BASES DE DATOS DISTRIBUIDAS

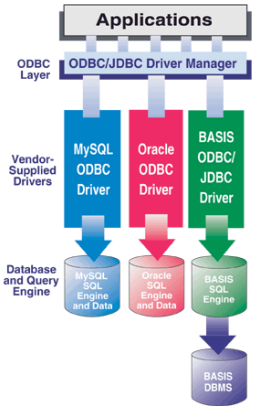
Sistemas informáticos I

3.3 SQL EN LAS APLICACIONES

Bases de datos distribuidas

SQL INTERACTIVO VS. NO INTERACTIVO

- **SQL interactivo**: uso de SQL en el cliente del SGBD
  - Definir la estructura de la base de datos
  - Probar consultas
  - Realizar prototipos
- Las aplicaciones no acceden a datos almacenados en bases de datos relacionales a través de SQL interactivo, sino a través de un *middleware*, el *driver de la base de datos* (*ODBC*, *JDBC*)
  - Específico del SGBD y el lenguaje de alto nivel en el que está programada la aplicación
  - Define una API de conexión común:
    - Conexión a la base de datos
    - Manipulación/acceso a datos. Hay flujo de información:
      - Desde el lenguaje de alto nivel hacia la base de datos
      - Desde la base de datos hacia el lenguaje de alto nivel
    - Desconexión de la base de datos



EPS

EVOLUCIÓN DE LOS MECANISMOS DE ACCESO A DATOS

- **SQL embebido** (inmerso/incrustado) en el código – *embedded SQL*
  - Las sentencias SQL están incrustadas dentro del propio código
  - Se construyen como Strings (dinámicos) que se pasan al SGBD a través del driver
- **Sentencias preparadas** – *Prepared statements*
  - Sentencia SQL precompilada que acepta parámetros
  - Mejora el tiempo de respuesta y/o la seguridad
  - Útil cuando una misma sentencia se utiliza muchas veces
    - Reduce el envío de datos desde la aplicación
- **DataSources lógicos**
  - Como patrón de diseño → mejora la reusabilidad/mantenibilidad
  - Como herramienta de acceso a datos → mejora de los tiempos de acceso

EPS

EVOLUCIÓN DE LOS MECANISMOS DE ACCESO A DATOS

- Separación de responsabilidades/*Separation of concerns*
  - Mejora de la legibilidad y mantenibilidad del código
  - Frameworks y bibliotecas que permiten que las sentencias SQL no aparezcan inmersas en el código funcional
- **ORM** – *Object-Relational Mapping*
  - Abstracción del acceso a datos
  - Inicialmente parte del diseño de la aplicación, en la actualidad, en muchos casos se gestiona automáticamente
    - Frameworks de persistencia que almacenan y recuperan objetos de una base de datos relacional de forma transparente para el programador de la lógica de negocio
  - Los programas invocan a la capa de persistencia como a cualquier otro elemento de la lógica de negocio (el código SQL no existe)

SQL EMBEBIDO EN PHP

```php
<?php
  try {
    // Conexión con la base de datos
    $conexion = new PDO('sqlite:../sqlite.db');
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Selección de datos
    $rs = $conexion->query('SELECT * FROM Departamento');
    echo "Listado de departamentos...\n";
    foreach($rs as $departamento) {
      echo $departamento['idDepartamento'] . "\t" . $departamento['nombre'] . "\n";
    }

    // Insert
    $conexion->exec("INSERT INTO Departamento (nombre) VALUES ('Arquitectura TI')");

    $conexion = null;
  }
  catch(PDOException $e) {
    echo $e->getMessage();
  }
?>
```

SQL EMBEBIDO EN PHP

- PDO → *PHP Data Object* http://www.php.net/manual/en/book.pdo.php
  - Extensión de PHP para crear acceso uniforme a bases de datos
  - Permite crear código portable entre plataformas
  - Soporte para múltiples SGBD
- Establecimiento de la conexión con la base de datos:
  - Crear una instancia de la clase PDO pasando al constructor distintos parámetros en función del SGBD

```
<?php
try {
    $db = new PDO("pgsql:dbname=pdo; host=localhost", "username", "password" );
    /*** use the database connection ***/
}
catch(PDOException $e) {
    echo $e->getMessage();
}
```

SQL EMBEBIDO EN PHP

- Interacciones con la base de datos:
  - El método `PDO::exec` se utiliza con las sentencias SQL que no devuelven un conjunto de resultados
    - Retorna el total de filas afectadas

```
$count=$db->exec("INSERT INTO actor(id,nombre) VALUES (15,'Clint Eastwood')");
echo $count;
```

```
1
```

  - El método `PDO::query` se utiliza para obtener datos de la base de datos
    - Retorna una instancia de la clase *PDOStatement*

```
$sql = "SELECT * FROM pelicula";
foreach ($db->query($sql) as $row) {
   print $row['titulo'] .' - '. $row['puntuacion'] . '<br />';
}
```

SQL EMBEBIDO EN PHP

- Algunos métodos implementados por la clase *PDOStatement*:
  - `fetch`: retorna la próxima fila
  - `fetchAll`: retorna un array con todas las filas
  - `fetchColumn`: retorna un solo campo de la próxima fila
  - `fetchObject`: retorna la próxima fila como un objeto
  - `execute`: ejecuta una sentencia preparada
  - `bindColumn`: liga una columna a una variable PHP
- Los resultados pueden ser devueltos (fila a fila) de distinta forma según se especifique con el método `setFetchMode`
  - `PDO::FETCH_NUM`: array indexado numéricamente
  - `PDO::FETCH_ASSOC`: como array asociativo
  - `PDO::FETCH_OBJ`: como un objeto

```
$stmt = $db->query($sql);
$result = $stmt->setFetchMode(PDO::FETCH_NUM);
while ($row = $stmt->fetch()) {
    print $row[0] . "\t" . $row[1] . "\t" . $row[2] . "\n";
  }
```

SQL EMBEBIDO EN PHP

- Para cerrar la conexión hay que destruir el objeto → eliminar toda referencia al objeto
  - Automáticamente al finalizar la ejecución del script
  - También se puede cerrar asignado un valor `null` a la referencia

```
$db = new PDO("mysql:host=$hostname;dbname=mysql", $username, $password);

    /*** use the database connection ***/

$db = null;
```

SQL EMBEBIDO EN JAVA

https://docs.oracle.com/javase/10/docs/api/java.sql-frame.html

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

public class TesterSQLEmbebidoMySQL {

    public static void main(String[] args) {
        Connection conexion = null;

        try {
            // Adquisición del driver y conexión con la base de datos
            Class.forName("com.mysql.jdbc.Driver");
            conexion = DriverManager.getConnection ("jdbc:mysql://localhost/test", "usuario_bd", "la_clave");

            // Consultas normales
            Statement qry = conexion.createStatement();
            ResultSet rs = qry.executeQuery("SELECT tarifa, nombre FROM Empleado WHERE tarifa > 50 ORDER BY nombre");
            System.out.println("Resultado de la consulta sin preparar:");
            while(rs.next())
            {
                System.out.println(rs.getString("nombre") + "\t" + rs.getString(1));
            }

            // PreparedStatement
            PreparedStatement preparedQry = conexion.prepareStatement("SELECT * FROM Empleado WHERE tarifa > ?");
            preparedQry.setDouble(1, 10.0);
            rs = preparedQry.executeQuery();
            System.out.println("Resultado de la consulta preparada:");
            while(rs.next())
            {
                System.out.println(rs.getString("nombre") + "\t" + rs.getDouble("tarifa"));
            }
```

SQL EMBEBIDO EN JAVA

```
            // Ejecución de sentencia
            conexion.setAutoCommit(false);
            qry.execute("UPDATE Empleado SET nombre = UPPER(nombre)");

            // Se reutiliza la consulta preparada
            preparedQry.setDouble(1, 100.0);
            rs = preparedQry.executeQuery();
            System.out.println("Resultado de la consulta preparada:");
            while(rs.next())
            {
                System.out.println(rs.getString("nombre") + "\t" + rs.getDouble("tarifa"));
            }

            conexion.commit();

            conexion.close();
        }
        catch(Exception err) {
            err.printStackTrace();
            if(conexion != null)
            {
                try {
                    conexion.rollback();
                    conexion.close();
                }
                catch(SQLException err2) {}
            }
        }
    }
```

SQL EMBEBIDO EN JAVA

- *java.sql.DataSource* = factoría de conexiones al origen de datos físico
- Incluido en la JDK a partir de la versión 1.4
- Es la forma recomendada de obtener una conexión con la base de datos → ofrece ventajas frente a obtenerla del *DriverManager* (depende de la implementación del driver)
  - Facilidades para el registro y la inyección de dependencias
  - Pool de conexiones
  - Transacciones distribuidas

```
import java.sql.DataSource;
import java.sql.Connection;

DataSource ds = MyDataSourceFactory.getDataSource();
Connection conexion = ds.getConnection();
```

https://docs.oracle.com/javase/10/docs/api/javax/sql/DataSource.html

SQL EMBEBIDO EN JAVA

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

import org.sqlite.SQLiteDataSource;

public class TesterDatasource {

   public static void main(String args[]) {
       Connection conexion = null;

       try {
           SQLiteDataSource ds = new SQLiteDataSource();
           ds.setUrl(TesterSQLEmbebidoSQLite.URL_BD);

           conexion = ds.getConnection();

           Statement qry = conexion.createStatement();
           ResultSet rs = qry.executeQuery("SELECT tarifa, nombre FROM Empleado WHERE tarifa > 50 ORDER BY nombre");
           System.out.println("Resultado de la consulta sin preparar:");
           while(rs.next())
           {
               System.out.println(rs.getString("nombre") + "\t" + rs.getString(1));
           }
       }
       catch(Exception err) {
           err.printStackTrace();
       }
   }
```

SEPARACIÓN DE RESPONSABILIDADES

- Antiguamente iBatis, hasta la versión 3.0
- Según su propia web: *MyBatis es un framework de persistencia que soporta SQL, procedimientos almacenados y mapeos avanzados. MyBatis elimina casi todo el código JDBC, el establecimiento manual de los parámetros y la obtención de resultados. MyBatis puede configurarse con XML o anotaciones y permite mapear mapas y POJOs (Plain Old Java Objects) con registros de base de datos.*

MyBatis

www.mybatis.org

ORM, *OBJECT-RELATIONAL MAPPING*

- Hibernate ORM
  - Hibernate ORM enables developers to more easily write applications whose data outlives the application process. As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC)

http://hibernate.org/

ORM, *OBJECT-RELATIONAL MAPPING*

- *Django ORM*
  - Clases que heredan de *django.db.models.Models*
    - Representa una entidad y definen todos sus campos y relaciones
      - ForeignKey (uno a muchos)
      - OneToOneField
      - ManyToManyField
  - El modelo de datos lógico se convierte automáticamente en un modelo de datos físico en el SGBD
    - Opciones migrate y makemigrations del manage.py
  - Una vez cargado en la base de datos, *django* ofrece una funcionalidad estándar para trabajar con los datos mediante un patrón DAO

django

ORM, *OBJECT-RELATIONAL MAPPING*

```python
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models


class Departamento(models.Model):
    nombre = models.CharField(max_length=128, unique=True)

    def __str__(self):
        return str(self.id) + "\t" + self.nombre

    def __unicode__(self):
        return str(self.id) + "\t" + self.nombre

class Empleado(models.Model):
    nombre = models.CharField(max_length=128)
    tarifa = models.DecimalField(max_digits=10, decimal_places=2)
    categoria = models.CharField(max_length=128)
    superior = models.ForeignKey("self", null=True, default=None)
    departamento = models.ManyToManyField(Departamento)
    direccion = models.CharField(max_length=128, default='Desconocida')

    def __str__(self):
        return str(self.id) + "\t" + self.nombre

    def __unicode__(self):
        return str(self.id) + "\t" + self.nombre
```