

Créer une application avec un SGBD

Nicolas Travers

Equipe Vertigo - Laboratoire CEDRIC
Conservatoire National des Arts & Métiers, Paris, France

Développement d'application BD

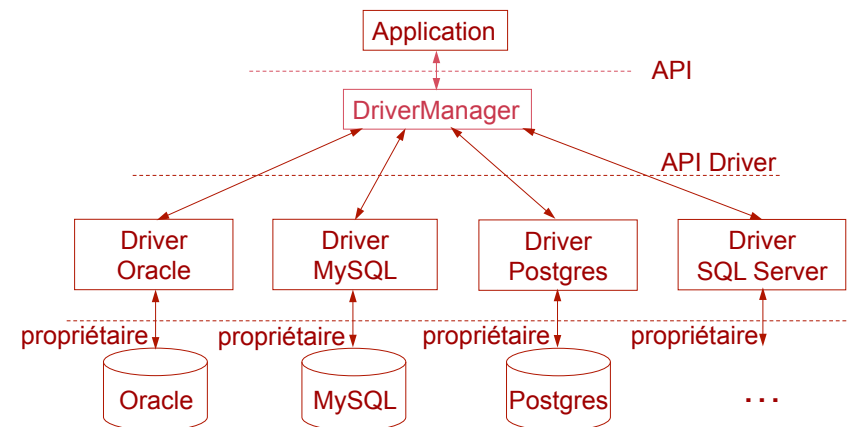
- Besoins :
 - Connexion à la base de données
 - Gérer les différents SGBD
 - Exécution de requêtes / procédures
 - Traduire les particularités des SGBD (non standards)
 - Optimiser le transfert de données
 - A la demande
 - Gérer les types de données

Contenu du cours

- Application BD
 - Architecture
 - Pilotes / Drivers
 - En pratique sous Python avec ODBC

(D'après les supports de Michel Crucianu, Cédric du Mouza et Philippe Rigaux)

Architecture



API – Interface Appli / Driver

- Objectif : interface uniforme assurant l'indépendance du SGBDR cible
- Réalité : indépendance **relative** du SGBDR, l'interface étant assurée par un pilote (*driver*) fourni par l'éditeur du SGBDR ou par un tiers...

Modèles d'interaction

- **Modèle 2-tiers** : interaction directe entre le client (application) et le SGBDR
 - Avantage : facilité de mise en œuvre
 - Désavantages : tout le traitement est du côté du client, dépendance forte entre le client et le SGBDR
- **Modèle 3-tiers** : interaction par l'intermédiaire d'un serveur *middleware*
 - Avantages : une partie du traitement peut être transférée au serveur middleware, flexibilité pour l'interaction avec le client, le client peut être totalement indépendant du SGBDR
 - Désavantages : difficulté de mise en œuvre, exige des compétences plus vastes

API – Interface Driver / SGBD

- Pilote (ou Driver)
 - Intègre les spécificités du SGBD (langage, informations, métadonnées)
 - Gère la connexion
 - Gère les transferts de données
 - Transferts des n-uplets sur demande (curseurs)
 - Respect des spécifications
 - Langage de programmation
 - SGBD

Différents Pilotes

- **Java** : JDBC - *Java Database Connectivity*, JDO (objets)
- **Python** : PEP 249
- **C++** : SQLAPI
- **C#** : ADO
- **PHP** : PDO, ADOdb
- **API ODBC** : Open Database Connectivity
 - API standardisée pour toute base de données
 - ++ Compatibilité, Portabilité, tout langage
 - -- Performance, Pas de spécificités (SGBD)

Structures communes

- Dans la plupart des Drivers se trouve :
 - **Statement** : requêtes simples
 - **PreparedStatement** : Instructions SQL paramétrées
 - **CallableStatement** : Procédures et fonctions
 - **Cursor/ResultSet** : Résultat d'une requête

PEP 249

- “Python Enhancement Proposal”
 - Connexion à la base de données
 - Spécification => pas de documentation (tutoriels)
 - Module python à intégrer au programme :
 - **ceODBC** ou pyodbc
 - Appelé “package”
- NB: pour MySQL
 - ODBC installé, mais driver MySQL non intégré (à télécharger)

Curseurs / ResultSet

- Résultat d'une requête
 - Pas de données lors de l'exécution (présente sur le serveur)
 - Fournie à la demande : next()
 - Fermé & Remplacé automatique si
 - Nouvelle requête
 - Sur le même statement
 - Les données du curseur peuvent être modifiable
 - Update, Delete, Insert

Programmation sous Python (1/2)

1. Ouvrir connexion (BD)
 - **maConnection = connect()**
 - Objet “Connection”
2. Création d'un curseur
 - **Curs = maConnection.cursor()**
 - Curseur pour exécuter les requêtes (peut être unique pour toutes les requêtes)
3. Exécution d'une requête
 - **Curs.execute(“SELECT idEtudiant FROM Etudiant”);**
 - **Curs.execute(“SELECT idEtudiant FROM Etudiant where Nom=?”, input);**
 - !!! Pas de résultats au “execute”

Programmation sous Python (2/2)

4. Lire le résultat

- N-uplet par n-uplet : `Curs.fetchone()`
- Liste de n-uplets : `Curs.fetchall()`
- Itérateur : `For result in curs`

5. Fermer le curseur

- `Curs.close();`
- Transactions :
 - Début au premier accès (requête)
 - Mises à jour possible (UPDATE, INSERT, DELETE);
 - Validation : `maConnection.commit();`
 - Annulation : `maConnection.rollback();`