Les 9 – Python & MySQL

Inleiding

We kunnen nu databanken aanmaken, invullen en records opvragen met MySQL. Dit doen we rechtstreeks in de databank met phpmyadmin. (DDL, DML, DQL)

In de realiteit willen we dit echter via (zelfgeschreven) software doen. Hier komt de kennis van Python van pas.
We beperken ons in deze module tot het uitvoeren van basis MySQL via Python. Analyseren, visueel voorstellen en machine learning is dan de volgende stap. Maar dit is voor de vervolgmodules.

Python module

We moeten dus via Python verbinding maken met onze databanken. Hiervoor hebben een module nodig, een *mysql connector*. Dit is geen standaard module en zal eerst moeten worden geïnstalleerd.

Open de cmd of terminal en voeg volgende code in:

```
python -m pip install mysql-connector-python
```

Open Atom en maak een nieuw Python bestand aan: databank.py

Importeer deze module in het bestand.

```
import mysql.connector
```

Test het programma in de cmd of terminal.
Geen Errors? Dan is de module goed geïnstalleerd.

LET OP! Voor de mac-gebruikers => python3 !

Connectie maken

We kunnen de gegevens om een connectie te maken opslaan in een variabele, een object aanmaken. Zo hoeven we niet elke keer als we een MySQL commando willen uitvoeren alle informatie uittypen.

```
import mysql.connector

mijnDB = mysql.connector.connect(
    host="localhost",
    user="yourusername",
    password="yourpassword"
)
```

host => waar zijn de databanken terug te vinden? 'localhost' voor lokale databanken.
User => de gebruikersnaam van de databanken. Standaard = 'root'
password => paswoord van de databanken. Standaard = '' (leeg)

LET OP! Dit zijn de standaardinstellingen en hebben we normaal nergens aangepast. Moest je dit wel gedaan hebben, moet je natuurlijk de aangepaste gegevens invullen.

Om nu specifieke acties uit te voeren heeft de module een class met voorgeprogrammeerde methodes/functies voorzien. Dit is de ***cursor*** class. We passen die vervolgens op ons nieuw databankconnectie-object.

```
mijnCursor = mijnDB.cursor()
```

Testen! We gaan na of de connectie succesvol was en tonen alle databanken in onze localhost. Hiervoor gebruiken we de methode *execute()* van de cursor class.

```
mijnCursor.execute('SHOW DATABASES')

for databank in mijnCursor:
    print(databank)
```

Al de databanken op de localhost worden getoond.

DDL met Python

Eenmaal we een connectie hebben met de localhost kunnen SQL commando's uitvoeren zoals het aanmaken of verwijderen van een databank.

```
mijnCursor.execute("CREATE DATABASE db_naam")

of

mijnCursor.execute("DROP DATABASE db_naam")
```

Als we een databank hebben aangemaakt moeten we deze specifiek aanspreken bij de connectie:

```
mijnDB = mysql.connector.connect(
    host = "localhost",
    user = "root",
    password = "",
    database = "databanknaam"
)
```

Dan kunnen we in deze databank tabellen aanmaken of verwijderen:

```
mijnCursor.execute("CREATE TABLE t_naam(PKid INT NOT
NULL PRIMARY KEY AUTO_INCREMENT")#minstens 1 kolom
instellen

of

mijnCursor.execute("DROP TABLE t_naam")
```

Of extra kolommen toevoegen:

```
mycursor.execute("ALTER TABLE t_naam ADD COLUMN
d_naam data_type&constraints")
```

OEFN

Maak een nieuw python bestand aan: *inventarisDB*.
Maak connectie met de localhost.

Maak een nieuwe databank met de naam *inventaris* aan.
Voer het programma uit.

Pas je py bestand aan om een tabel *t_inventaris* aan te maken.
Geef alleen de PK kolom mee.
Verwijder de oude code niet, maar zet ze in commentaar.

Voeg aan de tabel de kolommen *d_product, d_prijs en d_btw* toe.
Vergeet de juiste datatypes niet.

DML met Python

Nu de databank, tabellen en kolommen aangemaakt zijn kunnen we deze ook vullen met data.

Hierbij gaan we gebruik maken van een aantal tussen variabelen. Om de SQL en de waarden op te splitsen. Nadien voegen we die dan toe aan de databank. Dit doen we met de methode *commit()*.

```
sql = "INSERT INTO t_naam (d_naam1, d_naam2) VALUES
(%s, %s)"
values = ("waarde1, waarde2")

mijnCursor.execute(sql,values)

mijnDB.commit() #databank updaten
```

Meerdere waardes ingeven, alles in een lijst zetten:

```
values = [("waarde1, waarde2"),
          ("waarde1, waarde2"),
          ("waarde1, waarde2")]
```

Data verwijderen:

```
sql = "DELETE FROM t_naam WHERE d_naam = 'waarde'"
mijnCursor.execute(sql)

mijnDB.commit()
```

Data wijzigen:

```
sql = "UPDATE t_naam SET d_naam = 'nieuwe_waarde'
WHERE d_naam = 'oude_waarde'"

mijnCursor.execute(sql)

mijnDB.commit()
```

OEFN

Voeg in de tabel inventaris van de vorige oefening 5 rijen met verschillende waardes in.

Gebruik simpele namen zoals product1, product2.
Kies voor de btw 6,12of 21 en wissel af.

Verander de BTW van product2 naar 6% btw.

Verander de prijs van product3.

DQL met Python

Als we dan een databank met tabellen en daar nog eens waarden in hebben kunnen we deze data uitaard ook opvragen. Dit doen we met het commando *fetchall() of fetchone()*.
Meestal in combinatie met een for-loop om dan alle opgevraagde records 1 voor 1 te tonen.

```
mijnCursor.execute("SELECT * FROM t_naam")

mijnResultaat = mijnCursor.fetchall()

for record in mijnResultaat:
    print(record)

of

mijnCursor.execute("SELECT d_naam1, d_naam2 FROM
t_naam")

mijnResultaat = mijnCursor.fetchall()

for record in mijnResultaat:
    print(record)
```

OEFN

Vraag alle waarden uit de tabel t_inventaris op.
Toon deze in de cmd of terminal.

Toon vervolgens alleen de productnamen.

Conclusie

De waarden worden in tuples die in een lijst zitten verzameld. Vanaf hier kunnen we deze data weer gebruiken zoals we in de cursus start to program hebben gezien.

Dit ook in combinatie met commando's zoals WHERE, LIMIT, JOIN, ORDER BY,...

Als de sql-commando's te uitgebreid worden gebruik dan tussen variabelen. Of sla input van gebruikers op in variabelen om ze dan nadien op te slaan in een databank.

OEFN

> Probeer de andere oefeningen van databanken uit te voeren via Python.
>
> Maak eventueel verschillende scripts, maar onderzoek het commando: IF NOT EXISTS of IF EXISTS op om alles samen te voegen in 1 script.