SHAKE THE FUTURE

CENTRALE NANTES

Bases de Données

Accéder aux bases de données - API

JY Martin

Les différents mécanismes d'accès

- Outils d'administration
- Outils de gestion : ETL, EAI, ESB, ...
- Outils de BI
- **API de connexion**

Plan

La problématique

- Les outils (ETL, BI, ...) ne suffisent pas toujours
- Ces outils, il faut bien les programmer...

=> nécessité de disposer d'un mécanisme, au niveau programmation, pour accéder aux bases de données et d'échange avec elles : API

La problématique

Au niveau de l'API, il faut :

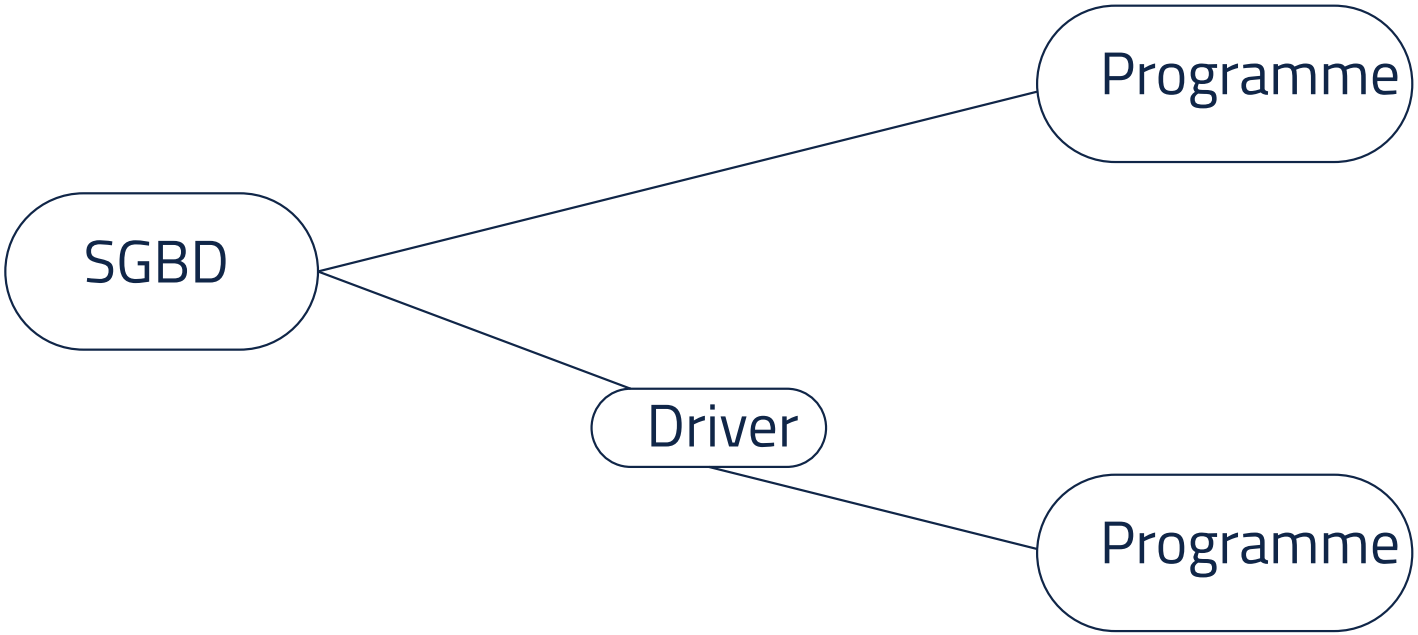
- se connecter au SGBD
- Accéder à la base de données (gestion des droits)
- Sélectionner des informations (gestion des droits)
- Modifier des informations (gestion des droits)

=> S'appuyer sur SQL

2 façon de travailler

- Utilisation directe de l'API
  Ensemble de fonctions et méthodes d'échange avec le SGBD
- Utilisation d'ORM **plus de sécurité**
  Object Relational Mapping
  Utilisation d'un framework qui masque les accès au SGBD

2 façon de procéder



Quelques conseils

- Ne faites pas une connexion / déconnexion à chaque requête
- On ne peut pas faire une requête sur 2 bases de données différentes en même temps
- Par contre il est possible de se connecter en même temps à plusieurs bases de données

Plan

Se connecter à partir de PHP

- API de connexion dépendant du SGBD (obsolète depuis PHP 7)
  - fonctions spécifiques de manipulmation de la base de données
  - utilisation d'un driver ODBC
    A éviter
- Classe PDO **comme Driver**
  Fortement conseillée

PHP - PDO

La communication passe par un objet de la classe **PDO**
Le type de base de données est précisé dans le **protocole de connexion**

- La création d'un objet PDO crée une connexion à la base de données
  conn = new PDO('protocole', 'login', 'password') ; niveau de droit
- Les méthodes de la classe PDO permettent de faire des requêtes et de récupérer les résultats
- la destruction de l'objet clos la connexion

D'un type de SGBD à l'autre, la seule chose qui change est le protocole de communication.

PHP - PDO

Le protocole : protocole :host=myHost ;dbname=myDatabase

- protocole est le protocole utilisé
  mysql, pgsql, oci, ...
- myHost est le nom du serveur
- myDatabase est le nom de la base de données

Exemple :

```
$conn = new PDO( "pgsql :host=localhost ;dbname=test" , $user, $pass) ;
```

PHP - PDO

Les méthodes classiques

- **prepare** définit une requête SQL à exécuter
  On peut définir des paramètres de la façon suivante :
  - ? : valeur du paramètre indiquée dans execute
  - :nomParam : valeur du paramètre indiquée dans bindParam
- **bindParam** : définition des valeurs paramètres d'une requête
- **execute** exécute une requeête (définie avec prepare). execute permet de définir les valeurs des paramètres.
- **fetchAll** récupère dans un tableau les lignes résultats
- **closeCursor** clos la requête SQL et récupère l'espace mémoire
  ...

PHP - PDO

Exemple :

```
$conn = new PDO('pgsql :host=localhost ;dbname=test', 'login', 'password') ;
$query = $conn->prepare("SELECT * FROM Personne WHERE Personne_ID= ?") ;
$query->setFetchMode ( PDO ::FETCH_ASSOC ) ;
$query->execute(array(25)) ;
$rows = $query->fetchAll() ;
foreach ( $rows as $aRow ) {
        echo $aRow["nom"]." ".$aRow["prenom"]."<br/ >\n" ;
}
$query->closeCursor() ;
```

PHP - ORM

Object Relationnal Mapping :

- Introduisent une couche d'abstraction entre la base de données et le langage de programmation
- Permet de s'abstraire du type de SGBD qui est alors géré par l'ORM

Exemples :

- Propel
- Doctrine
- Zend

PHP - Frameworks

S'appuient généralement sur un ORM et fournissent un certain nombre d'outils au dessus de l'ORM pour facimliter la programmation.

Exemples :

- Symfony (utilise doctrine ou propel)
- Codeigniter (qui a son propre ORM)
- cakePHP (qui a son propre ORM)

Plan

JDBC

Java DataBase Connectivity

LE mécanisme d'accès aux bases de données

- S'inspire du fonctionnement d'ODBC
- Driver indépendant du système d'exploitation (basé sur java)
- Nécessite l'installation au préalable du driver JDBC adéquat

framework : Maven

JDBC

API de connexion aux bases de données

1 driver pour chaque type de base de données

Les drivers sont classés en 4 types

- Type 1 : passerelles JBDC - OBDC
- Type 2 : API natif (connexion directe à la base)
- Type 3 : conversion des appels JDBC en un protocole indépendant du SGBD
- Type 4 : conversion des appels JDBC en un protocole réseau exploité par le SGBD

Il est **vivement** recommandé d'utiliser le type 4.

Mise en œuvre

L'interface java.sql gère le driver JDBC est s'occupe de la partie abstraction de l'accès à la base de données.

- Importer le package java.sql.*
- Charger le driver JDBC (1 seule fois dans le programme)
- Etablir la connexion à la base de données
- Effectuer les requêtes SQL
- Libérer la connexion
- Libérer le driver (1 seule fois dans le programme)

Charger / Libérer le driver JDBC

Charger :

```
try {
        Class.forName("myDriver.ClassName") ;
}
catch (java.lang.ClassNotFoundException e) {
        Logger.getLogger(MaClasse.class.getName()).log(Level.SEVERE, null, ex) ;
}
```

—> prendre un fichier Jar et mettre en mémoire

capteur l'erreur (provoqué par l'erreur)—> générer log erreur

Remarque : Le fichier JAR correspondant au driver JDBC doit être présent dans votre projet.

Charger / Libérer le driver JDBC

Libérer :

```
Driver theDriver = DriverManager.getDriver(dbURL) ;
DriverManager.deregisterDriver(theDriver) ;
```

Les types de driver

Dans le slide précédent, pour monter le driver JDBC en mémoire, on utilise la syntaxe myDriver.ClassName

La syntaxe à utiliser vous est habituellement fournie sur le site sur lequel vous pouvez télécharger le driver.

Concrètement :

- mySQL : com.mysql.jdbc.Driver
- PostgreSQL : org.postgresql.Driver
- Oracle : oracle.jdbc.driver.OracleDriver
- ...

Etablir une connexion / Se déconnecter

Attention : Le driver JDBC doit avoir été monté correctement pour que la connexion puisse s'établir.

On utilise, pour les échanges avec la base de données, un objet **Connection**.

Se connecter :

```
String dbURL= "jdbc :protocol :urlbase" ;
Connection connect = DriverManager.getConnection(dbURL, "myLogin",
"myPassword") ;
```

Etablir une connexion / Se déconnecter

protocol.urlbase est de la forme :

- jdbc :mysql ://<host>/<bd_name>
- jdbc :postgresql ://<host>/<bd_name>
- jdbc :oracle :oci8 :@<database>
- ...

Etablir une connexion / Se déconnecter

Se déconnecter :

```
connect.close() ;
```

Effectuer une requête SQL

Il existe plusieurs classes pour faire des requêtes SQL. Celles utilisées couramment sont :

- Statement
- PreparedStatement

Il est **tres fortement** conseillé d'éviter la première classe qui est sensible aux injections SQL et donc n'offre pas toutes les garanties de sécurité de manipulation de la base de données.

Lors de leur création, ces objets permettent également de préciser des éléments sur ce qu'il sera possible de faire sur le résultat produit (revenir en arrière, modifier le résultat, ...).

Effectuer une requête SQL

```
String query = "..." ;
PreparedStatement stmt = connect.prepareStatement(query ) ;
```

La requête peut comporter des paramètres qui seront définis par des ' ?'
Les valeurs des paramètres seront alors définies par les méthodes set...

Effectuer une requête SQL

Si la requête doit retourner un résultat, on utilise la méthode **executeQuery** sur le PreparedStatement.
Si elle ne retourne aucun résulktat (update, delete, ...) on utilise la méthode **executeUpdate**.

La récupération du résultat d'un executeQuery se fait via un objet **ResultSet**.
Les méthodes de ResultSet permettent alors de récupérer les différentes informations.

Effectuer une requête SQL - Les paramètres

Les requêtes définies pour un PreparedStatement peuvent contenir des paramètres. Chaque paramètre est défini par un ?.

Pour donner une valeur on utilisera une méthode set... (setString, setInt, setFloat, setDate, ...)
La méthode comporte 2 paramètres :

- Le premier paramètre est un entier indiquant le numéro d'apparition du paramètre dans la requête (1, 2, ...)
- le second paramètre est la valeur du paramètre.

Effectuer une requête SQL - Les paramètres

```
String query = "SELECT Personne_Id FROM Personne WHERE Personne_nom= ?" ;
PreparedStatement stmt = connect.prepareStatement( query ) ; stmt.setString(1, "ALBAN") ;
```

Effectuer une requête SQL - Les résultats

- Si la requête ne retourne aucun résultat, on utilise executeUpdate

```
PreparedStatement stmt = connect.prepareStatement( "DELETE FROM Personne
WHERE Personne_ID=1" );
stmt.executeUpdate();
```

- Si la requête retourne un résultat, on utilise executeQuery

```
PreparedStatement stmt = connect.prepareStatement( "Select Personne_Nom
FROM Personne WHERE Personne_ID=1" );
ResultSet rs = stmt.executeQuery();
```

Effectuer une requête SQL - Parcourir le résultat

ResultSet permet de parcourir les lignes du résultat.

- next() permet de passer à la ligne suivante. null si elle n'existe pas
- previous(), s'il est autorisé lors de la définition du PreparedStatement, permet de revenir à la ligne précédente.
- get... (getString, getInt, ...) permet de récupérer une colonne de la ligne. Le paramètre est soit le numéro de la colonne (ordre d'apparition dans la liste des colonnes remontées), soit par son nom.

Exemple

```
import java.sql.* ;
public class testSQL {
        public static void main(String[] argv) {
                try {
                    Class.forName("org.postgresql.Driver") ;
                    Connection connect = DriverManager.getConnection("jdbc :postgresql ://localhost/test", "prweb", "prweb") ;
                    String query = "SELECT * FROM Personne WHERE nom= ?" ;
                    PreparedStatement stmt = connect.prepareStatement(query ) ;
                    stmt.setString(1, "ALBAN") ;
                    ResultSet res = stmt.executeQuery() ;
                    while (res.next()) {
                            System.out.println("Info : "+rs.getString("nom")) ;
                    }
                    stmt.close() ;
                    connect.close() ;
                } catch(java.lang.ClassNotFoundException e) {
                    System.err.println("ClassNotFoundException : " + e.getMessage()) ;
                } catch(SQLException ex) {
                    System.err.println("SQLException : " + ex.getMessage()) ;
                }
        }
}
```

Les DataSource

Certains objects comme les **DataSource** permettent de gérer des pools de connexion.

- Comme pour les connexions, il faut indiquer le driver utilisé, et des éléments tels que login, mot de passe, ...
- La connexion est demandée et rendue au pool de connexion
- Une connexion renduée n'est pas fermée, mais mise de coté. Elle n'est fermée que si elle n'est pas utilisée pendant un certain temps.
- Lors d'une demande de connexion, on commence par vérifier qu'il n'existe pas de connexion en attente, ce qui permet d'éviter de repasser des connexions.

Les ORMs

Comme dans la plupart des langages, il existe des bibliothèques de fonctions qui permettent d'assusrer la connexion aux bases de données et le mapping entre objets et tables.

- Hibernate
- JPA

Ces outils reposent généralement sur des fichiers XML de configuration et des classes JAVA correspondant aux objets manipulés.

Les Frameworks

¨Pour aller plus loin, vous pouvez également vous appuyer sur des Frameworks qui vont masquer , au moins partiellement,l'utilisation d'une base de données

- Spring
- Springboot
- JSF
- Struts, Tapestry
- Play !

Plan

Se connecter en Python

- API spécifiques
- Des bibliothèques de fonctions pour de nombreux types de base de données
- Nécessite l'importation et l'installation de modules correspondant au type de base de données utilisée

Se connecter en Python : les étapes

Comme la plupart des langages, la connexion à, et l'exploitation d'une base de donénes se fait en plusieurs étapes :

- Importer le module correspondant au SGBD
- Ouvrir une connexion à la base
- Créer un curseur pour une requête SQL
- Exécuter une requête SQL via le curseur
- Récupérer les résultats
- Clore le curseur
- Clore la connexion

Les APIs de connexion

L'API de connexion dépend de la base de données. Il faut :

- Que le module ait été importé (pip install)
  python -m pip install ...
  - mysql : mysql-connector
  - postgresql : postgres
  - oracle : ... ben non, il faut le télécharger depuis le site d'Oracle et l'installer à la main avec les variables d'environnement.
- Que le module soit déclaré en début de script
  import ...
  - mysql : mysql.connector
  - postgresql : psycopg2
  - oracle : cx_oracle

Se connecter en Python : la connexion

La connexion se fait à partir de la fonction **connect**.

4 éléments à indiquer dans le paramètre :

- host : le serveur
- dbname : le nom de la base de données
- user : le login de connexion
- password : son mot de passe

Toute connexion ouverte doit être fermée via **close**

Se connecter en Python : Les curseurs

Pour effectuer une requête on passe par la définition d'un curseur.

- Un curseur est défini à partir d'une connexion à une base de données
- Il s'appuie sur une requête SQL
- Il permet d'exécuter une requête, et d'en récupérer le résultat
- Il permet enfin de récupérer le résultat soit en totalité, soit ligne par ligne.
  - fetchone : la ligne suivante
  - fetchall : toutes les lignes

Tout curseur ouvert doit être fermé (via close) avant d'être à nouveau utilisé.

Se connecter en Python : Exemple 1

```
import psycopg2
conn = psycopg2.connect("host=localhost dbname=test user=prweb password=prweb")
cursor = conn.cursor()
cursor.execute("select * from Personne")
rows = cursor.fetchall()
for row in rows :
        print(row)
cursor.close()
conn.close()
```

Se connecter en Python : Exemple 2

On peut aussi utiliser le curseur dans les boucles

```
import psycopg2
conn = psycopg2.connect("host=localhost dbname=test user=prweb password=prweb")
cursor = conn.cursor()
cursor.execute("select * from Personne")
for row in cursor :
        print(row)
cursor.close()
conn.close()
```

Se connecter en Python : Injections SQL

Pour éviter les injections SQL, on utilise des dictionnaires en corrélation avec les curseurs

```
cursor = comm.cursor()
data = {"nom" : "DUBOIS", "prenom" : "Jacques"}
cursor.execute("INSERT INTO personne(nom, prenom) VALUES (:nom, :prenom)",
data)
comm.commit()
```

Plan

API spécifiques

Tres majoritairement, les connexions à des bases de données en C / C++ se font via des API spécifiques.

- mysql : Connector/C++ (https ://dev.mysql.com/doc/connector-cpp/8.0/en/connector-cpp-introduction.html)
- postgreSQL : libpqxx (http ://pqxx.org/)
- Oracle : OTL (http ://otl.sourceforge.net/)

ODBC - La solution Microsoft

Une autre solution, fonctionnant uniquement sous Windows, est de basser par un driver ODBC.

- Le driver ODBC est défini dans le système.
- La connexion se fait depuis une bibliothèque de fonction, qui se connecte au driver ODBC

Plan

Conclusion

Dans la plupart des langages, il existe un moyen de se connecter aux bases de données

Quelques suggestions :

- S'il existe des frameworks, appuyez vous sur eux
- S'il existe des ORMs également,
- Privilégiez une connexion via un driver plutôt qu'une API spécifique lorsque c'est possible
  Pensez à fermer vos connexions, et à libérer les drivers lorsque c'est possible.


Success
NEXT EXIT