

# Gestion avancée d'image sous Oracle avec Java (ORDImage)

par [Benoît Maréchal](#)

Date de publication : 12 septembre 2006

Dernière mise à jour : 13 septembre 2006

Oracle offre des fonctionnalités très avancées sur les images, et permet entre autres, d'effectuer des recherches sur les propriétés physiques et sémantiques des images. Il peut également les manipuler, en les copiant, en les rognant, en changeant leur contraste, leur gamma etc. Nous verrons dans cet article l'ensemble des ces fonctionnalités, et comment les utiliser depuis un programme Java.

## Avant-Propos

- 0.1 - Remerciement
- 0.2 - Pré-requis
- 0.3 - Introduction
- 1 - Un peu de théorie : La gestion d'images sous Oracle et Java
  - 1.1 - Gestion d'images coté SQL
    - 1.1.1 - Exemple d'une table permettant de gérer des images
    - 1.1.2 - Le stockage des données multimédia : Le type OrdSource
    - 1.1.3 - Gestion des images et de ses métas données par Oracle : Le type ORDImage
    - 1.1.4 - La comparaison d'image : Le type ORDImageSignature
      - 1.1.4.a - Génération de signature (en PL/SQL)
      - 1.1.4.b - Comparaison d'images
  - 1.2 - La gestion d'images coté Java
    - 1.2.1 - Configuration de l'environnement Java
    - 1.2.2 - La classe OrdImage
    - 1.2.3 - La classe OrdImageSignature
- 2 - Manipulation d'images en Java
  - 2.1 - Importation des packages
  - 2.2 - Ajout d'un ORDImage
    - 2.2.1 - Etape 1 : Connexion à la base de données Oracle
    - 2.2.2 - Etape 2 : Désactiver l'autocommit
    - 2.2.3 - Etape 3 : Initialiser l'attribut ORDImage
    - 2.2.4 - Etape 4 : Récupérer le descripteur
    - 2.2.5 - Etape 5 : Envoyer l'image
    - 2.2.6 - Etape 6 : Générer les propriétés de l'image et mettre à jours la base
    - 2.2.7 - Etape 7 : Validation manuel des données
    - 2.2.8 - Etape 8 : Fermeture des connexions et remise à true de l'autocommit.
    - 2.2.9 - Exemple complet
  - 2.3 - Récupération d'une image et de ses métas données depuis un ORDImage
    - 2.3.1 - Etape 1 : Connexion à la base et désactivation de l'autocommit
    - 2.3.2 - Etape 2 : Récupération du descripteur
    - 2.3.3 - Etape 3 : Récupération de l'image
    - 2.3.4 - Etape 4 : Affichage des propriétés de l'image
    - 2.3.5 - Etape 5 : Fermeture des connexions
    - 2.3.6 - Exemple complet
  - 2.4 - Génération de signature
    - 2.4.1 - Etape 1 : Connexion à la base et mise à false de l'autocommit
    - 2.4.2 - Etape 2 : Initialisation de l'attribut ORDImageSignature
    - 2.4.3 - Etape 3 : Récupération du descripteur de l'ORDImage
    - 2.4.4 - Etape 4 : Récupération du descripteur de l'ORDImageSignature
    - 2.4.5 - Etape 5 : Génération de la signature
    - 2.4.6 - Etape 6 : Mise à jours de la signature dans la base
    - 2.4.7 - Etape 7 : Validation des modifications
    - 2.4.8 - Etape 8 : Remise à true de l'autocommit et fermeture des flux
    - 2.4.9 - Exemple complet
  - 2.5 - Comparaison d'image
    - 2.5.1 - Etape 1 : Se connecter à la base de données et désactiver l'autocommit
    - 2.5.2 - Etape 2 : Récupérer les descripteurs des signatures des deux images à comparer
    - 2.5.3 - Etape 3 : Comparer des signatures
    - 2.5.4 - Exemple complet
- 3 - Conclusion
- 4 - Téléchargement

## Avant-Propos

### 0.1 - Remerciement

Je tiens tout d'abord à remercier [Ricky81](#) et [Jaouad](#) qui m'ont aidé pour mes premiers pas en tant que rédacteur sur [developpez.com](#). Un grand merci également à [Maximilian](#) pour sa relecture ;)

### 0.2 - Pré-requis

Pour comprendre ce tutorial il est nécessaire que le lecteur connaisse et maîtrise les notions suivantes :

- Les instructions de base du langage SQL
- Quelques notions de PL/SQL
- Le langage Java
- Quelques connaissances sur la gestion des LOB en Oracle/Java sont également les bienvenues

### 0.3 - Introduction

Les bases de données sont réputées pour leur facilité à gérer les données textuelles. Mais l'évolution des technologies permet de nos jours la manipulation de documents multimédia contenant également sons, images et vidéos. Les bases de données modernes doivent donc répondre à diverses questions et besoins sur la manipulation d'images.

Cet article se concentre sur la base de données Oracle (version 9i ou supérieur) ainsi que le langage Java. Son but : vous permettre de manipuler des images contenues dans une base de données Oracle depuis un programme Java, en répondant aux questions suivantes :

*Quels mécanismes ont été mis en œuvre par Oracle pour gérer des images ? Comment Java permet-il de manipuler les images depuis une base Oracle ?*

Avant de pouvoir programmer en java la manipulation d'images avec Oracle, il est nécessaire d'avoir un ensemble de connaissances suffisantes pour comprendre le mécanisme mis en place par Oracle. Pour cela j'ai réalisé une première partie présentant en détails ce fonctionnement. Une lecture attentive de cette partie mélangeant théorie et référence complète de certaines classes est très importante si vous ne connaissez pas Oracle Intermedia et les OrdImages. Cependant, si vous maîtrisez déjà ses différents concepts vous pouvez très bien commencer la lecture dès la deuxième partie : [La manipulation d'OrdImage en Java](#). Cette deuxième partie s'attardera uniquement sur la pratique de la manipulation d'OrdImage en java en donnant des exemples fonctionnels complets venant étayer les explications. Cette seconde partie ne reviendra donc absolument pas sur les notions et concepts expliqués dans la première partie. Enfin, nous terminerons par un QCM qui permettra de vérifier la bonne compréhension du tutorial par le lecteur.

## 1 - Un peu de théorie : La gestion d'images sous Oracle et Java

Nous allons voir dans cette partie les concepts mis en place par Oracle pour la manipulation d'image. La documentation française manquant cruellement sur les OrdImages, cette partie contient par moment des références complètes des méthodes ou classes notamment pour la classe Java OrdImage. Ces parties très utiles lors de la conception de programme peuvent être fastidieuses pour une première lecture et peuvent donc, dans un premier temps, être passées puis consultées après la lecture du tutorial, puisqu'elles n'empêchent pas la compréhension de ce dernier. Cette partie pourra également être consultée en tant que référence par le lecteur, lorsqu'il aura acquis un certain niveau.

### 1.1 - Gestion d'images coté SQL

Oracle définit un ensemble d' "Objets-Relationnels" (également appelé "Types Abstraits de Données", ou encore "classe") définissant les attributs et méthodes des données multimédia, images, vidéos et audios. Ces types sont organisés hiérarchiquement dans un package nommé ORDSYS. Pour la gestion d'images nous nous intéresserons seulement aux types ORDSOURCE, ORDImage et ORDImageSignature de ce package.

*Dans la suite de ce tutorial, nous parlerons de "type" pour les Objets relationnels tels que ORDImage et de "classe" pour les classes Java telles que OrdImage. Notez également l'utilisation de casses différentes.*

#### 1.1.1 - Exemple d'une table permettant de gérer des images

Pour gérer des images et pouvoir les manipuler, les comparer et voir leurs métas données deux attributs sont obligatoires dans la table qui les gère : Un premier qui doit être de type ORDSYS.ORDImage, et un second qui doit être de type ORDSYS.ORDImageSignature.

Voici un exemple de création d'une telle table :

##### Création d'une table permettant une gestion avancée d'images sous Oracle

```
-- Création d'une table permettant la gestion d'images
-----
CREATE TABLE MaTable (
    id INTEGER PRIMARY KEY,
    image ORDSYS.ORDImage,
    signature ORDSYS.ORDImageSignature
);
```

Cette table nous permettra de réaliser quelques exemples de code dans la seconde partie consacrée à la manipulation d'image en Java.

Les points suivants vont s'attarder à l'explication et la description des types ORDImage et ORDImageSignature du package SQL : ORDSYS.

#### 1.1.2 - Le stockage des données multimédia : Le type OrdSource

Au sommet de la hiérarchie du package ORDSYS se trouve le type ORDSOURCE. Ce type est chargé de contenir le document multimédia (une image dans notre cas) ainsi que diverses informations sur son stockage. Il est également chargé de supporter les accès aux données stockées de type BLOB, BFILE ou URL.

Ces attributs sont :

**localData BLOB** : Si l'image est contenue dans la base et non en externe, ce champ contient les données multimédia stockées localement comme un BLOB.

**srcType VARCHAR2(4000)** : Identifie le type de source des données (FILE, HTTP).

**srcLocation VARCHAR2(4000)** : Emplacement de la source. Si srcType vaut FILE alors srcLocation contient le nom du dossier logique d'Oracle , Si srcType vaut HTTP alors srcLocation contient une URL. Il peut valoir également "user-défini".

*Un dossier logique d'Oracle permet d'associer un nom à un emplacement physique. Il est créé par l'instruction CREATE DIRECTORY.*

**srcName VARCHAR2(4000)** : Nom des données.

**updateTime DATE** : Détermine si le fichier est interne à la base ou non.

**local NUMBER** : Identifie le type de source des données (FILE, HTTP).

### 1.1.3 - Gestion des images et de ses métas données par Oracle : Le type ORDImage

Afin d'apporter des attributs et méthodes spécifiques aux images, le package ORDSYS contient un type nommé ORDImage. Ce type possède un attribut ORDSource pour le stockage de l'image. Il supporte un grand nombre de formats d'images des plus connus aux plus spécifiques.

Ces attributs sont :

**source ORDSource** : Source de l'image, pour les informations sur le stockage de l'image et l'image elle-même.

**height INTEGER** : Hauteur de l'image, en pixel.

**width INTEGER** : Largeur de l'image, en pixel.

**contentLength INTEGER** : Taille de l'image, en bytes.

**fileFormat VARCHAR2(4000)** : Type dans lequel l'image est stockée (jpg, tiff, gif, etc.)

**contentFormat VARCHAR2(4000)** : Type de l'image (monochrome, gris 8-bit, etc.)

**compressionFormat VARCHAR2(4000)** : Algorithme de compression utilisé (LZW, JPEG, etc.)

**contentType VARCHAR2(4000)** : type MIME de l'image (image/gif, image/jpeg, etc.)

Avant de pouvoir insérer une image dans un ORDImage il est nécessaire de l'initialiser. On utilise pour cela un constructeur appelé init(). Ce dernier possède deux formes :

**init()** : Sans paramètres, ce constructeur initialise par des valeurs NULL tous les attributs sauf updateTime mis à la date du jour (SYSDATE), local défini sur 1 et localData initialisé par empty\_blob().

**init(srcType,srcLocation,srcName)** : Avec paramètres, ce constructeur définit les attributs srcType, srcLocation, srcName par les valeurs données en paramètre mais contrairement à init() sans paramètre, l'attribut local est défini

à 0. Ce constructeur est donc à utiliser lorsqu'on veut gérer une image stockée à l'extérieur de la base de données.

#### Insertion d'une image gérée à l'extérieur de la base de données

```
INSERT INTO images VALUES  
(ORDSYS.ORDImage.init('file','AUDIR','image.gif'))
```

Voyons maintenant les principales méthodes que propose ORDImage.

Méthodes permettant de récupérer les propriétés de l'image :

**getCompressionFormat()** : Renvoie l'algorithme de compression de l'image.

**getContentFormat()** : Renvoie le type de l'image.

**getContentLength()** : Renvoie la taille de l'image.

**getFileFormat()** : Renvoie le type dans lequel l'image est stockée.

**getHeight()** : Renvoie la hauteur de l'image.

**getWidth()** : Renvoie la largeur de l'image.

Méthode permettant de gérer les métas données :

**setProperty()** : Après avoir ajouté l'image dans le type ORDImage on peut générer automatiquement les propriétés de l'image grâce à cette méthode ( détermine automatiquement la hauteur, la largeur, le format, etc.)

**checkProperties()** : La méthode checkProperties() permet de vérifier si les propriétés extraites de l'image correspondent à l'image.

Il est également possible de manipuler et modifier directement les images de la base de données grâce aux méthodes :

**process(commande)** : Permet de manipuler une image en changeant diverses propriétés ou en effectuant des transformations ou en appliquant des opérations.

**processcopy(commande, nouvelle\_image)** : Permet de copier une image dans une nouvelle\_img en changeant au passage diverses propriétés définies dans la commande passée en paramètre.

La commande a la syntaxe suivante :

**'propriété1= argument1 argument2[, propriété2=argument1 # ]#'**

**process(commande)** : Permet de manipuler une image en changeant diverses propriétés ou en effectuant des transformations ou en appliquant des opérations.

Les propriétés possibles sont :

**compressionFormat** : Change de format de compression (valeurs possibles : JPEG,LZW,GIFLZW, etc.).

**compressionQuality** : Change la qualité de compression (valeurs possibles : LOWCOMP, MEDCOMP, etc.).

**contentFormat** : Fixe le nombre de couleurs possible par pixel (valeurs possibles : MONOCHROME, GREY Scale, RGB, etc.).

**cut** : Permet de découper l'image selon 4 entiers : origin.x, origin.y, largeur, hauteur.

**FileFormat** : Change le format de l'image (valeurs possibles : GIFF, BMPF, etc.)

**FixedScale** : Redimensionne l'image selon deux entiers (largeur, hauteur).

**MaxScale** : Redimensionne l'image selon deux entiers (largeur, hauteur), en conservant les proportions.

**scale** : Spécifie un facteur de redimensionnement sur l'image.

**xscale, yscale** : Spécifie un facteur de redimensionnement selon les x ou les y respectivement.

Il existe encore d'autres propriétés un peu plus marginales telles que : flip, rotate, mirror, gamma, contrast etc.

Exemple :

#### Exemple d'utilisation de processcopy()

```
image.processcopy('fileFormat=JPEG, contentFormat=RGB', nouvelle_image);
```

*Comme nous l'avons vu le type ORDImage permet de gérer l'ensemble des informations d'une image sous forme d'attributs (également appelé les métas données de l'image). Mais qu'en est-il des caractéristiques physiques de l'image ? Comment peut-on comparer deux images entre elles ?*

### 1.1.4 - La comparaison d'image : Le type ORDImageSignature

Pour comparer des images sur leurs attributs visuels, il faut générer ce qu'on appelle la " signature " de l'image. La signature d'une image est obtenue grâce à une analyse de l'image par Oracle avec la méthode generateSignature() du type ORDImage. Cette signature contient des informations de couleurs, de textures et de formes selon chaque zone de l'image, elle contient aussi des informations sur le fond de l'image. Le résultat de cette analyse est alors contenu dans un type appelé ORDImageSignature.

#### 1.1.4.a - Génération de signature (en PL/SQL)

L'ensemble de ces attributs visuels est stocké dans un BLOB, seul attribut de ORDImageSignature. Avant de pouvoir générer la signature d'une image, il faut, à l'instar du type ORDImage appeler un constructeur, également appelé init(), qui permet d'initialiser l'attribut BLOB à un BLOB vide. Une fois ceci réalisé, il suffit d'appeler la méthode generateSignature en donnant en paramètre l'attribut ORDImage dont on veut obtenir les caractéristiques visuelles.

Exemple en PL/SQL :

#### Génération de signature d'image en PL/SQL

```
DECLARE
t_image ORDSYS.ORDImage;
image_sig ORDSYS.ORDImageSignature;
BEGIN
```

## Génération de signature d'image en PL/SQL

```

FOR c1 IN (SELECT * FROM images where numero<>-1) LOOP
SELECT photo_high_def, photo_sig INTO t_image, image_sig
FROM images
WHERE numero=c1.numero FOR UPDATE;
--génération de la signature
image_sig.generateSignature(t_image);
UPDATE images SET photo_sig = image_sig WHERE
numero=c1.numero;
end loop;
END;
/

```

## 1.1.4.b - Comparaison d'images

Pour pouvoir comparer des signatures `ORDImageSignature` propose deux méthodes : `evaluateScore()` et `isSimilar()`.

*`evaluateScore()` et `isSimilar()` sont des méthodes dites "statique". Ce qui signifie qu'elles sont toujours appelées par le type `ORDImageSignature` et non par une instance d'`ORDImageSignature`.*

*Exemples d'appel de la méthode `evaluateScore` : `ORDSYS.ORDImageSignature.evaluateScore(#)`*

Avant de préciser le fonctionnement de ces deux méthodes, il faut que nous expliquions le principe d'un paramètre qu'elles ont en commun : Le paramètre "commande". Il s'agit d'une chaîne de caractères (`VARCHAR2`) permettant d'accorder plus ou moins d'importance à certains critères de recherche, comme la forme, la texture, la couleur et la localisation. Ainsi chaque critère se voit attribuer un coefficient représentant l'importance qu'on lui accorde lors de la comparaison. Lorsque ce coefficient vaut 1.0 cela signifie que les images doivent être complètement identiques sur ce critère, alors qu'un coefficient de 0.0 permet de ne pas prendre en compte ce critère.

Syntaxe du paramètre " commande ":

'critere1="coefficient"[, critere2=" coefficient " ]#'

Exemple :

'color="1.0",texture="0.5",shape="0.7",location="0.2"'

*Vocabulaire : Dans les documentations officielles, le "coefficient" est aussi appelé "poids", et le paramètre "commande" est également appelé "chaînes de valeur d'attributs".*

*La syntaxe présentée ici est la syntaxe SQL de ce paramètre et non la syntaxe Java qui diffère quelque peu.*

Explication des méthodes de comparaison de signature :

**`evaluateScore(signature1, signature2, commande)`** : Permet de réaliser la comparaison de deux signatures (donc de type `ORDImageSignature`), selon les paramètres définis par le paramètre commande (dont nous venons de voir la syntaxe). Cette méthode renvoie un nombre décimal (`FLOAT`) indiquant le résultat de la comparaison sous forme d'un nombre allant de 0.0 à 100.0. Si le score est de 0.0 les images sont parfaitement identiques sur les critères demandés, alors qu'un score de 100.0 indique des images complètement différentes.

**`isSimilar(signature1, signature2, commande, seuil)`** : Comme pour `evaluateScore()` cette méthode réalise la comparaison de deux signatures. Cependant celle-ci permet de définir en plus un seuil de ressemblance. Elle renvoie un entier (`INTEGER`) qui peut valoir 1 si le score obtenu est inférieur au seuil et 0 dans le cas contraire.

## 1.2 - La gestion d'images coté Java

Tout comme pour les gros objets (LOB), Oracle offre la possibilité de manipuler les images en Java grâce à un package : `oracle.ord.im`

### 1.2.1 - Configuration de l'environnement Java

Avant d'utiliser les classes Java permettant de manipuler les images depuis Oracle, il faut configurer l'environnement Java. Pour cela il est nécessaire de s'assurer que trois packages font bien partie du ClassPath du compilateur JAVA :

- 1 Oracle JDBC drivers : Package permettant d'utiliser les classes JDBC spécifique à Oracle. Il est contenu dans le dossier d'Oracle à l'emplacement `%oracle_dir%/jdbc/lib/ojdbc14.jar` (pour Oracle 9i)
- 2 SQLJ run-time library : Package nécessaire au fonctionnement de certaines méthodes appelées par la classe `OrdImage`. Il est contenu dans le dossier d'Oracle à l'emplacement `%oracle_dir%/sqlj/lib/runtime12.jar`
- 3 Oracle interMedia Java Client library : Package contenant les classes d'interMedia Oracle pour la manipulation d'image mais aussi de fichier audio et vidéo. Il est contenu dans le dossier d'Oracle à l'emplacement `%oracle_dir%/ord/jlib/ordim.jar`

*Notez que le driver Oracle JDBC "classes12.jar", est déprécié mais qu'il fonctionne tout de même.*

### 1.2.2 - La classe OrdImage

En Java, c'est la classe `OrdImage` contenue dans le package `oracle.ord.im`, qui permet de contenir le descripteur de l'image. Cette classe va nous permettre d'agir sur la base de données Oracle, comme si nous le faisons en SQL avec le type `ORDImage`. Cependant la classe `OrdImage` dispose de quelques méthodes supplémentaires comme nous allons le voir ci-après.

Etant donné qu'il n'existe aucune version française des références des méthodes de la classe `OrdImage`, et que même la javadoc anglaise n'est pas contenu dans les package `oracle.ord.im`. Je vous propose de donner, dans cette partie de description de la classe `OrdImage`, une référence en français des méthodes de cette classe :

**getBFILE()** : Renvoie un `oracle.sql.BFILE` contenant le fichier image, lorsque celui ci est stocké sous forme de BFILE (attribut `srcType= "FILE"`), ce BFILE porte alors le nom contenu dans l'attribut `srcName` et référence le fichier situé à l'emplacement `srcLocation`.

**getCompressionFormat()** : Renvoie le nom de l'algorithme utilisé pour la compression de l'image.

**getContent()** : Renvoie l'objet `oracle.sql.BLOB` contenu dans l'attribut `localData` de la source (`ORDSource`) de l'`ORDImage`, uniquement lorsque l'attribut `local` vaut 1 (c'est-à-dire lorsque le fichier est interne à la base).

**getContentFormat()** : Renvoie une chaîne de caractère (`String`) contenant le format du fichier de l'image.

**getContentLength()** : Renvoie un entier (`int`) contenant la taille en byte du fichier de l'image.

**getDataInByteArray()** : Renvoie un tableau de donnée (`byte[]`) contenant l'image, lorsque celle-ci est un BLOB.

**getDataInFile(String chemin)** : Récupère le fichier de l'image dans le chemin vers un fichier donné en paramètre, lorsque celle-ci est un BLOB. Cette méthode renvoi un booléen contenant `true` si la récupération est un succès.

*Si une erreur survient, la méthode lance une exception (SQLException ou IOException). Elle ne renvoie donc jamais false.*

**getDataInStream()** : Renvoie un flux d'entrée (InputStream) permettant de lire les données du fichier de l'image depuis le BLOB de la base de données.

**getORADDataFactory()** : Remplaçante de la méthode getFactory(), elle renvoie l'implémentation de la classe OrdImage. Cette méthode est à mettre au paramètre factory de la méthode getORADData() d'une instance d'un OracleResultSet lors de la récupération du descripteur de l'image. Nous verrons plus en détails la récupération d'un descripteur dans la prochaine partie.

**getFormat()** : Renvoie une chaîne de caractère (String) contenant le format du fichier de l'image.

**getHeight()** : Renvoie dans un entier (int), la hauteur en pixels de l'image.

**getMimeType()** : Renvoie une chaîne de caractères (String) contenant le type mime du fichier de l'image.

**getSource()** : Renvoie une chaîne de caractère (String) contenant les informations sur le stockage de l'image, selon la forme : srcType://srcLocation/srcName, lorsque celle-ci est gérée en BFILE.

**getSourceLocation()** : Renvoie une chaîne de caractère contenant l'emplacement du fichier de l'image lorsque celle-ci est contenue dans un BFILE.

**getSourceName()** : Renvoie une chaîne de caractère contenant le nom du fichier de l'image lorsque celle-ci est contenu dans un BFILE.

**getSourceType()** : Renvoie une chaîne de caractère contenant le type de la source du fichier de l'image lorsque celle-ci est contenu dans un BFILE.

**getUpdateTime()** : Renvoie la date de la dernière modification sous forme de timestamp (java.sql.Timestamp).

**getWidth()** : Renvoie dans un entier (int) contenant la largeur en pixels de l'image.

**importData(byte[ ] [ ] ctx)** : Lorsque le fichier image est stocké dans un BFILE, l'appel de cette méthode permet de le transférer localement à la base de données, dans le BLOB. Le paramètre ctx sert de buffer pour le transfert des données. Après transfert, cette méthode appelle automatiquement la méthode setProperties() afin de générer les méta données de l'image.

**importFrom(byte[ ] [ ] ctx, String srcType, String srcLocation, String srcName)** : Importe dans le BLOB de l'image, une image stockée selon le type srcType (FILE,HTTP,#), depuis le dossier logique d'Oracle srcLocation, qui porte le nom srcName. Le paramètre ctx sert de buffer pour le transfert des données. Après transfert, cette méthode appelle automatiquement la méthode setProperties() afin de générer les méta données de l'image.

*Pour les méthodes importData et importFrom. Si le format de l'image est un format inconnu d'Oracle, il faut désactiver l'appel automatique de setProperties(). Pour cela, il est nécessaire d'appeler la méthode setFormat(String), pour spécifier le nom du format du fichier en commençant par " other ".*

**isLocal()** : Renvoie un booléen permettant de savoir si l'image est stockée localement dans le BLOB ou non.

**loadDataFromByteArray(byte[ ] tableau\_de\_donnee)** : Charge les données contenues dans le tableau\_de\_donnee dans le BLOB. Si des données étaient stockées dans le BLOB avant l'appel de cette méthode

elle sont alors écrasées. Renvoie true si les données ont pu être chargées.

**loadDataFromFile(String fichier)** : Charge les données contenues dans fichier dans le BLOB. Si des données étaient stockées dans le BLOB avant l'appel de cette méthode elle sont alors écrasées. Renvoie true si les données ont pu être chargées.

**process(String commande)** : Permet de manipuler l'image contenue dans le BLOB, comme le ferait la méthode process du type ORDImage. La syntaxe de la commande est donc identique à celle de la méthode process du type ORDImage.

**processCopy(String commande, OrdImage destination)** : Copie l'image contenue dans le descripteur appelant cette méthode, dans le BLOB du descripteur de destination en effectuant au passage les transformations demandées par la commande.

**setCompressionFormat(String compressionFormat)** : Change la propriété contenant le format de compression. Elle ne modifie donc en rien l'image elle-même.

**setContentFormat(String contentFormat)** : Change la propriété contenant le format de l'image. Elle ne modifie donc en rien l'image elle-même.

**setContentLength(int contentLength)** : Change la propriété contenant la taille en byte de l'image. Elle ne modifie donc en rien l'image elle-même.

**setFormat(String fileFormat)** : Change la propriété contenant le format de fichier de l'image.

**setHeight(int height)** : Change la propriété contenant la hauteur de l'image.

**setLocal()** : Définit sur true la propriété spécifiant que l'image est stockée localement à la base, dans le BLOB.

**setMimeType(String mimeType)** : Change la propriété contenant le type mime de l'image.

**setProperties()** : Génère automatiquement les propriétés de l'image (méta-données) et les stocks dans l'objet OrdImage.

**setSource(String srcType, String srcLocation, String srcName)** Définit les 3 paramètres, srcType, srcLocation et srcName, lorsque l'image est contenue à l'extérieur de la base.

**setUpdateTime(java.sql.Timestamp timestampActuel)** : Change la propriété contenant la dernière modification de l'image par le timestampActuel (java.sql.Timestamp) donnée en paramètre.

**setWidth(int width)** : Change la propriété contenant la largeur de l'image.

*Les méthodes permettant de changer les propriétés de l'image ne modifient en rien l'image elle-même. Il est préférable de laisser la méthode setProperties() générer les propriétés de l'image plutôt que de définir manuellement toutes ces informations.*

*Toutes les méthodes peuvent lancer des exceptions de type SQLException, voir également des IOException. Nous verrons dans la suite comment gérer cela.*

### 1.2.3 - La classe OrdImageSignature

La classe `OrdImageSignature` du package `oracle.ord.im`, représente une instance du type `ORDSYS.ORDImageSignature` dans une application Java. Tout comme le type `ORDImageSignature` cette classe possède quatre méthodes dont deux statiques.

Ces méthodes sont :

**`evaluateScore(OrdImageSignature signature1, OrdImageSignature signature2, String commande)`** : Permet de comparer deux signatures selon les critères données par la commande. Le principe de cette méthode est exactement la même que celle du type `ORDImageSignature`.

**`generateSignature(OrdImage image)`** : Génère la signature à partir de l'image contenue dans l'`OrdImage` donnée en paramètre.

**`getORADDataFactory()`** : Remplaçante de la méthode `getFactory()`, elle renvoie l'implémentation de la classe `OrdImageSignature`. Cette méthode est à mettre au paramètre `factory` de la méthode `getORADData()` d'une instance d'un `OracleResultSet` lors de la récupération du descripteur d'une signature. Nous verrons plus en détails la récupération d'un descripteur de signature dans la prochaine partie.

**`isSimilar(OrdImageSignature signature1, OrdImageSignature signature2, String commande, float seuil)`** : Comme pour `evaluateScore()` cette méthode réalise la comparaison de deux signatures. Cependant celle-ci permet de définir en plus un seuil de ressemblance. Elle renvoie 1 si le score obtenu est inférieur au seuil et 0 sinon.

*Cette première partie nous a permis de comprendre la manière dont Oracle a choisi d'administrer les images aussi bien au niveau des types SQL que des classes Java. Mais il nous reste à voir cette gestion au niveau de la pratique. Comment peut-on utiliser toutes les méthodes et attributs vus dans cette partie, dans un programme Java ?*

## 2 - Manipulation d'images en Java

Comme nous l'avons vu, Oracle a défini un package Java spécial pour l'interaction avec le package SQL de gestion des images. Cette partie va vous permettre de réaliser les étapes nécessaires à l'ajout, la récupération d'image et des métas données, la génération de la signature et la comparaison d'image dans un programme Java, grâce à plusieurs exemples fonctionnels complets.

*Les exemples exposés dans cette partie ont seulement un but pédagogique. C'est pourquoi ils sont écrits dans un contexte statique, et que les vérifications et clause try {} catch() ont été limitées au strict nécessaire.*

### 2.1 - Importation des packages

Avant de pouvoir utiliser les classes JDBC et Oracle permettant de manipuler les images, il est nécessaire d'importer dans l'application les packages suivants :

- API JDBC de Java : **java.sql.\***
- La gestion des entrées/sorties : **java.io.\***
- Les pilotes Oracle : **oracle.jdbc.\***
- Les types spécifiques d'Oracle : **oracle.sql.\***
- La classe OrdImage : **oracle.ord.im.OrdImage**
- La classe OrdImageSignature : **oracle.ord.im.OrdImageSignature**

### 2.2 - Ajout d'un ORDImage

Nous allons maintenant voir étape par étape, l'ajout d'un ORDImage dans une base de données Oracle. Par ajout d'un ORDImage nous entendons l'ajout d'une image et la génération de ses métas données dans un attribut de type ORDImage. On considère qu'une insertion a déjà été réalisée dans la table de test appelée "MaTable" table avec l'attribut "id" de l'image fixé à 1 et qu'il ne reste plus qu'à définir l'attribut ORDImage.

#### 2.2.1 - Etape 1 : Connexion à la base de données Oracle

Il faut dans un premier temps enregistrer le pilote Oracle grâce à la méthode statique `registerDriver(OracleDriver)`, de la classe `DriverManager`. Ensuite, il faut créer une instance d'un objet `Connection`, grâce à la méthode statique `getConnection()` de la classe `DriverManager`. La méthode `getConnection()` est composée de trois surdéfinitions prenant en compte des paramètres différents à chaque fois. Nous utiliserons la surdéfinition qui prend en compte 3 paramètres : l'URL de la connexion, le nom d'utilisateur (login), et le mot de passe (password).

Voir aussi la FAQ Java : [Comment charger un driver ?](#) et [Comment ouvrir une connexion à une base de données \(DriverManager\) ?](#)

##### Connexion à la base de données Oracle

```
// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion avec la BD
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe
```

#### 2.2.2 - Etape 2 : Désactiver l'autocommit

On utilise pour cela la méthode `setAutoCommit(boolean)` dans la classe `Connection`.

#### Désactivation de l'autocommit

```
// Mise à false de l'autocommit
conn.setAutoCommit(false);
```

### 2.2.3 - Etape 3 : Initialiser l'attribut ORDImage

L'initialisation se fait par une mise à jour de la base de données grâce à l'instruction `UPDATE` et la méthode `init()` du type `ORDImage`. Pour exécuter une requête simple en Java, utiliser l'interface `Statement`.

#### Initialisation de l'attribut ORDImage

```
// Ecriture de la requête SQL
String sql = "UPDATE MaTable SET image=ORDSYS.ORDImage.init() WHERE id=1";

// Création d'une instance d'un Objet Statement
Statement stmt = conn.createStatement();

// Execution de la requête
stmt.execute(sql);
```

*Cette étape est facultative, si l'utilisateur a déjà initialisé l'attribut `ORDImage` lors de l'insertion dans la table.*

*Si on veut gérer l'image depuis un `BFILE` il faut utiliser le constructeur `init()` avec paramètres. Exemple de la requête SQL pour un fichier externe de nom "img.jpg" stocké sur la machine locale dans le dossier logique d'Oracle de nom "PHOTOS" : `String sql = "UPDATE maTable SET image=ORDSYS.ORDImage.init('file', 'PHOTOS', 'img.jpg')";`*

### 2.2.4 - Etape 4 : Récupérer le descripteur

Il est nécessaire de créer une requête SQL de type `SELECT` devant renvoyer le type `ORDImage` désiré. Puis d'exécuter alors la requête par l'instruction `executeQuery` de la classe `Statement` qui renvoi un `ResultSet`. Comme il s'agit d'un descripteur du type `ORDImage` il est nécessaire d'utiliser la classe `OracleResultSet`. Enfin, il faut utiliser la méthode `getORAData()` de la classe `OracleResultSet` pour récupérer le descripteur dans une instance de la classe `OrdImage`.

#### Récupération du descripteur

```
// Ecriture de la requête SQL
String sql2 = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";

// Execution de la requête et récupération du résultat
OracleResultSet rset=(OracleResultSet) stmt.executeQuery(sql2);

// S'il y a un résultat
if( rset.next())

// Récupération du descripteur d'OrdImage
OrdImage imgObj = (OrdImage) rset.getORAData(1, OrdImage.getORADataFactory() );
```

### 2.2.5 - Etape 5 : Envoyer l'image

Il existe plusieurs manières d'envoyer l'image dans le descripteur. Nous utiliserons la plus simple qui consiste à appeler la méthode `loadDataFromFile(String)` de la classe `OrdImage`. Une autre solution consiste à utiliser un flux d'entrée (`InputStream`) sur le fichier contenant l'image et d'utiliser la méthode `loadDataFromInputStream(InputStream)`.

### Envoi de l'image

```
// Création d'un bloc try{}catch pour l'exception d'entrée/sortie
try{
    // Envoi de l'image dans l'attribut localData du type ORDImage
    imgObj.loadDataFromFile("c:\\image.jpg");
}
catch (IOException e){ e.printStackTrace(); }
```

La méthode `loadDataFromFile` peut lancer une exception de type `IOException`, d'où l'utilisation du bloc `try{}catch()` gérant cette exception.

## 2.2.6 - Etape 6 : Générer les propriétés de l'image et mettre à jours la base

Pour générer les propriétés de l'image il faut appeler la méthode `setProperties()` de la classe `OrdImage`. Ensuite, la méthode `checkProperties()` permet de vérifier si les propriétés générées sont correctes. Pour mettre à jour la base de données, on utilise la classe `OraclePreparedStatement` qui permet de fixer le descripteur grâce à la méthode `setORAData()`.

### Génération des propriétés de l'image

```
// Génération des métas données (propriétés de l'image)
imgObj.setProperties();

// Vérification de la génération des propriétés
if(imgObj.checkProperties()) {

// Ecriture de la requête SQL pour mettre à jour l'attribut
String sql3 = "UPDATE maTable SET image=? WHERE id=1";

// Création d'une instance de l'objet OraclePreparedStatement
OraclePreparedStatement pstmt = (OraclePreparedStatement) conn.prepareStatement(sql3);

// Ajout de l'instance d'OrdImage dans la requête
pstmt.setORAData(1, imgObj);

// Execution de la requête
pstmt.execute();
// Fermeture
pstmt.close();
```

## 2.2.7 - Etape 7 : Validation manuel des données

Comme on a désactivé l'autocommit, il faut valider manuellement les données dans la base. Pour cela on utilise la méthode `commit()` de la classe `Connection`.

```
// Validation manuel dans la base
conn.commit();
```

## 2.2.8 - Etape 8 : Fermeture des connexions et remise à true de l'autocommit.

### Fermeture des connexions et remise à true de l'autocommit

```
// Fermeture des connexions et remise à true
stmt.close();

// Remise à true de l'auto commit
conn.setAutoCommit(true);

// fermeture de la connexion
conn.close();
```

Toutes les méthodes interagissant avec une base de donnée peuvent lever une exception de type

*SQLException. Il est donc nécessaire d'encadrer le code par un bloc try{ }catch(SQLException err) afin de gérer l'exception en Java.*

*Il n'est pas nécessaire ici de fermer l'OracleResultSet car la fermeture du Statement entraîne automatiquement la fermeture des connexions qu'il avait engendré.*

## 2.2.9 - Exemple complet

### Ajout d'une image dans une base ORACLE

```
// Importation des packages importants
import java.sql.*; // Pour la connexion avec Oracle
import java.io.*; // Pour les entrée sorties
import oracle.jdbc.*; // Pour les pilotes Oracle
import oracle.sql.*; // Pour les spécificités SQL d'Oracle
import oracle.ord.im.OrdImage; // Pour la classe OrdImage
import oracle.ord.im.OrdImageSignature; // Pour la classe OrdImageSignature

public class Ajout_OrdImage {

    public static void main(String[] args) {

        try {

// Etape 1 : Création de la connexion avec la base

// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion avec la BD
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe

// Etape 2 : Mise à false de l'autocommit
conn.setAutoCommit(false);

// Etape 3 : Initialisation de l'attribut ORDImage

// Ecriture de la requête SQL
String sql = "UPDATE MaTable SET image=ORDSYS.OrdImage.init() WHERE id=1";

// Création d'une instance d'un Objet Statement
Statement stmt = conn.createStatement();

// Execution de la requête
stmt.execute(sql);

// Etape 4 : Récupération du descripteur

// Ecriture de la requête SQL
String sql2 = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";

// Execution de la requête et récupération du résultat
OracleResultSet rset=(OracleResultSet) stmt.executeQuery(sql2);

// S'il y a un résultat
if( rset.next()) {

// Récupération du descripteur d'OrdImage
OrdImage imgObj = (OrdImage) rset.getORADData(1, OrdImage.getORADDataFactory() );

// Etape 5 : Envoi de l'image

// Création d'un bloc try{ }catch pour l'exception d'entrée/sortie
try{
// Envoi de l'image dans l'attribut localData du type ORDImage
imgObj.loadDataFromFile("c:\\image.jpg");
}

}
```

## Ajout d'une image dans une base ORACLE

```

        catch (IOException e){ e.printStackTrace(); }

// Etape 6 : Génération des métas données
// Génération des métas données (propriétés de l'image)
imgObj.setProperties();

// Vérification de la génération des propriétés
if(imgObj.checkProperties()) {

    // Ecriture de la requête SQL pour mettre à jour l'attribut
    String sql3 = "UPDATE maTable SET image=? WHERE id=1";

    // Création d'une instance de l'objet OraclePreparedStatement
    OraclePreparedStatement pstmt = (OraclePreparedStatement) conn.prepareStatement(sql3);

    // Ajout de l'instance d'OrdImage dans la requête
    pstmt.setORAData(1, imgObj);

    // Execution de la requête
    pstmt.execute();
    // Fermeture
    pstmt.close();

// Etape 7 : Validation manuel de la base
conn.commit();

}
}
// Etape 8 : Fermeture des connexions et remise à true
stmt.close();

// Remise à true de l'auto commit
conn.setAutoCommit(true);

// fermeture de la connexion
conn.close();
} catch (SQLException e) { e.printStackTrace(); }
} // fin

```

## 2.3 - Récupération d'une image et de ses métas données depuis un ORDImage

La récupération d'une image depuis Java se fait de manière très simple. Dans un premier temps on récupère le descripteur de L'ORDImage, puis on charge l'image dans un fichier local par la méthode `getDataInFile(String fichier)` de la classe `OrdImage`. Ensuite il suffit d'afficher les propriétés de l'image une à une. Enfin on ferme les connexions.

## 2.3.1 - Etape 1 : Connexion à la base et désactivation de l'autocommit

## Connexion à la base et désactivation de l'autocommit

```

// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe

// Mise à false de l'autocommit
conn.setAutoCommit(false);

```

## 2.3.2 - Etape 2 : Récupération du descripteur

Il s'agit de la même étape que l'étape 4 de l'ajout d'une image.

#### Récupération du descripteur

```
// Création d'une instance de l'objet Statement
Statement stmt = conn.createStatement();

// Ecriture de la requete SQL pour récupérer l'attribut de type ORDImage
String sql = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";
// Execution de la requête et récupération du résultat
OracleResultSet rset =(OracleResultSet) stmt.executeQuery(sql);

// déclaration d'une instance de l'objet OrdImage
OrdImage imgObj = null;

// S'il y a un résultat
if(rset.next()) {
// Récupération du descripteur
imgObj =(OrdImage) rset.getORADData(1, OrdImage.getORADDataFactory());
}
```

### 2.3.3 - Etape 3 : Récupération de l'image

On utilise la méthode `getDataInFile(String)` de la classe `OrdImage` pour charger l'image dans un fichier local.

#### Récupération de l'image

```
try{
// Récupération de l'image sur le disque local
imgObj.getDataInFile("c:\\recup_image.jpg");
} catch(IOException e) { e.printStackTrace() ;}
```

*La méthode `getDataInFile` peut lancer une exception de type `IOException`, d'où l'utilisation du bloc `try{}catch()` gérant cette exception.*

### 2.3-4 - Etape 4 : Affichage des propriétés de l'image

On appelle la méthode `checkProperties` pour vérifier si les propriétés ont été générées correctement lors de l'ajout de l'`ORDImage` dans la base. Puis on utilise les méthodes permettant de récupérer ces propriétés.

#### Affichage des propriétés de l'image

```
if(imgObj.checkProperties()) {
System.out.println(
"Source : " + imgObj.getSource() +
"Type mime : " + imgObj.getMimeType() +
"Format de fichier : " + imgObj.getFormat() +
"Hauteur : "+ imgObj.getHeight() +
"Largeur : "+ imgObj.getWidth() +
"Poid en bytes : "+ imgObj.getContentLength() +
"Type : "+ imgObj.getContentFormat() +
"Compression : "+ imgObj.getCompressionFormat() );
}
```

### 2.3.5 - Etape 5 : Fermeture des connexions

```
// Fermeture
stmt.close();

// fermeture de la connexion
conn.close();
```

### 2.3.6 - Exemple complet

## Récupération d'une image et de ses métas données

```

// Importation des packages importants
import java.sql.*; // Pour la connexion avec Oracle
import java.io.*; // Pour les entrées sorties
import oracle.jdbc.*; // Pour les pilotes Oracle
import oracle.sql.*; // Pour les spécificités SQL d'Oracle
import oracle.ord.im.OrdImage; // Pour la classe OrdImage
import oracle.ord.im.OrdImageSignature; // Pour la classe OrdImageSignature

public class Recup_OrdImage {
    public static void main(String[] args) {
        try {

// Etape 1 : Création de la connexion avec la base
// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe

// Mise à false de l'autocommit
conn.setAutoCommit(false);

// Etape 2 : Récupération du descripteur

// Création d'une instance de l'objet Statement
Statement stmt = conn.createStatement();

// Ecriture de la requête SQL pour récupérer l'attribut de type ORDImage
String sql = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";
// Exécution de la requête et récupération du résultat
OracleResultSet rset =(OracleResultSet) stmt.executeQuery(sql);

// Déclaration d'une instance de l'objet OrdImage
OrdImage imgObj = null;

// S'il y a un résultat
if(rset.next()) {
// Récupération du descripteur
imgObj =(OrdImage) rset.getORADData(1, OrdImage.getORADDataFactory());

// Etape 3 : Récupération de l'image

try{
// Récupération de l'image sur le disque local
imgObj.getDataInFile("c:\\recup_image.jpg");
} catch(IOException e) { e.printStackTrace() ;}

// Etape 4 : Affichage des propriétés de l'image
if(imgObj.checkProperties()) {
System.out.println(
    "Source : " + imgObj.getSource() +
    "\nType mime : " + imgObj.getMimeType() +
    "\nFormat de fichier : " + imgObj.getFormat() +
    "\nHauteur : " + imgObj.getHeight() +
    "\nLargeur : " + imgObj.getWidth() +
    "\nPoid en bytes : " + imgObj.getContentLength() +
    "\nType : " + imgObj.getContentType() +
    "\nCompression : " + imgObj.getCompressionFormat() ;
) else
System.out.println("Propriété non générées");
} else
System.out.println("pas de résultat");
// Etape 5 : Fermeture des connexions

// Fermeture
stmt.close();

// fermeture de la connexion
conn.close();

} catch(SQLException e) { e.printStackTrace(); }

```

## Récupération d'une image et de ses métas données

```
}
}
```

## 2.4 - Génération de signature

Comme nous l'avons vu dans la partie SQL de ce tutorial, il est nécessaire de générer ce qu'on appelle la signature d'une image pour pouvoir faire des comparaisons d'image sur les caractéristiques visuelles de celles-ci. Il est donc obligatoire de créer un attribut de type `ORDImageSignature` lié à chacune des images que l'on veut comparer.

Voyons pas à pas les étapes de sa génération.

## 2.4.1 - Etape 1 : Connexion à la base et mise à false de l'autocommit

## Connexion à la base et mise à false de l'autocommit

```
// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@pagnol:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe

// Mise à false de l'autocommit
conn.setAutoCommit(false);
```

2.4.2 - Etape 2 : Initialisation de l'attribut `ORDImageSignature`

A l'instar de l'ajout d'un `ORDImage` dans une table, l'attribut doit être initialisé par le constructeur `init()`, avant de pouvoir récupérer un descripteur.

Initialisation de l'attribut `ORDImageSignature`

```
// Création d'une instance de l'objet Statement pour l'exécution de requête SQL
Statement stmt = conn.createStatement();
// Ecriture de la requête SQL permettant d'initialiser l'attribut ORDImageSignature
String sql = "UPDATE maTable SET signature=ORDSYS.ORDImageSignature.init() WHERE id=1";

// Exécution de la requête
stmt.execute(sql);
```

2.4.3 - Etape 3 : Récupération du descripteur de l'`ORDImage`

La méthode `generateSignature` prend en paramètre une instance de la classe `OrdImage` contenant le descripteur de l'image dont on veut générer la signature. On doit donc le récupérer puis générer sa signature.

```
// Ecriture de la requête SQL
String sql2 = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";
// Exécution de la requête et récupération du résultat
OracleResultSet rset =(OracleResultSet) stmt.executeQuery(sql2);

// Déclaration d'une instance de l'objet OrdImage
OrdImage imgObj = null;

// S'il y a un résultat
if(rset.next())
    // Récupération du descripteur
    imgObj =(OrdImage) rset.getORAData(1, OrdImage.getORADataFactory());
```

## 2.4.4 - Etape 4 : Récupération du descripteur de l'ORDImageSignature

On récupère le descripteur, toujours à l'aide de la méthode `getORADDataFactory()`, mais de la classe `OrdImageSignature` lorsqu'on veut récupérer le descripteur d'une signature.

### Récupération du descripteur de l'ORDImageSignature

```
// Ecriture de la requête SQL
String sql3 = "SELECT signature FROM maTable WHERE id=1 FOR UPDATE";

// Exécution de la requête et récupération du résultat
OracleResultSet rset2 = (OracleResultSet) stmt.executeQuery(sql3);

// Déclaration d'une instance de l'objet OrdImageSignature
OrdImageSignature imgSig = null;

// S'il y a un résultat
if(rset2.next()) {
// Récupération du descripteur
imgSig = (OrdImageSignature) rset2.getORADData(1, OrdImageSignature.getORADDataFactory());
}
```

## 2.4.5 - Etape 5 : Génération de la signature

On génère la signature simplement en appelant la méthode `generateSignature` de la classe `OrdImageSignature`, et en donnant le descripteur de l'`OrdImage` en paramètre.

```
// Utilisation de la méthode generateSignature de l'objet OrdImageSignature et
// passage en paramètre du descripteur de l'ORDImage
imgSig.generateSignature(imgObj);
```

## 2.4.6 - Etape 6 : Mise à jours de la signature dans la base

Pour mettre à jour la signature générée dans la base de données, on utilise la méthode `setORADData()` avec le descripteur de la signature en paramètre, afin de mettre l'attribut.

### Mise à jours de la signature dans la base

```
// Ecriture de la requête SQL
String sql4 = "UPDATE MaTable SET signature = ? WHERE id=1";

// Préparation de la requête
OraclePreparedStatement update_sig = (OraclePreparedStatement)
conn.prepareStatement(sql4);

// Ajout du descripteur de la signature
update_sig.setORADData(1, imgSig);
// Exécution de la requête
update_sig.execute();
// Fermeture
update_sig.close();
```

## 2.4.7 - Etape 7 : Validation des modifications

### Validation des modifications

```
// Validation manuel
conn.commit();
```

## 2.4.8 - Etape 8 : Remise à true de l'autocommit et fermeture des flux

## Remise à true de l'autocommit et fermeture

```
// Remise à true de l'autocommit
conn.setAutoCommit(true);

// Fermetures
stmt.close();
conn.close();
```

## 2.4-9 - Exemple complet

## Génération des signatures

```
// Importation des packages importants
import java.sql.*; // Pour la connexion avec Oracle
import java.io.*; // Pour les entrées sorties
import oracle.jdbc.*; // Pour les pilotes Oracle
import oracle.sql.*; // Pour les spécificités SQL d'Oracle
import oracle.ord.im.OrdImage; // Pour la classe OrdImage
import oracle.ord.im.OrdImageSignature; // Pour la classe OrdImageSignature

public class GenererSignature {

    public static void main(String[] args) {

        try {

// Etape 1 : Connexion à la base et mise à false de l'autocommit

            // Enregistrement du pilote Oracle
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

            // Ouverture de la connexion
            Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
                "benoit", // utilisateur
                "secret"); // mot de passe

            // Mise à false de l'auto commit
            conn.setAutoCommit(false);

// Etape 2 : Initialisation de l'attribut OrdImageSignature
// Création d'une instance de l'objet Statement pour l'exécution de requête SQL
            Statement stmt = conn.createStatement();
// Ecriture de la requête SQL permettant d'initialiser l'attribut OrdImageSignature
            String sql = "UPDATE maTable SET signature=ORDSYS.OrdImageSignature.init() WHERE id=1";

            // Exécution de la requête
            stmt.execute(sql);

// Etape 3 : Récupération du descripteur de l'OrdImage

            // Ecriture de la requête SQL
            String sql2 = "SELECT image FROM MaTable WHERE id=1 FOR UPDATE";
            // Exécution de la requête et récupération du résultat
            OracleResultSet rset = (OracleResultSet) stmt.executeQuery(sql2);

            // Déclaration d'une instance de l'objet OrdImage
            OrdImage imgObj = null;

            // S'il y a un résultat
            if(rset.next()) {
                // Récupération du descripteur
                imgObj = (OrdImage) rset.getORAData(1, OrdImage.getORADataFactory());
            }

// Etape 4 : Récupération du descripteur de l'OrdImageSignature

            // Ecriture de la requête SQL
            String sql3 = "SELECT signature FROM maTable WHERE id=1 FOR UPDATE";

            // Exécution de la requête et récupération du résultat
            OracleResultSet rset2 = (OracleResultSet) stmt.executeQuery(sql3);
```

### Génération des signatures

```

// Déclaration d'une instance de l'objet OrdImageSignature
OrdImageSignature imgSig = null;

// S'il y a un résultat
if(rset2.next()) {
// Récupération du descripteur
imgSig = (OrdImageSignature) rset2.getORADData(1, OrdImageSignature.getORADDataFactory());

// Etape 5 : Génération de la signature

// Utilisation de la méthode generateSignature de l'objet OrdImageSignature et //passage en
paramètre du descripteur de l'ORDImage
    imgSig.generateSignature(imgObj);

// Etape 6 : Mise à jours de la signature dans la base

// Ecriture de la requête SQL
String sql4 = "UPDATE MaTable SET signature = ? WHERE id=1";

// Préparation de la requête
OraclePreparedStatement update_sig = (OraclePreparedStatement)
    conn.prepareStatement(sql4);

// Ajout du descripteur de la signature
update_sig.setORADData(1, imgSig);
// Execution de la requête
update_sig.execute();
// Fermeture
update_sig.close();

// Etape 7 : Validation des modifications
    System.out.println("Signature généré");
    // Validation manuel
    conn.commit();
}

// Etape 8 : Remise à true de l'autocommit et fermeture des flux
// Remise à true de l'autocommit
conn.setAutoCommit(true);

// Fermetures
stmt.close();
conn.close();

} catch (SQLException e) { e.printStackTrace(); }
}

```

## 2.5 - Comparaison d'image

Pour comparer deux images, il suffit de récupérer les descripteurs des signatures des deux images, puis d'appeler la méthode statique `evaluateScore()` ou `isSimilar()` de la classe `OrdImageSignature`.

### 2.5.1 - Etape 1 : Se connecter à la base de données et désactiver l'autocommit

#### Connexion à la base de donnée et désactivation de l'autocommit

```

// Enregistrement du pilote Oracle
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

// Ouverture de la connexion
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@host:1521:nomdb", // url de la base
    "benoit", // utilisateur
    "secret"); // mot de passe

// Mise à false de l'autocommit

```

## Connexion à la base de donnée et désactivation de l'autocommit

```
conn.setAutoCommit(false);
```

## 2.5.2 - Etape 2 : Récupérer les descripteurs des signatures des deux images à comparer

Le principe est le même que pour l'étape 4 de la génération de la signature.

## Récupération des descripteurs des signatures des deux images

```
// Création d'une instance de l'objet Statement pour l'exécution de requête
Statement stmt = conn.createStatement();

// Ecriture de la requête de récupération des signatures des 2 images
String sql = "SELECT signature FROM maTable WHERE id BETWEEN 1 AND 2 FOR UPDATE";

// Exécution et récupération du résultat e la requête
OracleResultSet rset = (OracleResultSet) stmt.executeQuery(sql);
// Déclaration des instances de l'objet OrdImageSignature
OrdImageSignature signature1 = null;
OrdImageSignature signature2 = null;

// s'il y a un résultat
if(rset.next()) {
// Récupération du descripteur de l'OrdImageSignature de l'image 1
signature1 = (OrdImageSignature) rset.getORADData(1,
OrdImageSignature.getORADDataFactory());

// s'il y a un 2ieme résultat
if(rset.next())

// Récupération du descripteur de l'OrdImageSignature de l'image 2
signature2 = (OrdImageSignature) rset.getORADData(1,
OrdImageSignature.getORADDataFactory());
```

## 2.5.3 - Etape 3 : Comparer des signatures

Comme nous venons de récupérer les descripteurs des signatures des deux images, il ne reste plus qu'à écrire la commande définissant les coefficients des critères de comparaison. Nous pouvons alors appeler l'une des méthodes statique `evaluateScore()` ou `isSimilar()` de la classe `OrdImageSignature`, pour comparer les signatures.

## Comparaison des signatures

```
// Ecriture de la commande définissant les coef. des critères de // comparaisons
String commande = "color=1 texture=0 shape=0 location=0";

// Comparaison par évaluation du score
float f = OrdImageSignature.evaluateScore(signature1, signature2,commande);

// Définition du seuil
float seuil = 20;

// Comparaison par la méthode isSimilar()
int similaire = OrdImageSignature.isSimilar(signature1,signature2, commande,seuil);
```

## 2.5.4 - Exemple complet

## Comparaison d'image

```
// Importation des packages importants
import java.sql.*; // Pour la connexion avec Oracle
import java.io.*; // Pour les entrée sorties
import oracle.jdbc.*; // Pour les pilotes Oracle
import oracle.sql.*; // Pour les spécificité SQL d'Oracle
import oracle.ord.im.OrdImage; // Pour la classe OrdImage
import oracle.ord.im.OrdImageSignature; // Pour la classe OrdImageSignature
```

## Comparaison d'image

```

public class Comparer_image {

    public static void main(String[] args) {

        try {
// Etape 1 : Connexion à la base et désactivation de l'autocommit
// Enregistrement du pilote Oracle
        DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

            // Ouverture de la connexion
            Connection conn = DriverManager.getConnection(
                "jdbc:oracle:thin:@pagnol:1521:m205", // url de la base
                "benoit", // utilisateur
                "secret"); // mot de passe

            // Mise à false de l'autocommit
            conn.setAutoCommit(false);

// Etape 2 : Récupération des descripteurs de signatures des deux images

            // Création d'une instance de l'objet Statement pour l'exécution de requête
            Statement stmt = conn.createStatement();

// Ecriture de la requête de récupération des signatures des 2 images
String sql = "SELECT signature FROM maTable WHERE id BETWEEN 1 AND 2 FOR UPDATE";

// Execution et récupération du résultat e la requête
OracleResultSet rset = (OracleResultSet) stmt.executeQuery(sql);
// Déclaration des instances de l'objet OrdImageSignature
OrdImageSignature signature1 = null;
OrdImageSignature signature2 = null;

            // s'il y a un résultat
            if(rset.next()) {
                // Récupération du descripteur de l'OrdImageSignature de l'image 1
                signature1 = (OrdImageSignature) rset.getORADData(1,
OrdImageSignature.getORADDataFactory());

                // s'il y a un 2ieme résultat
                if(rset.next()) {

                    // Récupération du descripteur de l'OrdImageSignature de l'image 2
                    signature2 = (OrdImageSignature) rset.getORADData(1,
OrdImageSignature.getORADDataFactory());

// Etape 3 : Comparaison des signatures
                // Ecriture de la commande définissant les coef. des critères de // comparaisons
                String commande = "color=1 texture=0 shape=0 location=0";

                // Comparaison par évaluation du score
                float f = OrdImageSignature.evaluateScore(signature1, signature2,commande);
// Affichage de la comparaison
                System.out.println("Resultat de la comparaison : "+f);

                // Définition du seuil
                float seuil = 20;

                // Comparaison par la méthode isSimilar()
                int similaire = OrdImageSignature.isSimilar(signature1,signature2,
commande,seuil);

                // Affichage de la comparaison
                System.out.println("image 1 similaire à l'image 2 ? "+similaire);
            }
        }

// Etape 4 : Fermeture
// Fermeture
        stmt.close();
        conn.close();
    } catch (SQLException e) { e.printStackTrace() ;}
}

```

Il existe beaucoup d'autre manières pour implémenter les exemples fonctionnels de ce tutorial. J'ai seulement donné ici l'implémentation qui me semble la plus claire et la plus simple, en utilisant les méthodes conseillées par les documentations officielles.

### 3 - Conclusion

Ce tutorial vous a donné toutes les clés nécessaires à la conception de programme Java pour manipuler des images depuis une base de donnée Oracle. Pour concevoir ce genre de programme vous avez pu remarquer qu'il faut une méthodologie rigoureuse ainsi qu'une bonne compréhension des mécanismes mis en #uvre par Oracle. C'est pourquoi j'ai pris beaucoup de temps pour définir ces mécanismes dans la première partie.

## 4 - Téléchargement

Ce tutorial est également disponible en version PDF : [FTP](#) et [HTTP](#)

Et en version ZIP : [FTP](#) et [HTTP](#)