

# Bases de Datos

Clase 13: Programación y SQL

# Programación y SQL

- Hasta ahora hemos visto a la Base de Datos como un componente aislado
- Pero una Base de Datos no tiene sentido si no podemos conectarla a una aplicación

# Programación y SQL

Contamos con:

- Un DBMS
- Un entorno de programación (Java, Python, PHP...)

**Obs:** La mayoría de esta clase se orienta a conectar un lenguaje de programación con el Sistema PostgreSQL

# Conexión a la Base de Datos

Java

Primero debemos importar la librería:

```
import java.sql.*;
```

# Conexión a la Base de Datos

Java

Cargar el driver (en este caso el de PostgreSQL):

```
Class.forName("org.postgresql.Driver");
```

# Conexión a la Base de Datos

Java

Objeto para conectar:

```
Connection db = DriverManager.getConnection(url,  
                                             username, password);
```

Donde la URL es:

```
jdbc:postgresql://host:puerto/nombre_basededatos
```

# Conexión a la Base de Datos

Python

Primero debemos importar la librería (esta sirve sólo con Postgres):

```
import psycopg2
```

# Conexión a la Base de Datos

Python

Objeto para conectar:

```
try:
    conn = psycopg2.connect(
        database="database",
        user="user",
        host="localhost",
        port=5432,
        password="pass")
except:
    print("No me pude conectar")
```



# Consideraciones

- Los tipos de datos de los lenguajes pueden no ser soportados por SQL o vice versa
- SQL funciona en base a tuplas, los lenguajes de programación en general no

# Inserción

Java

```
String query = "INSERT INTO Peliculas VALUES (" +  
                titulo + ", " + año +  
                ", " + director + ")";  
  
try {  
    stmt = con.createStatement();  
    stmt.executeQuery(query);  
} catch (SQLException e) {  
    error = e.getSQLState();  
    System.out.println(error);  
}
```

# SQLSTATE y SQLException

Java

```
catch (SQLException e ) {  
    error = e.getSQLState();  
    System.out.println(error);  
}
```

# SQLSTATE y SQLException

## Java

SQLSTATE es una variable especial con la que SQL se comunica al programa

Convenciones:

- SQLSTATE = 00000 - No hay error
- SQLSTATE = 02000 - No encontré tupla

# Inserción

Python

```
query = "INSERT INTO Peliculas VALUES (" + \  
        titulo + ", " + ano + \  
        ", " + director + ")" ;  
cur = conn.cursor()  
try:  
    cur.execute(query)  
except psycopg2.Error as e:  
    print(e.pgcode)
```

# Cursores

Sabemos hacer consultas, ¿pero cómo le entregamos los resultados al lenguaje de programación?

Debemos usar **cursores**, que nos entregan los resultados de una fila a la vez

# Cursores

Sabemos hacer consultas, ¿pero cómo le entregamos los resultados al lenguaje de programación?

Debemos usar **cursores**, que nos entregan los resultados de una fila a la vez

¿Por qué no nos conviene entregar todos los resultados de una sola vez?

# Cursores

Java

```
String query = "SELECT * FROM Cervezas";
try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        String nombreCafe = rs.getString("nombre");
        int proveedor = rs.getInt("id_proveedor");
        float precio = rs.getFloat("precio");
        System.out.println(nombre + "\t" + proveedor +
                           "\t" + precio);
    }
}
```



# Cursores

Python

```
import psycopg2

try:
    conn = psycopg2.connect(database="dbname",
                            user="dbuser", host="localhost",
                            password="dbpass")
    cur = conn.cursor()
    cur.execute("SELECT * FROM R")
    row = cur.fetchone()
    while row:
        print(row)
        row = cur.fetchone()
except:
    print("Hubo algún problema")
```

# Cursores

Python - Fetchall

```
try:
    conn = psycopg2.connect(database="dbname",
                             user="dbuser", host="localhost",
                             password="dbpass")
    cur = conn.cursor()
    cur.execute("SELECT * FROM R")
    rows = cur.fetchall()
    for row in rows:
        print(row)
except:
    print("Hubo algún problema")
```

# SQL Parametrizado

- Se trata de preparar una consulta con variables
- Cuando conozco el valor de las variables se instancia la nueva consulta concreta
- A su debido tiempo la ejecuto

# SQL Parametrizado

Preparación

```
Statement stmt;  
try {  
    stmt = db.createStatement( );  
}  
catch (SQLException e) {  
  
}
```

# SQL Parametrizado

En nuestro sistema, el usuario ingresa la película que desea consultar y se almacena en la variable `pel`

```
String sql;  
sql = "SELECT cine, hora  
      FROM Programacion  
      WHERE pelicula LIKE '%" + pel + "%'";  
ResultSet rs = stmt.executeQuery(sql);
```

Luego navegamos por el `ResultSet` como si fuese un iterador

# SQL Parametrizado

En nuestro sistema, el usuario ingresa la película que desea eliminar y se almacena en la variable **pel**

```
String sql;  
sql = "DELETE FROM Programacion  
      WHERE pelicula LIKE '%" + pel + "%'";  
int tuplasmodificadas = stmt.executeUpdate(sql);
```

Si mi consulta inserta, elimina o actualiza, **executeUpdate()** retorna el número de tuplas afectadas

# SQL Parametrizado

¿Qué pasa si modifico varias películas?

```
String sql;  
sql = "SELECT cine, hora  
      FROM Programacion  
      WHERE pelicula LIKE '%" + pel + "%'";  
ResultSet rs = stmt.executeQuery(sql);
```

# SQL Parametrizado

¿Qué pasa si modifico varias películas?

```
String sql;  
sql = "SELECT cine, hora  
      FROM Programacion  
      WHERE pelicula LIKE '%" + pel + "%'";  
ResultSet rs = stmt.executeQuery(sql);
```

Peor solución: ir modificando el string



# SQL Parametrizado

```
String sql;  
sql = "SELECT DISTINCT titulo  
      FROM Peliculas  
      WHERE director LIKE ?";  
stmt = db.prepareStatement(sql);  
stmt.setString(1, "%Tarantino%")  
stmt.executeQuery();
```

# SQL Parametrizado

```
stmt.setString(1, "%Tarantino%")  
stmt.executeQuery();
```

# SQL Parametrizado

```
stmt.setString(1, "%Tarantino%")  
stmt.executeQuery();
```

- Voy actualizando los directores con `setString()`
- Existen “`setXYZ()`” para los distintos tipos de datos

# SQL Parametrizado

Agregando más variables

```
String sql = "SELECT rut FROM Empleados  
             WHERE apellido = ? AND sueldo =?";  
stmt = db.prepareStatement(sql);  
  
apellido = leer("Ingrese el apellido");  
stmt.setString(1, apellido);  
sueldo = leer("Ingrese el sueldo");  
stmt.setString(2, sueldo);  
  
ResultSet rs = stmt.executeQuery();  
while (rs.next()){  
    System.out.println("rut: " + rs.getString(1));  
}
```

# SQL Parametrizado

Python

```
query = "SELECT * FROM Students  
        WHERE edad =%(edad)d  AND nombre = %(nombre)s"  
cursor.execute(query,  
                {"nombre": "Rivera", "edad": 11 })
```

Pero profesor, ¿qué tiene de malo concatenar strings para generar consultas?

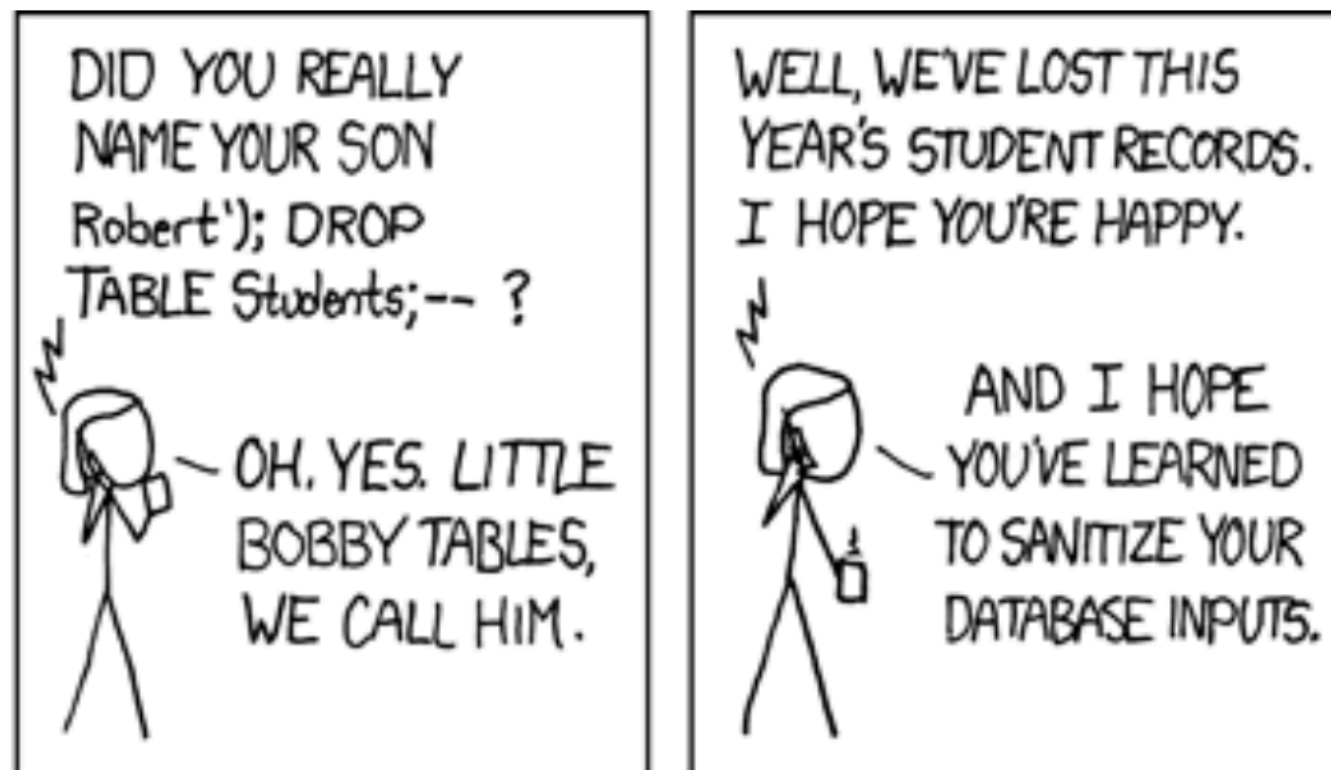
# Por qué no concatenar

```
query =  
    "INSERT INTO Students (edad, nombre) VALUES (" +  
        "'" + edad + "'" +  
        "'" + nombre + "'" +  
cursor.execute(query)
```

# Por qué no concatenar

<http://xkcd.com/327/>

```
query =  
    "INSERT INTO Students (edad, nombre) VALUES (" +  
        "'" + edad + "'" +  
        "'" + nombre + "'" +  
cursor.execute(query)
```





# Comparación con Procedimientos Almacenados

# Comparación con Procedimientos Almacenados

¡Son distintos! Procedimientos ejecutan el código en el DBMS

Ventajas:

- Muy rápidos, trabajan nativamente con tablas
- Abstracción

Desventajas:

- Poco portables
- Si cambiamos el DBMS, probablemente debamos rehacer el código