Curso: Programación Lógica y Funcional

Unidad 2: Programación funcional

Semana 08 - Sesión 16: Base de Datos dinámicas y externas

Docente: Carlos R. P. Tovar

Universidad Tecnológica del Perú

Dudas de la anterior sesión

INICIO
Objetivo de la sesión

Al final de la sesión, el estudiante será capaz de interactuar con bases de datos externas desde Haskell y manejar datos dinámicos usando técnicas funcionales.

Transición de conceptos

- **De lo estático a lo dinámico:** De listas en memoria a datos persistentes
- **Motivación:** Aplicaciones reales requieren almacenamiento persistente
- **Pregunta detonante:**
  "¿Cómo podemos mantener y consultar datos de manera persistente en un lenguaje funcional puro como Haskell?"

TRANSFORMACIÓN
Tipos de persistencia en Haskell

```
-- 1. Archivos de texto planos
readFile "datos.txt"
writeFile "salida.txt" contenido

-- 2. Bases de datos SQL (PostgreSQL, SQLite)
-- 3. Almacenamiento clave-valor
-- 4. Serialización binaria
```

Trabajando con archivos - Entrada/Salida

```
import System.IO

-- Lectura básica
leerBD :: FilePath -> IO String
leerBD archivo = readFile archivo

-- Escritura básica
guardarBD :: FilePath -> String -> IO ()
guardarBD archivo datos = writeFile archivo datos
```

```
-- Ejemplo de uso
main :: IO ()
main = do
  contenido <- leerBD "base_datos.txt"
  putStrLn $ "Datos: " ++ contenido
```

Estructura de datos dinámicos

```
-- Tipo de dato para representar registros
data Persona = Persona {
  nombre :: String,
  edad :: Int,
  email :: String
} deriving (Show, Read)

-- Base de datos como lista dinámica
type BaseDatos = [Persona]
```

```
-- Ejemplo de datos
personasEjemplo :: BaseDatos
personasEjemplo = [
  Persona "Ana" 25 "ana@email.com",
  Persona "Luis" 30 "luis@email.com"
  ]
```

Operaciones CRUD básicas

```
-- Create - Agregar persona
agregarPersona :: BaseDatos -> Persona -> BaseDatos
agregarPersona bd persona = persona : bd

-- Read - Buscar por nombre
buscarPorNombre :: BaseDatos -> String -> [Persona]
buscarPorNombre bd nombreBuscado =
  filter (\p -> nombre p == nombreBuscado) bd
```

Operaciones CRUD básicas

```
-- Update - Actualizar email
actualizarEmail :: BaseDatos -> String -> String -> BaseDatos
actualizarEmail bd nombreBuscado nuevoEmail =
  map (\p -> if nombre p == nombreBuscado
        then p { email = nuevoEmail }
        else p) bd

-- Delete - Eliminar persona
eliminarPersona :: BaseDatos -> String -> BaseDatos
eliminarPersona bd nombreBuscado =
  filter (\p -> nombre p /= nombreBuscado) bd
```

PRACTICA
Ejercicio guiado - Sistema de inventario

```
data Producto = Producto {
  idProducto :: Int,
  nombreProd :: String,
  precio :: Double,
  stock :: Int
} deriving (Show, Read)

type Inventario = [Producto]
```

```
-- Función para guardar inventario en archivo
guardarInventario :: FilePath -> Inventario ->
IO ()
guardarInventario archivo inventario =
  writeFile archivo (show inventario)


-- Función para cargar inventario desde
archivo
cargarInventario :: FilePath -> IO Inventario
cargarInventario archivo = do
  contenido <- readFile archivo
  return (read contenido)
```

Integración con SQLite (HDBC)

```
import Database.HDBC
import Database.HDBC.Sqlite3

-- Conexión a base de datos
conectarBD :: IO Connection
conectarBD = connectSqlite3 "mi_base.db"

-- Consulta de productos
consultarProductos :: Connection -> IO [[SqlValue]]
consultarProductos conn =
  quickQuery' conn "SELECT * FROM productos" []
```

```
-- Insertar nuevo producto
insertarProducto :: Connection -> String -> Double -> Int -> IO ()
insertarProducto conn nombre precio stock = do
  run conn "INSERT INTO productos (nombre, precio, stock) VALUES (?, ?, ?)"
      [toSql nombre, toSql precio, toSql stock]
  commit conn
```

Aplicación práctica - Menú interactivo

```
menuPrincipal :: IO ()
menuPrincipal = do
  putStrLn "=== SISTEMA DE INVENTARIO ==="
  putStrLn "1. Ver todos los productos"
  putStrLn "2. Agregar producto"
  putStrLn "3. Buscar producto"
  putStrLn "4. Actualizar stock"
  putStrLn "5. Salir"
  putStr "Seleccione una opción: "
  opcion <- getLine
  procesarOpcion opcion
```

```
procesarOpcion :: String -> IO ()
procesarOpcion "1" = verProductos >> menuPrincipal
procesarOpcion "2" = agregarProductoMenu >> menuPrincipal
procesarOpcion "5" = putStrLn "Saliendo del sistema..."
procesarOpcion _   = putStrLn "Opción inválida" >> menuPrincipal
```

CIERRE
Manejo de errores y transacciones

```
-- Manejo seguro de operaciones de BD
operacionSegura :: IO ()
operacionSegura = do
  conn <- conectarBD
  result <- try (insertarProducto conn "Nuevo" 10.0 5) :: IO (Either SomeException ())
  case result of
    Left ex -> putStrLn $ "Error: " ++ show ex
    Right _ -> putStrLn "Operación exitosa"
  disconnect conn
```

```
-- Transacción atómica
transaccionCompleja :: Connection -> IO Bool
transaccionCompleja conn = do
  commit conn
  result <- try $ do
    run conn "UPDATE productos SET stock = stock - 1 WHERE id = 1" []
    run conn "INSERT INTO ventas (producto_id, cantidad) VALUES (1, 1)" []
    commit conn
  case result of
    Left _ -> rollback conn >> return False
    Right _ -> return True
```

Patrones y mejores prácticas

- **Separación de concerns:** Lógica de negocio vs. acceso a datos
- **Manejo de errores:** Usar Either o Maybe para operaciones fallibles
- **Pureza:** Mantener las funciones puras cuando sea posible
- **Testing:** Mocking de bases de datos para pruebas unitarias

Conclusiones

- Haskell puede interactuar efectivamente con bases de datos externas
- Las técnicas funcionales se aplican bien al manejo de datos persistentes
- La inmutabilidad ayuda en la consistencia de los datos

Tarea

- Implementar un CRUD completo para una entidad simple
- Experimentar con diferentes backends (archivos, SQLite)
- Documentar el manejo de errore

Librerías recomendadas

-- Para SQLite
-- dependencia: HDBC, HDBC-sqlite3

-- Para PostgreSQL
-- dependencia: postgresql-simple

-- Para MongoDB
-- dependencia: mongoDB

-- ORM funcional
-- dependencia: persistent, esqueleto

Ejercicios de práctica:

- Sistema de registro de estudiantes con persistencia en archivo
- Conexión a SQLite con consultas parametrizadas
- Aplicación de agenda telefónica con búsqueda avanzada
- Sistema de reservas con transacciones atómicas

Universidad
Tecnológica
del Perú