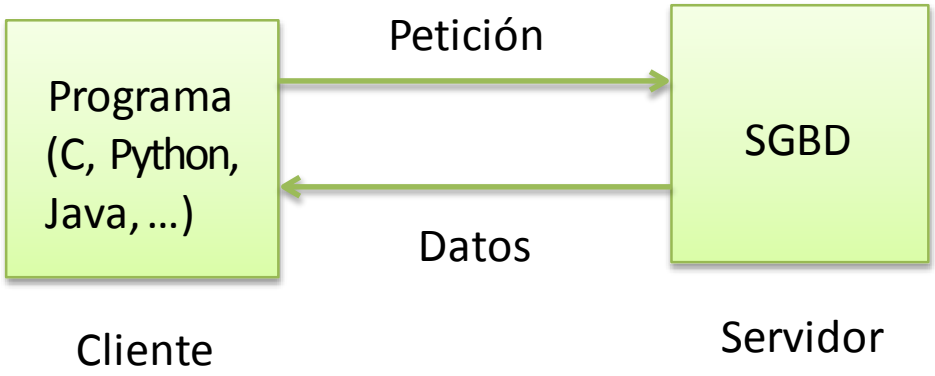
Introducción a SQL embebido

Contenidos

- Introducción
- Conexión a la base de datos
- Ejecución de consultas

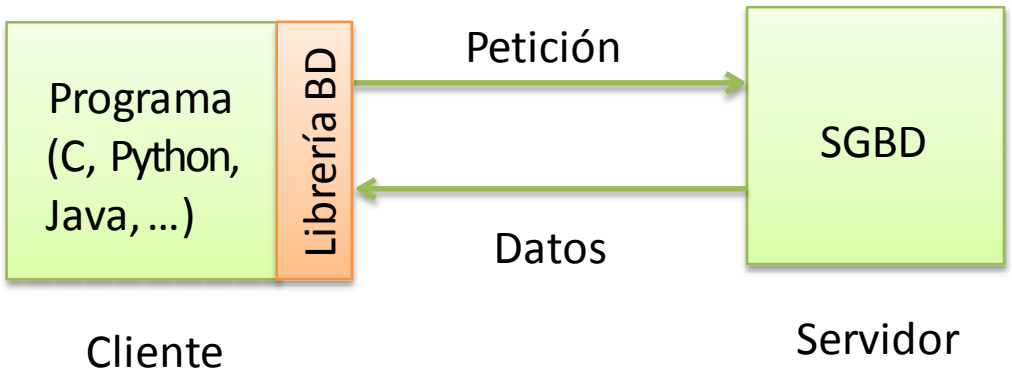
Introducción

- ¿Cómo podemos acceder a un SGBD desde nuestros programas?



Introducción

- Para acceder a un SGBD, utilizamos librerías que nos proporcionan funciones de alto nivel para:
  - Obtener conexiones al SGBD
  - Enviar peticiones y recibir su resultado
  - Gestionar la interacción (ej. gestión de transacciones)



Introducción

- **Psycopg**
  - Es el adaptador a PostgreSQL más popular para Python.
  - Implementa la API 2.0 de Python.

Introducción

- Instalar **Psycopg**
  - Descarga `psycopg2-2.6.1.win32-py2.7-pg9.4.4-release.exe` de Moodle.
  - Abre un terminal (Inicio+cmd)
  - Colócate en el directorio donde está `psycopg2-2.6.1.win32-py2.7-pg9.4.4-release.exe`
  - Ejecuta
    - `easy_install psycopg2-2.6.1.win32-py2.7-pg9.4.4-release.exe`

Conexión a la BD

```
import psycopg2

conn =
psycopg2.connect("dbname=‘<Nombre BD>'
user=‘<usuario>' host=‘<host>'
password=‘<password>'")
```

Ejemplo conexión

```
#!/usr/bin/python2.4
#
# Small script to show PostgreSQL and Pyscopg together
#

import psycopg2

try:
    conn = psycopg2.connect("dbname=‘Practicas'
user=‘postgres' host='localhost' password=‘<Vuestra Pass>'")

    print “Conectado!”

except:
    print "I am unable to connect to the database"
```

Ejecución de consultas

- Ejecución de consultas
  - Para ejecutar una consulta tenemos crear un cursor:

```
cur = conn.cursor()
```

  - Ahora podemos ejecutar una consulta:

```
cur.execute("SELECT * from emp ")
```

Ejecución de consultas

- Esto trae TODAS las filas y las mete en una lista:

```
rows = cur.fetchall()
```

- Ahora el resultado de la consulta está dentro la variable que llamamos `rows`
- Ahora tenemos que iterar:

```
for row in rows:
    print "   ", row[0], " ", row[1]
```

- row[0] obtiene la primera columna, row[1] la segunda. etc

Ejecución de consultas

- Para imprimir toda la fila:

```
for row in rows:
  print row
```

Ejecución de consultas

- La opción anterior se trae al cliente todas las filas, esto puede no ser posible por no tener memoria RAM suficiente el cliente. Para evitar este problema:

```
while True:

        row = cur.fetchone()

        if row == None:
            break

        print row[0], row[1], row[2]
```

Ejecución de consultas

- Para podernos referir a las columnas por su nombre debemos crear el cursor como sigue

```
cur =
conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
```

- Y ahora nos referimos a las columnas con

```
row[“<nombre>"]
```

- Hay que incluir al principio

```
import psycopg2.extras
```

Consultas parametrizadas

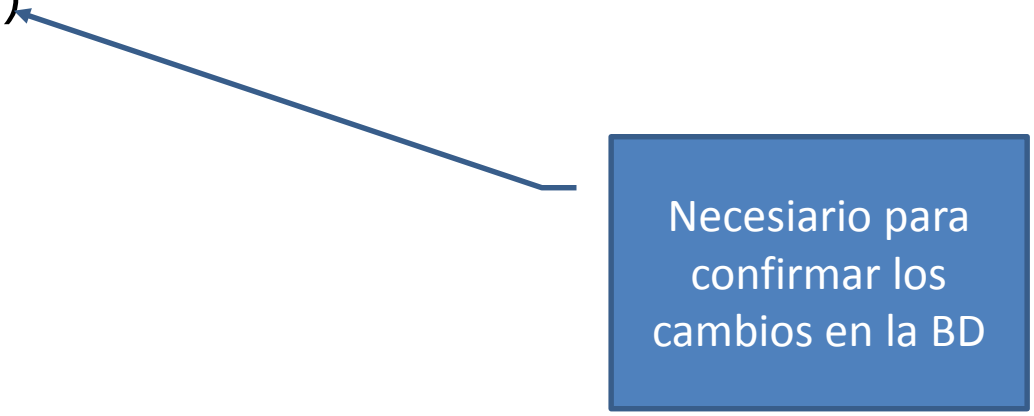
- Se coloca %s donde vamos colocar un valor proveniente de una variable, y luego entre paréntesis, separado por comas, las variables origen de los valores:

```
cur.execute(“Select * from emp where
job=%s and deptno=%s", (trabajo, dept))
```

Sentencias no consultas

```
cur.execute(“
INSERT INTO emp (empno, ename, job, hiredate, sal, comm,
deptno)
VALUES(9999,'pepe','CLERK','12-01-2015', 3000, NULL, 10)
")

conn.commit()
```



Sentencias no consultas parametrizadas

```
num_emp=9999
nombre='Pepe'
trabajo='CLERK'
fecha_con='12-01-2015'
salario=1000
comision=100
departamento=10

try:
    conn=psycopg2.connect("dbname='Practicas'
user='postgres' host='localhost' password='<password>'")
    cur = conn.cursor()
    cur.execute("INSERT INTO emp (empno, ename, job,
hiredate, sal, comm, deptno)
VALUES(%s,%s,%s,%s,%s,%s,%s)",(num_emp,nombre, trabajo,
fecha_con,salario,comision,departamento))
    conn.commit()
```