lava et les Bases de Données

Introduction

Parler de bases de données ou de SGBD, sous entend généralement le langage de requêtes SQL. Ce langage joue un rôle important dans la création, la manipulation et la gestion des bases de données relationnelles. SQL étant un langage spécifique à des applications manipulant des bases de données, ses commandes sont très expressives et peuvent donc invoquer des actions de très haut niveau tel que le tri d'une base de donnée.

Bien qu'il soit simple, SQL fut standardisé en 1992, et ceci pour permettre aux développeurs d'écrire une seule application, à travers laquelle ils pourront communiquer avec la plupart des systèmes de bases de données sans changer leurs commandes SQL. Mais cela n'était pas suffisant parce que l'interrogation d'une base de données nécessite au préalable la connexion à cette dernière et malheureusement chaque SGBD possède sa propre interface permettant d'établir cette connexion, donc le problème de standardisation n'a pas été complètement résolu. Pour apporter une solution à cette restriction ou autrement dit, pour standardiser la manière de se connecter à une base de données, chaque constructeur d'un SGBD a développé une passerelle entre son système et les applications qui y accèdent, il s'agit de l'API ODBC, écrite en langage C et chargée d'effectuer toutes les opérations nécessaires pour établir la connexion à une base de données sans que le développeur sache comment cela se fait. Et donc, grâce à ODBC et SQL, on pouvait se connecter et manipuler une base de données d'une manière standard avec la plupart des SGBD.

D'autre part, SQL est incomplet et exige de recourir à son intégration dans un langage de programmation de haut niveau tel que C et C++, et d'ailleurs de nombreux utilisateurs préféraient développer leurs interfaces indépendamment des moteurs des SGBD, car c'est sans doute la façon la plus rapide d'implémenter un accès étendu sur toutes les variétés de moteurs. Malgré cela et malheureusement, on ne pouvait écrire un programme qui puisse s'exécuter sur toutes les plates-formes, pour la simple raison que ces langages ne sont pas portables. Java apporte une solution à ce problème, étant donné qu'un programme Java peut facilement s'exécuter sur n'importe quelle plate-forme.

Java est donc un langage excellent pour des applications de base de données, il a suffit seulement qu'il intégre un moyen pour qu'il puisse communiquer avec une variété de bases de données différentes, c'est le mécanisme assuré par l'API JDBC.

Pourquoi JDBC ?

Au début de 1995, la première version de Java Development Kit (JDK) n'avait intégré aucun support pour accéder aux bases de données. D'autre part, les fonctionnalités de Java permettaient facilement de construire des outils pour l'accès aux bases de données mais aucune prescription n'a été indiquée sur la façon dont vont fonctionner de tels drivers. Plusieurs constructeurs ont déjà développé des outils pour l'accès aux bases de données mais aucun guide général sur l'écriture de ces outils n'a été spécifié.

En présence de ce chaos, JavaSoft a construit la première version de Java Databasc Connectivitv (JDBC) en mai 1996. JDBC est une API Java de bas niveau, représentant pour les vendeurs un cadre neutre et universel qui va leur servir pour construire leurs propres drivers pour l'accès aux différentes bases de données.

L'API JDBC n'est donc qu'une collection d'interfaces et de classes abstraites que chaque vendeur doit implémenter pour développer son propre JDBC-driver.

Sun a donc construit JDBC en partant du travail du groupe X/OPEN qui a également servi de base à l'interface ODBC de Microsoft. Etant donné que plusieurs réseaux de PC utilisent ODBC pour leurs applications clientes en C ou C++, JavaSoft et Intersolv ont construit ensemble le driver JDBC-ODBC bridge, une couche ou une abstraction offrant un accès aux bases de données relationnelles.

Java étant un langage de choix pour les applications d'Internet et d'Intranet, le nombre de choix des drivers JDBC ne cesse d'augmenter.

Les caractéristiques de JDBC

L'API ODBC de Microsoft est largement utilisée pour accéder aux bases de données relationnelles. Elle offre la possibilité de se connecter à la plupart des bases de données sur la plupart des plates-formes. La question qu'on se pose est que : pourquoi ne pas juste utiliser ODBC à partir de Java? La solution est que cela est possible mais avec l'aide de JDBC sous forme de JDBC-ODBC bridge. La question devient alors "pourquoi le besoin de JDBC"? On peut répondre à cette question par plusieurs réponses :

#ODBC n'est pas approprié pour être utilisé directement à partir de Java, car il est écrit en C, et les appels du code natif C à partir de Java pose des problèmes de sécurité, de robustesse et de probabilité automatique des applications.

• ODBC est difficile à apprendre, il combine des caractéristiques aussi simples qui avancées et possède des options trop complexes même s'il s'agit de simples requêtes.

• Une API Java tel que JDBC est nécessaire dans le but d'assurer une solution "pure java".

Les fonctionnalités de JDBC

JDBC utilise une simple hiérarchie de classes pour les objets de base de données. Ces classes sont contenues dans le package java.sql.* à partir de la version 1.1 de JDK. JDBC n'est en réalité qu'une spécification c'est à dire que ses classes ne sont que des descriptions de classes et de méthodes qui doivent être surdéfinies pour produire un driver JDBC.

Une session d'une base de données, à partir de la connexion jusqu'à la disconnexion, se fait de la manière suivante :

Etablir une connexion

Il existe trois classes nécessaires pour établir une connexion à une base de données.

• Java.sql.DriverManager : pouvant supporter des drivers multiples pour se connecter à différentes bases de données. Sa fonction principale est de charger le driver le plus approprié à une source de données.

• Java.sql.Connection : représente l'objet Connection.

• Java.sql.DatabaseMctaData : retourne des informations sur la connexion et des informations nécessaires sur la base de données.

Exécuter des commandes SQL

Après connexion à une base de données, le client exécutera un ensemble de requêtes SQL (sélection, insertion, update ou delete). Deux classes sont nécessaires pour exécuter ces commandes SQL sur une base de données.

•Java.sql.Statment : Compose et exécute un ensemble de requêtes SQL.

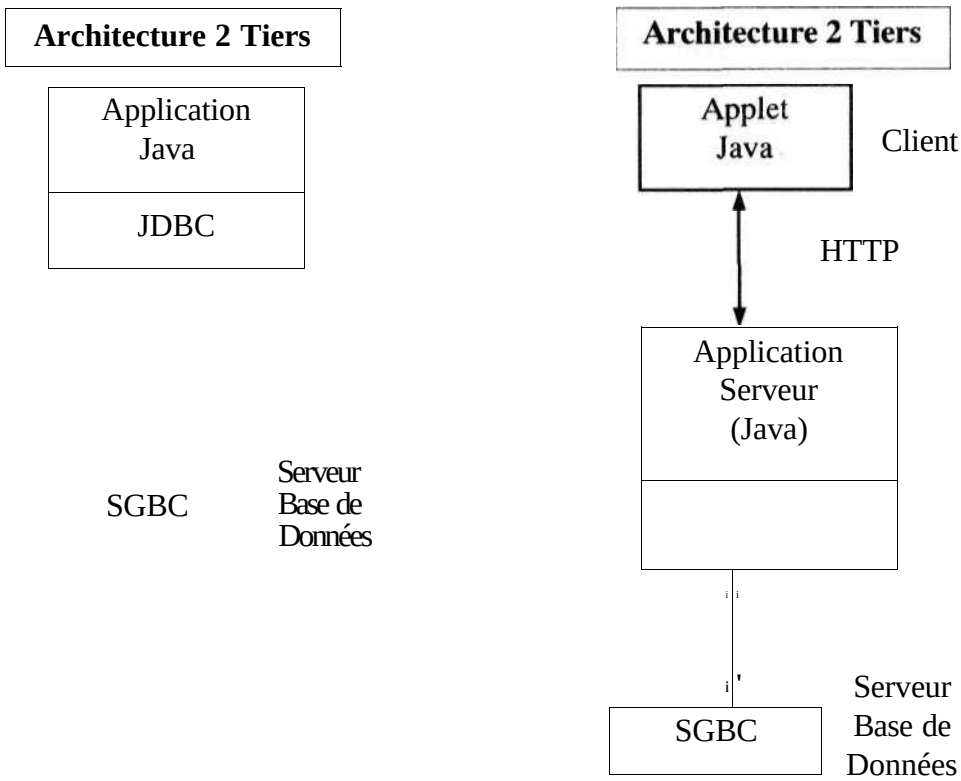
• Java.sql.ResultSet : objet contenant les résultats des requêtes.

Disconnexion et traitement des résultats

Architectures de JDBC

JDBC supporte deux architectures pour l'accès aux bases de données.

1. Architecture 2 tiers
2. Architecture 3 tiers

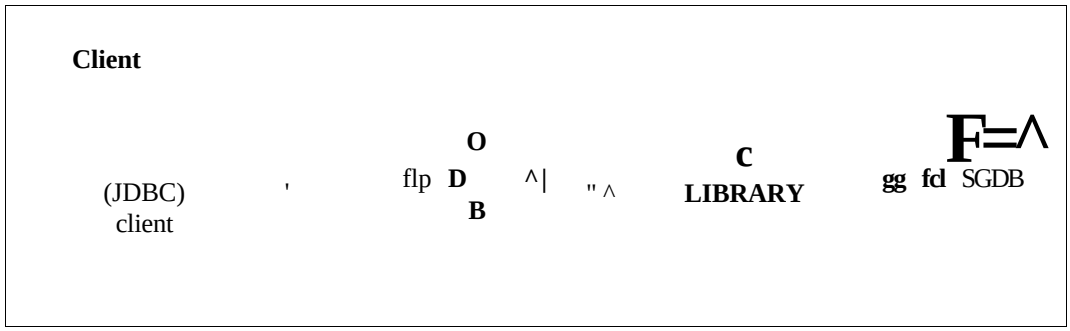


Comment JDBC a été implémenté par les constructeurs ?

Les implémentations de l'architecture 2 tiers

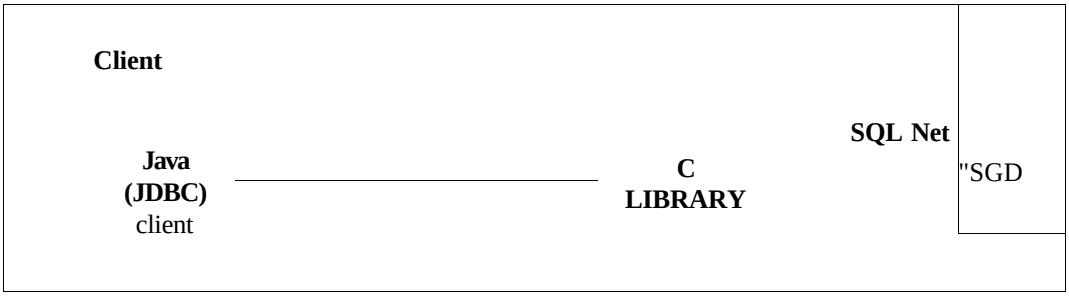
1. Driver JDBC-ODBC bridge et ODBC

Actuellement, la plupart des implémentations de l'API JDBC qui sont disponibles reposent sur l'API ODBC et le driver JDBC-ODBC bridge. Dans **ce** cas, ODBC agit comme une couche intermédiaire entre le driver JDBC et les librairies du constructeur du SGBD. L'avantage de cette approche est ressenti lorsqu'il s'agit d'une application nécessitant l'accès à une base de données pour laquelle il n'existe pas un driver JDBC (pour le moment).



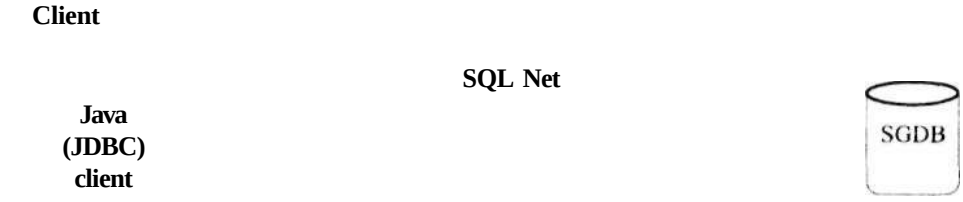
2. Drivers JDBC utilisant les librairies du SGBD

Dans cette architecture, l'accès à une base de données se fait par l'intermédiaire de librairies fournies avec le SGBD. Ces API natives sont appelées des drivers-partiellement-java, elles sont généralement écrites en C ou en C++. L'implémentation de JDBC doit utiliser une couche de C ou de C++ pour pouvoir invoquer les routines des librairies. Par conséquent, chaque client doit avoir une copie locale de librairie .DLL contenant la couche C/C++ permettant de communiquer avec ces librairies. Ces drivers JDBC utilisant des méthodes natives ne peuvent pas pour le moment être utilisés dans le cas des applets pour des raisons de sécurité.



3. Drivers Java-only

Les drivers de cette catégorie sont appelés des drivers-pures-java qui ne font aucun appel aux librairies de SGBD.

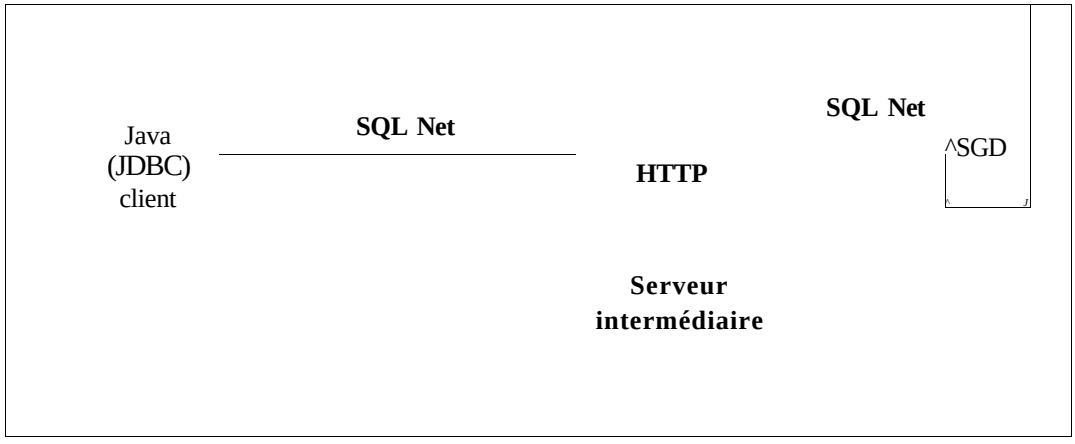


Les implémentations de l'architecture 3 tiers

l. Accès JDBC via un serveur HTTP intermédiaire

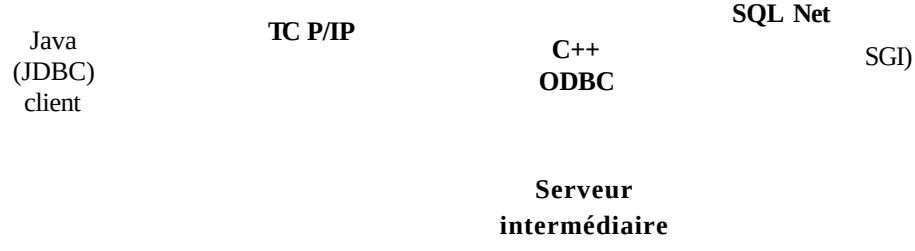
L'accès à une base de données se fait par l'intermédiaire d'un serveur HTTP acceptant des requêtes au port 80 pour les diriger vers le SGBD.
Avantage : facile à implémenter car il ne suffit d'écrire que le programme client (applet).

Inconvénient : problème de performance (temps de réponse trop lent) vu que le serveur HTTP est désigné pour servir des pages Web plutôt qu'un serveur de bases de données intermédiaires.



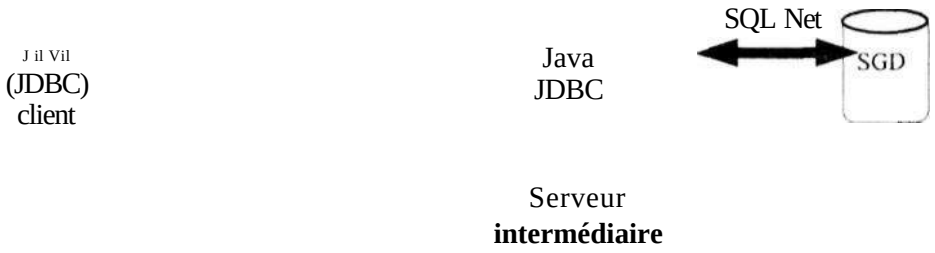
2. Accès **JDBC via un serveur C++ et ODBC**

Dans ce cas le serveur intermédiaire est écrit en C ou en C++ nécessitant l'utilisation d'ODBC.



3. Accès **JDBC via un serveur intermédiaire** Java

Dans cette architecture le serveur intermédiaire est entièrement écrit en Java. C'est la solution idéale dans le cas des applets.



Conclusion

JDBC est une API de bas niveau permettant d'interroger très facilement et de manière virtuelle n'importe quelle base de données relationnelle. De même, il n'est pas nécessaire d'écrire un programme pour accéder à une base de données Sybase, un autre programme pour accéder à une base de données Oracle et un autre pour accéder à une base de données Informix, etc. Un seul programme est écrit une seule fois et pouvant s'exécuter partout.

JDBC étend les possibilités de Java. En effet, il est très possible de publier une page Web contenant une applet qui utilise de l'information acquise à partir d'une base de données distante. Ou par exemple, dans le cas d'une entreprise, elle peut utiliser JDBC pour connecter tous ses employés (même s'ils sont sur des machines différentes) à une ou plusieurs bases de données internes via un Intranet. De même, pour les administrateurs de bases de données, grâce à Java et JDBC, ils peuvent propager de l'information d'une manière facile et économique : Le temps de développement pour de nouvelles applications est nettement court, l'installation et le contrôle des différentes versions sont plus simples.

L^ii^'"'*'*'"'| Références Bibliographiques I

"Choosinga Java Database Connectivity driver", 1997
http://www.weblogic.com/vvhitcpapers/jdbc.html

"JDBC Guide : Getting Started", Mars 1997
JDK1.1.3 Documentation

"The JDBC connection"
David S. Linthicom, Octobre 1996
DBMS

"Java-boost your databases"
Gutierrez, Dan D., Novembre 1996
Data Based Advisor

"Java database class libraries"
Ewbank, Kay, Mai 1997
DBMS