Programmation SQL

Programmation SQL(1)

- Comment passer une commande dans la base telle que :

PROCEDURE COMM
Si l'article est disponible dans la quantité commandée
ET le livreur disponible à la date de livraison désirée
→ Insérer sa commande dans la base
Sinon abandonner la commande

- Comment trouver tous les chemins de Paris à Toulouse ?

Routes	Départ	Arrivée	typeRoute	distance
	Paris	Bordeaux	AU	...
	Paris	Clermont-F	AU	...
	Bordeaux	Toulouse	AU	...
	Clermont-F	Millau	NA	...
	Millau	Toulouse	NA	...

- Impossible en SQL pur (SQL n'est pas un langage complet)
  - Pas de structure de contrôle : itérations, tests …
  - Besoin d'un langage complet pour programmer des actions sur les BD

Programmation SQL (2)

- **SQL est un langage**
  - déclaratif
  - manipulant des ensembles de tuples
  - ayant ses propres types de données
- ***Impedance mismatch*** avec tout langage procédural (C, Java ...)
  - conversion de types (ex: varchar(n) SQL => quel type C ?)
  - traitement des valeurs nulles (True, False, Unknown)
  - parcours itératif des résultats de requêtes SQL

Programmation SQL (3)

- Embedded SQL
  - Programme écrit dans un langage classique (C, C++, Cobol, etc.)
    … avec des instructions SQL directement dans le programme
  - Précompilation : le source (ex: C+SQL) est transformé en code compilable
  - Suivi d'une compilation classique (ex: C pur)
  - Ex: Pro*C (Oracle), ECPG (Postgres), Pro*Cobol (Oracle), etc…
- API de communication
  - SQL-CLI (Call Level Interface) intégré dans SQL2
  - popularisé par les médiateurs ODBC, JDBC
- Procédures stockées
  - Extension de SQL à des éléments de programmation procédurale
  - Procédures stockées et exécutées par le serveur
  - Ex: PL/SQL (Oracle), PL/pgSQL (PostgreSQL), T-SQL (SQLServer, Sybase)
  - Standardisé dans SQL3 (SQL/PSM)
- *API Propriétaires fournies par un éditeur de SGBD*

I. Embedded SQL

Conversion de types : exemple

```
SQL             C/C++

SMALLINT        short
INTEGER         int
FLOAT           float
DOUBLE          double
DECIMAL(p,s)    double      // risque de troncatures
MONEY(p,s)      double      // et d'erreurs d'arrondi
CHAR(n)         char x[n+1] // caractère de terminaison
VARCHAR(n)      struct {short len;
                        char arr[n+1];
                       }
DATE            char[12]    // "dd-mmm-yyyy"
TIMESTAMP       char[28]    // "dd-mmm-yyyy hh:mm:ss »
...
```


EMBEDDED SQL: EXEMPLE

Toute instruction SQL dans le programme source est repérée par le mot-clé EXEC SQL pour pouvoir être précompilée

```
...
EXEC SQL CONNECT COOPERATIVE // non normalisé
main()
{
    EXEC SQL BEGIN DECLARE SECTION
    char sqlstate[6], cru[11], annee[5];
    short int numvin;
    EXEC SQL END DECLARE SECTION

    numvin = 10;
    strcpy(cru,"BEAUJOLAIS");
    strcpy(annee,"1996");

    EXEC SQL INSERT INTO VINS
        VALUES (:numvin, :cru, :annee);
    ...
}
```


EMBEDDED SQL: CURSEURS

Objectif: itérer sur le résultat d'une requête ensembliste

```
def_curseur ::=
        DECLARE <nom_curseur> [INSENSITIVE] [SCROLL]
        CURSOR FOR <requête_SQL>
        [FOR [READ ONLY | UPDATE ]]

OPEN <nom_curseur>
CLOSE <nom_curseur>

FETCH       [NEXT | PRIOR | FIRST | LAST | ABSOLUTE n |
            RELATIVE n ]
[FROM] <nom_curseur>
INTO <variable> *
```


EMBEDDED SQL: CURSEURS

```
EXEC SQL DECLARE C1 CURSOR FOR
    SELECT numvin, cru
    FROM VINS
    WHERE degre > 10;

main()
{
    EXEC SQL BEGIN DECLARE SECTION
    char sqlstate[6], cru[11];
    short int numvin;
    EXEC SQL END DECLARE SECTION

    EXEC SQL OPEN C1 ;

    while (notEndofResult(sqlstate) {

        EXEC SQL FETCH C1 INTO :numvin, :cru;
        ...
    }

    EXEC SQL CLOSE C1 ;
}
```

SQL DYNAMIQUE

Permet d'exécuter des requêtes paramétrées

```
main()
{
    EXEC SQL BEGIN DECLARE SECTION
    char sqlstate[6]];
    short int numvin, prix;
    char cmd_updateVins[256];
    EXEC SQL END DECLARE SECTION

    strcpy(cmd_updateVins , "update vins
                             set prix = ?
                             where numvin = ?");

    EXEC SQL PREPARE cmd FROM cmd_updateVins;

    while (...) {
        printf("numéro du vin à modifier :"); scanf("%d",&numvin);
        printf("\n nouveau prix :"); scanf("%d",&prix);

        EXEC SQL EXECUTE cmd USING :prix, :numvin;
        ...
    }
}
```

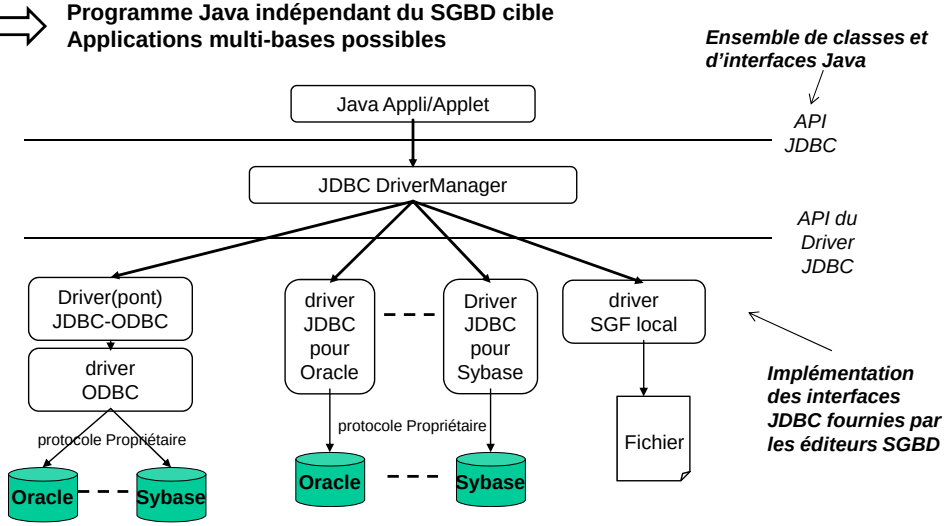

II. API de communication

- API indépendantes du SGBD (standardisées)
  - SQL-CLI (Call Level Interface)
    - Permet de se connecter et d'envoyer des requêtes SQL à tout serveur SQL, quel que soit son type, sa localisation, le mode de connexion ...
    - Intégrée dans le standard SQL2
  - ODBC (Open DataBase Connectivity)
    - Créée par Microsoft
    - Interface C de connection à un SGBD, administration via une interface Microsoft
    - Pour chaque SGBD, un pilote ODBC est necessaire
    - Basée sur le standard SQL-CLI
    - Disponible aussi sur plateformes autres que Windows : http://www.unixodbc.org/
  - JDBC (Java DataBase Connectivity)
    - Créée par SUN (racheté par Oracle)
    - Interface JAVA de connection à un SGBD
    - Pour chaque SGBD, un pilote JDBC est necessaire
    - Basée sur le standard SQL-CLI

Architecture logicielle de JDBC

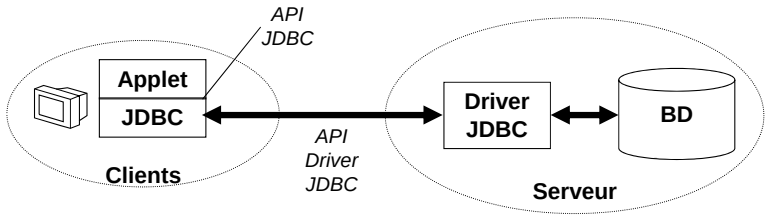
- Chaque BD utilise un pilote (driver) qui convertit les requêtes JDBC dans le langage natif du SGBD

⇨ **Programme Java indépendant du SGBD cible**
Applications multi-bases possibles

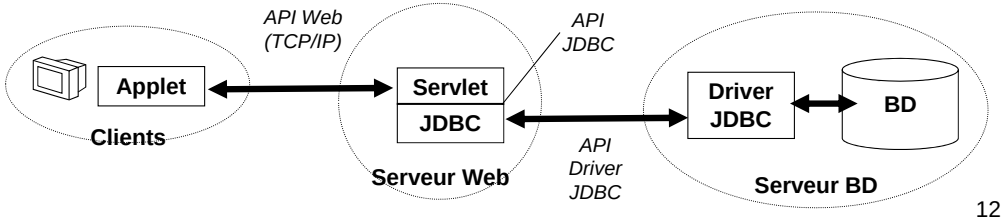


Exemples d'utilisations de JDBC

- Dans un client léger classique



- Dans une architecture J2E



L'API JDBC

- API fournie par le package java.sql
- Interfaces (principales) du package
  - Driver // gestion des pilotes
  - Connection // gestion des connexions
  - Statement // exécution de requêtes classiques
  - PreparedStatement // préparation de requêtes dynamiques
  - CallableStatement // appel de procédures stockées
  - ResultSet // manipulation du résultat
  - ResultSetMetaData // description du résultat
  - DatabaseMetaData // description de la base
- Les extensions dans javax.sql

Connexion au serveur

- Chargement du pilote
  - méthode *Class.forName*
  - Paramètre = chemin de la classe du driver (fournit dans la doc. du driver)
- Connexion au serveur
  - méthode getConnection de la classe DriverManager
  - Paramètre : URL de la source de données, nom et password de l'utilisateur
- Exemple

```
import java.sql.*;

public class ObjetUtilisantJDBC
{
     public static void main(String[] Args)
     {    // chargement du driver
          try {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); }
          catch (Exception E) {System.err.println(« Chemin du driver incorrect!");}

          // Connexion à la BD
          Connection con = DriverManager.getConnection(jdbc:odbc:Oracle, "scott", "tiger");
     }
}
```


Créer une instruction

- Une instruction est représentée par une instance de la classe
  - *Statement*, (instruction SQL statique)
  - *CallableStatement*, (appeler une procédure stockée)
  - ou *PreparedStatement* (instruction SQL précompilée et paramétrée)
- La création se fait à partir de l'objet *Connexion*
- Exemple

```
import java.sql.*;
public class ObjetUtilisantJDBC
{
     public static void main(String[] Args)
     {
          // ... Etapes précedentes...
          Statement req1 = con.createStatement();
          CallableStatement req2 = con.prepareCall(str);
          PreparedStatement req3 = con.prepareStatement(str);
     }
}
```


Exécuter une instruction simple/statique

- Méthodes (principales) pour exécuter une instruction
  - *executeQuery* pour les requêtes de type SELECT ...
    - Rend un ensemble de tuples lisibles via un curseur
  - *executeUpdate* pour les CREATE ou INSERT/DELETE/UDATE...
    - Rend le nombre de tuples impactés par la requête
- Paramètre = chaîne de caractères contenant la requête
- Exemple

```
import java.sql.*;
public class ObjetUtilisantJDBC
{
     public static void main(String[] Args)
     {
     int  nbTuples
          // ... Etapes précedentes...
          req1.executeUpdate("CREATE TABLE contact (nom VARCHAR(50), tel CHAR(10))");
          req1.executeUpdate("INSERT INTO contact VALUES ('dupond', '0102030405')");
          nbTuples := req1.executeUpdate("UPDATE contact " +
                                         "SET nom = 'dupont' " +
                                         "WHERE nom = 'dupond'");

     }
}
```

Itérer sur un résultat: curseur

- Curseur représenté par l'interface *ResultSet*
- Fonctionnement
  - Le curseur d'un objet *ResultSet* est positionné avant le premier tuple
  - La méthode *next* permet de faire avancer ce curseur sur le tuple suivant
    - retour = *false* ➔ il n'y a plus de tuples à traiter
  - Les valeurs des attributs sont obtenues grâce aux méthodes *getXXX( )*
    - Paramètre = numéro/nom de colonne dans le résultat
- Exemple

```
// ...
ResultSet rs = req1.executeQuery("SELECT * FROM contact");
while(rs.next()) {
     String nom = rs.getString("nom"); // ou String nom = rs.getString(1);
     String tel = rs.getString("tel"); // ou String nom = rs.getString(2);
     // ...
// fermer le curseur en fin de traitement pour libérer l'espace
rs.close();
}
```


Requête précompilée

- Une requête peut être précompilée en utilisant *PreparedStatement*
  - Dans un objectif de performance
  - Des variables ('?') peuvent être utilisées pour paramétrer la requête
  - Avant d'exécuter la requête, on fixe les paramètres (remplacement des ? par des valeurs) grâce aux méthodes setXXX
    - Premier argument = numéro d'ordre du '?' à remplacer
    - Deuxième argument = valeur de remplacement
- Exemple

```
// ...
// précompiler la requête
String req = "UPDATE repertoire SET tel = ? WHERE nom = ?";
PreparedStatement updateTel = con.prepareStatement(req);

// fixer les paramètres
updateTel.setString(1, "0908070605");
updateTel.setString(2, "durand");
// exécuter la requête
updateTel.executeUpdate();

// et recommencer …
updateTel.setString(1, "0900000000");
updateTel.setString(2, "dupond");
updateTel.executeUpdate();
// ...
```


Procédures stockées

- L'interface *CallableStatement* permet d'invoquer une procédure stockée
  - NB: la procédure est stockée au niveau du SGBD…
  - … la création se faisant avec la méthode *prepareCall* de *Connection*

```
// ...
CallableStatement cs = c.prepareCall("{call nom_procedure_stockee}");
// ...
```

- La syntaxe de l'appel de procédure (call ...) est traduit par le driver JDBC dans la syntaxe native du SGBD
- L'exécution de l'instruction se fait de façon classique en fonction de ce que retourne la procédure
- L'interface *CallableStatement* est sous-interface de *PreparedStatement* et accepte donc des paramètres

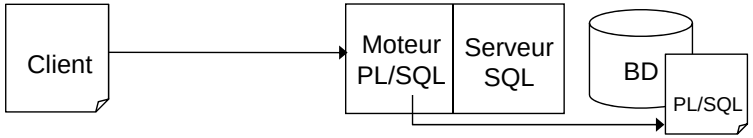
Accès aux méta-données

- Procurent des informations (méta-données) sur
  - le SGBD et le driver JDBC (*DatabaseMetaData*),
  - ou sur le résultat d'une requête (*ResultSetMetaData*)
- Exemples de méthodes de l'interface *DatabaseMetaData*
  - *getSchemas* pour la liste des schémas disponibles
  - *getTables* pour la liste des tables
- Exemples d'utilisation de *ResultSetMetaData*

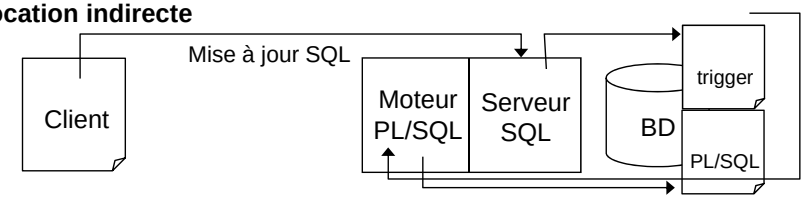
```
// ...
ResultSet rs = req1.executeQuery("SELECT * FROM contact");
ResultSetMetaData rsmd = rs.getMetaData();
int nbColonnes = rsmd.getColumnCount();
// ...
```

III. Procédures stockées (PL/SQL)

- **Procédure stockée**
  - considérée comme un objet (exécutable) de la BD
  - pouvant donc être partagée (si privilège EXECUTE)
  - invocable à distance
  - minimisant le trafic réseau si le traitement nécessite de multiples exécutions de requêtes ou de nombreux échanges de données entre un client et un serveur
- **Invocation directe**



- **Invocation indirecte**



Package PL/SQL

Package = regroupement de programmes dans un objet de la BD

```
CREATE OR REPLACE PACKAGE nom_Package
AS
    liste des signatures de
    procédures/fonctions PL/SQL
END nom_Package ;
```

Visible par l 'application

```
CREATE OR REPLACE PACKAGE BODY nom_Package
AS
    déclaration des variables globales
    déclaration du corps des
    procédures/fonctions PL/SQL
END nom_Package
```

Interne au package

Types et variables PL/SQL

- **Types de base SQL2**
  - Char, Varchar2, Number, Date, Boolean …
- **Types complexes**
  - User Defined Types (SQL3)
  - Record : TYPE maStructure IS RECORD (champ1 NUMBER, champ2 VARCHAR2) ;
  - Table : TYPE maListe IS TABLE OF NUMBER ;
- **Déclaration de variables**
  - maVar VARCHAR2 DEFAULT 'ROUGE'
  - maVar Personne.nom %TYPE => type de l'attribut concerné
  - maVar Personne %ROWTYPE
  - maVar MonCurseur %ROWTYPE

Type RECORD

Procédures et fonctions PL/SQL

```
PROCEDURE nom_Proc
(param1 IN CHAR,              // param entrée
 param2 OUT NUMBER,           // param sortie
 param3 INOUT VARCHAR2);      // param entrée/sortie

PROCEDURE nom_Proc
DECLARE
    déclaration des variables locales
BEGIN
    corps de la procédure
    EXCEPTION
    traitement des exceptions
END nom_Proc

FUNCTION nom_Fonct (param1 IN CHAR) RETURN NUMBER ;
```

Structures de contrôle PL/SQL

- Traitement conditionnel

```
IF condition THEN traitement1
[ELSIF condition THEN traitement2]
[ELSE traitement3]
END IF;
```

- Itérations

```
WHILE condition LOOP
traitement
END LOOP;

FOR i IN 1..n LOOP
traitement
END LOOP;

LOOP
traitement
EXIT WHEN condition END LOOP;
```


Curseurs PL/SQL : utilisation

- Itérer sur le résultat d'une requête SQL
  - Déclaration:
  - Manipulation:
    - Commandes : *OPEN*, *FETCH*, *CLOSE*
    - Utilisation des attributs de curseur : %NOTFOUND, %ISOPEN …

```
DECLARE
   CURSOR monCurseur IS requête_SQL;
```

```
DECLARE
   CURSOR dept_10 IS SELECT Nom, Salaire
        FROM employes WHERE NumDept = 10;
   tuple dept_10%ROWTYPE;
BEGIN
   OPEN dept_10;
   LOOP
        FETCH dept_10 INTO tuple;
        .........
   EXIT WHEN(dept_10%NOTFOUND) END LOOP;
   CLOSE dept_10;
END;
```


Curseurs PL/SQL : utilisation simplifiée

- Dans la pratique
  - Déclaration implicite du curseur
  - Déclaration implicite de l'enregistrement récepteur
  - Ouverture et fermeture du curseur implicites
  - Ordre de lecture pas à pas *fetch* implicite
  - Condition de sortie implicite

```
BEGIN
   FOR tuple IN
        (SELECT Nom, Salaire FROM employes WHERE NumDept = 10)
        LOOP
   .........
   END LOOP;
END;
```


Curseurs PL/SQL : exemple

- Augmente de 5% le salaire des employés du service compta…

```
DECLARE
BEGIN
   FOR Emp IN (SELECT nom, salaire FROM Employe WHERE service =
   'comptabilité') LOOP
        IF Emp.salaire IS NOT NULL AND Emp.Salaire < 30.000 THEN
           UPDATE Employe SET salaire = salaire*1,05 WHERE nom = Emp.nom;
        END IF;
   END LOOP;
END;
```

Exceptions prédéfinies

- Levées automatiquement par le moteur PL/SQL
  - CURSOR_ALREADY_OPEN, INVALID_CURSOR
  - NO_DATA_FOUND, LOGIN_DENIED, PROGRAM_ERROR
  - ZERO_DIVIDE, DUP_VAL_ON_INDEX…
- Traitées par la procédure PL/SQL section *begin end*

```
BEGIN
   …
   EXCEPTION
        WHEN CURSOR_ALREADY_OPEN THEN
           BEGIN Affiche_Err (-20000, 'problème…') ; END;
END;
```


Exceptions définies par l'utilisateur

Exceptions levées par l'instruction RAISE
Et traitées dans la sous-section EXCEPTION

```
DECLARE
Err_Temp EXCEPTION;
MaxTemp NUMBER;
BEGIN
    SELECT Max(temp) FROM Four INTO MaxTemp ;
    IF MaxTemp > 1000 THEN RAISE Err_Temp END IF;
    ...
    EXCEPTION
      WHEN Err_Temp
         BEGIN Affiche_Err (-200, 'le four va exploser') ; END;
END
```


Exemple complet

```
CREATE PACKAGE traitements_vendeurs IS
   FUNCTION chiffre_affaire (id_Vendeur IN NUMBER) RETURN NUMBER;
   PROCEDURE modif_com (id IN NUMBER, tx IN NUMBER);
END traitements_vendeurs;
```

```
CREATE PACKAGE BODY traitements_vendeurs IS
   FUNCTION chiffre_affaire (id_Vendeur IN NUMBER) RETURN NUMBER
        IS
         ca NUMBER;
        BEGIN
         SELECT SUM(montant) INTO ca FROM vendeurs WHERE id_vendeur = id;
         RETURN ca;
        END;
   PROCEDURE modif_com (id IN NUMBER, taux IN NUMBER)
        IS
        BEGIN
         UPDATE vendeur v
         SET v.com = v.com*(1+taux)
         WHERE v.id_vendeur = id;
        END;
END traitements_vendeurs;
```


Conclusion : programmation SGBD

- Les langages de type PL/SQL constituent le mode d'utilisation le plus courant des bases de données relationnelles
  - Utilisé pour programmer des procédures stockées
  - Utilisé pour programmer des triggers
  - Performant (réduit le transfert réseau)
  - Bien adapté aux clients légers (ex: smartphones…) car déporte les traitements sur le serveur
- Les programmes SGBD écrits dans des langages de programmation classiques sont utilisés
  - pour programmer des traitements sur un serveur d'application (J2E)
  - Pour programmer des applications de présentation
  - Peuvent être indépendantes du SGBD (JDBC/ODBC) ou dédiées