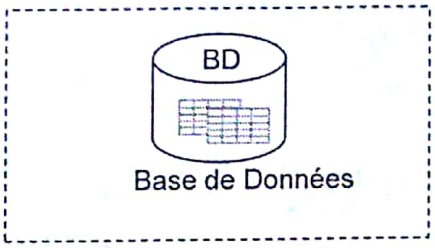
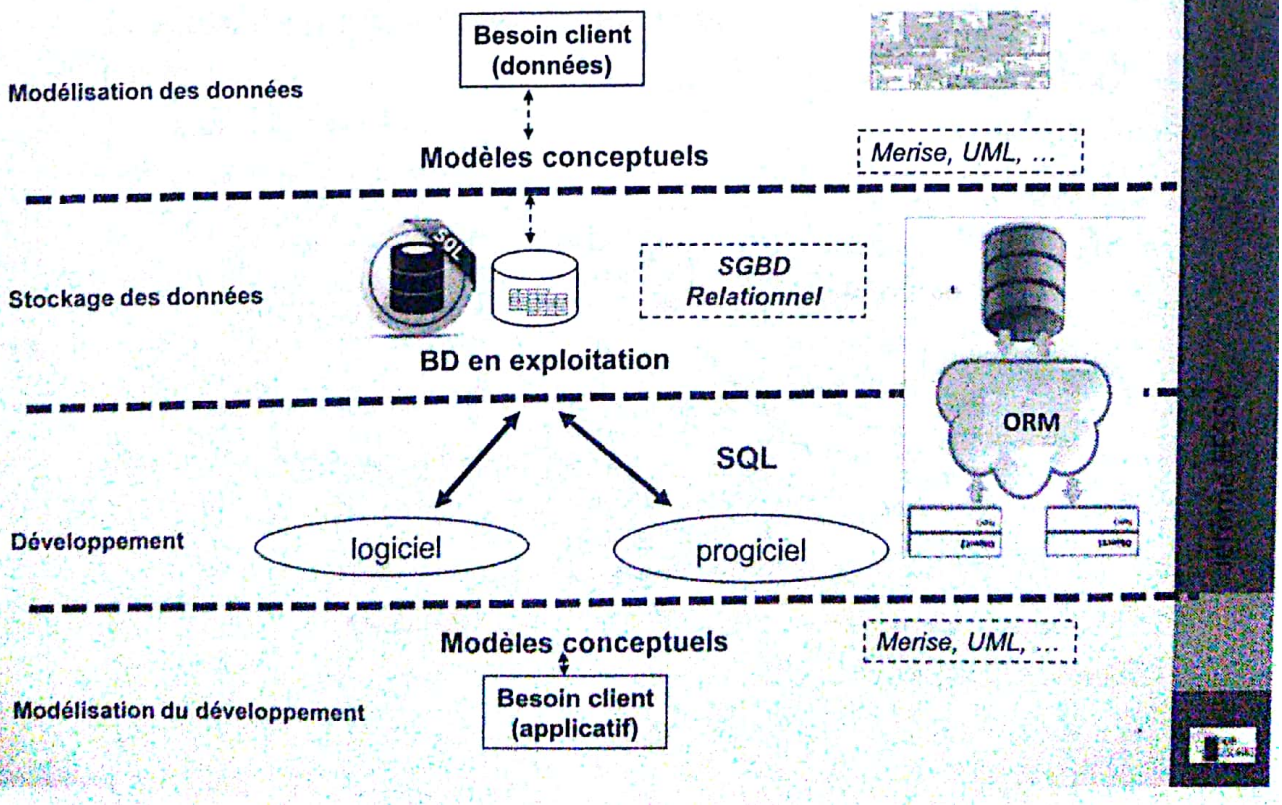
SGBD RELATIONNELS

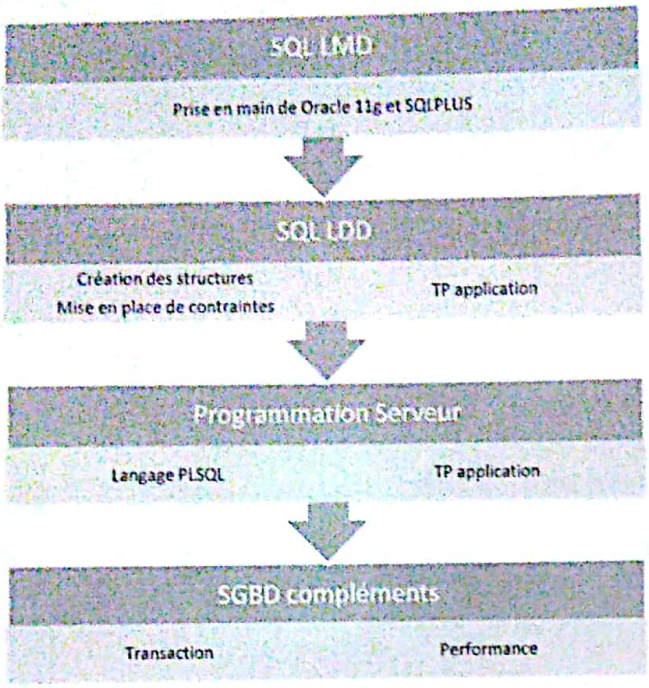
Programmation serveur



Etapes d'un projet SI standard



Le module SGBD2 (M2106)?



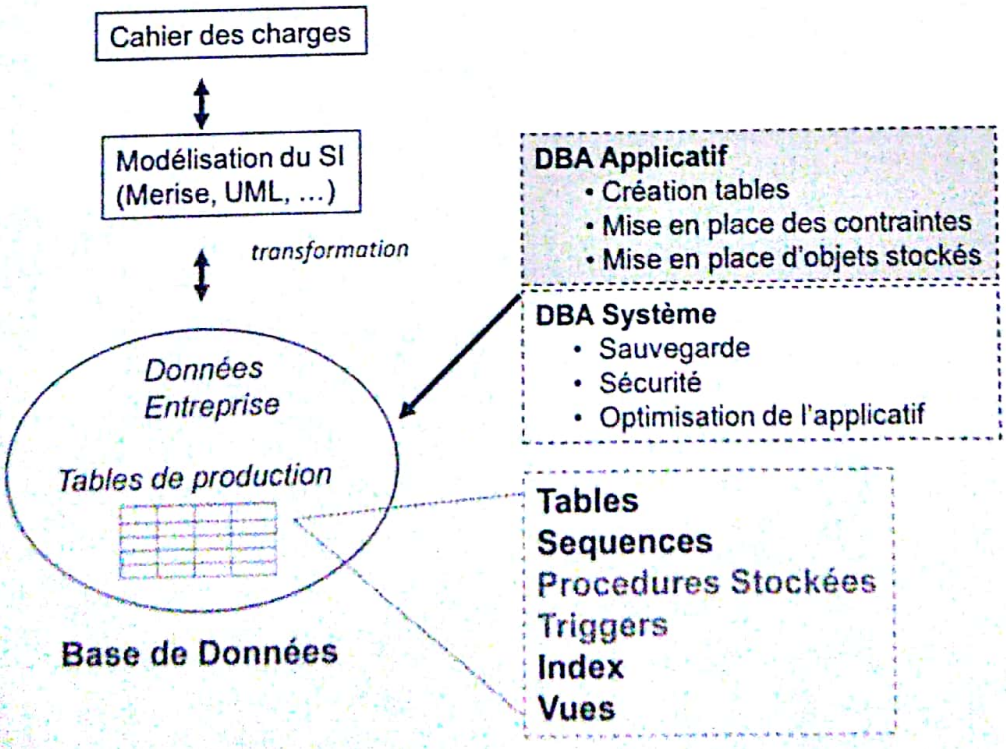
Application à
ORACLE 11G

DST avec document
Basé sur un projet de révision

Objectif

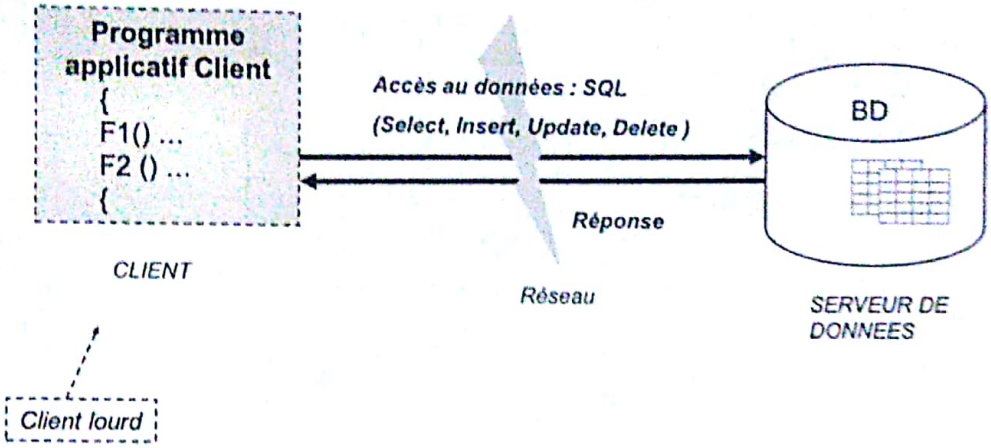
Maîtriser la mise en exploitation
des données d'une entreprise

Base de données de production



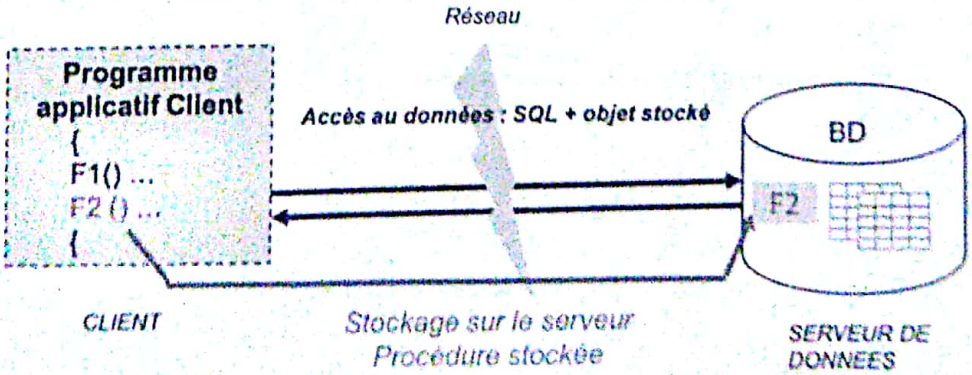
Architecture Client / Serveur

- Dichotomie Données / Programme applicatif
- Tout accès SQL se fait par appel de requête SQL
- Embedded SQL



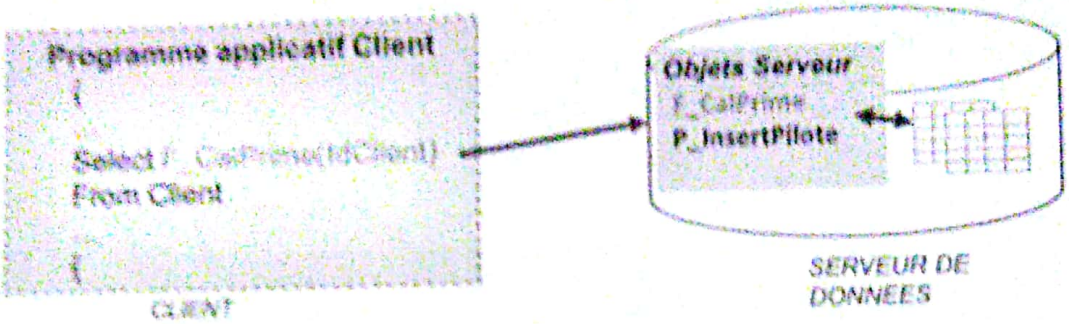
Architecture C/S et Objets Serveur

- Code applicatif sur le serveur de données
- Objectifs :
  1. Factoriser du code
  2. Renforcer la maintenance
  3. Augmenter la performance



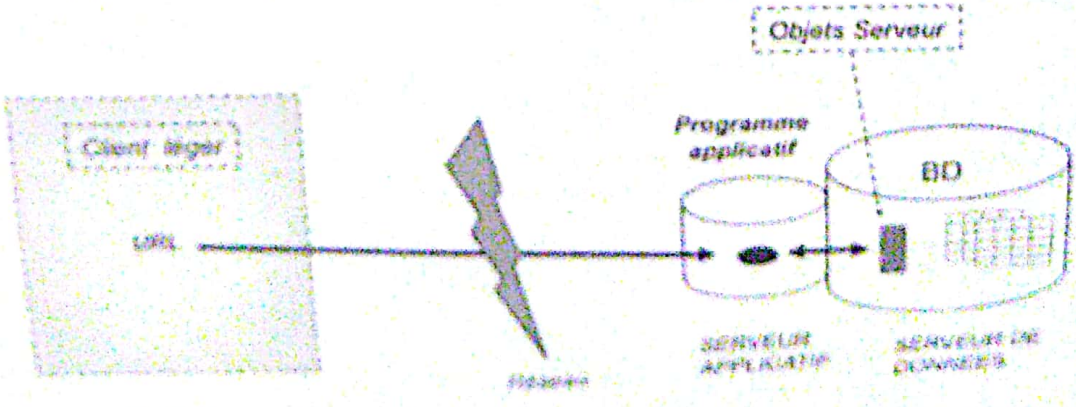
Objets Serveur

- Procédure stockées : exécute un programme
- Fonction stockées : exécute un programme et rend un résultat
  - Une fonction s'utilise dans du SQL
- Package : Librairie d'organisation des objets serveur
- Triggers : déclenche un programme suite à un évènement



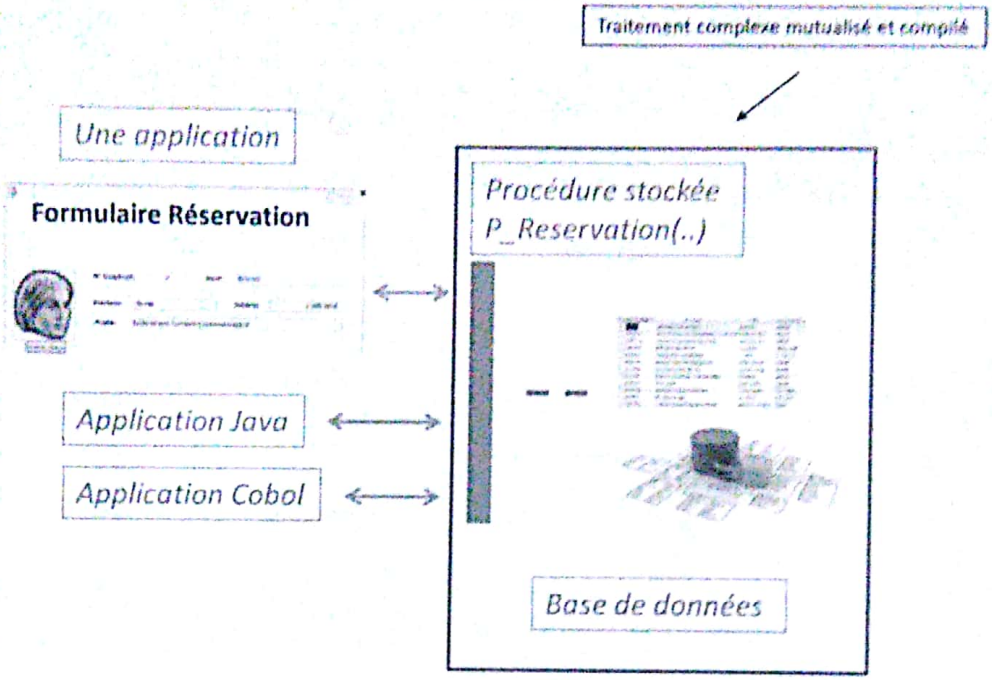
Architecture 3 tiers

- Serveur de Données / Serveur applicatif / Client léger

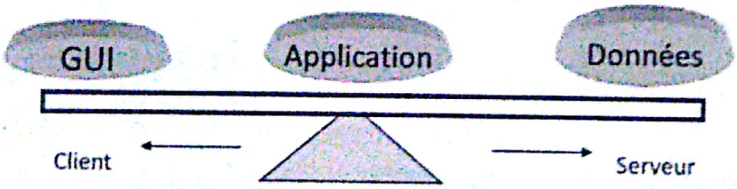


Intérêts des procédures stockées

Exemple d'une procédure de réservation



C/S à 2 niveaux



- **Modèle à 2 niveaux découpe en 2 la charge de l 'application**
  - Client LOURD
  - Toute la logique applicative s 'exécute sur le client qui par exemple envoie des requêtes SQL à une BD située sur le serveur. On appelle ce modèle architecture « orientée client », car le plus gros de l 'application tourne côté client

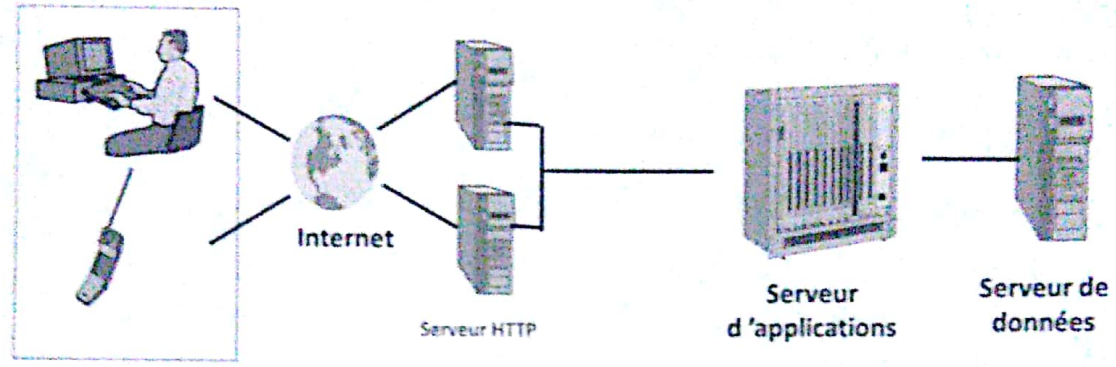
- **Modèle à 2 niveaux avec procédures stockées (2,5 niveaux):**
  - une partie de la charge est transférée du côté serveur.
  - Au lieu d 'invoquer des requêtes SQL à travers le réseau, des PS permettent d 'invoquer une fonction qui s 'exécute au sein de la base de données

C/S à 3 niveaux

- **Modèle à 3 niveaux répartit la charge**
  - Client LEGER
  - des clients qui exécute la logique de l 'interface graphique (GUI)

  Le serveur d 'application qui exécute la logique applicative

  La base de donnée architecture orientés « serveur »



Clients LEGER

C/S à 2 ou 3 niveaux ?

- C/S 2 niveaux :
  - simplicité de mise en œuvre
  - adaptée aux applications départementales avec 1 ou 2 serveurs homogènes et moins de 100 clients
  - SQL et procédures stockées

⇨ Le C/S à 3 niveaux (ou N niveaux) est bien adapté au vaste applications de l' internet et des intranets

Bases de données C/S : Langage de développement

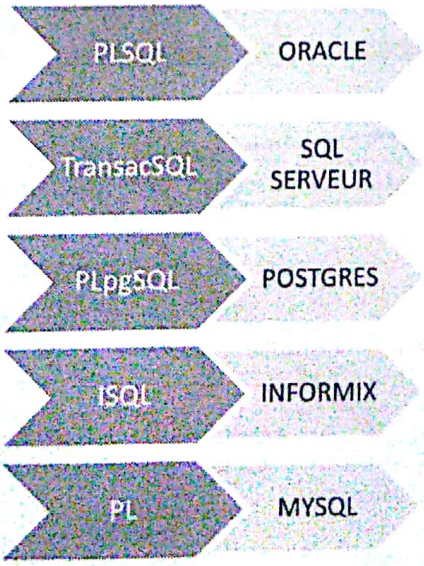
SQL est un langage BD limité

- ☹ Pas de fonctionnalités procédurales
- ☹ Pas de gestion de contexte
- ☹ Individualité des requêtes
- ☹ Ne répond pas au besoin du transactionnel

⇩

Besoin d'étendre le langage

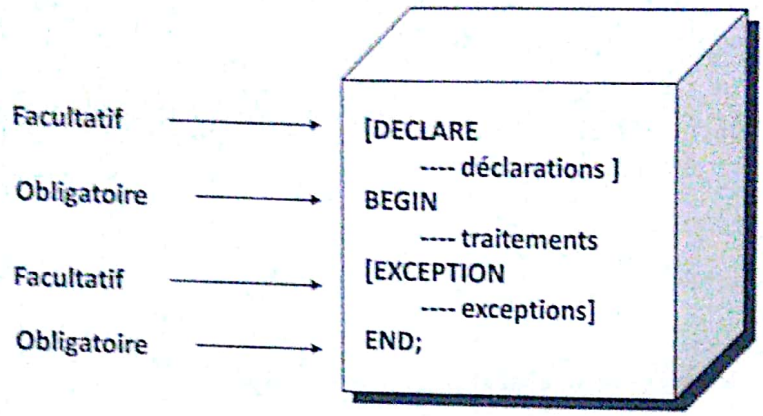
Programmation C/S : Les Langages des SGBDS relationnels



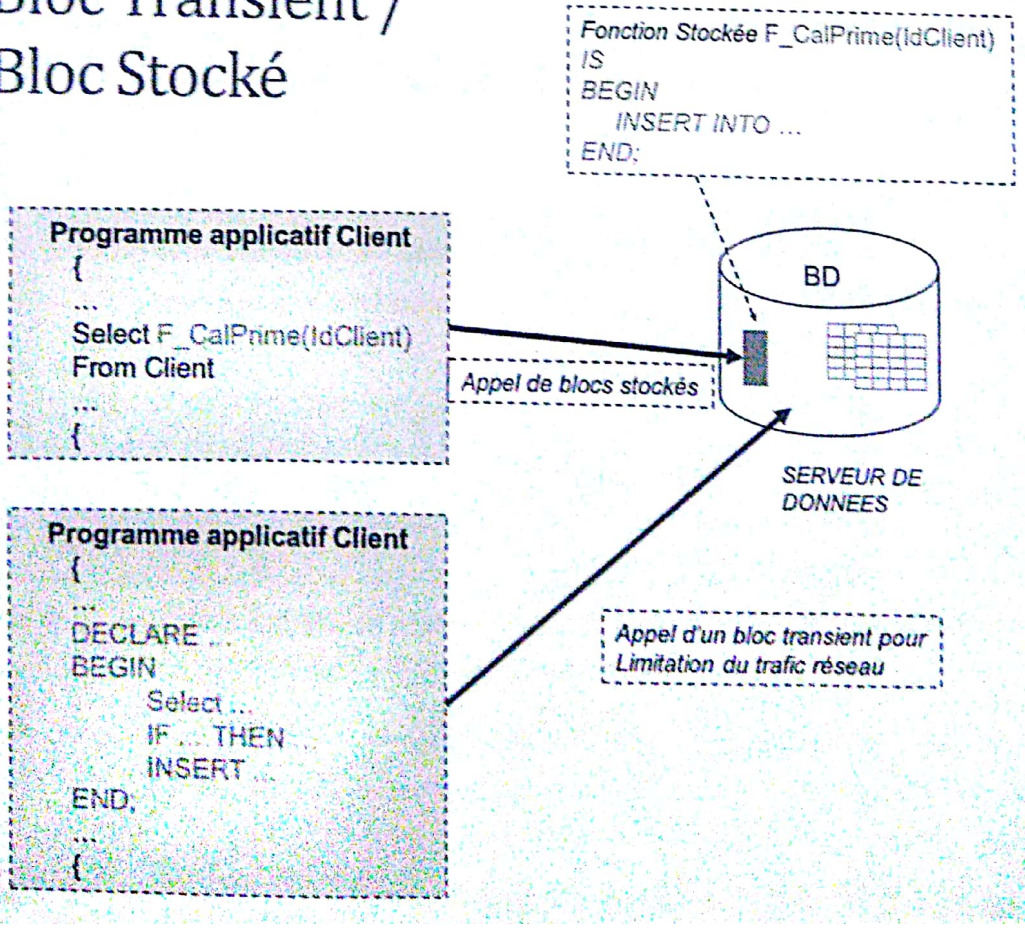
NORME SQL 2003

Structure d'un programme PL/SQL

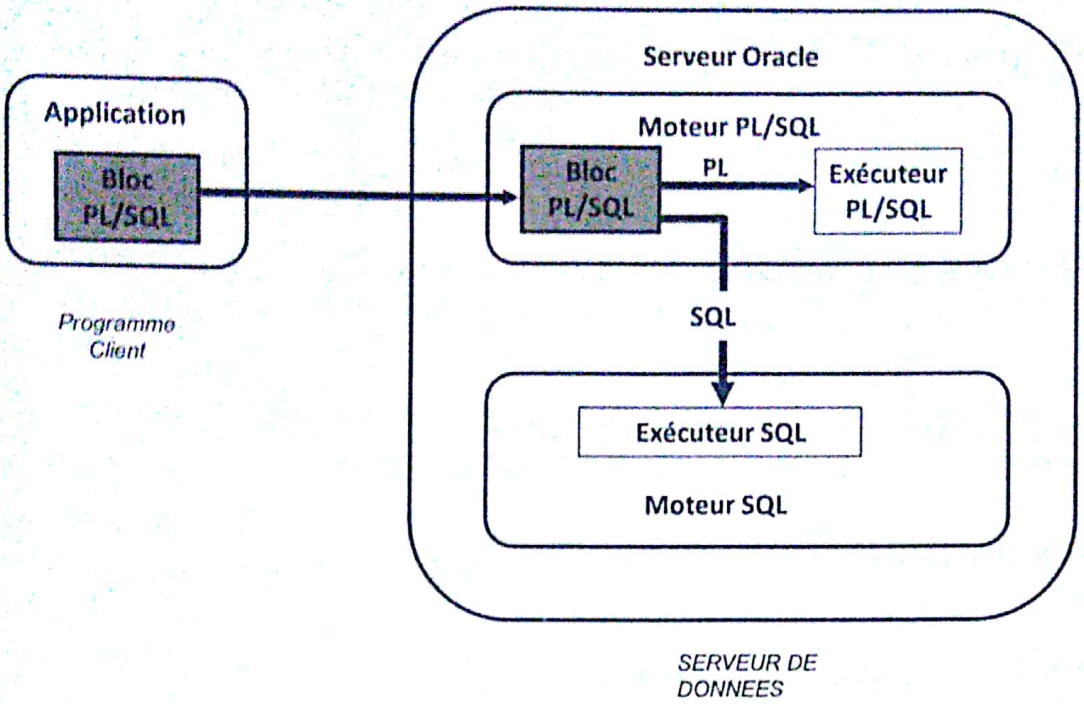
PL/SQL est une extension de SQL permettant d'avoir toutes les structures de programmation procédurale (Boucles, branchement, variables, ...) et tout en permettant l'utilisation d'instructions SQL



Bloc Transient / Bloc Stocké



PL/SQL et le serveur Oracle



Exemple de Bloc PL/SQL non stocké

```
DECLARE
   Vsal          NUMBER;
   Vsal_max      NUMBER NOT NULL := 400000;
   Vmat          INTEGER;
BEGIN
   SELECT Pilote.Salaire, matricule
   INTO Vsal, Vmat
   FROM Pilote WHERE Matricule = 10;
   IF Vsal + 10000 <  Vsal_max THEN
         UPDATE Pilote SET Salaire = Salaire + 10000 WHERE matricule = Vmat;
   ELSE
         NULL;
   END IF;
   COMMIT;
END;
```

Exemple de bloc Stocké : procédure d'insertion de pilote

Paramètres de la procédure

```
CREATE OR REPLACE PROCEDURE insert_pilote (p_nom VARCHAR,
                                           p_ville VARCHAR,
                                           p_age VARCHAR,
                                           p_sal NUMBER)
IS
BEGIN
INSERT INTO pilote VALUES (seq_pilote.nextval, p_nom, p_ville, p_age, p_sal);
END insert_pilote;
/

/* Appel de la procédure stockée sous SQL*Plus */
EXECUTE insert_pilote ('Richard Robian', 'Toulouse', 23, 230000);

/* Appel de la procédure stockée dans un bloc PL/SQL */

BEGIN
insert_pilote ('Pierre Corbin', 'Paris', 45, 300000);
END;
/
```

Exemple de bloc stocké: fonction de calcul de moyenne des salaires

```
CREATE OR REPLACE FUNCTION moyenne_pilote RETURN NUMBER
 IS
        Vmoy NUMBER (11,2);
BEGIN
        SELECT AVG(Salaire) INTO Vmoy FROM Pilote;
        RETURN Vmoy;
END;
/
```

Déclaration d'une variable local

Arrêt de la fonction. Retour du résultat

Le résultat de l'extraction est ranger dans la variable.
Valide ssi la requête n'extrait qu'une donnée.
Sinon utiliser un CURSEUR

```
SELECT moyenne_pilote FROM DUAL;
```

Appel de la fonction dans SQL

```
SET SERVEROUT ON
BEGIN
        DBMS_OUTPUT.Put_line ( 'Le salaire moyen est : ' || moyenne_pilote);
END;
/
```

Affiche un résultat à l'écran

Appel de la fonction dans PLSQL

```
SELECT TO_CHAR(salaire/moyenne_pilote,'0D99') FROM pilote;
```

Sous-programmes et traitements stockés

- Un sous-programme est un bloc PL/SQL nommé, paramétrable, et invocable.
- Deux types de sous-programmes sont disponibles : les procédures et les fonctions
- Il est possible de stocker les sous programmes dans la base de données afin de les réutiliser par différentes applications
- Il est possible de regrouper un ensemble de procédures, fonctions, variables et types et de les stocker ensemble dans la base
  - L'objet stocké ainsi s'appelle **PACKAGE** (paquetage)
  - La surcharge est autorisée dans un package

Sous-programmes stockés

Syntaxe SQL de Création d'un sous-programme stocké :

CREATE OR REPLACE PROCEDURE/FUNCTION *nom* [(*paramètres*)] IS
Définition du corps de sous-programme

- Exemple :

```
CREATE OR REPLACE PROCEDURE annul_vol (id_vol CHAR) IS
BEGIN
   DELETE FROM vol WHERE numvol = id_vol;
END;
```

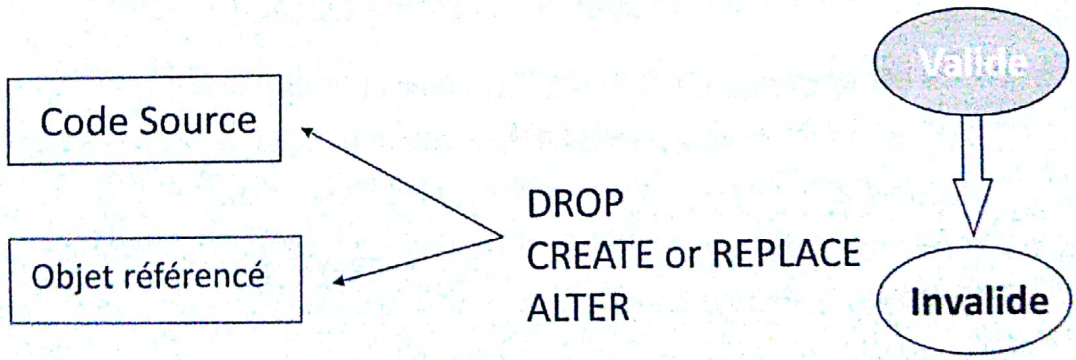
- Appel du sous programme :
  - dans PL/SQL : `annul_vol('AD30');`
  - dans SQL*PLUS :

```
EXECUTE annul_vol ('AD30');
CALL annul_vol ('AD30');
BEGIN annul_vol ('AD30'); END;
```

Et avec MYSQL ?

```
mysql> delimiter |

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END
-> |
Query OK, 0 rows affected (0.00 sec)

mysql> CALL simpleproc(@a)|
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a|
```


Etat des sous-programmes stockés



Syntaxes SQL :

- **Suppression :**
  DROP Procedure|function <nom sous programme>
- **Création et/ou modification :**
  CREATE OR REPLACE Procedure|function <nom sous programme> IS ..
- **Compilation :**
  ALTER Procedure|Function <nom sous programme> COMPILE

Exemple complémentaire : procédure de suppression d'une table

```
CREATE OR REPLACE PROCEDURE P_droptable (table_name IN VARCHAR2) IS

TSQL VARCHAR2(50);

BEGIN
   /* Construction de la requête à exécuter */
   TSQL := 'DROP TABLE ' || table_name || ' CASCADE CONSTRAINT PURGE';

   /* Execution dynamique de la commande DDL  */
   EXECUTE IMMEDIATE  TSQL;

END drop_table;
/
```

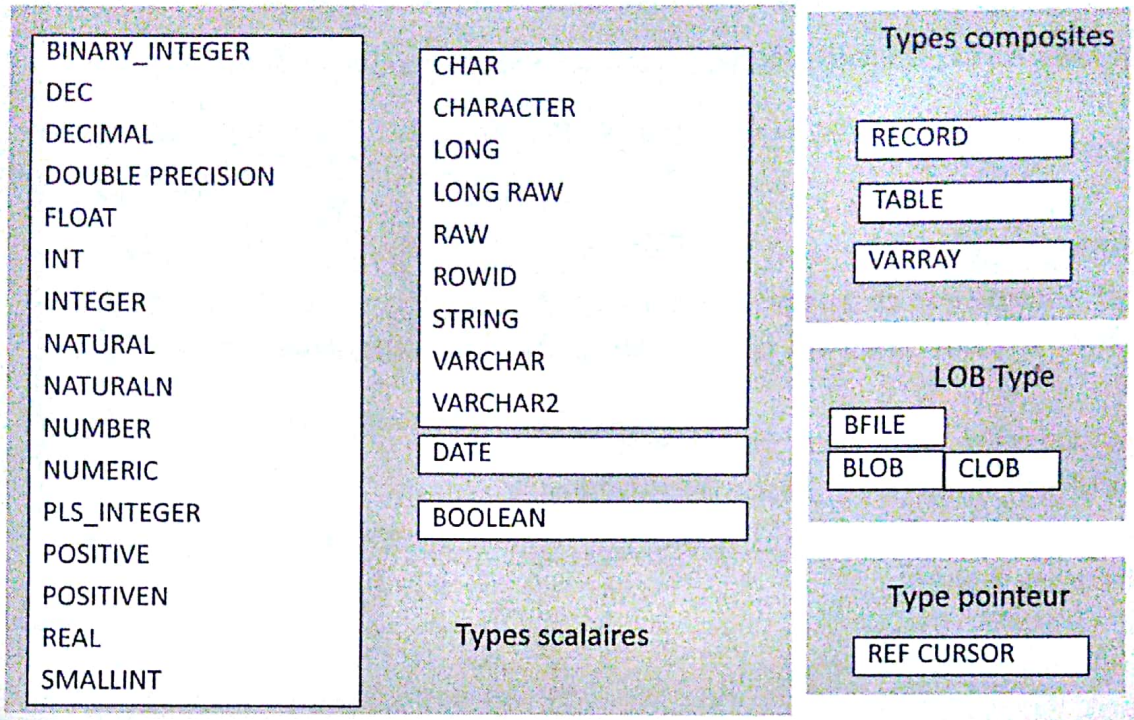
On construit dynamiquement la requête SQL de suppression de la table

Exécution dynamique de la requête construite durant le programme

Déclarations de variables en PLSQL

```
DECLARE (ou IS)
        /* Les variables doivent avoir des identifiants distincts */
  Date_naissance      DATE;
  Nombre_vols         SMALLINT := 0;
  Nom_pilote          CHAR(10)  NOT NULL := 'St-Exupéry';
  pi                  CONSTANT REAL := 3.14159;
  rayon                        REAL := 1;
  aire                REAL := pi * rayon **2;

BEGIN
        -- corps du programme
END;
```

TYPES AUTORISES EN PL/SQL



Les attributs de type

L'attribut %TYPE

Affecte à une variable le type d'une colonne d'une table ou d'une autre variable

exemple : nom_pilote pilote.nom%TYPE;

équivalent au type de la colonne **nom** de la table **pilote**

L'attribut %ROWTYPE

Affecte à une variable le type des tuples d'une table

exemple : pilote_rec pilote%ROWTYPE;

équivalent aux types des colonnes de la table **pilote**

Ambiguïté de nommage

- Les noms des colonnes l'emportent toujours sur les noms des variables
- Les noms des variables l'emportent toujours sur les noms des relations

```
DECLARE
        nom CHAR(15) := 'Serge Favret';
BEGIN
        DELETE FROM pilote WHERE nom=nom;
END;
```

Supprime tous les pilotes

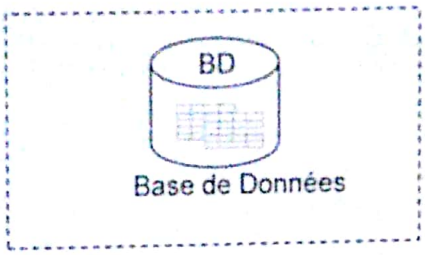
Quiz Objet Serveur

- A quoi sert un langage serveur ?
- Tous les SGBD ont-ils le même langage serveur ?
- Quelle est la différence entre une procédure et une fonction?
- Est-on obligé d'utiliser des objets serveurs dans une production ?

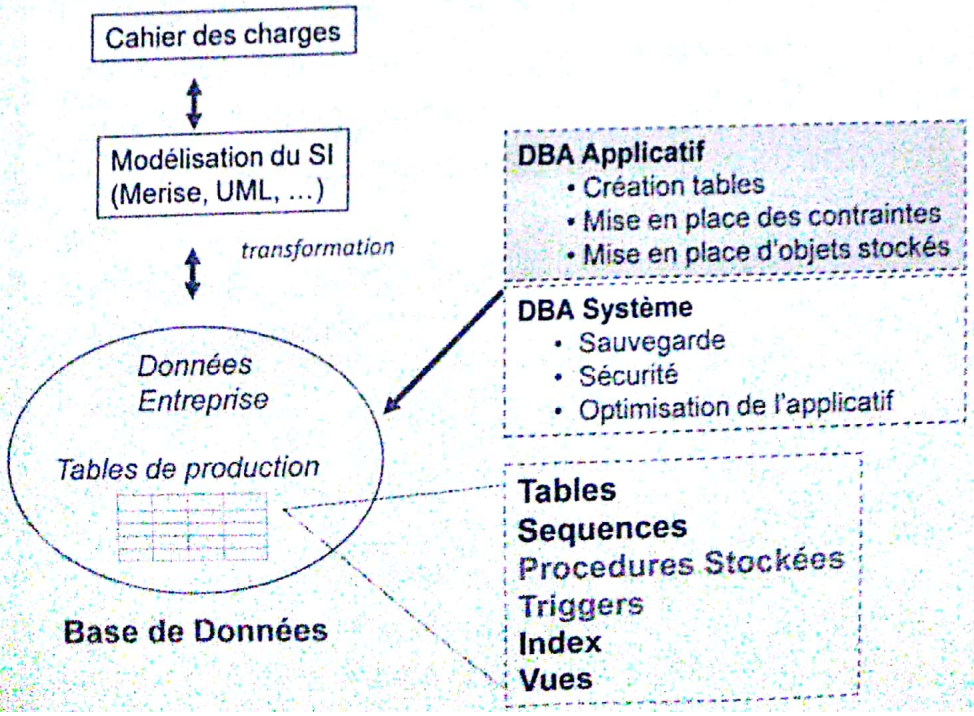
Module SGBD 2

Programmation serveur

Cours 2

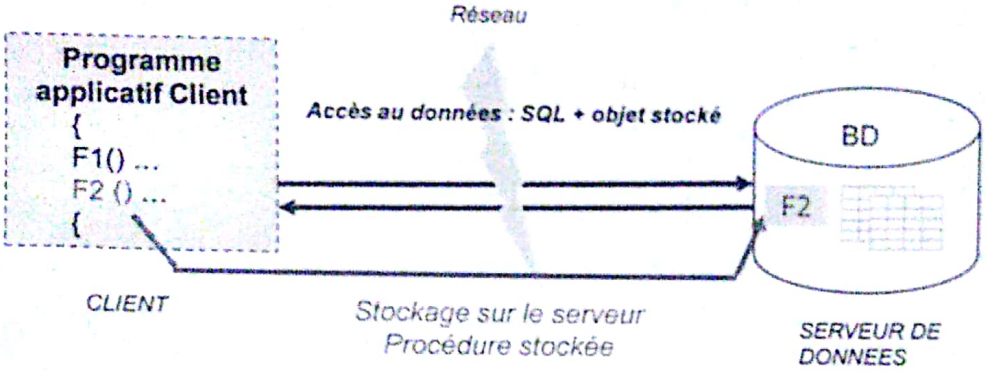


Base de données de production



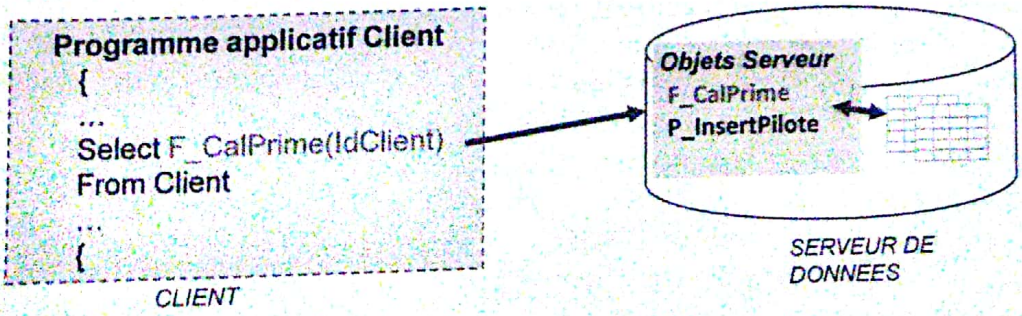
Architecture C/S et Objets Serveur

- Code applicatif sur le serveur de données
- Objectifs :
  1. Factoriser du code
  2. Renforcer la maintenance
  3. Augmenter la performance



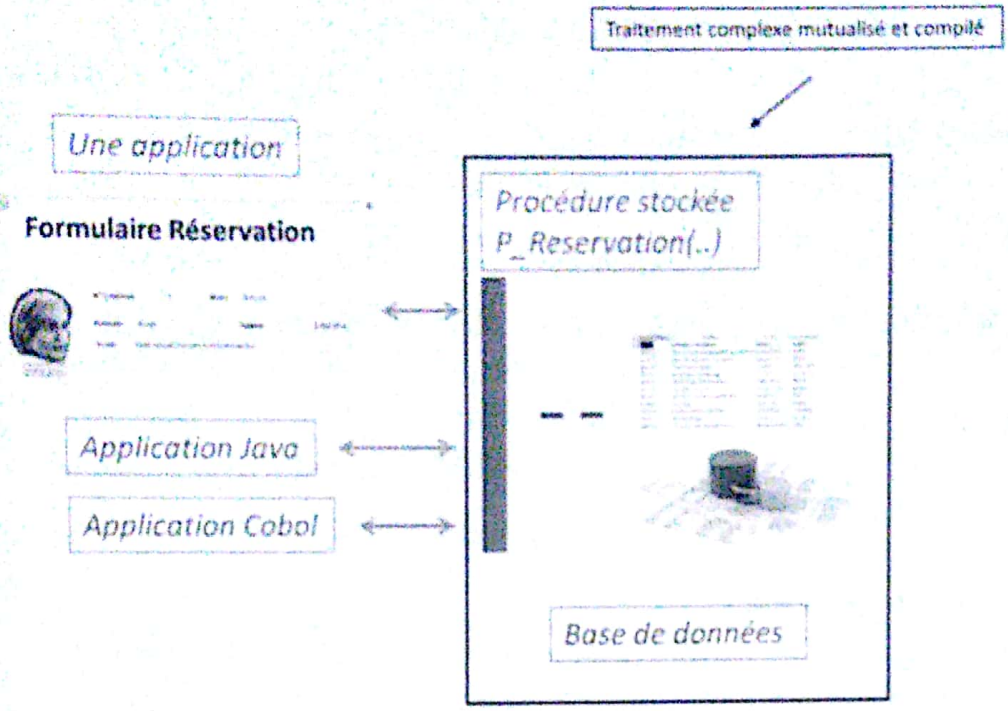
Objets Serveur

- Procédure stockées : exécute un programme
- Fonction stockées : exécute un programme et rend un résultat
  - Une fonction s'utilise dans du SQL
- Package : Librairie d'organisation des objets serveur
- Triggers : déclenche un programme suite à un évènement



Intérêts des procédures stockées

Exemple d'une procédure de réservation



Sous-programmes stockés

Syntaxe SQL de Création d'un sous-programme stocké :

CREATE OR REPLACE PROCEDURE/FUNCTION *nom* [(*paramètres*)] IS
Définition du *corps de sous-programme*

- Exemple :

```
CREATE OR REPLACE PROCEDURE annul_vol (id_vol CHAR) IS
BEGIN
   DELETE FROM vol WHERE numvol = id_vol;
END;
```

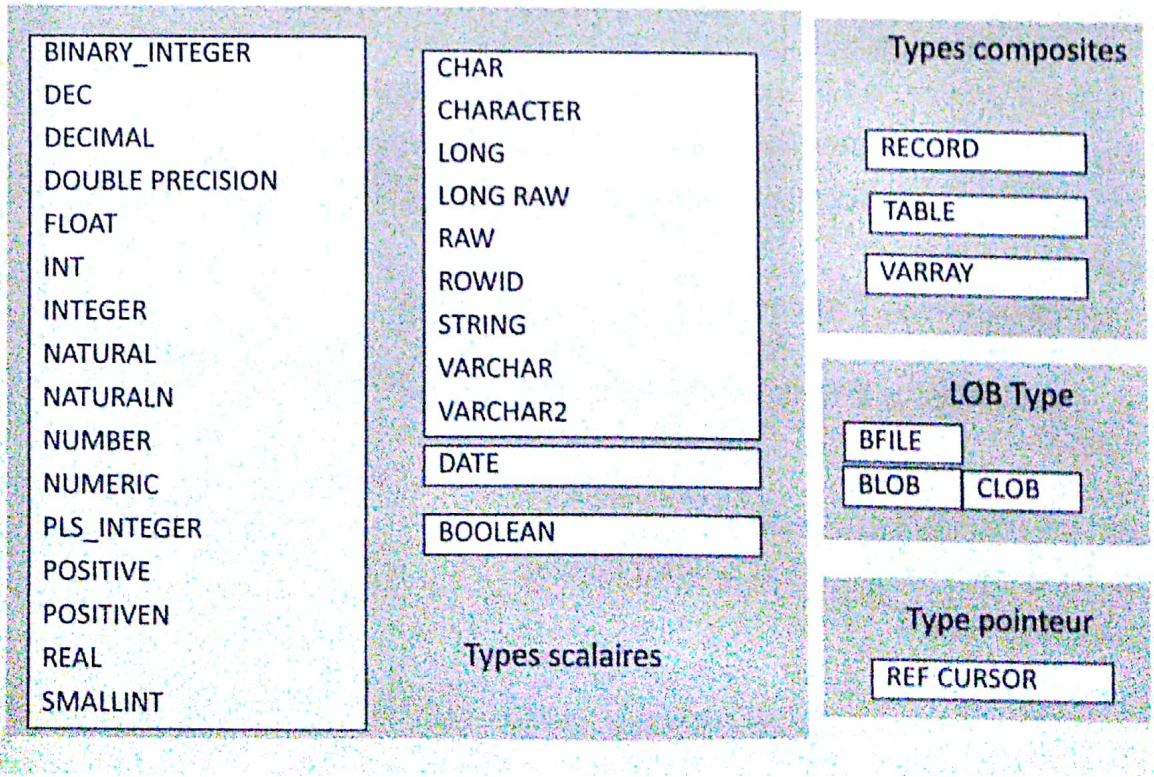
- Appel du sous programme :
  - dans PL/SQL : `annul_vol('AD30');`
  - dans SQL*PLUS :

```
EXECUTE annul_vol ('AD30');
CALL annul_vol ('AD30');
BEGIN annul_vol ('AD30'); END;
```

Déclarations de variables en PLSQL

```
DECLARE (ou IS)
        /* Les variables doivent avoir des identifiants distincts */
   Date_naissance      DATE;
   Nombre_vols         SMALLINT := 0;
   Nom_pilote          CHAR(10)  NOT NULL := 'St-Exupéry';
   pi                  CONSTANT REAL := 3.14159;
   rayon                        REAL := 1;
   aire                REAL := pi * rayon **2;

BEGIN
        -- corps du programme
END;
```

TYPES AUTORISES EN PL/SQL



Les attributs de type

L'attribut %TYPE

Affecte à une variable le type d'une colonne d'une table ou d'une autre variable

exemple : nom_pilote pilote.nom%TYPE;

équivalent au type de la colonne **nom** de la table **pilote**

L'attribut %ROWTYPE

Affecte à une variable le type des tuples d'une table

exemple : pilote_rec pilote%ROWTYPE;

équivalent aux types des colonnes de la table **pilote**

Exemple avec procédure d 'insertion de pilote

Paramètres de la procédure

```
CREATE OR REPLACE PROCEDURE insert_pilote (p_nom VARCHAR,
                                           p_ville VARCHAR,
                                           p_age VARCHAR,
                                           p_sal NUMBER)

IS
BEGIN
INSERT INTO pilote VALUES (seq_pilote.nextval, p_nom, p_ville, p_age, p_sal);
END insert_pilote;
/
```

Type de la colonne NOM de la table PILOTE

```
CREATE OR REPLACE PROCEDURE insert_pilote (p_nom PILOTE.NOM%TYPE,
                                           p_ville PILOTE.VILLE%TYPE,
                                           p_age PILOTE.AGE%TYPE,
                                           p_sal PILOTE.SALAIRE%TYPE)

IS
BEGIN
INSERT INTO pilote VALUES (seq_pilote.nextval, p_nom, p_ville, p_age, p_sal);
END insert_pilote;
/
```

Ambiguïté de nommage

- Les noms des colonnes l'emportent toujours sur les *noms des* variables
- Les noms des variables l'emportent toujours sur les *noms des* relations

```
DECLARE
        nom  CHAR(15) := 'Serge Favret';
BEGIN
        DELETE FROM pilote WHERE nom=nom;
END;
```

Supprime tous les pilotes

LES CURSEURS PLSQL :

Comment traiter une requête SELECT qui extrait plus d'un enregistrement ?

```
CREATE OR REPLACE FUNCTION moyenne_pilote RETURN NUMBER
 IS
Vmoy NUMBER (11,2);
Vres NUMBER(11,2);
CURSOR C_Pilote IS SELECT SALAIRE FROM PILOTE;
Vtuple C_Pilote%ROWTYPE;
BEGIN
        OPEN C_Pilote;
        LOOP
                FETCH C_Pilote INTO Vtuple;
                EXIT WHEN C1%NOTFOUND;
                Vres:= Vres+ Vtuple.salaire;
        END LOOP;

        Vmoy:= Vres/ C_pilote%ROWCOUNT;
        RETURN Vmoy;
        CLOSE C_Pilote;
END;
/

SELECT moyenne_pilote FROM DUAL;
```

Affectation de la requête à exécuter au CURSEUR C_Pilote

Variable structure basé sur le SELECT de C_PILOTE

Exécution de la requête

Extraction de la ligne courante du curseur

Fin de boucle lorsque le FETCH est sur EOF

Nombre de FETCH réalisés

Fermeture du curseur

Même programme optimisé

```
CREATE OR REPLACE FUNCTION moyenne_pilote RETURN NUMBER
 IS
        Vmoy NUMBER (11,2);
BEGIN
        SELECT AVG(Salaire) INTO Vmoy FROM Pilote;
        RETURN Vmoy;
END;
/
```

La bonne programmation consiste à utiliser au maximum le SQL avant de programmer

```
SELECT moyenne_pilote FROM DUAL;
```

LES CURSEURS PLSQL :

Comment traiter une requête SELECT qui extrait plus d'un enregistrement ?

Procédure qui permet d'afficher les pilotes et leur exéprience. Un pilote est considéré comme expérimenté s'il totalise plus de 100 vols répertoriés.

ON PENSE SQL AVANT DE PROGRAMMER
Déclaration de la requête

```
CREATE OR REPLACE PROCEDURE P_AffichePiloteExperimente
        T_exp       CHAR(15);

        CURSOR C_pilote IS      SELECT p.matricule, p. nom, count(d.numvol)  AS somme
                                FROM pilote p, depart d
                                WHERE p.matricule=d.matricule
                                GROUP BY p.matricule, p.nom;
BEGIN
   FOR tuple_pilote IN C_pilote LOOP
        IF (tuple_pilote.somme>100)
        THEN    T_exp:='EXPERIMENTE';
        ELSE    T_exp:='INEXPERIMENTE';
        END IF;
   DBMS_OUTPUT.PUT_LINE ('Le pilote ' || tuple_pilote.matricule || ' est '|| ' '|| T_exp);
   END LOOP;
END;
/
```

Nommage du curseur associé à la requête

Exécution de la requête associée à C_Pilote :
Chaque ligne résultat est rangée dans la variable autodéclarée « TUPLE »
TUPLE est une variable structure
La boucle s'arrête à la fin de la zone de résultat

Affichage écran

LES CURSEURS PLSQL :

Comment traiter une requête SELECT qui extrait plus d'un enregistrement ?

Procédure qui permet d'afficher les pilotes et leur exéprience. Un pilote est considéré comme expérimenté s'il totalise plus de 100 vols répertoriés.

ON PENSE SQL AVANT DE PROGRAMMER
Déclaration de la requête

Nommage du curseur associé à la requête

```
CREATE OR REPLACE PROCEDURE P_AffichePiloteExperimente
        T_exp       CHAR(15);

        CURSOR C_pilote IS      SELECT p.matricule, p. nom, count(d.numvol)  AS somme
                                FROM pilote p, depart d
                                WHERE p.matricule=d.matricule
                                GROUP BY p.matricule, p.nom;
BEGIN
   FOR tuple_pilote IN C_pilote LOOP
        IF (tuple_pilote.somme>100)
        THEN    T_exp:='EXPERIMENTE';
        ELSE    T_exp:='INEXPERIMENTE';
        END IF;
    DBMS_OUTPUT.PUT_LINE ('Le pilote ' || tuple_pilote.matricule || ' est '|| ' '|| T_exp);
  END LOOP;
END;
/
```

Exécution de la requête associée à C_Pilote :
Chaque ligne résultat est rangée dans la variable autodéclarée « TUPLE »
TUPLE est une variable structure
La boucle s'arrête à la fin de la zone de résultat

LES CURSEURS PLSQL :

Syntaxe simplifiée

On peut mettre la requête directement dans la syntaxe de la boucle FOR

```
CREATE OR REPLACE PROCEDURE P_AffichePiloteExperimente IS
        T_exp     CHAR(15);

FOR tuple_pilote IN (SELECT p.matricule, p. nom, count(d.numvol)  AS somme
                            FROM pilote p, depart d
                            WHERE p.matricule=d.matricule
                            GROUP BY p.matricule, p.nom )  LOOP
        IF (tuple_pilote.somme>100)
        THEN    T_exp:='EXPERIMENTE';
        ELSE    T_exp:='INEXPERIMENTE';
        END IF;
        DBMS_OUTPUT.PUT_LINE ('Le pilote ' || tuple_pilote.matricule ||
                                        ' est '|| ' '|| T_exp);
  END LOOP;
END;
/
```

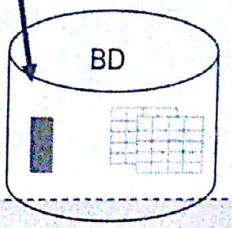
SQL DYNAMIQUE

Exemple : Traitement Admin

Objectif : Mettre en place une procédure qui permet de supprimer toutes les tables d'un compte

Tables d'un compte : SELECT tname FROM tab;

***Algo** : Pour chaque table du compte, générer la requête SQL de suppression correspondante.*
Exemple :
DROP TABLE PILOTE CASCADE CONSTRAINT PURGE



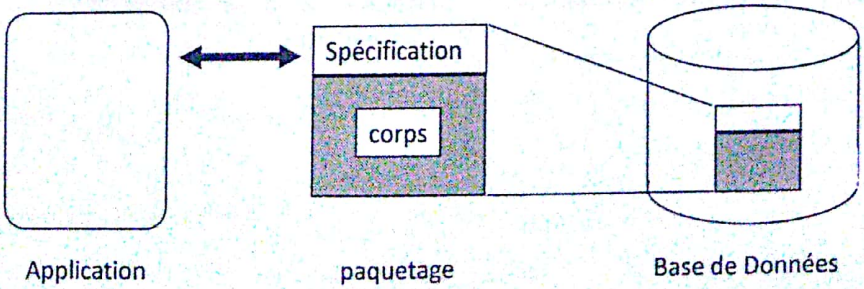
Programme PLSQL

```
CREATE OR REPLACE PROCEDURE EraseAllTables IS
Tsql VARCHAR(200);
BEGIN
FOR tuple IN ( SELECT tname FROM tab) LOOP

Tsql :=' DROP TABLE ' || tuple.tname || ' CASCADE CONSTRAINT PURGE';

EXECUTE IMMEDIATE Tsql;

DBMS_OUTPUT.PUT_LINE ( 'La table ' || tuple.tname || ' a été supprimée');
END LOOP;
END;
/
```

SQL DYNAMIQUE : exécution dynamique de la requête construite et contenue dans TSQL

Paquetages (packages)

- Objet PL/SQL qui regroupe logiquement des Types, curseurs, sous-programmes , Variable PL/SQL.
- Un paquetage se compose de deux parties :
  - Spécification
  - Corps
- Définition d'interfaces d'application



Création d'un paquetage

```
CREATE OR REPLACE PACKAGE nom
[ AUTHID { DEFINER | CURRENT_USER } ] { IS | AS }
        types publiques
        déclaration d'objets publiques
        spécifications des sous-programmes accessibles
END [ nom ];
```

```
CREATE OR REPLACE PACKAGE BODY nom { IS | AS }
        types privés et déclarations d'objets privés
        corps des sous-programmes
[ BEGIN
        Initialisation ]
END [ nom ];
```

Exemple (1)

```
CREATE OR REPLACE PACKAGE vol_actions IS

   PROCEDURE P_ADDVOL
        (           volid      VARCHAR2,
                    heuredep   DATE,
                    heurearr   DATE,
                    villedep   VARCHAR2,
                    villearr   VARCHAR2
        );
   FUNCTION F_NBREVOL RETURN number ;

END vol_actions;
```

Partie interface du package

Procedure P_ADDVOL

Fonction F_NBREVOL

Exemple (2)

Partie corps du package

```
CREATE OR REPLACE PACKAGE BODY vol_actions AS
    PROCEDURE P_ADDVOL (volid VARCHAR2, heuredep DATE, heurearr DATE,
                        villedep VARCHAR2, villearr VARCHAR2)
    IS
    BEGIN
        INSERT INTO vol VALUES (volid, heuredep, heurearr, villedep, villearr);
    END P_ADDVOL;
    FUNCTION NBREVOL RETURN number
    IS
        nbre INTEGER;
    BEGIN
        SELECT count( *) INTO nbre FROM vol;
        RETURN nbre;
    END Nombre_vol;
END vol_actions;
```

Corps de la procedure P_ADDVOL

Corps de la fonction NBREVOL

Appeler les objets d'un paquetage

- Il est possible de référencer le contenu d'un paquetage à partir de l'application cliente (PL/SQL, SQL*PLUS, ...) :

```
DECLARE
        new_vol CHAR(20); ...
BEGIN
        IF NOT (new_vol IS NULL) THEN
                vol_actions.PADDVOL (...);
                ...
        END IF;
END;
```

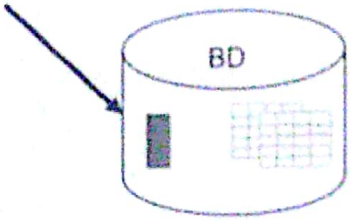
Référence à la procédure "add_vol" du package "Vol_actions dans un bloc PL/SQL

Intérêt d'un paquetage

Les packages permettent de :

- Structurer les objets stockés sur le serveur
- Optimiser leur appels

Au 1er appel du package, tout le package est monté en mémoire cache.
Les appels suivants seront plus rapides



Les principaux paquetages d'Oracle

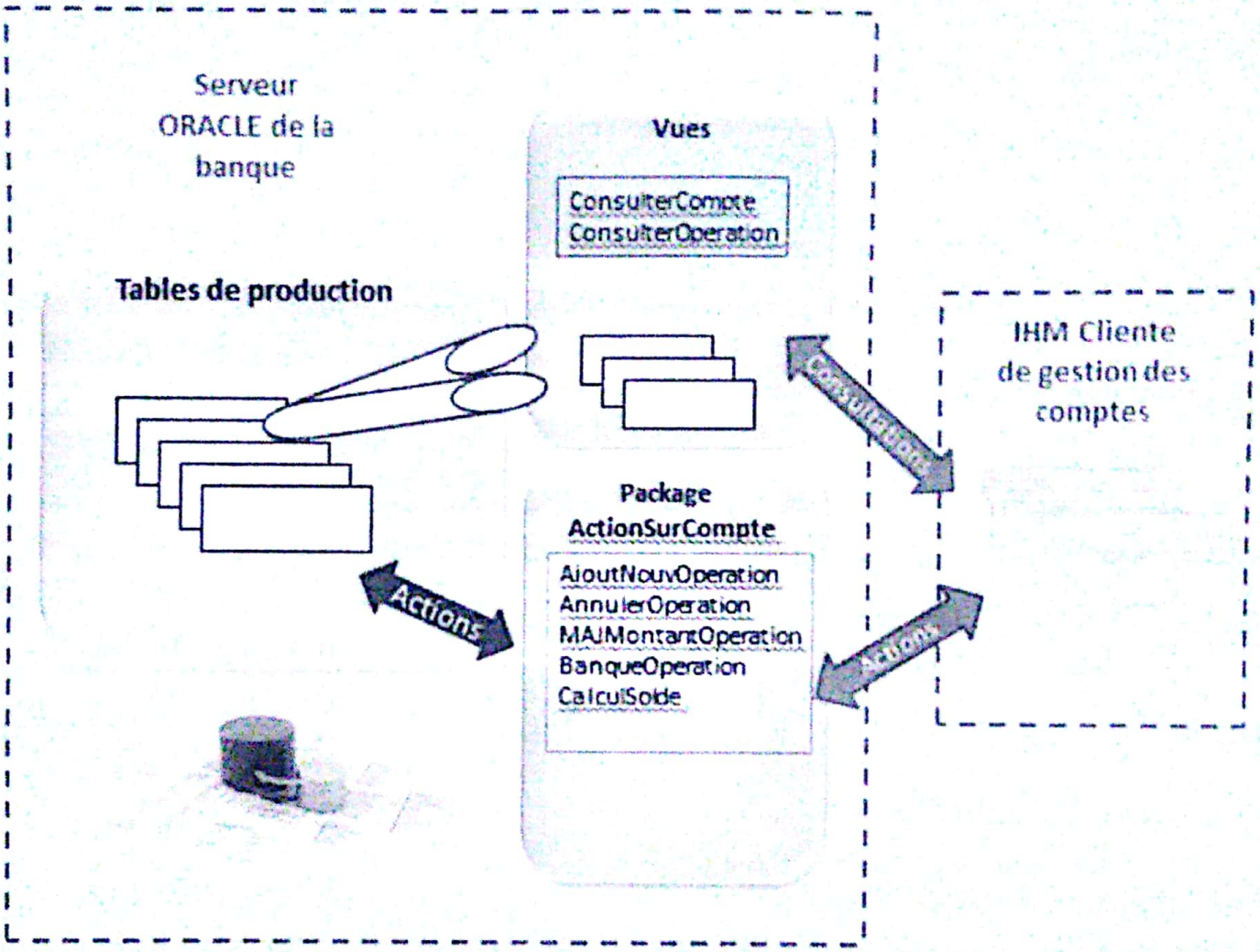
DBMS_STANDARD :	défini l'environnement PL/SQL. Il offre des fonctions standards, communes entre toutes les applications (fonctions de conversions, fonctions mathématiques, ...)
DBMS_OUTPUT :	permet de réaliser des entrée/sortie d'affichage dont «put_line».
DBMS_SQL :	permet d'exécuter des instructions SQL dynamiques.
DBMS_PIPE :	permet à plusieurs sessions de communiquer via des pipes.
UTL_FILE :	permet aux applications de lire et d 'écrire des fichiers texte sur le système.
DBMS_ALERT :	permet d'envoyer un message vers plusieurs sessions utilisateur.
DBMS_JOB :	permet de soumettre des travaux à une heure et une fréquence déterminées. Principe des « schedulers »
DBMS_AQ :	permet de construire des applications sophistiquées basées sur des files d'attente
UTL_HTML :	permet d'afficher des données en HTML
DBMS_LOB :	permet de lire et de modifier les objets BLOB, CLOB, BFILE

Quiz Objet Serveur

- A quoi sert un langage serveur ?
- Tous les SGBD ont-ils le même langage serveur ?
- Quelle est la différence entre une procédure et une fonction?
- A quoi sert un Package ?
- Est-on obligé d'utiliser des objets serveurs dans une production ?

Tout document autorisé
Durée: 2h00

Le contexte de cet examen est l'application bancaire travaillée ces deux dernières semaines. Le schéma ci-dessous résume son architecture :



On donne ci-dessous le schéma relationnel utile pour ce DST. On ne s'occupe donc pas ici des types de comptes. Les colonnes « SoldeCompte » et « Inactif » ont été ajoutées ainsi que la table « DECOUVERT ».

BANQUE *(IdBanque, LibelleBanque, VilleBanque)*
COMPTE*(IdCompte,LibelleCompte , IdBanque#, SoldeCompte, Inactif)*
OPERATION *(IdOp,DateOp,MontantOperation, #IdCompte)*

DECOUVERT*(IdCompte, LibelleCompte, SoldeCompte)*

Les clés primaires sont soulignées ; les clés étrangères précédées d'un #

On considère que les procédures et fonctions du package « ActionSurCompte » ainsi que les deux vues sont implantées et fonctionnent.

1. Ajout de la fonction VilleBanque

On souhaite ajouter au package la fonction « VILLEBANQUE » qui permet de retourner la ville de la banque dont le nom est passé en paramètre.

VILLEBANQUE (PNomBanque VARCHAR) RETURN VARCHAR

Donner le code PLSQL de cette fonction.

2. Mise à jour des soldes

On souhaite ajouter la procédure suivante au package ActionSurCompte :

- ✓ ***MAJSOLDE(PIdCompte INT, PMontant NUMBER)***
  Cette procédure permet, lorsqu'elle est appelée, de mettre à jour la colonne « SoldeCompte » du compte passé en paramètre en lui ajoutant le montant passé en paramètre.

  a) Donner le code PLSQL de cette procédure.
  b) Compléter le code la procédure « AjoutNouvOperation » pour que l'ajout d'une nouvelle opération mette à jour le solde du compte.

3. Affichage d'un classement

Le code de la procédure « P_CLASSEMENT » suivant est incomplet, des lignes de code ont été supprimées :

```
CREATE OR REPLACE PROCEDURE P_CLASSEMENT
IS
CURSOR C_CLAS IS
SELECT C.IdCompte, C.LibelleCompte, COUNT(*) AS NB
FROM Compte C, Operation O
WHERE C.IdCompte=O.IdCompte
GROUP BY C.IdCompte, C.LibelleCompte
ORDER BY NB DESC;

BEGIN

DBMS_OUTPUT.PUT_LINE('**********************************');
DBMS_OUTPUT.PUT_LINE('CLASSEMENT DES COMTES PAR NOMBRE
D''OPERATIONS');
DBMS_OUTPUT.PUT_LINE('**********************************');
... A COMPLETER

DBMS_OUTPUT.PUT_LINE('COMPTE ' || ligne.Idcompte || ' : ' || ligne.NB || '
OPERATIONS');
.... A COMPLETER

END;
/
```

On vous demande de compléter le code fourni pour que l'exécution de la procédure permette l'affichage du classement décroissant des comptes par nombre d'opérations.

Par exemple, l'exécution suivante de « P_CLASSEMENT » a été réalisée sur une base contenant 4 comptes ayant chacun plusieurs opérations :

```
**********************************************
CLASSEMENT DES COMTES PAR NOMBRE D'OPERATIONS
**********************************************
COMPTE 3 : 13 OPERATIONS
COMPTE 1 : 10 OPERATIONS
COMPTE 2 : 3 OPERATIONS
COMPTE 4 : 3 OPERATIONS
```

4. Comptes à découverts

La table « DECOUVERT » a été ajoutée à la base en production. Elle est destinée à contenir les comptes qui sont à découvert, c'est-à-dire qui ont un solde négatif. Chaque soir, après les opérations de la journée, cette table est vidée et de nouveau remplie par appel de la procédure stockée « TR_BATCH » sans paramètre.

On vous demande de donner le code PLSQL de cette procédure « TR_BATCH » appelée chaque soir.

5. Comptes inactifs

« TR_BATCH » permet également chaque soir de positionner la colonne « Inactif » de la table « COMPTE ». Cette colonne est par défaut à « **0** » et prend la valeur « **1** » si le compte est inactif. Un compte est considéré comme inactif s'il n'a pas eu d'opérations depuis au moins 1 an. La fonction SYSDATE permet d'obtenir la date du jour.

Donner le code à ajouter à la procédure « TR_BATCH » pour réaliser la mise à jour de la colonne « Inactif » chaque soir.

FIN