataBase Conectivity***) es un API incluida en el lenguaje Java para el acceso a BD.
- Es un conjunto de clases e interfaces que brinda un completo API para la programación de BD.
- JDBC incluye un conjunto de clases abstractas e interfaces que deben ser implementadas (en lenguaje Java) por los fabricantes de drivers JDBC.
- A los drivers que son compatibles con JDBC se les llama ***JDBC-Compliant Driver***
- Por estar escrito 100% en Java, JDBC mantiene la característica de ser independiente de la plataforma.

Introducción a JDBC (2)

- Además por el modo en el que fue diseñado se podrá utilizar la misma aplicación con distintos motores de bases de datos (ej: MySQL, SQL Server, ORACLE).
- Para lograrlo no se tiene que cambiar código alguno, solamente cambiar el driver JDBC que se utiliza.
- La versión 1.0 de JDBC fue introducida en Java en la versión 1.1.x del JDK que se correspondía con la versión 1.1 del lenguaje.
- La versión 2.0 de JDBC se introdujo en la versión 1.2 y 1.3 del JDK que se corresponde con la versión 2 del lenguaje.
- La versión 2.o de JDBC aportó importantes mejoras al modelo inicial, agregando a JDBC varias funcionalidades.

Introducción a JDBC (3)

- Mejoras de JDBC v2.0 sobre JDBC v1.0:
  - Se amplió la interfaz ResultSet. En la v1.0 las recorridas sobre el resultado de una consulta estaban muy limitadas; en la v2.0 se pueden hacer recorridas en cualquier dirección (inclusive directamente a una posición determinada).
  - Se pueden modificar los datos de la BD simplemente actualizando sus valores en el ResultSet sin necesidad de introducir sentencias SQL.
  - Se introduce la posibilidad de enviar a la BD múltiples actualizaciones a realizar en bloque; lo que se llama actualizaciones en batch.
  - Se agregan a los tipos de datos soportados por JDBC los tipos avanzados que se incluyen en SQL 3 (por ejemplo BLOB).

Introducción a JDBC (4)

- En resumen, con JDBC se puede:
  - Establecer una conexión con la BD.
  - Enviar a la misma sentencias SQL a ser ejecutadas.
  - Manipular los datos que se encuentran almacenados.
  - Procesar los datos resultantes de la ejecución de sentencias SQL (ej: SELECT).

Introducción a JDBC (5)

- Algunas clases e interfaces de JDBC:
  - ***java.sql.Driver***: interfaz que los drivers de cada motor debe implementar para poder ser utilizado.
  - ***java.sql.DriverManager***: clase que gestiona los drivers; carga y selecciona el driver adecuado para abrir la conexión con una BD.
  - ***java.sql.Connection***: representa la conexión con una BD.
  - ***java.sql.Statement***: contenedor de consultas SQL; interfaz que tiene 2 subtipos:
    - ***PreparedStatement***: para precompilar consultas y simplemente pasarles parámetros.
    - ***CallableStatement***: para la ejecución de procedimientos almacenados.

Introducción a JDBC (6)

- Algunas clases e interfaces de JDBC (cont.):
  - ***java.sql.ResultSet***: interfaz que controla el acceso a los resultados de una consulta ejecutada con un Statement (desde la v2.0 de JDBC admite la actualización de los datos).
  - ***java.sql.SQLException***: excepción para el manejo de errores en la BD. Estos errores pueden ser tanto en la conexión, desconexión, obtención o modificación de datos.

Crear una Conexión JDBC (1)

- Lo primero es cargar el driver correspondiente al motor de bases de datos a utilizar (esto debe hacerse una única vez, al comienzo).
- Para la instanciación de dicho driver se utiliza Reflection, proporcionando el String con el nombre completo de la clase (incluyendo paquetes), de forma tal que no se genere una dependencia con dicha clase.
- Se necesitarán los siguientes imports:

  java.sql.SQLException

  java.sql.DriverManager

  java.sql.Connection

Crear una Conexión JDBC (2)

- Pasos a seguir:
  1. Cargar el driver adecuado.
  2. Crear una conexión que utilice ese driver.

```
try {
   Class.forName("com.mysql.jdbc.Driver").newInstance();
   Connection con = DriverManager.getConnection(
                "jdbc:mysql://urlhost:puertohost/base",
                  usuario, password);
   // Utilizar la conexión para crear sentencias...
}
catch (Exception e)
{...}
```

Crear una Conexión JDBC (3)

A. Se cargó el driver que va a ser utilizado, en este caso el Connector/J que sirve para MySQL y se creó una conexión.
B. Notar que se invocó a newInstance() aunque no es necesario en todos los casos (depende del fabricante del driver).
C. Es necesario capturar las posibles excepciones que puedan ocurrir, tanto al cargar el driver (ej: que éste no se encuentre) como al obtener la conexión (tener en cuenta que se obtiene ya abierta).

Crear una Conexión JDBC (4)

- Algunos métodos de la interfaz **Connection**:
  - ***void close()*** → cierra la conexión con la base de datos.
  - ***void commit()*** → lleva a cabo todos los cambios realizados dentro de una transacción (desde el último commit o rollback) y libera los bloqueos correspondientes.
  - ***Statement createStatement()*** → crea una instancia de Statement que representa una sentencia SQL.
  - ***CallableStatement prepareCall(String)*** → crea una sentencia específica para la invocación de procedimientos almacenados.
  - ***PreparedStatement prepareStatement(String)*** → crea una sentencia preparada y parametrizada.

Crear una Conexión JDBC (5)

- Algunos métodos de la interfaz **Connection** (cont.):
  - ***void rollback()*** → deshace todas las modificaciones realizadas en una transacción y libera todos los bloqueos.
  - ***void setAutoCommit(boolean)*** → establece el valor de autocommit de la conexión para tratar con las transacciones.

Crear consultas con JDBC (1)

- Para poder realizar una sentencia sobre la BD se utilizan tres objetos:
  - Una conexión abierta (de tipo Connection),
  - Una consulta (de tipo Statement o subtipo)
  - El resultado de ejecutar la consulta (de tipo ResultSet).
- Para ejecutar la sentencia se utilizan 2 métodos de **Statement**:
  - ***ResultSet executeQuery(String)*** → ejecuta una sentencia de selección y devuelve el resultado. Devuelve un ResultSet conteniendo el resultado de la ejecución de la consulta que se pasa como parámetro, y sólo se utiliza para ejecutar sentencias de selección

Crear consultas con JDBC (2)

- Para ejecutar la sentencia se utilizan 2 métodos de **Statement** (cont.):
  - ***int executeUpdate(String)*** → ejecuta una sentencia de actualización y devuelve la cantidad de registros afectados.
- **Connection**, **Statement** y **ResultSet** son interfaces.
- Los métodos ***close*** (para las 3 interfaces) liberan los recursos utilizados por los diferentes objetos (¡cuidado! arrojan excepciones).

Crear consultas con JDBC (3)

- **Transacciones**: Una transacción es un conjunto de sentencias SQL que se deben ejecutar todas juntas o no ejecutarse ninguna. Para esto se debe:
  1. Deshabilitar el ***autocommit*** (que realiza el commit automáticamente luego de cada ejecución de sentencia).
  2. Ejecutar todas las sentencias de la transacción.
  3. Invocar al ***commit***.
  4. Habilitar nuevamente el *autocommit* y si hubieron errores, invocar al ***rollback***.

Crear consultas con JDBC (4)

- **Sentencias Preparadas**: Una sentencia preparada es una sentencia predefinida y precompilada a la cual se le pasan parámetros cada vez que se la ejecuta y es de tipo **PreparedStatement** (interfaz).
- Su objetivo es facilitar la ejecución reiterada de la misma sentencia y mejorar el rendimiento de ejecución.
- Permite el pasaje de parámetros en cada ejecución. Los parámetros se definen mediante el signo de interrogación (?) y sus valores son cargados mediante métodos ***set...(<param>,<valor>)*** que setean el valor para un parámetro.

Crear consultas con JDBC (5)

- **Sentencias Preparadas** (cont.)
- Cuidado, el índice del primer parámetro es 1 (y no cero como ocurre frecuentemente con los índices en Java).
- Existen múltiples operaciones para establecer los valores para cada parámetro de la sentencia preparada:

```
void setDate(int índiceParámetro, Date valor)
void setDouble(int índiceParámetro, double valor)
void setFloat(int índiceParámetro, float valor)
void setInt(int índiceParámetro, int valor)
void setLong(int índiceParámetro, long valor)
void setShort(int índiceParámetro, short valor)
void setString(int índiceParámetro, String valor)
void setTime(int índiceParámetro, Time valor)
void setTimestamp(int índiceParámetro, Timestamp valor)
```

Crear consultas con JDBC (6)

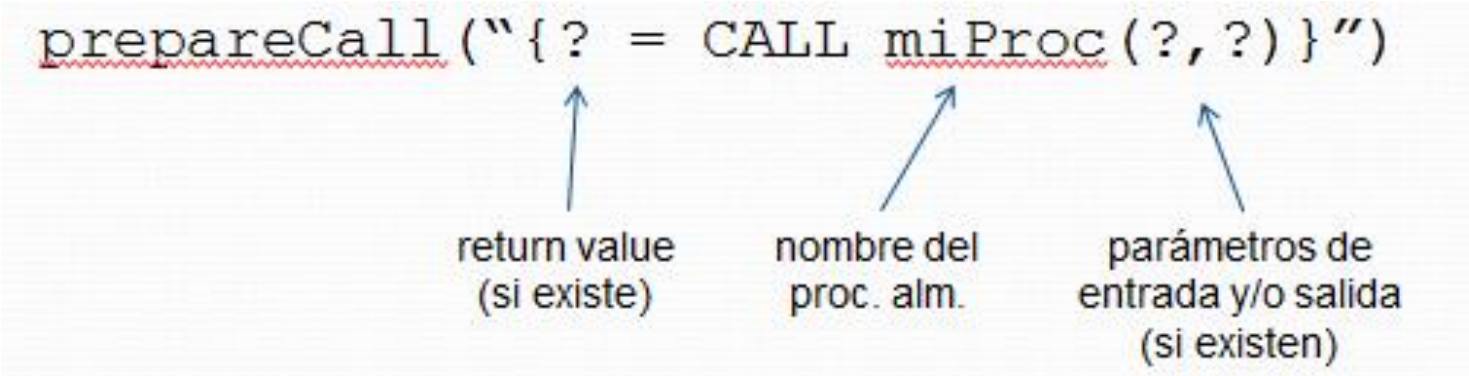
- **Procedimientos Almacenados**: un procedimiento almacenado (Stored Procedure) es un conjunto de sentencias SQL con parámetros tanto de entrada como de salida, alojado en el DBMS.
- Para invocar un SP se utiliza una sentencia de tipo **CallableStatement** (interfaz) subtipo de **PreparedStatement** (también interfaz).
- Al método **prepareCall(String)** de ***Connection*** se le pasa un texto cuya sintaxis depende de lo que el SP reciba y devuelva.
- Para su invocación se utilizan los métodos:

  executeQuery() cuando el SP consulta la BD

  executeUpdate() cuando el SP actualiza la BD

Crear consultas con JDBC (7)

- **Procedimientos Almacenados** (cont.):
- Un procedimiento almacenado puede retornar:
  - Un conjunto de resultados (ResultSet)
  - La cantidad de registros afectados por una sentencia de actualización (**int**)
  - El return value si se trata de una SF (Stored Function)
  - Parámetros de salida



Crear consultas con JDBC (8)

- **Procedimientos Almacenados** (cont.):
- Los parámetros de entrada se establecen de la misma forma que para las sentencias preparadas.
- Los parámetros de salida se establecen mediante el método ***registerOutParameter(<indice_param>,<tipo_SQL>)*** que recibe el índice del parámetro de salida y su tipo de dato (en términos de SQL*: java.sql.Types.<TIPO>*).
- Para obtener el valor del ***return value*** de una SF se debe registrar dicho parámetro como de salida. En MySql los SP no devuelve nada, si se necesita devolver un valor utilizar una SF o parámetros de salida.
- Para obtener el valor de cualquier parámetro de salida se utilizan los métodos **get…(<indice_param>)** existiendo una cantidad de métodos get, análoga a la vista para set.

Leer un ResultSet (1)

- Los objetos de tipo **ResultSet** contienen el resultado de una consulta de selección, por lo que interesará recorrer dicho resultado.
- Un *ResultSet* contiene un cursor que apunta al registro actual, por lo que para recorrer el resultado se debe mover el cursor y luego leer los campos del registro actual.
- Algunas operaciones de ResultSet v2.0:
  - ***boolean next()*** → mueve el cursor 1 lugar hacia adelante e indica si en la nueva posición existe algún resultado (true) o no (false).
  - ***boolean previous()*** → mueve el cursor hacia el registro anterior.

Leer un ResultSet (2)

- Algunas operaciones de ResultSet v2.0 (cont.) :
  - ***boolean first()*** → mueve el cursor hacia el primer registro.
  - ***boolean last()*** → mueve el cursor hacia el último registro.
  - ***boolean absolute(int pos)*** → mueve el cursor hacia la posición indicada.
  - ***boolean relative(int cant)*** → mueve el cursor la cantidad de registros indicada (ya sea hacia delante (cant > 0) o hacia atrás (cant < 0).
  - ***void beforeFirst()*** → mueve el cursor hacia antes del primer registro (por defecto cuando se retorna un ResultSet).

Leer un ResultSet (3)

- Algunas operaciones de ResultSet v2.0 (cont.) :
  - ***void afterLast()*** → mueve el cursor hacia después del último registro.
  - ***boolean isBeforeFirst()*** → indica si el cursor se encuentra antes del primer registro.
  - ***boolean isAfterLast()*** → indica si el cursor se encuentra luego del último registro.
  - ***boolean wasNull()*** → indica si el último valor obtenido es null en la BD. Por ejemplo: al obtener el valor de una columna int se obtiene 0 tanto si guarda 0 como si guarda null.

Leer un ResultSet (4)

- Una vez colocado el cursor en el registro deseado, las siguientes operaciones permiten obtener el valor de una determinada columna (ejemplo para int):

  int getInt(int índiceColumna)

  int getInt(String nombreColumna)

- Estas operaciones retornan el valor (en este caso un número entero) de la columna especificada (ya sea mediante su índice o mediante su nombre).
- Existen análogas operaciones para los restantes tipos de datos: String, double, float, long, short, boolean, etc.