Práctica 1: JDBC, Statements, PreparedStatements y Pool de Conexiones

Objetivos

- Conexiones a diversos SGBD: Oracle, HSQLdb
- Comparativa Statement/PreparedStatement
- - Uso de un Pool de conexiones y comparativa de tiempos

Preparación

1. Descarga el material del Campus Virtual (base de datos HSQLDB, script Oracle, drivers).
2. Crear un directorio workspace y arrancar Eclipse.
3. Arrancar el servidor HSQLdb (ejecutar data/startup).
4. Conectarse a Oracle y ejecutar el script para crear carworkshop.

Datos de conexión

Oracle
DRIVER = thin
IP = 156.35.94.98
PUERTO = 1521
Id Base de datos = desa19
USER = UO
PASS = password Oracle

HSQLDB
DRIVER = hsql
HOSTNAME = //localhost
USER = sa
PASS = ""

Ejercicios

1. **Reconstruir la base de datos Oracle y ejecutar script_create_carworkshop_Oracle.sql.**
    a. Ejecuta el commando

    ```
    SELECT 'DROP TABLE '||table_name||' CASCADE CONSTRAINTS;' FROM user_tables
    ```

    b. Copia el resultado y ejecútalo.
    c. A continuación, ejecuta el script **`script_create_carworkshop_Oracle.sql`**

2. Crear y configurar un proyecto Eclipse.

- Crea un proyecto en Eclipse para escribir los ejercicios.
- Configura el proyecto para hacer referencia a los dos drivers (Oracle, hsqldb)
    - Añadir los fichero .jar a una carpeta lib del proyecto.
    - Configura el build path de la aplicación.

3. Comparar el rendimiento de Statement vs. PreparedStatement.

El programa debe conectarse a la base de datos, actualizar unos datos y liberar los recursos. El objetivo es medir el tiempo empleado en esta tarea, realizándola de dos maneras diferentes: en primer lugar, mediante un objeto *Statement*, y posteriormente utilizando un objeto *PreparedStatement*.

Para que la diferencia de tiempo de ejecución sea significativa, la sentencia debe tener cierta complejidad y ejecutarse muchas veces, por ejemplo **100**, cambiando ligeramente cada vez.

Se sugiere la sentencia: **Actualizar la tabla TINVOICES, incrementando amount en el valor de la variable de control del bucle cuando el estado de sea 'PAID'.**

Para medir tiempos se puede utilizar `System.currentTimeMillis()` que nos devuelve la fecha/hora en número de milisegundos desde el 01/01/1970.

Ejecuta el programa conectándote a HSQLDB. Luego, ejecútalo nuevamente conectándote a Oracle.

Como esta segunda ejecución será un poco decepcionante, principalmente debido al retraso de la red, implementa una tercera versión que utilice procesamiento por lotes. Puedes encontrar un ejemplo aquí (https://www.baeldung.com/jdbc-batch-processing).

Se puede ver una discusión muy interesante al respecto de lo trabajado en esta práctica aquí.

4. Comprobar el coste de apertura y cierre de conexiones.

Una conexión a base de datos es un recurso costoso, tanto en tiempo como en memoria, ya que gestiona todas las comunicaciones entre la aplicación cliente y el servidor. El propósito de este ejercicio es medir y comparar el impacto de abrir y cerrar conexiones frente a reutilizarlas, desarrollando un programa que evalúe los tiempos de ejecución en dos escenarios distintos:

1. **Conexión dentro del bucle**

    - Ejecutar un bucle de 100 iteraciones.
    - En cada iteración:
        - Abrir una conexión.
        - Realizar una operación sencilla (por ejemplo, la misma que en el ejercicio anterior).
        - Cerrar la conexión.

2. **Conexión fuera del bucle**

    - Abrir una conexión antes de iniciar el bucle.
    - Ejecutar un bucle de 100 iteraciones en el que se realice la misma operación que en el caso anterior.
    - Cerrar la conexión al finalizar el bucle.

La comparación de ambos escenarios permitirá observar la diferencia de coste entre establecer una conexión repetidamente o reutilizar una conexión abierta durante varias operaciones.

5. Comprobar el beneficio de usar un pool de conexiones.

En este ejercicio utilizaremos el pool de conexiones c3p0, cuyos drivers se encuentran en la carpeta lib (descargados desde https://www.mchange.com/projects/c3p0/).

El objetivo de este ejercicio es **comparar el coste de crear conexiones físicas con la base de datos frente a la utilización de un pool de conexiones con la misma.**

Se accederá varias veces a una base de datos y en cada ocasión se realizará una operación. Como en los demás ejercicios, se implementarán dos escenarios distintos.

- **Sin pool de conexiones**: cada operación con la base de datos crea y cierra su propia conexión.
- **Con pool de conexiones**: se reutilizan conexiones gestionadas por un pool (c3p0), obteniéndolas y liberándolas para realizar cada operación.

El propósito es **medir los tiempos de ejecución** en ambos casos y analizar cómo el uso de un pool de conexiones mejora la eficiencia y el rendimiento del acceso a la base de datos.

Ejemplo de bucle sin pool	Ejemplo de bucle con pool

for (int i = 0; i < 1000; i++) {
  crearConexion();
  ejecutar sentencia();
  cerrarConexion();
}
	crearPoolConexiones();
	for (int i = 0; i < 1000; i++) {
	  getConexionFromPool();
	  ejecutar sentencia();
	  releaseConexion();
	}

En el segundo escenario, a las instrucciones de conexión y acceso a la base de datos, se le suma la creación y configuración de un pool de conexiones con las siguientes características (valores por defecto de la librería):

- Máximo de conexiones (maxPoolSize): 15
- Mínimo de conexiones (minPoolSize): 3
- Tamaño inicial del pool (initialPoolSize): 3

Ejemplo de inicialización de c3p0

A modo de referencia, se puede usar el siguiente extracto de código disponible en la web de c3p0 para crear un `PooledDataSource`:

```
DataSource ds_unpooled = DataSources.unpooledDataSource(URL, USER, PASS);
Map<String, Object> overrides = new HashMap<>();
/* Las siguientes tres instrucciones se pueden omitir si los valores que se
usan son los valores por defecto */
overrides.put("minPoolSize", 3);
overrides.put("maxPoolSize", 15);
overrides.put("initialPoolSize", 3);
DataSource ds_pooled = DataSources.pooledDataSource(ds_unpooled, overrides);

// Para obtener conexión del pool
Connection con = ds_pooled.getConnection();
```

Si la tarea fuese suficientemente costosa, sería todavía mejor dividirla en subtareas (Threads), cada uno de los cuales ejecutaría parte de las mismas.