








JAVA DANS LA BASE DE DONNEES ORACLE

Table des matières

 Documentation et références.....	2
 Glossaire.....	2
 Java et le SGBD Oracle : une combinaison robuste	2
 La JVM au sein du SGBD Oracle	3
 Principaux composants de la JVM Oracle.....	5
 Stratégie Oracle des applications Java	9
 Développer des procédures stockées en Java	12



Documentation et références

Documentation Oracle : Oracle Database Java Developer's Guide

http://docs.oracle.com/cd/E16655_01/java.121/e17658/chone.htm



Glossaire

AWT :	Abstract Window Toolkit. Bibliothèque graphique pour Java, faisant partie de JFC (Java Foundation Classes).
Headless mode :	Cette option indique à la JVM de tenter d'utiliser les bibliothèques graphiques natives même en l'absence d'environnement graphique.
JDBC :	Java DataBase Connectivity. API Java de bas niveau pour l'accès aux BD avec le langage SQL.
JRE :	Java Runtime Environment. Regroupe le nécessaire à l'exécution des applications : la JVM qui permet d'interpréter et d'exécuter du code Java et les bibliothèques standards, notamment le fichier archive rt.jar qui regroupe les API.
JSE :	Java Standard Edition. Applications destinées aux "ordinateurs de bureau" sont développées à l'aide de JSE, qui fournit les classes d'interface utilisateur nécessaires.
JVM :	Java Virtual Machine. Permet d'interpréter et d'exécuter du code Java.
JLS :	Java Language Specification. Les spécifications du langage de programmation Java.
OCI :	Oracle Call Interface. API comme JDBC et ODBC, spécifique à Oracle, pour accéder à une base de données Oracle.
SQLJ :	Embedded SQL in Java. Code SQL 'embarqué' dans Java.



Java et le SGBD Oracle : une combinaison robuste

La base de données Oracle supporte le développement, le stockage et le déploiement des applications Java.

La combinaison de Java et de la base de données Oracle vous aide à créer des applications orientées composants dans une architecture réseau. Ces applications peuvent s'adapter facilement aux changements des besoins du métier.

Avec ces deux technologies, vous pouvez migrer les applications et leurs espaces de stockage du bureau vers des architectures orientées réseau. Plus important encore, vous pouvez accéder à ces applications et aux données stockées à distance à partir de n'importe quel

terminal client.

La figure 1 montre une architecture 2-tiers (client-serveur) traditionnelle, dans laquelle le client appelle des procédures stockées écrites en Java de la même façon que vous appelez des procédures stockées écrites en PL/SQL. Cette figure montre aussi comment les gestionnaires des services de connexions peuvent combiner plusieurs types de connexion vers la même base de données Oracle.

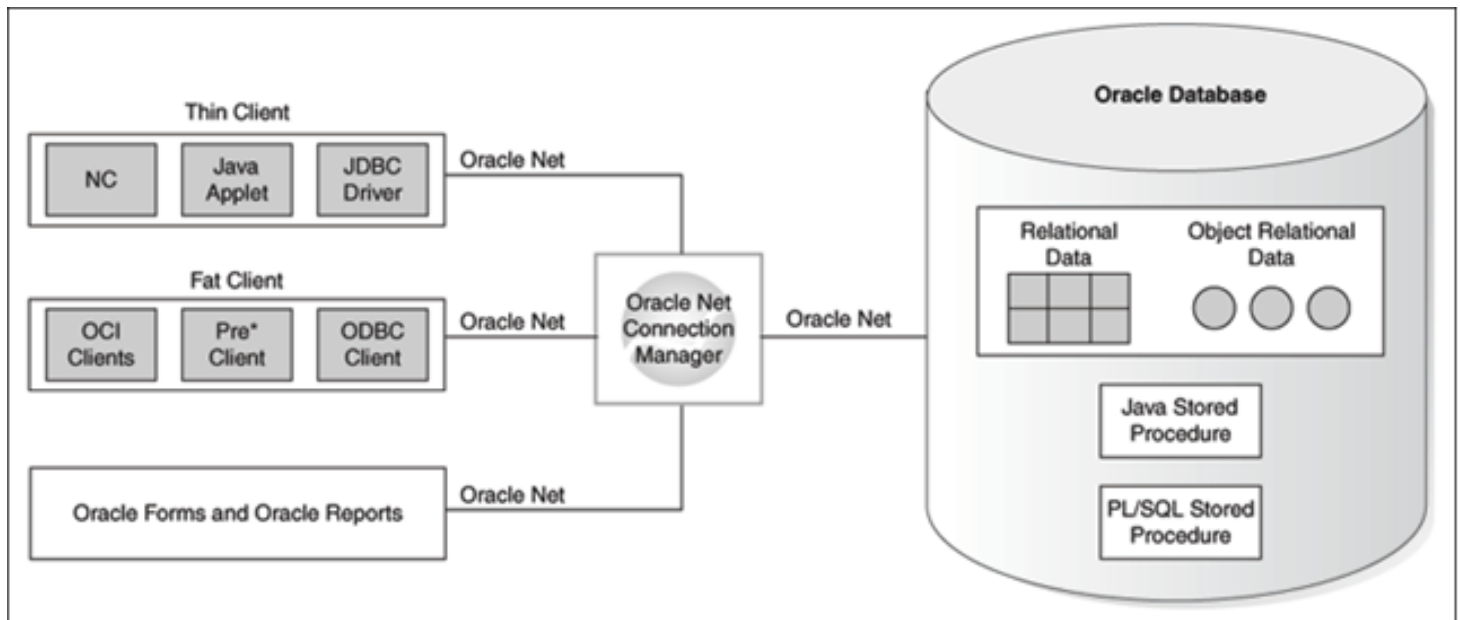


Figure 1 : source Oracle

La JVM au sein du SGBD Oracle

La JVM présente au sein du SGBD est une JVM standard, elle permet d'exécuter n'importe quelle application JAVA.

Elle est compatible avec la JLS (Java Language Specification) et la spécification de la JVM de Sun Microsystems. Elle supporte le format des classes Java standard et les API standard de la JSE. Elle supporte le chargement dynamique des classes JAVA.

Java dans Oracle introduit les termes suivants :

- **Session** : une session dans l'environnement Java du SGBD Oracle est identique à une session standard dans Oracle. Une session est limitée dans le temps par la connexion d'un seul utilisateur à la base de données. En tant qu'utilisateur qui appelle des procédures stockées en Java vous devez établir une session sur le serveur.
- **Appel** : quand un utilisateur appelle un code Java dans une session, on peut dire que l'on est dans un contexte d'un « Appel ». Un appel survient dans l'une des circonstances suivantes :
 - un programme SQL client exécute une procédure stockée écrite en Java.
 - un déclencheur exécute une procédure stockée écrite en Java.

- un programme écrit en PL/SQL appelle une procédure stockée écrite en Java.

La JVM Oracle, c'est à dire la JVM présente dans la base de données, utilise différentes techniques de nettoyage de la mémoire selon le type de la mémoire utilisée. Ces techniques ont une grande efficacité et un niveau optimisé d'utilisation de la mémoire.

Les deux types de mémoire utilisés dans la JVM Oracle sont : l'espace d'appel et l'espace de session.

Espace mémoire	Description
L'espace d'appel	Ce type de mémoire se caractérise par la vitesse de traitement des appels et son faible coût en mémoire. Il existe principalement pour la durée de l'appel. Il est divisé en deux parties : « new segment et old segment ». Tous les nouveaux objets sont dans le segment « new ». Tous les objets qui survivent à plusieurs passages du ramasse-miettes sont transférés dans le segment « old ».
L'espace de session	Ce type de mémoire se caractérise par un coût important concernant l'utilisation de la mémoire. Il existe tout au long de la durée de vie de la session. Toutes les variables statiques et tous les objets qui sont utilisés en dehors du cycle de vie d'un appel existent dans cet espace.

Méthode main()

Dans les applications Java de type clients lourds, vous déclarez une méthode avec la signature suivante :

```
public static void main (String [] args)
```

Cette méthode est le point d'entrée de l'exécution de l'application.

Les applications Java côté serveur (Côté Oracle) ne sont pas soumises à un seul point d'entrée représenté par cette méthode main().

Dans ce type d'application, chaque client commence une session, appelle les méthodes des classes sur le serveur à travers des points d'entrées du haut niveau. Dans ce schéma l'environnement du serveur cache le processus de gestion des sessions, le réseau et les ressources partagées des applications hébergées.

IHM sur le serveur

Le serveur ne permet pas l'utilisation des IHM, mais il permet l'utilisation de la logique qui les gère. La JVM Oracle supporte le headless mode de la librairie graphique AWT. Toutes les classes de la bibliothèque AWT sont disponibles et vous pouvez les utiliser dès lors que vous n'essayez pas de matérialiser une IHM sur le serveur.

Outils de développement

Vous pouvez développer votre code sur n'importe quel environnement de développement (IDE) de votre choix et effectuer les tests nécessaires et puis déployer ce code sur la base de données pour qu'il soit accessible par tous les clients connectés à la base de données.



Principaux composants de la JVM Oracle

La JVM Oracle est une JVM standard. Elle s'exécute dans le même espace de processus que le noyau (kernel) de la base de données. Elle partage sa mémoire et accède directement à ses données relationnelles.

La JVM Oracle supporte tous les aspects du langage Java et contient toutes les classes que l'on trouve dans l'API standard.

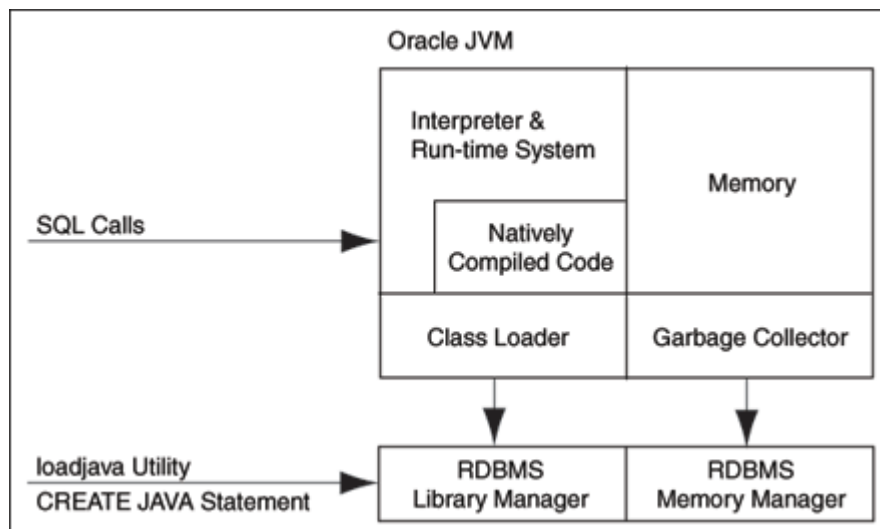


Figure 2 : source Oracle

La JVM Oracle inclut les espaces de nommage Java standards dans la base de données ce qui permet aux applications Java d'accéder à des objets Java stockés dans la base de données et le serveur d'applications à travers l'entreprise.

La JVM Oracle est étroitement intégrée dans une architecture évolutive. Les programmes

Java utilisent les « appels », les « sessions » Oracle et le cycle de vie des objets Java, efficacement, sans aucune intervention de l'utilisateur.

La section suivante montre un aperçu des composants de la JVM Oracle, le pilote JDBC et le traducteur SQLJ intégrés.

- Le gestionnaire des librairies Java.
- Le compilateur.
- L'interpréteur.
- Le « Class Loader ».
- Le vérificateur.
- Le pilote JDBC interne (côté serveur).
- Le traducteur SQLJ (côté serveur).
- Les Classes Système.

Gestionnaire des librairies Java

Pour stocker des classes Java dans une base de données Oracle, vous devez utiliser l'outil de ligne de commande **loadjava**. Ce gestionnaire de librairies permet de charger dans la base de données :

- des sources Java (qui seront compilées par le compilateur Java intégré),
- des classes Java (fichiers sources compilés),
- des ressources Java,

L'outil **loadjava** est un script (.bat) présent dans le répertoire \$Oracle_Home\Bin. Il utilise la commande SQL « CREATE JAVA ».

```
CREATE JAVA {SOURCE | CLASS | RESOURCE}
```

Documentation de CREATE JAVA :

http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_5013.htm

L'outil **loadjava** peut être exécuté à partir de l'invite MS-DOS ou de SQL*Plus.

Appel à partir de MS-DOS.

```
C:\> loadjava -user nomschema/motdepasse@instance ClasseJava
```

Appel à partir de SQL*plus.

```
CALL dbms_java.loadjava('... options...');
```

Ces objets sont stockés en tant qu'objets de schéma, au même titre que les tables ou les déclencheurs..., et ne sont accessibles (à l'utilisation) que par la JVM Oracle.

Vous pouvez supprimer une procédure Java de la base en utilisant l'outil **dropjava**. L'outil **dropjava** peut être exécuté à partir de l'invite MS-DOS ou de SQL*Plus.

Appel à partir de SQL*plus.

```
CALL dbms_java.dropjava('... options...');
```

Compilateur

La JVM Oracle inclut un compilateur Java standard. Il est l'équivalent de l'outil **javac** présent avec le JDK. Quand la commande CREATE JAVA SOURCE s'exécute, elle fait appel au compilateur pour compiler les sources et générer automatiquement les classes Java.

Interpréteur

Il est responsable de l'exécution du code Java. Il est l'équivalent de l'outil **java** présent avec le JDK ou la JRE.

Class Loader

En réponse aux requêtes du système, le Class Loader localise, charge et initialise les classes stockées dans la base de données. Il lit la classe et génère la structure de données nécessaire pour l'exécuter. Les données immuables et les métadonnées sont chargées dans une zone mémoire partagée. Le résultat de ce processus est l'optimisation de l'utilisation de la mémoire par la session.

Le Class Loader essaye de résoudre les références extérieures quand cela est nécessaire. De plus, si le code source est disponible alors il appelle le compilateur automatiquement quand une nouvelle version doit être recompilée.

Vérificateur

Le vérificateur empêche l'utilisation accidentelle des classes falsifiées qui peuvent violer les restrictions d'accès.

Pilote JDBC interne (côté serveur Oracle)

Possédant un point d'entrée de bas niveau, le pilote JDBC intégré fournit un accès direct aux

données dans les procédures stockées écrites en Java. Il supporte la spécification standard de JDBC, donc la partition d'une application entre le serveur et le client est rendu facile.

Téléchargement pilote : <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html>

Traducteur SQLJ (côté serveur Oracle)

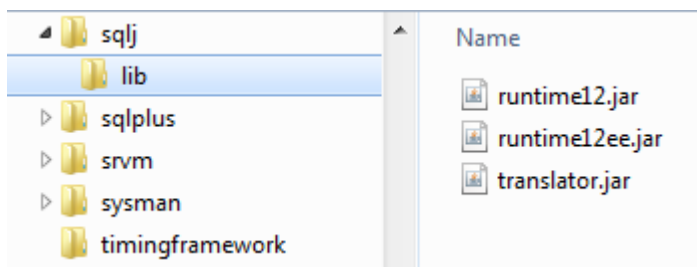
SQLJ vous permet d'intégrer des commandes SQL directement dans le code Java. Il est plus concis que JDBC, plus rapide pour des commandes SQL statiques et effectue des vérifications des types dans la phase de traduction.

SQL est un préprocesseur écrit en Java, il prend en entrée du code Java dans lequel on intègre des commandes SQLJ, puis le traducteur traduit ce code en classes java en implémentant les appels SQL en code Java après des vérifications de la syntaxe des appels et les types utilisés en paramètre.

Le traducteur SQLJ est optimisé pour la base de données Oracle, il donne un accès direct aux données en utilisant le pilote JDBC interne. La syntaxe SQLJ inclut les commandes SQL de manipulation des données (LMD), les commandes de définition des données (LDD), les commandes de contrôle de transactions et les appels des procédures stockées.

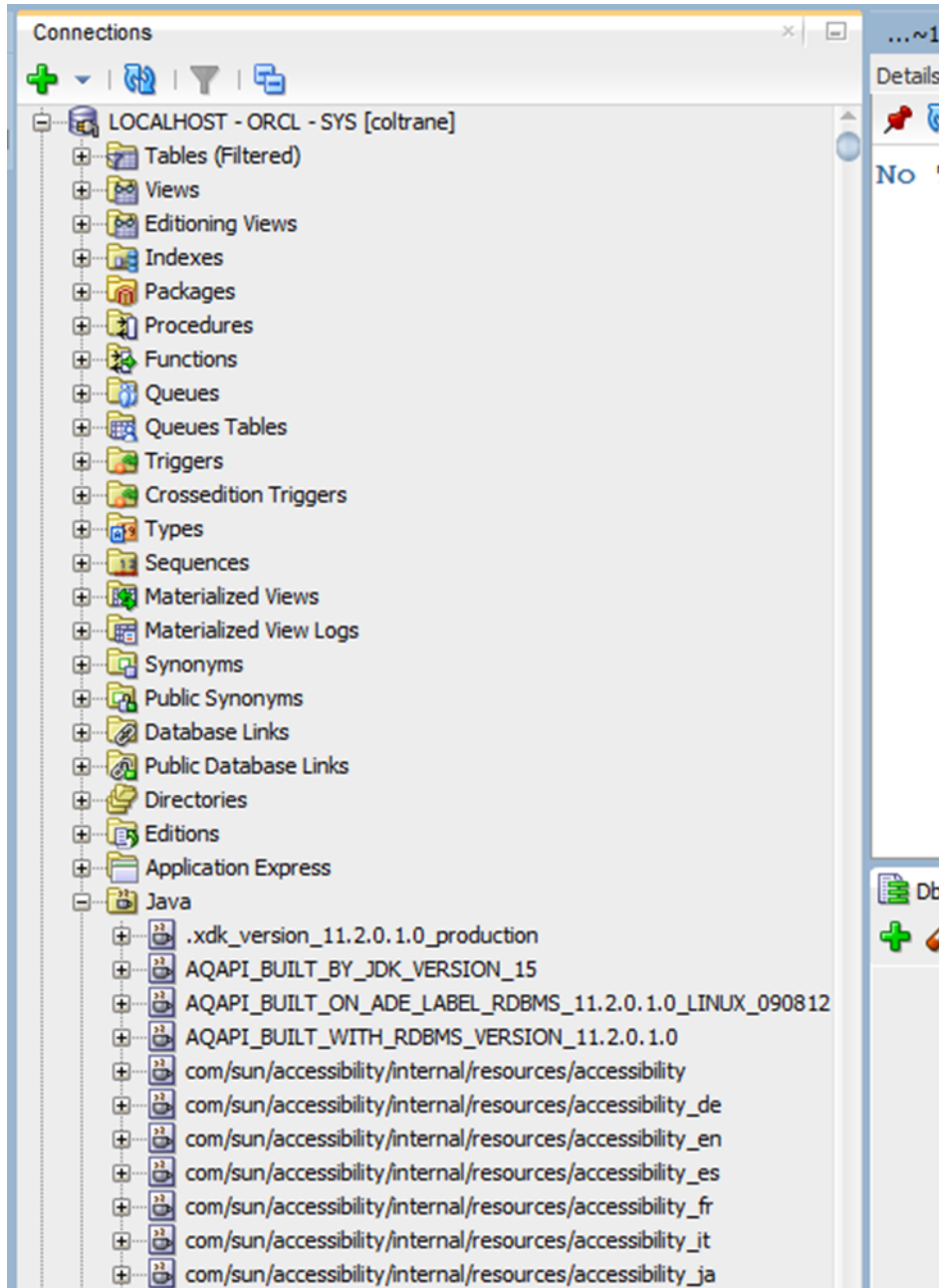
Le code SQLJ utilisé côté client peut être chargé et utilisé dans la base de données sans aucune modification.

Les librairies SQLJ sont présentes dans le répertoire \$ORACLE_HOME\sqlj\lib.



Classes Système

Une grande partie des classes de l'implémentation de Java (les API de la JRE) sont présentes dans Oracle. Elles sont connues sous le nom de « Classes Système », ces classes sont définies dans le schéma SYS et exportées vers tous les utilisateurs avec le synonyme public. Une classe qui a le même nom qu'une classe système peut être définie dans un autre schéma que SYS, mais cela est déconseillé car ça peut engendrer un comportement inattendu à l'exécution.



Stratégie Oracle des applications Java

Oracle fournit aux développeurs de nombreuses solutions pour la création, le déploiement et la gestion des applications Java. La totalité de ces solutions est constituée d'interfaces de programmation côté client et côté serveur, les outils de développement et la JVM Oracle intégrée dans la base de données. Ces produits sont pleinement compatibles avec les standards Java.

Dans cette section, nous aborderons les concepts suivants :

- Le développement des applications Java dans la base de données.

- L'environnement de programmation Java.
- Les procédures stockées écrites en Java.
- L'intégration du PL/SQL avec les fonctionnalités du SGBD Oracle.
- Les outils de développement.

Développement d'applications Java dans la base de données

Les aspects les plus importants du développement d'applications Java dans la base de données sont :

- Fournir une grande partie du standard J2SE pour l'accès aux données via JDBC et SQLJ.
- Fournir une interface d'interactions avec les standard de J2EE par :
 - L'interface d'appel de composants WEB externes comme les JSP et Servlets.
 - L'interface des services WEB et d'appel de services WEB.
- L'utilisation de la JVM Oracle comme une interface d'intégration dans les ERP.

Vous pouvez appeler des procédures Java à partir des procédures PL/SQL et vice versa. Vous pouvez utiliser les interfaces JDBC et SQLJ pour l'accès aux données.

Le tableau suivant peut vous aider à décider quand utiliser chacune des API.

Type de fonctionnalité	JAVA API
Pour avoir des procédures que l'on peut appeler à partir de SQL comme les déclencheurs	Procédures stockées JAVA
Pour faire appel à des commandes SQL simples et statiques dans des tables ayant des colonnes connues lors de l'appel à partir des objets Java.	SQLJ
Pour faire des appels dynamiques et complexes à partir des objets Java.	JDBC

Procédures stockées écrites en Java

Ce sont des programmes écrits en Java et déployés sur la SGBD. Ils s'exécutent comme les procédures PL/SQL. Vous pouvez les invoquer directement à partir de SQL*Plus ou indirectement avec les déclencheurs. Vous pouvez y accéder à partir de n'importe « Oracle Net Client » comme OCI, JDBC ou SQLJ.

Vous pouvez aussi utiliser Java pour développer des programmes côté serveur qui sont indépendants de PL/SQL.

Intégration du PL/SQL avec les fonctionnalités de la SGBD Oracle

Depuis un code PL/SQL existant, vous pouvez appeler des procédures Java et vice versa. Cette solution vous offre par exemple la puissance de l'API net de Java, la possibilité de déployer votre application sur le client et sur le serveur sans modification.

Pilotes JDBC

Les différents types de pilotes JDBC Oracle sont listés dans le tableau ci-dessous.

Pilote	Description
Pilote JDBC Thin	<p>Vous pouvez l'utiliser pour écrire des applications pures Java pour accéder aux données SQL.</p> <p>Il peut être téléchargé dynamiquement comme une applet JAVA.</p> <p>Il est souhaitable pour les applications WEB.</p>
Pilote JDBC OCI	<p>Il n'est pas pure Java.</p> <p>Il accède au code natif spécifique à Oracle.</p> <p>Il demande plus de mémoire.</p> <p>Il doit être installé côté client.</p>
Pilote JDBC interne (côté serveur)	<p>Il s'exécute dans la base de données pour accéder localement aux données.</p> <p>Il offre une haute performance.</p> <p>Il utilise les librairies de la SGBD directement sans le temps d'appel sur le réseau ou le surcoût d'une connexion entre le programme java et les données SQL.</p> <p>Oracle supporte le standard JDBC sans nécessité la réécriture du code déjà existant côté client.</p>

Préprocesseur SQLJ

Oracle, en collaboration avec d'autres vendeurs de SGBD, ont développé une interface

standard, appelée SQLJ, pour intégrer du code SQL au code Java.

Le résultat est le standard ANSI x.3.135.10-1998, qui offre une solution plus simple et hautement plus productive que JDBC.

Le développeur écrit l'application avec des appels du haut niveau à l'API SQLJ et passe le code au préprocesseur qui produit un code qui utilise l'API de JDBC.

SQLJ offre aux développeurs une interface puissante qui peut à la fois servir à développer des applications côté client ou côté serveur.

Dans Oracle, vous pouvez utiliser SQLJ dans :

- les procédures stockées.
- les déclencheurs.
- les fonctions.

Vous pouvez également combiner SQLJ avec JDBC.

Outil de développement JPublisher

JPublisher est un outil en ligne de commande écrit en Java qui peut :

- générer des classes Java pour présenter des entités de la base de données tels que les objets SQL et PL/SQL dans le client Java de la base de données.
- publier à partir de SQL, PL/SQL et Java côté serveur vers des services WEB.
- générer du code qui fait appel à des services WEB externes à la base de données à partir du code interne.

JPublisher peut créer des classes pour présenter les types d'entités suivants :

- Objets SQL définis par l'utilisateur.
- Références vers des objets.
- Collections SQL définies par l'utilisateur.
- Packages PL/SQL.
- Classes Java côté serveur.
- Commandes SQL de manipulation de données (LMD).

JPublisher offre la possibilité de spécifier et customiser le mapping de ces entités vers le code Java.

Documentation : http://docs.oracle.com/cd/E18283_01/java.112/e10587/intro.htm



Développer des procédures stockées en Java

Etapes de création de procédures stockées en Java

Vous pouvez exécuter des procédures stockées écrites en Java de la même manière que vous exécutez des procédures PL/SQL.

Normalement, l'exécution d'une procédure écrite en Java est le résultat d'une manipulation de la base de données via un déclencheur ou une commande SQL de type LMD (Langage de Manipulation des Données).

Pour pouvoir appeler une procédure écrite en Java, vous devez d'abord la télécharger et la publier par la spécification d'appel PL/SQL.

Quand on parle de téléchargement et de publication, on parle ici de deux tâches bien distinctes :

- Le téléchargement concerne toutes les classes de l'utilisateur.
- La publication concerne juste les méthodes que l'on souhaite appeler comme procédures. Les autres méthodes utilisées par la méthode de la procédure appelée ne sont jamais publiées.

Pour télécharger une procédure automatiquement, vous pouvez utiliser l'outil « loadjava ». Cet outil peut télécharger dans la base de données :

- des sources java.
- des classes java.
- des ressources utilisées par le code java.

Les étapes de développement sont les suivantes :

1. Créer les classes Java.
2. Télécharger les classes Java.
3. Publier les classes Java.
4. Appeler les procédures stockées.
5. Déboguer les procédures si nécessaire.

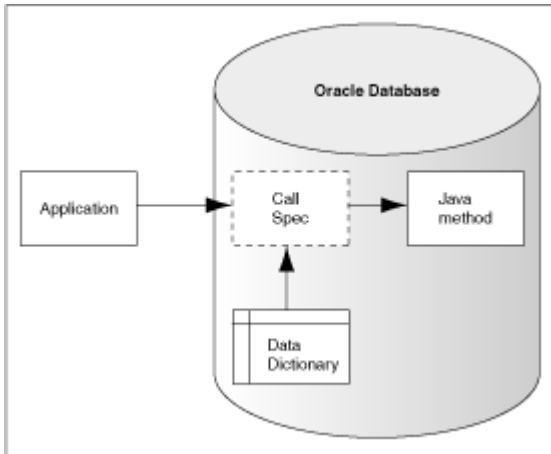
Comprendre les spécifications d'appel

Pour publier une méthode Java, vous devez créer une spécification de fonction ou de procédure (selon si la méthode retourne une valeur ou pas). Vous utilisez pour cela les commandes :

- CREATE FUNCTION
- CREATE PROCEDURE

A ces commandes, vous devez ajouter les mots clés LANGUAGE JAVA.

La figure suivante montre le schéma d'appel de la spécification sur la base de données.



Syntaxe

```

CREATE [OR REPLACE]
{ PROCEDURE procedure_name [(param[, param]...)]
| FUNCTION function_name [(param[, param]...)] RETURN sql_type}
[AUTHID {DEFINER | CURRENT_USER}]
[PARALLEL_ENABLE]
[DETERMINISTIC]
{IS | AS} LANGUAGE JAVA
NAME 'method_fullname (java_type_fullname[, java_type_fullname]...)
[return java_type_fullname]';

```

param := parameter_name [IN | OUT | IN OUT] sql_type

Définir la spécification d'appel

Une spécification et la méthode qu'elle appelle doivent résider dans le même schéma de la base de données.

À moins que la méthode ne soit déjà déclarée publique dans ce cas vous pouvez déclarer la spécification dans les contextes suivants:

- Une fonction ou procédure PL/SQL standalone.
- Une fonction ou procédure PL/SQL packagée.
- Une méthode membre d'un objet SQL.

Une spécification d'appel expose le point d'entrée du haut niveau de la méthode Java dans la base Oracle, cela restreint les méthodes que vous pouvez publier aux méthodes déclarées public static.

Cependant, vous pouvez publier des méthodes d'instance en tant que membres de type d'objets SQL.

Les spécifications d'appel packagées sont des points d'entrées de haut niveau, elles sont

définies dans le body du package PL/SQL, dans ce contexte de définition vous pouvez alors surcharger les spécifications d'appel.

Faire correspondre les types de données entre la spécification d'appel et les types Java

Le tableau suivant présente le mapping entre les types SQL et les types Java.

Types SQL	Classes Java
CHAR, VARCHAR2, LONG	java.lang.String , oracle.sql.CHAR
NUMBER	Boolean , char , byte , short , int , long , float , double Byte , Short , Integer ,Long ,Float ,Double il faut spécifier le nom complet de la classe, exemple : java.lang.Byte java.math.BigDecimal , oracle.sql.NUMBER
BINARY_INTEGER	Boolean , char , byte , short , int , long
BINARY_FLOAT	oracle.sql.BINARY_FLOAT
BINARY_DOUBLE	oracle.sql.BINARY_DOUBLE
DATE	oracle.sql.DATE
RAW	oracle.sql.RAW
BLOB	oracle.sql.BLOB
CLOB	oracle.sql.CLOB
BFILE	oracle.sql.BFILE
ROWID	oracle.sql.ROWID
TIMESTAMP	oracle.sql.TIMESTAMP
TIMESTAMP WITH TIME ZONE	oracle.sql.TIMESTAMPTZ
TIMESTAMP WITH LOCAL TIME ZONE	oracle.sql.TIMESTAMPLTZ
ref cursor	java.sql.ResultSet

Types SQL	Classes Java
	sqlj.runtime.ResultSetIterator
user defined named types, ADTs	oracle.sql.STRUCT , byte[]
opaque named types	oracle.sql.OPAQUE, byte[]
nested tables and VARRAY named types	oracle.sql.ARRAY, byte[]
references to named types	oracle.sql.REF, byte[]

Utiliser le pilote JDBC interne

Le pilote interne à la base de données s'exécute dans contexte de session et de transaction par défaut, En conséquence on est déjà connecté à la base de données, et tous nos appels SQL font partie de la transaction par défaut, de ce fait il n'est pas nécessaire d'enregistrer le pilote, pour obtenir une connexion.

Dans JDBC, on utilise l'URL du pilote interne par défaut :

```
Connection conn = DriverManager.getConnection("jdbc:default:connection:");
```

Vous pouvez utiliser cette connexion pour créer des Statements ou des PreparedStatement et même des CallableStatements pour appeler d'autres procédures stockées.

Il faut prendre ces points en considération quand vous utilisez le pilote JDBC interne :

- Les opérations SQL exécutées avec le pilote font partie d'un contexte de transaction par défaut. Il ne s'agit pas d'un contexte de transaction global, il est le même que celui implémenté par le standard Java (Java Transaction API ou le Java Transaction Service).
- Vous devez penser à refermer les Statements et les ResultSets après utilisation, ou bien, demander au DBA d'augmenter la limite de curseurs ouverts (le paramètre d'initialisation OPEN_CURSORS).
- Le pilote interne ne supporte pas l'autocommite, en conséquence, l'application doit faire le commit ou le rollback explicitement.
- Vous ne pouvez pas vous connecter à une autre base de données avec le pilote interne. Pour une connexion de serveur à serveur, vous devez utiliser le pilote Thin de JDBC.

- Vous ne pouvez pas fermer la connexion par défaut. Si vous appelez la méthode `close()` de la connexion, tous les objets qui référencent la connexion sont alors nettoyés et fermés. Pour obtenir une nouvelle connexion, vous devez utiliser la méthode `getConnection()`.