SGBD2 – DUT 2A

V- PROGRAMMATION CLIENT / SERVEUR AVEC LES SGBD

Nhan LE THANH

V- PROGRAMMATION CLIENT/SERVEUR

OBJECTIF

Mettre en place des applications externes de la base de données. Trois modes de programmations seront étudiés :

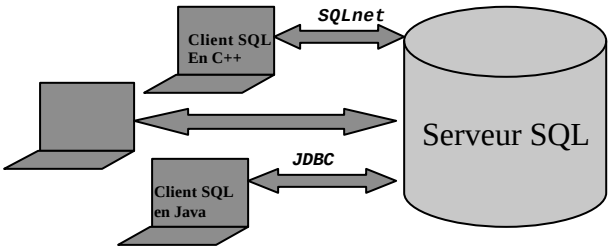
- Approche CL/SV avec pré-compilation : Pro-C

- Approche CL/SV avec un médiateur : JDBC

- Approche CL/SV via un serveur de métiers et HTTP : Servlet et JSP

Architectures Client/Serveurs

- **Architectures 2-tiers (2 étages)**



- **Serveur SGBD et/ou de services «** *métiers* **»**
  - **Données, programmes, démons, fonctions prédéfinies**
- **Client SQL en charge du dialogue et/ou de l'application**

Architectures Client/Serveurs

Architectures 2-tiers (2 étages)

- *Première approche* : Clients « musclés » et serveur SQL « léger »
  - **Client en charge de l'application** et **du dialogue**
  - **Serveur SQL limités aux données**
  - ***Avantages* :**
    - **BD relativement portable**
  - ***Inconvénients* :**
    - **Difficulté à déployer et à maintenir les clients**
    - **Peu de factorisation de programmation**

Architectures Client/Serveurs

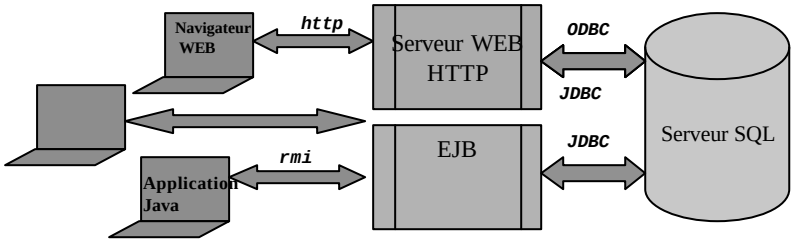
- **Architectures 2-tiers (2 étages)**

Deuxième approche **:** Clients « légers » et serveur SQL « lourd »

- Client limité aux dialogues et/ou aux interfaces graphiques
- Serveur SQL en charge des services métiers (procédures stockées, démons, …)
- ***Inconvénients*** **:**
  - **BD non portable (langage des serveurs non normalisés)**
- ***Avantages*** **:**
  - **Clients plus faciles à déployer et à maintenir**
  - **Factorisation de la programmation**
  - **Minimise les échanges réseau**

Architectures Client/Serveurs

- **Architectures n-tiers (n étages)**



- Serveur SGBD de données
- Serveur d'Applications centralisées (*services métiers*) – Client SQL
- Client léger / universel (navigateur web) en charge du dialogue et/ou des interfaces

Architectures Client/Serveurs

- **Architectures n-tiers (n étages - suite)**

- ***Avantages***
  - **Centralisation et factorisation de la couche «** *métier* **»**
  - **Client universel pour les serveurs de type HTTP**
  - **Déploiement facilité et/ou inutile**

- ***Inconvénients***
  - ***Gestion de plusieurs serveurs***
  - ***Coûts d'achat plus élevés***

- **Architectures Client/Serveurs**

Deux mode de programmation Client/Serveur

- **Une couche réseau propriétaire + API langage :**
  - **ODBC + pilote (*API C/C++ pour serveurs MS*)**
  - **SQLnet + OCI (*API C/C++ pour Oracle*)**

- **Une couche réseau propriétaire + extension langage**
  - **SQLJ pour Java**
  - **SQLC pour C**

V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

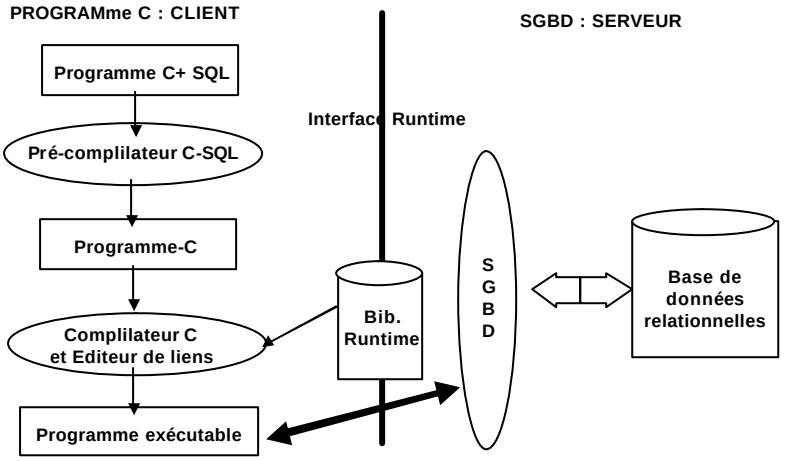
PRINCIPAUX POINTS ABORDÉS

* Utilisation des variables locales
* Connexion et communication
* Manipulation de données sans curseurs
* Manipulation de données avec curseurs
* Contrôles transactionnels
* Exécution dynamique
* Administration de base de données

V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

0- Architecture de l'approche



V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

1- Variables locales

Programme du langage hôte

- Une variable locale est une variable définie dans le programme du langage hôte (C) qui sera utilisée dans une ou plusieurs instructions SQL
-> Ces variables doivent être déclarées dans une "section de déclaration" du programme host avant d'être utilisées
-> Syntaxe

```
exec SQL begin declare section ;
   declaration_variables_local
exec SQL end declare session ;
```

- les variables définies peuvent être utilisées dans le programme source comme toutes les autres variables locales

- Exemple :

```
exec SQL begin declare section;
    interger age ; char nom[20];
exec SQL end declare session;
```

SGBD

- Une variable de Base de données (ou de objet de Base de données) est un nom symbolique désignant un objet de la base de donnée

- Exemple : noms d'attribut, noms de relation

- Dans une instruction SQL, pour distinguer la différence entre les variables locales et de base de données, toute référence à une variable locale doit être précédée par le symbole ":"

- Exemple :

```
exec SQL  SELECT E_NOM, E_AGE
          INTO :nom, :age
          FROM employe
          WHERE   E_DEPT = 10;
```


V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

1- Variables locales (2)

Utilisation des variables locales dans le programme C

->Définition des variables locales
Exemple

```
exec sql begin declare section;
        char      age;
        char      nom[20] ;
        unsigned  empno ;
        float     sal;
exec sql end declare section;
```

Les variables locales définies dans la "declare session" peuvent être utilisées dans le programme C comme toutes les autres variables locales C du programme

-> Utilisation d'un fichier externe des définitions
On peut intégrer un fichier externe contenant les déclarations des variables locales dans le programme par l'instruction SQL "include" :

- syntaxe : `exec sql include nom_fichier ;`

- exemple :

```
exec sql begin declare section;
        exec sql include 'mesvar.dec' ;
exec sql end declare section;
```

1- Variables locales (3)

Utilisation des variables locales dans les instructions C-SQL

-> Les variables locales peuvent substituer les objets de la base dans une instruction SQL. Il est nécessaire que elles soient de type compatible avec ce des objets substitués

-> Principales catégories d'utilisation

- dans une expression de données

```
(salaire = :sal) and (enom = :nom)
```

- dans une condition de recherche (WHERE)

```
... where (eno = :empnum) ;
```

- Dans la liste des objets des clauses "into", "select", ...

```
select    enom, salaire
          into      :nom, :salaire
```

- Dans les arguments des diverses clauses SQL

```
nombase = "personel" ;
exec sql connect :nombase ;
```


V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

1- Variables locales (4)

Utilisation des variables locales structurées

-> Les variables structurées peuvent être utilisées dans les clauses :
seclect, insert, fetch
pour simplifier le transfert de données entre la base de données et le programme

-> Les membres d'une variables structurées doivent avoir le type compatible avec ce de la colonne (au même rang) dans la table correspondante

-> Exemple

```
Exec sql begin declare session;
Struct emprec
{         unsigned  eno;
          char      name[20];
}
Exec sql begin declare session;
...
Exec sql  select    eno, enom
          into      :emprec
          where     eno = 23 ;
```

1- Variables locales (5)

Utilisation de variables locales comme variables d'indicateur de valeur NULLE

-> Les variables d'indicateur sont des variables locales utilisées pour indiquer l'existence des valeurs nulles dans le transfert de données entre la base et le programme :

- signaler s'il y a des valeurs nulles au résultat d'une requête de recherche de données

- signaler s'il y a des valeurs nulles aux données entrées pour une mise à jour,

- signaler si une chaîne de caractères au résultat d'une recherche déborde la taille d'une variable

-> Syntaxe :nom_var [indicator]:indicator_var

-> Exemple

```
exec sql   select enom, salaire
           into :nom:nom_null, sal:sal_null
           from employé
           where eno = 23;
if nom_null > -1 then printf("\n" nom) ;
```


1- Variables locales (6)

DECLARE SECTION DANS SQLPRO ORACLE

- Types :	CHAR	jusqu'à 255 caractères
	VARCHAR	type de caractères de longueur variable
	NUMBER(m,n)	réel et/ou entier
	INTEGER	entier
	DATE	plusieurs formats (DD-MM-YY)
	LONG	jusqu'à 64K
	RAW, LONG RAW	binaire
- Types particuliers	^int_pt INTERGER;	pointeur
	struct{...}	structure

- Variables systèmes LEVEL - UID - USER - SYSDATE - ROWID - ROWNUM

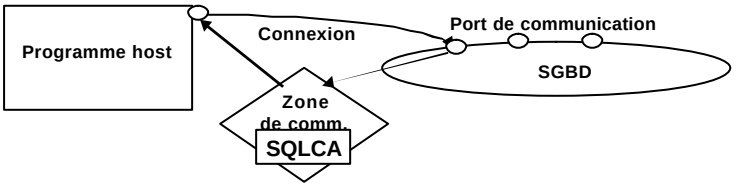
- Variables d'indicateur de valeur NULL :

INPUT: -1 valeur NULL en entrée
• 0 valeur non NULL en entrée

OUTPUT: -1 valeur NULL en sortie
0 valeur exacte
> 0 valeur exacte tronquée

Dans une instruction SQL, toute variable locale doit être préfixée par :

2- Connexion et Communication



-> La zone de communication (SQLCA) permet à SGBD de fournir au programme hôtedes informations sur l'état de l'exécution d'une instruction SQL : codes d'erreur, nombre de lignes au résultat, ...

->définir cette zonne de communication exec sql include sqlca ;

Les variables de la SQLCA (SGBD INGRES)

sqlcode	= 0	exécution réussite
	< 0	erreur d'exécution
	> 0	exécution réussite mais une condition d'exception est levée - la valeur 100 signale un résultat vide, - la valeur 700 signale qu'une message a été émis dans une procédure de la base

2- Connexion et Communication (2)

Les variables de la SQLCA (SGBD INGRES)(suite)

- sqlerrm chaine de longeur variable (70 max) contenant le message d'erreur correspondant

- sqlerrd tableau de 6 long entiers (4 octetsx6) : (errortext), sqlerrd(3), (sqlerrd(2) dans C) contient le nombre de lignes procédées par une des instructions :

"delete", "fetch", "insert", "update", "copy", "modify" ou "create as select"

- sqlwarn0 - sqlwarn78 octets qui désignent une anomalie d'exécution (valeur 'w')

0	s'il y a au moins une anomalie
1	s'il y a une troncature d'une chaîne
2	s'il y a une élimination d'une valeur nulle dans une agrégation
3	type ambigüe entre une variable locale et une colonne
4	si on prépare une "update" ou "delete" sans "where"
6	si la transaction termine anormalement
5 et 7	non utilisés

I- INTERFACE DE PROGRAMMATION PRO-C/C++

2.- Connexion et Communication (3)

Traitement des erreurs par les variables dans SQLCA

-> Le programme peut traiter les erreurs sans accéder directement dans sqlcode par l'instruction suivante

```
exec sql whenever condition  action ;
```

- condition peut être :
  - sqlwarning = vrai s'il y a une anomalie
  - sqlerror = vrai s'il y a une erreur d'exécution
  - not found = vrai s'il le résultat est vide
  - message = vrai s'il y a une diffusion de message dans une procédure

- action peut être :
  - continue : continuer le programme
  - stop : arrêter le programme
  - goto *label* : brancher à une autre partie du programme
  - call *procname* : appeler une procédure

-> Récupération des messages d'erreur : On peut récupérer les messages d'erreur par l'instruction

```
exec sql inquire_sql(chaine_var  = errortext) ;
```


I- INTERFACE DE PROGRAMMATION PRO-C/C++

2- Connexion et Communication (4)

Exemple de traitement des erreurs par les variables dans SQLCA

```
exec sql include sqlca;

exec sql begin declare section ;
        char nom[20] ;
        float       sal;
        unsigned    eno;
        char        message[80];
exec sql end declare section ;

...

exec sql   select enom, salaire  into :nom, :sal from employé where eno = 23 ;

exec sql whenever not found stop ;


if sql code < 0  then   exec sql inquire_sql (:message = errortext)
else                    printf("..." nom, sal) ;
```

I- INTERFACE DE PROGRAMMATION PRO-C/C++

2- Connexion et Communication (5)

Connexion à une base de données

-> La première instruction base de données à exécuter dans tout programme host doit être l'instruction "connect". Cette instruction établie la connexion du programme à une base de données parmi celles gérées par le SGBD

-> syntaxe **exec sql connect database_name | string_var ;**

-> la disconnexion du programme de la base de données se fait avec l'instruction"disconnect"

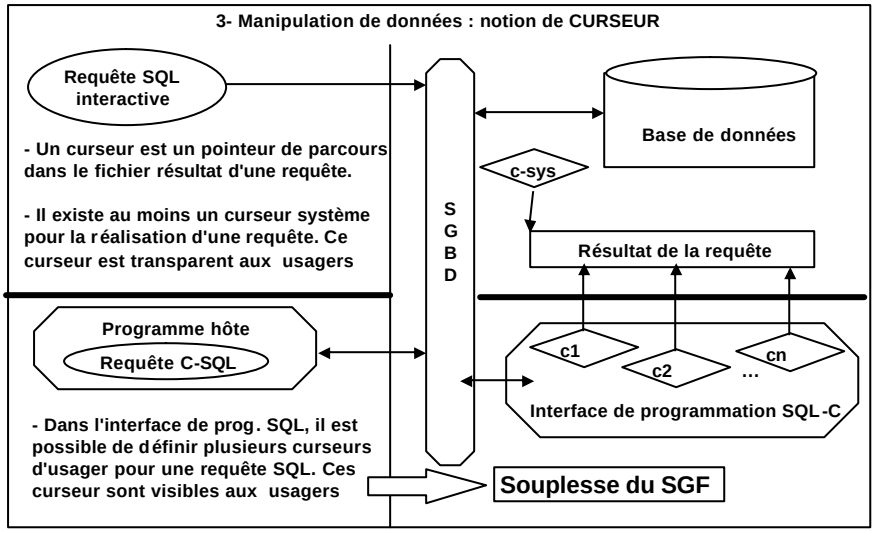
exec sql disconnect;

-> Quand l'instruction "stop" est exécutée par un "whenever", la "disconnect" est exécutée automatiquement avant l'arrêt du programme

-> Exemple **exec sql connect "BDpersonnelle" ;**

I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : notion de CURSEUR



3- Manipulation de données : sans CURSEURS visibles (2)

Recherche Singleton

- La recherche singleton rend au programme maximum une ligne de résultat

- S'il y a plusieurs lignes de résultat, seule la première ligne est prise en compte

- S'il n'y a aucune ligne de résultat, l'indicateur "not found" est levé à vrai

- Exemple

```
exec sql    select enom, salaire
            into :nom, :sal
            from employé
            where eno = 23 ;

if (errcode > 0) && (errcode != 100) then
            printf (... nom, sal)
else        prinf ('erreur') ;
```

Recherche bouclée

- La recherche bouclée permet de rechercher un résultat contenant plusieurs lignes et d'effectuer les instructions, qui suivent la requête, citées entre

"exec sql begin"
et
"exec sql end"

sur chaque ligne du résultat

- Durant l'exécution d'une recherche bouclée, aucune autre instruction SQL peut accéder à la base de donnée

- On peut sortir d'une recherche bouclée soit quand elle est complètement terminée soit par l'instruction 'endselect' : exec sql endselect.

- La sortie par "goto" est interdite

V- PROGRAMMATION CLIENT-SERVEUR

I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : sans CURSEURS visibles (3)

-> exemple de Recherche bouclée :

Imprimer un rapport contenant tous les employés de l'entreprise avec leur nom de leur département, regroupés par département et dans l'ordre de croissant des noms.

```
exec sql select eno, enom, age, salaire, dnom
            into :eno, :enom, :age, :sal, :dnom
            from employé e, departement d
            where e.dept = d.dno
            group by dnom
            order by enom ;

exec sql begin
   print_rapport (eno, enom, age, sal, dnom);
   if errcode < 1 then
            err = 1;
            exec sql endselect;
   end if;
exec sql end;
```

```
if err == 1 then
            printf ("erreur introduite après ligne",
            sqlca.sqlerrd(2))
else
            printf("exécution réussite");
            exec sql commit;
end if;
```

Attention:

- On ne peut pas accéder au contenu d'un tuple sauf par les valeurs récupérées dans les variables locales à l'aide de la clause INTO

- On ne peut pas effectuer plusieurs accès au même fichier résultat. A chaque fois, la requête doit être citée explicitement dans le programme

I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : avec CURSEURS visibles (4)

Limites de manipulation sans curseur	Avantages de manipulation avec curseur
- on ne peut accéder qu'à une séquence de données - le traitement sur cette séquence ne peut pas inclure des autres accès à la base de donnée - toute application ayant besoin plus de deux séquences d'accès en parallèle à la base de données, ne peut pas être réalisée avec cet outil	-- accéder ligne par ligne dans une table résultat - créer plusieurs séquences d'accès simultanément dans une ou plusieurs tables - échanger des données accédées avec le programme sans restrictions particulières

Instructions spéciales orientées curseur du SQL

declare *et* select	Définir un cursur et sélectionner l'ensemble de données associées
open	Commencer une séquence d'accès aux données
close	Fermer une séquence de parcours aux données
fetch	Prendre une nouvelle ligne sur cette séquence et la transférer dans les variables locales du programme
update	Modifier la ligne courante
delete	Supprimer la ligne courante

I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : avec CURSEURS visibles (5)
Déclaration des curseurs

-> But: Nommer un curseur et définir l'ensemble de données associées

Séquence recherche sans modification	Séquence recherche avec modification
Q1) Déclarer un curseur sur l'ensemble des employés (nom et salaire) du département "info" qui sont classés dans l'ordre ascendant des noms et descendant des salaires	Q2) Déclarer un curseur de mise à jour (salaire) sur l'ensemble des employés ayant un nom débutant par la lettre "A"
exec sql declare c1 cursor for select enom, salaire from employé where dept = select distint dno from département where dnom = 'info' order by 1 asc, 2 desc ;	critère = 'A%'; exec sql declare c2 cursor for select enom, salaire from employé where enom like :critère for update of salaire ;

3- Manipulation de données : avec CURSEURS visibles (6)
Déclaration des curseurs

-> Syntaxe

- séquence recherche sans modification

```
exec sql declare curseur_nom |string_var cursor
for          full_select ;
```

- séquence recherche pour les mises à jour

```
exec sql declare curseur_nom |string_var cursor
for          update_select
[for [deferred | direct] update of column {, column}];

Update_select

select result_expression {, result_expression }
from         table_name [corr_name ]
[where search_condition ]
```


3- Manipulation de données : avec CURSEURS visibles (7)
Déclaration des curseurs

-> Remarques

- La déclaration d'un curseur doit être faite avant son l'utilisation dans le programme

- Le nom du curseur peut être cité ou dans une variable locale de type chaine de caractères. Dans ce cas toute référence à ce curseur doit passer par cette variabe mais non par son nom

- Plusieurs curseurs peut être définis sur la même table mais un curseur ne peut impliquer dans qu'une séquence

- La séquence de données sera effectivement évaluée au moment de l'ouverture du curseur par "open"

- L'option "for update" doit être précisé s'il y a des mises à jour de données. Elle sera absente si c'est une suppression

-La clause "select" dans la définition d'un "update curseur" doit porter sur une seul table et n'inclue pas une des options suivantes :

- distinct - group by, having - order by - union

I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : avec CURSEURS visibles (8)
Ouverture, fermeture et parcours dans une séquence

Description

- **Open :** Initialiser une séquence d'accès par un curseur défini préalablement. Evaluer la requête "select" dans la définition du cursur correspondant

- **Close :** Terminer une séquence d'accès par un curseur déjà ouvert par un Open. Supprimer les structures temporaires créées par Open

- **Fetch :** déplacer le curseur sur la nouvelle ligne après la ligne courante dans la séquence du curseur ouvert citée. Transférer les données correspondantes vers les variables locales citées dans la clause INTO

Syntaxe

```
exec sql open curseur_nom [for readonly] ;

exec sql close curseur_nom ;

exec sql fetch curseur_nom into variable {, variable} ;
```


I- INTERFACE DE PROGRAMMATION PRO-C/C++

3- Manipulation de données : avec CURSEURS visibles (9)
Ouverture, fermeture et parcours dans une séquence

->Remarques

- Les instructions transactionnelles "commit" et "rollback" prennent fin de la transaction en cours et fermement tous les curseurs ouverts dans cette transaction

- L'instruction "fetch" opère sur une seul ligne à la fois. Elle peut donc être utilisée dans une boucle du langage hôte

- Quand "fetch" applique sur la dernière ligne de la séquence, le curseur maintient sa dernière position et la variable "sqlcode" de la SQLCA prend la valeur 100. Aucun changement sera produit par fetch sur les variables locales citées dans la clause "into"

- On peut utiliser l'instruction "whenever" pour contrôler la fin d'une boucle de parcours avec "fetch" en testant l'indicateur "not found" qui sera signalé quand "sqlcode" a la valeur 100. Dans ce cas, l'instruction "goto" est souvent utilisée pour mettre en ouvre une rupture de séquence

3- Manipulation de données : avec CURSEURS visibles (10)
Ouverture, fermeture et parcours dans une séquence

-> Exemple

Q) lister le nom et le salaire de tous les employés du département n° 5 dans l'ordre croissant des noms et décroissant des salaires

```
exec sql include sqlca ;
exec sql begin declare section ;
        char    nom[20] ;
        unsigned  age ;
exec sql end declare section ;
...
exec sql declare c1 cursor for
        select enom, salaire
        from employé
        where dept = 5
        order by 1 asc, 2 desc ;
```

```
exec sql whenever not found goto close_cursor;

exec sql open c1;
repeat
      exec sql fetch c1 into :nom, :age ;
      printf(... nom, age) ;
end repeat ;

close_cursor :
exec sql whenever not_found continue ;
exec sql close c1;
```


3- Manipulation de données : avec CURSEURS visibles (11)
Utilisation de curseur dans la mise à jour

-> Syntaxe

```
exec sql update tabname
        set column = expression, {column = expression}
        where curent of cursor_name ;

exec sql delete from tabname
        where curent of cursor_name ;
```

-> Description

- Les curseurs impliquées dans une mise à jour ou une suppression doivent préalablement être déclarés avec la syntaxe de "update_cursors"

- Pour les mises à jour, si l'option "direct" est mentionnée, toutes les mises à jours effectuées dans une séquence peuvent être visibles dans les autres séquences sur cette table sans attendre la clôture de la séquence

3- Manipulation de données : avec CURSEURS visibles (12)
Utilisation de curseur dans la mise à jour

-> Exemple
Q) Augmenter 10% de salaire de tous les employés ayant 1 salaire inférieur à 60 000 et supprimer tous les employés ayant un salaire supérieur à 300 000

```
...
exec sql declare c1 cursor for
            select enom, salaire
            from employé
            for update of salaire ;

exec sql open c1;

exec sql whenever not found
        goto close_cursor;
```

```
repeat
            exec sql fetch c1 into :nom, :salaire ;
            if salaire < 60000 then
                        exec sql update employé
                                    set salaire = salaire * 1.1
                                    where curent of c1
            else
                    if salaire >< 300 000 then
                        exec sql delete from employé
                        where current of c1 ;
                    end_if;
            end_if;
end_repeat ;
close_cursor :
exec sql whenever not_found continue ;
exec sql close c1;
```


4- Contrôle d'exécution transactionnelle

1- Index

```
exec sql create [unique] index index_name
            on table_name (column {, column })
            [with with_option_list ]

exec sql drop index  index_name {, index_name } ;
```

2- Privilège

```
exec sql grant all [privileges] on table_name  {,table_name }
to public | user_name{,user_name};

exec sql grant priv{, priv } on table_name  {,table_name }
 to public | user_name{,user_name};

priv :          delete, execute, insert, select, update (column {,column })
```

4- Contrôle d'exécution transactionnelle (2)

3- Contrôle d'exécution transactionnel

-> Dans un programme, la nouvelle transaction commence quand une première instruction parmi : seclect, insert, update, delete, create est exécutée après une des instructions : connect ou commit ou rollback

-> la fin d'une transaction, avec la validité de la transaction en cours, est marquée par un des trois cas suivants :

- **commit est exécutée**
- **disconnect est exécutée**
- **fin de programme est exécuté**

-> la fin d'une transaction, avec l'invalidité de la transaction en cours, est marquée par un des quatre cas suivants :

- **rollback est exécutée**
- **stop est exécutée**
- **sortie anormale du programme (system failure)**
- **abortement forcé du transaction (deadlock, ...)**

->On peut utiliser les instructions "commit", "rollback" et des indicateurs d' interblocage (sqlcode = deadlock) ou de suicide (sqlcode = forceabort)

4- Contrôle d'exécution transactionnelle (3)

3- Contrôle d'exécution transactionnel

-> Exemple 1 : sans curseur

```
exec sql whenever not found continue ;

exec sql whenever sqlwarning continue ;

exec sql whenever sqlerror goto err ;

exec sql commit ;

start :
exec sql insert into ... ;
exec sql update ... ;
exec sql select ... ;
exec sql commit ;
goto end;
```

```
err:
exec sql whenever sqlerror call sqlprint;

if (sqlca.sqlcode = deadlock) or
   (sqlca.sqlcode = forceabort) then goto start ;
else
   if (sqlcasqlcode < 0) then
        exec sql inquire_sql (:err_msg = errortext);
        exec sql rollback ; print 'error', err_msg;
    endif;
endif;
end:
...
```

I- INTERFACE DE PROGRAMMATION PRO-C/C++

4- Contrôle d'exécution transactionnelle (4)

3- Contrôle d'exécution transactionnel (continue)
-> exemple 2 : avec curseur

```
exec sql whenever not found continue ;
exec sql whenever sqlwarning continue ;
exec sql whenever sqlerror goto err ;
exec sql declare c1 cursor for ... ;
exec sql commit ;
start :
exec sql open c1 ;
while more rows loop
     exec sql fetch c1 into ...;
     if (sqlcq.sqlcode = zero_rows) then
              exec sql close c1;
              exec sql commit;
              goto end;
      endif;
              exec sql insert into ... ;
              exec sql update ... ;
              exec sql select ... ;
end loop;
```

```
err:
exec sql whenever sqlerror call sqlprint;
if (sqlca.sqlcode = deadlock) or
   (sqlca.sqlcode = forceabort) then goto start ;
else if (sqlcasqlcode < 0) then
     exec sql inquire_sql (:err_msg = errortext);
    exec sql rollback ; print 'error', err_msg;
endif;
end:
```


I- INTERFACE DE PROGRAMMATION PRO-C/C++

5- Exemple C-SQLPRO-C ORACLE

Création d'une table nommée EMP avec les attributs ENO, ENOM, EJOB, EDENT, ESAL, DEPNO, ECHEF

```
#include<stdio.h>

exec SQL BEGIN DECLARE SECTION;
            VARCHAR uid[20];
            VARCHAR pwd[20];
exec SQL BEGIN DECLARE SECTION;
exec SQL INCLUDE SQLCA;

main()
{
 /* entrer à l'ORACLE */

strcpy(uid.arr, "SCOTT");
uid.len=strlen(uid.arr);
strcpy(pwd.arr, "TIGER");
pwd.len=strlen (pwd.arr);

exec SQL CONNECT :UID IDENTIFIED BY :pwd;
```

```
/* Création de la table EMP */

exec SQL CREATE TABLE emp
            emo       number,
            enom      char(15),
            ejob      char(10),
            edent     date,
            esal      number,
            deptno    number,
            echef     number);

printf("table EMP est créée \n");

/* commit et déconnexion de l'ORACLE */

exec SQL COMMIT WORK RELEASE;
exit(0);
}
```

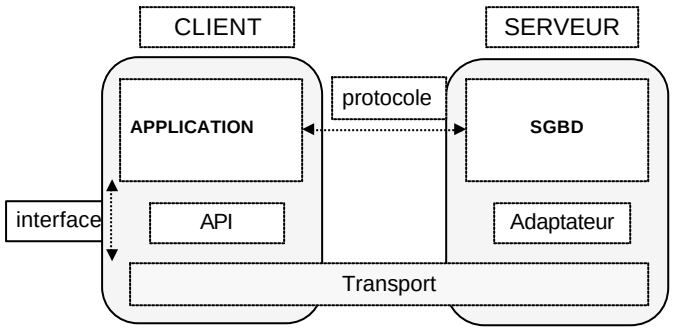
Le middleware

•Le *middleware* est le logiciel du milieu qui assure les dialogues entre clients et serveurs souvent hétérogènes.

•Ensemble des services logiciels construits au-dessus d'un protocole de transport afin de permettre l'échange de requêtes et des réponses associées entre client et serveur de manière transparente

• Un *système ouvert* est un système dont les interfaces obéissent à des standards internationaux établis au sein de structures accessibles à tous. De nombreux groupes proposent des standards, dont l'ISO, l'ANSI, le CCITT, l'IEEE.

- **API (*Application Programming Interface*)**
  - **Bibliothèque de fonctions permettant de développer des applications client serveur**
  - **Les programmes clients envoyent leurs requêtes au serveur par des appels à des fonctions contenues dans l'API**



0. Introduction

- **API propriétaire du SGBD**
  - fourni par l'éditeur du SGBD
  - permet uniquement l'accès à la base pour laquelle elle a été développée
  - Exemples
    - » OCI d'Oracle
    - » DB-Lib de Sybase
    - » SQL/Services de RDB
- **API indépendante du SGBD**
  - fourni par un constructeur indépendant du SGBD permet l'accès à des SGBD différents
  - Exemples
    - » ODBC de Microsoft
    - » IDAPI de Borland, Novell et IBM

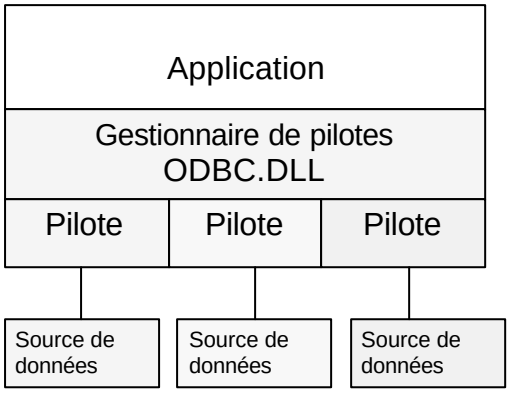
0. Introduction

- **Les principaux serveurs SQL**
  - ORACLE - DB2 - INFORMIX - SQL SERVER - SYBASE
  - INGRES, POSTGRES
- **Norme client SQL SGBD (*Call Level Interface*)**
  - interface applicative SQL
  - interface unique permettant l'accès à des SGBDR différents
  - travaux du SAG (SQL Access Group)
  - standard X/Open
- **RDA (*Remote Data Access*)**
  - protocole d'application construit au-dessus des couches présentation et session de l'architecture OSI de l'ISO
  - les messages permettent le transport des requêtes générées par l'interface CLI et les réponses associées
  - standard ISO

1. Généralités

- Open Data Base Connectivity
- Implémentation du standard CLI
- Accès normalisé à des SGBD relationnels différents (Oracle, DB2 ...)
- Accès même à des pseudo-SGBD, ou des tableurs, ou encore des gestionnaires de fichiers
- Interopérabilité avec des sources de données hétérogènes
- Avec ODBC, il est possible de développer une application sans se soucier de la source de données qui sera utilisée en exploitation
- API C (SDK ODBC) et classes C++ (MFC)

2. Composants ODBC



Application

- Connexion à un SGBD
- Envoi de requêtes SQL
- Récupération des résultats
- Gestion des erreurs
- Gestion des transactions
- Déconnexion

3. Gestionnaire de pilotes

- Charge dynamiquement les pilotes correspondant aux sources de données auxquelles l'application souhaite se connecter
- Consulte le fichier ODBC.INI / administrateur OGBC pour retrouver le pilote
- Transmet les requêtes au pilote
- Transmet les résultats à l'application
- Pour accéder à un nouveau SGBD, il suffit d'installer un pilote spécifique à ce SGBD (aucun changement dans l'application)
- Une application peut établir plusieurs connexions à différentes sources de données

• Fichier ODBC.INI

- Définit des sources de données
- Exemple
  - » [ODBC Data Sources]
  - » iut1=Oracle73 Ver 2.5 (32 bit)
  - » ...
  - » [iut1]
  - » Driver32=C:\ORANT\ODBC250\sqo32_73.dll
- La section [ODBC Data Sources] donne le nom de chaque source disponible et le pilote associé
- A chaque source correspond une section particulière donnant des informations supplémentaires : le nom du serveur, le protocole utilisé pour les communications ...

- **Administrateur ODBC**
  - ajoute les sources de données dans le fichier ODBC.INI en utilisant l'utilitaire *ODBC Administrator*
  - installe les pilotes ODBC
  - établit les connexions avec des BD physiques
- **Pilote : Deux types de pilotes**
  - Pilotes traitants (single-tier)
    - » traitent les requêtes SQL
    - » destinés à des BD non-SQL
    - » analyse, traduit les instructions SQL en opérations élémentaires de fichier et les transmet à la source de données
  - Pilotes transparents (multiple-tier)
    - » transmettent les requêtes SQL à un serveur qui les traitent
- **Source de données**
  - Données auxquelles un utilisateur souhaite accéder
  - Identifiée par une entrée dans le fichier ODBC.INI
  - Chaque entrée de nom de source dans ODBC.INI spécifie des informations de connexion

4. Niveaux de conformité

- En principe, une application ODBC devrait pouvoir interopérer avec n'importe quelle source de données.
- Mais en pratique, les pilotes et les sources de données associées n'offrent pas tous les mêmes possibilités de fonctionalités de l'API et de requêtes SQL
- Niveaux de conformité API
  - » Définit différents niveaux de fonctions de l'API
  - » Un pilote particulier précise son niveau de conformité API
- Niveaux de conformité SQL
  - » Définit différents niveaux de grammaire SQL

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

1. ODBC

- **Niveaux de conformité API**
  - Le niveau noyau (Core API)
  - Correspond au standard CLI de l'X/Open
  - Allocation et libération de descripteurs d'environnement, de connexion et d'instruction
  - Fonction de connexion
  - Préparation et exécution d'instruction SQL
  - Exécution directe d'instructions SQL
  - Liaison pour des paramètres SQL et des colonnes de résultats
  - Validation ou annulation de transactions
  - Récupération d'informations sur des erreurs
  - Le niveau 1 (Level 1 API)
    - » Noyau +
    - » Fonctions permettant d'obtenir des informations issues du catalogue d'une base, ainsi que des informations sur un pilote ou une source de données
  - **Le niveau 2 (Level 2 API)**
    - » Niveau 1 +
    - » Fonctions de gestion des curseurs

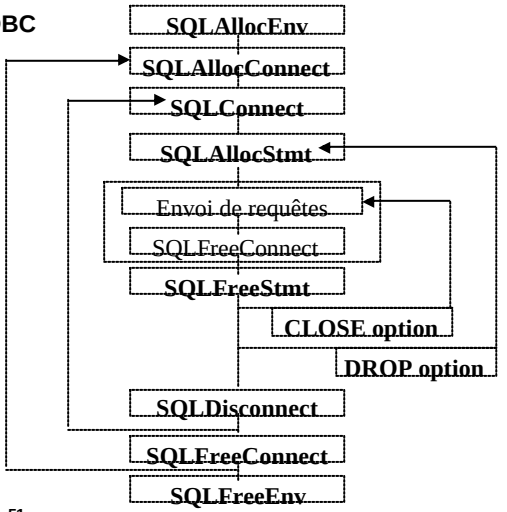
II- APPROCHE DE MEDIATEUR : ODBC et JDBC

1. ODBC

- **Niveaux de conformité SQL**
  - Grammaire SQL minimale
    - » LDD : CREATE et DROP TABLE
    - » LMD : SELECT, INSERT, UPDATE, DELETE
    - » Expressions simples dans les critères
  - Grammaire SQL noyau
    - » SQL min +
    - » LDD : ALTER TABLE, CREATE INDEX, CREATE VIEW, GRANT, REVOKE
    - » LMD : SELECT complet

5. Structure d'un programme ODBC

- **SQLAllocEnv**
  - » définit un descripteur d'environnement pour l'application
  - » ce descripteur est l'adresse d'une zone mémoire où seront placées des informations globales pour l'application, par exemple, le descripteur de la connexion courante
- **SQLAllocConnect**
  - » définit un descripteur de connexion
  - » ce descripteur est l'adresse d'une zone mémoire où seront placées des informations concernant une connexion
  - » un descripteur de connexion est toujours associé à un descripteur



- **SQLConnect**
  - » charge un pilote et établit une connexion entre l'application et une source de données
- **SQLDisconnect**
  - » termine une connexion entre l'application et une source de données
- **SQLFreeConnect**
  - » libère un descripteur de connexion
- **SQLFreeEnv**
  - » libère un descripteur d'environnement

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

1. ODBC

- **Exemple de programmation**

```
#include<windows.h>
#include<sql.h>
#include<sqlext.h>
HENV henv; // descripteur d'environnement
HDBC hdbc; // descripteur de connexion
// Allouer un descripteur d'environnement
SQLAllocEnv(&henv);
// Allouer un descripteur de connexion
SQLAllocConnect(henv, &hdbc);
// Etablir la connexion
SQLConnect( hdbc, "oracle", SQL_NTS,
            "scott", SQL_NTS,
            "tiger", SQL_NTS );
/* TRAITER LES REQUETES SQL     */
// Terminer la connexion
SQLDisconnect(hdbc);
// Liberer  les descripteurs
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```


II- APPROCHE DE MEDIATEUR : ODBC et JDBC

1. ODBC

6. Transactions dans ODBC

- Deux modes de validation des instructions SQL :
  - » Mode AUTO_COMMIT
    - chaque instruction SQL est automatiquement validée aprés son exécution
    - pas de notion de transaction dans ce mode
    - option par défaut
    - les pilotes qui ne supportent pas la notion de transaction sont toujours en mode AUTO_COMMIT
  - » Mode transactionnel
    - le programmeur gère explicitement la fin (validation ou annulation) des transactions);
- SQLConnectOptions
  - » permet de spécifier différentes options de connexion, en particulier le mode de validation
  - » il faut utiliser SQLConnectOptions avant d'établir la connexion
- SQLTransact
  - » termine une transaction
  - » soit en la validant \verb+SQL_COMMIT+
  - » soit en l'annulant \verb+SQL_ROLLBACK+

- **Exemple de programmation**

```
SQLAllocEnv(&henv);
SQLAllocConnect(henv, &hdbc);
SQLSetConnectOption(hdbc, SQL_AUTOCOMMIT,
    SQL_AUTOCOMMIT_OFF);
SQLConnect( hdbc, "oracle", SQL_NTS, "scott", SQL_NTS, "tiger",
    SQL_NTS  );
/* mise a jour no 1 */
/* mise a jour no 2 */
/* mise a jour no 3 */
SQLTransact(henv, hdbc, SQL_COMMIT);
SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

- **L'appel à SQLTransact(henv, hdbc, SQL_COMMIT) permet de valider en bloc les 3 mises à jour effectuées dans le contexte de la connexion hdbc.**

7. Envoi de requêtes SQL

- Deux manières pour soumettre une requête SQL:
- Envoi pour exécution directe
  - » ce cas concerne les instructions qui ne seront exécutées qu'une seule fois
  - » l'instruction est préparée et exécutée en une seule étape au moyen d'un appel à SQLExecDirect
- Envoi pour préparation puis demandes d'exécution
  - » ce cas concerne les instructions qui seront exécutées plusieurs fois
  - » l'instruction est préparée une seule fois en faisant appel à SQLPrepare
  - » l'instruction est ensuite exécutée au moyen de SQLExecute

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

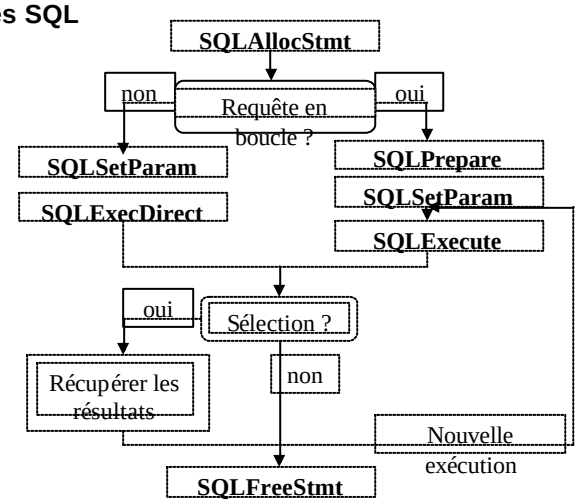
1. ODBC

- **Gestion de requêtes SQL**

SQLPrepare
- prépare une instruction SQL

SQLExecute
- exécute une instruction SQL préparée en utilisant les valeurs courantes des éventuels paramètres
- on utilise cette fonction lorsqu'on doit exécuter plusieurs fois la même instruction dans l'application, dans ce cas, l'instruction n'est préparée qu'une seule fois

SQLSetParam
- permet d'associer à un paramètre d'une instruction SQL, une variable contenant la valeur du paramètre
- l'utilisation de cette fonction est déconseillé depuis ODBC v2.0 où elle a été remplacée par SQLBindParameter mais qui est une fonction du niveau 1



II- APPROCHE DE MEDIATEUR : ODBC et JDBC

1. ODBC

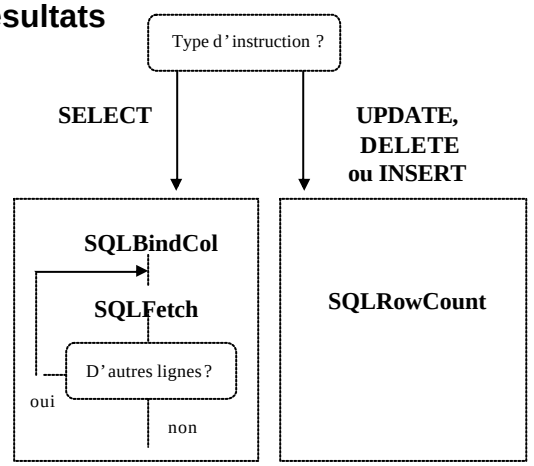
- **Terminer le traitement d'une instruction SQL**
  - La fonction SQLFreeStmt permet de libérer les ressources associées à un descripteur d'instruction
  - Elle possède quatre options:
- **SQL_CLOSE**
  - Ferme le curseur éventuellement
  - Le descripteur d'instruction peut être utilisé à nouveau
- **SQL_DROP**
  - Ferme le curseur éventuellement
  - Libère toutes les ressources associées au descripteur d'instruction
- **SQL_UNBIND**
  - Libère tous les buffers liés par SQLBindCol
- **SQL_RESET_PARAMS**
  - Libère tous les buffers requis par SQLBindParameter

- **Exemple de rogrammation**

```
rc = SQLAllocEnv(&henv);
rc = SQLAllocConnect(henv, &hdbc);
rc = SQLConnect(hdbc, "oracle",
     SQL_NTS,
     "scott", SQL_NTS, "tiger",
     SQL_NTS);
rc = SQLAllocStmt(hdbc, &hstmt);
rc = SQLExecDirect(hstmt,
     "select * from employe ",
     SQL_NTS);
/* RECUPERATION DES RESULTATS
   */
SQLFreeStmt(hstmt, SQL_UNBIND);
SQLFreeStmt(hstmt, SQL_DROP);
SQLFreeStmt(hstmt, SQL_CLOSE);
SQLDisconnect(hdbc);
SQLFreeConnect(hdbc);
SQLFreeEnv(henv);
```

8. Récupération des résultats

- **Liaison d'une colonne à une zone mémoire (BINDING)**
  - L'association d'une zone mémoire à une colonne de l'ensemble résultat se fait en utilisant la fonction SQLBindCol
  - Paramètres de SQLBindCol
    - » un descripteur d'instruction
    - » le numéro de la colonne résultat
    - » le type C de la colonne
    - » l'adresse de la variable qui recevra les valeurs de cette colonne
    - » le nombre d'octets maximum de la zone mémoire
    - » le nombre d'octets écrits dans la zone mémoire
  - Récupérer les lignes (FETCH)
    - » Les lignes de l'ensemble résultat sont récupérées en utilisant la fonction SQLFetch



- Exemple de programmation

```
rc = SQLExecDirect(hstmt,
   "select no, nom from employe",
    SQL_NTS);
rc = SQLBindCol(hstmt, 1,
       SQL_C_FLOAT, &no, 0, &cbno);
rc = SQLBindCol(hstmt, 2,
       SQL_C_CHAR, &nom, 20+1 ,
    &cbnom);
while (1) {
  rc = SQLFetch(hstmt);
  if (rc == SQL_NO_DATA_FOUND) break;
  if (rc != SQL_SUCCES) {
    printf("\n**Erreur fatale...\n");
    break;
  }
  printf("%f %20s", no, nom);
}
SQLFreeStmt(hstmt, SQL_UNBIND);
SQLFreeStmt(hstmt, SQL_CLOSE);
```

9. Détection des erreurs

- **Codes de retour des fonctions**
  - SQL_SUCCESS
  - SQL_SUCCESS_WITH_INFO
  - SQL_NO_DATA_FOUND aucune ligne retrournée avec FETCH
  - SQL_ERROR
  - SQL_INVALID_HANDLE
  - SQL_STILL_EXECUTING
  - SQL_NEED_DATA
- **Récupérer les messages d'erreurs**
  - La fonction SQLError permet d'obtenir des informations supplémentaires, lorsqu'une fonction ODBC retourne le code SQL_ERROR ou SQL_SUCCESS_WITH_INFO

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

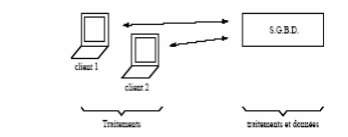
2. JDBC (Java Data Base Connectivity)

- JDBC est base sur
  - ANSI SQL-2
  - ODBC (Microsoft)
  - API Propriétaires
  - SQLX/OPEN CLI (Call Level Interface).
- Objectfs :
  - Simple,
  - Complet (en cours...),
  - Portable,
  - Modules réutilisables et/ou génériques,
  - Integration aux ateliers de développement

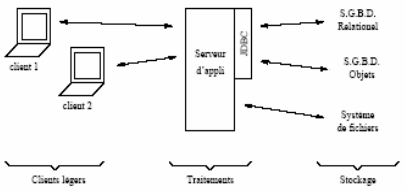
II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : architecture de mise en oeuvre

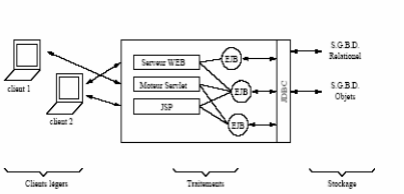
Client/Serveur (architecture deux parties) :



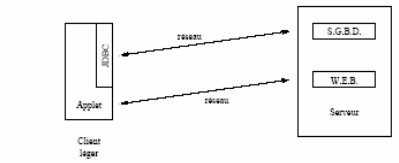
Architecture trois parties :



Architecture d'exécution répartie dans la norme J2EE :

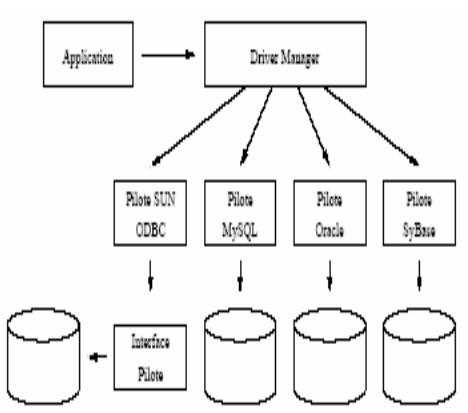


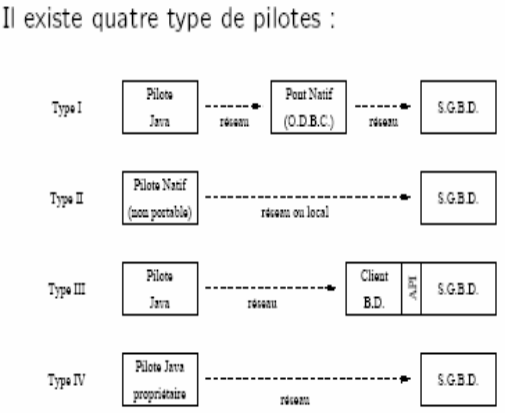
Utilisation de JDBC dans un client léger :



II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Structure logicielle





II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Déclaration du pilote JDBC

- L'appel a **`forName`** déclenche un chargement dynamique du pilote.
- Un programme peut utiliser plusieurs pilotes, un pour chaque base de données.
- Le pilote doit être accessible à partir de la variable d'environnement CLASSPATH.

```
import java.sql.*;

public class ExempleJDBC
{
public static void main(String[] Args)
    {
    try {
        Class.forName("org.gjt.mm.mysql.Driver");
        }
    catch (Exception E) {
        System.err.println("Pas de pilote !");
        }

    ... connexion et utilisation de la base ...
    }
}
```

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Connexion à la base de donnée

Voici le code type :

```
try {
String url = "jdbc:mysql://localhost/dbessai";
Connection conn = DriverManager.getConnection(URL, "user", "password");
... utilisation de la base ...
}
catch (SQLException E) {
System.err.println(E.getMessage());
}
```

L'URL est de la forme : jdbc:sous-protocole:sous-nom

Exemples :

jdbc:oracle:thin:@quad.unice.fr:1521:TDINFO"
jdbc:odbc:msql;USER=fred;PWD=secret

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Les requêtes en JDBC

Voici une utilisation type :

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(
"SELECT Nom,Prenom,Age FROM personne " + "ORDER BY age");
while (rs.next()) {
        System.out.println("Nom : " + rs.getString(1));
        System.out.println("Prenom : " + rs.getString(2));
        System.out.println("Age : " + rs.getString(3));
}
rs.close(); // Fermeture
st.close();
conn.close();
```

2. JDBC : La classe java.sql.ResultSet

- Accès aux valeurs :
  - ✓Type getType(int)
  - ✓Type getType(String)
  - ✓boolean next();

Le Type peut être

Byte	Bytes
Short	Date
Int	Time
Long	TimeStamp
Float	AsciiStream
BigDecimal	UnicodeStream
Boolean	BinarySt

2. JDBC : La classe java.sql.ResultSet

- **Les types Java / SQL**

SQL	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double

BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.TimeStamp

Pour les dates :
java.sql.Date : codage de la date,
java.sql.Time : codage de l'heure,
java.sql.TimeStamp : date et heure,

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Appel de procédures stockées dans le SGBD

Appel type :

```
CallableStatement st = conn.prepareCall("{call nom_de_fonction stockée[(?,?)]}"
);
// fixer le type de parametre de sortie
st.registerOutParameter(2, java.sql.Types.FLOAT);
st.setInt(1, valeur); // fixer la valeur du param ètre
st.execute();
System.out.println("resultat = " + st.getFloat(2));
```

- Avantages :
- efficacité (moins de transfert de données),
- compilation des procédures
- Inconvénient : pas de norme !

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Erreurs et warnings

- La classe java.sql.SQLException enrichit la classe java.lang.Exception :
  - SQLState : description de l'erreur au format XOPEN,
  - getNextException()
- La classe java.sql.SQLWarning enrichit la classe java.sql.getWarnings() :
  - Warning suivant (il réalise des appels répétés).

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Gestion des transactions

- Le mode par defaut est Auto Commit
- On peut utiliser le mode manuel en désactivant le mode Auto Commit :
  - connexion.setAutoCommit(false) ;
- Les commandes manuelles de gestion des transactions :
  - connexion.commit() ;
  - connexion.rollback() ;

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Méta informations sur les « Result Set » et Base de données

Exemple 1

```
ResultSetMetaData m = rs.getMetaData();
```

- **Informations disponibles :**
  - nombre de colonnes
  - Libelle d'une colonne
  - table d'origine
  - type associe a une colonne
  - la colonne est-elle nullable ?
  - etc.

Exemple 2

```
DataBaseMetaData dbmd = connexion.getMetaData() ;
```

- **Informations disponibles :**
  - tables existantes dans la base
  - nom d'utilisateur
  - version du pilote
  - prise en charge des jointure externes ?
  - etc.

- **Avantages :**
  - Code indépendant
  - Code réutilisable !

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Compléments sur les « Result Set »

- **Il existe quatre types de Result Set :**
  - Scroll-insensitive : Vision figée du résultat de la requête au moment de son évaluation.
  - Scroll-sensitive : Le Result Set montre l'état courant des données (modiees/detruites).
  - Read-only : Pas de modification possible (JDBC 1.0) donc un haut niveau de concurrence.
  - updatable : Possibilité de modification donc pose de verrou et faible niveau de concurrence.

- **JDBC 2.1 : déplacement dans un Result Set**
  - rs.first();
  - rs.beforeFirst();
  - rs.next();
  - rs.previous();
  - rs.afterLast();
  - rs.absolute(n);
  - rs.relative(n);

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Compléments sur les « Result Set »

- **Modication :**
  - rs.absolute(100);
  - rs.updateString("Nom", "Fred");
  - rs.updateInt("Age", 30);
  - rs.updateRow();
- **Destruction :**
  - rs.deleteRow();
- **Insertion :**
  - rs.moveToInsertRow();
  - rs.updateString("Nom", "Fred");
  - rs.updateInt("Age", 30);
  - rs.insertRow();
  - rs.first();

- **Regroupement de plusieurs mise a jour :**
  - connexion.setAutoCommit(false);
  - Statement st = connexion.createStatement();
  - st.addBatch("INSERT ...");
  - st.addBatch("INSERT ...");
  - int[] nb = st.executeBatch();

II- APPROCHE DE MEDIATEUR : ODBC et JDBC

2. JDBC : Compléments sur les « Result Set »

- **Améliorations :**
  - Save point : pose de point de sauvegarde.
  - Connection Pool : Gestion des ensembles de connexions partagees.
  - Support des sequences (auto generation de valeurs).
  - Augmentation et mise a jour des types

- **Les RowSet :**

```
javax.sql.rowset.CachedRowSet rs =
new com.sun.rowset.CachedRowSetImpl();
rs.setUrl("jdbc:mysql://localhost/dbessai");
rs.setCommand("SELECT * FROM personne");
rs.setUsername("massat");
rs.setPassword("...");
rs.setConcurrency (ResultSet.CONCUR_UPDATABLE);
rs.execute();
while (rs.next()) {System.out.println("Nom : " + rs.getString("nom"));
}
rs.close();
```

- Il existe trois types de RowSet :

- JDBCRowSet (base sur JDBC),
- CachedRowSet (déconnecte de la base),
- WebRowSet (échange base sur des flux XML)

III- APPROCHE DE programmation Web : Servlet et JSP

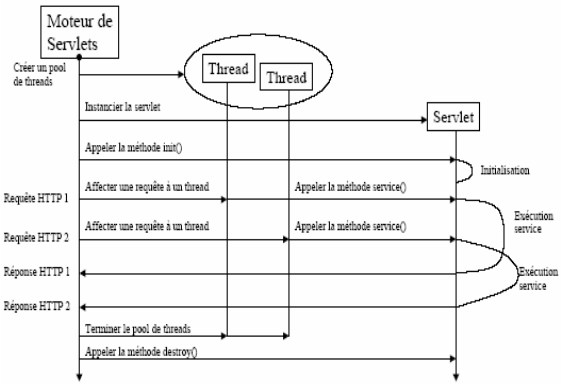
1. SERVLET

- **Qu'est-ce qu'un** `Servlet`
  - **programme composant d'un serveur Web ou d'un serveur d'applications**
  - **classe** `Java` **gérée par un** `Servlet Container`
    - **Chargée dynamiquement par le serveur**
    - **engendre des contenus dynamiques**
    - **interagit avec un client en mode** `request/response`
  - **correspond**
    - **aux** `cgi-bin`
    - **aux Nescape Server APIs**
    - **aux Modules Apache**

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET

- **Servlets plus efficaces**
  - Résidentes, pas de fork, pas de temps de lancement
  - Multithreads
  - Gestion de cache
  - Connexions persistantes (BD)
  - etc...



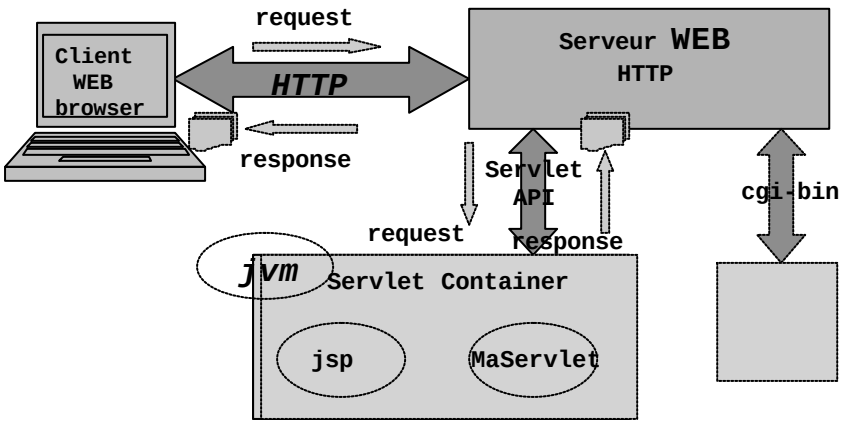
III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : présentation

- Servlet Container
  - **Partie d'un serveur WEB ou d'un serveur d'applications (type J2EE)**
    - **soit partie intégrante du serveur**
    - **soit composant ajouté**
    - **soit serveur lui-même**
  - **Protocoles supportés**
    - **HTTP1.1, HTTPS**
    - **Obligatoire dans un serveur Java2EnterpriseEdition(J2EE)**

1. SERVLET : présentation

- Servlet Container : fonctionnement



1. SERVLET : Modèle de programmation

- Une servlet doit implémenter l'interface **javax.servlet.Servlet**
  - soit directement,
  - soit en dérivant d'une classe implémentant déjà cette interface comme (**GenericServlet ou HttpServlet**)
- Cette interface possède les méthodes pour :
  - initialiser la servlet : **init()**
  - recevoir et répondre aux requêtes des clients : **service()**
  - détruire la servlet et ses ressources : **destroy()**

1. SERVLET : Le Package `javax.servlet`

- **L'interface** `Servlet`
  - **Abstraction racine de toutes les implémentations des servlets**
  - **Déclare les méthodes pour gérer un servlet et pour communiquer avec le client Web**
  - **Ces méthodes sont implémentées par exemple par la classe** `HttpServlet` **ou par une classe dérivée** `HttpServlet` **écrite par le programmeur**

1. SERVLET : Le Package `javax.servlet`

- Structure d'une servlet

```
import javax.servlet.*;
public class first implements Servlet {
public void init(ServletConf config)
throws ServletException {…}
public void service( ServletRequest req,
ServletResponse rep)
throws ServletException, IOException {…}
public void destroy() {…}
}
```

1. SERVLET : Le Package `javax.servlet`

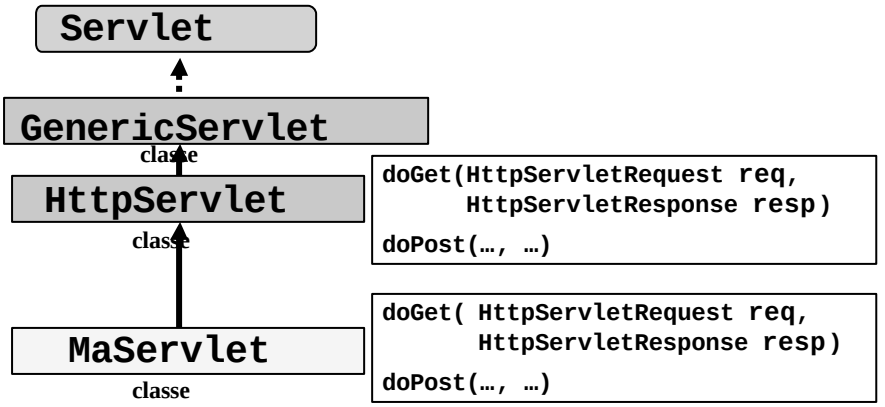
- Le cycle de vie d'une Servlet
  1. la *servlet* est crée puis initialisée (**init()**)
     - » cette méthode n'est appelée par le serveur qu'une seule fois lors du chargement en mémoire par le moteur de servlet
  2. le service du client est implémenté (**service()**)
     - » cette méthode est appelée automatiquement par le serveur à chaque requête de client
  3. la *servlet* est détruite (**destroy())**
     - » cette méthode n'est appelée par le serveur qu'une seule fois à la fin
     - » permet de libérer des ressources (allouées par init())

1. SERVLET : Le Package `javax.servlet`

- Une Servlet Web : **HttpServlet**
  - Pour faciliter le traitement particulier des serveurs Web, la classe **Servlet** est affinée en **javax.servlet.http.HttpServlet avec** 2 méthodes qui remplacent **service()** de la classe mère :
    - » **doGet()** : pour les requêtes Http de type GET
    - » **doPost()** : pour les requêtes Http de type POST
  - la classe *servlet* doit obligatoirement contenir l'une ou l'autre de ces 2 méthodes redéfinie, choisie selon le mode d'envoi du formulaire HTML qui l'exécute
  - **service()** de **HttpServlet** appelle automatiquement la bonne méthode en fonction du type de requêtes Http

1. SERVLET : Le Package `javax.servlet`

- **L'interface** `Servlet`



1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client du** `Servlet`
  - **Lors d'une communication avec un client un** `Servlet` **reçoit deux objets à travers une parmi 2 méthodes**
    - **Un objet** `ServletRequest` **qui encapsule la communication** venant du **client**
    - **Un objet** `ServletResponse` **qui encapsule la communication** vers **le client**

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client du** `Servlet`

  – **Capture des requêtes HTTP de type GET**

  - ***Lorsque le client envoie un URL ou un formulaire avec action=GET***

    `http://monserveur/unservlet?nom=toto`

  - **On peut répondre avec la méthode** `doGet` :

```
doGet(  HttpRequest req,
        HttpRequest resp)
```


1. SERVLET : Le Package `javax.servlet`

- Squelette d'une `Servlet` HTTP (GET)

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {
public void init(HttpServletConfig c)
throws ServletException {…}
public void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {…}
public void destroy() {…}
public String getServletInfo() {…}
}
```

1. SERVLET : Le Package javax.servlet

- **La communication avec le client d'une** `Servlet`
  - **Capture des requêtes HTTP de type POST**

➢ ***Lorsque le client envoie un formulaire*** **:**

```
<form action="http://monserveur/unservlet" method=POST>
   <input type=text size=20 name=nom>
</form>
```

➢ **On peut répondre avec la méthode** `doPost` **:**

```
doPost(  HttpRequest req,
         HttpResponse resp)
```


1. SERVLET : Le Package javax.servlet

- **La communication avec le client d'une** `Servlet`
  - ***On doit renvoyer le type MIME de la réponse***

```
public void doGet(  HttpServletRequest  req, HttpServletResponse resp)
        throws ServletException, IOException
  {resp.setContentType("text/html");
   PrintWriter out = resp.getWriter();
   out.println("<html>"); out.println("<body>");
   out.println("<head>");        out.println("<title>Essai1
 </title>");      out.println("</head>");
   out.println("<body>");
   out.println("<h3>ESSAI1 </h3> <br> Coucou");
   out.println("</body>");
   out.println("</html>");
  }
```

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **Structure de base d'une `Servlet`**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
      HttpServletResponse response)
    throws ServletException, IOException {
    // Use "request" to read incoming HTTP headers (e.g. cookies)
    // and HTML form data (e.g. data the user entered and submitted)
    …
    // Use "response" to specify the HTTP response line and headers
    // (e.g. specifying the content type, setting cookies).
    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
    }
}
```


III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletResponse`
  - **Interface implémenté par le** servlet engine
  - **Type de l'objet qui permet de répondre au client (dépendant de l'objet passé comme paramètre dans `doGet` ou `doPost`)**

III- APPROCHE DE programmation Web : Servlet et JSP

1. **SERVLET : Le Package** `javax.servlet`

- **La communication avec le client:** `HttpServletResponse`

➢ **Méthodes (implémentées par le Servlet Engine)**

```
public void setContentType( java.lang.String type)
```
// pour spécifier le type du document créé :`"text/html", ….`

```
public java.io.PrintWriter getWriter()
       throws java.io.IOException
```
// pour pouvoir ensuite écrire des données pour le client

III- APPROCHE DE programmation Web : Servlet et JSP

1. **SERVLET : Le Package** `javax.servlet`

- **La communication avec le client:** `HttpServletResponse`

➢ **Méthodes (implémentées par le Servlet Engine)**

```
public void addCookie(Cookie cookie)
```
// pour ajouter un cookie de plus dans la réponse

```
public void sendRedirect(java.lang.String URLlocation)
   throws java.io.IOException
```
// pour rediriger la réponse vers un autre URL
// (en fait le nouvel URL est renvoyé au client)

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ **Interface implémenté par le** servlet engine

➢ **Type de l'objet qui permet de récupérer les données du client (paramètre, IP, …)**

(dépensez de l'objet passé comme paramètre dans `doGet` **ou** `doPost`**)**

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ **Méthodes (implémentées par le Servlet Engine)**

```
public java.lang.String getParameter(
                    java.lang.String name)
```

- **pour récupérer le paramètre du formulaire grace à son nom**
- `null` **si pas de paramètre de ce nom**

Exemple

```
    if ( req.getParameter("serveur").equals("oracle") )
  {……}
```

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ **Méthodes (implémentées par le Servlet Engine)**

```
public java.lang.String []
    getParameterValues(java.lang.String name)
```

pour récupérer un tableau de valeurs pour un paramètre multi-valué du formulaire

Exemple

```
if ( req.getParameterValues.length() == 1 ){……}
```


III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ **Méthodes (implémentées par le Servlet Engine)**

```
public java.lang.Enumeration
    getParameterNames()
```

pour récupérer une énumération de tous les noms de paramètres du formulaire envoyé par le client

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ Méthodes (implémentées par le Servlet Engine)

```
public Cookie [] getCookies()
```

// pour récupérer un tableau de tous les cookies stockés chez le client
// `null` si pas de cookies

```
public HttpSession getSession()
```

// Pour récupérer l'objet représentant la session courante avec ses
// données persistantes. Si la session n'existe pas, en crée une

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpServletRequest`

➢ Méthodes (implémentées par le Servlet Engine)

```
public HttpSession getSession()
```

Pour récupérer l'objet représentant la session courante avec ses données persistantes

Si la session n'existe pas, en crée une

```
public HttpSession getSession(boolean create)
```

Si `create==true` **:**

Crée la session si elle n'existe pas

Si `create==false` **:**

Retourne `null` **si la session n'existe pas**

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpSession`

- **Interface (implémentées par le Servlet Engine)**
- **Type de l'objet session récupéré dans un requête**
- **Permet de garder des objets ("nommés") persistants**
- **Les objets stockés sont aussi appelés "valeurs"**
- **Implémenté avec un sessionId stocké dans les cookies du client**

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Le Package `javax.servlet`

- **La communication avec le client:** `HttpSession`

- **Méthodes (implémentées par le Servlet Engine)**

```
public void putValue(java.lang.String name,
                     java.lang.Object value)
```

Elimine de l'objet session tout couple pré-existant avec le même `name`
Stocke dans l'objet session le nouveau couple `name/value`

```
public void removeValue(java.lang.String name)
```

Elimine de la session le couple avec le même `name` **(s'il existe)**

```
public java.lang.Object getValue(java.lang.String name)
```

Récupère la valeur associée à `name` **dans la session** `null` **si le couple n'existe pas**

1. SERVLET : Le Package javax.servlet

- **Exemple : envoi d'informations au client**

```
import java.io.*; import java.servlet.*;
import javax.servlet.*;
public class DemandeInfos extends HttpServlet {
      public void doGet(HttpServletRequest request,HttpServletResponse
      response)
      throws IOException, ServletException
 {
      response.setContentType("text/html"); PrintWriter out =
      response.getWriter();
      out.println("<html>");out.println("<body>"); out.println("<head>");
      out.println("<title>DemandeInfos</title>"); out.println("</head>");
      out.println("<body>");
      out.println("<h3>Informations</h3>");
```


1. SERVLET : Le Package javax.servlet

- **Exemple : envoi un grille de saisie au client**

```
import java.io.*; import java.servlet.*;
import javax.servlet.*;
public class DemandeInfos extends HttpServlet {
      public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws IOException, ServletException
 {
      response.setContentType("text/html"); PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<body>");
      out.println("<head>");
      out.println("<title>DemandeInfos</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h3>Informations</h3>");
      out.println("Method: " + request.getMethod());
      out.println("Chaine URI:"+request.getRequestURI());
      out.println("AddresseIP:"+request.getRemoteAddr());
      out.println("</body>");
      out.println("</html>");
      }
```

1. SERVLET : Le Package javax.servlet

- **Exemple : envoi un grille de saisie au client (suite)**

```
import java.io.*; import java.servlet.*;
import javax.servlet.*;
public class DemandeInfos extends HttpServlet {
      public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws IOException, ServletException
  {
      … … …
  }

      public void doPost(HttpServletRequest req, HttpServletResponse rep)
         throws IOException, ServletException
  { doGet(req, rep);
  }
}
```


1. SERVLET : Application WEB avec TOMCAT

- **Exemple d'utilisation**

```
package hall;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
    }
}
```

• Compilation et installation

- Une application web = un espace virtuel
  - » Contient html, images, servlets, jsp...
  - » Avec Tomcat : Editer <tomcat_dir>/server.xml pour définir une application Web

```
<Context path=« nlt" docBase="pit"
defaultSessionTimeOut="30" isWARExpanded="true"
isWARValidated="false" isInvokerEnabled="true"
isWorkDirPersistent="false"
/>
```

- " Dans cet exemple, l'application web se nomme pit
- " Mettre les classes dans <tomcat_dir>/**nlt**/WEB-INF/classes
- " Editer <tomcat_dir>/**nlt**/WEB-INF/web.xml

• Application web

- L'installation varie d'un serveur à l'autre
- Pour compiler :
- Positionner le `classpath`, puis depuis le répertoire hall/ :
- `javac -classpath .:<tomcat_dir>/lib/servlet.jar HelloWorld.java`
- Deux manières de tester :
  - » Avec JSWDK, mettre hall.HelloWorld dans le répertoire d'exemple
  - » Avec Apache Tomcat : créer une « application web »

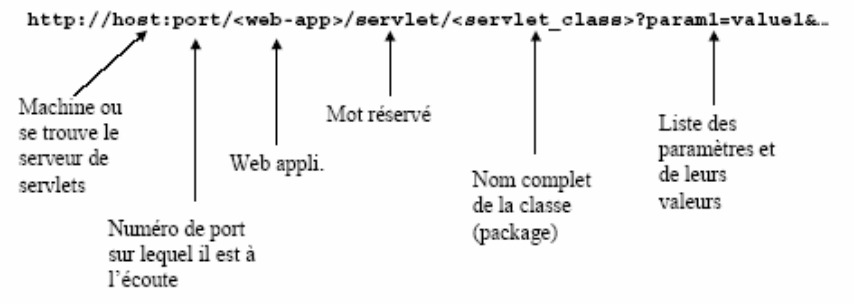
- **Application Web (suite)**
  Avec le fichier <tomcat_dir>/nlt/WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
    <servlet>
     <servlet-name>
            HelloWorld
     </servlet-name>
     <servlet-class>
            hall.HelloWorld
     </servlet-class>
    </servlet>
</web-app>
```


- **Application Web (suite)**
  - Une fois une application web créée, on peut mettre autant de servlets que l'on veut...
  - Relancer Tomcat à chaque modif des fichiers XML server.xml et/ou web.xml
  - Pour invoquer la servlet, utiliser l'alias :
    - » **http://host/nlt/servlet/HelloWorld**

  ou le nom complet...
    - » **http://host/nlt/servlet/hall.HelloWorld**

- **Chargement / invocation d'une servlet**
  - D'une manière générale, une URL du type :



- **Récupérer les paramètres passés à la servlet**
  - Utilisation des méthodes de **ServletRequest** :

```
public void doGet( HttpServletRequest req,
                   HttpServletResponse rep)
throws ServletException, IOException
{
   String[] values = req.getParameterValues();
   Enumeration list = req.getParameterNames();
   String value1 = req.getParameter("param1");
   if(value1 == null) ...
}
```

- **Un autre exemple de « Hello »**

```
package hall;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n" + "<HTML>\n" + "<HEAD><TITLE>Hello
    WWW</TITLE></HEAD>\n" + "<H1>Hello WWW</H1>\n" +"</BODY></HTML>");
    }
}
```


- La partie compliquée = paramètres du formulaire
- Visibles ou non dans l'URL (GET : visible /POST: non visible)
- Ces paramètres doivent être décodés !
  - " Partie la plus difficile. Encodage = norme CGI

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Gestion des formulaires HTML

- Récupèrer les paramètres
- Méthode getParameter() de HttpServletRequest (Fonctionne avec GET ou POST)

```
public class ShowParameters extends HttpServlet {
    public void doGet(HttpServletRequest request,...) {
    out.println(request.getParameter("param1") );
    Enumeration paramNames=request.getParameterNames();
    String[] paramValues =
            request.getParameterValues(paramNames);
   }
   }
```


III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Gestion des cookies

- C'est quoi ? : Morceaux d'informations envoyés par le serveur … et renvoyés par le client quand il revient visiter le même URL
  - Durée de vie réglable
  - Permet la persistance
- A quoi ça sert ?
  - Identification des utilisateurs (e-commerce)
    - » Eviter la saisie d'informations à répétition
    - » login, password, adresse, téléphone…
  - Gérer des « préférences utilisateur »
    - » sites portails …

1. SERVLET : Gestion des cookies

- Attention : sécurité :
  - Jamais interprété ou exécuté : pas de virus
  - Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
  - Bien pour rendre privées des données non sensibles
    - » nom, adresse, … mais pas No CB !
  - … mais ne constitue pas un traitement sérieux de la sécurité

1. SERVLET : Gestion des cookies

- Manipulation de cookies :
  - Utiliser les fonctions de l 'API des servlets…
    - » créer un cookie : classe **Cookie**,
    - » écrire/lire un cookie : **addCookie(cookie), getCookies()**,
    - » positionner des attributs d'un cookie : **cookie.setXxx**(…)
  - Exemple d'envoi d'un cookie :

```
...
String nom = request.getParameter("nom");
Cookie unCookie = new Cookie("nom", nom);
...ici positionner des attributs si on le désire
response.addCookie(unCookie);
```

- Manipulation de cookies : Création d'un cookie

```
Cookie unCookie = new Cookie(name, value);
```

  - 2 arguments de type **java.lang.String** :
    - » **name** et **value**
  - caractères non autorisés :
    - » espace blanc
    - » [ ] ( ) = , " / ? @ : ;

- Manipulation de cookies : Attributs des cookies
  - **getValue/setValue**
  - **getName/setName**
  - **getComment/setComment**
  - **getMaxAge/setMaxAge** : délai restant avant expiration du cookie (en seconde)
    - » par défaut : pour la session courante
  - **getPath/setPath** : répertoire où s'applique le cookie
    - » dir. courant ou pages spécifiques

- Manipulation de cookies : Récupération des cookies

```
Cookie [] cookies = request.getCookies();
String nom = getCookieValue(cookies, "nom", "non trouvé");
...
public static String getCookieValue (Cookie [] cookies, String
    cookieName, String defaultValue) {
    for(int i=0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if(cookieName.equals(cookie.getName())
    return(cookie.getValue());
    }
return(defaultValue);
}
```


- Manipulation de cookies : Expiration des cookies
  - Par défaut, durée de vie d'un cookie = la connexion.
  - Si on veut que le cookie soit sauvé sur disque, modifier sa durée de vie :

```
public static final int SECONDS_PER_YEAR =
60*60*24*365;
cookie.setMaxAge(SECONDS_PER_YEAR);
```

1. SERVLET : Gestion des session

- Problématique
  - Protocole HTTP = protocole Internet déconnecté
    - » différent de Telnet, Ftp, …
    - » traite les requêtes et les réponses comme transactions simples et isolées (requêtes non apparentées)
  - Certaines applications Web (e-commerce : caddie) ont besoin de maintenir une "mémoire" entre deux requêtes
    - » ie. maintenir une connexion de l'utilisateur sur le serveur
    - » pour se faire : concept de "suivi de sessions"

1. SERVLET : Gestion des session

- Suivie des sessions :
  - Mémoire de ce que fait l'utilisateur d'une page à l'autre
    - » consiste au transfert de données générées par une requête vers les requêtes suivantes
  - 4 méthodes avec les servlets Java
    1) utilisation des cookies (déjà vu)
    2) réécriture d'URL : passage de paramètres
    3) utilisation des champs de formulaire "hidden"
    4) utilisation du JSDK (HttpSession API)

1. SERVLET : Gestion des session

- Réécriture de l'URL : Principe
  - ajouter dans la chaîne de requête de la servlet des informations supplémentaires identifiant la session

```
…
<a href="http://liserv1.unice.fr/servlet/foo?uid=nlt">Acheter
</a>
```

  - l'ID utilisateur est transmis en même temps que la requête; il est accédé par chaque `servlet` mentionnée qui récupère les informations persistantes (BD, fichiers) à partir de cet ID
  - Limitations :
    - » données volumineuses, caractères autorisés, longueur URL, données visibles (sécurité)

1. SERVLET : Gestion des session

- Champ de formulaire caché : Principe
  - on cache les données de session dans des champs "hidden" :

```
<INPUT TYPE="HIDDEN" NAME="uid" VALUE=nlt">
```

  - Limitations :
    - » idem la "réécriture d'URL" sauf pour la sécurité (utilisation de POST)

1. SERVLET : Gestion des session

- Servlet session
  - Très simple avec l'API des servlets (JSDK)
    - » objet **HttpSession**
  - Principe :
    - » Un objet "session" peut être associé *avec chaque requête*
    - » Il va servir de "container" pour des informations persistantes
    - » Durée de vie limitée et réglable

1. SERVLET : Gestion des session

- Servlet session : modèle basique

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getValue("caddy");
if(caddy != null) {
   // le caddy n'est pas vide !
   afficheLeContenuDuCaddy(caddy);
   } else {
   caddy = new Caddy();
...
caddy.ajouterUnAchat(request.getParameter("NoArticle"));
session.putValue("caddy", caddy);
}....
```

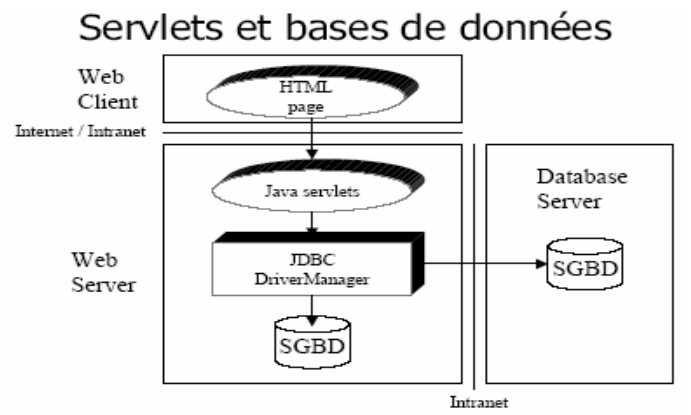
III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Gestion des session

- Servlet session : les méthodes de la classe httpSession
  - `getID()`
  - `isNew()`
  - `getCreationTime() / getLastAccessedTime()`
  - `getMaxInactiveInterval()`
  - …
  - `getValue(), removeValue(), putValue()`
  - …
  - `invalidate()`

III- APPROCHE DE programmation Web : Servlet et JSP

1. SERVLET : Connexion Base de données



III- APPROCHE DE programmation Web : Servlet et JSP

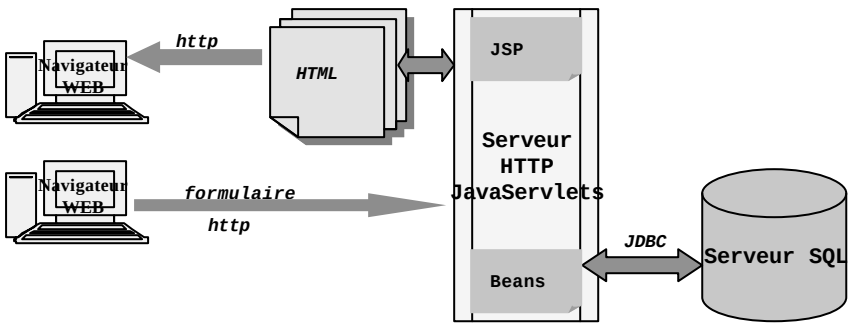
2. JSP (Java Server Pages)

- Solution de Java alternative à ASP/PHP, etc...
- Technologie qui permet de mixer Java et HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
<H1>Welcome to Our Store</H1>
<SMALL>Welcome,
<!-- User name is "New User" for first-time visitors -->
<% out.println(Utils.getUserNameFromCookie(request)); %>
To access your account settings, click
<A HREF="Account-Settings.html">here.</A></SMALL>
<P>
Regular HTML for all the rest of the on-line store's Web page.
</BODY></HTML>
```


III- APPROCHE DE programmation Web : Servlet et JSP

2. JSP (Java Server Pages)



- **beans : services** Java **réalisant la couche métier**
- **JSP : permet me mixage** Java/HTML**. En charge du dialogue**

2. JSP (Java Server Pages)

- **Un exemple :**

```
<html><head><title>Un exemple de page JSP</title></head><body>
<!-- définit les informations globales a la page -->
<%@page language="java" %>
<!-- Déclare la variable c -->
    <%! char c = 0; %>
<!-- Scriplet (code java) %>
<%
    for(int i = 0; i < 26; i++){
     for(int j = 0; j < 26; j++){
              c = (char)(0x41 + (26 - i + j)%26);
%>
<%= c %>
<% } %>
<br>
<% } %>
</body></html>
```


2. JSP : Trois types de balise

- Trois types de balise utilisés dans html :

1 - *Scripting elements* : pour le code java

2 - *directives* : pour le contrôle de la structure

3 - *actions* : pour l'importation de composants existants

2. JSP : Trois types de balise

- *Scripting elements* :
  - Forme : <%= expression %>
  - Exemple : `Il est` **`<%= new java.util.Date() %>`** `<P> et votre hostname est` **`<%= request.getRemoteHost() %>`**
  - permet d'intégrer des valeurs dans le code HTML
  - ces valeurs sont évaluées, converties en chaînes de caractères et affichées
  - les objets implicites (request, response, session, out, ...) disponibles

2. JSP : Trois types de balise

- *Scripting elements (suite)*:
  - Forme « scriplets » : <% code Java %>
  - Exemple :
```
<% String nom = request.getParameter("nom");
...
out.println("Nom de l'utilisateur " + nom);
%>
```
  - c'est un bloc de code Java placé dans **_jspService()** de la servlet générée ayant accès :
    - » aux variables et *beans* déclarés (**<%! … %>**)
    - » aux objets implicites (voir plus loin)

2. JSP : Trois types de balise

- *Scripting elements (suite)*:
  - Forme «déclations» : <%! déclations %>
  - Exemple :

```
<%!   private int accessCount = 0;
            private int incrementCount() {accessCount++;}
%>
...
  <H2>Nombre et liste des articles</H2>
Nombre d'articles : <%= incrementCount() %>
```

  - définitions des méthodes et variables de classe à utiliser dans toute la page
  - définit les méthodes **jspInit()** et **jspDestroy()**

2. JSP : Trois types de balise

- *Directives* :
  - Forme : `<%@ directive attribut1="valeur" attribut2="valeur"... %>`
  - 2 directives possibles (jsp1.0) :
    - » page : informations relatives à la page
    - » include : fichiers à inclure littéralement

- *Directives* (suite) : *Page*
  - Valeurs possibles :
    - » <%@ page language="java"
    - » <%@ page import="java.util.*, java.net.*" %>
    - » <%@ page contentType="text/plain" %>
    - » <%@ page session="true|false " %>
    - » <%@ page errorPage="pathToErrorPage" %>
    - » <%@ page isErrorPage="true|false" %>
    - » n<%@ page …

- *Directives* (suite) : *include*
  - Valeurs possibles :
    - » <%@ include file="chemin relatif du fichier" %>
    - » Exemple : pour se référer au home dir du serveur Web : `<%@ include file="/toto.html" %>`
    - » Interprété littéralement, le fichier peut être :
      - HTML, scripting elements, directives, actions, ...
      - L'insertion se fait au moment de la traduction de la page...

2. JSP : Variables prédéfinies

- Ou "objets implicites", ils sont accessibles dans les *scripting elements* :
  - request : le **HttpServletRequest**
  - response : le **HttpServletResponse**
  - session **:** le **HttpSession**
  - out : flot de sortie (idem **response.getWriter())**
  - application : le **ServletContext**

  (idem **getServletConfig().getContext()**)
  - config, pageContext, page... : peu utiles

2. JSP : Actions

- Permettent de faire des actions au moment où la page est demandée par un client :
  - inclure dynamiquement un fichier
  - utiliser des *beans*
  - rediriger vers une autre page
  - etc...
- Syntaxe XML

2. JSP : Actions (suite)

- **<jsp:include page="relative URL" flush="true" />**
  - inclusion au moment où la page est servie, pas au moment où elle est traduite en servlet.
- **<jsp:usebean id="name" class="package.class" />**
  - permet d'instancier un *bean* depuis une page JSP.
  - nécessite de connaître le mécanisme des *beans*…
  - associé à **<jsp:getProperty.../>** et **<jsp:setProperty.../>**
- **<jsp:forward page="/unAutreURI" />**
  - redirige vers un autre URI/URL

2. JSP : Usebean et getProperty

- Mécanisme très puissant !
- **Format : <jsp:usebean**

  id="name" (référence l'instance du composant)
  class="paquetage.class" (nom qualifié de la classe)
  scope="page|request|session|application" (portée)
  />
- Pour lire une propriété du *bean* :
- **<jsp:getProperty name="name" property="property" />**

2. JSP : Usebean et getProperty

- Pour modifier une propriété du *bean* :

```
<jsp:setProperty
    name="name"
    property="property"
    value="value" />
<jsp:setProperty name="name" property="*" />
```

- Initialise tous les attributs de l'objet **name** avec les paramètres HTTP du même nom
- En 2 lignes !

2. JSP : Usebean et getProperty

- Exemple d'utilisation d'un *bean* :
  - La page JSP

```
<html> ...
<jsp:usebean id="test" class="inria.SimpleBean" />
<jsp:setProperty name="test" property="message"
value="Hello !!" />
<h1>Le message est : <i>
<jsp:getProperty name="test" property="message" />
</i></h1>…
</html>
```

2. JSP : Usebean et getProperty

- Exemple d'utilisation d'un *bean* : (suite)
  - Le code source Java du *bean* :

SimpleBean.java

```
package tpinfo;
public class SimpleBean {
    private String message = "no message";
    public String getMessage() {
            return message;
    }
    public void setMessage(String message) {
            this.message = message;
    }
}
```


V- PROGRAMMATION CLIENT-SERVEUR

III- APPROCHE DE programmation Web : Servlet et JSP

2. JSP : Usebean et getProperty

- Exemple d'utilisation d'un *bean* : (suite)
  - Le code source Java du *bean* :

SimpleBean.java

```
package tpinfo;
public class SimpleBean {
    private String message = "no message";
    public String getMessage() {
            return message;
    }
    public void setMessage(String message) {
            this.message = message;
    }
}
```