Programmation Procédurale et SQL

Didier DONSEZ
Université Joseph Fourier
IMA –IMAG/LSR/ADELE
Didier.Donsez@imag.fr,
Didier.Donsez@ieee.org

Motivations (i)

- Limites de SQL
  - Langage déclaratif (non procédural)
    - facile d ’exprimer des requêtes
    - MAIS pas de structure de contrôle

      boucle itérative, contrôle séquentiel
  - Il existe parfois une solution en « pur SQL »
    - Exemple du puzzle de Joe Celko
    - Certes, il faut se creuser la tête
- Sinon
  - Langage Procédural + SQL

Motivations (ii)

- Besoins
  - Procédure
    - variables locales
    - structures de contrôle
      boucle itérative, test, exception
    - parcours du résultat d 'une requête ligne à ligne
      curseur
      imbrication des parcours (curseurs imbriqués)
  - Exécution
    - par le client / par le serveur
- Curseur
  - « Pointeur » sur la ligne courante
  - Transfert des valeurs
    entre la ligne pointée et les variables de la procédure

3 Solutions

- SQL Dynamique
  - Interface SQL/CLI et Middleware ODBC et JDBC
- Embedded SQL in 3GL
- « SQL Procédural »

Remarque 1:
Problème de l ’Impedance Mismatch

- Typage différent entre SQL et C, C++, Java, ...
  - conversion, arithmétique

    NUMBER(x) ⌦ **int ? , long ?**
    DECIMAL(x,y), NUMBER(x,y) ⌦ **float ? double ?**
    VARCHAR(x) ⌦ **char[x+1]?, string ?**
  - constructeur de type
- Les valeurs NULL
  - représentent les valeurs manquantes ou non renseignées
  - absente des langages hôtes
- Logique à 3 niveaux
  - **TRUE, UNKNOW, FALSE**

    ((1,NULL,4) = (1,2,NULL)) IS UNKNOW

Remarque 2:
Risques de la programmation procédurale

- La solution SQL pur n 'est pas trivial
  - le développeur propose une solution procédurale
    - suite de plusieurs requêtes SQL
  - le développeur se substitue parfois à l 'optimiseur
- Exemple
  - imbrication de curseurs sur deux tables
  - pour réaliser une jointure

Interface SQL/CLI et Middleware ODBC et JDBC

- Motivations
- Dynamic SQL
- SQL/CLI
- ODBC
- JDBC
- DBI pour PERL
- Avantages et Inconvénients

Motivations

- Connexions simultanées vers plusieurs bases
  - Consultation et Modification
  - La requête n ’est analysée qu’à l ’exécution
    - son code SQL peut être généré ou saisis qu’à l ’exécution

- 2 formes
  - Dynamic SQL
    - utilisation de sections (proche d’Embedded SQL)

      nécessite un précompilateur
  - SQL/CLI
    - API bas niveau pour les applications

Dynamic SQL

- Requêtes SQL
  - chaîne de caractères décrivant la requête SQL
  - la requête est analysée (PREPARE) à l 'exécution puis exécutée (EXECUTE)
    - Erreur de Syntaxe
    - + Absence de la source de données
    + Génération automatique de requêtes

Exemple en C

```
EXEC SQL BEGIN DECLARE SECTION;
    char requete[MAX_QUERY_LENGTH];
EXEC SQL END DECLARE SECTION;
while(1) {          printf("\nNouvelle requête:"); scanf("%s",requete);
                    EXEC SQL PREPARE q FROM : requete;
                    EXEC SQL EXECUTE q;
                    printf("\nNouvelle requête:"); scanf("%s",requete);
                    EXEC SQL EXECUTE IMMEDIATE : requete;
}
```

L ’interface SQL/CLI

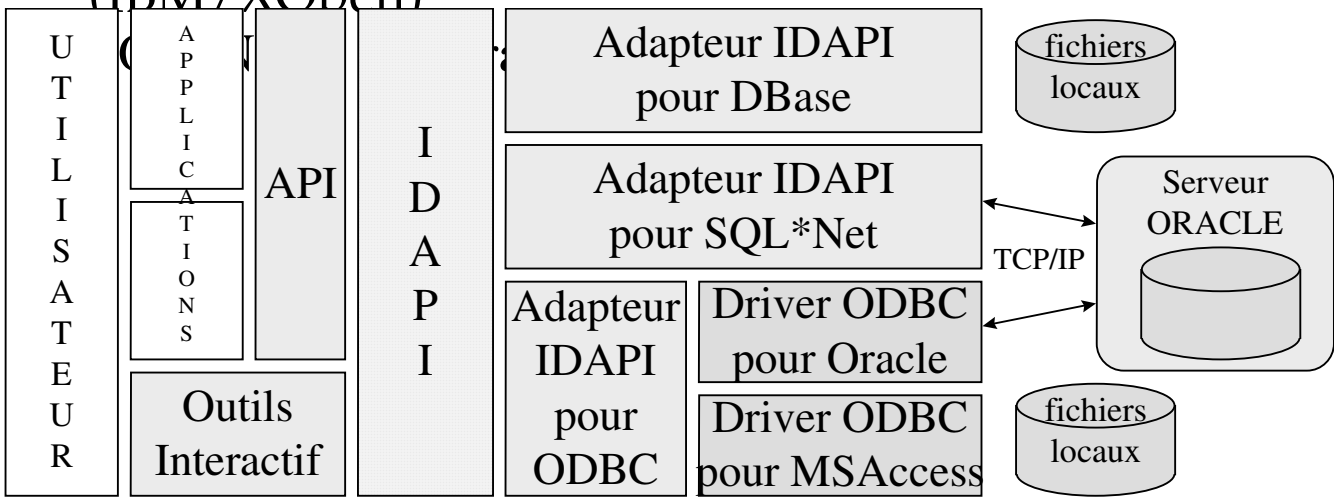
- CLI : Call Level Interface
  - API normalisé par l ’ANSI

- Evolution dans SQL3, dans Oracle 8, ...
  - prise en compte des extensions Objet

Les Middlewares SQL/CLI

Plusieurs Offres

- ODBC Open DataBase Connectivity (MicroSoft) *SQL/CLI*
- JDBC Java DataBase Connectivity (JavaSoft) *SQL/CLI*
- IDAPI Integrated Database Application Interface (Borland)
- DAL Data Access Language (DEC/Apple)
- DRDA Distributed Remote Database Access (IBM/XOpen)



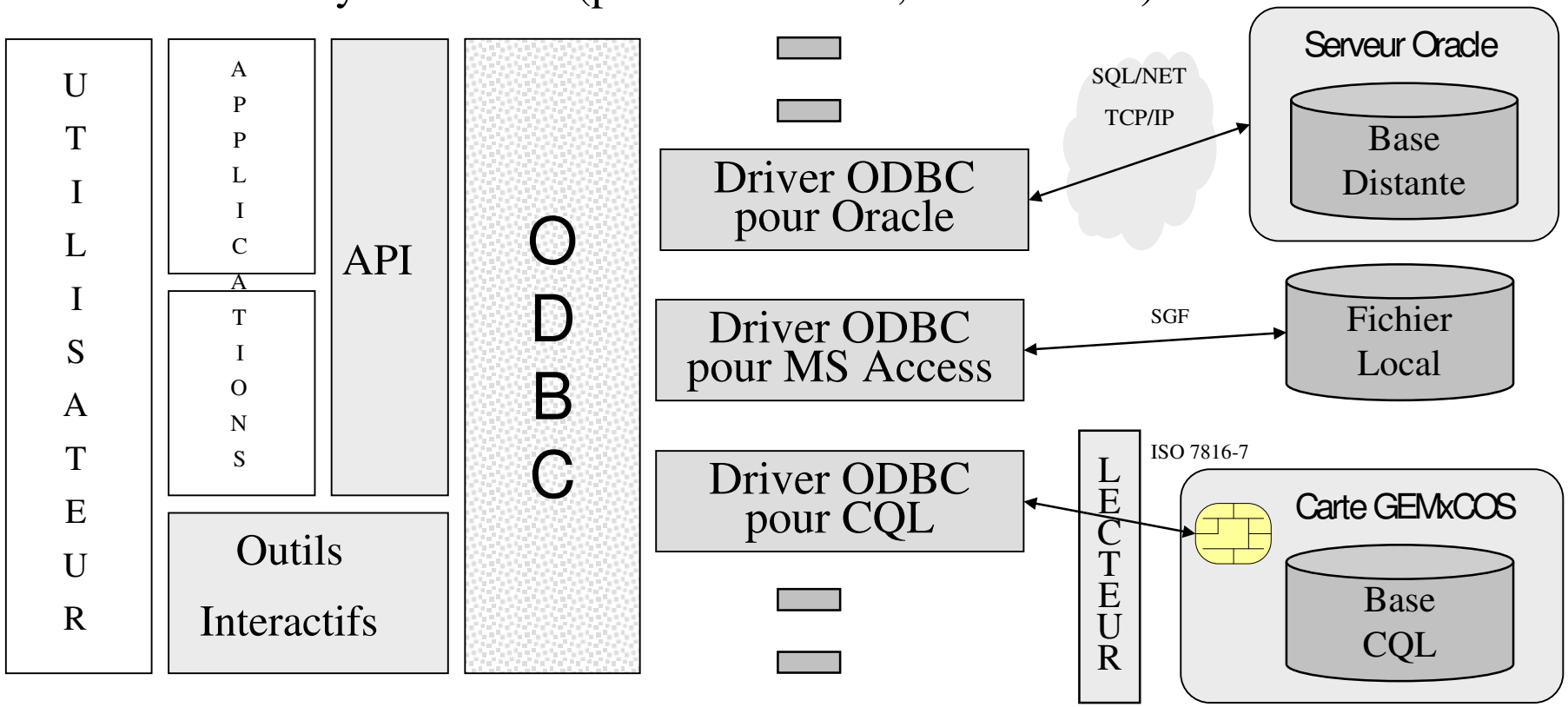
ODBC *Open DataBase Connectivity*

- Objectif
  - Le "MiddleWare" offre une interface unique d'accéder aux données quelque soit le format, la localisation, ...
  - Indépendance Application / Serveurs BD
    (MultiVendeurs)
- Principes
  - abstractions/concepts de SQL/CLI (ISO et ANSI)
  - API MS Windows pour manipuler des Tables SQL
    - dans des fichiers locaux
    - servies par des serveurs SGBD Relationnels
      en mode Client/Serveur
  - plusieurs sources accessibles simultanément

Architecture d 'ODBC

Architecture Modulaire

- utilisant des drivers d 'adaptation
  - au format de la source de données
  - au moyen d 'accès (protocole réseau, fichier local)



Configuration des sources ODBC

MicroSoft ADO *ActiveX Data Objet*

- Composant ActiveX

ADO dans un ASP (i)

Parcours d ’une table avec un curseur

```
<%@ LANGUAGE = JScript %>
<!--#include file="adojavas.inc"-->
<HTML><HEAD><TITLE>Requête Simple</TITLE></HEAD><BODY>
<h1>Liste des Auteurs<h1><hr>
<%
  var curDir = Server.MapPath("\\iissamples\\sdk\\asp\\database\\authors.mdb");
  var oConn = Server.CreateObject("ADODB.Connection");
  oConn.Open("DBQ="+curDir+";Driver={Microsoft Access Driver (*.mdb)};"
             +"DriverId=25;FIL=MS Access;")
  var oRs = oConn.Execute("SELECT * From authors");
%>
<TABLE border = 1>
<%  while (!oRs.eof) { %>
<tr>
<%    for(Index=0; Index < (oRs.fields.count); Index++) { %>
             <TD VAlign=top><% = oRs(Index)%></TD>
<%    } %>
</tr>
<%    oRs.MoveNext();
   } %>
</TABLE>
<%  oRs.close(); oConn.close(); %>
</BODY></HTML>
```

ADO dans un ASP (ii)

Ajout et Suppression d 'une ligne

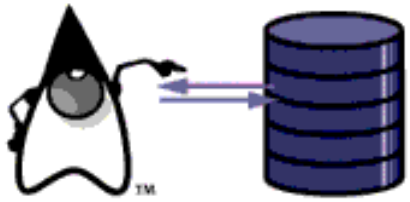
```
...
<h1> Suppression et Ajout d 'un auteur<h1><hr>
<% …
  oRs = Server.CreateObject("ADODB.Recordset");
  oRs.ActiveConnection = oConn;
  oRs.Source = "SELECT * FROM authors Where 1=2";
  oRs.CursorType = adOpenStatic; oRs.LockType = adLockOptimistic;
  oRs.Open(); oRs.Addnew();
  oRs("Author").Value = "John Steinbeck";
  oRs("YearBorn").Value = 1902;
  oRs.Update();
  Response.Write("<p>Auteur Inseré: "+oRs("Author")+"," + oRs("YearBorn")); oRs.Close();

  oRs = Server.CreateObject("ADODB.Recordset");
  oRs.ActiveConnection = oConn;
  oRs.Source = "SELECT * FROM authors WHERE YearBorn=1902 and Author='John Steinbecl
  oRs.CursorType = adOpenForwardOnly;   oRs.LockType = adLockOptimistic;
  oRs.Open(); oRs.Delete(); oRs.Update();
  Response.Write("<p>Auteur Supprimé: Paul Enfield, 1967");      oRs.Close();
%>
</BODY></HTML>
```

ADO dans un ASP (iii)

Invocation d 'une Procédure Stockée

```
<%@ LANGUAGE = JScript %>
<!--#include file="adojavas.inc"-->
<HTML><HEAD><TITLE> Invocation d 'une procédure stocké </TITLE></HEAD>
<BODY>
<h1>Invocation d 'une procédure stocké<h1><hr>
<%
  var oConn = Server.CreateObject("ADODB.Connection");
  var oCmd = Server.CreateObject("ADODB.Command");
  oConn.Open("DSN=LocalServer;UID=sa;PWD=;DATABASE=pubs");
  oCmd.ActiveConnection = oConn;
  oCmd.CommandText = "{call byroyalty(?)}";
  oCmd.Parameters.Append(o
            Cmd.CreateParameter("@Percentage", adInteger, adParamInput));
  oCmd("@Percentage") = 75;
  var oRs = oCmd.Execute();
%>
ID de l'auteur = <% Response.Write(oRs("au_id")) %><BR>
</BODY></HTML>
```

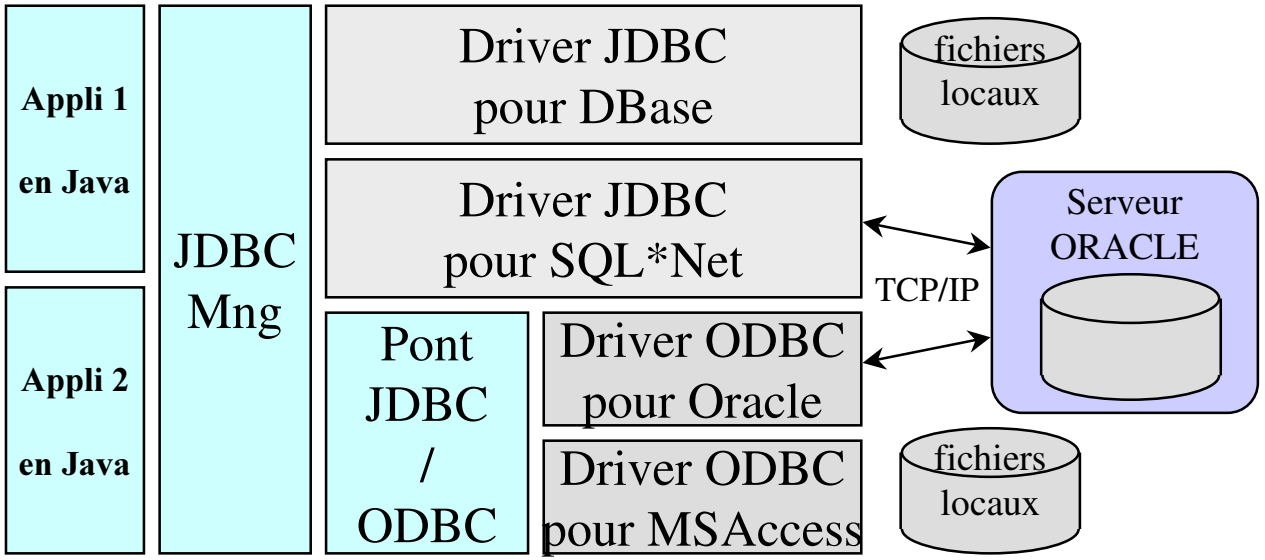
JDBC
Java DataBase Connectivity



- Motivations
  - API Java pour manipuler des Tables SQL
    - dans des fichiers locaux
    - servies par un serveur BD
  - une seule API uniforme
    pour tous les SGBDs (relationnels)
    - abstractions/concepts de X/Open SQL Call Level Intf
    - Même principe que les Middlewares comme ODBC

Architecture JDBC

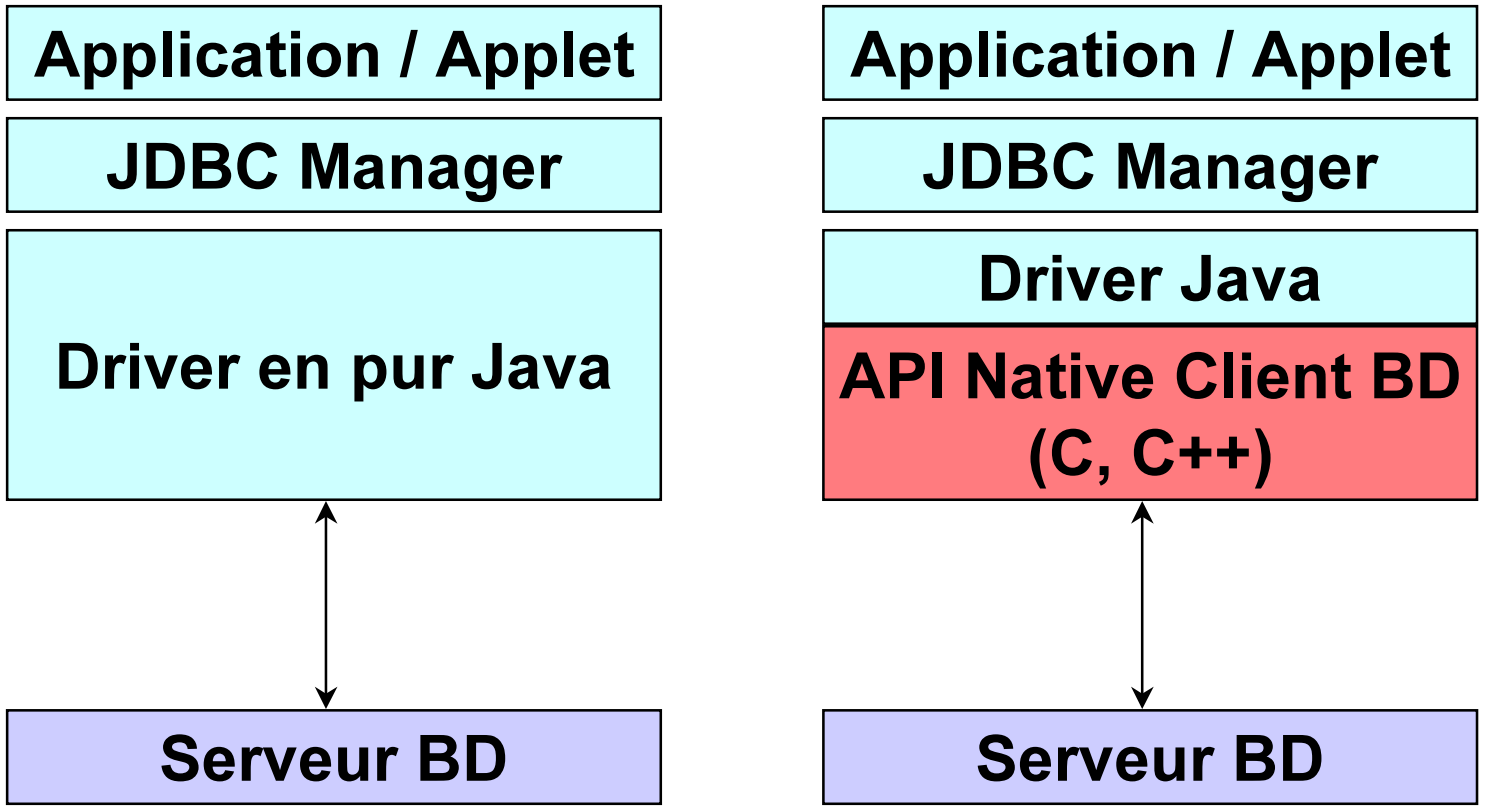
- Composants d'adaptation (Drivers)
  - un driver pour chaque SGBD (Oracle, Sybase, ...)
  - un driver pour chaque format de fichier (Dbase,Paradox)



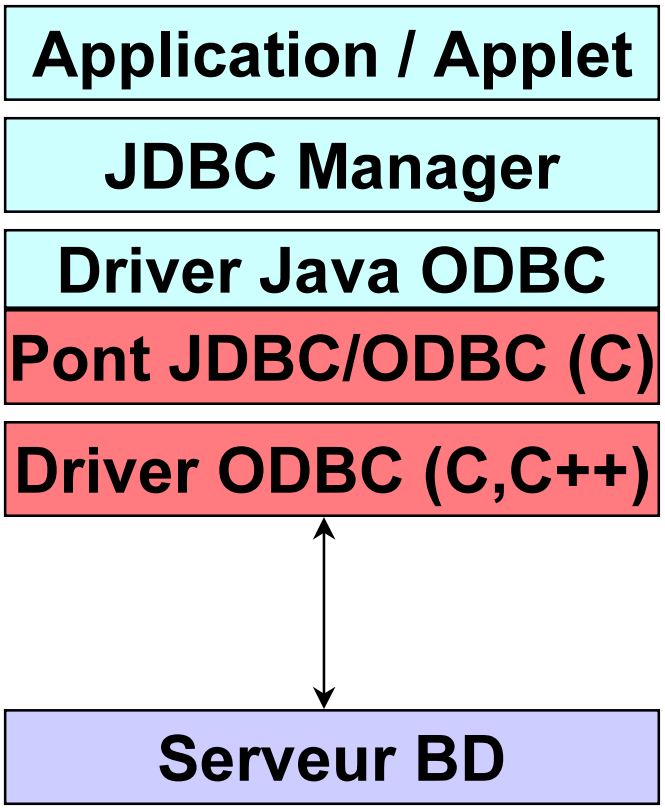
Drivers JDBC

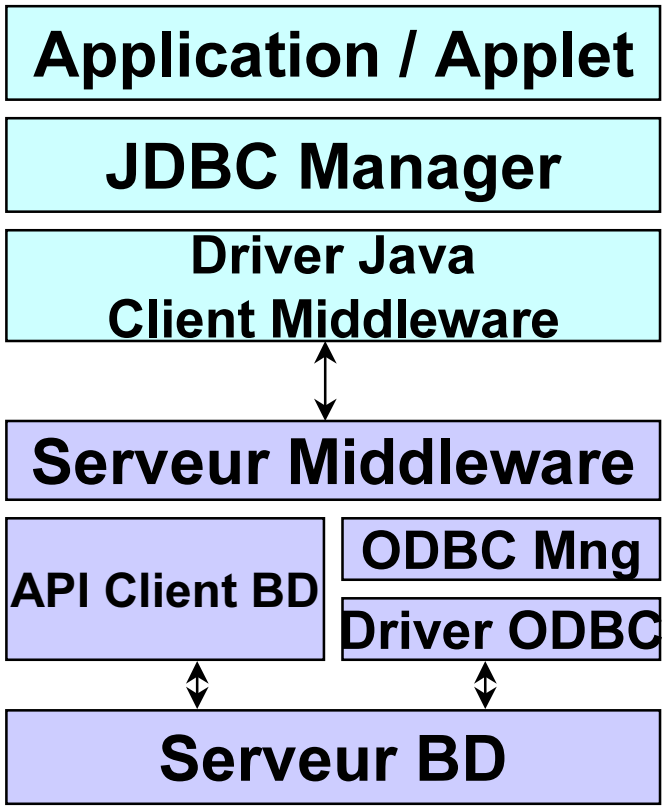
- 4 types de drivers JDBC
  - pur Java
    + DONC téléchargeable
  - natif
    - installation du driver sur le client
  - pont JDBC / ODBC
    - installation d'ODBC et des drivers sur le client
    + utilisation des drivers ODBC existants
  - serveur Middleware
    - encombrement serveur
    + protocole autre que TCP/IP

Drivers JDBC : pur Java vs Natif



Drivers JDBC : pont ODBC vs Middleware





Package `java.sql`

- Interfaces
  - DriverManager
  - Connection, PooledConnection, XAConnection
  - Statement
    - PreparedStatement
    - CallableStatement
  - ResultSet
  - ResultSetMetaData
  - DatabaseMetaData
  - RowSet
  - SQLData, SQLInput, SQLOutput
- Classes
  - DriverManager, DriverPropertyInfo
  - Date, Time, TimeStamp, Types, Struct,
  - *SQLException, SQLWarning, DataTruncation*

Connexion JDBC

- classe java.sql.Connection
- URL d'une source de données (relationnelle)

  jdbc:<subprotocol>:<subname>

  jdbc:dcenaming:accounts-payable

  jdbc:dbnet://dbms.mycomp.com:356/ecom

  jdbc:msql://dbsv.acme.com/user

  jdbc:oracle:thin:@enee:1521:ISTV

  jdbc:odbc:ECOM

  jdbc:odbc:ECOM;CacheSize=20;ExtensionCase=LOWER

  jdbc:odbc:ECOM;UID=admin;PWD=nimda

  jdbc:GemDBJdbc:/1F00/1F10

  ...
- Extension JDBC3.0
  - aux fichiers plats et aux feuilles de calcul

JDBC *« Curseur »*

```
class Employe {
   public static void main (String args [] )
                  throws SQLException, ClassNotFoundException { try {
   Class.forName ("oracle.jdbc.driver.OracleDriver");
//  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   String dburl = "jdbc:oracle:oci8:@";
   Connection conn = DriverManager.getConnection(dburl, "toto", "passemot");
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery
                    ("SELECT numemp, name, salary FROM EMPLOYE");
   while (rs.next()) {
     String s = rs.getString(2);
     float f = rs.getFloat("salary");
     System.out.println (s + " gagne "+ f +" $");
   } rs.close();
} catch(Exception e) { e.printStackTrace(); } }
```

JDBC *valeur nulle*

Rappel

- une valeur nulle (NULL) en SQL signifie :
- valeur inconnue, valeur non renseignée, ...

```
ResultSet rs = stmt.executeQuery
                ("SELECT numemp, name, salary FROM EMPLOYE");
  while (rs.next()) { String s = rs.getString(2);
    float f = rs.getFloat("salary");
    if(rs.wasNull())     System.out.println (s + " n"a pas de salaire");
                 else    System.out.println (s + " gagne "+ f +" $");
  } rs.close();
```

Remarque

- la méthode isNull() de JDBC0.5 permettait

```
if(rs.isNull(3)) System.out.println (s + " n"a pas de salaire");
                 else    System.out.println (s + " gagne "+ rs.getFloat(3) +" $");
```

- cependant l 'implantation étant difficile, elle a été abandonnée

JDBC

passage d 'arguments, modification, transaction

```
class Employe {
   ...
   public static int updateEmploye (int num, String nom )
                throws SQLException, ClassNotFoundException { try {
   Class.forName ("oracle.jdbc.driver.OracleDriver");
   String dburl = "jdbc:oracle:oci8:@";
   Connection conn = DriverManager.getConnection(dburl, "toto", "passemot");
   conn.setAutoCommit(false);
  PreparedStatement pstmt = conn.prepareStatement(
              "UPDATE Employe SET salary = ?, name = ? WHERE numemp = ?");
  // psmt.clearParameters();
   psmt.setNull(1); psmt.setString(2,nom); psmt.setInt(3, num);
   int nbLignesModifiees = psmt.executeUpdate();
   if(nbLignesModifiees==1) conn.commit(); else conn.rollback();
} catch(Exception e) {
   e.printStackTrace(); } }
```

JDBC *appel d 'une procédure stockée*

```
class Employe {
   ...
   public static int payraise (Connection conn, int num)
                throws SQLException { try {
  CallableStatement cstmt = conn.prepareCall( "{call sp_payraise(?,?)}");
   cstmt.registerOutParameter(2, java.sql.Types.INT);
   psmt.setInt(1, num);
   csmt.execute();
   return cstmt.getInt(2);
} catch(Exception e) {
   e.printStackTrace(); } }
```

JDBC

Streams ASCII et Binaires

```
class Employe {
   public static void info (Connection conn, int num)
                throws SQLException { try {
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery(
       "SELECT name, photo, cv FROM EMPLOYE");
   while (rs.next()) {
     String s = rs.getString(1);
     BufferedReader cv = new BufferedReader(
                                  new InputStreamReader(rs.getAsciiStream("cv")));
     while(cv.ready() { out.println(cv.readline());}
     BufferedInputStream gifData = new BufferedInputStream (
                                  new BinaryInputStream(rs.getBinaryStream("photo")));
     while(len = gifData.read(buf, 0, buf.length) != -1){
        ...
     }
```

JDBC *ResultSetMetaData*

```
class HTMLResultSet { // Livre Servlet Ex9.2 p 253
    private ResultSet rs;
    public HTMLResultSet (ResultSet rs ) { this.rs=rs;}
    public String toString() {
        StringBuffer out = new StringBuffer(); out.append("<TABLE>");
        ResultSetMetaData rsmd=rs.getMetaData();
        int numcols=rsmd.getColumnCount();
        out.append("<TR>");
        for(int i=0;i<numcols;i++) {
            out.append("<TH>").append(rsmd.getColumnLabel(i));
        }
        out.append("</TR>");
        ...
```

JDBC

getObject

```
... // suite de HTMLResultSet.toString()

        while (rs.next()) {
            out.append("<TR>");
            for(int i=0;i<numcols;i++) {
                out.append("<TH>");
                Object obj=rs.getObject(i);
                out.append((obj==null)?" ":obj.toString());
                out.append("</TH>");
            }
            out.append("</TR>")
        }
        out.append("</TABLE>");
...
```

JDBC *getObject*

```
ResultSet rs = stmt.executeQuery
                ("SELECT numemp, name, salary FROM EMPLOYE");
...
while (rs.next()) {
   String s = rs.getString(2);
   float f = rs.getFloat("salary");
   if(rs.wasNull())  ... else ...;  // wasNull() teste si la dernière valeur est NULL
}
...
while (rs.next()) { // alternative avec getObject()
   Object obj;
   obj = rs.getObject(2); String s = obj.toString();
   obj = rs.getObject("salary");
   if(obj==null) ... else {     // teste si la valeur est NULL
       Float f = (Float)obj; ... // peut lever CastException
} }
```

Correspondance de type SQL-Java (i)

Type SQL	Type Java	Type Java retourné par getObject()	Méthode recommandée au lieu de getObject()
NUMERIC	java.Math.BigDecimal	java.Math.BigDecimal	java.Math.BigDecimal getBigDecimal()
DECIMAL	java.Math.BigDecimal	java.Math.BigDecimal	java.Math.BigDecimal getBigDecimal()
BIT	boolean	Boolean	boolean getBoolean()
TINYINT	byte	Integer	byte getByte()
SMALLINT	short	Integer	short getShort()
INTEGER	integer	Integer	integer getInt()
BIGINT	long	Long	long getLong()
REAL	float	Float	float getFloat()
FLOAT	double	Double	double getDouble()
DOUBLE	double	Double	double getDouble()

Doit être utilisé pour les valeurs monétaires !! (plutôt que Float)

Correspondance de type SQL-Java (ii)

Type SQL	Type Java	Type Java retourné par getObject()	Méthode recommandée au lieu de getObject()
CHAR	String	String	String getString()
VARCHAR	String	String	String getString()
LONGVARCHAR	String	String	InputStream getAsciiStream() InputStream getUnicodeStream()
BINARY	byte[]	byte[]	byte[] getBytes()
VARBINARY	byte[]	byte[]	byte[] getBytes()
LONGVARBINARY	byte[]	byte[]	InputStream getBinaryStream()
DATE	java.sql.Date	java.sql.Date	java.sql.Date getDate()
TIME	java.sql.Time	java.sql.Time	java.sql.Time getTime()
TIMESTAMP	java.sql.Timestamp	java.sql.Timestamp	java.sql.Timestamp getTimestamp()

Nouveaux types dans JDBC3.0

- BOOLEAN
- DATALINK
  - Prise en chaque de données externes au SGBD
    - (fichier d'images, …)

Les exceptions dans JDBC

SQLException, SQLWarning, DataTruncation

```
...
try {
  PreparedStatement pstmt = conn.prepareStatement(
            "UPDATE Employe SET salary = ?, name = ? WHERE numemp = ?");
  psmt.setNull(1); psmt.setString(2,nom); psmt.setInt(3, num);
  nbLignesModifiees = psmt.executeUpdate();
}
catch(DataTruncation e) {
   // Des données ont été tronquées : on décide de ne rien faire
}
catch(SQLException e) {
   System.out.println(e.getMessage());
   while((e = e.getNextException()) != null) {
     System.out.println(e.getMessage());
    }
}
```

Evolution de JDBC

- JDBC 1.0 (Janvier 1996)
- JDBC 2.0 (Mars 1998)
- JDBC 3.0 (Septembre 2000)

- Extensions de JDBC 2.x CORE et SE (Standard Exception)
  - Parcours avant-arrière d 'un ResultSet
  - Mise à jour depuis un ResultSet
  - Batch de plusieurs ordres
  - SQL3 datatypes
    - les types étendus UDT (SQL_DATA) et références REF
    - les types longs (BLOB, CLOB) et les ARRAY
  - Validation à deux phases des transactions (XA)
  - Pool de connexion, Cache de lignes sur le client
  - RowSet (Composants JavaBean), …
  - DataSource et JNDI

Echapement SQL dans les Statements

```
Statement stmt = con.createStatement();
stmt.setEscapeProcessing(true);
ResultSet rs;
// echapement de wildcard SQL _ et %
rs = stmt.executeQuery("SELECT numemp FROM EMPLOYE "
    + "WHERE name LIKE '\\_%' {escape '\\' }");
// appel à des fonctions du SGBD
rs = stmt.executeQuery("SELECT numemp FROM EMPLOYE "
    + "WHERE name = {fn user() }");
// jointure externe
rs = stmt.executeQuery("SELECT numemp, SUM(amount) "
    + "FROM {oj EMPLOYE LEFT OUTER JOIN SALE USING (numemp)}");
 // appel de procédure stockée
    {? = call procedure_name[(?,?,? ..., ?)]}
    {call procedure_name[(?,?,? ..., ?)]}
```

Positionnement dans un ResultSet

- Méthodes : first , last , beforeFirst , afterLast, absolute, previous , relative , moveToCurrentRow

```
Statement stmt = con.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATEABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM Employe");

rs.first();
rs.updateFloat("Salary", 10000.00f);        rs.updateRow();
rs.absolute(2);
Float sal=rs.getFloat("Salary", sal);
rs.updateFloat(3, sal*1.10);                rs.updateRow();
rs.relative(-1); // rs.previous();
rs.updateFloat("Salary", 11000.00f);        rs.updateRow();
```

Mise à jour depuis Java

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_UPDATEABLE);
ResultSet uprs = stmt.executeQuery(
    "UPDATE Employe SET Salary=20000 WHERE NumEmp=102");
uprs.last();
uprs.updateFloat("Salary", 20000);
uprs.cancelRowUpdates();

uprs.updateFloat("Salary", 15000);
uprs.updateRow();
```

Insertion et suppression depuis Java

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATEABLE);
ResultSet uprs = stmt.executeQuery("SELECT * FROM Employe");

uprs.moveToInsertRow(); /* autres méthodes  : first , last , beforeFirst, afterLast
                          , absolute, previous , relative , moveToCurrentRow */
uprs.updateString("Name", "Joseph");
uprs.updateInt(1, 100);
uprs.updateFloat("Salary", 12000.00f);
uprs.insertRow();

uprs.last();
uprs.deleteRow();
```

Batch de plusieurs ordres

```
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.addBatch("INSERT INTO Employe VALUES(100, 'Jacques', 40000.00f)");
stmt.addBatch("INSERT INTO Employe VALUES(101, 'Paul', 10000.00f)");
stmt.addBatch("INSERT INTO Employe VALUES(102, 'Marie', NULL)");
int [] updateCounts = stmt.executeBatch();

PreparedStatement pstmt = con.preparedStatement(
    "INSERT INTO Employe VALUES(?, ?, ?)");
pstmt.setInt(1,103); pstmt.setString(2,'Pierre'); pstmt.setFloat(3,30000.00f);
pstmt.addBatch();
pstmt.setInt(1,104); pstmt.setString(2, 'Madeleine'); pstmt.setNull(3);
pstmt.addBatch();
int [] pupdateCounts = pstmt.executeBatch();
con.commit();
```

Interface RowSet

- Composants JavaBean
  pour des sources de données tabulaires
  - étend ResultSet
  - Modèle d 'événement JavaBean
    addRowSetListener, removeRowSetListener
    peut être connecté à un bean PieChart par exemple

Implémentations possibles

- JDBCRowSet (JDBC/TCP)
- CachedRowSet (RMI/IIOP) *utilisables par des PDA*
- WebRowSet (HTTP/XML)

  - *voir [White et al] p231, 609, 681*

DataSource

- Motivation
  - rendre les programmes indépendants des sources de données
- Datasource
  - une instance regroupe les informations de connexion (driver, dburl, user, password)
  - puis est enregistrée dans un service de répertoire via JNDI
  - puis récupérée via JNDI

```
Context ctx=new InitialContext();
DataSource ds=(DataSource) ctx.lookup("jdbc/EmployeeDB");
Connection cnx=ds.getConnection();
...
con.close();
```

- Sous-classes
  - XADataSource, ConnectionPoolDataSource

Pool de Connexions JDBC

- Motivation
  - réutiliser les connexions JDBC entre plusieurs threads (servlets, entity beans, …) au lieu de les créer puis de les clore
    - impact important sur les performances d 'un serveur Servlet,
- Classes
  - PooledConnection
    - représente une des connexions gérées par le pool
  - ConnectionEventListener
    - permet de notifier les opérations effectuées sur le PooledConnection
  - ConnectionPoolDataSource
    - permet de récupérer un PooledConnection via JNDI

      PoolConnection getPooledConnection()

Autres interfaces

- Array
  - représente un tableau SQL
    - **Object getArray(), ResultSet getResultSet(), ...**
- Struct
  - représente une structure SQL
    - **Object[] getAttributes(), Object[] getAttributes(Map m),String getSQLNameType()**
- Ref
  - représente une référence à une structure SQL
- SQLInput et SQLOutput
  - représentent un type user-defined sous la forme d 'un flot (entrée-sortie)
    - **int readInt()/ void writeInt(int attr), ...**
- SQLData
  - assure une correspondance personnalisable pour les types user-defined

Java-Aware Database

getObject() et setObject()

- SGBDOO et JDBMS

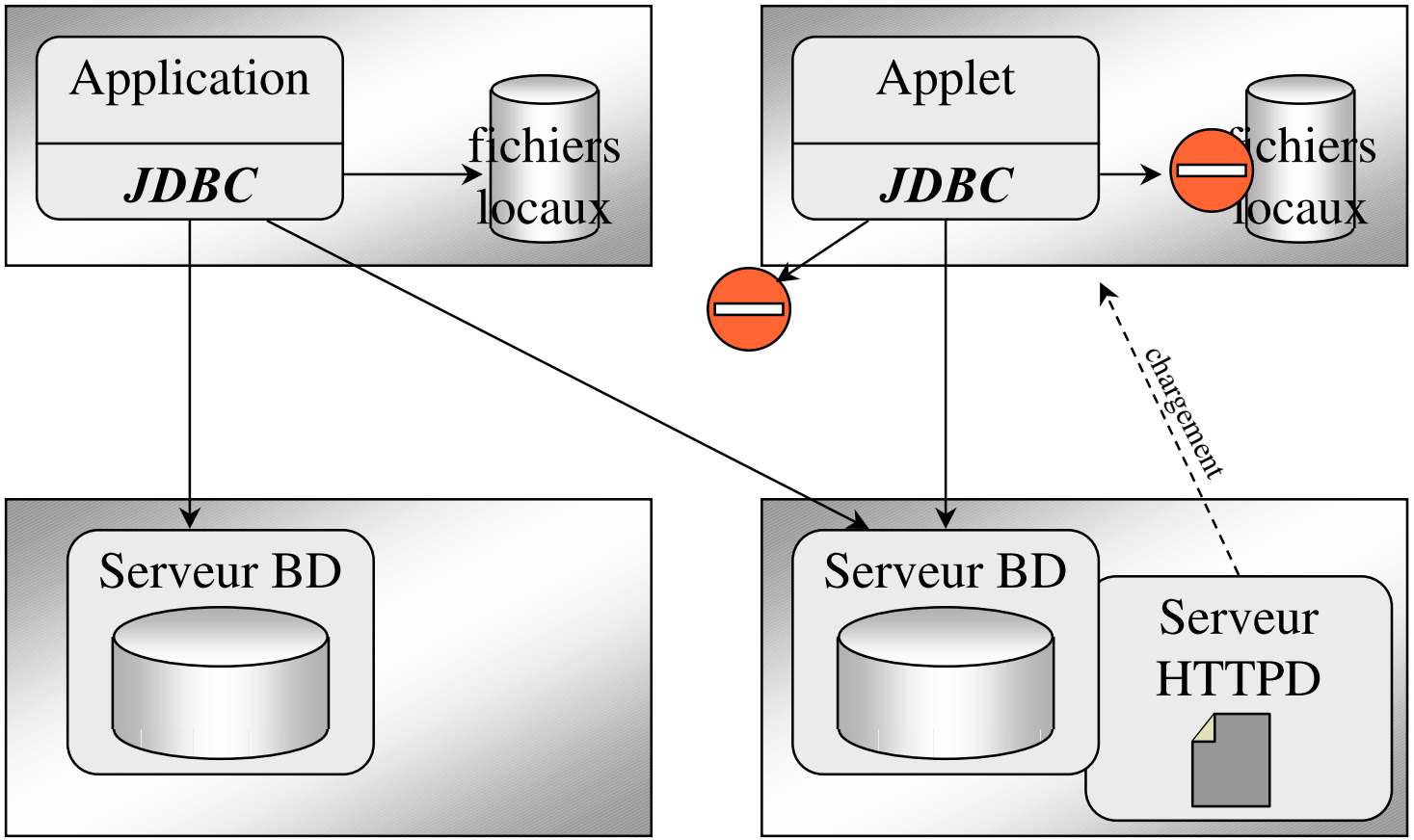
```
ResultSet rs = stmt.executeQuery("SELECT * FROM Employe");
rs.next();
Employe e = (Employe) rs.getObject(1);
...
Employe m=new Employe(110,"Mathieu");
PreparedStatement pstmt= con.preparedStatement(
    "INSERT INTO Employe (Employe_t) VALUE (?)");
pstmt.setObject(1,m);
pstmt.executeUpdate();
// Remarque: le bytecode n 'est pas stocké : il faut utiliser
    Class.forName()
```

Principe de Sécurité de JDBC

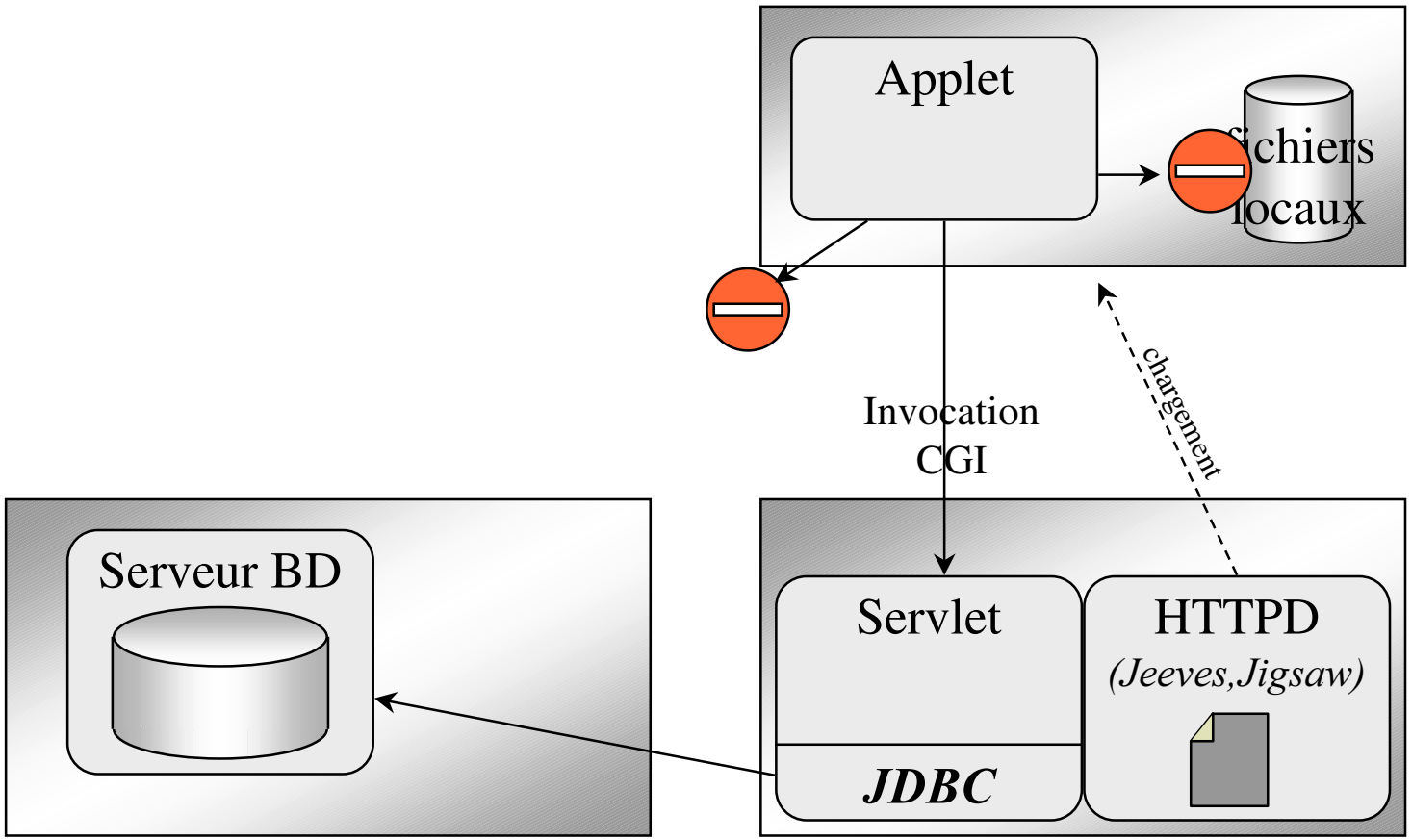
- respecte les principes Sécurité de Java
  - JDK 1.0 et 1.1 / en changement avec JDK1.2
- Application / Thrusted Applets
  - *bases locales*
  - *serveur BD*
- Untrusted Applets / Untrusted JDBC Driver
  - *connexion au serveur BD*

    si = @ du site de chargement

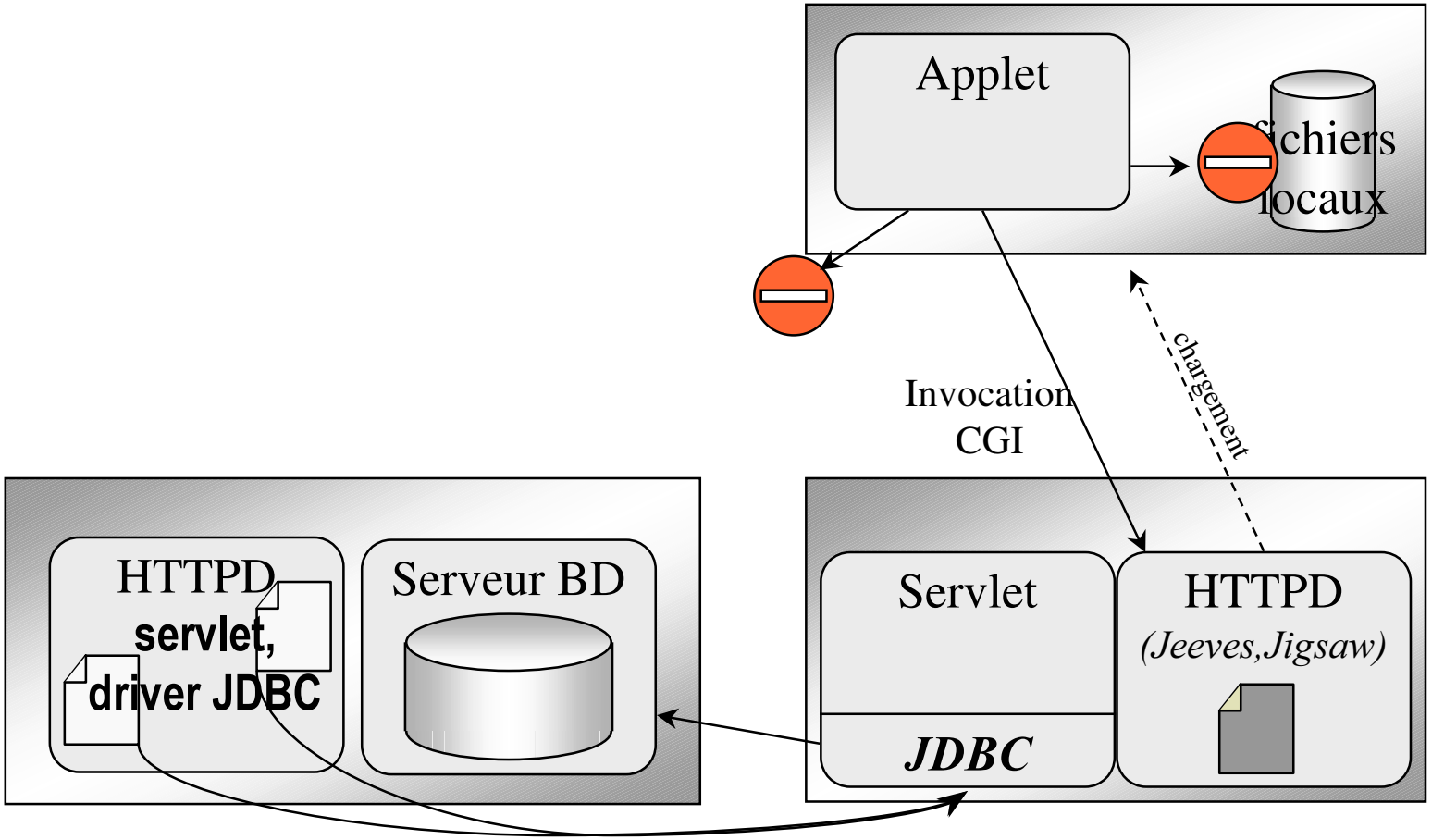
Utilisation : Application et Applets



Utilisation : Servlets (i)



Utilisation : Servlets (ii)



JDBC et ANT

- Tâche ANT pour envoyer des ordres SQL
  - Via un driver JDBC
- Exemple avec McKoi

```
<echo>Query tables ...</echo>
<sql
  driver="${driver}" url="${url}" userid="${userid}"  password="${password}"
  print="yes" classpathref="sqldriver.path">
<transaction><![CDATA[
                SELECT * FROM AirCraft;
                SELECT * FROM Flight;
]]></transaction>
<transaction><![CDATA[
                SELECT * FROM Ticket;
                SELECT * FROM Customer;
]]></transaction></sql>
```

Le module DBI de PERL (i)

Accès aux BDs par des scripts PERL

http://www.hermetica.com/technologia/DBI/

```
use strict; use DBI;
my $dbh = DBI->connect( 'dbi:Oracle:orcl', 'toto','passedemot',
                  { RaiseError => 1, AutoCommit => 0 }
                ) || die "Database connection not made: $DBI::errstr";
my @names = (  "Du%", "Ma%" );
my $sql = qq{ SELECT name, salary FROM employees WHERE name LIKE ? };
my $sth = $dbh->prepare( $sql );
for( @names ) {
    $sth->bind_param( 1, $_, $DBI::SQL_VARCHAR );  $sth->execute();
    my($nom, $sal ); $sth->bind_columns( undef, \$nom, \$sal );
    while( $sth->fetch() ) { print "$nom est payé $sal\n"; }
}
$sth->finish(); $dbh->disconnect();
```

Le module DBI de PERL (ii)

Transactions

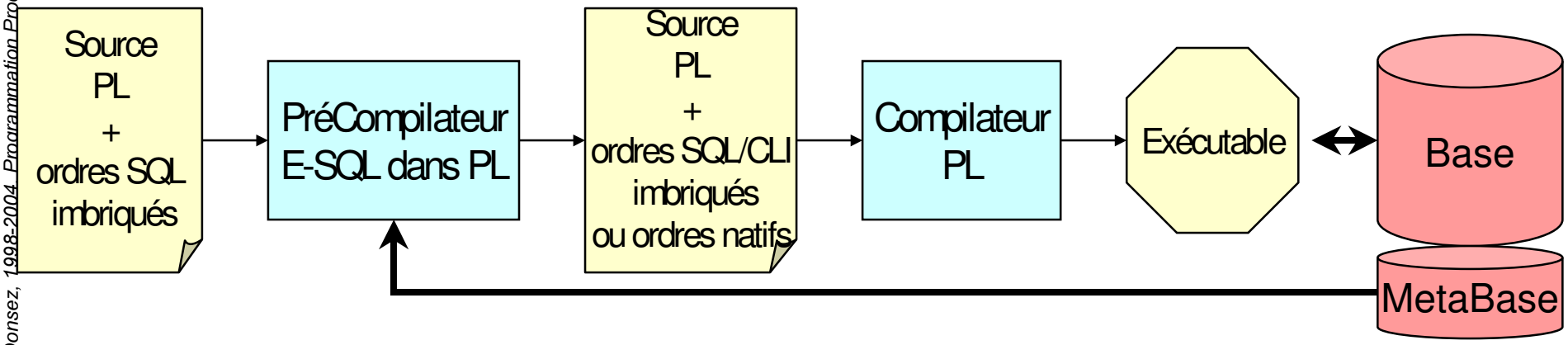
```
use strict; use DBI;
my $dbh = DBI->connect( 'dbi:Oracle:orcl', 'jeffrey','jeffspassword',
                    { RaiseError => 1, AutoCommit => 0 }
                  ) || die "Database connection not made: $DBI::errstr";
my @records = (
          [ 0, "Dupont", 10000 ], [ 1, "Durand", 20000 ], [ 2, "Martin", 25000 ] );
my $sql = qq{ INSERT INTO employees VALUES ( ?, ?, ? ) };
my $sth = $dbh->prepare( $sql );
for( @records ) {
    eval {
          $sth->bind_param( 1, @$_->[0], $DBI::SQL_INTEGER );
          $sth->bind_param( 2, @$_->[1], $DBI::SQL_VARCHAR );
          $sth->bind_param( 3, @$_->[2], $DBI:: SQL_INTEGER );
          $sth->execute(); $dbh->commit();
        };
    if( $@ ) {    warn "Database error: $DBI::errstr\n"; $dbh->rollback();  }
}
$sth->finish(); $dbh->disconnect();
```

Embedded SQL

- Motivations
- Oracle Pro*C
- Informix ESQL
- SQLJ

Motivation

- Syntaxe plus concise que SQL/CLI
- Analyse statique
  - Contrôle de la Syntaxe et du Typage
  - Typage curseur dépendant de la Métabase
- Précompilation



Embedded SQL

- Sections spéciales
  - **EXEC SQL BEGIN/END DECLARE SECTION**
    - pour les déclarations de variables 3GL partagées avec SQL
  - **EXEC SQL SELECT ... INTO ...**
    - pour l 'exécution d 'une requête SQL
- Précompilateurs
  - C
    - Informix ESQL/C
    - Oracle Pro*C
  - Java
    - SQLJ *(Oracle, Tandem, IBM, Sybase)*
    - Java Relational Binding *(Ardent Software)*

Exemple Pro*C

```
EXEC SQL INCLUDE SQLCA; /* manipulation des erreurs */
EXEC SQL BEGIN DECLARE SECTION;
char nom[21]; float salaire;
EXEC SQL END DECLARE SECTION;
    scanf("%s",nom);
EXEC SQL EXECUTE
    SELECT salary INTO :salaire
      FROM Employe WHERE name = :nom;
END-EXEC;
if (sqlca.sqlcode != 0) /* sqlerrmc message d 'erreur / sqlerrml sa longueur */
    printf(" Erreur d"execution.\n %.70s\n",
                                sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
else
    printf ("%s gagne %d $\n", nom, salaire);
```

Exemple de Curseur en Pro*C

```
EXEC SQL BEGIN DECLARE SECTION;
   char nom[21];      float salaire;
EXEC SQL END DECLARE SECTION;
. . .
EXEC SQL DECLARE c CURSOR FOR
         SELECT name, salary FROM Employe WHERE salary > 10000;
EXEC SQL OPEN CURSOR c;
while(1) {
   EXEC SQL FETCH c  INTO :nom, :salaire;
   if(NOT FOUND) break else printf ("%s gagne %d $\n", nom, salaire);
}
EXEC SQL CLOSE CURSOR c;
```

Informix ESQL/C

```
EXEC SQL DECLARE cursemp CURSOR FOR
                SELECT name, salary INTO :nom,  :sal:null_flag
                FROM Employe FROM items
   FOR READ ONLY;
EXEC SQL OPEN cursemp;
   while(SQLCODE == 0) {
     EXEC SQL FETCH cursemp;
     if(SQLCODE == 0)
                 if (null_flag < 0) printf("%d gagne rien\n", nom)
                else                printf("%s gagne %d$\n", nom, sal);
   }
EXEC SQL CLOSE cursemp;
```

QLJ - SQL Java

- *proposé par Oracle, IBM, Sybase, Tandem à l 'ISO/ANSI*

Embedded-SQL dans Java

- vers un source Java avec des appels JDBC

Exemple

```
void print_salary (String nom) throws SQLException {
    int sal;
     #sql context cnxDRH;
    #sql { SELECT salary INTO :sal FROM Employe WHERE :nom = name };
   System.out.println( nom + " est payé " + sal + " $" );
    #sql [cnxREC] { DELETE FROM Employe }; // sur la connexion cnxREC
}
```

SQLJ - Itérateurs

- *Notion de curseur*

```
#sql public iterator IterEmp (String, int); // déclaration d 'une classe d 'itérateur

        IterEmp iter; // déclaration d 'un objet itérateur
        String nom; int sal; int c:=1;
        #sql iter = { SELECT name, salary FROM Employe };
        while (true) {
              #sql { FETCH :iter INTO :nom, :sal };
              if (iter.endFetch()) break;
              if(c++%2) System.out.println( nom + " est payé " + sal + " $" );
        }
        iter.first(); // se repositionne au premier résultat
        while(iter.next(2)) {// se positionne sur les résultats en position impaire
              System.out.println( iter.name() + " est payé " +iter.salary() + " $" );
        }
```

SQLJ - Autres

- Modification au niveau de l 'itérateur

```
IterEmp iter; // déclaration d 'un objet itérateur
#sql iter = { SELECT name, salary FROM Employe }; ...
#sql { UPDATE Employe SET salary = salary*1.1 WHERE CURRENT OF :iter };
```

- Valeurs Nulles

```
java.sql.Date d=null; int s = null;
#sql { INSERT INTO Employe (name, salary, birthday, hobby)
                    VALUES ('Dupond', :s, :d, NULL ) };
```

- Blocs et Atomicité

```
void Transfert(int x; int y; int m) {
#sql { ATOMIC BEGIN
 UPDATE Compte SET solde = solde + :m WHERE numcpt =:x;
 UPDATE Compte SET solde = solde - :m WHERE numcpt =:y;
 INSERT INTO Transferts(cptcredit, cptdebit,montant) VALUES (:x, :y, :m);
 END;}; }
```

SQLJ - Interopérabilité avec JDBC

- Parcours d ’une requête SQLJ à partir d ’un Result*Set*

```
sqlj.runtime.ResultSetIterator iter;
#sql iter = {SELECT name, salary FROM Employe };
java.sql.ResultSet rs = iter.getResultSet();
while (rs.next()) {
        String s = rs.getString(2); float f = rs.getFloat("salary");
        System.out.println (s + " gagne "+ f +" $");
  }  rs.close();
```

- Construction d ’un itérateur à partir d ’un ResultSet

```
IterEmp iter;
ResultSet rs = stmt.executeQuery("SELECT name, salary FROM Employe");
#sql iter = rs ;
while(iter.next()) {
        System.out.println( iter.name() + " est payé " +iter.salary() + " $" );
}
```

JavaBlend

- Mapping transparent
  d 'objets Java avec les lignes d 'une base relationnelle
  - Utilise l 'ODL de l 'ODMG pour la description

Extra JDBC

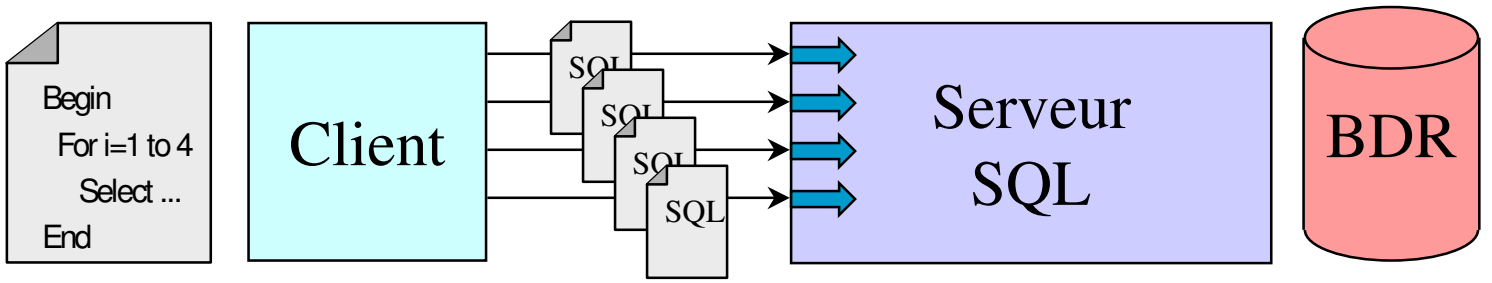
- p6spy : espion pour surveiller le « trafic » JDBC
- ObjectWeb' XAPool : pool de connexions pour JDBC

SQL « Procédural »

- Motivations
- Architecture
- SQL3 / PSM
- Informix SPL
- Oracle PL/SQL
- Un nouveau venu : Java

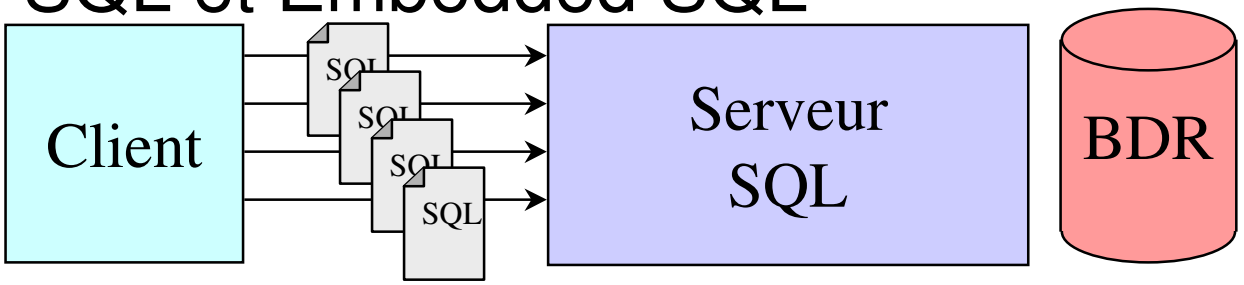
Motivations pour un SQL procédural

- Inconvénients
  - Dynamic SQL et Middleware SQL (ODBC, JDBC, ...)
    - requête vérifiée à l 'exécution (runtime)
    - typage faible variable hôte et curseur
  - Embedded SQL in 3GL
    - précompilation
- Dans les deux cas
  - « impedance mismatch »
  - code procédural du coté client (coût réseau)

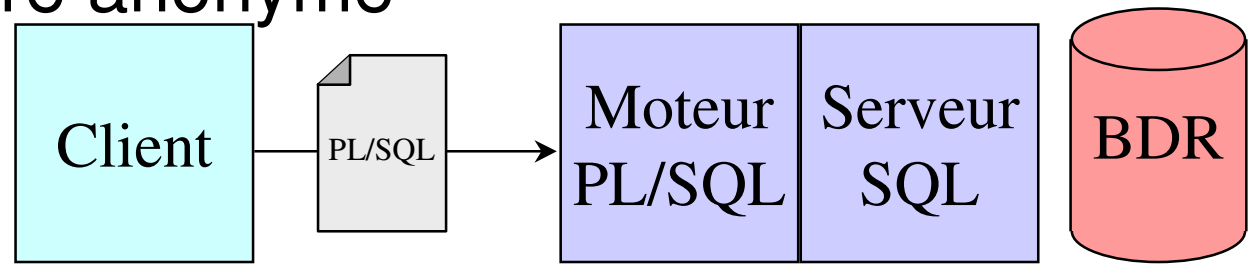


Architecture Client-Serveur

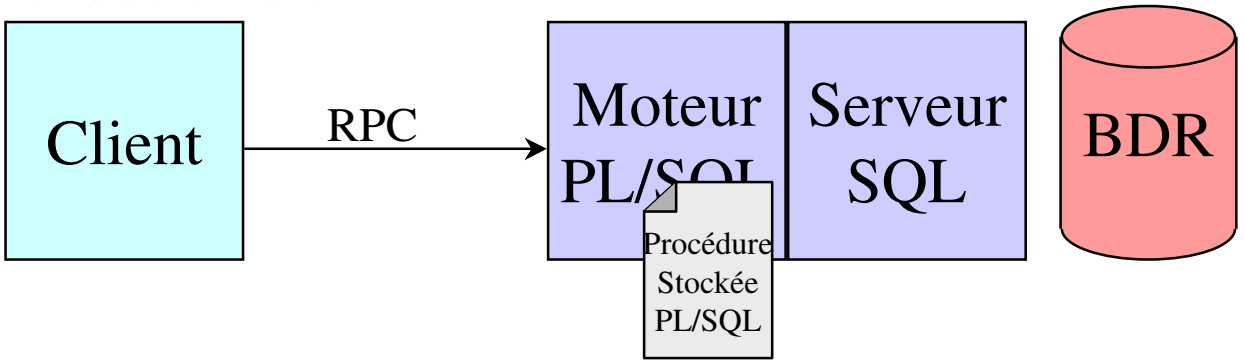
- Dynamic SQL et Embedded SQL



- Procédure anonyme

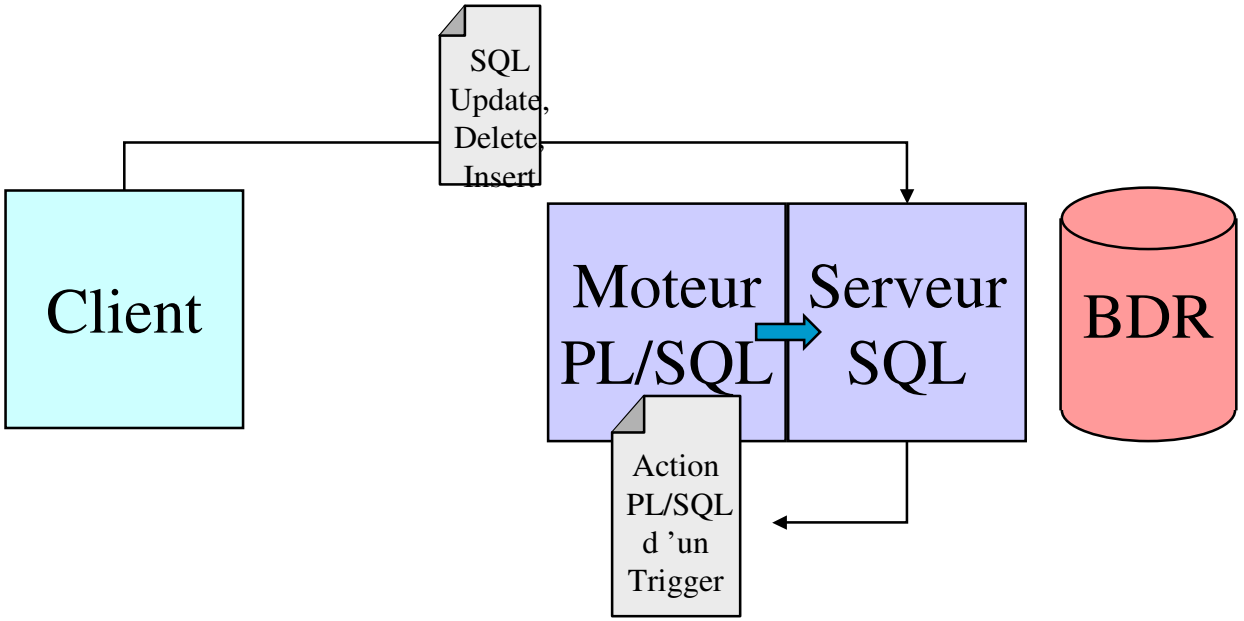


- Procédure stockée



Architecture Client-Serveur

- Triggers



Standard et Langages

- Le standard
  - SQL3 / PSM
- Les éditeurs
  - Informix SPL
  - Oracle PL/SQL
  - …
- Remarque
  - Java est de plus en plus utilisé pour les « Java Stored Procedures »
    - JVM sur le serveur BD

Utilisation dans Oracle

- Procédures anonymes
  - Oracle SQL*PLUS, SQL*DBA
- Procédures Stockées
- Actions des Déclencheur Triggers
  - Oracle SQL*MENU, SQL*FORMS

Bloc (Procédure) anonyme

- envoyé par le client au serveur

Syntaxe

```
DECLARE        déclarations des variables locales      -- optionel
BEGIN          suite d'instructions PL/SQL
               ou de blocs anonymes
EXCEPTION      suite d'instructions PL/SQL             -- optionel
               ou de blocs anonymes
END;
.
Run;           -- provoque l'exécution de la procédure anonyme
```

Typage des Variables

- Type de Données SQL
  - CHAR, VARCHAR2, NUMBER, DATE, BOOLEAN, LONG, RAW, ROWID
- Constructeur de type complexe
  - RECORD, TABLE, TABLE of RECORDS
- ADT Abstract Data Type
  - type « objet »
- Désignation du type d'une colonne d'un table

<*nom de table*>.<*nom de colonne*>%TYPE

Instructions

- Affectation
- Instructions SQL
  - Requête

    SELECT, INSERT, UPDATE, DELETE
  - Transaction

    COMMIT, ROLLBACK, SAVEPOINT
  - Curseur

    DECLARE, OPEN, FETCH, CLOSE, WHERE CURRENT OF
- Structures de Contrôle

  imbrication illimitée
  - IF...THEN...ELSIF...END IF
  - boucles FOR, WHILE, LOOP et EXIT, GOTO
- Exceptions
  - Exceptions internes au SGBD ou définies par l 'utilisateur
  - Plusieurs niveaux d 'exception

Instructions

- Affectation

  <*variable*> := <*variable ou expression*>;

  SELECT <*variable*> INTO <*variable*> <*suite de la clause FROM-WHERE*> ;

- Test

  IF <*condition*> THEN <*instructions*> END IF;

  IF <*condition*> THEN <*instructions*> ELSE <*instructions*> END IF;

  IF <*condition*> THEN <*instructions*> ELSE <*instructions*>
  ELSIF <*condition*> THEN <*instructions*> END IF;

- Boucles

  LOOP <*instructions*> EXIT WHEN <*conditiondesortie*>;
  <*instructions*> END LOOP;

  WHILE <*condition*> LOOP <*instructions*> END LOOP;

  FOR <*variable*> IN <*valeur*> …<*valeur*> LOOP <*instructions*> END LOOP;

Affection à partir d'une requête

- 1 seule ligne retournée sinon erreur

SELECT INTO

```
DECLARE
    masseSalairialePlus10000 Employe.salary%TYPE;
BEGIN
   SELECT          SUM(salary)
   INTO            masseSalairialePlus10000
   FROM            Employe
   WHERE           salary > 10000;
END;
```

RETURNING

```
DECLARE nom Employe.name%TYPE, nouveausal Employe.salary%TYPE
BEGIN
        UPDATE Employe SET   salary = salary * 1.1 WHERE  numemp = 100
        RETURNING name, salary INTO nom, nouveausal;
END;
```

Exceptions (i)

```
declare
    s Employe.salary%TYPE;          --variable locale
begin
   begin
      select salary into s from Employe where numemp = num;
   exception
      when no_data_found
        begin
          s := -1; notifieErreur (num);     -- Appel d 'une autre procédure
        end;
   end;
exception
    when no_data_found
        begin
            notifieErreur2 (num);     -- Jamais atteint
        end;
        when others then null;
end;
```

Exceptions (ii)

```
declare
    e_Depassement       EXCEPTION;   -- déclaration d 'une exception utilisateur
    s                   Employe.salary%TYPE;
begin
   begin
      select sum(salary) into s from Employe;
      if s > 1000000 then
                  raise e_Depassement;
      end if;
   exception
      when e_Depassement
      begin
          RAISE_APPLICATION_ERROR(-20001, 'La masse salariale a explosé ! ');
      end;
      when no_data_found
      begin
           notifieErreur2 (num);     -- Jamais atteint
      end;
      when others then null;
end;
```

Curseurs

Déclaration

CURSOR <*nomcur*> IS <*requête SELECT*>;

Usage

- Ouverture

  OPEN <*nomcur*>;

- Parcours dans un boucle
  - test d'arrêt

  EXIT WHEN <*nomcur*>%NOTFOUND;

  WHILE <*nomcur*>%FOUND LOOP ...
  - récupération des valeurs dans des variables

  FETCH <*nomcur*> INTO <*liste de variables*> ;

- Fermeture

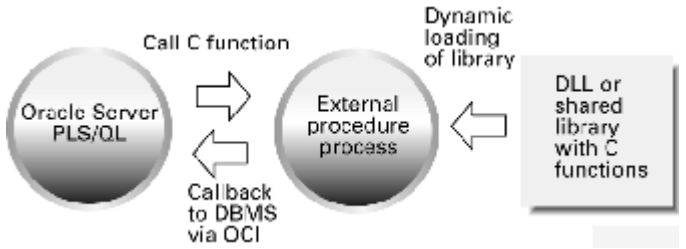
  CLOSE <*nomcur*>;

Exemple de Curseur

```
CREATE OR REPLACE PROCEDURE augmentationSalaire(
            seuil                IN Employe.salary%TYPE,
            augmentation         IN NUMBER(2)
) AS
   sal Employe.salary%TYPE;
   num Employe.numemp%TYPE;
   CURSOR c IS SELECT salary, numemp FROM Employe;
BEGIN
      OPEN c;
      FETCH c INTO sal, num;      -- attention à l'ordre : types compatibles
      WHILE c%FOUND LOOP
            IF sal IS NOT NULL AND sal < seuil THEN
               UDPATE Employe SET salary = salary*(augmentation + 100.0)/100
                  WHERE numemp = num;
            END IF;
            FETCH c INTO sal, num;
      END LOOP;
      CLOSE c;
END;
```

Procédures et Fonctions

- Plusieurs Types
  - *Bloc (procédure) anonyme*
    - *envoyé par le client au serveur*
  - Procédure stockée, Fonction stockée
    - stockées sur le serveur
  - Procédure membre, Fonction membre
    - méthodes membres des types objets
  - Procédure externe
    - écrit en LG3, utilise l 'API d 'Oracle
      OCI : *Oracle Call Interface*
    - peut être invoqué depuis PL/SQL



- Fonctionnalités générales
  - Récursion illimitée
  - Surcharge des paramètres
  - Passage des paramètres avec des modes IN, OUT, INOUT

Procédures Stockées

- Déclaration

```
CREATE OR REPLACE PROCEDURE <nomproc> ( <listarg> )        AS
                            déclarations des variables locales (optionel)
BEGIN                       suite d'instructions PL/SQL
EXCEPTIONS                  suite d'instructions PL/SQL
END;
.
Run;                        -- déclaration de la procédure stockée
```

- Liste des Arguments
  - *nomarg mode type*
  - *mode* : IN, OUT ou INOUT
  - *type* : de base (SQL92), NUMBER, ADT (SQL3)
    - désignation du type d'une colonne d'un table

      <*table*>.<*colonne*>%TYPE

Procédures Stockées

- Exemple

```
CREATE OR REPLACE PROCEDURE nouvelEmploye(
        n IN Employe.name%TYPE,  s IN Employe.salary%TYPE
) AS BEGIN
   INSERT INTO Employe VALUE ( 123, n, s, NULL);
END;
.
RUN;
```

- Invocation depuis un bloc anonyme

```
BEGIN
    nouvelEmploye('Martin', 10000);
    nouvelEmploye('Dupont', 20000);
END;
.
RUN;
```

Fonctions

```
create or replace function get_sal (num in Employe.numemp%TYPE)
                    return     Employe.salary%TYPE is
   s Employe.salary%TYPE;          --variable locale
begin
   begin
      select salary into s from Employe where numemp = num;
   exception
      when no_data_found
        begin
          s := -1; notifierErreur (num);     -- Appel d 'une autre procédure
        end;
      when others then null;
   end; -- bloc imbriqué
   return s;
end; -- function
```

Informix SPL

- SPL (Stored Procedure Language)

- Définition de Procédures et de Fonctions Stockées

```
CREATE PROCEDURE raise_prices( per_cent INT)
...
END PROCEDURE
DOCUMENT "USAGE: EXECUTE PROCEDURE raise_prices (xxx)",
                 "xxx = percentage from 1 - 100";

CREATE FUNCTION find_group( id INT )
RETURNING INT, REAL;
...
END FUNCTION;

DROP PROCEDURE raise_prices;
```

Informix SPL - Imbrication de bloc

```
CREATE PROCEDURE scope()
    DEFINE x,y,z INT;
    LET x = 5;
    LET y = 10;
    LET z = x + y; --z is 15
    BEGIN
      DEFINE x, q INT;
      DEFINE z CHAR(5);
      LET x = 100;
      LET q = x + y; -- q = 110
      LET z = 'silly'; -- z receives a character value
    END
    LET y = x; -- y is now 5
    LET x = z; -- z is now 15, not 'silly'
END PROCEDURE;
```

Informix SPL - Curseur

```
CREATE_PROCEDURE increase_by_pct( pct INTEGER )
DEFINE s INTEGER;
FOREACH sal_cursor FOR
SELECT salary INTO s FROM employee
WHERE salary > 35000
LET s = s + s * ( pct/100 );
UPDATE employee SET salary = s
WHERE CURRENT OF sal_cursor;
END FOREACH
END PROCEDURE;
```

Informix SPL - Exception

```
BEGIN
    ON EXCEPTION IN (1)
    END EXCEPTION WITH RESUME -- do nothing significant (cont)
    BEGIN
    FOR i IN (1 TO 1000)
      FOREACH select ..INTO aa FROM t
                IF aa < 0 THEN
                        RAISE EXCEPTION 1 ; -- emergency exit
                END IF
      END FOREACH
    END FOR
    RETURN 1;
END
--do something; -- emergency exit to this statement.
TRACE 'Negative value returned';
RETURN -10;
END
```

Java, Persistance et Bases de Données

- Rendre persistant des objets Java
  - 35% du travail du développeur passe dans le mapping Objet/JDBC
- Plusieurs solutions de stockage
  - Sérialisation + Fichier
    - ☹ ne permet pas le partage et la recherche, n 'est pas incrémental
  - JDBC
    - API bas-niveau
      - ☹ impedance mismatch
  - SQLJ
    - Embedded SQL in Java
      - ☹ impedance mismatch
  - JavaBlend
  - JDO

JavaBlend

- OML Java de l ’ODMG2.0 (www.odmg.org)
  - classes additionnelles
    - `PersistentRoot` et `OID`
    - `Dcollection`, ...
- Transparence au SQL
  - Surcouche à JDBC

Java Data Objects (JDO)

- Permet de rendre persistants des instances de n'importe quelles classes
  - Persistance transparente
    - Accès direct aux membres
      Pas des méthodes set/get (accesseur/mutateur)
    - Déréferenciation par .
  - Instances persistante / transiente
- Architecture de base
  - PersistenceManager
    - Gère les accès, la sauvegarde, les transactions et les recherches entre les applications et les Data Stores
  - Transaction
  - Query
    - Recherche sur critère.
  - classes PersistenceCapable
    - Interface que doit implémenter une classe dont des instances peuvent être persistentes

JDO - Query

- Exemple

```
class Employee {
   String name;
   Integer salary;
   Employee manager;
}
...
Collection extent = persistMngr.getExtent(Class.forName("Employee"), false);
Query q = persistMngr.newQuery (
            Class.forName ("Employee"), // class
            extent,                     // candidates
            "salary > 50000"            // filter
          );
Collection resultSet = q.execute();
```

Java coté SGBD

- Java Stored Procedure/Function (Oracle)
  - Procédures/Fonctions stockées écrites en Java (au lieu de PL/SQL) et utilisant JDBC ou SQLJ
- Java Triggers (Oracle)
  - Actions des triggers écrites en Java (au lieu de PL/SQL) et utilisant JDBC ou SQLJ

Bibliographie - Autre

PL/SQL

- Scott Urman , « Oracle8 PL/SQL Programming », ed Osborne-McGraw-Hill, Oracle Press Series, ISBN 0-07-882305-6.
- Steven Feuerstein, « Oracle PL/SQL Programming »,2nd Edition, ed O'Reilly. ISBN 1-56592-335-9.
- Steven Feuerstein, « Advanced Oracle PL/SQL Programming with Packages », ed O'Reilly, ISBN 1-56592-238-7.
- http://technet.oracle.com

ADO

- La référence du programmeur ADO 2.0 (mi juin 99), Ed Eyrolles

Procédures stockées en .NET

- Motivations
  - Procédures et fonctions stockées
  - Actions des Triggers
  - ADT et Index sur ADT
  - Fonctions complexes d'agrégat
- Langage CLR : C#, Managed C++, J#, VB, …
- Principes
  - Chargement des assemblies
  - Création des procédures

Chargement des assemblies

- CREATE ASSEMBLY <assembly id>
- FROM <path>
- WITH PERMISSION_SET=[SAFE|EXTERNAL_ACCESS|UNSAFE]

- CREATE ASSEMBLY Customer
- FROM 'C:\build\cust\customer.dll'

- 2 tables systèmes: sys.assemblies et sys.assembly_files

Création de procédures

```
CREATE PROCEDURE <procname>
AS EXTERNAL NAME <assembly is>::<type name>::<method name>
```

```
class Salary {
    public static int compute(int sal) { …}
}
```

```
CREATE PROCEDURE ComputeSalary
  @sal int
AS EXTERNAL NAME Customer::Salary::compute
```

```
DECLARE @s int
SET @s=10000
EXEC ComputeSalary 100000 @s OUTPUT
SELECT @s
```

Bibliographie

JDBC

- S. White, M. Fisher, R. Cattell, G. Hamilton, M. Hapner, "JDBC API Tutorial and Reference", Ed Addison-Wesley, ISBN 0-201-63459-7
  - très complet (le plus complet)
- George Reese, « Database Programming with JDBC and Java with Packages », 1st Edition June 1997, ed O'Reilly, ISBN. 1-56592-270-0 (ISBN 2-84177-042-7 en français)
  - un peu incomplet
- Robert Orfali, Dan Harkey, « Client/Server Programming with Java and Corba », 2ème édition, 1998, Ed Wiley, ISBN 0-471-24578-X. Chapitres 23 à 26.
  - mise en œuvre de JDBC en architecture multi-tiers
  - benchmark TPC/AB avec JDBC
- Tutorial JDBC de http://java.sun.com

Bibliographie

SQLJ

- Nirva Morisseau-Leroy, Martin Solomon, Gerald Momplaisir, Oracle9i SQLJ Programming, Oracle Press 2001, ISBN: 0072190930

Bibliographie

- Revues
  - DBMS Magazine www.dbmsmag.com
  - DB2 Magazine www.ibm.com
  - Oracle On-line Magazine www.oramag.com
    - y sont publiés des « tips » sur PL/SQL