

Bases de Datos

Clase 10: Programando con 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.
- En esta clase veremos dos formas de programar conectado a una base de datos.

SQL en Python

Conexión a la Base de Datos

- La forma más simple de usar una base de datos desde un programa es usando una librería especializada para el motor que estemos usando.
- Por ejemplo en Python existen `sqlite3` y `psycopg2` (para postgres).

Conexión a la Base de Datos

Python

Por ejemplo, para sqlite importamos la librería correspondiente e iniciamos una conexión a la db:

```
import sqlite3  
  
db_connection = sqlite3.connect('some_db.db')  
  
# hacer cosas con la db  
  
db_connection.close() # cerramos la conexion
```

La conexión significa que la base de datos está esperando instrucciones por parte de nuestro programa.

Cursores

Python

Para ejecutar comandos SQL desde Python, nos falta instanciar un **cursor**.

Un objeto de la clase cursor nos permite ejecutar un comando SQL en Python y representa un “puntero” al output del comando.

Modificando la DB

Python

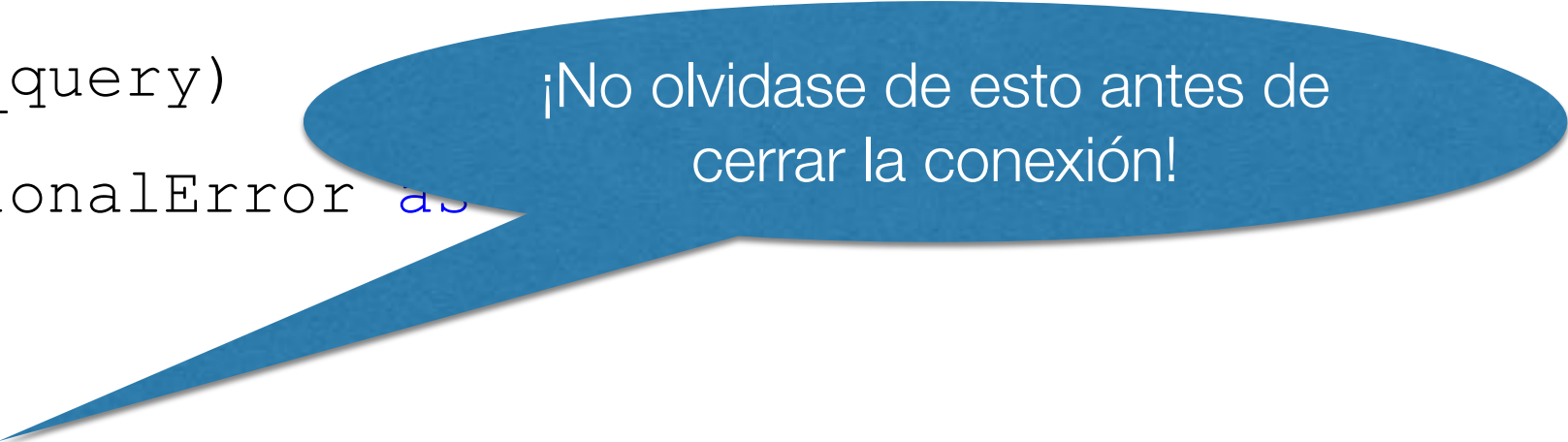
```
cur = db_connection.cursor() # instanciamos el cursor
create_query = "CREATE TABLE Peliculas(id INT, titulo
VARCHAR(100));"
insert_query = "INSERT INTO Peliculas VALUES (1, 'Minions');"
try:
    cur.execute(create_query)
    cur.execute(insert_query)
except sqlite3.OperationalError as e:
    print(e)

db_connection.commit() # guardamos los cambios en disco.
db_connection.close() # cerramos la conexi3n
```

Modificando la DB

Python

```
cur = db_connection.cursor() # instanciamos el cursor
create_query = "CREATE TABLE Peliculas(id INT, titulo
VARCHAR(100));"
insert_query = "INSERT INTO Peliculas VALUES (1, 'Minions');"
try:
    cur.execute(create_query)
    cur.execute(insert_query)
except sqlite3.OperationalError as e:
    print(e)
```



¡No olvidase de esto antes de cerrar la conexión!

```
db_connection.commit() # guardamos los cambios en disco.
db_connection.close() # cerramos la conexión
```


Consultas básicas

Python

```
select_query = "SELECT * FROM Peliculas"
try:
    cur.execute(select_query)
except sqlite3.OperationalError as e:
    print(e)
```

... y cómo accedemos al resultado? 🤔

Cursores

El **Cursor** representa un puntero que va recorriendo los resultados. Podemos usar el método `fetchone()` para traer uno o `fetchall()` para traerlos todos (Ojo con la memoria! 🤯).

Adicionalmente en Python el Cursor está implementado como un iterable.

Cursores

Obteniendo resultados de consultas

```
select_query = "SELECT * FROM Peliculas"
```

```
cur.execute(select_query)
```

```
first_row = cur.fetchone()
```

```
second_row = cur.fetchone()
```

```
print(first_row)
```

```
print(second_row)
```

```
cur.execute(select_query)
```

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

```
print(rows)
```

```
cur.execute(select_query)
```

```
for row in cur:
```


```
    print(row)
```

Ahora lo importante

- Usuarios de mi aplicación ingresan datos
- Basado en el input se hace una consulta a mi base de datos y retorno algo.

Un ejemplo inocente...

```
id = input("id de la película?")
select_query = "SELECT * FROM Peliculas WHERE id="+id
cur.execute(select_query)
print(cur.fetchone())
print("Exitó!")
```

 <https://www.psycopg.org/docs/usage.html>

Psycopg can automatically convert Python objects to and from SQL literals: using this feature your code will be more robust and reliable. We must stress this point:

Warning: Never, **never**, **NEVER** use Python string concatenation (+) or string parameters interpolation (%) to pass variables to a SQL query string. Not even at gunpoint.

The correct way to pass variables in a SQL command is using the second argument of the `execute()` method:

```
>>> SQL = "INSERT INTO authors (name) VALUES (%s);" # Note: no quotes
>>> data = ("O'Reilly", )
>>> cur.execute(SQL, data) # Note: no % operator
```

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

Un ejemplo inocente...

```
id = input("id de la película?")
select_query = "SELECT * FROM Peliculas WHERE id="+id
cur.execute(select_query)
print(cur.fetchone())
print("Exitó!")
```

```
>> id de la película? 1; DROP TABLE Peliculas; --
```

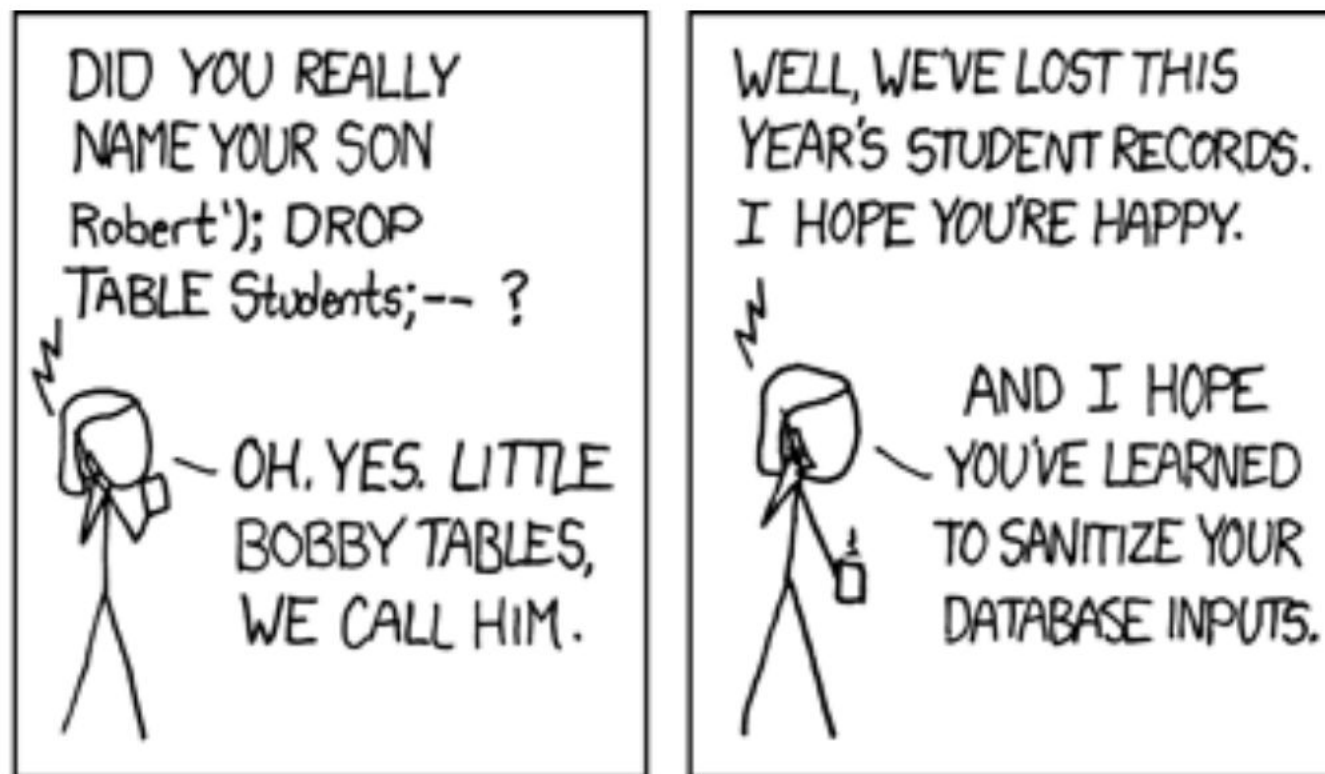
Un ejemplo inocente...

```
id = input("id de la película?")
select_query = "SELECT * FROM Peliculas WHERE id="+id
cur.execute(select_query)
print(cur.fetchone())
print("Exitó!")
```

```
>> id de la película? 1; DROP TABLE Peliculas; --
```


Por qué no concatenar

```
query = "SELECT * FROM Students WHERE name = " + nombre  
  
cursor.execute(query)
```



Nos exponemos a una forma de ataque: [SQL Injection](#)

Por qué no concatenar
Good luck speed cameras.



SQL Parametrizado

Python

Si van a usar parámetros para una consulta provenientes del input de usuario, hay que hacerlo así:

```
query = "SELECT * FROM Students \
        WHERE edad =%(edad)d  AND nombre = %(nombre)s"

cursor.execute(query, {"nombre": "Rivera", "edad": 11 })
```

A esto le llamamos “sanitizar” el input, dejamos que la librería parsee y asegure los parámetros antes de pasarlos a la DB.

ORMs

ORMs

- Del inglés *Object Relational Mapping*. Es un patrón de diseño altamente adoptado en desarrollo web, a través de librerías que vienen incluidas en los *frameworks*.
- Fundamentalmente consiste en tener para cada tabla de la DB una Clase (OOP) en el código de la aplicación.
- Nacen de la necesidad de sacar provecho a las ventajas de la programación orientada a objetos, manteniendo la estructura del esquema relacional que por debajo guarda los datos.
- Proporcionan una capa de abstracción sobre la base de datos.

ORMs

Ejemplos

- [ActiveRecord](#) del framework de Ruby, Ruby on Rails.
- El Framework web de Python: Django, también tiene el [suoy](#).
- En Javascript (Node) se suele usar [Sequelize](#).
- Hay disponibles para la mayoría de lenguajes / frameworks.

Veremos algunos componentes básicos de su uso.

Usaremos ActiveRecord para ejemplificar

Modelos

- Generalmente se le llama Modelos a las clases que representan las tablas en el código.
- Cada modelo tiene como atributos de instancia a cada columna de la tabla, además de varios métodos para manejo de los datos.
- También tienen métodos de clase para hacer consultas simples.

Modelos

```
1 class Deposit < ApplicationRecord
2
3   def bananas
4     Money.from_amount(amount, 'BAN')
5   end
6 end
7
8 # == Schema Information
9 #
10 # Table name: deposits
11 #
12 #   id          :bigint(8)        not null, primary key
13 #   monkey_id   :bigint(8)        not null
14 #   amount      :integer
15 #   created_at  :datetime         not null
16 #   updated_at  :datetime         not null
17 #
18 # Indexes
19 #
20 #   index_deposits_on_monkey_id (monkey_id)
21 #
22 # Foreign Keys
23 #
24 #   fk_rails_... (monkey_id => monkeys.id)
25 #
```


Modelos

```
1 class Deposit < ApplicationRecord
2
3   def bananas
4     Money.from_amount(amount, 'BAN')
5   end
6 end
7
8 # == Schema Information
9 #
10 # Table name: deposits
11 #
12 #   id          :bigint(8)      not null, primary key
13 #   monkey_id   :bigint(8)      not null
14 #   amount      :integer
15 #   created_at  :datetime       not null
16 #   updated_at  :datetime       not null
17 #
18 # Indexes
19 #
20 #   index_deposits_on_monkey_id (monkey_id)
21 #
22 # Foreign Keys
23 #
24 #   fk_rails_... (monkey_id => monkeys.id)
25 #
```

Clase que hereda de la clase base del ORM.

Método del modelo.

Esquema de la tabla asociada al modelo.

Modelos

```
1 class Deposit < ApplicationRecord
2
3   def bananas
4     Money.from_amount(amount, 'BAN')
5   end
6 end
7
8 # == Schema Information
9 #
10 # Table name: deposits
11 #
12 #  id      :bigint(8)      not null, primary key
13 #  monkey_id :bigint(8)    not null
14 #  amount   :integer
15 #  created_at :datetime    not null
16 #  updated_at :datetime    not null
17 #
18 # Indexes
19 #
20 #  index_deposits_on_monkey_id (monkey_id)
21 #
22 # Foreign Keys
23 #
24 #  fk_rails_... (monkey_id => monkeys.id)
25 #
```



A yellow box highlights the 'amount' parameter in the 'bananas' method on line 4. A yellow arrow points from this box to another yellow box that highlights the 'amount' column in the schema information on line 14. This illustrates how the ORM maps the method's argument to the database column.

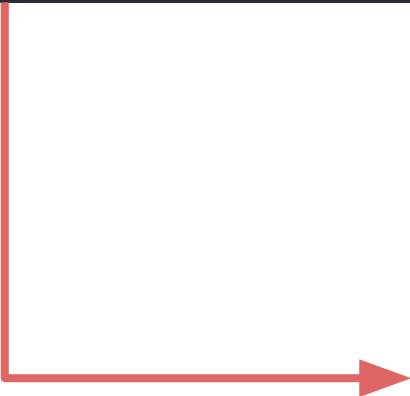
Veamos lo que hace el ORM en vivo

Migraciones

- Las migraciones son “scripts” encargadas de generar los cambios en el esquema de la DB. Es decir traducen código en el lenguaje de programación, a las sentencias CREATE, ALTER o DELETE TABLE de SQL.
- Algunos ORMs las generan automáticamente a partir del código del Modelo respectivo o mediante comandos de consola.
- Para hacer cosas más complejas (como restricciones de integridad) suele ser necesario programarlas a mano.
- Cada migración representa un cambio pequeño en el esquema, lo que ayuda a la mantenibilidad de este y facilita el desarrollo del software.

Migraciones

```
1 class CreateDeposits < ActiveRecord::Migration[6.0]
2   def change
3     create_table :deposits do |t|
4       t.references :monkey, null: false, foreign_key: true
5       t.integer :amount
6
7       t.timestamps
8     end
9   end
10 end
```



```
CREATE TABLE deposits(
  id SERIAL,
  amount int,
  monkey_id int NOT NULL,
  created_at date,
  updated_at date,
  PRIMARY KEY (id),
  FOREIGN KEY(monkey_id) REFERENCES monkeys(id)
)
```

Asociaciones

- El ORM relaciona los modelos unos con otros de la misma forma que las tablas (1:N, N:N, 1:1).
- Asociar los modelos a nivel de código facilita la manera de hacer consultas.
- Podemos hacer cosas tipo `deposit.monkey` o `monkeys.deposits`

Asociaciones

Ejemplo N:1

```
1 class Deposit < ApplicationRecord
2   include PowerTypes::Observable
3   include LedgerizerDocument
4
5   belongs_to :monkey
6
7   def bananas
8     Money.from_amount(amount, 'BAN')
9   end
10 end
11
12 # == Schema Information
13 #
14 # Table name: deposits
15 #
16 #   id          :bigint(8)          not null, primary key
17 #   monkey_id   :bigint(8)          not null
18 #   amount      :integer
19 #   created_at  :datetime           not null
20 #   updated_at  :datetime           not null
21 #
22 # Indexes
23 #
24 #   index_deposits_on_monkey_id (monkey_id)
25 #
26 # Foreign Keys
27 #
28 #   fk_rails_... (monkey_id => monkeys.id)
29 #
```

```
1 class Monkey < ApplicationRecord
2   include LedgerizerAccountable
3
4   belongs_to :casino
5   has_many :deposits, dependent: :destroy
6   has_many :withdrawals, dependent: :destroy
7
8   def wallet_account
9     accounts.find_by(name: :wallet)
10   end
11 end
12
13 # == Schema Information
14 #
15 # Table name: monkeys
16 #
17 #   id          :bigint(8)          not null, primary key
18 #   casino_id  :bigint(8)          not null
19 #   name        :string
20 #   created_at  :datetime           not null
21 #   updated_at  :datetime           not null
22 #
23 # Indexes
24 #
25 #   index_monkeys_on_casino_id (casino_id)
26 #
27 # Foreign Keys
28 #
29 #   fk_rails_... (casino_id => casinos.id)
30 #
```


Asociaciones

Ejemplo N:1

```
1 class Deposit < ApplicationRecord
2   include PowerTypes::Observable
3   include LedgerizerDocument
4   belongs_to :monkey
5
6   def bananas
7     Money.from_amount(amount, 'BAN')
8   end
9 end
10
11 # == Schema Information
12 #
13 # Table name: deposits
14 #
15 #   id          :bigint(8)      not null, primary key
16 #   monkey_id   :bigint(8)      not null
17 #   amount      :integer
18 #   created_at  :datetime       not null
19 #   updated_at  :datetime       not null
20 #
21 # Indexes
22 #
23 #   index_deposits_on_monkey_id (monkey_id)
24 #
25 # Foreign Keys
26 #
27 #   fk_rails_... (monkey_id => monkeys.id)
28 #
29 #
30
1 class Monkey < ApplicationRecord
2   include LedgerizerAccountable
3
4   belongs_to :casino
5   has_many :deposits, dependent: :destroy
6   has_many :withdrawals, dependent: :destroy
7
8   def wallet_account
9     accounts.find_by(name: :wallet)
10  end
11 end
12
13 # == Schema Information
14 #
15 # Table name: monkeys
16 #
17 #   id          :bigint(8)      not null, primary key
18 #   casino_id   :bigint(8)      not null
19 #   name         :string
20 #   created_at   :datetime       not null
21 #   updated_at   :datetime       not null
22 #
23 # Indexes
24 #
25 #   index_monkeys_on_casino_id (casino_id)
26 #
27 # Foreign Keys
28 #
29 #   fk_rails_... (casino_id => casinos.id)
30 #
```

Veamos lo que hace el ORM en vivo

Pero profesor,
¿por qué aprendimos SQL si en la
práctica se usa un ORM?

- Los ORMs agregan una capa por encima de la DB. Por debajo se sigue escribiendo SQL y eso es lo que finalmente se ejecuta.
- El uso de la base de datos es un componente clave de cada aplicación, si no entendemos las consultas que el ORM está ejecutando en verdad estamos ciegos a lo que estamos programando.
- Las consultas simples hacen parecer que el ORM te facilita la vida. Pero apenas se pone un poco complejo dan ganas de volver a SQL.
- La verdad es que estamos agregando una capa más de complejidad al sistema!!

Conclusión: Usar un ORM aporta mucho valor, pero no podemos olvidarnos de SQL.