Universidad Tecnológica de Panamá

Facultad de Ingeniería de Sistemas Computacionales

Licenciatura en Desarrollo y Gestión de Software

Base de Datos II

9GS-121

Título:

TRABAJO DE RECUPERACIÓN

Nombres y **Cédulas**:

Heath Katrina 8-1019-1329

Troetsch David 8-1009-694

Profesora:

Gionella L. Araujo

Fecha:

19/06/2024

ÍNDICE

Contenido

INTRODUCCIÓN

La gestión de bases de datos SQL constituye una parte esencial del desarrollo de software, proporcionando la infraestructura necesaria para el almacenamiento, acceso y manipulación de datos de manera eficiente. Este trabajo abarca desde la creación y administración de tablas, hasta la inserción, actualización y eliminación de registros. Además, se explorará cómo las herramientas y técnicas avanzadas, como las API de SQL y los ORM, facilitan la integración de estas capacidades en aplicaciones modernas, garantizando la seguridad y optimización del rendimiento. A través de una comprensión profunda de estas prácticas, los desarrolladores pueden construir aplicaciones robustas y escalables, capaces de manejar grandes volúmenes de datos y satisfacer las necesidades de los entornos empresariales actuales.

Colecciones y registros

Cuando se trabaja con bases de datos SQL, es común que surjan los términos "colecciones" y "registros". A continuación, se explica cada uno de estos conceptos y cómo se relacionan con SQL:

Colecciones

En el contexto de bases de datos, las "colecciones" suelen referirse a un conjunto de elementos similares. En las bases de datos NoSQL, como MongoDB, una colección es un conjunto de documentos. Sin embargo, en SQL, el término equivalente más cercano sería una tabla. Una tabla en SQL es una colección de filas (registros) que comparten la misma estructura.

Ejemplo en SQL

```
CREATE TABLE empleados (
    id INT PRIMARY KEY,
    nombre VARCHAR(50),
    posicion VARCHAR(50),
    salario DECIMAL(10, 2)
);
```

En este caso, empleados es una tabla que actúa como una colección de registros sobre los empleados.

Registros

Un "registro" en SQL se refiere a una fila de datos dentro de una tabla. Cada registro contiene datos para las columnas definidas en la tabla.

Ejemplo en SQL

```
INSERT INTO empleados (id, nombre, posicion, salario)
VALUES (1, 'Juan Pérez', 'Desarrollador', 50000.00);
```

Aquí, estamos insertando un registro en la tabla empleados con información sobre un empleado específico.

Operaciones Comunes con Colecciones y Registros en SQL

Crear una tabla (colección)

```
CREATE TABLE productos (
    id INT PRIMARY KEY,
    nombre VARCHAR(100),
    precio DECIMAL(10, 2),
    stock INT
);
```

Insertar un registro en una tabla

```
INSERT INTO productos (id, nombre, precio, stock)
VALUES (1, 'Laptop', 799.99, 50);
```

Seleccionar registros de una tabla

```
SELECT * FROM productos;
```

Actualizar un registro en una tabla

```
UPDATE productos
SET precio = 749.99
WHERE id = 1;
```

Eliminar un registro de una tabla

```
DELETE FROM productos
WHERE id = 1;
```

Consultas y Manejo de Datos

Las consultas SQL permiten seleccionar, insertar, actualizar y eliminar registros en las tablas. Aquí hay algunos ejemplos adicionales de cómo se manejan estas operaciones:

Selección de registros con condiciones

```
SELECT nombre, precio FROM productos WHERE stock > 10;
```

Agrupar registros y aplicar funciones de agregación

```
SELECT posicion, AVG(salario) AS salario_promedio
FROM empleados
GROUP BY posicion;
```

Uniones entre tablas

```
SELECT e.nombre, p.nombre AS nombre_producto
FROM empleados e
JOIN ventas v ON e.id = v.empleado_id
JOIN productos p ON v.producto_id = p.id;
```

Técnicas para el uso del SQL en los programas de aplicación

El uso de SQL en programas de aplicación es fundamental para interactuar con bases de datos y manejar datos de manera eficiente. Aquí te presento algunas técnicas clave que pueden ayudarte a integrar SQL en tus aplicaciones de manera efectiva:

Uso de ORM (Object-Relational Mapping)

Un ORM mapea las clases de tu aplicación a tablas de la base de datos, permitiendo interactuar con la base de datos utilizando objetos en lugar de SQL explícito.

Ejemplos de ORM:

- Django ORM (Python)
- SQLAlchemy (Python)
- Hibernate (Java)
- Entity Framework (C#)

Ejemplo con SQLAlchemy (Python):

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    age = Column(Integer)

engine = create_engine('sqlite:///app.db')
Base.metadata.create_all(engine)
```

```
Session = sessionmaker(bind=engine)
session = Session()

new_user = User(name='Alice', age=25)
session.add(new_user)
session.commit()
```

Consultas Preparadas y Parámetros

Las consultas preparadas ayudan a prevenir inyecciones SQL y mejoran la eficiencia de las consultas repetidas.

Ejemplo en Python con sqlite3:

```
import sqlite3

conn = sqlite3.connect('app.db')
cursor = conn.cursor()

cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)")

name = 'Bob'
age = 30

cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", (name, age))
conn.commit()
```

Manejo de Transacciones

Las transacciones aseguran que un conjunto de operaciones SQL se ejecute de manera atómica, es decir, todo o nada.

Ejemplo en Python:

```
try:
    conn = sqlite3.connect('app.db')
    cursor = conn.cursor()
    cursor.execute("BEGIN TRANSACTION")

    cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('Charlie', 28))
    cursor.execute("UPDATE users SET age = ? WHERE name = ?", (29, 'Alice'))

    conn.commit()
except Exception as e:
    conn.rollback()
    print(f"Transaction failed: {e}")
finally:
    conn.close()
```

Conexiones a Bases de Datos

Administrar conexiones de manera eficiente es crucial para la performance de la aplicación. Utilizar un pool de conexiones es una práctica común.

Ejemplo en Java con JDBC y HikariCP:

```
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
public class Main {
    public static void main(String[] args) {
```

```
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://localhost:3306/mydb");
        config.setUsername("user");
        config.setPassword("password");

        HikariDataSource ds = new HikariDataSource(config);

        try (Connection conn = ds.getConnection()) {
            PreparedStatement ps = conn.prepareStatement("SELECT * FROM users");
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                System.out.println(rs.getString("name"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Optimización de Consultas

Optimizar consultas SQL es fundamental para mejorar el rendimiento de tu aplicación.

- **Índices:** Crear índices en columnas que se usan frecuentemente en cláusulas WHERE o JOIN.
- **EXPLAIN:** Utilizar la palabra clave EXPLAIN para analizar cómo se ejecutará una consulta y optimizarla.
- **Limitación de Resultados:** Utilizar LIMIT para reducir la cantidad de datos devueltos por una consulta.

Ejemplo de Índices:

```
CREATE INDEX idx_users_name ON users(name);
```

Integración con Frameworks y Librerías

La mayoría de los frameworks de desarrollo ofrecen bibliotecas y utilidades para trabajar con SQL de manera más sencilla.

- **Django (Python):** Utiliza un ORM robusto que simplifica las interacciones con la base de datos.
- **Spring Data JPA (Java):** Ofrece una abstracción sobre JPA para trabajar con bases de datos de manera declarativa.
- **Ruby on Rails (Ruby):** Incluye ActiveRecord como su ORM para facilitar las operaciones CRUD.

Manejo de Errores y Excepciones

Capturar y manejar adecuadamente las excepciones relacionadas con la base de datos es crucial para la estabilidad de la aplicación.

Ejemplo en Python:

```
import sqlite3


try:
    conn = sqlite3.connect('app.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM non_existing_table")
except sqlite3.Error as e:
    print(f"Database error: {e}")
finally:
    if conn:
        conn.close()
```

Seguridad y Buenas Prácticas

- **Validación de Entradas:** Asegúrate de validar y sanear todas las entradas del usuario.
- **Principio de Menor Privilegio:** Da a los usuarios de la base de datos solo los permisos necesarios.
- **Cifrado:** Utiliza cifrado para datos sensibles.

SQL incorporado

El SQL incorporado se refiere a la técnica de incluir código SQL directamente dentro de un lenguaje de programación, permitiendo que la aplicación interactúe con una base de datos relacional sin la necesidad de herramientas intermedias. A continuación, se describen algunas maneras de incorporar SQL en varios lenguajes de programación, junto con ejemplos prácticos.

SQL Incorporado en C

En C, se puede utilizar Embedded SQL (SQL embebido) con bibliotecas como sqlite3 o usando SQL precompilado con Pro*C (en el caso de Oracle).

Ejemplo con sqlite3:

```
#include <stdio.h>
#include <sqlite3.h>

int main() {
   sqlite3 *db;
   char *errMsg = 0;
   int rc;
   const char *sql;

   // Open database
   rc = sqlite3_open("test.db", &db);

   if (rc) {
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
      return 0;
   } else {
      fprintf(stderr, "Opened database successfully\n");
   }

   // Create SQL statement
   sql = "CREATE TABLE IF NOT EXISTS COMPANY("
```

```
        "ID INT PRIMARY KEY NOT NULL,"
        "NAME TEXT NOT NULL,"
        "AGE INT NOT NULL,"
        "ADDRESS CHAR(50),"
        "SALARY REAL );";

    // Execute SQL statement
    rc = sqlite3_exec(db, sql, 0, 0, &errMsg);

    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", errMsg);
        sqlite3_free(errMsg);
    } else {
        fprintf(stdout, "Table created successfully\n");
    }

    sqlite3_close(db);
    return 0;
}
```

SQL Incorporado en Python

Python utiliza varias bibliotecas para interactuar con bases de datos SQL, como sqlite3, psycopg2 (para PostgreSQL), y MySQLdb.

Ejemplo con sqlite3:

```
import sqlite3

# Connect to the database
conn = sqlite3.connect('test.db')
cursor = conn.cursor()
# Create table
cursor.execute('''CREATE TABLE IF NOT EXISTS company
                (id INT PRIMARY KEY NOT NULL,
```

```
            name TEXT NOT NULL,
            age INT NOT NULL,
            address CHAR(50),
            salary REAL);"")

# Insert a row of data
cursor.execute("INSERT INTO company (id, name, age, address, salary) VALUES (1, 'Paul', 32, 'California', 20000.00)")

# Save (commit) the changes
conn.commit()

# Close the connection
conn.close()
```

SQL Incorporado en Java

En Java, el uso de JDBC (Java Database Connectivity) permite incorporar SQL en aplicaciones Java.

Ejemplo con JDBC:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class Main {
    public static void main(String[] args) {
        String url = "jdbc:sqlite:test.db";

        try (Connection conn = DriverManager.getConnection(url);
             Statement stmt = conn.createStatement()) {
```

```
            // Create table
            String sql = "CREATE TABLE IF NOT EXISTS company (" +
                    "id INT PRIMARY KEY NOT NULL," +
                    "name TEXT NOT NULL," +
                    "age INT NOT NULL," +
                    "address CHAR(50)," +
                    "salary REAL)";
            stmt.execute(sql);


            // Insert a row of data
            sql = "INSERT INTO company (id, name, age, address, salary) VALUES (1, 'Paul', 32, 'California', 20000.00)";
            stmt.execute(sql);


            System.out.println("Table created and data inserted successfully");


        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Consideraciones Finales

1. **Seguridad:** Siempre utilizar consultas preparadas o parametrizadas para evitar inyecciones SQL.
2. **Manejo de Errores:** Implementar manejo de errores para capturar y gestionar posibles fallos durante la ejecución de las consultas.
3. **Transacciones:** Utilizar transacciones para asegurar la integridad de los datos en operaciones que involucren múltiples consultas.
4. **Conexión Eficiente:** Usar pooling de conexiones cuando sea necesario para mejorar el rendimiento de la aplicación.

Estas técnicas te ayudarán a integrar SQL de manera segura y eficiente en tus aplicaciones, independientemente del lenguaje de programación que estés utilizando.

API

¿Qué son las API?

Una API (Application Programming Interface, Interfaz de Programación de Aplicaciones) es un conjunto de reglas y herramientas que permiten a los desarrolladores crear software y aplicaciones. Las API definen métodos y estructuras que los desarrolladores pueden usar para interactuar con otros programas o servicios, permitiendo la comunicación entre diferentes sistemas.

Las API pueden ser de varios tipos:

- **API de Sistema Operativo:** Permiten a las aplicaciones interactuar con el sistema operativo.
- **API de Biblioteca:** Proveen funciones que las aplicaciones pueden usar.
- **API Web:** Permiten la interacción entre aplicaciones a través de la web.
- **API de Base de Datos:** Permiten a las aplicaciones interactuar con bases de datos.

¿Qué son las API de SQL?

Las API de SQL son interfaces que permiten a las aplicaciones comunicarse con bases de datos SQL. Estas API proporcionan métodos y estructuras para ejecutar consultas SQL, obtener resultados, manejar transacciones, y realizar otras operaciones relacionadas con bases de datos. Las API de SQL se integran en diversos lenguajes de programación y proporcionan una capa de abstracción que facilita el acceso y la manipulación de bases de datos relacionales.

Principales API de SQL y sus Explicaciones

1. **JDBC (Java Database Connectivity)**
   JDBC es una API de Java que permite a los desarrolladores ejecutar operaciones SQL en bases de datos. JDBC proporciona un conjunto de clases e interfaces para conectar a bases de datos, ejecutar consultas, y manejar resultados.

Características:

- **Conexión a Bases de Datos:** Provee métodos para conectar a diferentes bases de datos (MySQL, PostgreSQL, Oracle, etc.).
- **Ejecutar Consultas:** Permite ejecutar consultas SQL, tanto estáticas como preparadas.

- **Manejo de Resultados:** Ofrece mecanismos para recorrer y manipular los resultados de las consultas.
- **Transacciones:** Soporta transacciones, permitiendo operaciones atómicas.

Ejemplo de Uso:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Main {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try (Connection conn = DriverManager.getConnection(url, user, password);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery("SELECT * FROM users")) {

            while (rs.next()) {
                System.out.println(rs.getString("username"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2. **ODBC (Open Database Connectivity)**

   ODBC es una API que permite a las aplicaciones acceder a bases de datos de manera independiente del sistema gestor de bases de datos (DBMS). Provee una interfaz estándar para el acceso a datos.

Características:

- **Independencia de DBMS:** Permite a las aplicaciones acceder a diferentes DBMS de manera uniforme.
- **Conectividad Heterogénea:** Soporta la conexión a múltiples tipos de bases de datos.
- **Consulta y Actualización:** Permite ejecutar consultas y actualizaciones SQL.

3. **ADO.NET**

   ADO.NET es una API de Microsoft que permite a las aplicaciones .NET interactuar con bases de datos. Es parte del .NET Framework y proporciona una rica colección de clases para manejar datos.

Características:

- **Data Providers:** Soporta múltiples proveedores de datos como SQL Server, Oracle, etc.
- **Conectividad:** Proporciona objetos para conexión y comandos SQL.
- **DataSets y DataTables:** Permite trabajar con datos en memoria de forma desconectada.

Ejemplo de Uso en C#:

```
using System;
using System.Data.SqlClient;


class Program {
   static void Main() {
      string connectionString = "Data Source=localhost;Initial Catalog=mydatabase;User ID=user;Password=password";
      using (SqlConnection connection = new SqlConnection(connectionString)) {
```

```
            connection.Open();

            SqlCommand command = new SqlCommand("SELECT * FROM users", connection);
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read()) {
                Console.WriteLine(reader["username"]);
            }
        }
    }
}
```

4. **DB-API (Python)**

   DB-API es una especificación para interfaces de acceso a bases de datos en Python. Define una serie de métodos y propiedades que los módulos de bases de datos deben implementar.

Características:

- **Estándar:** Proporciona una interfaz estándar para la interacción con bases de datos SQL.
- **Soporte para Múltiples DBMS:** Existen módulos para PostgreSQL (psycopg2), MySQL (mysql-connector-python), SQLite (sqlite3), entre otros.
- **Operaciones SQL:** Permite la ejecución de consultas, manejo de transacciones y acceso a resultados.

Las API de SQL son cruciales para integrar y gestionar bases de datos en aplicaciones. Facilitan la conexión, ejecución de consultas, manejo de transacciones y procesamiento de resultados de manera estandarizada, independiente del lenguaje de programación o sistema gestor de bases de datos utilizado. Utilizando estas API, los desarrolladores pueden crear aplicaciones robustas y eficientes que interactúan con bases de datos de manera segura y eficaz.

CONCLUSIÓN

La gestión de bases de datos SQL constituye un pilar esencial en el desarrollo de software moderno, proporcionando la infraestructura necesaria para el almacenamiento, acceso y manipulación de datos estructurados.

Las tablas en SQL, equivalentes a colecciones en bases de datos NoSQL, organizan los datos en filas y columnas, facilitando operaciones como la creación, modificación, selección y eliminación de registros. Estas operaciones básicas son fundamentales para mantener y gestionar datos de manera estructurada.

La integración de SQL en aplicaciones se ve enormemente facilitada por diversas API y frameworks como JDBC, ODBC, ADO.NET y DB-API, que proporcionan métodos estandarizados para la conexión, ejecución de consultas y gestión de transacciones.

La seguridad es un aspecto crítico en la gestión de bases de datos. La adopción de buenas prácticas como la validación de entradas, el uso de consultas preparadas y la implementación de mecanismos de cifrado es esencial para proteger los datos y prevenir vulnerabilidades.

La aplicación de técnicas y principios avanzados en la gestión de bases de datos SQL permiten a los desarrolladores construir aplicaciones seguras y eficientes. La habilidad para gestionar datos de manera efectiva es una competencia crucial para cualquier desarrollador de software, asegurando que las aplicaciones puedan manejar grandes volúmenes de datos de manera efectiva y segura.

BIBLIOGRAFÍA

Referencias

Oracle. (n.d.). JDBC Overview. https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

GeeksforGeeks. (n.d.). JDBC - Java Database Connectivity. https://www.geeksforgeeks.org/jdbc-java-database-connectivity/

Oracle. (2021). ODBC Programmer's Reference. https://docs.oracle.com/en/database/oracle/oracle-database/21/odbc/ODBC-Programmer-s-Reference.html

Microsoft. (n.d.). Getting Started with ODBC. https://docs.microsoft.com/en-us/sql/odbc/microsoft/odbc-driver-for-sql-server

Microsoft. (2021). ADO.NET Overview. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ado-net-overview

ZetCode. (n.d.). C# ADO.NET Tutorial. https://zetcode.com/csharp/adonet/

PEP 249 – Python Database API Specification v2.0. (2001). Python Software Foundation. https://www.python.org/dev/peps/pep-0249/

Tutorialspoint. (n.d.). Python SQLite. https://www.tutorialspoint.com/sqlite/sqlite_python.htm