

Accès à une Base de Données depuis Python

Erick STATTNER

Maitre de Conférences en Informatique

Université des Antilles

erick.stattner@univ-ag.fr

www.erickstattner.com

Description de l'enseignement

Objectifs pédagogiques

- Accéder à une base de données depuis un programme Python
- Réaliser les principales opérations CRUD
- Traiter les données récupérées à l'aide de requêtes sur la BD
- Se familiariser aux problèmes de sécurité soulevés

Sommaire

1. Introduction
2. Accès à MySQL depuis un programme Python
3. Injection SQL

Introduction



Introduction

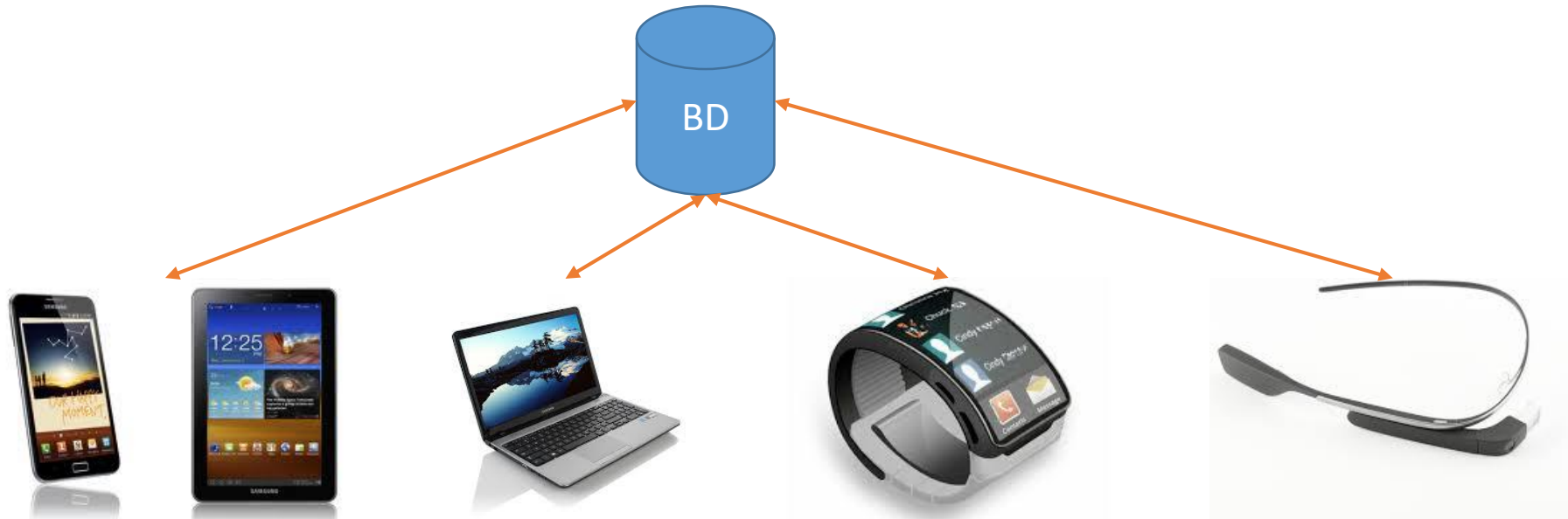
Vers une escalade des périphériques



Introduction

Contexte

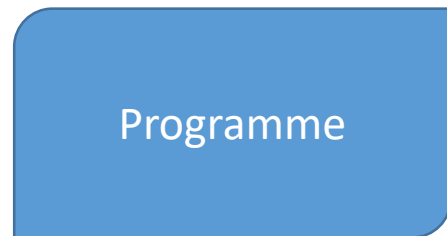
- Les données doivent être
 - Accessibles pour soi
 - Parfois, rendues accessibles aux autres



Introduction

Objectif:

- Accès depuis un programme en Python
- Problèmes soulevés



?



Différents types de BD

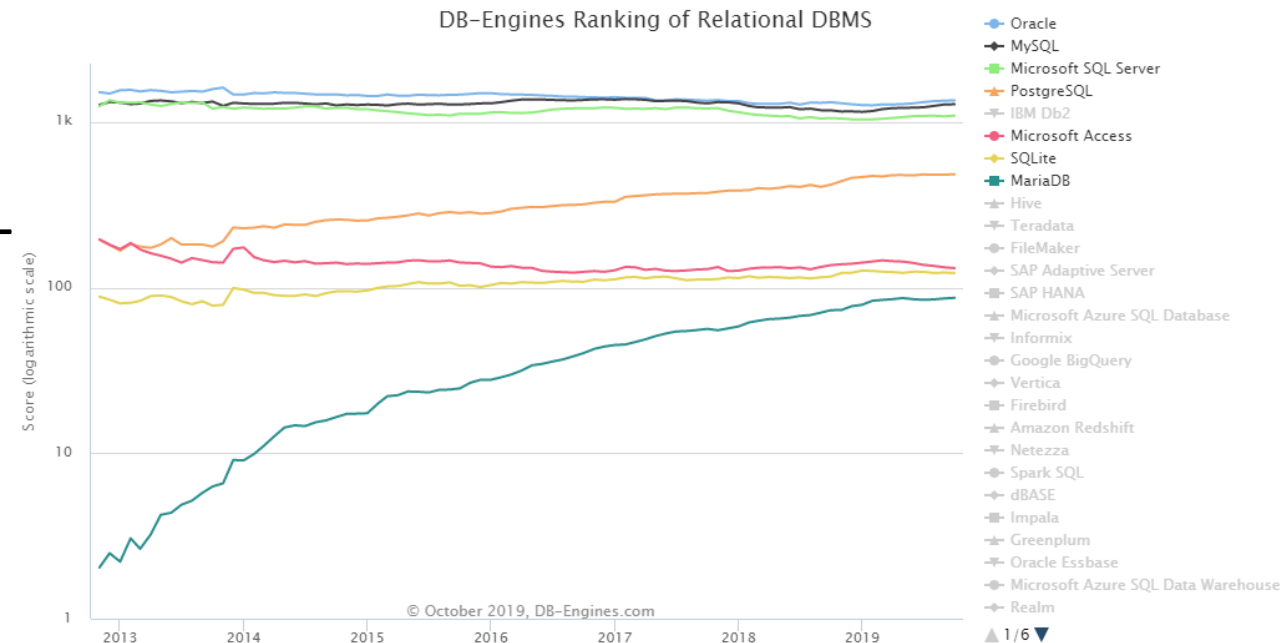
Base de données relationnelle (BDR)

- Type de BD dans laquelle les données sont liées
- Les données sont sous forme de tables liées entre elles
- Interagir avec un unique langage

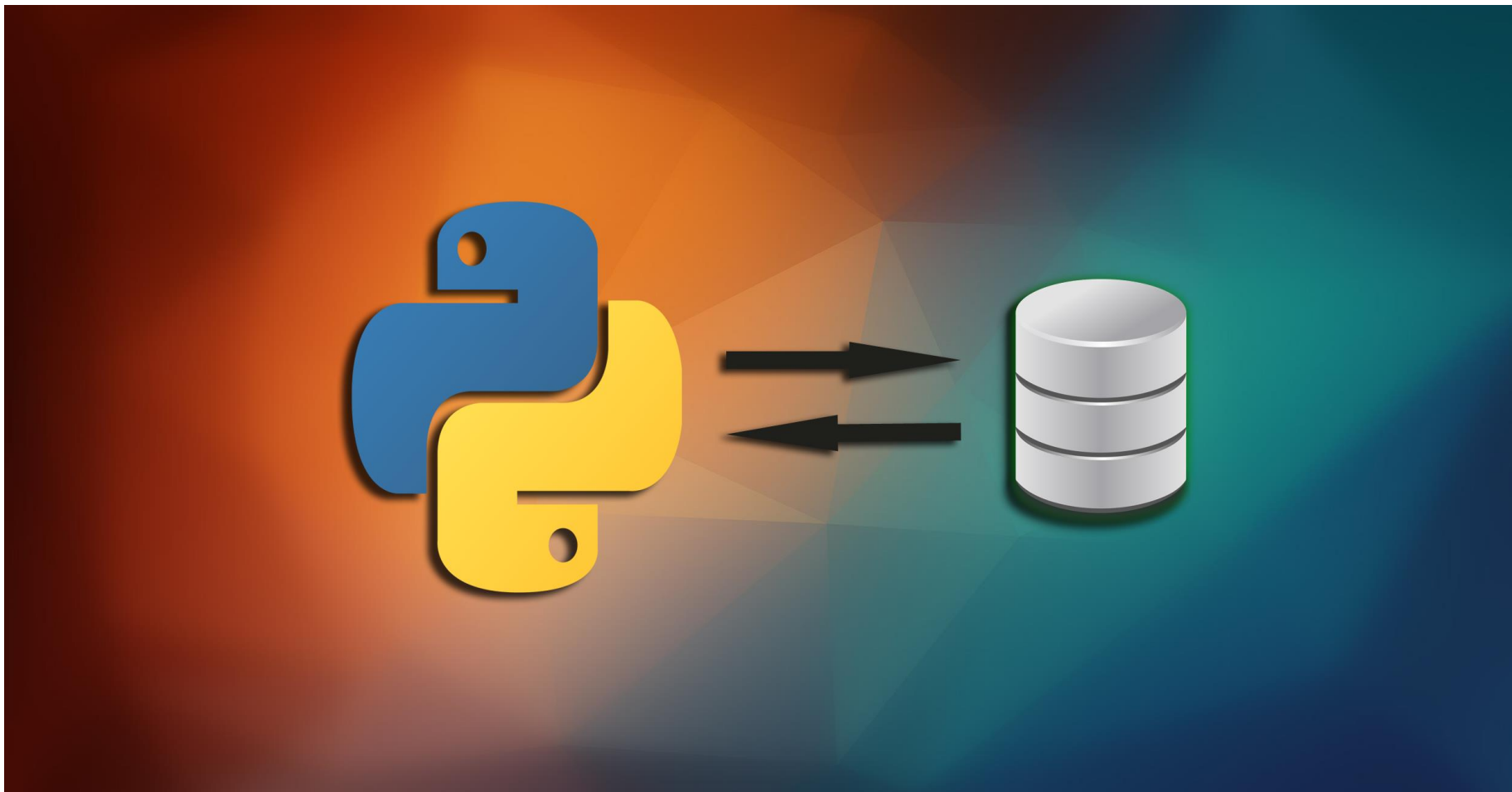
SQL (4^e génération)

Nombreuses BDR

- Oracle, MS SQL Server, PostgreSQL
- MS Access, Libre base, SQL Lite
- MARIA DB, **MySQL**
- etc.



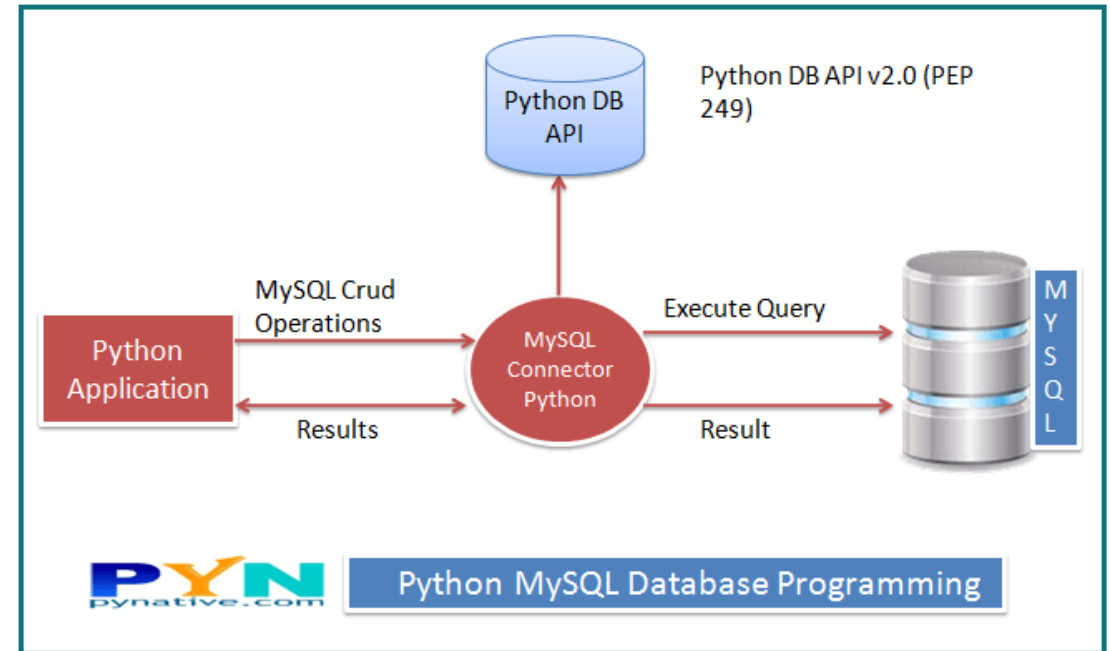
Accès à MySQL depuis un programme Python



Accès à MySQL depuis un programme Python

Depuis un programme Python

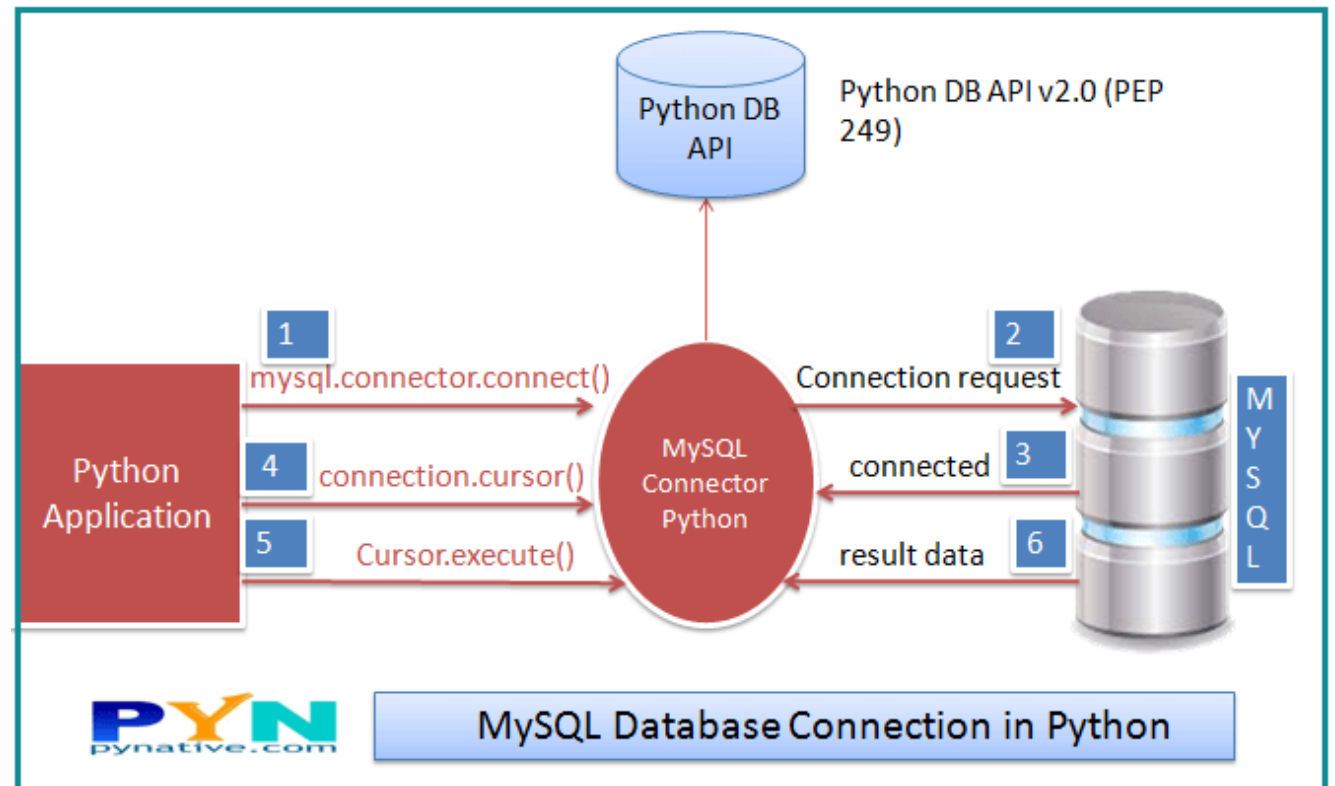
- Python fournit une API avec un ensemble de fonctions pour interagir avec MySQL
- **MySQL Connector Python**
 - API Officielle écrite en Python
 - Permet de réaliser toutes les opérations CRUD
 - Compatible avec Python 3



Accès à MySQL depuis un programme Python

5 étapes pour accéder à la BD en Python

0. Installer MySQL Connector Python
1. Etablir la connexion au serveur
2. Créer un curseur
3. Exécuter la requête
4. Traiter les résultats
5. Fermer la connexion



Accès à MySQL depuis un programme Python

Etape 0) Installer l'API MySQL Connector Python

- Choisir la bonne version de l'API à installer

Connector/Python Version	MySQL Server Versions	Supported Python Versions
8.0	8.0, 5.7, 5.6, 5.5	3.6, 3.5, 3.4, 2.7
2.2	5.7, 5.6, 5.5	3.5, 3.4, 2.7
2.1	5.7, 5.6, 5.5	3.5, 3.4, 2.7, 2.6
2.0	5.7, 5.6, 5.5	3.5, 3.4, 2.7, 2.6
1.2	5.7, 5.6, 5.5 (5.1, 5.0, 4.1)	3.4, 3.3, 3.2, 3.1, 2.7, 2.6

- **Installer l'API**

- via l'installer
<https://dev.mysql.com/downloads/connector/python/>
ou
- via **pip**
`pip install mysql-connector-python==8.0.18`

Accès à MySQL depuis un programme Python

Etape 1) Etablir la connexion au serveur

- Avant de communiquer avec la BD
- Nécessité de s'authentifier auprès du SGBD
- Fonction: `mysql.connector.connect()`

Parameter	Description
<i>host</i>	The server name or IP address on which MySQL is running
<i>database</i>	Database name to which you want to connect
<i>user</i>	Username that you use to work with MySQL Server
<i>password</i>	Password of the user

- Renvoie un objet représentant la connexion au serveur MySQL

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
```

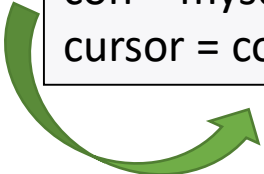
Accès à MySQL depuis un programme Python

Etape 2) Créer un curseur

- Avant d'exécuter une requête, créer un objet *curseur*
- L'objet curseur permet ensuite d'exécuter des requêtes
- Fonction:
`connection.cursor()`
- Obtenu avec l'objet *connection* renvoyé lors de la connexion à la base
- Pas de paramètre

```
import mysql.connector
```

```
con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')  
cursor = con.cursor()
```



Accès à MySQL depuis un programme Python

Etape 3) Exécuter une requête

- Une fois le curseur créé, il peut être utilisé pour exécuter une requête
- Fonction

`cursor.execute()`

Parameter	Description
<i>query</i>	Specifies the query string

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "SELECT * FROM etudiants"
cursor.execute(req)
```

Accès à MySQL depuis un programme Python

Etape 3) Exécuter une requête

- Par défaut, Python désactive l'*autocommit*
- Si requête de type *[UPDATE, DELETE, INSERT INTO]*
- Nécessité de demander explicitement la validation

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "INSERT INTO etudiants(nom, prenom) VALUES('toto', 'alain')"
cursor.execute(req)

con.commit()
```


Accès à MySQL depuis un programme Python

Etape 3) Exécuter une requête

- Possibilité de connaître le nombre de lignes
 - Renvoyées par *SELECT*
 - Affectées par *INSERT*, *UPDATE* ou *DELETE*

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "DELETE FROM etudiants WHERE note < 10"
cursor.execute(req)

nb = cursor.rowcount
Print(nb + "etudiants ont été supprimés")

con.commit()
```

Accès à MySQL depuis un programme Python

Etape 4) Traiter les résultats

- Lors d'une requête de type SELECT, les résultats sont renvoyés dans un *resultSet*
- Possibilité de le transformer en liste avec *fetchall()*

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "SELECT * FROM etudiants"
cursor.execute(req)

tableau = cursor.fetchall()
for ligne in tableau:
    print(ligne[0]+" "+ligne[1])
```

Accès à MySQL depuis un programme Python

Etape 4) Traiter les résultats

- Lire directement les colonnes

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "SELECT nom, prenom FROM etudiants"
cursor.execute(req)

for (nom, prenom) in cursor:
    print(nom+" "+prenom)
```

Accès à MySQL depuis un programme Python

Etape 5) Fermer la connexion

- Il est recommandé fermer la connexion avec la BD
- Fonction `close()`

```
import mysql.connector

con = mysql.connector.connect(host='localhost', database='films', user='root', password='root')
cursor = con.cursor()

req = "SELECT nom, prenom FROM etudiants"
cursor.execute(req)

for (nom, prenom) in cursor:
    print(nom+" "+prenom)

con.close()
```

Accès à MySQL depuis un programme Python

Pour aller plus loin

- Guide officiel
<https://dev.mysql.com/doc/connector-python/en/preface.html>
- Python MySQL Tutorials
<https://pynative.com/python-mysql-select-query-to-fetch-data/>

II. Injection SQL



II. Injection SQL

Alimenter une base de données avec les informations saisies par l'utilisateur soulève un certain nombre de problèmes:

- Champs vides
- Données incohérentes / erreurs de frappe
- Utilisateurs malveillants

S'il ne sont pas traités affecte:

- La base de données
- Le programme

II. Injection SQL

Champs vides

- Forcer l'utilisateur à saisir les données attendus
Ex. Le champ nom est obligatoire pour continuer

Données incohérentes / erreurs de frappe

- Vérifier les données avant l'enregistrement en base
Ex. age > 0 et adrMail est correcte

II. Injection SQL

Injection SQL

- Type d'attaque qui cible les applications interagissant avec une BD
- Vise à insérer du code SQL lors des interactions avec la BD dans le but de modifier le comportement des requêtes
 - Enchaîner plusieurs requêtes
 - Ignorer une partie de la requête
 - Modifier son comportement
- Type d'attaque le plus répandu et facile à mettre en œuvre

II. Injection SQL

Conséquence

- Contournement formulaire d'authentification
- Vol d'informations dans la base ou Dump de la totalité de la BD
- Compromettre l'intégrité de la base
- Exécution de code malveillant
- Planter l'application
- Modifier l'affichage

II. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

login = input("Login: ")

mdp = input("Password: ")

req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

II. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

Login:	Root'; --
Mdp:	Lol je t'ai eu

- Que devient la requête précédente

```
login    = input("Login: ")
mdp      = input("Password: ")
req      = "SELECT id, login, mdp FROM membre
           WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";
```

II. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

Login:	Root'; --
Mdp:	Lol je t'ai eu

- Que devient la requête précédente

login = input("Login: ")

mdp = input("Password: ")

req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

- **Probleme:**

L'utilisateur root n'existe peut etre pas !

IV. Injection SQL

Exemple 2: Contourner authentication

- Que devient la requête

login = input("Login: ")

mdp = input("Password: ")

String req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

- Si l'utilisateur saisit:

Login: ' OR 1=1; --

Mdp: Je t'ai encore eu !

Login: toto

Mdp: ' or 1=1; --

Login: ' OR 1=1

Mdp: ' OR 1=1; --

II. Injection SQL



II. Injection SQL

Solution générale:

- Vérifier le format des données saisies et notamment la présence de caractères spéciaux
- Limiter la taille des champs
- Ne pas afficher dans les messages d'erreurs
 - Une partie de la requête
 - Des informations sur la structure des bases de données
- Ne pas conserver les utilisateurs par défaut
- Restreindre au minimum les privilèges des comptes utilisés
- Ne pas stocker directement les mot de passe dans la base, mais un hash