Curso de ProgramaciÃ³n Desde Cero

Tema 6: Bases de Datos y SQL

6.1 IntroducciÃ³n a las Bases de Datos

Las bases de datos son sistemas organizados para almacenar, gestionar y recuperar informaciÃ³n de manera eficiente. Son fundamentales en casi todas las aplicaciones modernas, desde simples sitios web hasta complejos sistemas empresariales que manejan millones de registros.

Â¿Por quÃ© son Importantes las Bases de Datos?

â€¢ Persistencia: Los datos sobreviven al cierre de la aplicaciÃ³n
â€¢ Integridad: Garantizan la consistencia y validez de los datos
â€¢ Concurrencia: MÃºltiples usuarios pueden acceder simultÃ¡neamente
â€¢ Escalabilidad: Manejan grandes volÃºmenes de informaciÃ³n
â€¢ Seguridad: Controlan el acceso y protegen informaciÃ³n sensible
â€¢ Eficiencia: Optimizan las consultas y el rendimiento

Tipos de Bases de Datos

Relacionales (SQL)
â€¢ Organizan datos en tablas con filas y columnas
â€¢ Usan relaciones entre tablas (foreign keys)
â€¢ Ejemplos: MySQL, PostgreSQL, SQLite, Oracle, SQL Server
â€¢ Ideales para: aplicaciones con estructura de datos clara

No Relacionales (NoSQL)
â€¢ Documentos: MongoDB, CouchDB
â€¢ Clave-Valor: Redis, DynamoDB
â€¢ Grafos: Neo4j, Amazon Neptune
â€¢ Columnas: Cassandra, HBase
â€¢ Ideales para: datos no estructurados, escalabilidad horizontal

Â¿QuÃ© es SQL?

SQL (Structured Query Language) es el lenguaje estÃ¡ndar para comunicarse con bases de datos relacionales. Permite realizar operaciones como:
â€¢ Consultar datos (SELECT)
â€¢ Insertar nuevos registros (INSERT)
â€¢ Actualizar datos existentes (UPDATE)
â€¢ Eliminar registros (DELETE)
â€¢ Crear y modificar estructura (CREATE, ALTER, DROP)

Conceptos Fundamentales

Tabla
Estructura que organiza datos en filas y columnas, como una hoja de cÃ¡lculo.

6.3 DiseÃ±o de Base de Datos

Modelado de Datos

Antes de crear tablas, debemos diseÃ±ar la estructura de datos:

Ejemplo: Sistema de Blog

Entidades principales:
â€¢ Usuarios: autores del blog
â€¢ Posts: artÃ-culos publicados
â€¢ CategorÃ-as: clasificaciÃ³n de posts
â€¢ Comentarios: respuestas de lectores

CreaciÃ³n de Base de Datos y Tablas

```
-- Crear base de datos
CREATE DATABASE blog_db;

-- Usar la base de datos
\c blog_db;

-- Tabla de usuarios
CREATE TABLE usuarios (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activo BOOLEAN DEFAULT true
);

-- Tabla de categorÃ-as
CREATE TABLE categorias (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) UNIQUE NOT NULL,
    descripcion TEXT,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Tabla de posts
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(255) NOT NULL,
    contenido TEXT NOT NULL,
    resumen TEXT,
    usuario_id INTEGER REFERENCES usuarios(id) ON DELETE CASCADE,
    categoria_id INTEGER REFERENCES categorias(id) ON DELETE SET NULL,
    fecha_publicacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    publicado BOOLEAN DEFAULT false,
    vistas INTEGER DEFAULT 0
);

-- Tabla de comentarios
CREATE TABLE comentarios (
    id SERIAL PRIMARY KEY,
    contenido TEXT NOT NULL,
    autor_nombre VARCHAR(100) NOT NULL,
    autor_email VARCHAR(255) NOT NULL,
    post_id INTEGER REFERENCES posts(id) ON DELETE CASCADE,
    fecha_comentario TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    aprobado BOOLEAN DEFAULT false
);
```

6.4 Consultas SQL BÃ¡sicas

INSERT: Insertar Datos

```
-- Insertar un registro
INSERT INTO usuarios (nombre, email, password_hash)
VALUES ('Juan PÃ©rez', 'juan@email.com', 'hash_seguro_123');

-- Insertar mÃºltiples registros
INSERT INTO categorias (nombre, descripcion) VALUES
    ('TecnologÃ-a', 'ArtÃ-culos sobre tecnologÃ-a y programaciÃ³n'),
    ('Viajes', 'Experiencias y guÃ-as de viaje'),
    ('Cocina', 'Recetas y tÃ©cnicas culinarias');

-- Insertar con subconsulta
INSERT INTO posts (titulo, contenido, usuario_id, categoria_id)
SELECT
    'Mi primer post',
    'Contenido del primer post',
    u.id,
    c.id
FROM usuarios u, categorias c
WHERE u.email = 'juan@email.com' AND c.nombre = 'TecnologÃ-a';
```

SELECT: Consultar Datos

```
-- Seleccionar todas las columnas
SELECT * FROM usuarios;

-- Seleccionar columnas especÃ-ficas
SELECT nombre, email FROM usuarios;

-- Condiciones con WHERE
SELECT * FROM usuarios WHERE activo = true;
SELECT * FROM posts WHERE fecha_publicacion >= '2024-01-01';

-- Operadores de comparaciÃ³n
SELECT * FROM posts WHERE vistas > 100;
SELECT * FROM posts WHERE vistas BETWEEN 50 AND 200;
SELECT * FROM usuarios WHERE nombre LIKE 'Juan%';
SELECT * FROM posts WHERE categoria_id IN (1, 2, 3);

-- Ordenamiento
SELECT * FROM posts ORDER BY fecha_publicacion DESC;
SELECT * FROM posts ORDER BY vistas DESC, titulo ASC;

-- LimitaciÃ³n de resultados
SELECT * FROM posts ORDER BY fecha_publicacion DESC LIMIT 10;
SELECT * FROM posts ORDER BY id LIMIT 10 OFFSET 20;
```

UPDATE: Actualizar Datos

```
-- Actualizar un registro especÃ-fico
UPDATE usuarios
SET nombre = 'Juan Carlos PÃ©rez'
WHERE email = 'juan@email.com';

-- Actualizar mÃºltiples campos
UPDATE posts
SET publicado = true, fecha_actualizacion = CURRENT_TIMESTAMP
WHERE id = 1;

-- Actualizar con cÃ¡lculos
UPDATE posts
SET vistas = vistas + 1
WHERE id = 1;

-- Actualizar con subconsulta
```

6.5 Consultas Avanzadas y JOINs

INNER JOIN: UniÃ³n Interna

Combina filas de dos o mÃ¡s tablas basÃ¡ndose en una condiciÃ³n relacionada:

```
-- Posts con informaciÃ³n del autor
SELECT
    p.titulo,
    p.fecha_publicacion,
    u.nombre as autor,
    p.vistas
FROM posts p
INNER JOIN usuarios u ON p.usuario_id = u.id
WHERE p.publicado = true
ORDER BY p.fecha_publicacion DESC;

-- Posts con autor y categorÃ-a
SELECT
    p.titulo,
    u.nombre as autor,
    c.nombre as categoria,
    p.vistas
FROM posts p
INNER JOIN usuarios u ON p.usuario_id = u.id
INNER JOIN categorias c ON p.categoria_id = c.id
WHERE p.publicado = true;
```

LEFT JOIN: UniÃ³n Izquierda

Incluye todos los registros de la tabla izquierda, aunque no tengan coincidencia:

```
-- Todas las categorÃ-as con conteo de posts (incluso sin posts)
SELECT
    c.nombre as categoria,
    COUNT(p.id) as cantidad_posts
FROM categorias c
LEFT JOIN posts p ON c.id = p.categoria_id AND p.publicado = true
GROUP BY c.id, c.nombre
ORDER BY cantidad_posts DESC;

-- Usuarios con sus posts (incluso usuarios sin posts)
SELECT
    u.nombre,
    u.email,
    COUNT(p.id) as total_posts,
    COALESCE(SUM(p.vistas), 0) as total_vistas
FROM usuarios u
LEFT JOIN posts p ON u.id = p.usuario_id
GROUP BY u.id, u.nombre, u.email
ORDER BY total_posts DESC;
```

RIGHT JOIN y FULL OUTER JOIN

```
-- RIGHT JOIN (menos comÃºn)
SELECT * FROM posts p
RIGHT JOIN usuarios u ON p.usuario_id = u.id;

-- FULL OUTER JOIN (incluye todos los registros de ambas tablas)
SELECT
    u.nombre,
    p.titulo
FROM usuarios u
```

6.6 IntegraciÃ³n con Node.js

InstalaciÃ³n de Cliente PostgreSQL

```
npm install pg dotenv
```

ConfiguraciÃ³n de ConexiÃ³n

```
// config/database.js
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  host: process.env.DB_HOST || 'localhost',
  port: process.env.DB_PORT || 5432,
  database: process.env.DB_NAME || 'blog_db',
  user: process.env.DB_USER || 'postgres',
  password: process.env.DB_PASSWORD,
  max: 20,
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

module.exports = pool;
```

Operaciones CRUD BÃ¡sicas

```
// models/Usuario.js
const pool = require('../config/database');

class Usuario {
  static async crear(nombre, email, passwordHash) {
    const query = `
      INSERT INTO usuarios (nombre, email, password_hash)
      VALUES ($1, $2, $3)
      RETURNING id, nombre, email, fecha_registro`
    ;
    const result = await pool.query(query, [nombre, email, passwordHash]);
    return result.rows[0];
  }

  static async buscarPorEmail(email) {
    const query = 'SELECT * FROM usuarios WHERE email = $1';
    const result = await pool.query(query, [email]);
    return result.rows[0];
  }

  static async obtenerTodos() {
    const query = `
      SELECT id, nombre, email, fecha_registro, activo
      FROM usuarios
      ORDER BY fecha_registro DESC`
    ;
    const result = await pool.query(query);
    return result.rows;
  }

  static async actualizar(id, datos) {
    const campos = Object.keys(datos);
    const valores = Object.values(datos);
    const placeholders = campos.map((_, i) => `$${i + 1}`);

    const query = `UPDATE usuarios SET ${campos.map((campo, i) =>
', ')} WHERE id = $${campos.length + 1}`;

    const result = await pool.query(query, [...valores, id]);
```

Transacciones

Para operaciones que requieren consistencia de datos:

```
// utils/transacciones.js
const pool = require('../config/database');

async function crearPostConComentario(postData, comentarioData) {
  const client = await pool.connect();

  try {
    await client.query('BEGIN');

    // Crear post
    const postQuery = `INSERT INTO posts (titulo, contenido, usuario_id)
      VALUES ($1, $2, $3) RETURNING id`;
    const postResult = await client.query(postQuery,
      [postData.titulo, postData.contenido, postData.usuarioId]);

    const postId = postResult.rows[0].id;

    // Crear comentario
    const comentarioQuery = `INSERT INTO comentarios
      (contenido, autor_nombre, autor_email, post_id)
      VALUES ($1, $2, $3, $4)`;
    await client.query(comentarioQuery, [
      comentarioData.contenido,
      comentarioData.autorNombre,
      comentarioData.autorEmail,
      postId
    ]);

    await client.query('COMMIT');
    return postId;

  } catch (error) {
    await client.query('ROLLBACK');
    throw error;
  } finally {
    client.release();
  }
}
```

MigraÃ§Ã£o y Scripts de BD

```
// scripts/migrate.js
const fs = require('fs');
const path = require('path');
const pool = require('../config/database');

async function ejecutarMigraciones() {
  try {
    // Crear tabla de migraciones si no existe
    await pool.query(`CREATE TABLE IF NOT EXISTS migraciones (
      id SERIAL PRIMARY KEY,
      nombre VARCHAR(255) UNIQUE NOT NULL,
      ejecutada_en TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )`);

    // Leer archivos de migraciÃ³n
    const migracionesDir = path.join(__dirname, 'migraciones');
    const archivos = fs.readdirSync(migracionesDir);

    for (const archivo of archivos) {
      // Verificar si ya se ejecutÃ³
      const yaEjecutada = await pool.query(
        'SELECT id FROM migraciones WHERE nombre = $1', [archivo]);

      if (yaEjecutada.rows.length === 0) {
        console.log(`Ejecutando migraciÃ³n: ${archivo}`);
```

6.9 Ejercicios PrÃ¡cticos

Ejercicio 1: Sistema de E-commerce

DiseÃ±a una base de datos para un sistema de comercio electrÃ³nico:

```
-- Tablas principales
CREATE TABLE clientes (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    telefono VARCHAR(20),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE productos (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(200) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10,2) CHECK (precio >= 0),
    stock INTEGER CHECK (stock >= 0),
    categoria_id INTEGER,
    activo BOOLEAN DEFAULT true
);
CREATE TABLE pedidos (
    id SERIAL PRIMARY KEY,
    cliente_id INTEGER REFERENCES clientes(id),
    fecha_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado VARCHAR(20) DEFAULT 'pendiente',
    total DECIMAL(10,2)
);
CREATE TABLE detalle_pedidos (
    id SERIAL PRIMARY KEY,
    pedido_id INTEGER REFERENCES pedidos(id),
    producto_id INTEGER REFERENCES productos(id),
    cantidad INTEGER CHECK (cantidad > 0),
    precio_unitario DECIMAL(10,2)
);
```

Ejercicio 2: Consultas de AnÃ¡lisis

```
-- Top 10 productos mÃ¡s vendidos
SELECT
    p.nombre,
    SUM(dp.cantidad) as total_vendido,
    SUM(dp.cantidad * dp.precio_unitario) as ingresos
FROM productos p
JOIN detalle_pedidos dp ON p.id = dp.producto_id
GROUP BY p.id, p.nombre
ORDER BY total_vendido DESC
LIMIT 10;
-- Clientes mÃ¡s valiosos
SELECT
    c.nombre,
    c.email,
    COUNT(pe.id) as total_pedidos,
    SUM(pe.total) as valor_total
FROM clientes c
LEFT JOIN pedidos pe ON c.id = pe.cliente_id
GROUP BY c.id, c.nombre, c.email
ORDER BY valor_total DESC
```