Université du Québec
École de technologie supérieure

Relationnel-objet en Oracle

Survol du cours

- Relationel-objet en Oracle
  - Le paquetage DBMS_LOB et la manipulation de LOBs (4.3.1)
- PL/SQL, fonctions et procédures (4.2)
- Interfaces entre SQL et une application
  - API (JDBC) (livre de Godin)
  - Java enchassé (SQLJ) (livre de Godin)

Historique – Relationnel-objet

- Modèle conceptuel versus modèle entité-relation
- Ajout de méthodes et type défini (custom)
- Exemples de SGBD relationnel-objet
  - Illustra (IBM)
  - Omniscience (Oracle)
  - PostgreSQL (libre)

Oracle Multimédia (anciennement interMedia)

- Depuis 11g

Création d'un nouveau type adresse

```
CREATE OR REPLACE TYPE adresse_t AS OBJECT
(
norue                   INTEGER,
rue                     VARCHAR2(50),
cite                    VARCHAR2(30),
ville                   VARCHAR2(30),
province                VARCHAR2(30),
code_postal             CHAR(6)
);
```

Création d'un nouveau type Personne (avec une adresse)

```
CREATE OR REPLACE TYPE person_t AS OBJECT
(id                 INTEGER,
nom                 VARCHAR2(30),
prenom              VARCHAR2(30),
date_nais           DATE,
sexe                CHAR(1),
tel                 INTEGER,
mail                VARCHAR2(30),
adresse             adresse_t) NOT FINAL;
```

Création d'un type MM et une table associée

```
CREATE OR REPLACE TYPE multimedia_t AS OBJECT
(path_name  VARCHAR2(500)) NOT FINAL;

CREATE OR REPLACE TYPE multimedia_table AS TABLE
   OF multimedia_t;
```

Création d'un type pour le MM

```
#---audio
CREATE OR REPLACE TYPE codec_t AS
   OBJECT
(id      INTEGER,
 type    CHAR(1),
 nom     VARCHAR2(255));

CREATE OR REPLACE TYPE audio_t
   UNDER multimedia_t
(num_audio        INTEGER,
freq_echant       FLOAT,
bit_echant        INTEGER,
nb_ca             INTEGER,
audio             BLOB,
codec             REF codec_t,
duree             FLOAT(126))
FINAL;
```

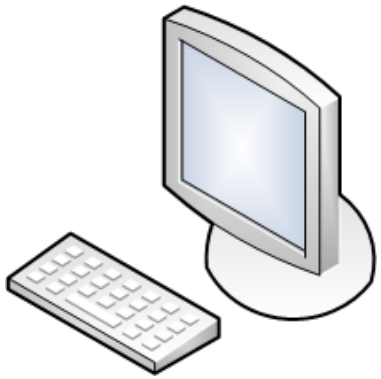
```
#---video
CREATE OR REPLACE TYPE video_t
   under multimedia_t
(num_video        INTEGER,
taille            FLOAT,
nb_bits_p         INTEGER,
format            VARCHAR2(20),
video             BLOB,
codec             REF codec_t,
duree             FLOAT(126),
freq_echant       FLOAT(126))
   FINAL;
```

Université du Québec
École de technologie supérieure

Interface entre SQL et une application

JDBC versus SQLJ

Schéma d'une application multimédia



Interface utilisateur

Connecteur SGBD

Base de données Oracle 11g

JDBC ODBC SQLJ

Connexion au SGBD

- API spécifique au SGBD commerciale
  - ex. Oracle Call Interface (OCI)
  - non portable
- API normalisée par l'industrie
  - standard de facto ODBC
  - développé par Microsoft pour le C
  - pilote ODBC pour client/serveur

Connexion au SGBD

- SQL/CLI de SQL:1999 inspirée de ODBC
  - JDBC pour Java
  - Norme ISO/IEC 9075-3:2003.

SQL enchassé (Embedded SQL)

- Code SQL dans le source du langage hôte
- Pré-compilation
  - Oracle : pro*C/C++, pro*COBOL, …, SQLJ
- Moins portable
  - pré-compilateur spécifique au SGBD
    - traduit en API du SGBD
    - SQLJ (partie 0) traduit en API standard JDBC

Qu'est-ce que SQL-PSM ?

- Extension procédurale à SQL
- Inclut les notions de :
  - variables
  - structures de contrôle (if, for, while)
- PL/SQL est le nom commercial du SQL-PSM pour Oracle, mais l'équivalent existe pour d'autres SGDB (Sybase, DB2 …)
- Norme ISO existe : SQL-PSM (Persistent Stored Module) (SQL3)

Qu'est-ce que SQL-PSM ?

- Permet de définir :
  - des fonctions utilisateur;
  - des procédures stockées (procédures déclenchées à la demande);
  - des déclencheurs *"triggers"* (procédures déclenchées automatiquement) pour implémenter
    - des contraintes d'intégrité complexes;
    - des mises à jour de champs redondants.
  - des paquetages.

Fonction dans PL/SQL

- Une fonction définit un calcul plus ou moins complexe;
  - Elle est compilée;
  - Elle est appelée avec des paramètres;
  - Elle retourne une valeur (numérique ou alphanumérique);
- Elle s'intègre aux requêtes SQL comme n'importe quelle fonction incluse par défaut avec le SGBD (Oracle, DB2…).

Exemple de fonction

```
CREATE OR REPLACE FUNCTION carre (x NUMBER)
   RETURN NUMBER
   AS result NUMBER;
BEGIN
   result := x*x;
   RETURN result;
END;
/

SELECT carre(1)
FROM DUAL
/
```

Déclaration de paramètres et variables PL/SQL (De Robert Godin)

```
leNoClient          Client.noClient%TYPE;

leNoClient          INTEGER;
```

Affectation en PL/SQL (De Robert Godin)

```
laQuantitéEnAttente:=
                    laQuantitéCommandée    -
                    laQuantitéLivrée;
```

Transfert d'une valeur de colonne d'un SELECT dans une variable (clause INTO) (De Robert Godin)

```
SELECT      noClient, dateCommande
INTO        leNoClient, laDateCommande
FROM        Commande
WHERE       noCommande = leNoCommande;
```

Commande

noCommande	dateCommande	noClient
50	09-Mai-2005	64598
51	10-Mai-2005	24538

Structures de contrôle IF de programme PL/SQL (De Robert Godin)

- IF <condition> THEN
  <commandes>;
  END IF;

- IF <condition> THEN
  <commandes>;
  ELSE
  <commandes>;
  END IF;

- IF <condition> THEN
  <commandes>;
  ELSIF <condition> THEN
  <commandes>;
  …
  ELSIF <condition> THEN
  <commandes>;
  ELSE
  <commandes>;
  END IF;

Structures de contrôle CASE de programme PL/SQL (De Robert Godin)

```
CASE /* variable */
WHEN /* valeur 1 */ THEN
        /* instructions 1 */
WHEN /* valeur 2 */ THEN
        /* instructions 2 */
...
WHEN /* valeur n */ THEN
        /* instructions n */
ELSE
        /* instructions par défaut */
END CASE;
```

Structures de contrôle boucles de programme PL/SQL (De Robert Godin)

- LOOP
  <commandes>
  EXIT WHEN <condition>;
  END LOOP;
- FOR i IN 1..10 LOOP
  <commandes>
  END LOOP;
- WHILE <condition> LOOP
  <commandes>
  END LOOP;

Fonction PL/SQL stockées (De Robert Godin)

```
SQL> CREATE OR REPLACE FUNCTION fQuantitéEnStock
  2  (unNoArticle  Article.noArticle%TYPE)
  3  RETURN Article.quantitéEnStock%TYPE IS
  4
  5    uneQuantitéEnStock  Article.quantitéEnStock%TYPE;
  6  BEGIN
  7   SELECT  quantitéEnStock
  8   INTO   uneQuantitéEnStock
  9   FROM   Article
 10   WHERE   noArticle = unNoArticle;
 11   RETURN uneQuantitéEnStock;
 12  END fQuantitéEnStock;
 13
 14  /

Function created.
```

Article

noArticle	uneQuantitéEnStock
10	20
12	100

```
SQL> select fQuantitéEnStock(10) from dual;

FQUANTITÉENSTOCK(10)
--------------------
                  20
```

Curseur PL/SQL (CURSOR) (De Robert Godin)

```
CURSOR lignesCommande(unNoCommande Commande.noCommande%TYPE)IS
      SELECT      noArticle, quantité
      FROM        LigneCommande
      WHERE       LigneCommande.noCommande = unNoCommande ;
```

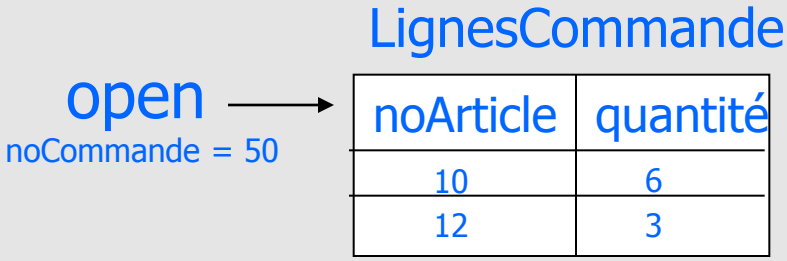
```
OPEN lignesCommande(leNoCommande);
-- Le OPEN ouvre le CURSOR en lui passant les paramètres

LOOP
    FETCH lignesCommande INTO leNoArticle, laQuantitéCommandée;
    -- Le FETCH retourne la ligne suivante

    EXIT WHEN lignesCommande%NOTFOUND;
    -- %NOTFOUND est un attribut du CURSOR qui permet de déterminer
    -- si le FETCH a atteint la fin de la table
...

END LOOP;

-- Le CLOSE ferme le CURSOR
CLOSE lignesCommande;
```



Déboguage du code PL/SQL

(De Robert Godin)

- SHOW ERRORS sous SQL*PLUS pour voir les erreurs de compilation
- Table USER_SOURCE dans la métabase

```
SQL> SELECT  text
  2  FROM   USER_SOURCE
  3  WHERE  name = 'FQUANTITÉENSTOCK' AND type = 'FUNCTION'
  4  ORDER BY  line;

TEXT
--------------------------------------------------------------------------------
FUNCTION fQuantitéEnStock
(unNoArticle  Article.noArticle%TYPE)
RETURN Article. quantitéEnStock%TYPE IS

  uneQuantitéEnStock  Article.quantitéEnStock%TYPE;
BEGIN
 SELECT  quantitéEnStock
 INTO   uneQuantitéEnStock
 FROM   Article
 WHERE   noArticle = unNoArticle;
 RETURN uneQuantitéEnStock;

TEXT
--------------------------------------------------------------------------------
END fQuantitéEnStock;


13 rows selected.
```

- Package DBMS_OUTPUT (Put, Put_line, new_line)

Trace dans une Boucle FOR PL/SQL

(De Robert Godin)

```
FOR uneLigne IN lignesCommande(leNoCommande) LOOP

      DBMS_OUTPUT.PUT('noArticle :');
      DBMS_OUTPUT.PUT(uneLigne.noArticle);
      DBMS_OUTPUT.PUT(' quantité commandée:');
      DBMS_OUTPUT.PUT(uneLigne.quantité);

      -- Chercher la quantité déjà livrée
      SELECT      SUM(quantitéLivrée)
      INTO        laQuantitéLivrée
      FROM        DétailLivraison
      WHERE       noArticle = uneLigne.noArticle AND
                  noCommande = leNoCommande ;

      IF (laQuantitéLivrée IS NULL) THEN
            DBMS_OUTPUT.PUT_LINE(' livraison en attente');
      ELSE
            laQuantitéEnAttente:= uneLigne.quantité -laQuantitéLivrée;
            IF (laQuantitéEnAttente = 0) THEN
                  DBMS_OUTPUT.PUT_LINE(' livraison complétée');
            ELSE
                  DBMS_OUTPUT.PUT (' quantité en attente :');
                  DBMS_OUTPUT.PUT_LINE(laQuantitéEnAttente);
            END IF ;
      END IF ;
END LOOP;
```

Bloc PL/SQL (De Robert Godin)

```
[DECLARE
      déclaration [déclaration] ...]

BEGIN
      séquenceEnoncés
[EXCEPTION
      exception_énoncé [exception_énoncé] ...]
END
```

Taper dans SQL*plus (De Robert Godin)

```
SQL> DECLARE
  2   laQuantitéEnStock Article.quantitéEnStock%TYPE;
  3  BEGIN
  4   SELECT  quantitéEnStock INTO laQuantitéEnStock
  5   FROM  Article
  6   WHERE  noArticle = 10;
  7   IF laQuantitéEnStock = 0 THEN
  8    DBMS_OUTPUT.PUT_LINE('L article est en rupture de stock');
  9   ELSE
 10    DBMS_OUTPUT.PUT('Quantité en stock :');
 11    DBMS_OUTPUT.PUT_LINE(laQuantitéEnStock);
 12   END IF;
 13  EXCEPTION
 14   WHEN NO_DATA_FOUND THEN
 15    DBMS_OUTPUT.PUT_LINE('Numéro d article inexistant');
 16   WHEN OTHERS THEN
 17    RAISE_APPLICATION_ERROR(-20001,'Erreur soulevée par le SELECT');
 18  END;
 19  /
Quantité en stock :20

PL/SQL procedure successfully completed.
```

Traitement d'exception (De Robert Godin)

- Déclarer

```
nomException EXCEPTION;
```

- Soulever

```
RAISE nomException;
```

- Attraper

```
WHEN nomException THEN
      séquenceÉnoncés;
```

Procédure stockée

- Une procédure stockée définit un calcul plus ou moins complexe;
  - Elle est compilée;
  - Elle est appelée avec des paramètres;
  - Elle ne retourne rien (sauf paramètres OUT);
- Le calcul se déroule entièrement sur le serveur (exécution centralisée);
- Ne nécessite aucun autre transfert de données que celui des paramètres;

Exemple de procédure stockée (De Robert Godin)

```
CREATE OR REPLACE PROCEDURE
  suppr_dept_sans_emp
IS
BEGIN
   DELETE FROM DEPT
   WHERE NOT EXISTS
      (SELECT * FROM EMP
      WHERE EMP.DEPTNO = DEPT.DEPTNO);
END;
/
```

Autre exemple de procédure stockée

(De Robert Godin)

```
CREATE OR REPLACE PROCEDURE stat_dept(dept IN
   INTEGER, nombre OUT INTEGER, salaire_min OUT
   INTEGER, salaire_max OUT INTEGER)
IS
BEGIN
    SELECT COUNT(*), MIN(sal), MAX(sal)
    INTO nombre, salaire_min, salaire_max
    FROM EMP
    WHERE DEPTNO = dept;
END;
/
```

Utile à l'intérieur d'un bloc PL/SQL pour récupérer les valeurs des variables OUT ou IN OUT

Exemple avec paramètre OUT

```
CREATE OR REPLACE PROCEDURE pNouveauNom
(unNoMus IN integer, leNom OUT varchar) IS
BEGIN
  SELECT nomMusicien INTO leNom FROM  Musicien
  WHERE  Musicien.noMusicien = unNoMus;
  leNom := leNom || '+++';
END pNouveauNom;
/
CREATE OR REPLACE PROCEDURE pModifierNom
(unNoMus IN integer) IS
  nouveauNom  VARCHAR2(20);
BEGIN
  pNouveauNom(unNoMus, nouveauNom);
  UPDATE Musicien SET nomMusicien = nouveauNom
  WHERE  Musicien.noMusicien = unNoMus;
END pModifierNom;
/
```

Procédure stockée (De Robert Godin)

```
SQL> CREATE PROCEDURE pModifierQuantitéEnStock
  2  (unNoArticle    Article.noArticle%TYPE,
  3  nouvelleQuantitéEnStock Article.quantitéEnStock%TYPE) IS
  4  BEGIN
  5    UPDATE  Article
  6    SET  quantitéEnStock = nouvelleQuantitéEnStock
  7    WHERE  noArticle = unNoArticle;
  8  END pModifierQuantitéEnStock;
  9  /

Procedure created.
```

```
SQL> EXECUTE pModifierQuantitéEnStock(10,20);

PL/SQL procedure successfully completed.

SQL> SELECT * FROM Article WHERE noArticle = 10;

 NOARTICLE DESCRIPTION          PRIXUNITAIRE QUANTITÉENSTOCK
---------- -------------------- ------------ ---------------
        10 Cèdre en boule              10,99              20
```

Compilation et stockage dans le schéma avec SQL*plus (De Robert Godin)

```
SQL> CREATE PROCEDURE pStatutCommande
  2  (leNoCommande Commande.noCommande%TYPE ) IS
  3
  4   -- Déclaration de variables
  5  leNoClient   Client.noClient%TYPE;
  6   laDateCommande  Commande.dateCommande%TYPE ;
  7   leNoArticle   Article.noArticle%TYPE ;
  8  laQuantitéCommandée LigneCommande.quantité%TYPE ;
  9  laQuantitéLivrée  INTEGER;
 10  laQuantitéEnAttente INTEGER;
 11
…
69  EXCEPTION
 70   WHEN NO_DATA_FOUND THEN
 71    DBMS_OUTPUT.PUT_LINE('Numéro de commande inexistant');
 72   WHEN OTHERS THEN
 73    RAISE_APPLICATION_ERROR(-20001,'Exception levée par la procédure');
 74  END pStatutCommande;
 75  /

Procedure created.
```

Appel de la fonction avec SQL*plus (De Robert Godin)

```
SQL> SET SERVEROUTPUT ON
SQL> EXECUTE pStatutCommande(2);
Commande #:2
noArticle :40 quantité :2 livraison complétée
noArticle :95 quantité :3 quantité en attente :2

PL/SQL procedure successfully completed.
SQL> EXECUTE pStatutCommande(5);
Commande #:5
noArticle :10 quantité :5 livraison en attente
noArticle :20 quantité :5 livraison en attente
noArticle :70 quantité :3 quantité en attente :1

PL/SQL procedure successfully completed.
SQL> EXECUTE pStatutCommande(10);
Commande #:10
Numéro de commande inexistant

PL/SQL procedure successfully completed.
```

Résumé: Extension procédurale à SQL (*Persistent Stored Modules* - SQL/PSM)

- Oracle PL/SQL
  - Un langage avec sa structure de contrôle;
  - procédures, fonctions, déclencheurs et packages;
  - support direct des types SQL pour les variables;
  - exécution au niveau serveur de BD (directement dans l'interface SQL*Plus);
  - Stockée dans le schéma relationnel.

JDBC

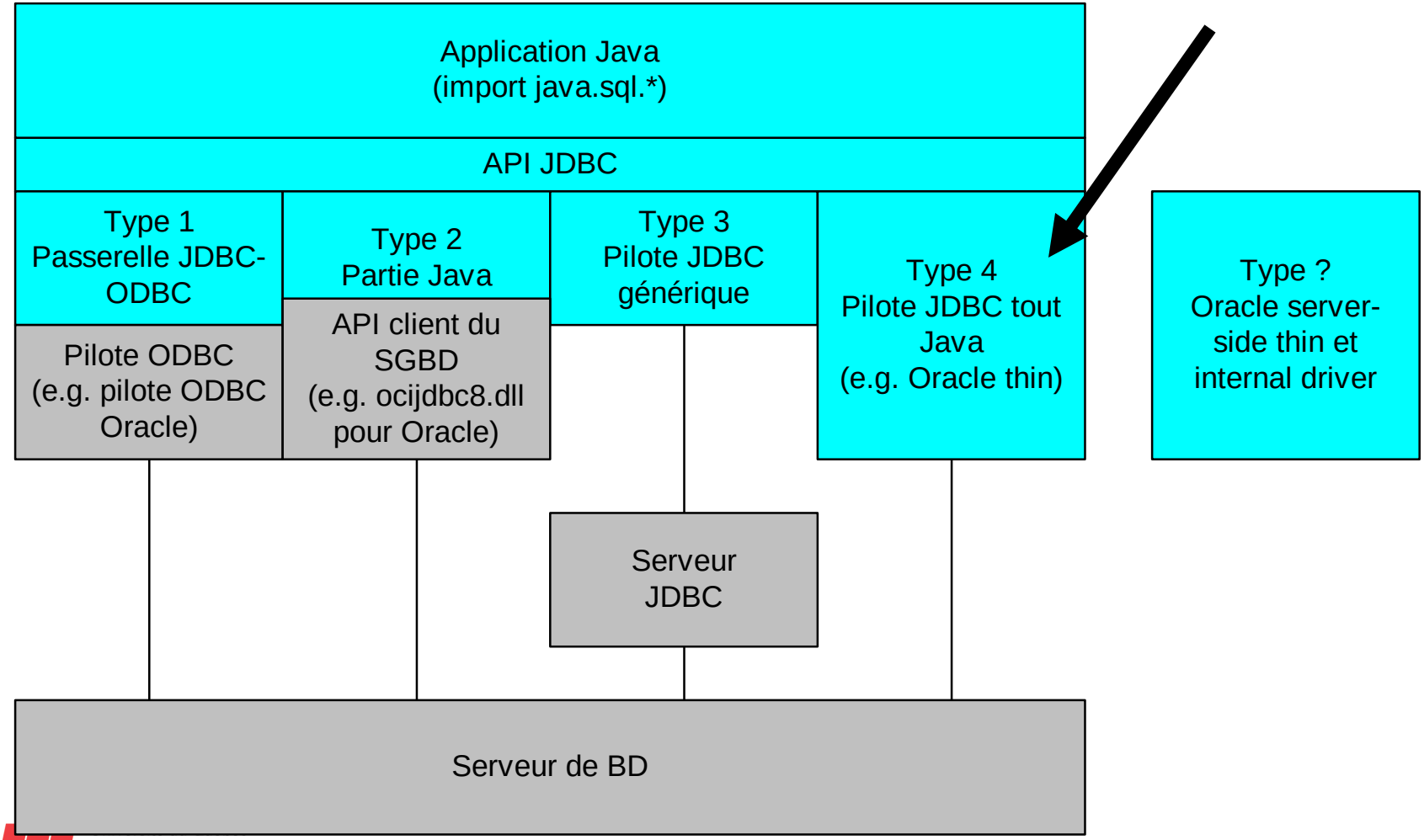
- API normalisée pour JAVA
- Ensemble de classes
- Besoin d'installer un pilote JDBC dans l'environnement JAVA

http://java.sun.com/products/jdbc/index.jsp

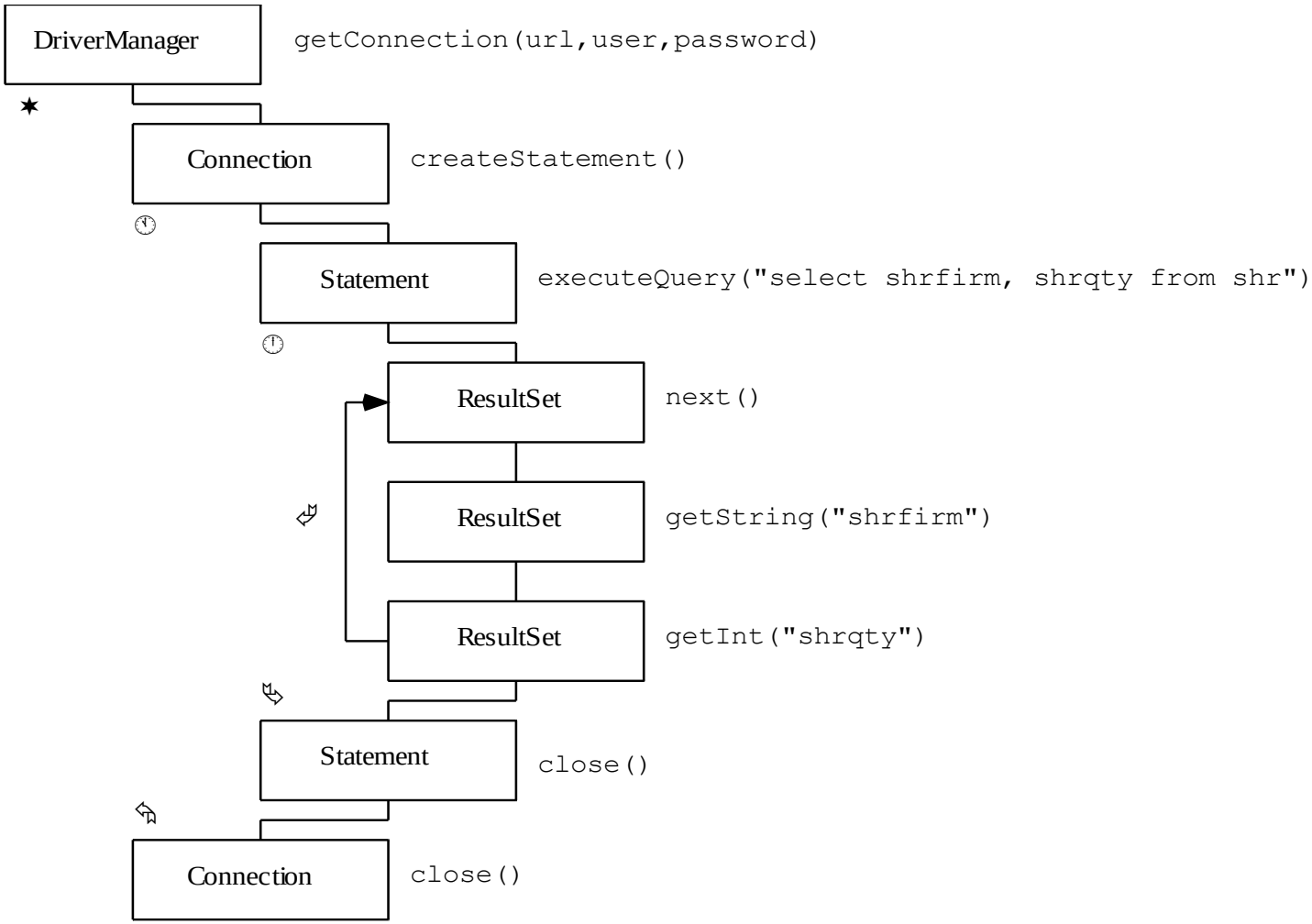
http://www-db.stanford.edu/~ullman/fcdb/oracle/or-jdbc.html

http://otn.oracle.com/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm

Architecture pour les pilotes JDBC (De Robert Godin)



Une requête SQL avec JDBC



Chargement d'un pilote JDBC (DriverManager) et établissement d'une connexion (Connection) (De Robert Godin)

- Charger les pilotes JDBC d'Oracle

  ```
  Class.forName ("oracle.jdbc.driver.OracleDriver");
  ```

  - N.B. La librairie des pilotes Oracle doit être accessible à la machine virtuelle Java.
  - Avec JDK charger le pilote : `DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());`
- Établir une connexion avec le pilote thin à l'ÉTS: `uneConnexion = DriverManager.getConnection("jdbc:oracle:thin:@fromrussia.ele.etsmtl.ca:1521:GTI440","code universel","mot de passe");`
- Créer un énoncé : `Statement enonceSQL = uneConnexion.createStatement();`
- *Le IP de fromrussia est: ? sur le port ?*

Exécution d'une opération de mise à jour (INSERT, DELETE, UPDATE) (De Robert Godin)

```
import java.sql.*;

class ClientInsertJDBC
{
  public static void main (String args [])
       throws SQLException, ClassNotFoundException, java.io.IOException
  {
    // Charger le pilote JDBC d'Oracle
    Class.forName (new oracle.jdbc.driver.OracleDriver());

    // Connexion à une BD
    Connection uneConnection =
      DriverManager.getConnection   ("jdbc:oracle:thin:@142.137.17.23:1521:rgodin
", "godin", "oracle");

    // Création d'un énoncé associé à la Connection
    Statement unEnoncéSQL = uneConnection.createStatement ();

    // Insertion d'une ligne dans la table Client
    int n = unEnoncéSQL.executeUpdate
      ("INSERT INTO CLIENT " +
       "VALUES (100, 'G. Lemoyne-Allaire', '911')");
    System.out.println ("Nombre de lignes inserees:" + n);

    // Fermeture de l'énoncé et de la connexion
    unEnoncéSQL.close();
    uneConnection.close();
  }
}
```

Exécution d'un SELECT (*ResultSet*) (De Robert Godin)

```
… Début analogue à l'exemple précédent

    // Création d'un énoncé associé à la Connexion
    Statement unEnoncéSQL = uneConnection.createStatement();

    // Exécution d'un SELECT
    ResultSet résultatSelect = unEnoncéSQL.executeQuery
     ("SELECT noClient, nomClient "+
     "FROM CLIENT " +
     "WHERE noClient > 40");

    // Itérer sur les lignes du résultat du SELECT et extraire les valeurs
    // des colonnes dans des variables JAVA

    while (résultatSelect.next ()){
       int noClient = résultatSelect.getInt ("noClient");
       String nomClient = résultatSelect.getString ("nomClient");
       System.out.println ("Numéro du client:" + noClient);
       System.out.println ("Nom du client:" + nomClient);
    }
  }
}
```

ResultSet défilable (*scrollable*), modifiable (updatable), sensible (sensitive) - JDBC2 (De Robert Godin)

```
// Création d'un énoncé avec ResultSet défilable (srollable)
Statement unEnoncéSQL =
uneConnection.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

// Exécution d'un SELECT
ResultSet résultatSelect = unEnoncéSQL.executeQuery
("SELECT noClient, nomClient FROM CLIENT");
```

```
//Positionnement à la première ligne du ResultSet
résultatSelect.first();

// Positionnement à la dernière ligne du ResultSet
résultatSelect.last();

// Positionnement à la troisième ligne
résultatSelect.absolute(3);

// Positionnement relatif (avancer de 2 lignes)
résultatSelect.relative(2);

// Positionnement à la ligne précédente
résultatSelect.previous();
```

ResultSet modifiable (*updatable*) (De Robert Godin)

createStatement (int resultSetType, int resultSetConcurrency);

resultSetType:

- TYPE_FORWARD_ONLY (défaut) défilement avant 1 fois MAJ non visibles
- TYPE_SCROLL_INSENSITIVE défilement av/ar MAJ non visibles
- TYPE_SCROLL_SENSITIVE défilement av/ar MAJ visibles

resultSetConcurrency:

- CONCUR_READ_ONLY (défaut) lecture seulement
- CONCUR_UPDATABLE select, insert, delete, update

ResultSet modifiable (*updatable*)

(De Robert Godin)

Client(noClient, nomClient)

noClient	nomClient
10	Luc Sansom
20	Dollar Tremblay
…	…
70	Simon Lecoq
{ insertRow()}	

Classe ResultSet

rowInserted()	rowDeleted()	rowUpdated()
❒	❒	❒
❒	❒	❒
❒	❒	❒
❒	❒	❒
❒	❒	❒

Navigation:

- first(); last(); next(); previous();
- absolute(int); relative(int);
- moveToInsertRow();

Mises à jour dans le ResultSet:

- updateXXX(); (XXX=type)
- cancelRowUpdates();

Mises à jour dans la BD:

- insertRow();
- deleteRow();
- updateRow();
- refreshRow();

ResultSet modifiable (*updatable*) (De Robert Godin)

```
// Création d'un énoncé avec ResultSet défilable (srollable)
Statement unEnoncéSQL =
uneConnection.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

// Exécution d'un SELECT
ResultSet résultatSelect = unEnoncéSQL.executeQuery
("SELECT noClient, nomClient, noTéléphone FROM CLIENT");

// Exemples de mises-à-jour par ResultSet updatable

//Positionnement à la première ligne du ResultSet
résultatSelect.first();
// Mise-à-jour de la colonne noTéléphone de la ligne courante
résultatSelect.updateString("noTéléphone","(111)111-1111");
// Effectuer le UPDATE
résultatSelect.updateRow();

// Positionnement à la dernière ligne de la table
résultatSelect.last();
// Supprimer la ligne courante
résultatSelect.deleteRow();

// Insertion d'une nouvelle ligne
résultatSelect.moveToInsertRow();
résultatSelect.updateInt("noClient", 100);
résultatSelect.updateString("nomClient", "G. Lemoyne-Allaire");
résultatSelect.updateString("noTéléphone", "911");
résultatSelect.insertRow();
```

mises à jour susceptibles d 'être vues pas le ResultSet

Support des types SQL3 sous JDBC 2

(De Robert Godin)

```
     CREATE TABLE tableBlob (
          idBlob    INTEGER PRIMARY KEY,
          imageBLOB)
```

```
// Chercher le BLOB locator
ResultSet unResultSet = unEnoncéSQL.executeQuery
     ("SELECT *  FROM tableBlob WHERE idBlob = 1");
if (unResultSet.next()){
 int idBlob = unResultSet.getInt(1);
 Blob unBlob = unResultSet.getBlob(2);

  // Chercher la taille du BLOB et l'afficher
  int taille = (int)unBlob.length();
  System.out.println("Taille du BLOB" + taille);
  }
// Lire le BLOB dans un tableau d'octets
byte octets[] = unBlob.getBytes(1, taille);

// Créer un fichier contenant les octets lus
FileOutputStream unFichier =
   new FileOutputStream("C:/forte4j/Development/ExemplesJDBC/CopieCoq1.gif");
unFichier.write(octets);
 unFichier.close();
```

Gestion des transactions (De Robert Godin)

- Par défaut : *auto-commit*

```
uneConnection.setAutoCommit(false);
```

- Pour modifier :
- Pour un commit explicite :

```
uneConnection.commit();
```

- Pour un rollback :

```
uneConnection.rollback();
```

Gestion des exceptions (De Robert Godin)

- Mécanisme *try catch* de JAVA

```
try
  {Class.forName ("oracle.jdbc.driver.OracleDriver");
  }
catch(ClassNotFoundException e)
  {System.err.println(" ClassNotFoundException:" + e.getMessage());
  }
```

Appel de procédure stockée (De Robert Godin)

```
CallableStatement unCall =
  uneConnection.prepareCall("{call pModifierQuantitéEnStock(?,?)}");

// Spécification des paramètres d'entrée
unCall.setInt(1,10);
unCall.setInt(2,20);
// Exécution de l'appel
unCall.execute();
unCall.close();
uneConnection.close();
```

Accès aux métadonnées (MetaData) (De Robert Godin)

```
ResultSet résultatSelect = unEnoncéSQL.executeQuery
  ("SELECT noClient, nomClient "+
  "FROM CLIENT " +
  "WHERE noClient > 40");

// Consultation de quelques méta-données du ResultSetMetaData
ResultSetMetaData unRSMD = résultatSelect.getMetaData();
int nombreColonnes = unRSMD.getColumnCount();
System.out.println("Le résultat du SELECT contient "+nombreColonnes+" colonnes");
for (int indice = 1; indice <= nombreColonnes; indice++){
    System.out.println("La colonne "+indice+
        " qui se nomme "+unRSMD.getColumnName(indice)+
        " est de type "+unRSMD.getColumnTypeName(indice));
```

Le résultat du SELECT contient 2 colonnes
La colonne 1 qui se nomme NOCLIENT est de type NUMBER
La colonne 2 qui se nomme NOMCLIENT est de type VARCHAR2

Plan de la présentation

- Interfaces entre SQL et un programme
- SQL-PSM, fonctions et procédures
- API JBDC
- **Java enchâssé SQLJ**
- Le paquetage DBMS_LOB et la manipulation de LOBs

SQLJ (De Robert Godin)

- Partie 0 de SQLJ
  - SQL enchâssée en Java
- Partie 1 de SQLJ
  - Routines stockées (*méthodes static*)
- Partie 2 de SQLJ
  - Utilisation de classes Java en tant que types SQL

http://otn.oracle.com/software/tech/java/sqlj_jdbc/index.html

SQL enchâssé en Java (SQLJ : partie 0) (De Robert Godin)

- Ne vise que le statique
- Traduit en JDBC
- Combine le SQLJ et le JDBC

```
#sql "{" énoncéSQL "}" ;
```

```
#sql {INSERT INTO CLIENT VALUES
     (100, 'G. Lemoyne-Allaire', '911')};
```

```
#sql {DELETE FROM CLIENT WHERE noClient = 10};
```

Variables partagées (De Robert Godin)

```
Int no;
String nom;
String tel;
#sql {INSERT INTO CLIENT VALUES (:no, :nom, :tel)};
```

Exemple (De Robert Godin)

```
//Exemple d'insertion d'un Client avec SQLJ
package ExemplesSQLJ;

import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import java.sql.*;

public class ClientInsertSQLJ{
    public static void main (String args [])
       throws SQLException, ClassNotFoundException, java.io.IOException
  {
    // Charger le pilote JDBC d'Oracle
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Création du contexte de connexion de défaut avec autocommit (true)
    DefaultContext unContexte = new DefaultContext
          ("jdbc:oracle:thin:@localhost:1521:ora817i", "godin", "oracle", true);
    DefaultContext.setDefaultContext(unContexte);

    // Insertion en utilisant le contexte de défaut
    #sql {INSERT INTO CLIENT VALUES (100, 'G. Lemoyne-Allaire', '911')};

    // Fermeture du contexte de connexion
    unContexte.close();

    System.out.println("Insertion réussie !");
  }
}
```

SELECT QUI RETOURNE UN record (CLAUSE INTO) (De Robert Godin)

```
String nom;
String tel;

// Utilisation de la clause INTO
#sql     { SELECT nomClient, noTéléphone
           INTO :nom, :tel
           FROM Client WHERE noClient = 10};
```

ITÉRATEUR DE RÉSULTAT SQLJ

(De Robert Godin)

```
// Définition de la classe IteratorClient avec liaison par nom
#sql iterator IteratorClient(int noClient, String nomClient);

// Création d'un objet itérateur
IteratorClient unIteratorClient;

// Liaison de l'énoncé SELECT de l'itérateur
#sql     unIteratorClient =
    { SELECT noClient, nomClient
      FROM Client WHERE noClient > 40};

// Accès au résultat du SELECT par itération sur les lignes
while (unIteratorClient.next()){
  System.out.println("Numéro du client : " + unIteratorClient.noClient());
  System.out.println("Nom du client : " + unIteratorClient.nomClient());
}

// Fermer l'itérateur
unIteratorClient.close();
```

Mise-à-jour par itérateur (De Robert Godin)

```
// Définition de la classe IteratorClient avec liaison par nom
#sql iterator IteratorClient implements sqlj.runtime.ForUpdate
  (int noClient, String nomClient);

// Création d'un objet itérateur
IteratorClient unIteratorClient;

// Liaison de l'énoncé SELECT de l'itérateur
#sql    unIteratorClient =
    { SELECT noClient, nomClient, noTéléphone
      FROM Client WHERE noClient > 40};

// Accès au résultat du SELECT par itération sur les lignes
while (unIteratorClient.next()){
  if (unIteratorClient.noClient()== 60){
    #sql {UPDATE Client
          SET noTéléphone = '(111)111-1111'
          WHERE CURRENT of :unIteratorClient};
  } else if (unIteratorClient.noClient()== 80){
    #sql {DELETE FROM Client
          WHERE CURRENT of :unIteratorClient};
  }
```

APPEL DE ROUTINES STOCKÉES EN SQLJ (De Robert Godin)

```
    int laQuantite = 0;
    int noArticle = 10 ;

    // Appel de la fonction stockée
    #sql laQuantite = {VALUES (fQuantitéEnStock(:noArticle))};

    // Fermeture du contexte de connexion
    unContexte.close();
    System.out.println("Quantité en stock :" + laQuantite);  }
```

```
    // Appel de la procédure stockée
    #sql {CALL pModifierQuantitéEnStock(:in noArticle,:in laQuantite)};
```

Le paquetage DBMS_LOB et la manipulation de LOBs

MANIPULER de grands OBJETS

Il y a trois approches pour l'accès à des bases de données multimédia dans *Oracle*

- L'interface d'appel Oracle (OCI, OCCI)
- Les APIs (JMF, JSP, JAI,…) on a vu JMF
- **Les paquetages :**
  - **DBMS_LOB**
  - **Oracle *inter*Média**

Le paquetage DBMS_LOB

- Ensemble de fonctions additionnelles pour la gestion des LOBs
- Accès à ces fonctions :
  - l'utilisateur SYS doit exécuter les scripts dbmslob.sql et prvtlob.plb, ou le script catproc.sql
  - il doit donner les droits d'accès à ce paquetage aux utilisateurs appropriés
- Utilisation via SQL, PL/SQL, SQLJ en général
- Ne distinguent pas explicitement CLOB et BLOB

Le paquetage DBMS_LOB

Enregistrer et manipuler des grands objects

BFILEs, BLOB, CLOB, NCLOB

- Max 4 gigabits
- 17 fonctions disponibles

Les fonctions DBMS_LOB

Name	Description	Use in SQL
APPEND	Appends the contents of a source internal LOB to a destination internal LOB	No
COMPARE	Compares two LOBs of the same type; parts of LOBs can also be compared	Yes
COPY	Copies all or part of the contents of a source internal LOB to a destination internal LOB	No
ERASE	Erases all or part of an internal LOB	No
FILECLOSE	Closes an open BFILE	No
FILECLOSEALL	Closes all open BFILEs	No

Les fonctions DBMS_LOB

Name	Description	Use in SQL
FILEEXISTS	Checks if a given file exists	Yes
FILEGETNAME	Returns directory alias and filename of given file locator	No
FILEOPEN	Opens a BFILE for read-only access	No
FILEISOPEN	Determines if a BFILE was opened with the given file locator	Yes
GETLENGTH	Returns the length of the input LOB; length is in bytes for BFILEs and BLOBs; length is in characters for CLOBs and NCLOBs	Yes

Les fonctions DBMS_LOB

Name	Description	Use in SQL
INSTR	Returns matching offset location in the input LOB of the Nth occurrence of a given pattern	Yes
LOADFROMFILE	Loads all or part of external LOB to internal LOB	No
READ	Provides piece-wise read access to a LOB	No
SUBSTR	Provides piece-wise read access to a LOB	Yes
TRIM	Trims the contents of an internal LOB to the length specified by the newlenparameter	No
WRITE	Writes a given number of bytes or characters to an internal LOB at a specified offset	No

Le paquetage DBMS_LOB

- Exemples d'appels de fonctions
  - E/S
    - DBMS_LOB.OPEN, DBMS_LOB.READ, DBMS_LOB.WRITE, DBMS_LOB.WRITEAPPEND, DBMS_LOB.LOADFROMFILE, DBMS_LOB.CLOSE
  - Manipulation de LOBs
    - DBMS_LOB.SUBSTR, DBMS_LOB.INSTR, DBMS_LOB.GETLENGTH, DBMS_LOB.TRIM, DBMS_LOB.APPEND, DBMS_LOB.ERASE …
  - Divers
    - DBMS_LOB.CREATETEMPORARY

Utiliser DBMS_LOB avec PL/SQL

- **Lob_locator** (localisateur) pour manipuler le LOB
- **Amount** (quantité) du LOB à manipuler
- **Buffer** établi la taille du tampon en PL/SQL
- **Position** établi l'endroit de départ pour la manipulation du LOB

Les variables PL/SQL **amount** et **buffer** sont initialisées à la taille maximale. Les données en binaire peut être lu à partir de la position 1 du fichier.

```
Lob_loc         BFILE;
Amount          INTEGER := 32767;
Position        INTEGER := 1;
Buffer          RAW(32767);
```

Commentaires

- Nombre max de fichiers ouverts
  - défini par SESSION_MAX_OPEN_FILES (fichier INIT.ORA)
  - défaut fixé à 10, modifiable par

```
SET SESSION_MAX_OPEN_FILES = 20;
```

- Taille max du tampon de PL/SQL : 32 Ko
  - VARCHAR2(n) limité à 4000 comme colonne d'une table
  - VARCHAR2(n) limité à 32 Ko comme variable PL/SQL
  - Pour charger des LOBs plus gros, utiliser SQL*LOADER (chargement en lot)

Section de déclaration PL/SQL

```
CREATE OR REPLACE PROCEDURE wine_read_bfile
IS
    Lob_loc         BFILE :=
  BFILENAME('PHOTO_DIR', 'chardonnay.jpg');
    Amount          INTEGER := 32767;
    Position        INTEGER := 1;
    Buffer          RAW(32767);
```

Corps de la Procédure

```
BEGIN
   DBMS_LOB.OPEN(Lob_loc, /*Open BFILE:*/
     DBMS_LOB.LOB_READONLY);

   DBMS_LOB.READ(Lob_loc, /*Read data:*/
           Amount, Position, Buffer);

   DBMS_LOB.CLOSE(Lob_loc);/*Close BFILE:*/
 END;
```

Mise à jour d'un LOB en PLSQL

```
BEGIN
   SELECT grape_text
   INTO Lob_loc
   FROM grape
   WHERE grape_name = 'Chardonnay' FOR UPDATE;

   Amount := LENGTH(newtext);
   DBMS_LOB.WRITEAPPEND(Lob_loc,Amount,
                        newText);
END;
```

Réduire la taille d'un texte (CLOB)

```
PROCEDURE Trim_Clob IS
   clob_loc CLOB;
BEGIN
-- get the LOB Locator
   SELECT data into clob_loc
   FROM lob_table
   WHERE id = 179 FOR UPDATE;
-- call the TRIM Routine
   DBMS_LOB.TRIM(clob_loc, 834004);
   COMMIT;
END;
```

Utiliser DBMS_LOB Package avec SQL ou PL/SQL

```
SELECT grape_name,
       DBMS_LOB.GETLENGTH(grape_text),
       DBMS_LOB.SUBSTR(grape_text, 10,10)
FROM grape
```

Formulation équivalente dans le cas d'un CLOB :

```
SELECT grape_name, LENGTH(grape_text),
       SUBSTR(grape_text, 10,10)
FROM grape
```

SQL avec la fonction BFILENAME

```
INSERT INTO grape (grape_name, picture)
VALUES ('chardonnay',
  BFILENAME('PHOTO_DIR', 'chardonnay.jpg');

UPDATE grape
SET picture = BFILENAME('PHOTO_DIR',
  'chardonnay.jpg');
```

Utiliser BLOBs ou BFILES

- BFILEs ne sont pas sous le contrôle de la base de donnée et les utilisateurs peuvent détruire les fichiers, les changer d'endroit
- Quand un item dépasse 4 Gig
- BFILE ne sont pas utilisables pour faire des requêtes sur le contenu
- Utile pour emmagasinage temporaire avant de passer à LOB

Avant d'insérer des données dans une colonne image vous devez

- Assurer que le localisateur est non-nul. Initialisation du localisateur de la colonne BFILE.
- Associer l'objet BFILE avec un répertoire dans votre ordinateur (lequel va contenir les fichiers)

```
CREATE DIRECTORY "PHOTO_DIR" AS
   'C:\PICTURES';
```

Lecture d'un BFILE référencé dans une table

```
CREATE OR REPLACE PROCEDURE wine_read_bfile  IS
    Lob_loc         BFILE;
    Amount          INTEGER := 32767;
    Position        INTEGER := 1;
    Buffer          RAW(32767);
BEGIN
    /* Select the LOB: */
    SELECT picture INTO Lob_loc FROM grape
    WHERE grape_name ='chardonnay';
    /* Open the BFILE: */
    DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
    /* Read data: */
    DBMS_LOB.READ(Lob_loc, Amount, Position, Buffer);
    /* Close the BFILE: */
    DBMS_LOB.CLOSE(Lob_loc);
END;
```

Sommaire – cours #3

- Accès divers à une BD par des langages:
  - Appels BD de plusieurs manières
  - Introduction PL/SQL, SQLJ et JDBC
  - L'utilisation de PL/SQL pour manipuler les LOBs
  - Le DBMS_LOB package

Travaux Personnels et labo

- Terminez votre premier labo !
- Commencer à planifier l'intégration avec le labo 2 !