Architecture d'un SI d'entreprise

- Les Principes
- Architecture 1 tier (stand alone)
- Architecture 2 tier
- Architecture 3 tier
- Architecture n tier

Définition d'un serveur d'application

Ce serveur est un ensemble de logiciel qui sert à séparer
la logique métier de la logique présentation et de la logique base de données.

la base de données peut être séparée physiquement de la logique métier.

Les difficultés à surmonter pour implémenter un serveur d'application

- Accès simultanés des clients
- Surcharge des réseaux
- Autorisation d'accès aux bases de données
- Tolérance aux pannes, performances
- Equilibrage des charges
- Sécurité des données

Le serveur d'application doit donner une souplesse pour le développement et la mise en production des données.

Architecture d'un SI

Architecture 2-tiers

L'architecture à deux niveaux se compose de plusieurs clients (100 user) et d'un serveur.

Le client se connecte au serveur via des protocoles de réseaux (TCP/IP).

Le serveur contient la logique métier et stocke les données.
Le processus client collecte les informations au serveur (logique métier).
Le serveur implémente la logique métier et valide les données.

Client lourd : une partie de la logique métier est déplacé sur le poste client.
Client léger : le serveur gère toute la partie métier.

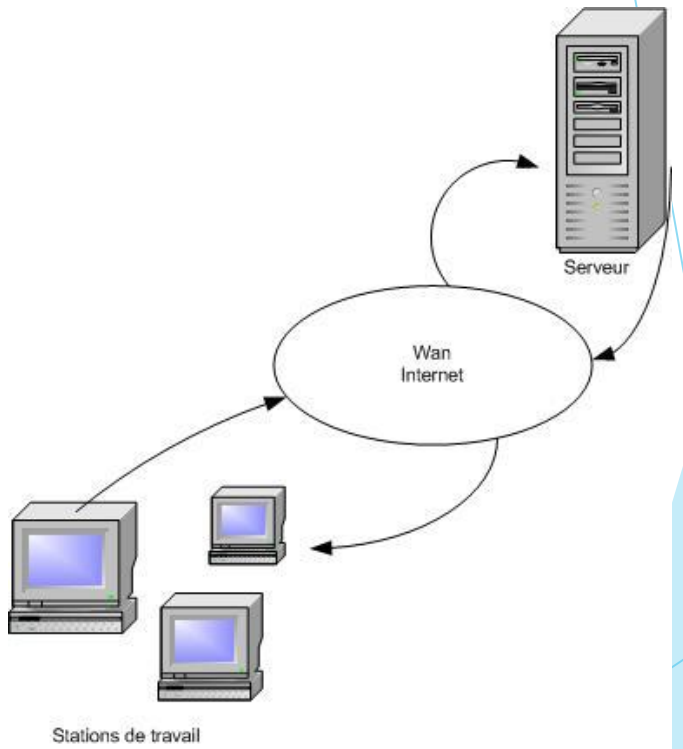
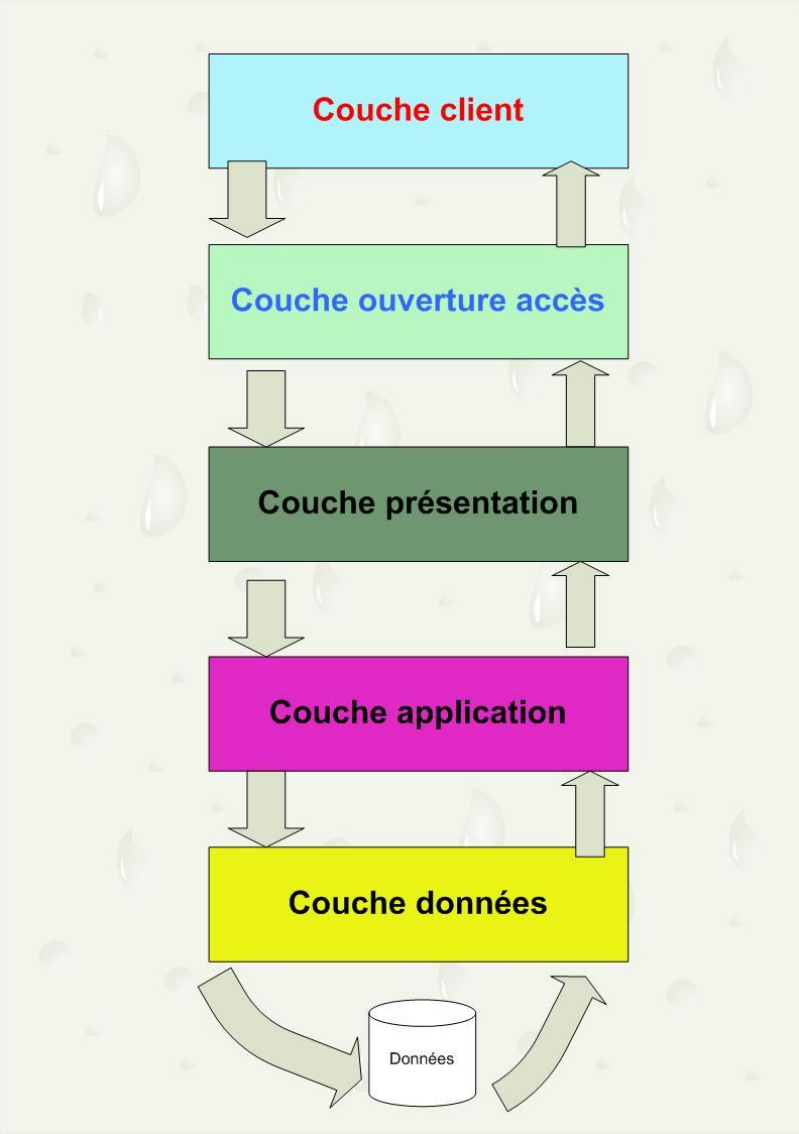
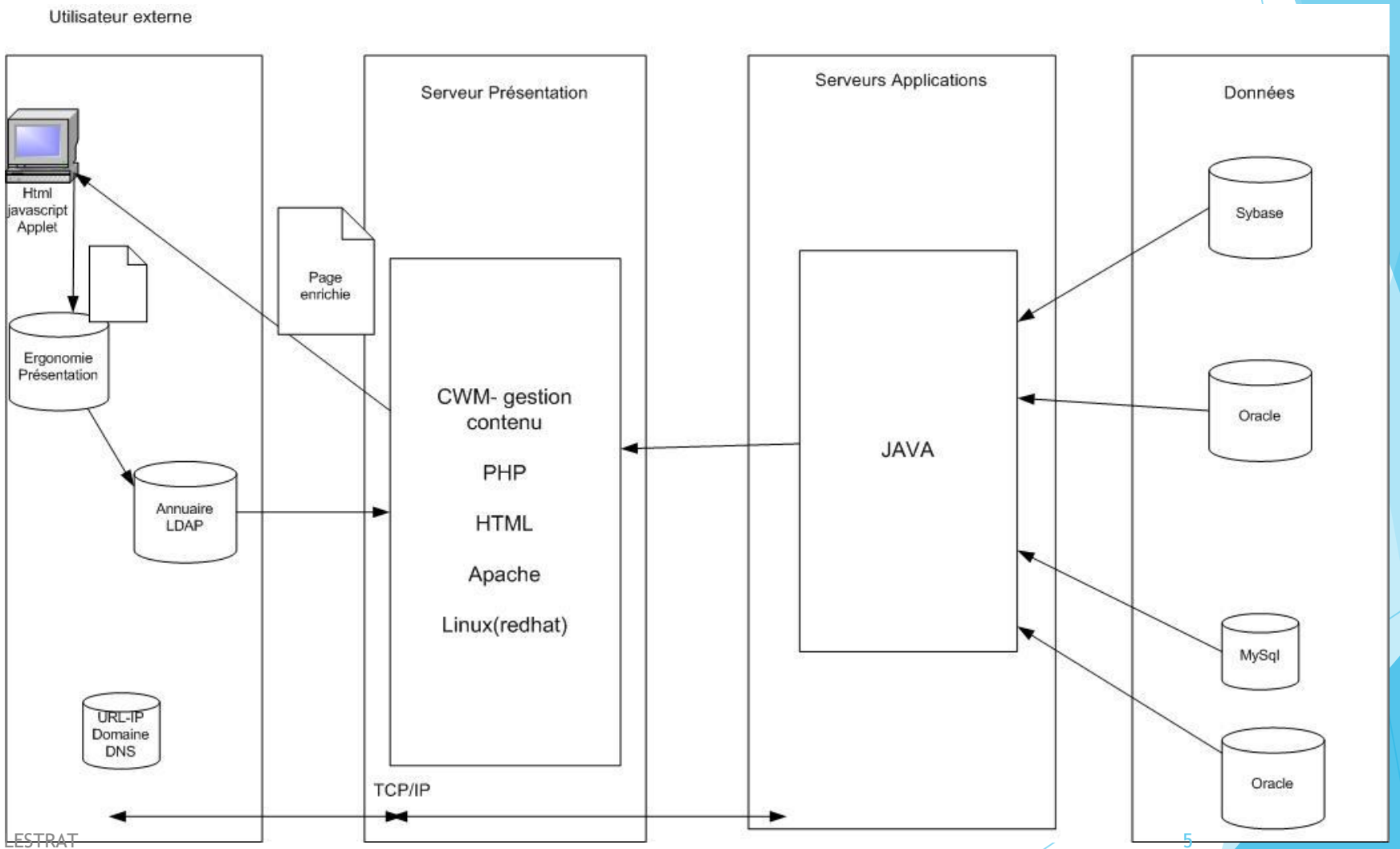


Figure3 : architecture 2-tiers

Architecture logique



Architecture physique



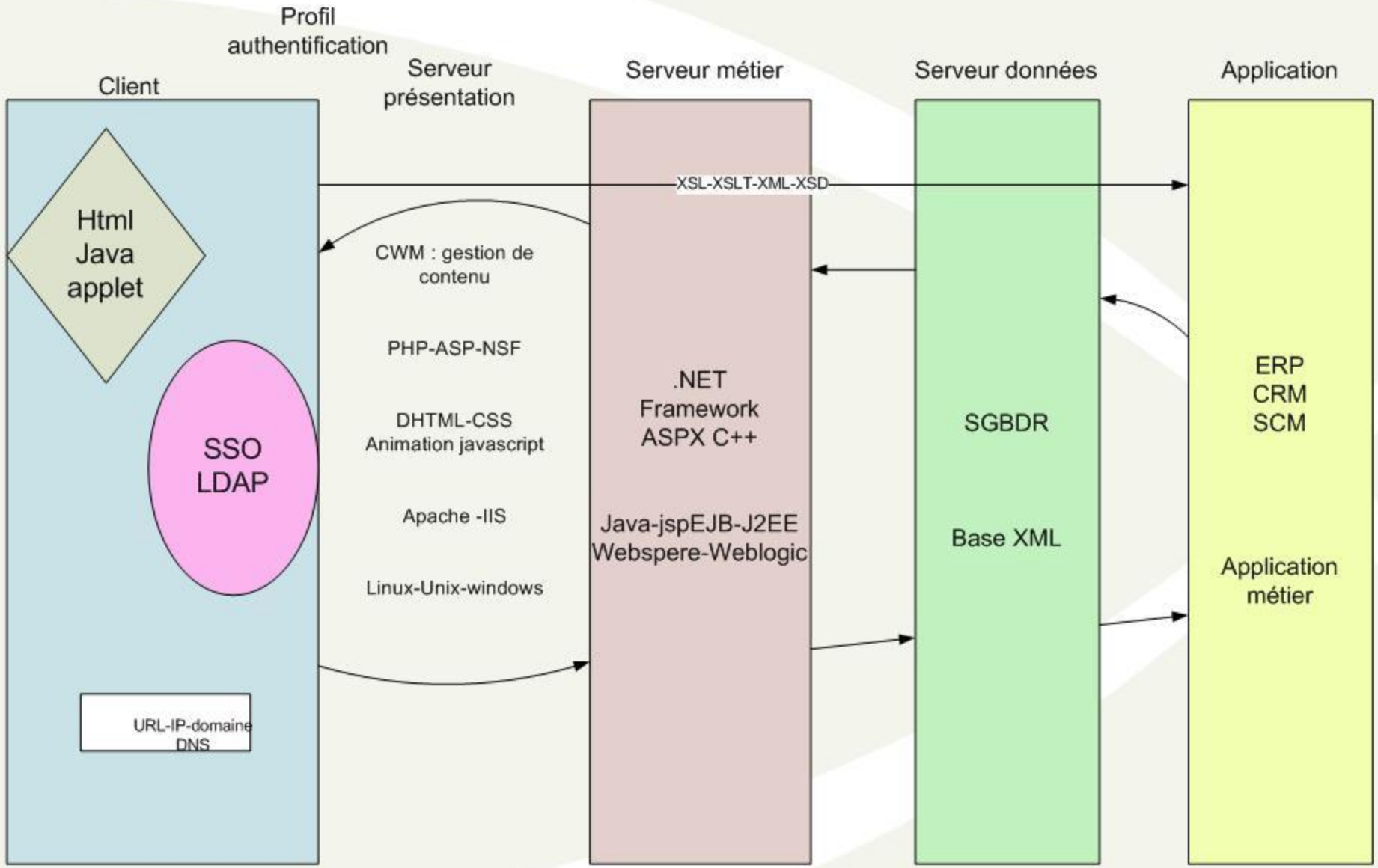


Figure 8 : Topologie d'un portail

SGBD ?

Objectifs

- Indépendance physique des données
- Indépendance logique
- Manipulation des données par des non informaticiens
- Efficacité d'accès
- Administration cohérente
- Pas de redondance
- Cohérence des données
- Partage des données
- Sécurité des données

Différence entre BDD et SGBD

- BDD : ensemble structuré de l'information
- SGBD : c'est la suite logicielle pour administrer l'ensemble de l'information

Catégorie d'instruction

DATA DEFINITION LANGUAGE (DDL)

- langage de manipulation des objets
  - Créer, manipuler, supprimer des objets
  - Autoriser ou interdire l'accès aux objets
  - Les instructions : Create, Alter, Drop, Grant, Revoke,Truncate
- interroger et mettre à jour les données sans préciser d'algorithme d'accès

- DATA MANIPULATION LANGUAGE (DML)

langage de manipulation des objets

- Ajout, suppression, modification de lignes
- Visualisation du contenu des tables
- Verrouillage des tables
- Les instructions : Insert, Update, Delete, Select, Lock Table.

Catégorie d'instruction

Transaction Control Language

- Gère les modifications des manipulations des données DML
  - Caractéristiques des transactions
  - Validation, annulation des modifications
  - Les instructions : Commit, Rollback, Savepoint

- Session Control Language

  Gérer la session des utilisateurs
  - Activation, désactivation des privilèges des utilisateurs
  - Les instructions : Create user, Alter user, Grant

Objectifs et avantages des SGBD

Que doit permettre un SGBD ?

- **Contrôler les données**
  - **Intégrité :** vérification de contraintes d'intégrité

ex.: le salaire doit être compris entre 4000 euros et 20000 euros

- **confidentialité** contrôle des droits d'accès, autorisation

⇒ langage de contrôle des données : DATA CONTROL LANGUAGE (DCL)

- **Partage**

une BD est partagée entre plusieurs utilisateurs en même temps

⇒ contrôle des accès concurrents notion de transaction

L'exécution d'une transaction doit préserver la cohérence de la BD

- **Sécurité**
  - reprise après panne, journalisation
  - Performances d'accès
- index (hashage, arbres balancés ...)

Objectifs et avantages des SGBD

Que doit permettre un SGBD ?

- **Indépendance physique**
  - Pouvoir modifier les structures de stockage ou les index sans que cela ait de répercussion au niveau des applications
  - Les disques, les méthodes d'accès, les modes de placement, le codage des données ne sont pas apparents

- **Indépendance logique**
  - Permettre aux différentes applications d'avoir des vues différentes des mêmes données
  - Permettre au DBA de modifier le schéma logique sans que cela ait de répercussion au niveau des applications

Modèle relationnel

A remplacer le modèle hiérarchique

- Le modèle de données relationnel est fondé sur la notion de relation : un tableau à deux dimensions qui contient un ensemble de n-uplets (les lignes).

Quand on se focalise plus sur le stockage, les relations sont souvent appelées des tables et **les n-uplets** des enregistrements.

- Les entrées dans les tables sont appelées des valeurs.

- Les relations représentent les entités du monde réel (comme des personnes, des objets, etc.) ou les associations entre ces entités
- Notions de clés primaires et étrangères

Modèle relationnel

- SIMPLICITE DE PRÉSENTATION :

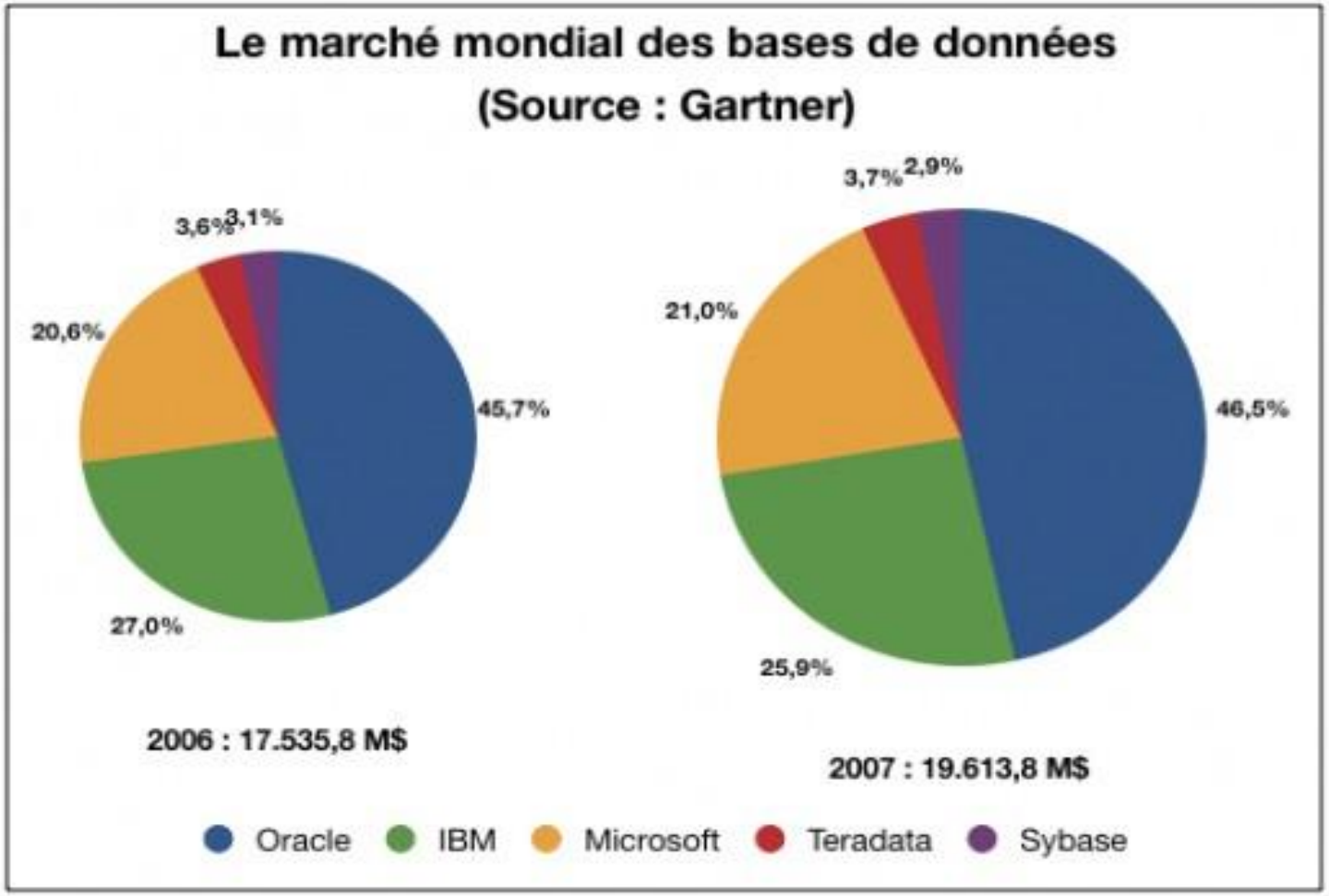
représentation sous forme de tables

- OPÉRATIONS RELATIONNELLES

algèbre relationnelle

- INDEPENDANCE PHYSIQUE
  - optimisation des accès
  - stratégie d'accès déterminée par le système
- INDEPENDANCE LOGIQUE
  - concept de VUES
- MAINTIEN DE L'INTEGRITÉ
  - contraintes d'intégrité définies au niveau du schéma

Les SGBD du marché



Evolution des SGBD du marché

PL/SQL

PL/SQL Oracle

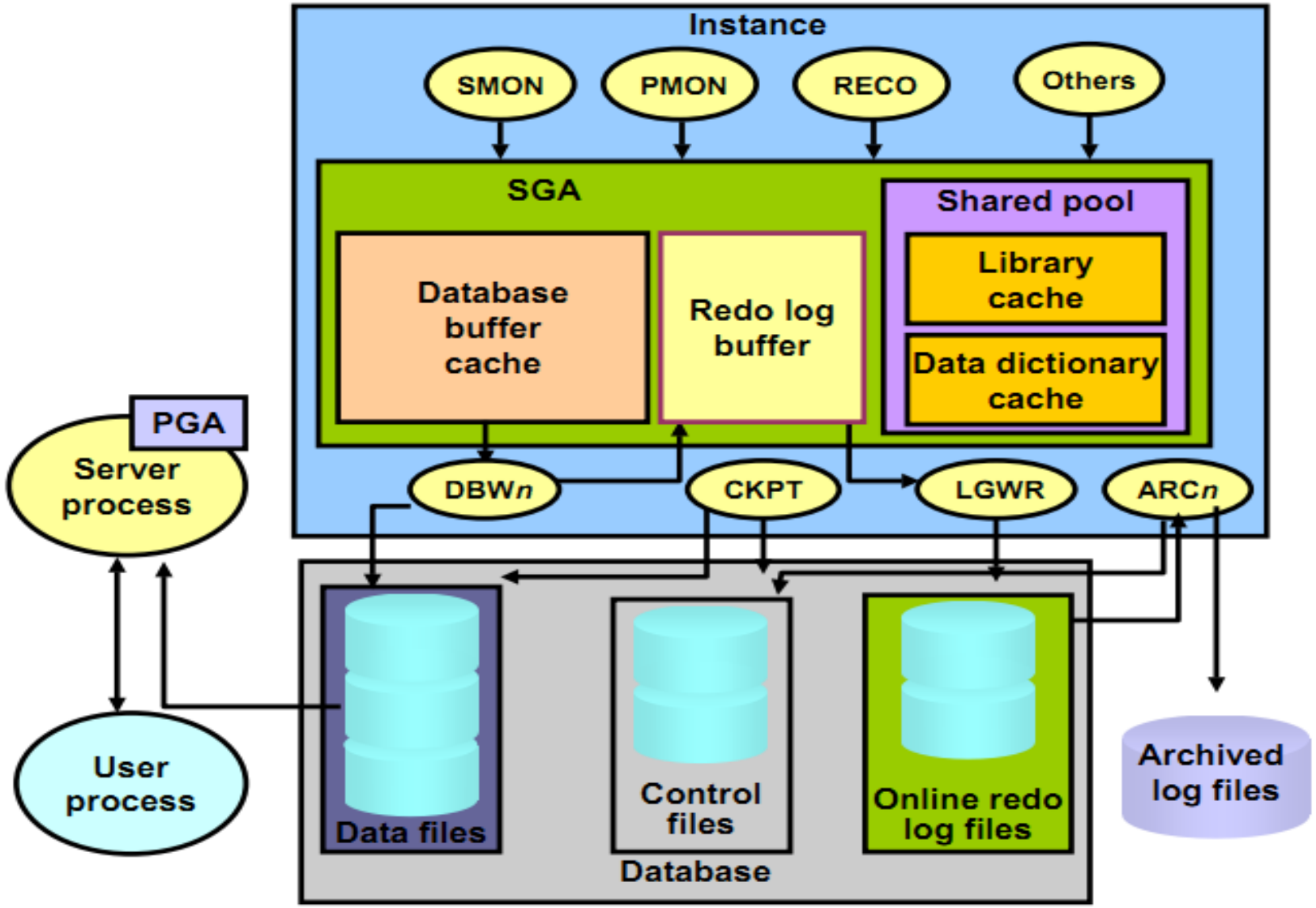
- C’est une extension au langage de programmation SQL.
- C’est un langage procédural d’ORACLE
- vous manipulez SQL en construisant des programmes, des fonctions.
- Vous devez déclarer des constantes et des variables, utiliser une bibliothèque d’objets.

PL/SQL est écrit en langage C, C++ et Java

- Intégration de SQL
- Traitement procéduraux
- Fonctionnalités supplémentaires
- Amélioration des performances
- Compilé et exécuter dans le moteur PL/SQL

- PL/pgSQL pour PostGreSQL
- TSQL Transact SQL de Microsoft

Architecture logique et physique



Structure d’un bloc PL/Sql

- Un programme ou une procédure PL/SQL est un ensemble de un ou plusieurs blocs.

Chaque bloc comporte trois sections :

- Section déclaration
- Section corps du bloc
- Section traitement des erreurs

Structure du bloc PL/SQL

DECLARE : facultatif

déclaration des variables, des constantes, des exceptions et des curseurs

Commence per le mot DECLARE

BEGIN obligatoire

instruction SQL, PL/SQL structure de contrôle

Introduite par le mot clé BEGIN

Se termine par le mot clé END

EXCEPTION facultatif

Traitement des erreurs

END;

On ajoute des commentaires :

/*texte*/

DECLARE est utilisé quand une variable est définie

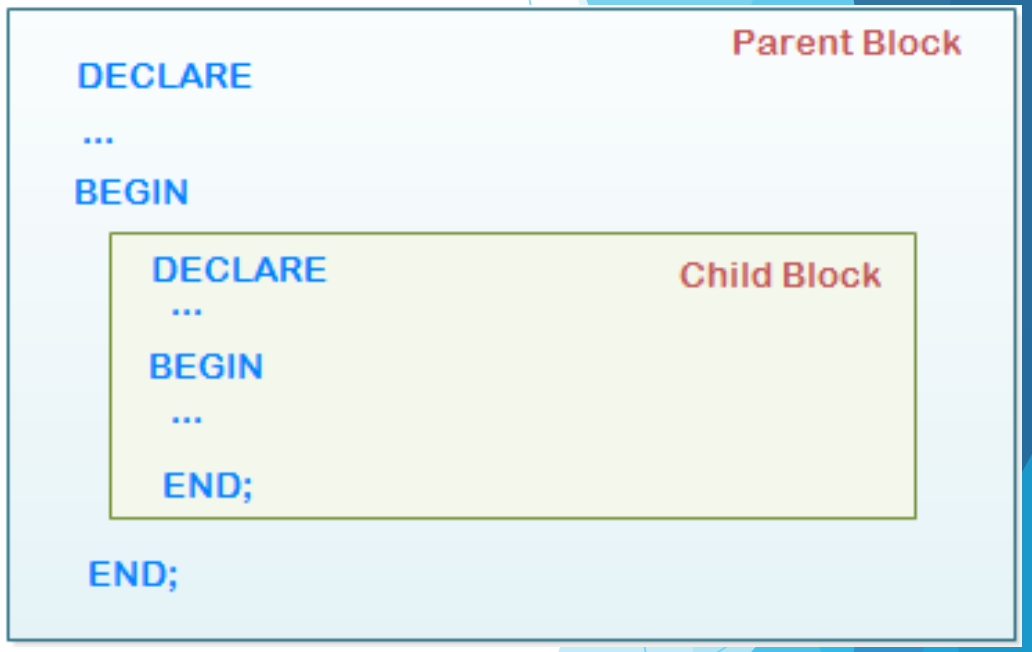
EXCEPTION est utilisé aussi lors de la gestion d'une erreur

Un bloc peut être imbriqué dans le code d'un autre bloc

(on parle de sous-bloc).

Un sous-bloc peut aussi se trouver dans la partie des exceptions.

Un sous-bloc commence par BEGIN et se termine par END.



Instructions SQL intégrées dans PL/SQL

Instructions SQL intégrées dans PL/SQL

- Interrogation : SELECT
- Manipulation : INSERT, UPDATE, DELETE
- Transactions : COMMIT, ROLLBACK, SAVEPOINT….
- LES FONCTIONS : TO_CHAR, TO_DATE, UPPER, ROUND…

Instructions spécifiques au PL/SQL

- Gestion des variables
- Structure de contrôle
- Gestion des curseurs
- Les traitements d'erreur

Déclaration d'une variable scalaire

Nom_variable [CONSTANT] typeDeDonnée [NOT NULL] [:= | DEFAULT expression];

- CONSTANT précise qu'il s'agit d'une constante ;
- NOT NULL empêche l'affectation d'une valeur nulle
- DEFAULT permet d'initialiser la variable (équivaut à l'affectation :=).
- Expression : valeur initiale affecté à la variable lors de l'exécution du bloc

```
DECLARE
v_dateNaissance DATE; /* équivaut à v_dateNaissance DATE:= NULL;*/
v_capacité   NUMBER(3) := 999;
v_téléphone  CHAR(14) NOT NULL := '06-76-85-14-89';
v_trouvé BOOLEAN NOT NULL := TRUE;
```

```
DECLARE c_pi CONSTANT NUMBER := 3.14159;
v_rayon  NUMBER := 1.5;
v_aire   NUMBER := c_pi * v_rayon**2;
v_groupeSanguin CHAR(3) := 'O+';/* équivaut à v_groupeSanguin CHAR(3) DEFAULT 'O+';*/
v_dateValeur DATE := SYSDATE + 2;
```

Syntaxe d'un bloc Pl/Sql

```
DECLARE
 déclaration
BEGIN
    corps-du-bloc
EXCEPTION
    traitement-des-erreurs
END;
```

Syntaxe d’un bloc Pl/Sql

Syntaxe à ajouter pour visualiser le résultat du bloc

```
SET SERVEROUTPUT ON
DECLARE X VARCHAR2(10);
BEGIN
X := ‘Bonjour’
      DBMS_OUTPUT.PUT_LINE(x);
END;
```

Gestion des variables

Le programme PL/SQL est capable de manipuler des variables et des constantes (dont la valeur est invariable).

Les variables et les constantes sont déclarées (et éventuellement initialisées) dans la section DECLARE.

Ces objets permettent de transmettre des valeurs à des sous programmes via des paramètres, ou d'afficher des états de sortie sous l'interface SQL*Plus.

Plusieurs types de variables sont manipulés par un programme PL/SQL :

Variables PL/SQL :

- scalaires recevant une seule valeur d'un type SQL (exemple : colonne d'une table) ;
- composites (%ROWTYPE, RECORD et %TYPE) ;
- références (REF) ;
- LOB (locators).

Utilisation des variables

Déclaration des variables scalaires

nom-variable nom-du-type;

nom-variable nom-table.nom-attribut%TYPE; référence à un attribut d'une table

Déclaration pour un enregistrement (record)

Soit par référence à une structure de table ou de curseur

en utilisant

ROWTYPE nom-variable nom-table%ROWTYPE;

nom-variable nom-curseur%ROWTYPE;

Déclaration pour un enregistrement (record) (2)

Soit par énumération des rubriques qui la composent.

Cela se fait en deux étapes :

Déclaration du type enregistrement

TYPE nom-du-type IS RECORD (nom-attribut1 type-attribut1, nom-attribut2 type-attribut2, ...);

Utilisation des variables

Les variables sont déclarées en dehors du bloc

```
Sql>variable x NUMBER
Sql>define t = JOBS
Sql>begin
2>select COUNT(*) INTO :x from &t;
3>end;
4>/
Sql>print x
Sql>
```

Valorisation de variable dans la section Begin

```
X:= 0
Vnom := 'Monsieur'||Vnon;
Y := (x+5) * y;
```

Affectation de la valeur

:= affectation directe, syntaxe :

nom_de_la_variable := expression

L'expression peut être soit une variable, un calcul, une constante

- Opérateurs arithmétiques +;-;*;/;**(exponentiation)
- Opérateur de concaténation ||

Affectations des variables

Il existe plusieurs possibilités pour affecter une valeur à une variable :

- l'affectation comme on la connaît dans les langages de programmation (variable := expression) ;

- par la directive DEFAULT ;

- par la directive INTO d'une requête (SELECT … INTO variable FROM …).

Variables %TYPE

- La directive %TYPE déclare une variable selon la définition d'une colonne d'une table ou d'une vue existante.
- Elle permet aussi de déclarer une variable conformément à une autre variable précédemment déclarée.
- Il faut faire préfixer la directive %TYPE avec le nom de la table et celui de la colonne

(identificateur nomTable.nomColonne%TYPE)

ou avec le nom d'une variable existante (identificateur2 identificateur1%TYPE).

```
DECLARE
v_employees emp.ID%TYPE;
v_ employees prend le type de la colonne ID de la table emp.

v_prime  NUMBER(5,2) := 500.50;
v_prime_min v_prime %TYPE := v_prime*0.9;
      v_prime_min prend le type de la variable v_prime
v_prime_min et est initialisée à 450,45.
BEGIN
```

Variables %ROWTYPE

- La directive %ROWTYPE permet de travailler au niveau d'un enregistrement .
- Ce dernier est composé d'un ensemble de colonnes.
- L'enregistrement peut contenir toutes les colonnes d'une table ou seulement certaines.
- Cette directive est très utile du point de vue de la maintenance des applicatifs. Utilisés à bon escient, elle diminue les changements à apporter au code en cas de modification des types des colonnes de la table.

- Il est aussi possible d'insérer dans une table ou de modifier une table en utilisant une variable du type %ROWTYPE.

DECLARE

La structure rty_employe est composée de toutes les colonnes de la table .

```
rty_pilote  pilote%ROWTYPE;
v_brevet Pilote.brevet%TYPE;
  SELECT * INTO rty_pilote
   FROM Pilote WHERE brevet='PL-1';
 v_brevet := rty_pilote.brevet;
rty_pilote.brevet := 'PL-9';
rty_pilote.nom := 'Pierre Bazex';
```

INSERT INTO Pilote VALUES rty_pilote; Insertion dans la table Pilote à partir d'un enregistrement.

Variables RECORD

- La directive %ROWTYPE permet de déclarer une structure composée de colonnes de tables, elle ne convient pas à des structures de données personnalisées.
- Le type de données RECORD définit vos propres structures de données
- Depuis la version 8, les types RECORD peuvent inclure des LOB (BLOB, CLOB et BFILE) ou des extensions objets (REF, TABLE ou VARRAY).

La syntaxe générale pour déclarer un RECORD est la suivante :

```
TYPE nomRecord IS RECORD
( nomChamp typeDonnées [[NOT NULL] {:= | DEFAULT} expression]
 [,nomChamp typeDonnées… ]… );
```

```
DECLARE
TYPE avionAirbus_rec IS RECORD
(nserie CHAR(10), nomAvion CHAR(20), usine  CHAR(10) := 'Blagnac', nbHVol NUMBER(7,2));
r_unA320  avionAirbus_rec; r_FGLFS   avionAirbus_rec;
BEGIN
r_unA320.nserie   := 'A1';
r_unA320.nomAvion := 'A320-200';
r_unA320.nbHVol   := 2500.60;
r_FGLFS  := r_unA320;
```

Variables RECORD

- Les types RECORD ne peuvent pas être stockés dans une table.
- En revanche, il est possible qu'un champ d'un RECORD soit lui-même un RECORD, ou soit déclaré avec les directives %TYPE ou %ROWTYPE.
- L'exemple suivant illustre le RECORD r_vols déclaré avec ces trois possibilités :

```
DECLARE
TYPE avionAirbus_rec IS RECORD
(nserie CHAR(10), nomAvion CHAR(20), usine  CHAR(10) := 'Blagnac', nbHVol NUMBER(7,2));
TYPE vols_rec IS RECORD
(r_aeronef avionAirbus_rec, dateVol DATE,
rty-coPilote%ROWTYPE, affrétéPar compagnie.camp%TYPE;
```

Paquetage DBMS_OUTPUT

Oracle possède un nombre de paquetages pour le développement des applications

- Ce paquetage assure la gestion des entrées-sorties de blocs PL/SQL.
- Les procédures du paquetage DBMS_OUTPUT les plus utiles pour la suite de ce cours sont :
  - ENABLE : active les entrées-sorties.
  - DISABLE : désactive les entrées-sorties.
  - PUT (expression) : place l'expression à la fin du buffer de sortie, c'est-à-dire affiche l'expression à l'écran.
  - NEW_LINE : ajoute le caractère fin de ligne dans le buffer, c'est-à-dire qu'on fait un retour à la ligne.
  - PUT_LINE (expression) : appel de PUT suivi d'un appel de NEW_LINE.
  - GET_LINE (out chaîne, out état) : lecture d'une ligne dans les paramètres chaîne. état prend la valeur 0 si la valeur lue est valide.
  - GET_LINES (out chaîne, in-out nbre_lignes) : permet la lecture de plusieurs lignes.
- Avant d'utiliser ce paquetage, on doit l'activer au préalable avec la commande SET SERVEROUTPUT ON.

Exemple : Paquetage DBMS_OUTPUT

```
 -- affichage.sql
-- activation du paquetage sous SQL*PLUS
SET SERVEROUTPUT ON
DECLARE
   name VARCHAR2(25) := 'Didier';
   job VARCHAR2(25) :='formateur';
   sal NUMBER := 500;

BEGIN

   DBMS_OUTPUT.ENABLE; -- Activation du paquetage sous PL/SQL
   DBMS_OUTPUT.PUT_LINE('Votre nom est        '|| name||',');
   DBMS_OUTPUT.PUT_LINE('votre fonction est   '|| job||',');
   DBMS_OUTPUT.PUT_LINE('et votre salaire est '|| sal||'.');

END;
/
```

La clause INTO

- La clause INTO des instructions permet la valorisation des variables à partir d'une ligne issue d'une requête

Atelier sur la Base SCOTT

But

- *Déclaration d'une variable %TYPE*
- *Déclaration d'une variable %ROWTYPE*
- *Valorisation de la variable dans le bloc BEGIN*

1. *Modification de la table JOBS à partir des valeurs de variables*

Toutes les employées qui ont un salaire minimum de 4000, on augmente le min de 400 et le max de 350.

Export ORACLE_SID=orcl

Sqlplus /nolog

Sql>connect /as sysdba

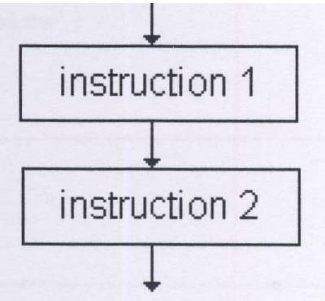
Sql>startup

Convention d'écriture

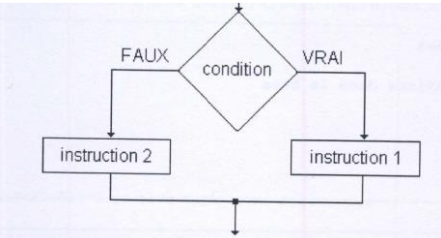
Objet	Convention	Exemple
Variable	v_nomVariable	v_compteur
Constante	c_nomConstante	c_pi
Exception	e_nomException	e_pasTrouvé
Type RECORD	nomRecord_rec	pilote_rec
Variable RECORD	r_nomVariable	r_pilote
Variable ROWTYPE	rty_nomVariable	rty_pilote
Type-tableau	nomTypeTableau_tytab	pilotes_tytab
Variable tableau	tab_nomTableau	tab_pilotes
Curseur	nomCurseur_cur	pilotes_cur
Variable de substitution (SQL*Plus)	s_nomVariable	s_brevet
Variable de session (globale)	g_nomVariable	g_brevet

Structure de contrôle permet de choisir l'exécution d'une instruction

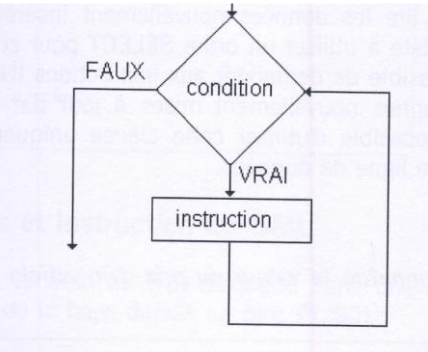
- La séquence : exécution des instructions les unes après les autres



- L'alternative (Sélection) instruction exécutée en fonction d'une condition



- La répétition (Itération): instruction exécutée plusieurs fois en fonction d'une condition



Traitement conditionnel

il lance une instruction en fonction du résultat d'une condition

- IF
  - Syntaxe

    IF condition1 THEN traitement1

    (ELSIF condition2 THEN traitement 2;)

    [ELSE

    Traitement 3;]

    [END IF;]

- Les opérateurs utilisés : =; <; > ; !; >= ; <= ; IS NULL, NOT NULL, BETWEEN, LIKE, AND, OR….

Traitement conditionnel

Avec IF THEN

```
IF condition THEN
   sequence_of_statements
END IF;
```

Avec IF-THEN-ELSE

```
IF condition THEN
   sequence_of_statements1
ELSE
   sequence_of_statements2
END IF;
```

IF-THEN-ELSIF

```
IF condition1 THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

Atelier 1 en utilisant les conditions IF

Avec IF THEN

Sur la base SCOTT :

Insérer le résultat dans la base : ajout de la prime au salaire

	Chiffre d'affaire	Objectif
commercial 1	35000	35000
commercial 2	30000	40000
commercial 3	50000	40000
commercial 4	65000	40000

Si un commercial a dépassé le montant des objectifs de vente , il percevra une prime équivalente à 10% du total de la vente.

Avec IF-THEN-ELSE

IF condition THEN

sequence_of_statements1

ELSE

sequence_of_statements2

END IF;

Si un commercial a dépassé le montant des objectifs de vente , il percevra une prime équivalente à 10% du total de la vente.

Si il a juste égalé le montant de ses ventes , il recevra une prime de fin d'année équivalente à 500 euros

Insérer le résultat dans la base : ajout de la prime au salaire

Atelier en utilisant les conditions IF

	Chiffre d'affaire	Objectif
commercial 1	35000	35000
commercial 2	30000	40000
commercial 3	50000	40000
commercial 4	65000	40000

IF-THEN-ELSIF

```
IF condition1 THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

Si un commercial a dépassé le montant de 55000, il percevra une prime équivalente à 10% du total de son chiffre d ‘affaire.

Si un commercial a dépassé le montant de 45 000 , il recevra une prime équivalente à 5% de son chiffre d ‘affaire

Sinon il recevra une prime de fin d’année équivalente à 500 euros

Insérer le résultat dans la base : ajout de la prime au salaire

Traitement conditionnel

- CASE exécution conditionnelle comme le IF mais particulièrement adaptée aux conditions comportant de nombreux choix
  - Syntaxe

<<Label>>
CASE élément de sélection
WHEN valeur1 THEN instruction1;
WHEN valeur2 THEN instruction2;
…..
[ELSE instructions;]
END CASE [étiquette]

<<Label>>

La condition ELSE est optionnelle

Environnement management : SQL*PUS

DESCRIBE : Description of table structure, view, synonym or stored codeSyntaxe :

DESC[CRIBE] objet

SPOOL : Output redirectionSyntaxe

SPO[OL] nomfic {/OFF/OUT}

Nomfic : file name

OFF ; close file

OUT : close file and Sends to the printer

Exemple :

spool /home/oracle/datesys;

SELECT SYSDATE FROM DUAL;

SPOOL OFF;

SHOW : Visualization of parameters

Syntax

SHO[W] {ALL/parameters}

Atelier en utilisant la condition CASE

Ex 1 Faire une comparaison avec une variable

Exemple : variable = 56 qui est un numéro de département.

Vous comparez la variable avec 5 noms de départements : Loire Atlantique, Ile et Vilaine, Finistère, Morbihan, Côtes d'Armor.

Résultats dans un fichier

Ex 2 Imbriquer deux instructions CASE

Faire une comparaison avec une variable

Exemple : variable = 56 qui est un numéro de département.

Vous comparez la variable avec 3 régions : Ile de France, Bretagne, Pays de Loire

Ile de France : n° département 75,77,91,95,78

Bretagne : n° département 56,29,22,35

Pays de Loire : n° département 44,85,49,53

Traitement répétitif

Instruction écrite une seule fois et exécutée plusieurs fois

- LOOP : utilisée seule, l'instruction initialise des boucles sans fin et systématiques

  Syntaxe

  LOOP [LABEL]

  Instructions;

  EXIT condition

  END LOOP [LABEL];

  L'instruction CONTINUE est arrivée dans la version 11 : permet d'interrompre l'itération en cours et de passer aussitôt à la suivante

Traitements répétitifs

- **Boucle FOR** : elle permet d'exécuter les instructions de la boucle en faisant varier un indice.

Les instructions sont exécutées autant de fois que l'indice change de valeur.

Syntaxe

FOR indice IN[REVERSE] exp1 exp2

LOOP

Instructions;

…

END LOOP [LABEL];

(Sans l'option REVERSE l'indice varie de 1, avec REVERSE l'indice varie de -1)

Atelier :
Faire une boucle afin de compter jusqu'à 15

- **Boucle WHILE** : l'entrée se fait si la condition est vraie et les instructions sont exécutées tant que la condition est vraie

La condition est une combinaison d'opérateurs <, >, =, !=, AND, OR , LIKE….

Syntaxe

WHILE condition LOOP

Instructions;

…

END LOOP [LABEL];

Les objets PL/SQL

Les Procédures

- Utilisées pour des actions spécifiques
- Elles utilisent le conditions comme IF-THEN, CASE, LOOP…

Les TRIGGERS

Un déclencheur est un bloc PL/SQL associé à une table. Ce bloc s'exécutera lorsqu'une instruction DML (INSERT, UPDATE, DELETE) sera demandée sur la table

Les FONCTIONS stockées : sert à renvoyer une valeur

Les CURSEURS : analyser toute ordre SQL

Les objets PL/SQL : les triggers

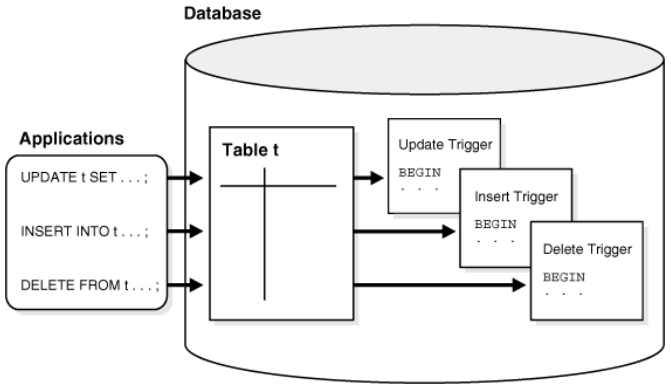
Les TRIGGERS

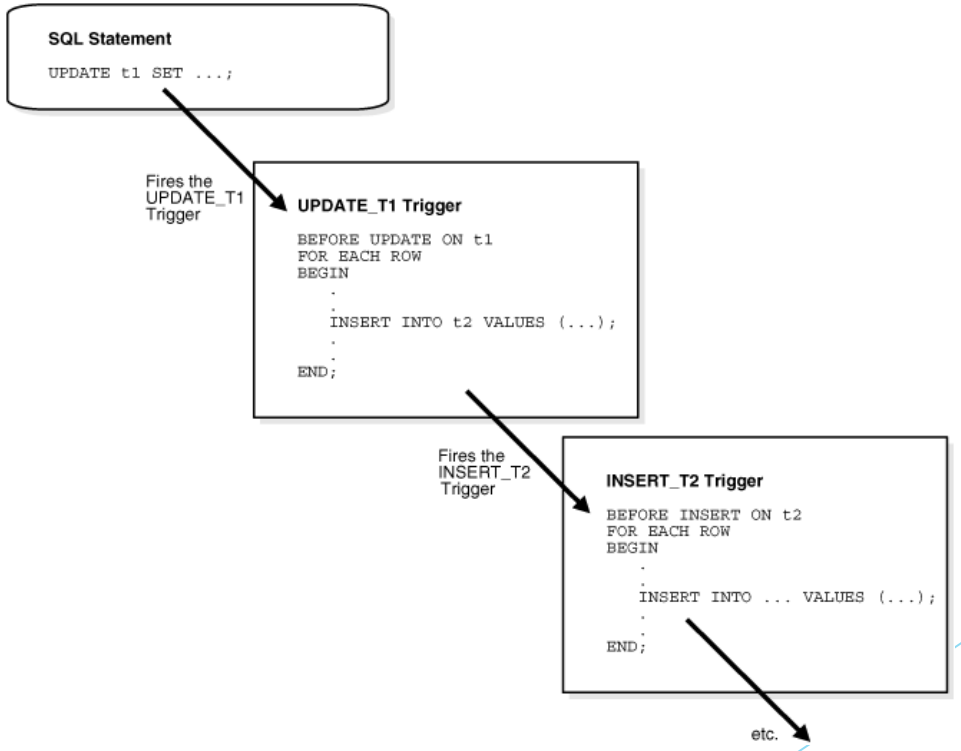
Un déclencheur est un bloc PL/SQL associé à une table.

Ce bloc s'exécutera lorsqu'une instruction DML (INSERT, UPDATE, DELETE) sera demandée sur la table

Ils ne sont pas paramétrables

Ils peuvent être désactivés





Les objets PL/SQL : les triggers

Les TRIGGERS

```
CREATE [OR REPLACE] TRIGGER schema.trigger_name
    BEFORE /AFTER/INSTEAD OF
    DELETE OR INSERT OR UPDATE
    ON schema.table_name [FOR EACH ROW]
WHEN (condition)
```

```
CREATE TRIGGER maj_sal
      AFTER  UPDATE OF sal on emp
      FOR EACH ROW
         WHEN (old.sal > new.sal)
```

Condition when

- La condition donnée doit être vérifié pour que le code s'exécute

- Seule la ligne en cours de modification est accessible à l'aide de deux variables RECORD, OLD et NEW

:OLD.nom_attribut,:new.nom_attribut

Exemple d’applications des triggers

- Empêcher des transactions invalides
- Vérifier les contraintes d’intégrité
- Générer automatiquement des valeurs de colonnes
- Compléter des procédures d’audit d’Oracle

Les triggers sur les évènements systèmes et utilisateurs

- Suivre les changements d'états du système

ora_client_ip_address	VARCHAR2	Adresse IP du poste client qui se connecte.
ora_database_name	VARCHAR2(50)	Nom de la base de données.
ora_des_encrypted_password	VARCHAR2	Mot de passe codé de l'utilisateur créé ou modifié.
ora_dict_obj_name	VARCHAR2(30)	Nom de l'objet du dictionnaire sur lequel l'opération LDD a eu lieu.
ora_dict_obj_name_list	BINARY_INTEGER	Liste de tous les noms d'objets qui ont été modifiés.
ora_dict_obj_owner	VARCHAR(30)	Propriétaire de l'objet de dictionnaire sur lequel l'opération LDD a eu lieu.
ora_dict_obj_owner_list	BINARY_INTEGER	Liste de tous les propriétaires d'objets qui ont été modifiés.
ora_dict_obj_type	VARCHAR(20)	Type de l'objet de dictionnaire sur lequel l'opération LDD a eu lieu.
ora_grantee	BINARY_INTEGER	Liste des utilisateurs qui possèdent ce privilège.

Les triggers sur les évènements utilisateurs

- Suivre les changements du aux utilisateurs

Syntaxe

Create trigger nom_trigger

AFTER/BEFORE

Evenement utilisateur on DATABASE/SCHEMA

Bloc pl/sql

After LOGON	VARCHAR2	Connexion serveur
Before logoff	VARCHAR2(50)	Connexion serveur
Before Create after create	VARCHAR2	Objet créé
Before DDL after DDl	VARCHAR2(30)	Ordres DDL
Before GRANT after GRANT	BINARY_INTEGER	Exécution Grant
Before revoke after revoke		
Before rename after rename		

Atelier TRIGGER

1 : Avec un TRIGGER : s'assurer si l'employé existe avant de l'insérer dans la base SCOTT

2 : Avec Un TRIGGER : avant d'ajouter la gratification de fin d'année,

Contrôler si la prime n'a pas été déjà ajoutée au salaire

(reprendre l'atelier sur le IF)

3 : Création d'un TRIGGER qui audite les modifications de salaire d'un employé