Bases de Données SQL et NoSQL
LI328 – Technologies Web

Mohamed-Amine Baazizi

(Slides de B. Amann)
UPMC - LIP6

SGBD → Universalité

Systèmes « SQL » :

- Facilité d'utilisation
- Cohérence des données
- Persistance des données
- Fiabilité (pannes)
- Efficacité
- **Universalité**

Vues **SQL**

Données structurées

Transactions

Optimisation de requêtes

Indexation des données

« One size fits all »

Les évolutions...

Nouvelles **Données** :

- Web 2.0 : Facebook, Twitter, news, blogs, ...
- LOD : graphes, ontologies, ...
- Flux : capteurs, GPS, …

Nouveaux **Traitements** :

- Moteurs de recherche
- Extraction, analyse, ...
- Recommandation, filtrage collaboratif, …

Nouvelles **Infrastructures** :

- Cluster, réseaux mobiles, microprocesseurs multi-coeurs, …

➔ très gros volumes, données pas ou faiblement structurées

➔ transformation, agrégation, indexation

➔ distribution, parallélisation, redondance

Évolution → Spécialisation

Systèmes « noSQL » (***n**ot **o**nly* SQL) :

- **Facilité d'utilisation**
- Cohérence des données
- **Persistence des données**
- Fiabilité (pannes)
- **Efficacité**
- ~~Universalité~~

Langages spécialisées

Données hétérogènes

Réplication

Parallélisation

Indexation de contenus

« Systèmes sur mesure »

SQL ↔ NoSQL

Cohérence forte :

- Logique : Schémas, contraintes
- Physique : Transactions ACID

Distribution des **données**

- Transactions distribuées

↔

Ressources limitées

- Optimisation de requêtes

Langage **standard** : SQL

Cohérence faible :

- ~~Schémas, contraintes~~
- Cohérence « à terme »

Distribution des **traitements** :

- Traitements « batch »
- MapReduce

Ressources « **illimitées** »

- Passage à l'échelle horizontal

Langages **spécialisés**, API

Plan

- 1re séance :
  - Introduction comparative des systèmes SQL et NoSQL
  - Bases de données SQL
- 2nde séance :
  - Bases de données NoSQL

Objectifs

- Objectif de ce cours :
  - Savoir développer des applications simples avec Java et un SGBD
- Pré-requis
  - Bases de données relationnelles (LI341) : SQL, curseurs, transactions
  - Programmation Java : classes, méthodes, exceptions

Plan

- Programmation Bases de Données
- JDBC premiers pas
  - Architectures
  - *Driver, Connection, Statement, Rowset*
- JDBC avancé
  - Métaschéma
  - Transactions
  - Exceptions
  - *Datasource*

Programmation Bases de Données

Programmation et Bases de Données

Langages de Programmation :

- JAVA, C, C#, JavaScript …
  - Langages procédural : « je **fais** quoi ? »
  - Accès itératif :
    - boucles
    - récursion
  - Types complexes

- **Programmation d'applications**
- **Données structurées en mémoire**

Langages de requêtes :

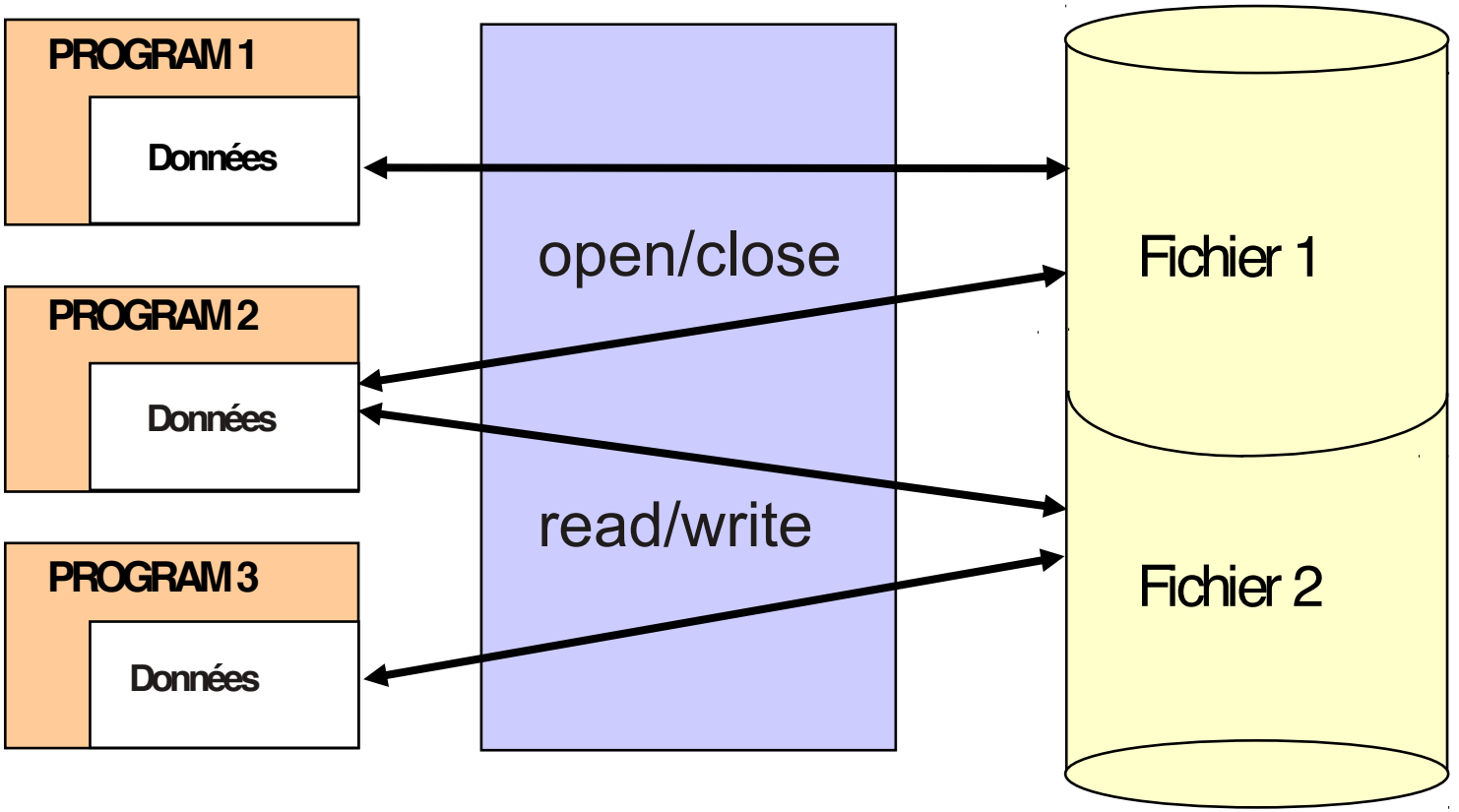
- SQL : Structured Query Language
  - Langage déclaratif : « je **veux** quoi ? »
  - Accès ensembliste : {a,b,c} → Q → {d,e}
  - Types simples

- **Accès aux données gérés par un SGBD**
- **Données structurées sur disque**

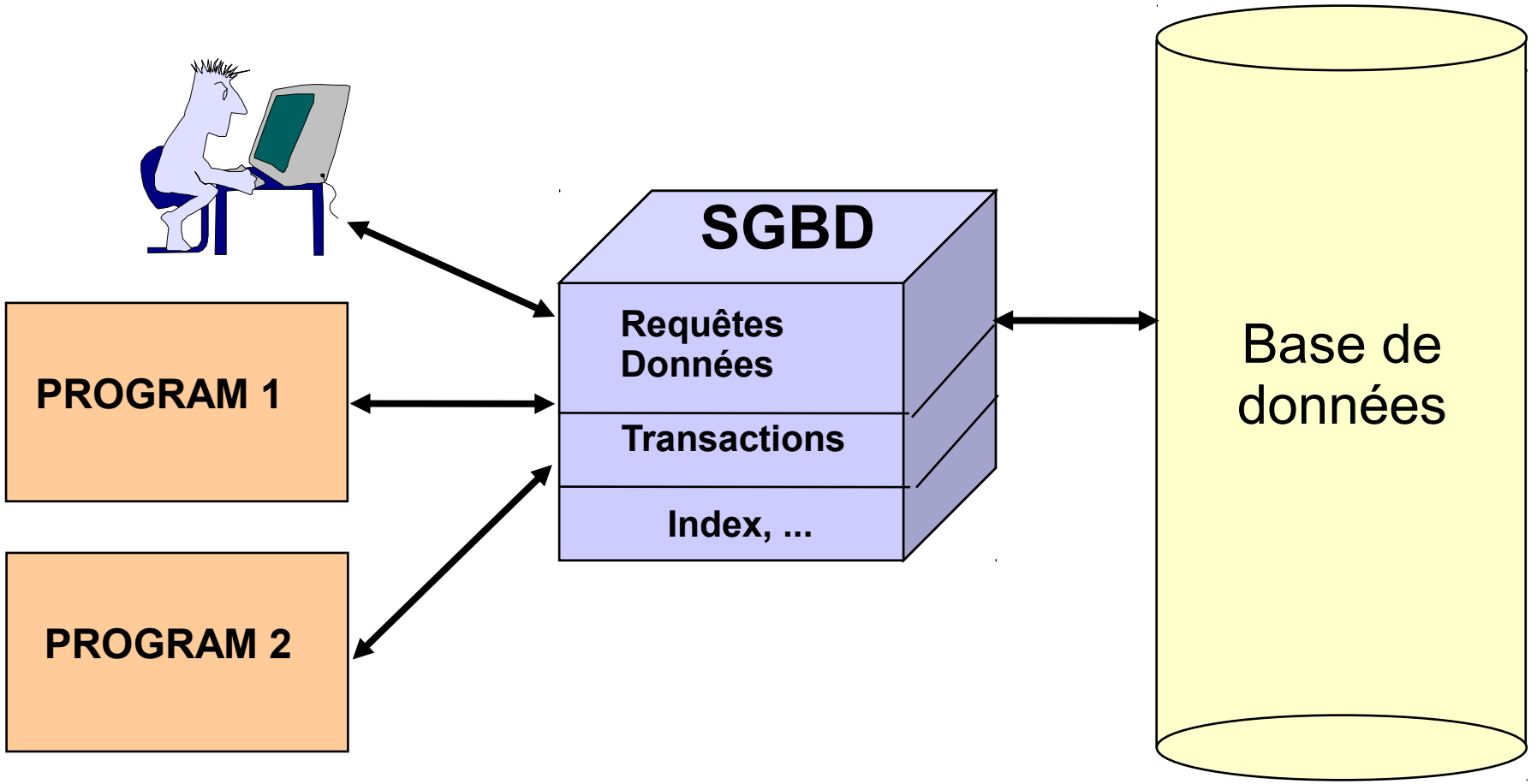
Java + Fichiers

Fichiers :

- opérations simples : ouvrir/fermer, lire/écrire
- différentes méthodes d'accès (séquentiel, indexé, haché, etc.)
- utilisation par plusieurs programmes difficile (format ?)



Java + BD


SGBD
Requêtes
Données
Transactions
Index, ...
PROGRAM 1
PROGRAM 2
Base de
données

Fichiers versus BD

Fichier :

- Faible structuration des données
- Dépendance entre programmes et fichiers
- Redondance des données
- Absence de contrôle de cohérence globale des données

Base de Données :

- Structuration des données à travers un schéma de données
- Indépendance entre programmes et données
- Données partagées
- Contrôle de la cohérence logique et physique (schémas, transactions)

SQL

SELECT	$var_i.A_{ik}$, …	attributs
FROM	R_{i1} var_1, R_{i2} var_2…	tables
WHERE	P	prédicat/condition

- var_i désigne la table R_{ij}
- Les variables dans la clause SELECT et dans la clause WHERE doivent être *liées* dans la clause FROM.

Simplifications :

- Si var_j n'est pas spécifiée, alors la variable s'appelle par défaut R_{ij}.
- Si une seule table/variable *var* possède l'attribut A, on peut écrire plus simplement A au lieu de *var*.A.

Requêtes

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)

Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

Noms et titres des employés qui travaillent dans un projet pendant plus de 17 mois?

```
SELECT Ename, Title
FROM   Emp, Works
WHERE  Dur > 17
AND Emp.Eno = Works.Eno
```

Noms et titres des employés qui travaillent dans un projet à Paris ?

```
SELECT Ename, Title
FROM   Emp E, Works W, Project P
WHERE  P.City = 'Paris'
AND    E.Eno = W.Eno AND W.Pno = P.Pno
```

Couplage SQL–langage de programmation (Embedded SQL)

Accès une BD depuis un programme d'application

- SQL n'est pas suffisant pour écrire des applications (SQL n’est pas « Turing complet »)

SQL a des liaisons (bindings) pour différents langages de programmation

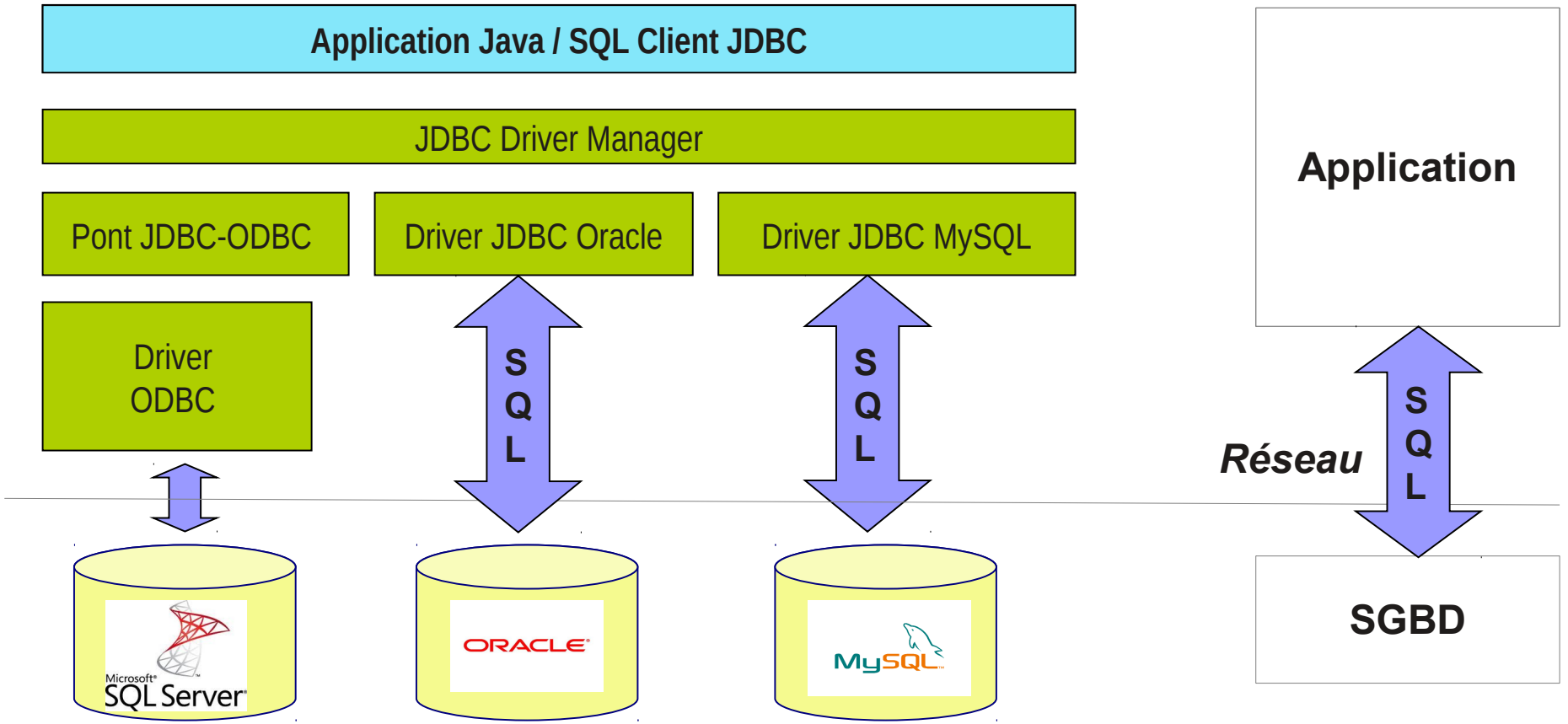
- C, C++, Java, PHP, etc.
- les liaisons décrivent la *connexion* de langages *hôtes* avec un SGBD relationnel

JDBC
Java Database Connectivity

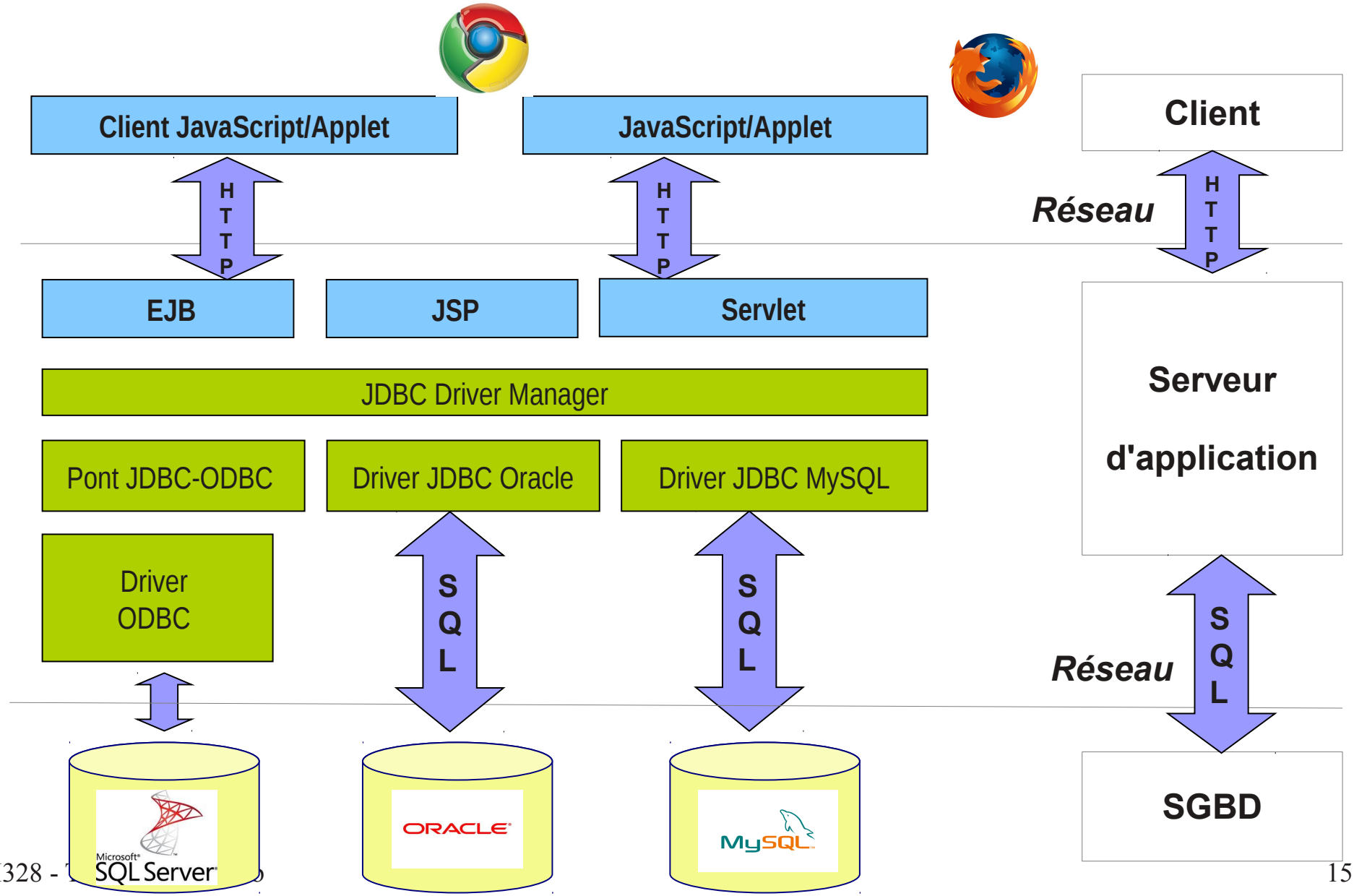
JDBC 2.0 : Fonctions

- API Java standard pour accéder à des données *tabulaires*
  - BD relationnelles, mais aussi feuilles Excel,...
- Bibliothèque Java 2 SDK
- Requêtes SQL (select, update, insert, delete)
- Gestion de transactions
- Traitement des exceptions / erreurs

JDBC : Architecture Logiciel



JDBC : Architecture Web



Programmation JDBC

1) Charger Driver JDBC
2) Créer connexion BD
3) Accès aux données (requêtes, maj) :
   - Créer ordre SQL (statement object)
     - Exécuter ordre SQL
     - Traiter le résultat (ResultSet, curseur)
   - Fermer ordre SQL (close)
4) Fermer la connexion (close)

Chargement Driver JDBC

- Driver = classe Java
- Class.forName(“package.classe_driver”)
  - Class.forName(”com.mysql.jdbc.driver");
  - Class.forName(”oracle.jdbc.driver.OracleDriver");
  - Class.forName(”sun.jdbc.odbc.JdbcOdbcDriver”);
- Exception / erreur :
  - Par exemple : le driver n'est pas accessible par CLASSPATH
  - java.lang.ClassNotFoundException e

Code pour MySQL

```
try {
    Class.forName("com.mysql.jdbc.Driver");
 }
catch(java.lang.ClassNotFoundException e) {
        System.err.print("Exception: ");
        System.err.println(e.getMessage());
 }
```

Connexion au SGBD (1)

- Établir une connexion :

  Connection c =

  Driver.getConnection(<url>, <login>, <mdp>)
- <url> = <driver>:<protocole>:@<addr_bd>
- Exemple :

  jdbc:oracle:thin:@oracle.ufr-info-p6.jussieu.fr:1521:ora10

  driver protocole addr_bd

Connexion au SGBD (2)

- Établir une connexion :
  Connection c = Driver.getConnection(<string>) ;
- <string> = toutes les infos de connexion
- Exemple :

jdbc:mysql://<host>:3306/<bd> user=<user>&password=<pw>”

driver host port nom_bd login mdp

Créer et exécuter des ordres (statement) SQL

Créer un ordre (statement) :

- Statement stmt = c.createStatement();
- c est une connexion ouverte/active vers une bases de données

Exécuter un ordre :

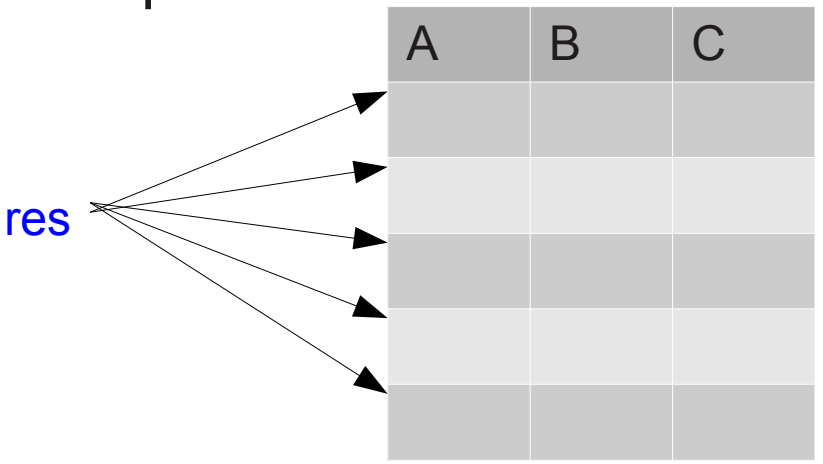
- stmt.executeQuery(<sql>) : interrogation (QL)
- stmt.executeUpdate(<sql>) : mises-à-jour données (DML) et schéma (DDL)

Exécution d'ordres

- **ResultSet** res = statement.executeQuery("select A,B,C from R")
- int executeUpdate(“insert ...”) → nombre de lignes insérées
- Int executeUpdate(“update ...”) → nombre de lignes modifiés
- int executeUpdate(“create table ...”)
- int executeUpdate(“drop table ...”)

Comment accéder au résultat d'une requête ?

- **ResultSet** res = **curseur**



ResultSet : accès à la réponse

- rs.beforeFirst() :
  - Place le curseur avant la première ligne (défaut à l'ouverture du curseur)
- rs.next() :
  - Avance vers la prochaine ligne dans la réponse (premier appel avance vers la première ligne)
  - Retourne false si fin ou erreur
- rs.close():
  - Libère les ressources JDBC/BD
  - rs est fermé automatiquement avec l'exécution d'une nouvelle requête

ResultSet (Continued)

- rs.get<type>(<attr>)
  - Retourne la valeur de l'attribut <attr> avec transformation vers le type java <type>
  - <attr> : nom ou position de l'attribut
  - <type> :

```
double      byte        int
Date        String      float
short       long        Time
Object
```

Transformation de types SQL ↔ JAVA

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	BOOLEAN	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP	CLOB	BLOB	ARRAY	REF	STRUCT	DATALINK	JAVA OBJECT
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x	x													
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x	x													
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x	x													
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x	x													
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x	x													
getDouble	x	x	x	x	x	X	X	x	x	x	x	x	x	x													
getBigDecimal	x	x	x	x	x	x	x	X	X	x	x	x	x	x													
getBoolean	x	x	x	x	x	x	x	x	x	X	X	x	x	x													
getString	x	x	x	x	x	x	x	x	x	x		X	X	x	x	x	x	x	x	x						x	
getBytes															X	X	x										
getDate												x	x	x				X		x							
getTime												x	x	x					X	x							
getTimestamp												x	x	x				x	x	X							
getAsciiStream												x	x	X	x	x	x										
getUnicodeStream												x	x	X	x	x	x										
getBinaryStream															x	x	X										
getClob																					X						
getBlob																						X					
getArray																							X				
getRef																								X			
getCharacterStream												x	x	X	x	x	x										
getURL																										X	
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	X	x	X

Exemple

```
try {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    connexion = DriverManager.getConnection(url, user, password);
    Statement lecture =  connexion.createStatement();
    ResultSet curseur = lecture.executeQuery(“select...”);
    while (curseur.next()) {
          out.println(curseur.getString(“nom”));
          out.println(curseur.getInt(“age”));
     }
    curseur.close();
    lecture.close();
    connexion.close();
 }
 catch(java.lang.ClassNotFoundException e) {
     System.err.print("Exception: ");
     System.err.println(e.getMessage());
 }
```

JDBC Avancé

createStatement, prepareStatement, prepareCall

- Ordre avec exécution unique :
  - Statement s = createStatement(<param_opt>);
- Ordre avec exécution multiple :
  - PreparedStatement ps = prepareStatement(<sql>, <param_opt>);
- Appel de procedure stockée :
  - CallableStatement pc = prepareCall(<sql>, <param_opt>);

PreparedStatement

- PreparedStatement
  - Requête SQL avec un '?' pour chaque paramètre
  - set<type>(<int>,<valeur>) :
    - setT(i,v) : remplace le i-ème '?' par la valeur v de type T
- Avantages :
  - Exécution plus rapide : une seule phase d'optimisation pour plusieurs exécutions
  - Réutilisation de code
  - Facilite le passage de paramètres :
    - par exemple chaînes avec “'”

Exemple

```
PreparedStatement updateSales;
String updateString = "update NOTES set NOTE = ? " +
                      "where NO_ETUDIANT = ?";
updateSales = con.prepareStatement(updateString);
int [] notes = {12,20,2,9};
int [] nums = {123456,289068,376876,465687};
int len = notes.length;
for(int i = 0; i < len; i++) {
        updateSales.setInt(1, notes[i]);
        updateSales.setInt(2, nums[i]);
        updateSales.executeUpdate();
}
```

Creation de statement : paramètres optionnels

- int resultSetType
  - TYPE_FORWARD_ONLY : lecture avant (par défaut)
  - TYPE_SCROLL_INSENSITIVE : lecture aléatoire
  - TYPE_SCROLL_SENSITIVE : lecture aléatoire, lectures sales
- int resultSetConcurrency
  - CONCUR_READ_ONLY : sans maj (verrous partagés), concurrence élevée
  - CONCUR_UPDATABLE : avec maj (verrous exclusives), concurrence plus faible
- int resultSetHoldability
  - HOLD_CURSORS_OVER_COMMIT : curseur reste ouvert au moment d'un commit
  - CLOSE_CURSORS_AT_COMMIT : curseur est fermé au moment d'un commit

Voir documentation pour des exemples...

ResultSet : accès aléatoire

- rs.previous():
  - Recule vers la ligne précédente
- rs.absolute(int <num>):
  - Aller vers la ligne <num>
- rs.relative (int <num>):
  - Avancer (positif) ou reculer (négatif) <num> lignes
- rs.first() / rs.last()
  - Aller vers la première / dernière ligne

ResultSet : métadonnées

- int rs.getMaxRows() / setMaxRows(int max):
  - Détermine la taille maximale (nombre de lignes) d'un ResultSet
  - 0 = taille illimitée
- int getQueryTimeout() / setQueryTimeout(int ms):
  - Détermine le temps d'attente maximal avant de déclencher une exception (SQLException)

ResultSet : Mises-à-jour

- Possible sous certaines contraintes :
  - Une seule table dans FOR
  - Pas de group by
  - Pas de jointures
- Conditions pour insert :
  - Tous les attributs non nul s et sans valeurs par défaut sont dans SELECT
- Sinon on utilise les ordres SQL:
  - UPDATE, DELETE, INSERT

ResultSet : Mises-à-jour

- Mise-à-jour d'une ligne :
  rs.update<type>(<attr>,<valeur>), ... : maj en mémoire
  rs.updateRow() : propagation dans la BD
- Effacement d'une ligne :
  - rs.deleteRow() : effacer ligne actuelle
- Insertion d'une ligne :
  - rs.moveToInsertRow();
  - rs.updateInt(1,123) ;
  - rs.updateString(2, 'toto') ;
  - rs.insertRow();

ResultSet : Métadonnées

- Méta-données :
  - Nombre d'attributs
  - Nom et type d'un attributs
  - …
- Méthodes :
  - ResultSetMetaData md = rs.GetMetaDataObject():
  - int md.getColumnCount();
  - String md.getColumnName(int column);
  - String md.getTableName(int column);
  - String md.getColumnTypeName(int column);

Traitements des exceptions

Pratiquement toutes les méthodes peuvent déclencher des exception SQLException avec

- SQLException sqle ;
- sqle.GetMessage() : description de l'erreur
- sqle.GetSQLState() : état SQLState (spécification Open Group SQL specification)
- sqle.GetErrorCode() : code d'erreur
- sqle.GetNextException() : exception suivante (si présente)

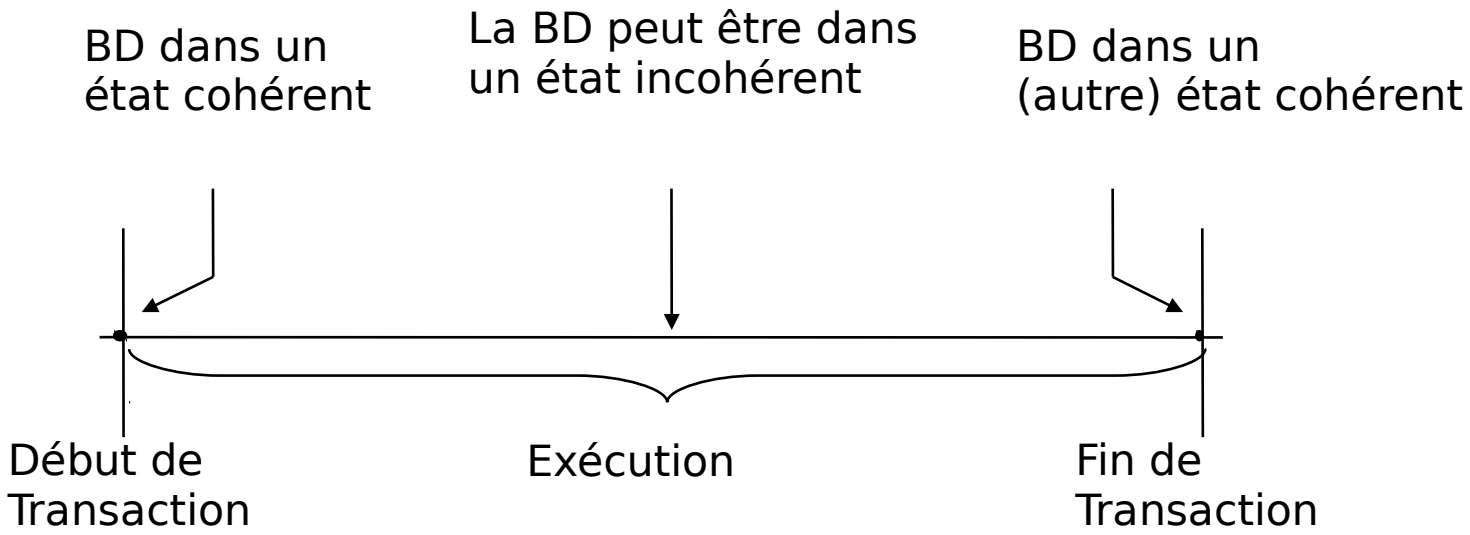
SQL Exception Example

```
try {

     ... // JDBC statement.

} catch (SQLException sqle) {
   while (sqle != null) {
      System.out.println("Message:"+sqle.getMessage());
      System.out.println("SQLState:" + sqle.getSQLState());
      System.out.println("Vendor Error:"+
                  sqle.getErrorCode());
      sqle.printStrackTrace(System.out);
      sqle = sqle.getNextException();
   }
}
```

Transactions

Transaction = séquence d'actions qui transforment une BD d'un état *cohérent* vers un autre état *cohérent*

- opérations de lecture et d'écriture de données de différentes granularités
- ***granules*** = tuples, tables, pages disque, etc…



Programmation

Une transaction est démarrée

- *implicitement* : au début du programme ou après la fin d'une transaction (Oracle) ou
- *explicitement* : par une instruction (start_transaction)

Une transaction comporte des opérations de :

- *lecture* ou *écriture* (SQL)
- *manipulation* : calculs, tests, etc. (JAVA)
- *transactionnelles* :
  - commit :
    - validation des modifications
    - explicite ou implicite à la fin
  - abort (ou rollback):
    - annulation de la transaction
    - on revient à l'état cohérent initial avant le début de la transaction

Transactions dans JDBC

- Transaction :
  - Exécution atomique (tout ou rien) d'une collection d'ordres
- Mode automatique (autocommit) :
  - mode par défaut à la création d'une connexion
  - chaque ordre est traité comme une transaction
- Contrôle explicite :
  - c.setAutoCommit(false);
- Validation explicite :
  - c.commit();
- Annulation explicite :
  - c.rollback();

Propriétés des transactions

ATOMICITE : Les opérations entre le début et la fin d'une transaction forment une *unité d'exécution*

DURABILITE: Les mises-à-jour des transactions validées *persistent*.

Gestion de **pannes**

- Cache
- Journalisation

COHERENCE : Chaque transaction accède et retourne une base de données dans un état cohérent (pas de violation de contrainte d'intégrité).

ISOLATION : Le résultat d'un ensemble de *transactions concurrentes* et validées correspond au résultat d'une exécution *successive* des mêmes transactions.

Gestion de **cohérence**

- Sérialisibilité
- Algorithmes de contrôle de concurrence

Degrés d'isolation SQL-92

Lecture sale (lecture d'une maj. non validées):

T1: Write(A); T2: Read(A); T1: abort

- Si T_2 annule, T_1 a lu des données qui n'existent pas dans la base de données

Lecture non-répétable (maj. intercalée) :

T1: Read(A); T2: Write(A); T2: commit; T1: Read(A);

- La deuxième lecture de A par T_1 peut donner un *résultat différent*

Fantômes (requête + insertion) :

T1: Select where R.A=…; T2: Insert Into R(A) Values (…);

- Les *insertions* par T2 ne sont pas détectées comme concurrentes pendant l'évaluation de la requête par T1 (résultat incohérent possible).

Conclusion

- Driver JDBC : connexion d'une application Java vers un SGBD.
- Six étapes :
  - Charger driver JDBC
  - Établir connexion vers SGBD
  - Création d'objet Statement
  - Exécution de requêtes
  - Traitement des résultats (ResultSet)
  - Fermeture de la connexion
- PreparedStatements
  - Plus rapides (optimisation)
  - Paramétrage
- Transactions :
  - Commit(), rollback()

Références

- JDBC Data Access API :
  - http://www.oracle.com/technetwork/java/javase/jdbc/index.html
- Tutoriel :
  - http://docs.oracle.com/javase/tutorial/

Alternatives à JDBC

- ODBC :
  - JDBC pour C (pointeurs)
  - Plus complexe
- Hibernate : Mapping Objet-Relationnel (ORM)
  - Classes Java *persistantes*
  - *HQL : Hibernate Query Language*

Exemple complet

Exemple complet

```
import java.sql.*;
import java.io.*;
public class Acces {
    String url =
                "jdbc:oracle:thin:@oracle.ufr-info-p6.jussieu.fr:1521:ora10";
    String user = "M1000";              // login
    String password = "M1000";    //  mot de passe
    Connection connexion = null;
    PrintStream out = System.out;
```

Exemple complet

```
public static void main(String in[]) {
        Acces c = new Acces();
        String sql = in[0];
        c.traiteRequete(sql);
}
public Acces(){
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        }
        catch(Exception e) {
            gestionDesErreurs(e);
        }
}
```

Exemple complet

```
public void traiteRequete(String requete) {
    out.println(requete);
    try {
        connexion = DriverManager.getConnection(url, user, password);
        Statement lecture =  connexion.createStatement();
        ResultSet curseur = lecture.executeQuery(requete);
        ResultSetMetaData infoResultat = curseur.getMetaData();
        int nbChamp =infoResultat.getColumnCount();
        out.println("le resultat de la requete est:");
        String entete = "";
        for(int i=1; i<= nbChamp; i++) {
            String nomChamp = infoResultat.getColumnName(i);
            entete = entete + nomChamp + "\t";
        }
        out.println(entete);
        out.println("--------------------------");
```

Exemple complet

```
        /* affichage des tuples */
        while (curseur.next()) {
            String tuple = "";
            for(int i=1; i<= nbChamp; i++) {
                String valeurChamp = curseur.getString(i);
                tuple = tuple + valeurChamp + "\t";
            }
            out.println(tuple);
        }
    curseur.close();
    lecture.close();
    connexion.close();
    }
    catch(Exception e) {
        gestionDesErreurs(e);
    }
} /* fin traiteRequete */
```

Exemple complet

```
protected void gestionDesErreurs(Exception e) {
    out.println("Exception: " + e);
    e.printStackTrace();
    try {
        if (connexion != null)
            connexion.close();
    }
    catch(Exception se) {
        out.println("Tout autre probleme: " + se);
    }
    throw new RuntimeException("Arret immediat");
}
```