



Universidad del  
**Rosario**

Educación Continua  
y Consultoría

# Diplomado en Ciencia de Datos



#URSolucionesInnovadoras

#URConsultoría



Universidad del  
**Rosario**

Educación Continua  
y Consultoría

**BASES DE DATOS 3**

# **Conexión con Python**



---

# Contenido

- Introducción
- psycopg
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML





---

# Contenido

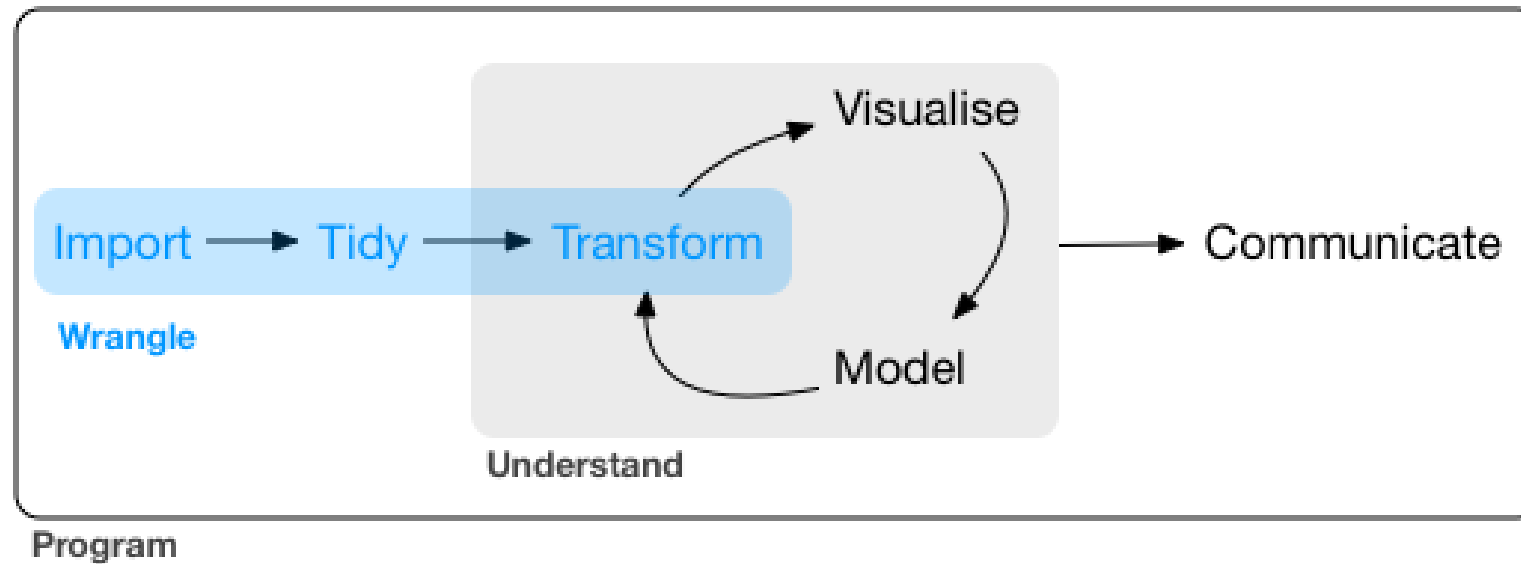
- **Introducción**
- **psql**
- **Python como cliente SQL**
- **SQLAlchemy**
- **Psycopg2**
- **Conexión a la base de datos**
- **Consultas SELECT desde Python**
- **Otras operaciones del DML**
- **Consideraciones extra sobre el DML**





## Introducción

# Flujo de trabajo de análisis





## Introducción

# ¿Dónde viven usualmente los datos?

## Archivos planos



\*Untitled - Notepad

File Edit Format View Help

```
"OrderID", "CustomerID", "OrderDate"  
"01", "001", "06/06/2021"  
"02", "369", "06/06/2021"  
"03", "151", "06/06/2021"  
"04", "014", "06/06/2021"  
"05", "061", "06/06/2021"  
"06", "220", "06/06/2021"
```





## Introducción

# ¿Dónde viven usualmente los datos?

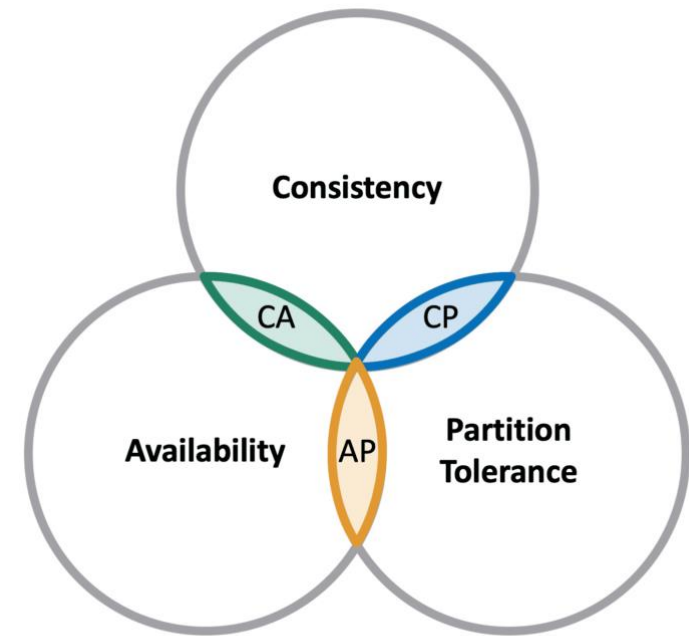
## Bases de datos

### Relacionales

- MySQL
- PostgreSQL
- Oracle
- IBM DB2
- otros

### NoSQL

- MongoDB
- Redis
- otros



Teorema CAP





Universidad del  
**Rosario**

Educación Continua  
y Consultoría

## Introducción

# ¿Dónde viven usualmente los datos?



## Web

The screenshot shows a web page titled "Municipios de Colombia por departamento". It features a table with columns for "Departamento", "Municipios", and "Áreas no municipalizadas". The "Amazonas" department is highlighted. Below the table, there is a description of Amazonas and a list of other departments. A DevTools console window is open, showing the HTML structure of the table, including a link to the Amazonas department and a link to the list of municipalities in Amazonas.

Departamento	Municipios	Áreas no municipalizadas
<a href="#">Amazonas</a>	2	9
<a href="#">Antioquia</a>		
<a href="#">Arauca</a>		
<a href="#">Atlántico</a>		
<a href="#">Bolívar</a>		
<a href="#">Boyacá</a>		
<a href="#">Caldas</a>		
<a href="#">Caquetá</a>		
<a href="#">Casana</a>		
<a href="#">Cauca</a>		
<a href="#">Cesar</a>		
<a href="#">Chocó</a>		
<a href="#">Córdoba</a>		

Amazonas es uno de los departamentos que, junto con la Capital, forman la República. Su capital es Leticia. Está situado en el sur del país, en gran parte en la zona ecuatorial, en la región del norte con Caquetá y Vichada.

```
<tr>
  <td>
    <span class="flagicon">...</span>
    <a href="/wiki/Amazonas_(Colombia)" title="Amazonas">Amazonas</a> == $0
  </td>
  <td>
    <a href="/wiki/Anexo:Municipios de Amazonas_(Colombia)" title="Anexo:Municipios de Amazonas (Colombia)">2</a>
  </td>
  <td>
    <a href="/wiki/Anexo:Municipios de Amazonas_(Colombia)" title="Anexo:Municipios de Amazonas (Colombia)">9</a>
  </td>
</tr>
```

HTML

```
{
  "departamento":8,
  "nombredepto":"Ventas",
  "director": "Juan Rodríguez",
  "empleados":[
    {
      "nombre":"Pedro",
      "apellido":"Fernández"
    },{
      "nombre":"Jacinto",
      "apellido":"Benavente"
    }
  ]
}
```

json





---

# Contenido

- Introducción
- **psql**
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML





---

psql

# La terminal de psql



```
SQL Shell (psql)
Database [postgres]: dvdrental
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (11.8, server 13.4)
WARNING: psql major version 11, server major version 13.
         Some psql features might not work.
WARNING: Console code page (437) differs from windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for windows users" for details.
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, b
its: 256, compression: off)
Type "help" for help.

dvdrental=> _
```



psql

# La terminal de psql



Result set en psql



```
dvdrental=> SELECT * FROM category;
category_id | name          | last_update
-----+-----+-----
1           | Action        | 2006-02-15 09:46:27
2           | Animation     | 2006-02-15 09:46:27
3           | Children      | 2006-02-15 09:46:27
4           | Classics      | 2006-02-15 09:46:27
5           | Comedy        | 2006-02-15 09:46:27
6           | Documentary   | 2006-02-15 09:46:27
7           | Drama         | 2006-02-15 09:46:27
8           | Family        | 2006-02-15 09:46:27
9           | Foreign       | 2006-02-15 09:46:27
10          | Games         | 2006-02-15 09:46:27
11          | Horror        | 2006-02-15 09:46:27
12          | Music         | 2006-02-15 09:46:27
13          | New           | 2006-02-15 09:46:27
14          | Sci-Fi        | 2006-02-15 09:46:27
15          | Sports        | 2006-02-15 09:46:27
16          | Travel        | 2006-02-15 09:46:27
(16 rows)
```



---

# Contenido

- Introducción
- `psql`
- **Python como cliente SQL**
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML





Universidad del  
**Rosario**

Educación Continua  
y Consultoría

Python como cliente SQL

# La función `pandas.read_sql`



`pandas.read_sql`



`pandas.read_sql_query`



`pandas.read_sql_table`



## Python como cliente SQL

# Parámetros de read\_sql

```
pandas.read_sql(sql, con, index_col=None, coerce_float=True, params=None, parse_dates=None, columns=None, chunksize=None)
```

**sql** : *str or SQLAlchemy Selectable (select or text object)*

SQL query to be executed or a table name.

**con** : *SQLAlchemy connectable, str, or sqlite3 connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable; str connections are closed automatically. See [here](#).

**index\_col** : *str or list of str, optional, default: None*

Column(s) to set as index(MultiIndex).

**coerce\_float** : *bool, default True*

Attempts to convert values of non-string, non-numeric objects (like decimal.Decimal) to floating point, useful for SQL result sets.

**params** : *list, tuple or dict, optional, default: None*

List of parameters to pass to execute method. The syntax used to pass parameters is database driver dependent. Check your database driver documentation for which of the five syntax styles, described in PEP 249's paramstyle, is supported. Eg. for psycopg2, uses %(name)s so use params={'name': 'value'}.

**parse\_dates** : *list or dict, default: None*

- List of column names to parse as dates.
- Dict of {column\_name: format string}, where format string is strftime compatible in case of parsing string times, or is one of (D, s, ns, ms, us) in case of parsing integer timestamps.
- Dict of {column\_name: arg dict}, where the arg dict corresponds to the keyword arguments of `pandas.to_datetime()` Especially useful with databases without native Datetime support, such as SQLite.

**columns** : *list, default: None*

List of column names to select from SQL table (only used when reading a table).

**chunksize** : *int, default None*

If specified, return an iterator where *chunksize* is the number of rows to include in each chunk.





## Python como cliente SQL

# ¿SQLAlchemy?

```
pandas.read_sql(sql, con, index_col=None, coerce_float=True, params=None,  
parse_dates=None, columns=None, chunksize=None)
```

**sql** : *str* or *SQLAlchemy Selectable* (*select* or *text* object)

SQL query to be executed or a table name.

**con** : *SQLAlchemy connectable*, *str*, or *sqlite3 connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable; str connections are closed automatically. See [here](#).



---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- **SQLAlchemy**
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML







## SQLAlchemy

# ¿Qué es?

- Kit de herramientas SQL de Python
- Tiene un mapeador relacional de objetos (ORM)

Relational database (such as PostgreSQL or MySQL)

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...	...	...	...

Python objects

```
class Person:  
    first_name = "John"  
    last_name = "Connor"  
    phone_number = "+16105551234"
```

```
class Person:  
    first_name = "Matt"  
    last_name = "Makai"  
    phone_number = "+12025555689"
```

```
class Person:  
    first_name = "Sarah"  
    last_name = "Smith"  
    phone_number = "+19735554512"
```

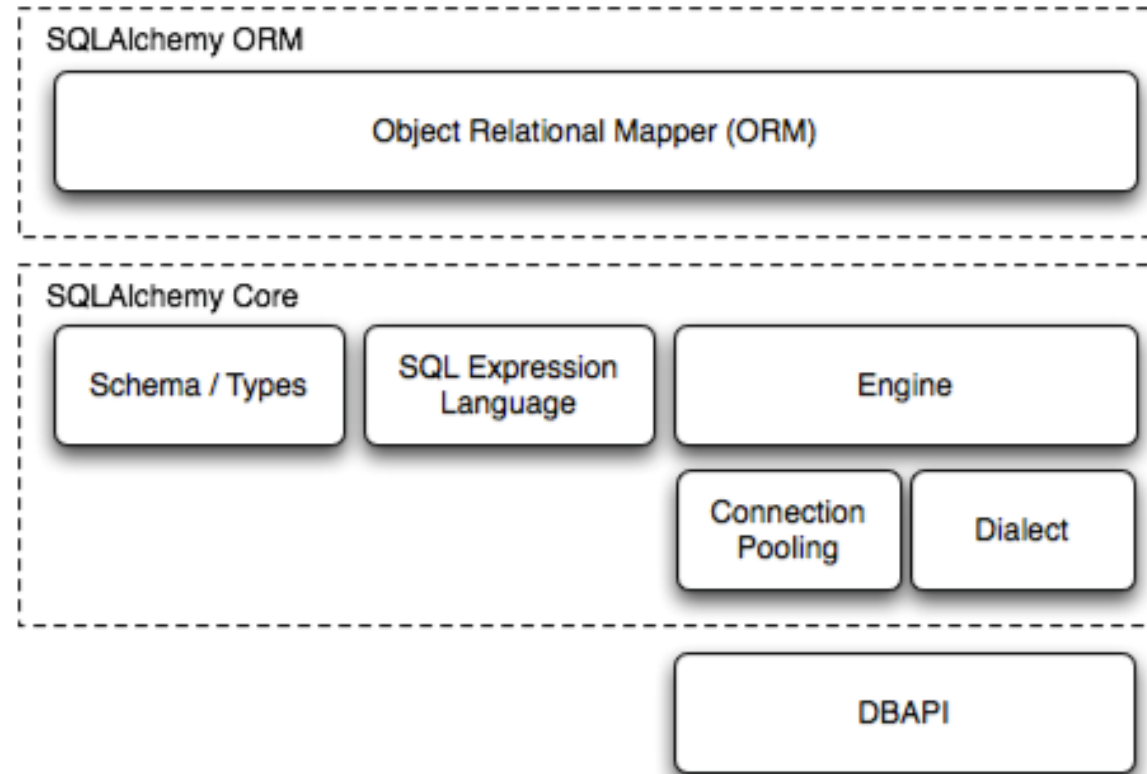
ORMs provide a bridge between  
**relational database tables, relationships  
and fields** and **Python objects**





## SQLAlchemy

# Componentes





## SQLAlchemy

# Instalación

- En la terminal se ejecuta:

```
pip install SQLAlchemy
```

- Si se instala desde Jupyter o desde Colab, se ejecuta:

```
!pip install SQLAlchemy
```

- Se comprueba la instalación con las siguientes líneas

```
import sqlalchemy  
sqlalchemy.__version__
```





## SQLAlchemy

# Dialectos

- Sistemas que usa SQLAlchemy para comunicarse con varios tipos de implementaciones y bases de datos DBAPI.
- Todos los dialectos requieren que se instale un controlador **DBAPI** adecuado.



### Support Levels for Included Dialects

The following table summarizes the support level for each included dialect.

Database	Fully tested in CI	Normal support	Best effort
Microsoft SQL Server	2017	2012+	2005+
MySQL / MariaDB	5.6, 5.7, 8.0 / 10.4, 10.5	5.6+ / 10+	5.0.2+ / 5.0.2+
Oracle	11.2, 18c	11+	8+
PostgreSQL	9.6, 10, 11, 12, 13, 14	9.6+	8+
SQLite	3.21, 3.28+	3.12+	3.7.16+



Universidad del  
**Rosario**

Educación Continua  
y Consultoría

SQLAlchemy

# Consulta

```
with engine.connect() as connection:  
    result = connection.execute("SELECT * FROM category;")  
    for row in result:  
        print("categoria:", row['name'])
```





---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- SQLAlchemy
- **Psycopg2**
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML





## psycopg2

# ¿Qué es?

- Adaptador de base de datos PostgreSQL más popular para el lenguaje de programación Python.
- Tiene implementación completa de la especificación Python DB API 2.0 y la seguridad de subprocessos (varios subprocessos pueden compartir la misma conexión).
- Fue diseñado para aplicaciones con múltiples subprocessos que crean y destruyen muchos cursores y hacen una gran cantidad de "INSERT" o "UPDATE" simultáneos.







psycopg2

# Instalación

- En la terminal se ejecuta:

```
pip install psycopg2
```

- Si se instala desde Jupyter o desde Colab, se ejecuta:

```
!pip install psycopg2
```

- Se comprueba la instalación con las siguientes líneas

```
import sqlalchemy  
sqlalchemy.__version__
```







---

psycopg2

## Otros BDAPIs similares

- pg8000

<https://github.com/tlocke/pg8000>

- asyncpg

<https://magicstack.github.io/asyncpg/current/>

- pygresql

<http://www.pygresql.org/>





---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- **Conexión a la base de datos**
- Consultas SELECT desde Python
- Otras operaciones del DML
- Consideraciones extra sobre el DML





## Conexión a la base de datos

# Cadena de conexión

```
postgresql+psycopg2://user:password@host:port/dbname
```

**con** : *SQLAlchemy connectable, str, or sqlite3 connection*

Using SQLAlchemy makes it possible to use any DB supported by that library. If a DBAPI2 object, only sqlite3 is supported. The user is responsible for engine disposal and connection closure for the SQLAlchemy connectable; str connections are closed automatically. See [here](#).

```
from sqlalchemy import create_engine

engine = create_engine('postgresql+psycopg2://postgres:MEeLaN2z@database-1.cdqqgt4oejnt.us-west-2.rds.amazonaws.com:5432/dvdrental')

pd.read_sql("SELECT * FROM category;", engine)
```



---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- **Consultas SELECT desde Python**
- Otras operaciones del DML
- Consideraciones extra sobre el DML





## Consultas SELECT desde Python

# Consultas con una tabla

- Número de actores o actrices registrados
- Número de actores o actrices registrados por inicial del apellido
- Cantidad de clientes activos e inactivos
- Número de películas por rating
- Duración promedio de las películas por rating





## Consultas SELECT desde Python

# Consultas con varias tablas

- Número de ciudades por país
- Número de clientes por ciudad
- Cantidad total recaudada por ciudad
- Cantidad total recaudada por país
- Número de películas registradas en cada categoría





Universidad del  
**Rosario**

Educación Continua  
y Consultoría

---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- **Otras operaciones del DML**
- Consideraciones extra sobre el DML





## Otras operaciones del DML

# CRUD

**DML:** Lenguaje de manipulación de datos

- Son comandos que no afectan la estructura de la base de datos
- Pueden afectar el contenido de la base de datos o extraer información de la base de datos

## Operaciones:

CREAR, LEER, MODIFICAR, BORRAR datos en las tablas



CREATE

C



READ

R



UPDATE

U



DELETE

D





## Otras operaciones del DML

# Actualizar la base de datos con INSERT

Para insertar una columna se usa el commando **INSERT INTO**, de acuerdo a la siguiente sintaxis:

```
INSERT INTO nombreTabla (col1, col2, ... )  
VALUES (valor1, valor2,...);
```

- **nombreTabla**: Tabla donde se agregará el registro
- **col1, col2, ...** : Nombre de las columnas de la table
- **valor1, valor2, ...** : Valores que irán en la nueva tupla



## Otras operaciones del DML

# Un poco de DDL

movies	
	title
	year
	length
	genre
	studioName
	producerC

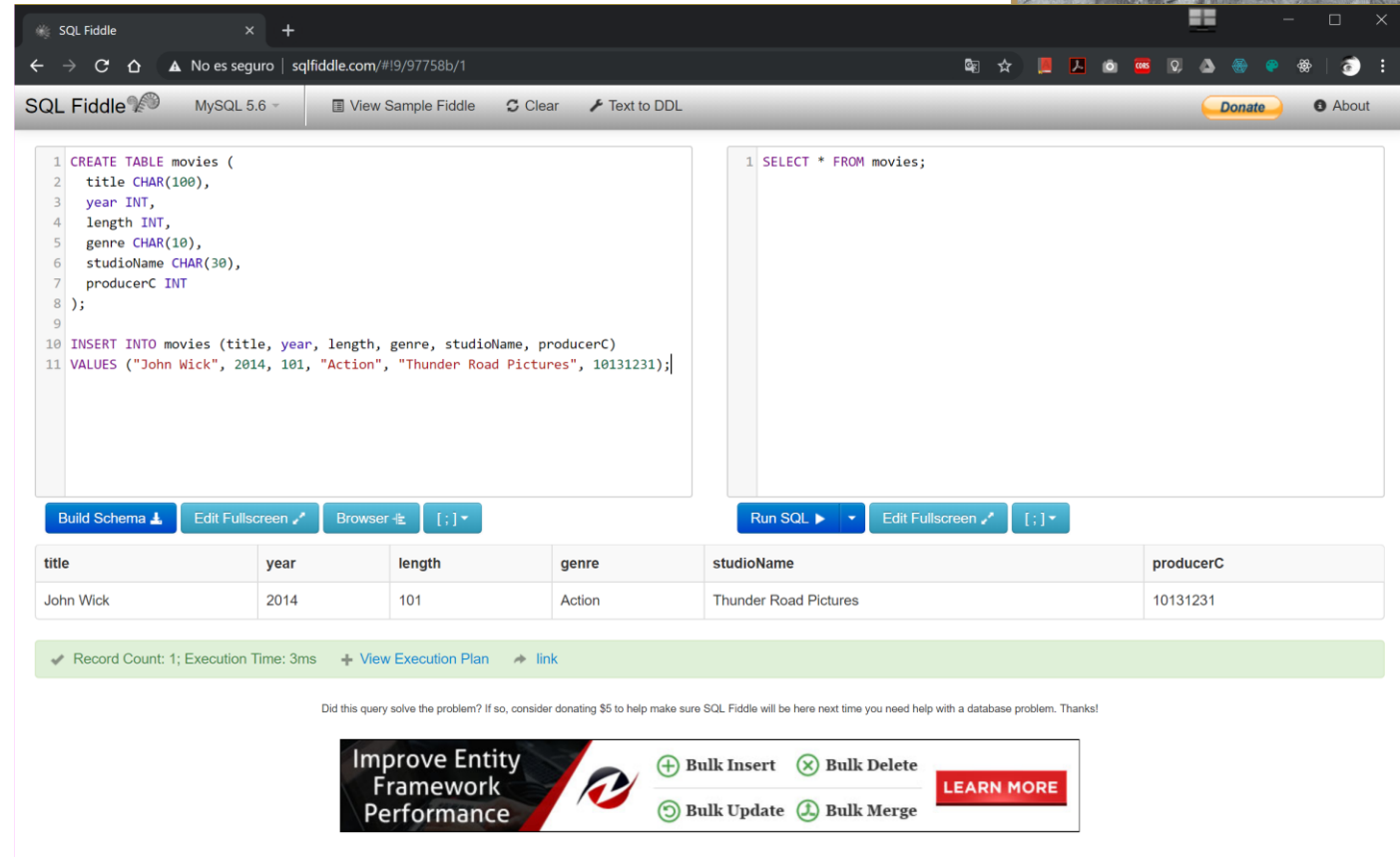
```
CREATE TABLE movies (  
  title CHAR(100),  
  year INT,  
  length INT,  
  genre CHAR(10),  
  studioName CHAR(30),  
  producerC INT  
);
```





## Otras operaciones del DML

# Actualizar la base de datos con INSERT



The screenshot shows the SQL Fiddle web application interface. The left pane contains the following SQL code:

```
1 CREATE TABLE movies (  
2   title CHAR(100),  
3   year INT,  
4   length INT,  
5   genre CHAR(10),  
6   studioName CHAR(30),  
7   producerC INT  
8 );  
9  
10 INSERT INTO movies (title, year, length, genre, studioName, producerC)  
11 VALUES ("John Wick", 2014, 101, "Action", "Thunder Road Pictures", 10131231);
```

The right pane contains the query:

```
1 SELECT * FROM movies;
```

Below the code panes, the "Run SQL" button has been clicked. A table displays the result of the query:

title	year	length	genre	studioName	producerC
John Wick	2014	101	Action	Thunder Road Pictures	10131231

A green status bar at the bottom indicates: "Record Count: 1; Execution Time: 3ms". Below this, there is a small text prompt: "Did this query solve the problem? If so, consider donating \$5 to help make sure SQL Fiddle will be here next time you need help with a database problem. Thanks!". At the very bottom, there is a banner for "Improve Entity Framework Performance" with links for Bulk Insert, Bulk Delete, Bulk Update, and Bulk Merge, and a "LEARN MORE" button.

```
INSERT INTO movies (title, year, length, genre, studioName, producerC)  
VALUES ("John Wick", 2014, 101, "Action", "Thunder Road Pictures", 10131231);
```

<http://sqlfiddle.com/>



Otras operaciones del DML

# Actualizar la base de datos con INSERT

**INSERT** sencillo

```
INSERT INTO orders  
  (order_id, order_date, amount, customer_id)  
VALUES (34, '03/14/1760', 45.6, 1);
```

**INSERT** con tabla como parámetro

```
INSERT INTO orders  
  (order_id, order_date,  
   amount, customer_id)  
VALUES  
  (SELECT * FROM orders WHERE order_id = 1);
```



## Otras operaciones del DML

# La sentencia UPDATE

Actualizar todas las filas

```
UPDATE orders  
SET amount = amount * 1.10;
```

Actualizar todas las filas que correspondan con la clausula **WHERE**

```
UPDATE orders  
SET amount = amount * 1.10  
WHERE customer_id = 1;
```

Actualizar columnas múltiples

```
UPDATE orders  
SET amount = amount * 1.10, order_date = REPLACE(order_date, '/', '-')  
WHERE customer_id = 1;
```



## Otras operaciones del DML

# La sentencia DELETE

Borra todas las filas 

```
DELETE FROM orders;
```

Borra todas las filas que correspondan con la clausula **WHERE**

```
DELETE FROM orders  
WHERE customer_id = 1;
```





---

# Contenido

- Introducción
- psql
- Python como cliente SQL
- SQLAlchemy
- Psycopg2
- Conexión a la base de datos
- Consultas SELECT desde Python
- Otras operaciones del DML
- **Consideraciones extra sobre el DML**





Universidad del  
**Rosario**

Educación Continua  
y Consultoría

Consideraciones extra sobre el DML

# Ejecución de operaciones **CRUD**

Creación de la conexión mediante psycopg2

```
import psycopg2

conn = psycopg2.connect(host=hostname,
                        port=port,
                        user=username,
                        password=password,
                        database=database)
```







## Consideraciones extra sobre el DML

# El elemento cursor

Permite que el código Python ejecute el comando PostgreSQL en una sesión de base de datos.

Se crea por el método **connection.cursor()**

```
cursor = conn.cursor()
cursor.execute("INSERT INTO a_table (c1, c2, c3)
VALUES(%s, %s, %s)", (v1, v2, v3))
```



## Consideraciones extra sobre el DML

# El elemento cursor

```
cursor = conn.cursor()  
cursor.execute("INSERT INTO a_table (c1, c2, c3)  
VALUES(%s, %s, %s)", (v1, v2, v3))
```

Después de la inserción se deben confirmar (commit) los cambios

```
conn.commit()
```

Al final se debe cerrar la conexión a la base de datos

```
cursor.close()  
conn.close()
```

```
import psycopg2

# Se conecta a una base de datos existente
conn = psycopg2.connect("dbname=test user=postgres")

# Abre un cursor para ejecutar operaciones en la base de datos
cur = conn.cursor()

# Ejecuta un comando DDL
cur.execute("CREATE TABLE test (id serial PRIMARY KEY, num integer,
data varchar);")

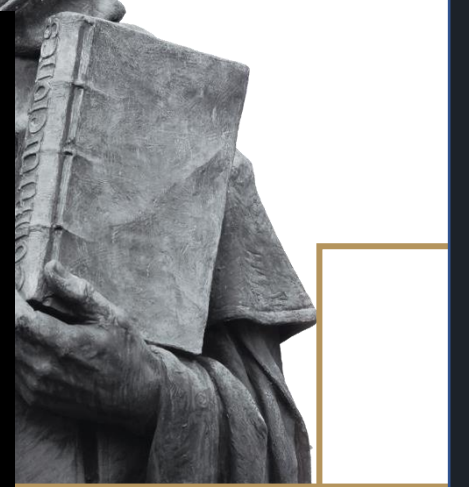
# Pasa los datos por posición (Evita SQL Injections)
cur.execute("INSERT INTO test (num, data) VALUES (%s, %s)",(100,
"abc'def"))

# Consulta la base de datos y obtiene objetos de Python
cur.execute("SELECT * FROM test;")
cur.fetchone()

# Puede iterar sobre un result set
cur.execute("SELECT * FROM test;")
for record in cur:
    print(record)

# Hace que los cambios sean persistentes
conn.commit()

# Cierra la comunicación con la base de datos
cur.close()
conn.close()
```



Unidad de Educación Continua y Consultoría  
construimos país desde

**#URSolucionesInnovadoras**  
**#URConsultoría**



@RosarioContinua



/EduContinuaURosario



@RosarioContinua