Guía de Laboratorio
Haskell + SQLite (HDBC)

Curso: Programación Lógica y Funcional
Docente: Carlos R. P. Tovar

Resumen

Esta guía muestra paso a paso cómo conectar un programa Haskell a una base de datos SQLite usando el paquete `HDBC` con el backend `HDBC-sqlite3`. El laboratorio incluye: crear la base y la tabla, insertar registros, consultar, actualizar y eliminar. Se provee código completo y comandos para compilar y verificar en la línea de comandos.

Objetivos

- Conectar desde Haskell a una base de datos SQLite.
- Crear una tabla y realizar operaciones CRUD (Create, Read, Update, Delete).
- Interpretar resultados y verificar persistencia con la herramienta `sqlite3`.
- Practicar uso de `HDBC` y funciones básicas de manejo de bases de datos en Haskell.

Requisitos previos

- GHC y cabal instalados en las maquinas del laboratorio (en el laboratorio usamos GHC 7.8.3).
- Paquetes Haskell: `HDBC` y `HDBC-sqlite3` instalados. Comando sugerido (si el administrador lo permite):

```
cabal update
cabal install HDBC HDBC-sqlite3
```

- Herramienta de línea de comandos `sqlite3` para inspeccionar la base: `sqlite3 tienda.db`.
- Editor de texto o entorno para editar ficheros `.hs`.

Archivo principal: Main.hs

A continuación está el programa completo empleado en este laboratorio. Este archivo realiza: conexion, creación de tabla, inserción de 10 productos, consulta, actualización y eliminación.

Listing 1: Main.hs (programa completo)

```haskell
{-# LANGUAGE OverloadedStrings #-}

import Database.HDBC
import Database.HDBC.Sqlite3

-- Funcion principal
main :: IO ()
main = do
    putStrLn "Paso 1: Conectando a la base de datos..."
    conn <- connectSqlite3 "tienda.db"

    putStrLn "Paso 2: Creando la tabla productos si no existe..."
    run conn
        "CREATE TABLE IF NOT EXISTS productos (\
        \id INTEGER PRIMARY KEY AUTOINCREMENT, \
        \nombre TEXT NOT NULL, \
        \precio REAL NOT NULL, \
        \stock INTEGER NOT NULL)"
        []
    commit conn
    putStrLn "Tabla lista"

    putStrLn "Paso 3: Insertando productos de ejemplo..."
    insertarProductos conn
    commit conn

    putStrLn "Paso 4: Mostrando todos los productos..."
    mostrarProductos conn

    putStrLn "Paso 5: Actualizando un producto..."
    actualizarProducto conn 1 "Laptop Gamer" 3500.0 5
    commit conn
    mostrarProductos conn

    putStrLn "Paso 6: Eliminando un producto..."
    eliminarProducto conn 2
    commit conn
    mostrarProductos conn

    putStrLn "Paso 7: Cerrando la conexion..."
    disconnect conn
    putStrLn "Fin del programa"


-- Funcion para insertar varios productos
insertarProductos :: Connection → IO ()
insertarProductos conn = do
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
       ?, ?)"
        [toSql ("Laptop" :: String), toSql (2500.0 :: Double), toSql
           (10 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
```

```
      ?, ?)"
       [toSql ("Mouse" :: String), toSql (50.0 :: Double), toSql (100
          :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Teclado" :: String), toSql (120.0 :: Double), toSql
          (50 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Monitor" :: String), toSql (800.0 :: Double), toSql
          (20 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Impresora" :: String), toSql (600.0 :: Double), toSql
          (15 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Tablet" :: String), toSql (1500.0 :: Double), toSql
          (25 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Smartphone" :: String), toSql (2000.0 :: Double),
          toSql (30 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Cargador" :: String), toSql (80.0 :: Double), toSql
          (60 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Audifonos" :: String), toSql (300.0 :: Double), toSql
          (40 :: Int)]
    run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?,
      ?, ?)"
       [toSql ("Camara Web" :: String), toSql (400.0 :: Double),
          toSql (12 :: Int)]
    putStrLn "Productos insertados"

-- Funcion para mostrar productos
mostrarProductos :: Connection → IO ()
mostrarProductos conn = do
    putStrLn "Consultando productos..."
    rows <- quickQuery' conn "SELECT id, nombre, precio, stock FROM
      productos" []
    mapM_ imprimirFila rows

-- Imprimir una fila
imprimirFila :: [SqlValue] → IO ()
imprimirFila [sqlId, sqlNombre, sqlPrecio, sqlStock] = do
    putStrLn $ "ID: " ++ fromSql sqlId
             ++ " | Nombre: " ++ fromSql sqlNombre
             ++ " | Precio: " ++ show (fromSql sqlPrecio :: Double)
             ++ " | Stock: " ++ show (fromSql sqlStock :: Int)
imprimirFila _ = putStrLn "Fila con formato inesperado"
```

```haskell
-- Funcion para actualizar un producto
actualizarProducto :: Connection → Int → String → Double → Int →
   IO ()
actualizarProducto conn id nombre precio stock = do
    run conn "UPDATE productos SET nombre = ?, precio = ?, stock = ?
       WHERE id = ?"
        [toSql nombre, toSql precio, toSql stock, toSql id]
    putStrLn $ "Producto con id " ++ show id ++ " actualizado"

-- Funcion para eliminar un producto
eliminarProducto :: Connection → Int → IO ()
eliminarProducto conn id = do
    run conn "DELETE FROM productos WHERE id = ?" [toSql id]
    putStrLn $ "Producto con id " ++ show id ++ " eliminado"
```

Procedimiento de laboratorio (paso a paso)

1. **Preparar el entorno.**
   - Confirme que GHC y cabal están disponibles: `ghc -version` y `cabal -version`.
   - Confirme que los paquetes `HDBC` y `HDBC-sqlite3` están instalados. Si no, pida al administrador su instalación o ejecute:

     ```
     cabal update
     cabal install HDBC HDBC-sqlite3
     ```

   - Verifique que la utilidad `sqlite3` esté disponible: `sqlite3 -version`.
2. **Crear el archivo `Main.hs`.**
   - Abra el editor y pegue el código mostrado arriba.
   - Guarde como `Main.hs` en el directorio de trabajo.
3. **Compilar el programa.**

   ```
   ghc Main.hs -o Main.exe
   ```

   Si hay errores de compilación, copie y pegue el mensaje de error en el foro de la clase o pida ayuda.
4. **Ejecutar el programa.**

   ```
   ./Main.exe     (o Main.exe en Windows)
   ```

   Debe ver mensajes de progreso indicando cada paso (conectar, crear tabla, insertar, etc.). Al final, la base de datos `tienda.db` existirá en el directorio.
5. **Verificar con sqlite3 (opcional).**

```
sqlite3 tienda.db
sqlite> .tables
sqlite> SELECT * FROM productos;
sqlite> .schema productos
sqlite> .quit
```

6. **Probar funciones en GHCi (opcional para investigación).**

```
ghci Main.hs
Prelude> :load "Main.hs"
Prelude Main> conn <- connectSqlite3 "tienda.db"
Prelude Main> mostrarProductos conn
Prelude Main> disconnect conn
```

Nota: en GHCi algunas funciones requieren pasar la conexión como argumento.

Explicación de las operaciones SQL utilizadas

- `CREATE TABLE IF NOT EXISTS productos (...)`: crea la tabla si no existe.
- `INSERT INTO productos (nombre, precio, stock) VALUES (?, ?, ?)`: inserta un registro; los signos de interrogación son marcadores de parámetro en HDBC.
- `SELECT id, nombre, precio, stock FROM productos`: selecciona las columnas para mostrar.
- `UPDATE productos SET ... WHERE id = ?`: actualiza los datos de un registro dado su id.
- `DELETE FROM productos WHERE id = ?`: elimina un registro por id.

Salida esperada

Cuando ejecute el programa usted debe ver una salida similar a:

```
Paso 1: Conectando a la base de datos...
Paso 2: Creando la tabla productos si no existe...
Tabla lista
Paso 3: Insertando productos de ejemplo...
Productos insertados
Paso 4: Mostrando todos los productos...
Consultando productos...
ID: 1 | Nombre: Laptop | Precio: 2500.0 | Stock: 10
...
Paso 5: Actualizando un producto...
Producto con id 1 actualizado
...
Paso 6: Eliminando un producto...
Producto con id 2 eliminado
```

```
...
Paso 7: Cerrando la conexion...
Fin del programa
```

Tareas y ejercicios propuestos

1. Modifique el programa para pedir datos por teclado y agregar un producto ingresado por el usuario.
2. Agregue una función de búsqueda por nombre que muestre todos los productos que contengan una subcadena.
3. Implemente paginación en la consulta de productos (limitar y desplazar, usando `LIMIT` y `OFFSET`).
4. Cree una versión que exporte los productos a un archivo CSV.
5. Refactorice el código separando responsabilidades en múltiples módulos: `DB`, `Model`, `Main`.

Criterios de entrega

- Archivo fuente `Main.hs` o el repositorio con el proyecto.
- Captura de pantalla o salida de consola mostrando la ejecución completa del programa.
- Archivo `tienda.db` opcional si se solicita revisión local.

Resolución de problemas comunes

- **No encuentra `HDBC` o `HDBC-sqlite3`**: asegúrese que los paquetes están instalados con `cabal install`.
- **El programa se queda colgado en la conexión**: copie `sqlite3.dll` al mismo directorio que el ejecutable o verifique que la versión de la DLL coincide con la arquitectura de GHC (x86_64).
- **Errores de codificación al imprimir acentos o s0̆0edmbolos**: evite caracteres no ASCII en las cadenas o cambie la tabla de caracteres de la consola a UTF-8 con `chcp 65001`.
- **Problemas al abrir la base desde la CLI de sqlite3**: abra la base exacta con `sqlite3 tienda.db` y ejecute `.tables` para verificar su contenido.

Referencias

- Documentación HDBC: `https://hackage.haskell.org/package/HDBC`
- Documentación HDBC-sqlite3: `https://hackage.haskell.org/package/HDBC-sqlite3`
- Documentación SQLite: `https://www.sqlite.org/docs.html`
- Tutorial básico de SQL: cualquier texto introductorio sobre SQL (por ejemplo, "SQL Tutorial.[en] sqlite.org).