Cours de
Systèmes de Gestion des Bases de Données

Institut Supérieur d'Informatique de Mahdia **AU : 2011-2012**

Proposé par : Riadh HADJ M'TIR
Riadh.hadjmtir@riadi.rnu.tn
Classe : LAI2

Objectifs du cours

- Comprendre l'architecture du SGBD Oracle
- Construire une base de données sous Oracle
- Administrer une base de données
- Manipuler les données avec SQL
- Ecrire des programmes en PL/SQL

- La mise en pratique en TP

Plan du cours

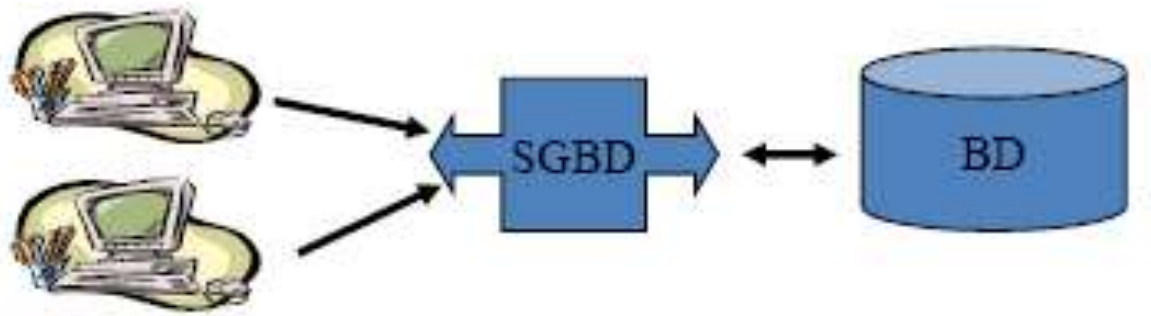
- Introduction
- Administration d'une BD Oracle
- Le langage SQL
- Le langage PL/SQL

Introduction Générale

Introduction

- Un **système de gestion de bases de données** (SGBD) est un logiciel qui permet d'interagir avec une base de données
- Un **SGBD** est un logiciel permettant de :

- Décrire		- la confidentialité
- Manipuler	***en assurant***	- l'intégrité
- Consulter les données		- la sécurité
- Définir des contraintes d'intégrités		- le partage des données

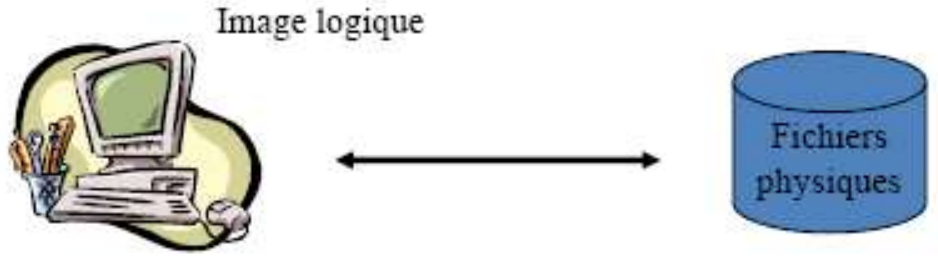


Objectifs des SGBD

- ***Indépendance physique*** (1) : Plus besoin de travailler directement sur les fichiers physiques (tels qu'ils sont enregistrés sur disque).

  Un SGBD nous permet de décrire les données et les liens entre elles d'une façon logique sans se soucier du comment cela va se faire physiquement dans les fichiers. On parle alors d'image logique de la base de données, (ou aussi description logique ou conceptuelle ou encore de schéma logique).

  Ce schéma est décrit dans un modèle de données par exemple le modèle de tables, appelé le **modèle relationnel**.



Introduction

Objectifs des SGBD (2)

- ***Indépendance physique*** (2) : La manipulation des données doit être facilitée en travaillant directement sur le schéma logique. On peut insérer, supprimer, modifier des données directement sur l'image logique. Le SGBD va s'occuper de faire le travail sur les fichiers physiques.

- ***Indépendance logique*** : Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrés dans une vision globale.

- ***Manipulations des données par des non informaticiens*** : Il faut pouvoir accéder aux données sans savoir programmer, ce qui signifie des langages « quasi naturels ».

- ***Efficacité des accès aux données*** : Ces langages doivent permettre d'obtenir des réponses aux interrogations en un temps « raisonnable ». Il doivent donc être optimisés et, entre autres, il faut un mécanisme permettant de minimiser le nombre d'accès disques. Tout ceci, bien sur, de façon complètement transparente pour l'utilisateur.

Introduction

Objectifs des SGBD (3)

- ***Administration centralisée des données*** : Des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.

- ***Cohérence des données*** : Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression de données, par exemple :
  - l'âge d'une personne supérieur à zéro
  - Salaire supérieur à zéro
  - Etc.

  Dés que l'on essaye de saisir une valeur qui ne respecte pas cette contrainte, le SGBD le refuse.

Objectifs des SGBD (4)

- ***Non redondance des données*** : Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.

- ***Partageabilité des données*** : Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations et quand on est dans un contexte mono-utilisateur, cela n'est plus le cas quand il s'agit de modifications dans un contexte multi-utilisateurs.

  Il s'agit alors de pouvoir :

  - Permettre à deux (ou plus) utilisateurs de modifier la même donnée « en même temps »;
  - Assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.

Introduction

Objectifs des SGBD (5)

- ***Sécurité des données*** : Les données doivent pouvoir être protégées contre les accès non autorisés.

  Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.

- ***Résistance aux pannes*** : Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles?

  Les pannes, bien qu'étant assez rares, se produisent quand même de temps en temps. Il faut pouvoir, lorsque l'une d'elles arrive, récupérer une base dans un état « sain ».

  Ainsi, après une panne intervenant au milieu d'une modification deux solutions sont possibles :

  - soit récupérer les données dans l'état dans lequel elles étaient avant la modification,
  - soit terminer l'opération interrompue.

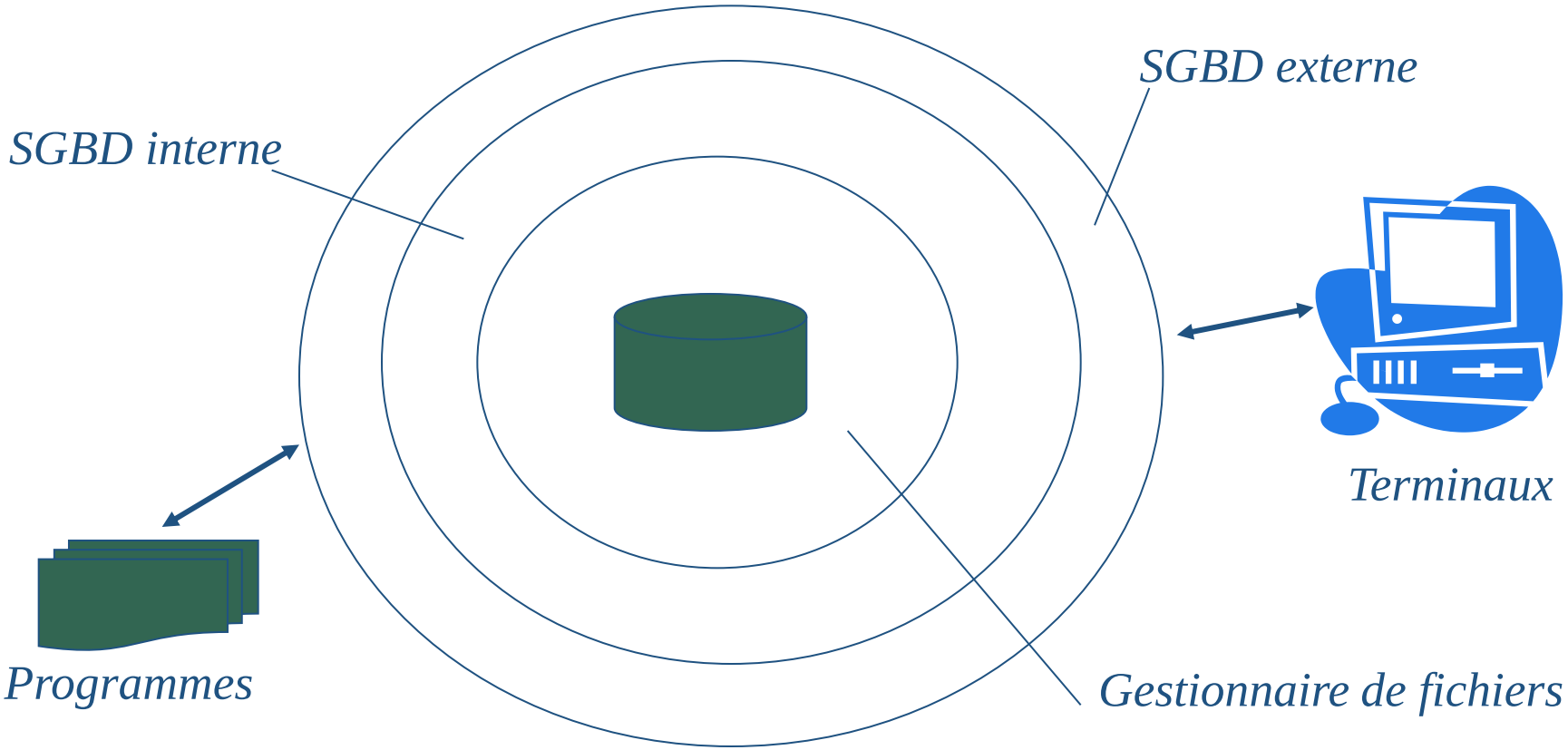
Introduction

Fonctionnalités d'un SGBD

- **Description des données** : codification structuration, grâce à un Langage de Description de Données (LDD)
- **Manipulation et restitution des données** (insertion, mise à jour, interrogation)
  - Mise en oeuvre à l'aide d'un Langage de Manipulation de Données (LMD)
  - S.Q.L. (Structures Query Langage) : Langage standard
- **Contrôle** (partage, intégrité, confidentialité, sécurité)
  - Schéma = structure + contraintes
  - Formule logique (E.g. Nom character 20, non NULL; age integer between 0 and 120; debit <= credit).
  - But: protéger les données

Introduction

vision simplifiée d'un SGBD



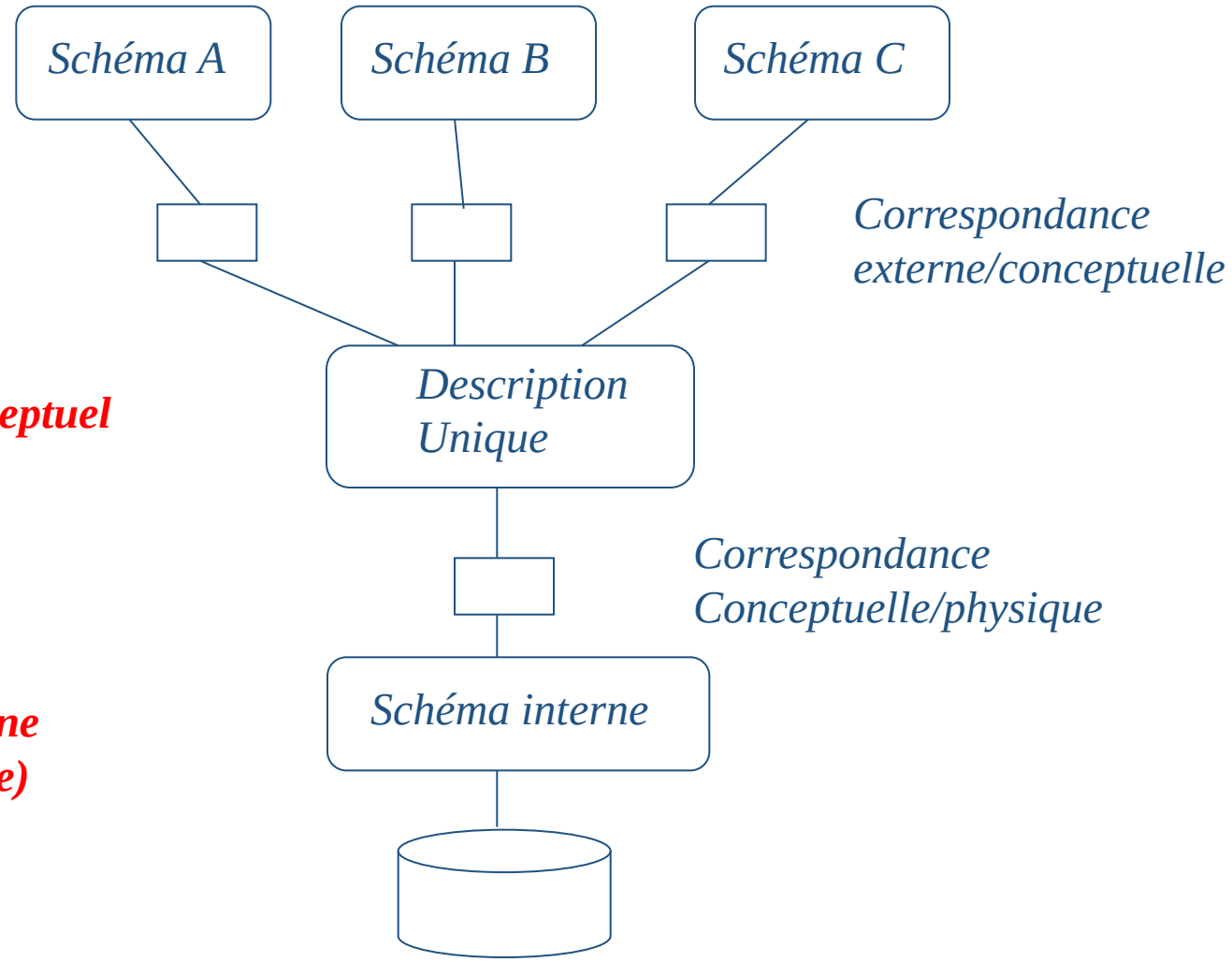
Introduction

Architecture fonctionnelle d'un SGBD

- **Architecture Ansi/Sparc – 3 niveaux:**
  - ***Niveau externe:*** le niveau d'expression des besoins des users. Il formalise la manière dont les utilisateurs voient les données.
    - Environnement de programmation / interfaces graphiques / deboggueurs / …
  - ***Niveau conceptuel:*** décrit la structure de la base de données globalement à tous les utilisateurs (limite la redondance) .
    - Compilation / optimisation des requêtes / maintien de l'intégrité / gestion de la confidentialité.
  - ***Niveau interne:*** relatif à la mémoire physique
    - Gestion de la mémoire secondaire (fichiers), schéma, index
    - Gestion de la concurrence
    - Reprise après panne

Introduction

Architecture à niveaux ANSI/SPARC



Introduction

Avantages de la séparation en 3 niveaux

- **Indépendance physique:** on peut modifier l'organisation des données sans toucher les programmes de traitement.
  - Limiter les modifications liées aux changements de matériel, de système d'exploitation ou des logiciels utilisés.

- **Indépendance logique:** une modification de l'organisation logique des fichiers (e.g. une nouvelle rubrique) n'entraîne pas de modification dans les programmes d'application non concernés.
  - la vision de chaque utilisateur est indépendante des visions des autres utilisateurs et n'est pas modifiée par les modifications du schéma conceptuel qui ne le concernent pas.

Introduction

Modèles de SGBD

- **SGBD Hiérarchique:**
  - Les données sont représentées dans la base sous forme de structure arborescente.
    - Manipulation des données (balayage ascendant/descendant).
  - E. g. IMS-IBM
- **SGBD réseau:**
  - Les données sont représentées dans la base sous forme d ’un graphe quelconque.
  - ***Les programmes***:
    - ne sont pas indépendants de la structure logique de la base
    - doivent indiquer le chemin d’accès aux données
    - utilisent un langage complexe pour travailler avec leurs données.
- **SGBD relationnel:**
  - fondé sur la théorie mathématique des relations;
  - représentation très simple des données (tables);
  - langage non procédural (déclaratif), puissant et simple d’emploi
  - SQL est un standard parmi ces langages
  - dominent le marché:
    - Exemples : Oracle, DB2, SQLServer, Access, DBase, Paradox, … etc.
- **SGBD Objet:**
  - enregistre les données sous forme d’objets
  - E. g. O2

LE SGBD ORACLE

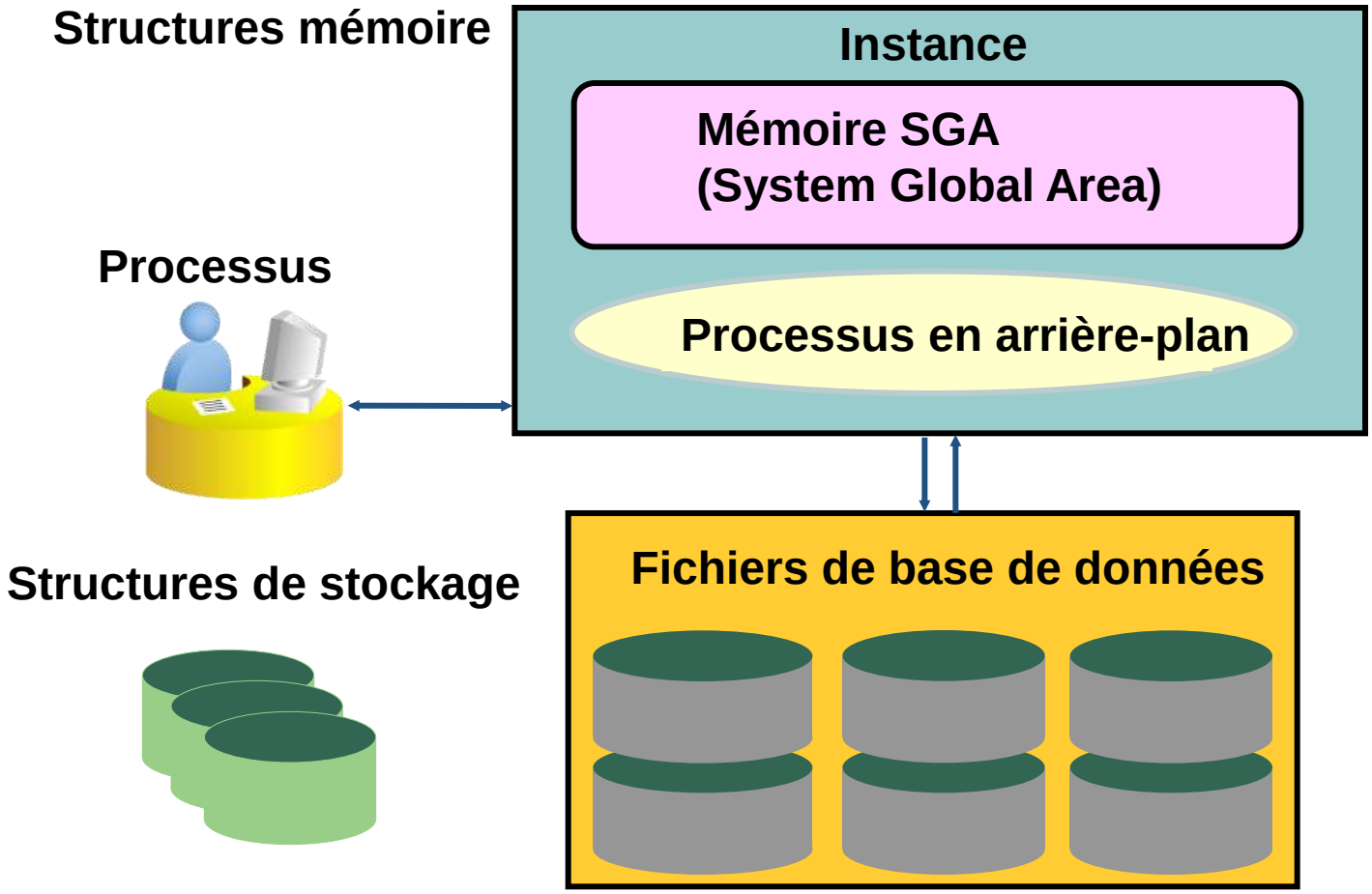
Chapitre 1

DBA
(Data Base Administration)

Objectifs du chapitre

- À la fin de ce chapitre, vous pourrez:
  - Comprendre la structuration du SGBD Oracle
  - Décrire l'architecture d'Oracle Database

Architecture d'une BD Oracle (1)



Architecture d'une BD Oracle (2)

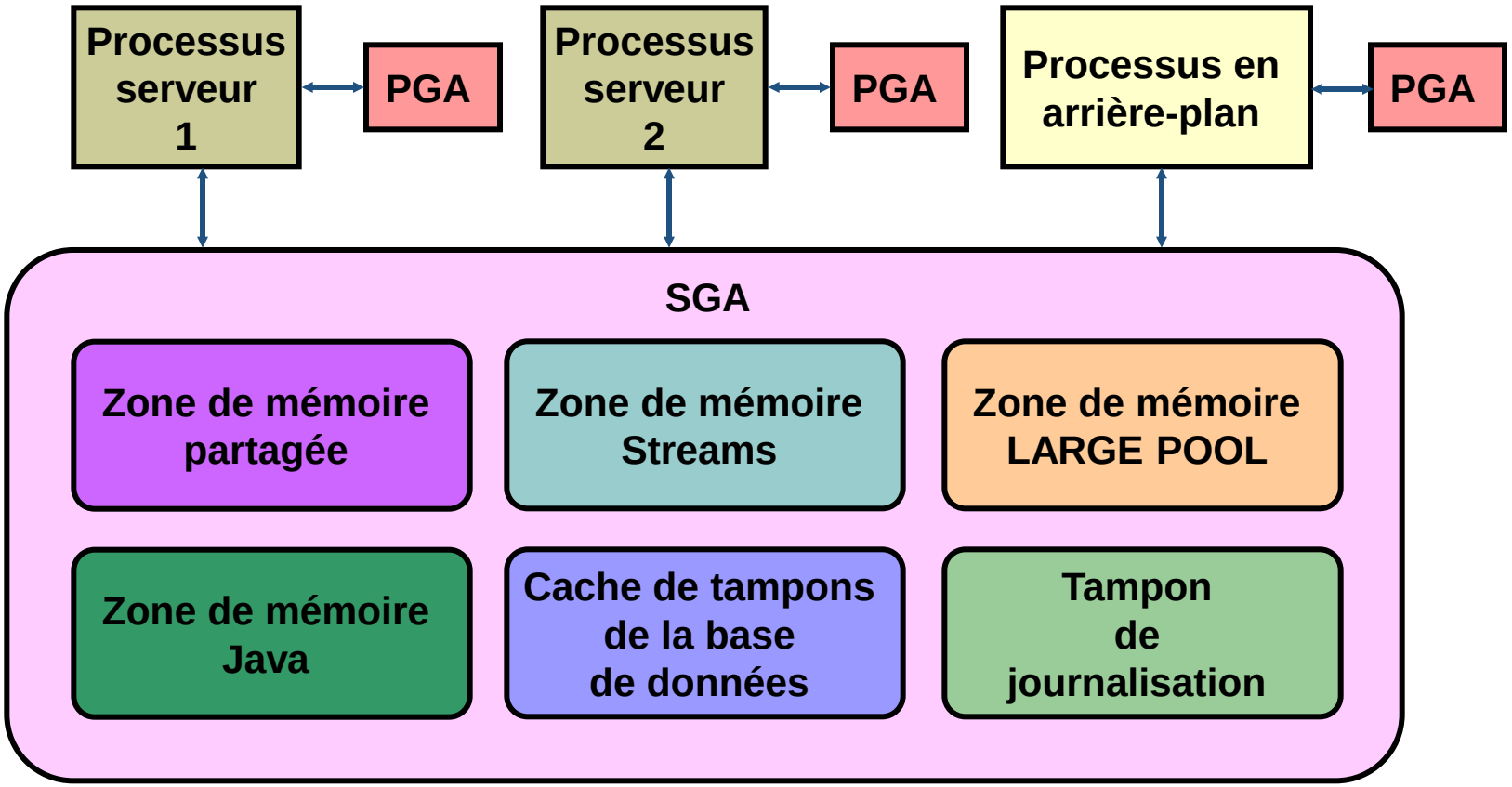
- Chaque BD qui s'exécute est associé à une instance oracle

- Une instance oracle est la combinaison d'une SGA et de processus oracle en arrière plan

Structures mémoire Oracle (1)

- Les structures mémoire de base associées à une instance Oracle sont :
  - **Mémoire SGA (System Global Area) :** partagée par tous les processus serveur et processus en arrière-plan
    - ✓ est une zone de mémoire contenant les données et les informations de contrôle de l'instance
  - **Mémoire PGA (Program Global Area) :** propre à chaque serveur et processus en arrière-plan
    - ✓ il existe une mémoire PGA pour chaque processus

Structures mémoire Oracle (2)

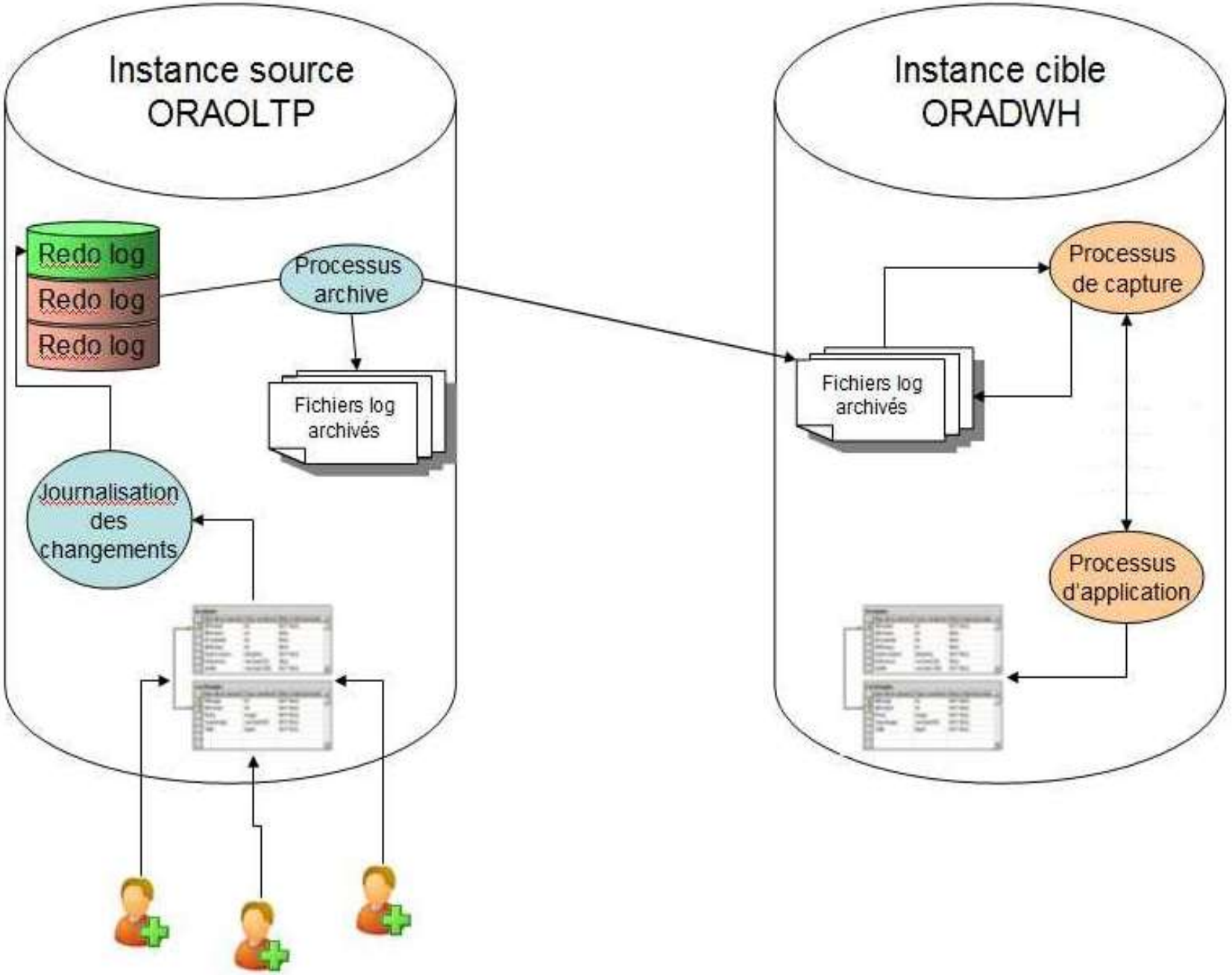


La mémoire SGA (1)

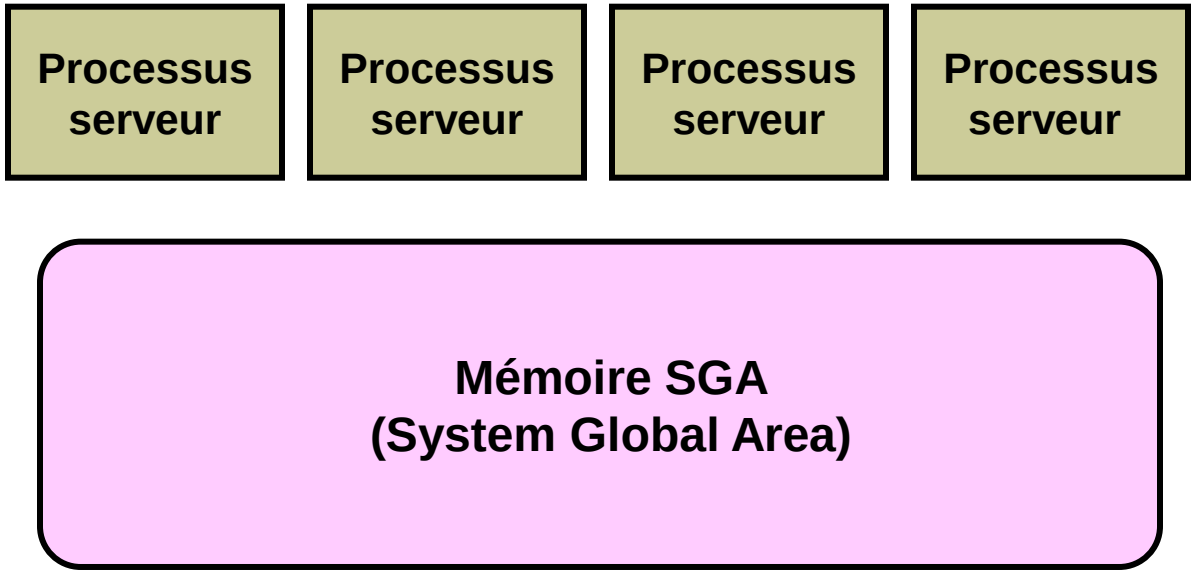
- La mémoire SGA contient les structures de données suivantes :
  - **Cache de tampons (buffer cache) de la base de données :** met en mémoire cache les blocs de données extraits de la base
  - **Tampon de journalisation (redo log buffer) :** met en mémoire cache les informations de journalisation (utilisées pour la récupération de l'instance) jusqu'à ce qu'elles puissent être écrites dans les fichiers de journalisation physiques stockés sur le disque
  - **Zone de mémoire partagée (Shared Pool) :** met en mémoire cache diverses structures pouvant être partagées par les utilisateurs,
    - utilisée pendant la phase d'analyse des ordres SQL passés à un processus Oracle, elle contient principalement le cache du dictionnaire de données (Dictionnary Cache) et le cache de bibliothèques (Library Cache)

La mémoire SGA (2)

- ...
  - **Zone de mémoire LARGE POOL :** zone facultative fournissant d'importantes allocations mémoire pour certains processus utilisant beaucoup de mémoire, tels que
    - les opérations de sauvegarde et de récupération Oracle et
    - les processus serveur d'entrée-sortie
  - **Zone de mémoire Java :** zone utilisée pour l'ensemble du code Java et des données propres à la session dans la JVM (Java Virtual Machine)
  - **Zone de mémoire Streams :** zone utilisée par Oracle Streams (pour la réplication)


Instance source
ORAOLTP
Redo log
Redo log
Redo log
Processus
archive
Fichiers log
archivés
Journalisation
des
changements
Instance cible
ORADWH
Processus
de capture
Fichiers log
archivés
Processus
d'application

Processus Oracle (1)

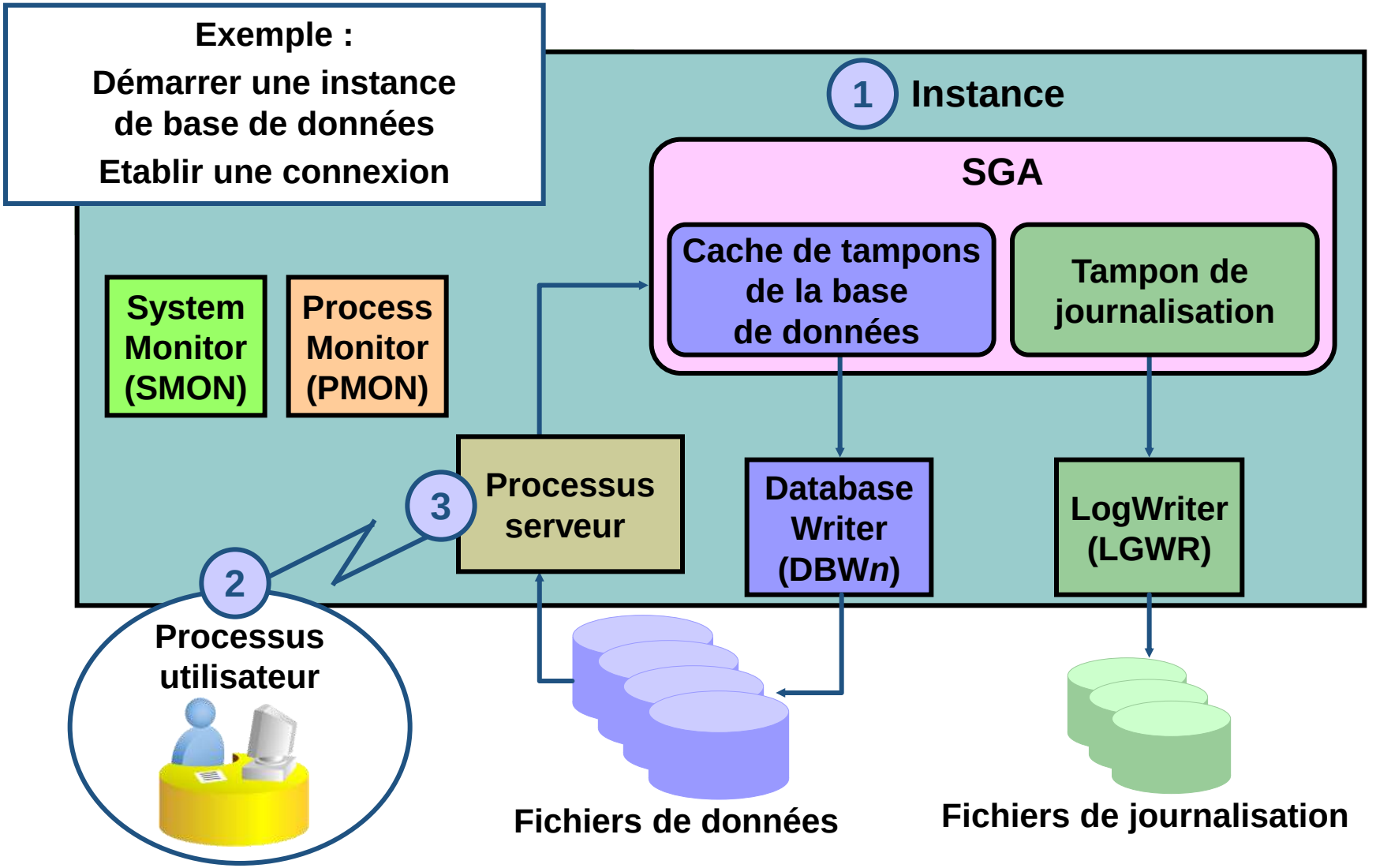


Processus en arrière-plan

Processus Oracle (2)

- **System Monitor (SMON) :** effectue une récupération après panne lorsque l'instance est démarrée après une défaillance.
- **Process Monitor (PMON) :** effectue un nettoyage de processus lorsqu'un processus utilisateur échoue.
- **Database Writer (DBW*n*) :** écrit les blocs modifiés du cache de tampons de la base dans des fichiers de données stockés sur disque.
- **Point de reprise (CKPT) :** met à jour l'ensemble des fichiers de données et de contrôle de la base afin d'indiquer le point de reprise le plus récent.
- **LogWriter (LGWR) :** écrit les entrées de journalisation sur le disque.
- **Processus d'archivage (ARC*n*) :** copie les fichiers de journalisation (fichiers redo log) dans le lieu destiné au stockage des archives lorsqu'un changement de fichier de journalisation se produit.

Gestion d'une instance Oracle (1)



Gestion d'une instance Oracle (1bis)

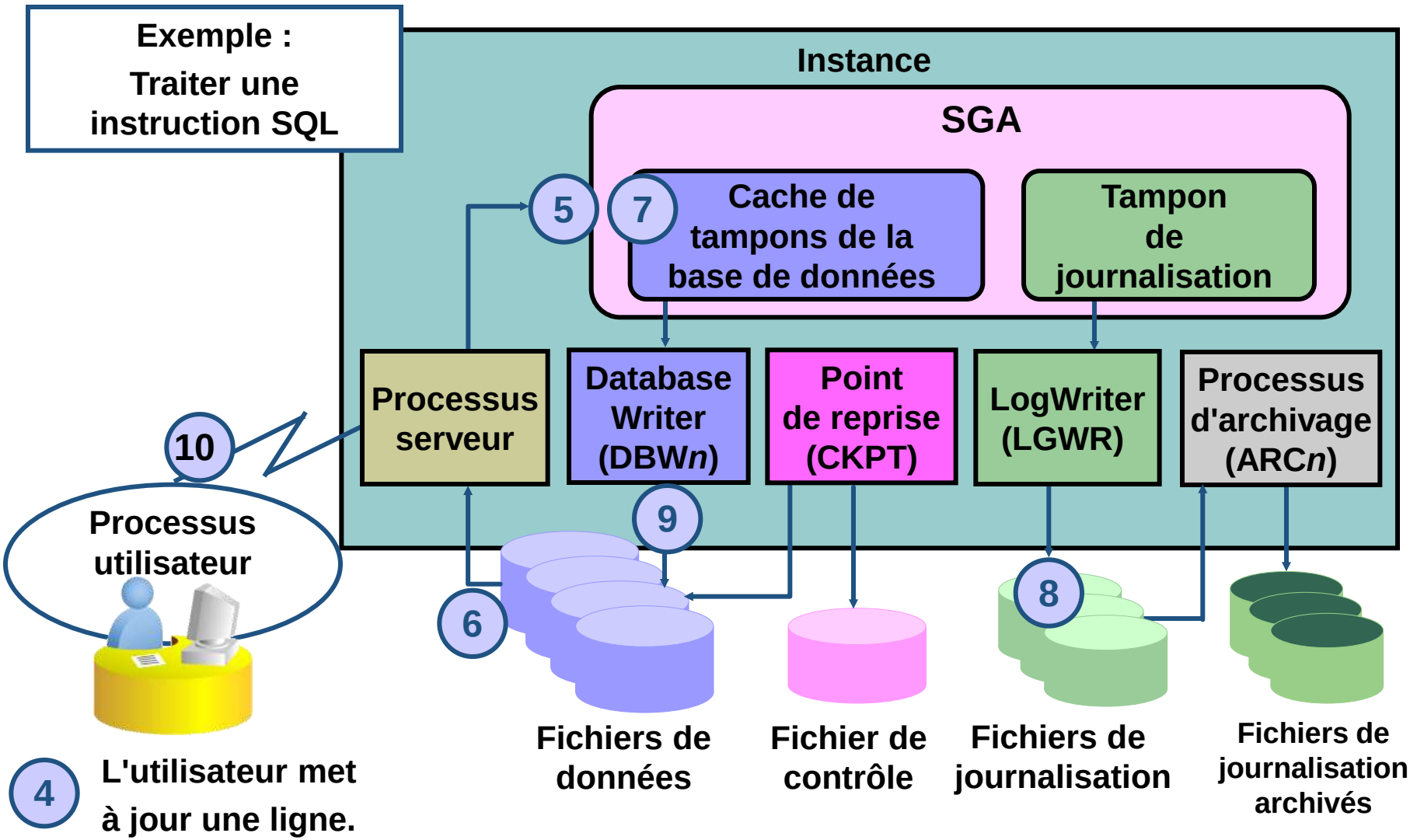
- L'exemple illustre une configuration Oracle dans laquelle l'utilisateur et les processus serveur associés utilisent des ordinateurs distincts (connectés entre eux via un réseau)

1. Une instance a été démarrée sur l'ordinateur exécutant Oracle (souvent appelé hôte ou serveur de base de données).

2. L'ordinateur exécutant l'application (ordinateur local ou poste client) utilise un processus utilisateur. L'application client tente d'établir une connexion avec l'instance en utilisant le pilote Oracle Net Services.

3. L'instance détecte la demande de connexion émanant de l'application et se connecte à un processus serveur pour le compte du processus utilisateur.

Gestion d'une instance Oracle (2)



Gestion d'une instance Oracle (2bis)

- …traiter une instruction SQL

4. L'utilisateur met à jour une ligne

5. Le processus serveur reçoit l'instruction et vérifie si elle se trouve déjà dans la zone de mémoire partagée de la mémoire SGA
   - ✓ Si une zone SQL partagée est détectée, le processus serveur vérifie les privilèges d'accès de l'utilisateur par rapport aux données demandées et la zone SQL partagée existante est utilisée pour le traitement de l'instruction.
   - ✓ Si l'instruction ne se trouve pas dans la zone de mémoire partagée, une nouvelle zone SQL partagée est allouée pour celle-ci afin qu'elle puisse être analysée et traitée.

6. Le processus serveur extrait les valeurs des données nécessaires du fichier de données (table) ou des blocs de données stockés dans la mémoire SGA

Gestion d'une instance Oracle (2bis)

- ...

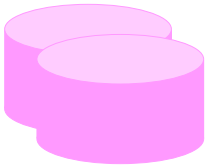
7. Le processus serveur modifie les données de la table dans la mémoire SGA.

8. Lorsque la transaction est validée (commit), le processus LGWR enregistre immédiatement la transaction dans le fichier de journalisation (fichier redo log).

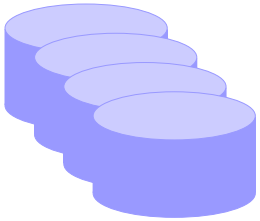
9. Le processus DBW*n* écrit les blocs modifiés sur le disque lorsque cela s'avère utile.

10. Le processus serveur envoie un message de succès ou d'erreur à l'application via le réseau.

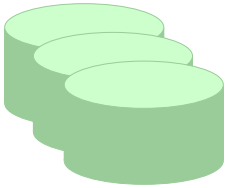
Structure physique d'une BD (1)



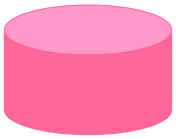
- Fichiers de contrôle



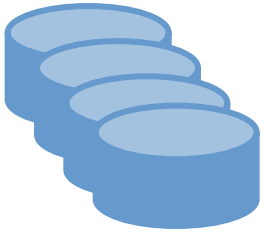
- Fichiers de données



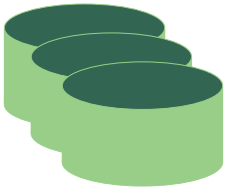
- Fichiers de journalisation en ligne



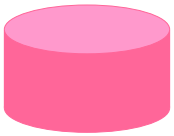
- Fichier de paramètres



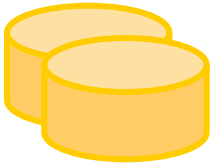
- Fichiers de sauvegarde



- Fichiers de journalisation archivés



- Fichier de mots de passe



- Fichiers d'alertes et fichiers trace

Structure physique d'une BD (2)

- Les fichiers constituant une base de données Oracle sont organisés de la façon suivante :
  - **Fichiers de contrôle :** contiennent des données sur la base elle-même (informations sur la structure physique de la base de données). Ces fichiers sont d'une importance capitale pour la base. Sans eux, vous ne pouvez pas ouvrir de fichiers de données pour accéder aux données de la base.
  - **Fichiers de données :** contiennent les données utilisateur ou les données d'application de la base.
  - **Fichiers de journalisation en ligne :** permettent la récupération d'une instance de base de données. S'il se produit une panne de la base sans perte des fichiers de données, l'instance peut récupérer la base grâce aux informations contenues dans ces fichiers.

Structure physique d'une BD (3)

- Les fichiers supplémentaires ci-dessous permettent à la base de données de s'exécuter correctement :
  - **Fichier de paramètres :** permet de définir comment l'instance est configurée lors de son lancement.
  - **Fichier de mots de passe :** permet aux utilisateurs de se connecter à distance à la base de données et d'effectuer des tâches d'administration.
  - **Fichiers de sauvegarde :** ces fichiers sont utilisés pour la récupération de la base de données. Les fichiers de sauvegarde sont généralement restaurés lorsqu'une défaillance physique ou une erreur utilisateur a endommagé ou supprimé les fichiers d'origine.

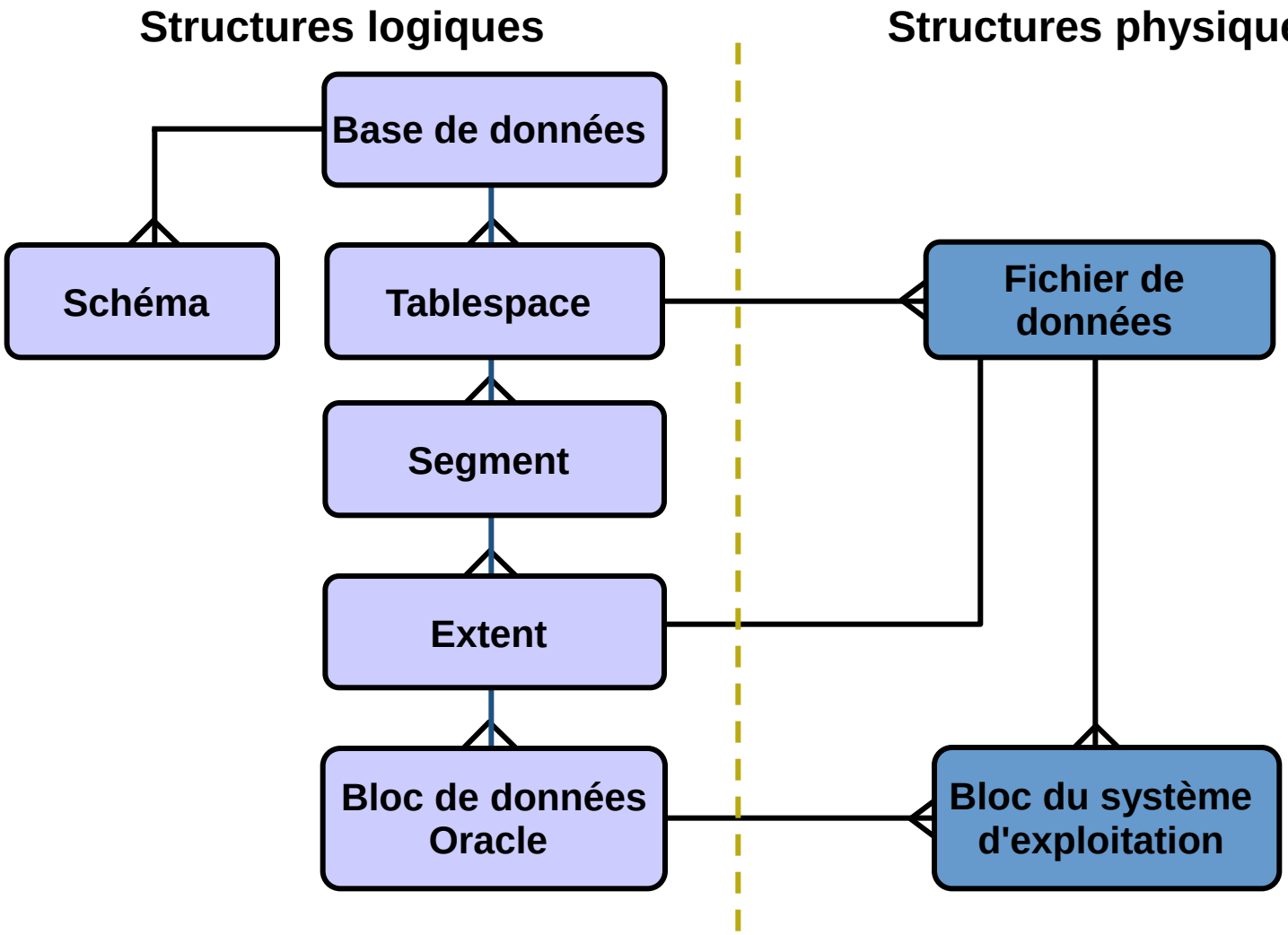
Structure physique d'une BD (4)

- ...
  - **Fichiers de journalisation archivés :** contiennent l'historique complet des modifications de données (informations de journalisation) générées par l'instance.
    - Vous pouvez, à l'aide de ces fichiers et d'une sauvegarde de la base, restaurer un fichier de données perdu.
    - Les fichiers de journalisation archivés permettent de récupérer des fichiers de données restaurés.
  - **Fichiers trace :** chaque processus serveur et chaque processus en arrière-plan peut écrire dans un fichier trace associé.
    - Lorsqu'une erreur interne est détectée par un processus, ce dernier procède à un dump des informations sur l'erreur vers son fichier trace.
    - Certaines informations écrites dans un fichier trace sont destinées à l'administrateur de base de données, tandis que d'autres s'adressent aux services de support technique Oracle. .
  - **Fichiers d'alertes :** sont des fichiers trace spécifiques. Le fichier d'alertes d'une base de données est un journal chronologique des messages et des erreurs. Oracle recommande de consulter ces fichiers.

Structures logiques et physiques d'une BD (1)

- Une base de données Oracle est un ensemble de données traité comme un tout.
- L'objectif général d'une base de données est de stocker et d'extraire des informations associées.
- Une base de données comprend des structures logiques et des structures physiques

Structures logiques et physiques d'une BD (2)



Tablespaces

- Une base de données est divisée en unités de stockage logiques appelées tablespaces, qui regroupent des structures logiques associées.
- Par exemple, les tablespaces regroupent habituellement tous les objets d'une application pour simplifier certaines opérations d'administration.
- Vous pouvez disposer d'un premier tablespace pour les données d'application et d'un second pour les index d'application.

Schémas

- Un schéma est un ensemble d'objets de base de données appartenant à un utilisateur de la base.
- Les objets de schéma sont les structures logiques qui font directement référence aux données de la base.
- Par exemple, ces structures peuvent être des tables, des vues, des séquences, des procédures stockées, des synonymes, des index, des clusters et des liens de base de données.
- En général, les objets de schéma incluent tout ce que l'application crée dans la base de données.

Blocs de Données

- Au niveau de détail le plus fin, les données d'une base Oracle sont stockées dans des blocs de données.
- Un bloc de données correspond à un nombre d'octets spécifique d'espace physique sur le disque.
- La taille du bloc de données est indiquée pour chaque tablespace lors de la création de celui-ci.
- Chaque base utilise et alloue de l'espace libre de base de données dans les blocs de données Oracle.

Extents

- **Extents (ensembles de blocs contigus)**
- Le niveau logique d'une base de données s'appelle un extent.
- Un extent est un nombre défini de blocs de données contigus (obtenus par une allocation unique) permettant de stocker un type spécifique d'informations.

Segments

- Le niveau logique de stockage situé au-dessus d'un extent s'appelle un segment.
- Un segment est un ensemble d'extents alloués pour une certaine structure logique.
- ***Seuls les objets 'physiques' peuvent être des segments. Ainsi une vue ou un synonyme n'est pas un segment...***

Segments (2)

- Les différents types de segment :
  - ***Segments de données*** **:** Chaque table comporte un segment de données. Toutes les données de la table sont stockées dans les extents de son segment de données.
  - ***Segments d'index*** **:** Chaque index comporte un segment d'index qui stocke l'ensemble de ses données.
  - ***Segments d'annulation (rollback segment)*** **:** qui stocke l'image avant modification des données.
    - Un tablespace `UNDO` est créé par l'administrateur de base de données afin de stocker temporairement les informations *d'annulation*.
    - Les informations contenues dans un segment d'annulation permettent de générer des informations de base de données cohérentes en lecture et, lors de la récupération de la base, d'annuler les transactions non validées pour les utilisateurs.
  - ***Segments temporaires*** **:** La base de données Oracle crée des segments temporaires lorsqu'une instruction SQL requiert une zone de travail temporaire pour son exécution (tri par exemple).
    - Lorsque l'exécution de l'instruction est terminée, les extents du segment temporaire sont rendus à l'instance en vue d'une utilisation ultérieure.
    - On peut indiquer un tablespace temporaire par défaut pour chaque utilisateur ou à l'échelle de la base de données.

BD Oracle : Récapitulatif des composants structurels

- Structures mémoire
  - **Mémoire SGA** (System Global Area) : cache de tampons (buffer cache) de la base de données, tampon de journalisation (redo buffer) et différents pools
  - **Mémoire PGA** (Program Global Area)
- Processus
  - **Processus utilisateur** et **processus serveur**
  - **Processus en arrière-plan** : SMON, PMON, DBWn, CKPT, LGWR, ARCn, etc.
- Structures de stockage
  - **Structures logiques** : base de données, schémas, tablespaces, segments, extents et blocs Oracle
  - **Structures physiques** : fichiers de données, fichiers de paramètres, fichiers de journalisation et blocs du système d'exploitation

Gestion d'une instance

Une instance Oracle

=

Ensemble des zones mémoires

+

Processus alloués à une base de données

Gestion d'une instance (2)

- Une instance de base de données Oracle peut être dans 4 statuts :
  - **Arrêtée**
  - **Etat NoMount**
  - **Etat Mount (montée)**
  - **Ouverte**

Gestion d'une instance (3)

Démarrage d'une base de données Oracle

```
sqlplus "/ as sysdba"
SQL> startup
```

Il est possible de démarrer les bases de données avec les commandes suivantes:

```
SQL> startup {nomount | mount}
SQL> alter database mount;
SQL> alter database open;
```

Gestion d'une instance (4)

- **Arrêt d'une base de données Oracle :**

  4 types d'arrêt

  - **Normal**
  - **Transactionnel**
  - **Immediate**
  - **Abort**

```
sqlplus "/ as sysdba"
SQL> shutdown (transactionnal/immediate/Abort)
```

Gestion d'une instance (5)

- Modes d'arrêt

Chapitre 2

Gestion des utilisateurs dans une BD Oracle

Les privilèges système

Les privilèges objet

Qu'est ce qu'un privilège?

- Les privilèges sont des droits pour exécuter des requêtes
- Le plus haut niveau de privilèges sont des privilèges DBA, il a la possibilité de donner aux utilisateurs l'accès à la base de données
- Les utilisateurs doivent posséder des privilèges système pour se connecter à la base de données, et les privilèges objets pour manipuler des données

Les privilèges Système

Les privilèges DBA

Privilège système	Opération autorisée
CREATE USER	Autorise de créer des utilisateurs
DROP USER	Autorise de supprimer des utilisateurs
DROP ANY TABLE	Autorise de supprimer toutes les tables dans tous les schémas
BACKUP ANY TABLE	Autorise de sauvegarder toutes les tables dans tous les schémas.
SELECT ANY TABLE	Autorise d'effectuer les requêtes SELECT dans tous les schémas.
CREATE ANY TABLE	Autorise de créer des tables dans tous les schémas.

Créer un utilisateur

- Un DBA peut créer des utilisateurs en utilisant la requête **CREATE USER**
- Lorsqu'un utilisateur est créé, il ne possède aucun privilège. Le DBA doit lui donner des privilèges souhaités

```
CREATE USER     utilisateur
IDENTIFIED BY   motdepasse;
```

Privilèges système accordés à un utilisateur

- Lorsque le DBA a créé un utilisateur, il lui donne des privilèges
- Exemple : Le DBA donne à l'utilisateur la possibilité de se connecter à la base de données. Ce privilège est donné grâce à CREATE SESSION

```
GRANT    privilege [,privilege, ... ]
TO       user [, user, role, PUBLIC ...];
```

Privilèges système accordés à un utilisateur (2)

Privilège	Opération autorisée
CREATE SESSION	Autorise à se connecter sur la base de données
CREATE TABLE	Autorise à créer des tables
CREATE SEQUENCE	Autorise à créer des séquences
CREATE VIEW	Autorise à créer des vues
CREATE PROCEDURE	Autorise à créer des procédures, des fonctions ou des packages

Accorder un privilège

- Pour accorder un privilège il faut suivre les étapes suivantes :

  1) Créer un nouvel utilisateur

  2) Donner le privilège **CREATE SESSION** à l'utilisateur

  3) Donner au nouvel utilisateur le privilège **CREATE TABLE**

Accorder un privilège (2)

```
SQL> CREATE USER         scott2
  2  IDENTIFIED BY       tiger2;

Utilisateur crée.

SQL> GRANT  CREATE SESSION
  2  TO                  scott2;

Autorisation de privilèges (GRANT) acceptée.

SQL> GRANT CREATE TABLE
  2  TO SCOTT2;

Autorisation de privilèges (GRANT) acceptée.
```

Créer et accorder un privilège à un rôle

- Un rôle est un ensemble de privilèges
- Lorsque le rôle est créé, le DBA utilise la requête GRANT pour assigner ce rôle aux utilisateurs

CREATE ROLE *nomrole;*

```
SQL> CREATE ROLE        manager;

Rôle crée.


SQL> GRANT CREATE TABLE, SELECT ANY TABLE
  2  TO                 manager;

Autorisation de privilèges (GRANT) acceptée.
```

Modification du mot de passe

ALTER USER *utilisateur*
IDENTIFIED BY *nouveaupassword;*

```
SQL> ALTER USER          scott2
  2  IDENTIFIED BY       oracle;

Utilisateur modifié.
```

Les privilèges Objet

Les privilèges Objet

Privilège objet	Table	Vue	Séquence	Procédure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X			
INSERT	X	X		
REFERENCES	X	X		
SELECT	X	X	X	
UPDATE	X	X		

Accorder les privilèges Objet

GRANT *privilege* **[(column)]**
ON *objectname*
TO *username* | *role* | *PUBLIC*
[WITH GRANT OPTION];

```
SQL> GRANT  UPDATE
  2  ON           emp
  3  TO           scott;

Autorisation de privilèges (GRANT) acceptée.
```

Les mots clés WITH GRANT OPTION et PUBLIC

- Le privilège accordé avec la clause WITH GRANT OPTION donne la possibilité au nouvel utilisateur d'accorder les privilèges sur cet objet aux autres utilisateurs
- Si vous enlevez ensuite le privilège à cet utilisateur, tous les utilisateurs à qui il aura donné ce privilège se le verront enlevé aussi de manière automatique

```
SQL> GRANT                SELECT, UPDATE
  2  ON                     dept
  3  TO                     scott
  4  WITH GRANT OPTION;

Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> GRANT          SELECT
  2  ON             emp
  3  TO PUBLIC;

Autorisation de privilèges (GRANT) acceptée.
```

Retirer les privilèges

REVOKE *privilege[,.., ALL]*
ON *objectname*
FROM *user | role | PUBLIC;*
[CASCADE CONSTRAINTS]

```
SQL> REVOKE      SELECT
  2  ON          emp
  3  FROM        scott;

Suppression de privilèges (REVOKE) acceptée.
```

Chapitre 3

Le Langage SQL

Partie I : Les ordre de SQL

Présentation

- SQL (Structured Query Language) ou langage de requêtes structuré est un langage destiné à interroger et manipuler les SGBDR
- Il a été introduit par IBM en 1979
- Le langage SQL est devenu Standard ANSI en 86 et ISO en 87
- Plusieurs révisions: SQL1 en 1989, SQL2 en 1992 en , SQL3 en 1999 et SQL2003

Présentation (suite)

- Le langage SQL comporte:
  - **LDD** : Langage de Définition de Données (définir les tables, les vues, ..)
  - **LMD** : Langage de Manipulation de Données (MAJ, Requêtes, ..)

Les ordres de SQL

Requête	Type de langage	Définition
SELECT	**DRL Data Retrieval Language** (Langage de récupération des données)	Ensemble de commandes qui permettent de récupérer les données contenues dans une ou plusieurs tables de la base.
INSERT *UPDATE* *DELETE* *MERGE*	**DML Data Manipulation Language** (Langage de manipulation des données)	Ensemble de commandes qui permettent de modifier les données de la base.
CREATE *ALTER* *DROP* *RENAME* *TRUNCATE*	**DDL Data Definition Language** (Langage de définition des données)	Ensemble de commandes qui permettent de modifier la structure de la base.
COMMIT *ROLLBACK* *SAVEPOINT*	**TCS Transaction Control Statement** (Ordre de contrôle de transaction)	Ensemble de commandes qui permettent d'administrer les changements effectués par les commandes DML.
GRANT *REVOKE*	**DCL Data Control Language** (Langage de contrôle des données)	Ensemble de commandes qui permettent de contrôler les accès utilisateur à la base de données.

Les types de données SQL

Types de données	Description
NUMBER	Chiffre
CHAR	Chaîne de caractères
VARCHAR2	Chaîne de caractères de longueur variable
DATE	La valeur de date et d'heure entre 1er Janvier 4712 avant JC et 31 Décembre 9999

Création d'une table

```
CREATE TABLE nom_de_table(
 attribut_1 type_1 [PRIMARY KEY],
  …
 attribut_n type_n
 //contrainte clé primaire
 [CONSTRAINT nom_const PRIMARY KEY(liste_attributs)],
 //contrainte clé étrangère
 [CONSTRAINT nom_const FOREIGN KEY(liste_attributs)
  REFERENCES nom_table(liste_attributs)],
 //contrainte d'intégrité
 [CONSTRAINT nom_const CHECK(condition)]

);
```

Exemple de Bases de données

Colloque (nomC, universiteC, adresseC, dateC)

Participant(numeroP, nomP, prenomP, adresseP)

Organisateur (numeroP#, telephone, nomC#)

Conferencier (numeroP#, institition)

Expose (titre, resume)

Inscrit (numeroP#, nomC#, droitinscription)

Programme(nomC#, numeroP#, titre#)

Création d'une table : exemple

```
CREATE TABLE participant(
    numeroP number(5),
    nomP char(30),
    prenomP char(30),
    addressP char(50)
);
```

Les contraintes : exemple

```
CREATE TABLE participant(
    numeroP number(5) PRIMARY KEY,
    nomP char(30) NOT NULL,
    prenomP char(30),
    addressP char(50)
);
```

Les contraintes : exemple (suite)

```
CREATE TABLE inscrit(
   numeroP number(5),
   nomC char(30),
   droitinscription number(5),
   constraint pri_key Primary Key (numeroP,
   nomC),
   constraint for_key1 Foreign Key (numeroP)
   references participant(numeroP),
   constraint for_key2 Foreign Key (nomC)
   references colloque(nomC)
);
```

Modification de tables

```
ALTER TABLE nom_table
  ADD|MODIFY(attribut TYPE,…);
```

Exemple:

```
ALTER TABLE participant MODIFY(numeroP number(3));
```

Suppression de contraintes

```
ALTER TABLE nom_table
  DROP CONSTRAINT nom_const;
```

La requête SELECT

```
SELECT liste_d_attributs
FROM   liste_de_tables
WHERE conditions;
```

SELECT indique la liste des attributs constituant le résultat

FROM le (les) tables utiles à la requête

WHERE indique les conditions que doivent satisfaire les tuples de la base pour faire partie du résultat

La requête SELECT (2)

Exemple 1 :

```
Select * from participant;
```

* remplace tous les tuples

Cette requête retourne tous les tuples de la table participant

La requête SELECT (3)

Exemple 2 :

```
Select numeroP, nomP
from participant
where prenomP = 'Salah';
```

Cette requête retourne le numéro et le nom de tous les participants ayant pour prénom **'Salah'**

Les doublons

Contrairement à l'algèbre relationnelle, SQL ne supprime pas les doublons

Exemple :

```
Select prenomP from participant
```

prenomP
Ali
Salah
Salma
Inès
Ali
Mohamed
Salah
Anis
Ali

Les doublons (suite)

Pour éviter d'avoir des tuples identiques en utilise le mot clé **DISTINCT**

Exemple :

```
Select DISTINCT(prenomP)
from participant;
```

PrenomP
Salah
Salma
Inès
Mohamed
Anis
Ali

Trier les résultats

- Il est possible de trier les données à l'aide de la clause **ORDER BY** suivie de la liste des attributs servant comme critère de tri

Exemple :

```
Select *
from participant
ORDER BY nomP, prenomP
```

La clause WHERE

- Dans la clause **WHERE** on spécifie une condition portant sur les attributs des relations mentionnées par la clause **FROM**
- Les opérateurs utilisés:
  - **AND**, **OR** et **NOT**
  - Les opérateurs de comparaison (<,>,<=,>=, <>)
  - **BETWEEN** pour les intervalles
  - **IN** pour un ensemble de valeurs

La clause WHERE (suite)

- Exemples:

```
select NomP
from Inscrit
where droitinscription BETWEEN 100 and 1000;
```

```
select NomP
from Participant
Where prenomP IN ('Ali','Amine','Hend');
```

Les chaînes de caractères

- SQL ne distingue pas entre les minuscules et majuscules pour les mots clés. Ceci n'est pas vrai pour les valeurs, **`'Tunisie'`** est différente de **`'TUNISIE'`**
- SQL fournit des options pour les recherches par motif (pattern) à l'aide de la clause **`LIKE`**
  - `'%'` désigne n'importe qu'elle chaîne de caractères
  - `'_'` désigne une lettre

Les chaînes de caractères (2)

- Exemples :

```
Select *
from colloque
where nomC LIKE '%a';
```

Tous les colloques dont le nom finit par '**a**'

```
Select *
from colloque
where nomC LIKE '__a%';
```

Tous les colloques dont la quatrième lettre est '**a**'

Les dates

Dans SQL une date s'écrit au format

'DD-MON-YY'

Exemple :

```
Select nomC
from colloque
where dateC BETWEEN '03-NOV-05' AND
'03-NOV-06'
```

Les valeurs nulles

- Les valeurs de certains attributs sont inconnus. Ils sont mis à **NULL**
- Toute comparaison avec **NULL** donne un résultat qui n'est ni vrai ni faux mais une troisième valeur **UNKNOWN**
- **NULL** est un mot clé et non pas une constante

  - **TRUE 1**
  - **UNKNOWN ½**
  - **FALSE 0**

Les opérations binaires

- Les connecteurs logiques deviennent
  - **x AND y = min(x,y)**
  - **x OR y = max (x,y)**
  - **NOT x = 1 - x**

Les requêtes sur plusieurs tables

- Les requêtes SQL décrites dans cette section permettent de manipuler simultanément plusieurs tables et d'exprimer des opérateurs d'algèbre relationnelle:
  - jointure,
  - union,
  - intersection,
  - ...

Jointures

- Une jointure permet d'exprimer des requêtes portant sur des données réparties sur plusieurs tables
- Exemple: Donner le nom et prénom de tous les participants d'un colloque

```
Select NomP, PrenomP
from Inscrit, Participant
where Inscrit.NumeroP =
Participant. NumeroP and
NomC='Colloque1';
```

Jointures (2)

- Dans l'ambiguïté (deux attributs ayant le même nom), on met le nom du table comme préfixe
- Comme ce n'est pas pratique de recopier intégralement le nom d'une table, on peut définir et utiliser des **synonymes**
- Exemple:

```
Select NomP, PrenomP
from Inscrit as I, Participant as P
where I.NumeroP =  P.NumeroP and
  NomC='Colloque1';
```

Jointures (3)

- On peut faire intervenir plus que deux tables
- Exemple:

```
Select NomP, PrenomP, DateC
from Inscrit I, Participant P, colloque C
where I.NumeroP =  P.NumeroP and
      I.NomC = C.NomC;
```

Jointure sur la même table

- Exemple:

  Donner des couples de colloques organisés par la même université

```
Select C1.NomC, C2.NomC
from colloque C1, colloque C2
Where C1.Universite = C2.Universite
```

Requêtes imbriquées

- Le résultat d'une requête sert comme condition dans la deuxième requête
- Ces types de requêtes peuvent porter sur plusieurs tables

Requêtes imbriquées (2)

❖Conditions portant sur des relations

Exemple:

```
Select numeroP
from inscrit
where nomC In ( select nomC
        from Colloque
        where Universite = 'Monastir');
```

On peut utiliser '=' à la place de **IN** si **on est sûr** que la sous requête ramène **un et un seul tuple**

Requêtes imbriquées (3)

❖Sous requêtes corrélées

Exemple:

Donner tous les participants ayant organisé au moins une fois et fait une présentation dans un colloque

```
Select numeroP
from organisateur O
where Exists (select *
        from conferencier C
        where O.numeroP = C.numeroP);
```

Agrégation

- Les fonctions d'agrégation permettent de faire des opérations sur une sélection de champ
- Elles effectuent un calcul sur un ensemble de valeurs et retournent une **valeur unique**
- À l'exception de COUNT, les fonctions d'agrégation ignorent les valeurs NULL
  - **`COUNT`** : nombre de lignes, ou de valeurs non nulles
  - **`MAX`** : calcule la valeur maximale d'une colonne
  - **`MIN`** : calcule la valeur minimale d'une colonne
  - **`AVG`** : calcule la moyenne d'une colonne
  - **`SUM`** : calcule la somme sur une colonne
- Les fonctions d'agrégation sont souvent utilisées avec la clause **GROUP BY** de l'instruction SELECT

Agrégation : exemples

```
Select count (NomC)
From colloque;

Select sum (droitinscription)
From inscrit
Where NumeroP = 100;

Select count(numStation), AVG (tarif), MAX
  (tarif), MIN (tarif)
From Station;
```

La clause GROUP BY

- Construit des groupes en associant les tuples partageant la même valeur pour une ou plusieurs colonnes

Exemple:

```
Select universite, count (NomC)
from colloque
Group by universite;
```

La clause HAVING

- On peut ajouter une condition sur le groupe

Exemple:

```
Select universite, count (NomC)
From colloque
Group by universite
Having count (NomC) >= 3;
```

Mise à jour d'une table : Insertion

```
Insert into R(a1,a2, .., aN )
values (v1,v2, .., vN );
```

Exemple :

```
Insert into Colloque (nomC, universiteC,
  adresseC, dateC)
values ('JJC','Manouba', 'Campus Manouba',
  '2007-03-30');

Insert into Inscrit (numeroP, nomC,
  droitinscription)
values ('153','JJC', 300);
```

Mise à jour d'une table : Destruction

```
Delete from R
Where condition;
```

Exemple :

```
Delete from Participant where NumeroP = 1210;
Delete from colloque where NomC = 'JAE';
```

Il faut respecter les contraintes d'intégrité référentielle !!

Mise à jour d'une table : Modification

```
Update R set a1=v1, a2=v2, … aN=vN;
```

Exemple :

```
Update inscrit set droitinscription =
  droitinscription/6.52;

Update colloque set universite = 'monastir';
```

Les vues

- Une vue est une sorte de fenêtre par laquelle des données peuvent être sélectionnées et échangées
- Une vue restreint l'accès aux données
- Une vue peut être utilisée pour construire des requêtes

CREATE VIEW nom_vue AS requête;

Partie II : SQL avancé

Les requêtes simples
Les sous-requêtes
Les fonctions de groupes
Le regroupement
Les fonctions individuelles
La conversion des données
Les fonctions générales
Les expressions conditionnelles
Les opérateurs de comparaison
Les vues
Les séquences
Les index
Les synonymes

Les requêtes simples

Syntaxe générale d'une recherche

SELECT <liste des attributs projetés>

FROM <liste des relations touchées par la question>

[**WHERE** <liste des critères de restriction>]

[**GROUP BY** <liste des attributs d'agrégation>]

[**HAVING** <liste des critères de restriction d'agrégats>]

[**ORDER BY** <liste des attributs de tri du résultat>]**;**

Ordre et choix des colonnes (SELECT)

- Exemple:

```
SELECT customer_name, city
FROM T_Customer;
```

customer_name	city
Contemporary Casuals	Gainsville
Value Furniture	Plano
Home Furnishings	Albany
Eastern Furniture	Carteret
Impressions	Sacramento
Furniture Gallery	Boulder
Period Furnishings	Seattle
California Classics	Santa Clara
M & H Casual Furniture	Clearwater
Seminole Interiors	Seminole
American Euro Lifestyles	Prospect Park
Battle Creek Furniture	Battle Creek
Heritage Furnishings	Carlisle
Kaneohe Homes	Kaneohe
Mountain Scenes	Ogden

Ordre des lignes (ORDER BY)

- Exemples:

```
SELECT customer_name, city
FROM T_Customer
ORDER BY customer_name;
```

customer_name	city
American Euro Lifestyles	Prospect Park
Battle Creek Furniture	Battle Creek
California Classics	Santa Clara
Contemporary Casuals	Gainsville
Eastern Furniture	Carteret
Furniture Gallery	Boulder
Heritage Furnishings	Carlisle
Home Furnishings	Albany
Impressions	Sacramento
Kaneohe Homes	Kaneohe
M & H Casual Furniture	Clearwater
Mountain Scenes	Ogden
Period Furnishings	Seattle
Seminole Interiors	Seminole
Value Furniture	Plano

```
SELECT customer_name, city
FROM T_Customer
ORDER BY city;
```

customer_name	city
Home Furnishings	Albany
Battle Creek Furniture	Battle Creek
Furniture Gallery	Boulder
Heritage Furnishings	Carlisle
Eastern Furniture	Carteret
M & H Casual Furniture	Clearwater
Contemporary Casuals	Gainsville
Kaneohe Homes	Kaneohe
Mountain Scenes	Ogden
Value Furniture	Plano
American Euro Lifestyles	Prospect Park
Impressions	Sacramento
California Classics	Santa Clara
Period Furnishings	Seattle
Seminole Interiors	Seminole

Expressions numériques

- **Opérations**
  - addition (+), soustraction (-)
  - multiplication (*), division (/)
  - modulo (%)
- **Ordre d'évaluation**

  1 - parenthèses

  2 - multiplication et division de gauche à droite

  3 - addition et soustraction de gauche à droite

Expressions numériques : Multiplication

- Exemple:

```
SELECT Material_ID, unit_price, unit_price*1.12
FROM T_Raws_Materials;
```

Material_ID	Unit_price	Unit_price*1.12
1	7,89	8,8368
2	12,05	13,496
3	13,67	15,3104
4	7,66	8,5792
5	7,23	8,0976
6	15,19	17,0128
7	13,02	14,5824
8	15,45	17,304
9	10,88	12,1856
10	15,55	17,416
11	8,79	9,8448
12	14,26	15,9712
13	15,82	17,7184
14	13,75	15,4
15	16,72	18,7264
16	5,70	6,384
17	7,95	8,904
18	11,13	12,4656

Expression textuelle : Concaténation

- Exemple:

```
SELECT customer_name, city || ', ' || state
FROM T_Customer
ORDER BY city;
```

```
customer_name            
-------------------------- ------------------
Home Furnishings           Albany, NY
Battle Creek Furniture     Battle Creek, MI
Furniture Gallery          Boulder, CO
Heritage Furnishings       Carlisle, PA
Eastern Furniture          Carteret, NJ
M & H Casual Furniture     Clearwater, FL
Contemporary Casuals       Gainsville, FL
Kaneohe Homes              Kaneohe, HI
Mountain Scenes            Ogden, UT
Value Furniture            Plano, TX
American Euro Lifestyles   Prospect Park, NJ
Impressions                Sacramento, CA
California Classics        Santa Clara, CA
Period Furnishings         Seattle, WA
Seminole Interiors         Seminole, FL
```

Les sous-requêtes

Les sous-requêtes

- SQL permet d'utiliser des requêtes pour élaborer des conditions plus complexes et "dynamiques"

un critère de recherche employé dans la clause WHERE soit lui même le résultat d'un SELECT

- **Types de sous-requêtes**
  - Sous-requêtes simples
  - Sous-requêtes synchronisées

Les sous-requêtes simples

- Dans ce cas la sous-requête est **d'abord évaluée**
- **puis le résultat est utilisé** pour exécuter la requête principale
- Exemple
  - sélectionner la liste des employés ayant même poste que JONES

```
select FIRST_NAME from EMPLOYEES
where JOB_ID = ( select JOB_ID from EMPLOYEES
                 where upper(FIRST_NAME)  = 'JONES' );
```

- Remarque
  - Les sous-requêtes sont plus lisibles que des jointures

Les sous-requêtes synchronisées

- Dans ce cas la sous-requête est **évaluée pour chaque n-uplet** de la requête principale
- Exemple
  - sélectionner des employés ne travaillant pas dans le même département que leur manager

```
select FIRST_NAME from EMPLOYEES e
where DEP_ID != (  select DEP_ID from EMPLOYEES
                   where  e.MANAGER_ID = EMPLOYEE_ID);
```

- Remarque
  - le synonyme « **e** » de la requête principale est utilisé dans la sous-requête

Les opérateurs associés aux sous-requêtes (1)

- L'opérateur EXISTS permet de construire un prédicat vrai si la sous-requête qui suit ramène au moins une ligne
- Syntaxe

```
select --- from nom-table where exists ( select --- )
```

- Exercice
  - Liste des employés travaillant dans un département qui contient au moins un ANALYSTE ?

Les opérateurs associés aux sous-requêtes (2)

Les opérateurs ensemblistes

- Dans les exemples précédents, le SELECT de la sous-requête ramenait **un seul n-uplet**, car à droite du signe "**=**" se trouvait une seule valeur
- Cependant une sous-requête peut ramener plusieurs n-uplets (une liste de valeur)
- Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont les opérateurs obtenus en ajoutant **ANY** et **ALL** à la suite d'un opérateur de comparaison
  - **ANY** : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble
  - **ALL** : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble

Les opérateurs associés aux sous-requêtes (3)

- Exercice
  - sélectionner les employés gagnant plus que tous les employés du département 30
- Remarque
  - L'opérateur IN est équivalent à l'opérateur = ANY
  - L'opérateur NOT IN est équivalent à l'opérateur != ALL
- Exercice
  - sélectionner les employés du département "RESEARCH" embauchés le même jour que quelqu'un du département "SALES"

Les opérateurs associés aux sous-requêtes (4)

- Les sous-requêtes ramenant plusieurs colonnes
  - Il est possible de comparer le résultat d'un SELECT ramenant **plusieurs colonnes** à une liste de colonnes
  - La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison
  - Syntaxe

```
select col1, ... from nom table
where (col2,col3,…) = (select col2, col3,… ---)
```

  - Exercice
    - sélectionner la liste des employés ayant même job et même salaire que FORD

Les fonctions de groupes

Les fonctions de groupes (1)

- Il est possible d'utiliser certaines fonctions en demandant de grouper les résultats selon une ou plusieurs colonnes
- Les fonctions de groupe sont :
  - **avg (col)**
    - moyenne des valeurs (les valeurs NULL sont ignorées)
  - **count (col)**
    - nombre de n-uplet satisfaisant à la condition WHERE. Les valeurs NULL sont ignorées.
  - **max (col)**
    - valeur maximale des valeurs de la colonne
  - **min (col)**
    - valeur minimale des valeurs de la colonne
  - **sum (col)**
    - somme des valeurs de la colonne
  - **variance (col)**
    - variance des valeurs de la colonne

Les fonctions de groupes (2)

SELECT product_id, product_name, product_description
FROM T_Product ;

```
Product_ID  Product_Name           Product_Description
----------- ---------------------  -------------------
1           End Table
2           Coffee Table
3           Computer Desk          Computer Desk 48
4           Entertainment Center
5           Writer's Desk
6           8-Drawer Desk
7           Dining Table
8           Computer Desk          Computer Desk 64
```

SELECT COUNT(*)
FROM T_Product ;

```
COUNT(*)
--------
       8
```

SELECT COUNT(product_description)
FROM T_Product ;

```
COUNT(product_description)
--------------------------
                         2
```

Les fonctions de groupes (3)

```
SELECT MIN(unit_price), MAX(unit_price)
FROM T_Product ;
```

MIN(unit_price)	MAX(unit_price)
175,00	800,00

```
SELECT MAX(order_date)
FROM T_Order;
```

MAX(order_date)
05/11/1998

Les fonctions de groupes (4)

```
SELECT SUM(on_hand*unit_price)
FROM T_Product;
```

```
11 150.00
```

```
SELECT AVG(unit_price)
FROM T_Product;
```

```
440.63
```

Regroupement

Calcul sur plusieurs groupes

- Il est possible de subdiviser la table en groupes,
- chaque groupe étant l'ensemble des lignes ayant une valeur commune.
  - Syntaxe

```
group by expr [,expr2 ...]
```

- Exemple
  - donner la somme des salaires pour chaque département

```
select SUM(SALARY) from EMPLOYEES
group by DEPARTEMENT_ID;
```

Regroupement: Ordre des critères

```
SELECT NO_CLIENT, NO_VENDEUR, SUM(MONTANT)
FROM COMMANDE
GROUP BY NO_CLIENT, NO_VENDEUR ;
```

NO_CLIENT	NO_VENDEUR	SUM(MONTANT)
1233	455	493,28
1233	687	132,33
4333	132	32,35
4333	687	137,98
4333	754	122,1
4436	132	100
5026	455	100
5026	687	44,32

```
GROUP BY NO_VENDEUR, NO_CLIENT ;
```

NO_CLIENT	NO_VENDEUR	SUM(MONTANT)
4333	132	32,35
4436	132	100
1233	455	493,28
5026	455	100
1233	687	132,33
4333	687	137,98
5026	687	44,32
4333	754	122,1

Regroupement: clause SELECT

- Dans un regroupement la clause SELECT a deux parties:
  - liste des attributs servant au regroupement
  - liste des fonctions ensemblistes évaluées sur chaque groupe


```
SELECT NO_CLIENT, SUM(MONTANT)
FROM COMMANDE
GROUP BY NO_CLIENT;
```

Regroupement: DISTINCT

- Le mot-clé **distinct** permet de supprimer les lignes identiques du résultat

```
SELECT NO_CLIENT
FROM COMMANDE ;
```

```
NO_CLIENT
---------
     1233
     4333
     4436
     1233
     1233
     4333
     1233
     5026
     4333
     5026
```

```
SELECT COUNT( NO_CLIENT)
FROM COMMANDE ;
```

```
COUNT(NO_CLIENT)
----------------
              10
```

```
SELECT DISTINCT NO_CLIENT
FROM COMMANDE ;
```

```
NO_CLIENT
---------
     1233
     4333
     4436
     5026
```

```
SELECT COUNT(DISTINCT NO_CLIENT)
FROM COMMANDE ;
```

```
COUNT(DISTINCTNO_CLIENT)
------------------------
                       4
```

Regroupement: clause HAVING

- La clause HAVING permet, de façon analogue à la clause WHERE, de poser des conditions sur le résultat du regroupement

```
select col1 [,col2 ...]
from table1 [, table2]
[where prédicat]
[group by expression1 [, expression2…]
[having prédicat] ];
```

- Le prédicat figurant dans la clause HAVING **ne peut porter que sur des caractéristiques des fonctions de groupe** figurant dans la clause GROUP BY

Fonctions individuelles

Fonctions de manipulation de casse

Fonctions	Définitions	Exemples	Résultats
INITCAP (colonne | chaîne)	Convertit la première lettre de chaque mot d'une chaîne de caractères en majuscule et les autres lettres en minuscule	Initcap('Cours de SQL')	Cours De Sql
LOWER (colonne | chaîne)	Convertit une chaîne de caractères en minuscule	lower('Cours de SQL')	cours de sql
UPPER (colonne | chaîne)	Convertit une chaîne de caractères en majuscule	upper('Cours de SQL')	COURS DE SQL

Fonctions de manipulation de caractères

Fonctions	Définitions	Exemples	Résultats
length (chaîne)	Nombre de caractères dans la chaîne	length(‘Bonjour')	7
instr(chaîne, sous-chaîne)	Emplacement d’une sous-chaîne spécifiée dans une chaîne	instr(‘Bonjour', ‘j')	4
substr(chaîne, i, j)	Extrait une sous-chaîne du $i^{ème}$ caractère jusqu’au $j^{ème}$ caractère	substr(‘Bonjour', 1,3)	Bon
lpad(chaîne, x [,text])	Remplit chaîne pour avoir une chaîne de longueur x en ajoutant le text (un espace par défaut)	lpad(sal,10, ‘*‘) Si chaîne a une taille supérieure à x alors elle est tronquée	******5000
rpad(chaîne, x [,text])		rpad(sal,10, ‘*')	5000******
concat(chaîne1, chaîne2)	Concaténer des chaînes	concat(‘Bon’,’jour’)	Bonjour
Trim([leading|trailing|both] [caractère] FROM chaîne)	Supprime la plus grande chaîne contenant seulement les caractères (un espace par défaut) à partir du début, de la fin ou des deux extrémités de la chaîne	trim(‘S’ FROM ‘SSSMITH’) trim(both 'x' from 'xTomxx')	MITH Tom

Fonctions de manipulation des dates (1)

- SYSDATE affiche la date système

```
SQL>        SELECT        SYSDATE
  2         FROM          dual;

SYSDATE
--------
04/08/04
```

Fonctions de manipulation des dates (2)

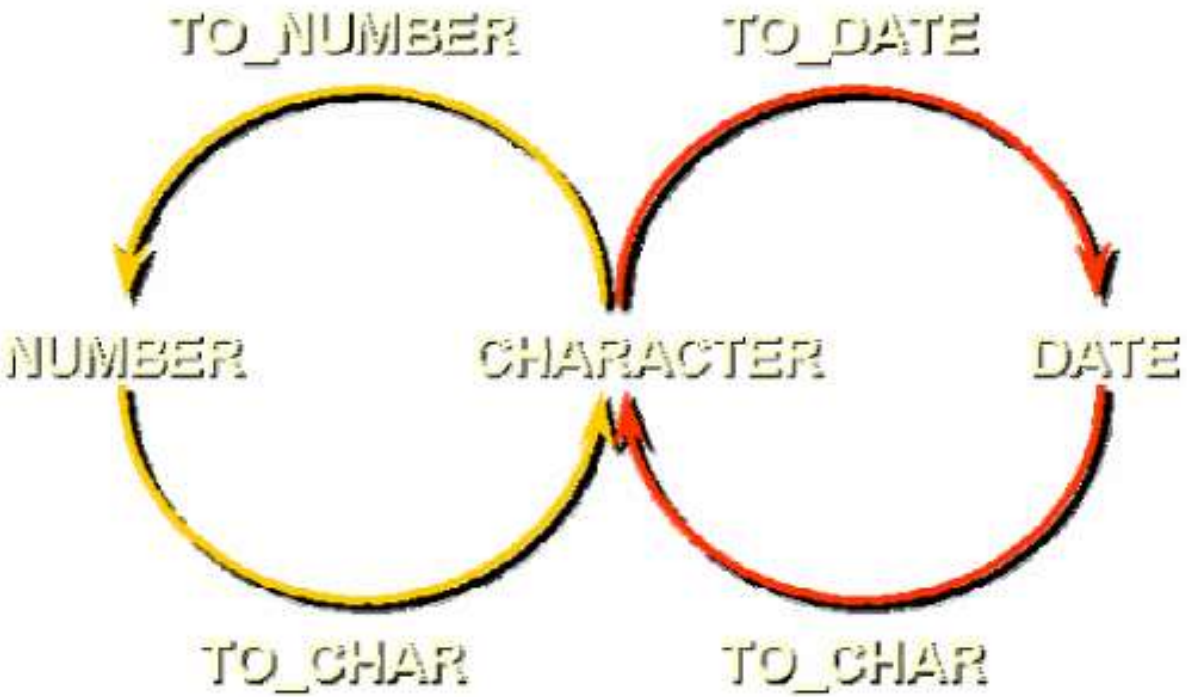
Opération	Résultat	Description
date + nombre	date	Ajoute un nombre de jours à une date
date - nombre	date	Soustrait un nombre de jours à une date
date - date	nombre de jours	Soustrait une date à une autre date
date + nombre/24	date	Ajoute un nombre d'heures à une date

Fonctions de manipulation des dates (3)

Fonction	Définition	Exemple	Résultat
MONTHS_BETWEEN (*date1*,*date2*)	Retourne le nombre de mois séparant deux dates. Le résultat peut-être positif ou négatif. Si *date1* est inférieure à la *date2*, le résultat est positif. Si *date1* est plus récente que *date2*, le résultat est négatif.	MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS(*date*,*n*)	Ajoute *n* mois à une date. *n* doit être un entier positif ou négatif.	ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY(*date*, '*day of week*')	Trouve la date du prochain jour de la semaine (*day of week*) suivant *date*. La valeur de *day of week* doit être un nombre représentant le jour ou une chaîne de caractères.	NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'

Conversion de données

Conversion



Conversion

Fonction	Définition
TO_CHAR(*number|date,[fmt],[nlsparams]*)	Convertit un nombre ou une date en une chaîne de caractères
TO_NUMBER *(char ['format'])*	Convertit une chaîne de caractères en un nombre
TO_DATE *(char [,'format'])*	Convertit une chaîne de caractères en une date

```
SQL>        SELECT        ename, TO_CHAR(hiredate,'MM/YY')
  2         FROM          emp
  3         WHERE         ename = 'KING';

ENAME      TO_CHAR
---------- -----
KING       11/81
```

```
SQL> SELECT       ename,
  2               TO_CHAR(hiredate, 'fmDD Month YYY') HIREDATE
  3  FROM         emp

ENAME      HIREDATE
---------- ----------------
SMITH      17 Décembre 980
ALLEN      20 Février 981
WARD       22 Février 981
JONES      2 Avril 981
MARTIN     28 Septembre 981
BLAKE      1 Mai 981
…
14 ligne(s) sélectionnée(s).
```

Les fonctions générales

La fonction NVL

- Syntaxe : NVL(expr1,expr2)
- Fonction : Substituer les valeurs nulles d'une colonne par une valeur choisie
- Exemple

```
SQL>        SELECT ename, NVL(TO CHAR(comm),'Pas de commission') comm
  2         FROM   emp;

ENAME      COMM
---------- ----------------------------------------
SMITH      3000
ALLEN      500
WARD       500
JONES      Pas de commission
MARTIN     1400
BLAKE      Pas de commission
CLARK      Pas de commission
SCOTT      Pas de commission
KING       Pas de commission
…
14 ligne(s) sélectionnée(s).
```

La fonction NVL2

- Syntaxe : NVL2(expr1,expr2,expr3)
- Fonction : Afficher expr2 si expr1 est non nulle, sinon afficher expr3
- Exemple

```
SQL>        SELECT ename, sal, comm, NVL2(comm,'SAL+COMM', 'SAL')
  2         FROM   emp;

ENAME             SAL       COMM NVL2(COM
---------- ---------- ---------- --------
SMITH             800       3000 SAL+COMM
ALLEN            1600        500 SAL+COMM
WARD             1250        500 SAL+COMM
JONES            2975            SAL
MARTIN           1250       1400 SAL+COMM
BLAKE            2850            SAL
…
14 ligne(s) sélectionnée(s).
```

La fonction NULLIF

- Syntaxe : NULLIF(expr1,expr2)
- Fonction : Afficher NULL si expr1=expr2, sinon afficher expr1
- Exemple

```
SQL> SELECT        ename, LENGTH(ename), job, LENGTH(job),
  2                NULLIF (LENGTH(ename),LENGTH(job)) RESULTAT
  3  FROM          emp;

ENAME      LENGTH(ENAME) JOB       LENGTH(JOB)   RESULTAT
---------- ------------- --------- ----------- ----------
SMITH                5 CLERK               5
ALLEN                5 SALESMAN            8          5
WARD                 4 SALESMAN            8          4
JONES                5 MANAGER             7          5
MARTIN               6 SALESMAN            8          6
BLAKE                5 MANAGER             7          5
…
14 ligne(s) sélectionnée(s).
```

Les expressions conditionnelles

L'expression CASE

CASE expr **WHEN** *comparison_expr1* **THEN** *return_expr1*
[**WHEN** *comparison_expr2* **THEN** *return_expr2*
WHEN *comparison_expr3* **THEN** *return_expr3*
ELSE *else_expr*]
END

```
SQL>        SELECT        ename, job, sal,
  2                       CASE job
  3                       WHEN 'CLERK' THEN 1.10*sal
  4                       WHEN'PRESIDENT' THEN 1.20*sal
  5                       ELSE sal
  6                       END "Nouveau salaire"
  7         FROM          emp ;

ENAME      JOB                SAL Nouveau salaire
---------- --------- ---------- ---------------
SMITH      CLERK              800             880
ALLEN      SALESMAN          1600            1600
WARD       SALESMAN          1250            1250
JONES      MANAGER           2975            2975
MARTIN     SALESMAN          1250            1250
BLAKE      MANAGER           2850            2850
…
14 ligne(s) sélectionnée(s).
```

L'expression DECODE

DECODE *(column_name | expr , search1*
[, search2, result2,]
[search3, result3,]
[…,…]
[résultat par default])

```
SQL>        SELECT        ename, job, sal,
  2                       DECODE (job,'CLERK',1.10*sal,
  3                       'PRESIDENT',1.20*sal,
  4                       sal)"Nouveau salaire"
  6         FROM          emp;

ENAME      JOB                SAL Nouveau salaire
---------- --------- ---------- ---------------
SMITH      CLERK              800             880
ALLEN      SALESMAN          1600            1600
WARD       SALESMAN          1250            1250
JONES      MANAGER           2975            2975
MARTIN     SALESMAN          1250            1250
BLAKE      MANAGER           2850            2850
…
14 ligne(s) sélectionnée(s).
```

Les opérateurs de comparaison

Opérateurs de comparaison

- =, <, >, <=, >=, <>,
- BETWEEN
- IS NULL
- LIKE

Opérateurs de comparaison (=)

- SELECT *
  FROM T_Customer
  WHERE customer_name = 'IMPRESSIONS';

- SELECT *
  FROM T_Customer
  WHERE customer_id = 2;

Opérateurs de comparaison (>, <)

- SELECT *
  FROM COMMANDE
  WHERE MONTANT < 500;

- SELECT *
  FROM T_Customer
  WHERE customer_name < 'M';

Opérateurs de comparaison (>=, <=)

- SELECT *
  FROM COMMANDE
  WHERE MONTANT >= 500;

- SELECT *
  FROM T_Customer
  WHERE customer_name <= 'M';

Opérateurs de comparaison (<>)

- SELECT *
  FROM COMMANDE
  WHERE NO_VENDEUR <> 9;

- SELECT *
  FROM T_Customer
  WHERE customer_name <> 'IMPRESSIONS';

Opérateurs de comparaison (BETWEEN)

- BETWEEN : Intervalle fermé
- Exemple:

```
SELECT *
FROM COMMANDE
WHERE MONTANT BETWEEN 100 AND 200;
```

MONTANT appartient à l'intervalle [100, 200]

Opérateurs de comparaison (NOT BETWEEN)

- NOT BETWEEN : Intervalle ouvert
- Exemple:

```
SELECT *
FROM COMMANDE
WHERE MONTANT NOT BETWEEN 100 AND 200;
```

MONTANT appartient à l'intervalle $]-\infty, 100[\cup]200, +\infty[$

Opérateurs de comparaison (IS NULL)

- L'opérateur teste si l'attribut a la valeur NULL, c'est à dire qu'aucune valeur n'a été fournie.
- Exemple:

```
SELECT COUNT(product_id)
FROM T_Product
WHERE product_description IS NULL;
```

Opérateurs de comparaison (LIKE)

- L'opérateur LIKE utilise les caractères spéciaux % et _
- Le caractère % remplace n'importe quel nombre de n'importe quels caractères
  - Il remplace 0 à n caractères
- Le caractère _ remplace n'importe quel caractère
  - Il remplace 1 seul caractère

```
SELECT product_id, product_name
FROM T_Product
WHERE product_name LIKE '%Desk%'
```

```
product_id product_name
---------- -------------
3          Computer Desk
5          Writer's Desk
6          8-Drawer Desk
8          Computer Desk
```

Les vues

Les vues

- Definitions
  - Une vue est une table virtuelle
  - elle n'existe pas dans la base
  - elle est construite à partir du résultat d'un SELECT.
  - La vue sera vu par l'utilisateur comme une table réelle.
- Les vues permettent
  - des accès simplifiés aux données
  - l'indépendance des données
  - la confidentialité des données : restreint les droits d'accès à certaines colonnes ou à certains n-uplets.

Les vues (2)

- Création d'une vue : CREATE VIEW
- La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue
- Syntaxe

```
create view nom [(col1, ...) ] as
select col1, col2, ...
from tab
where prédicat
[with check option]
```

- La spécification des noms de colonnes de la vue est facultative
- Par défaut, les colonnes de la vue ont pour nom les noms des colonnes résultat de SELECT

Les vues (3)

- Création de vues à partir de plusieurs tables
- Exemple
  - créer une vue comportant le nom des employés, le nom du service et le lieu de travail.
  - create view EMPLOYES2
    as select ENAME, DNAME, LOC
    from EMP emp, DEPT dept
    where emp.DEPTNO = dept.DEPTNO
- Requêtes et vues
  - Pour récupérer les données de vues, on procédera comme si l'on était en face d'une table classique
  - select * from EMPLOYES2 …
  - En réalité, cette table est virtuelle et est reconstruite à chaque appel de la vue EMPLOYES2 par exécution du SELECT constituant la définition de la vue

Les vues (4)

- Suppression d'une vue
  - Une vue peut être détruite par la commande drop view nom-vue
- Renommer une vue
  - Une vue peut être renommée par la commande

    rename ancien-nom-vue to nouveau-nom-vue

Les séquences

Séquence

- Une séquence est un objet créé par l'utilisateur
- Elle sert à créer des valeurs pour les clés primaires, qui sont incrémentées ou décrémentées par le serveur Oracle
- Noter que la séquence est stockée et générée indépendamment de la table, et une séquence peut être utilisée pour plusieurs tables

Séquence : création

```
CREATE SEQUENCE      nomsequence
        [INCREMENT BY n]
        [START WITH n]
        [{MAXVALUE n}]
        [{MINVALUE n}]
        [{CYCLE | NOCYCLE}]
        [{CACHE n | NOCACHE}];
```

Séquence (3)

- Exemple

```
SQL> CREATE SEQUENCE dept deptno seq
  2     INCREMENT BY 10
  3     START WITH 10
  4     NOCYCLE
  5     NOCACHE;

Séquence crée.
```

```
SQL> SELECT       dept_deptno_seq.NEXTVAL
  2  FROM         dual;

     NEXTVAL
     ----------
           50

SQL> SELECT       dept_deptno_seq.CURRVAL
  2  FROM         dual;

     CURRVAL
     ----------
           50
```

```
SQL> INSERT INTO  dept
  2               (deptno, dname, loc)
  3  VALUES       (dept_deptno_seq.NEXTVAL,'SUPPORT','NY');

1 ligne créée.
```

Séquence : modification

ALTER SEQUENCE *nomsequence*
[INCREMENT BY n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYLE | NOCYCLE]
[CACHE n | NOCACHE]

Séquence : suppression

DROP SEQUENCE *nomsequence;*

Exemple:

```
SQL> DROP SEQUENCE dept deptno seq;

Séquence supprimée.
```

Index

- Un index est un objet qui peut augmenter la vitesse de récupération des lignes en utilisant les pointeurs
- Les index peuvent être créés automatiquement par le serveur Oracle ou manuellement par l'utilisateur
- Ils sont indépendants, donc lorsque vous supprimez ou modifiez un index les tables ne sont pas affectées

Index (2)

CREATE INDEX *nomindex*
ON *table (column1, column2 ...);*

```
SQL> CREATE INDEX        emp_ename_idx
  2  ON                  emp(ename);

Index crée.
```

Index (3)

- Quand créer un index
  - La colonne contient une large plage de valeurs
  - La colonne contient plusieurs valeurs nulles
  - Une ou plusieurs colonnes sont fréquemment utilisées dans la clause WHERE ou pour les conditions de jointure
  - La table est grande et la plupart des requêtes recherchent moins de 2-4% des lignes
- Quand ne pas créer des index
  - La table est petite
  - Les colonnes ne sont pas souvent utilisées
  - Les requêtes recherchent plus de 2-4% des lignes
  - La table est mise à jour fréquemment
  - Les colonnes indexées sont référencées comme une partie de l'expression

Synonyme

- Un synonyme est un nom alternatif pour désigner un objet de la base de données.
- C'est aussi un objet de la base de données

Synonyme (2)

CREATE [PUBLIC] SYNONYM *nomsynonym*
FOR *object;*

PUBLIC:	Spécifie que le synonyme peut être accédé par tous les utilisateurs.
Object:	Le nom de l'objet pour lequel le synonyme est crée.

Synonyme (3)

```
SQL> CREATE SYNONYM      emplo
  2  FOR                 emp;

Synonyme crée.
```

```
SQL> SELECT        ename
  2  FROM          emplo;

     ENAME
     ----------
     SMITH
     ALLEN
     WARD
     JONES
     MARTIN
     BLAKE
     CLARK
     TURNER
     ADAMS
     ...
   15 lignes sélectionnées.
```

Chapitre 4

PL/SQL et les Triggers

Partie I

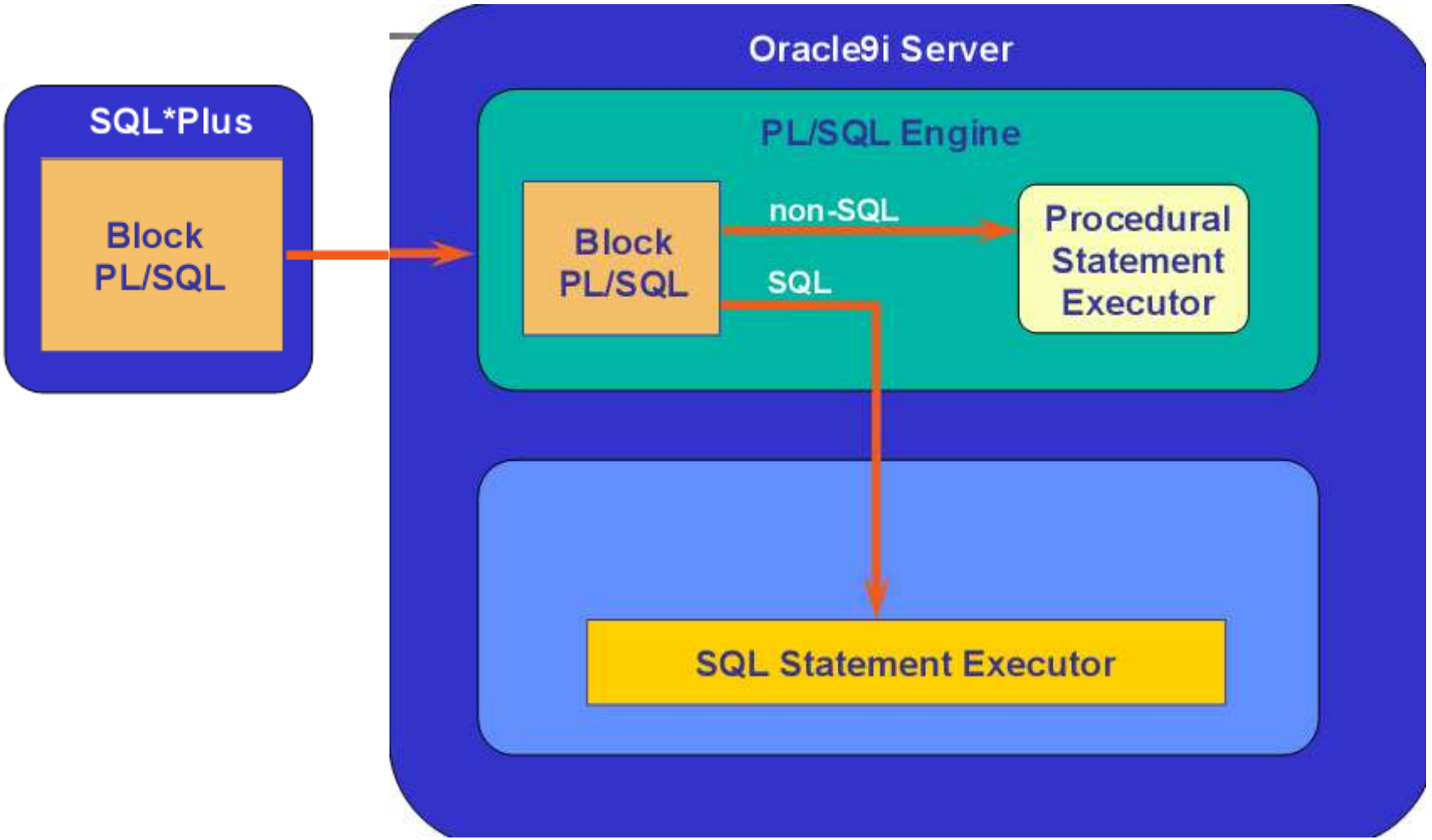
Le langage PL/SQL
Exploitation des requêtes SQL

Le langage PL/SQL

Introduction

- PL/SQL : ***P****rocedural* ***L****anguage with* ***SQL***
- C'est un langage de programmation propriétaire (Oracle) inspiré de ADA.
- Ce langage a fortement inspiré de la norme SQL3 car il est bien adapté à la manipulation d'une base de donnée relationnelle
  - il utilise les même types que SQL
  - il permet d'intégrer facilement des requêtes dans le code
  - il permet de définir des curseurs pour parcourir séquentiellement le résultat d'une requête

Moteur PL/SQL dans Oracle



Bloc PL/SQL

```
[DECLARE

# déclarations et initialisation]

BEGIN

# instructions exécutables

[EXCEPTION

# interception des erreurs]

END;
```

Structures de contrôle

```
if condition then instr
   {elsif condition then instr}
   [else  instr]
end if ;
```

```
case variable
        {when expression then instr}
        [else  instr]
end case;
```

Les Structures itératives

```
for i in [reverse] deb .. fin loop
   instr
end loop;
```

```
while condition loop
        instr
end loop;
```

```
loop
        instr
        exit when condition;
        instr
end loop;
```

Exemple : IF-THEN-ELSE

Définir une fonction factorielle :

```
CREATE OR REPLACE FUNCTION FAC (n POSITIVE)
RETURN INTEGER IS
BEGIN
   IF n = 1 THEN
      RETURN 1;
   ELSE
      RETURN n * FAC(n-1);
   END IF;
END FAC;
/
```

appel

```
SQL> SELECT fac(5) FROM DUAL;
   FAC(5)
---------
       120
```

Règles syntaxiques

- Les instructions peuvent être écrites sur plusieurs lignes
- Les identifiants peuvent contenir jusqu'à 30 caractères et doivent :
  - Être encadrés de guillemets s'ils contiennent un mot réservé
  - Commencer par une lettre
  - Avoir un nom distinct de celui d'une table ou d'une colonne de la base

Gestion PL/SQL

- Déclarer et initialiser les variables dans la section déclarative
- Assigner une valeur à une variable
- Transmettre des valeurs dans les blocs PL/SQL à l'aide de paramètres
- Visualiser les résultats à la console ou dans un fichier :

  DBMS_OUTPUT.PUT_LINE

  UTL_FILE

La syntaxe des variables

identificateur [CONSTANT] TypeDeDonnées
[NOT NULL] [(:= | DEFAULT) expression];

Exemples

num NUMBER(4) ;

num2 NUMBER NOT NULL := 3.5 ;

en_stock BOOLEAN := false ;

limite CONSTANT REAL := 5000.00 ;

La syntaxe des variables (suite)

- La contrainte NOT NULL doit être suivie d'une clause d'initialisation

- Les déclarations multiples ne sont pas permises. Donc, on ne peut pas écrire :

```
v1 , v2 NUMBER ;
```

Exemple de programme PL/SQL

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     nom VARCHAR2(15)  := 'SCOTT';
  3  BEGIN
  4     DBMS_OUTPUT.PUT_LINE(nom);
  5  END;
  6  /
SCOTT

PL/SQL procedure successfully completed.
```

Déclaration d'un type composé

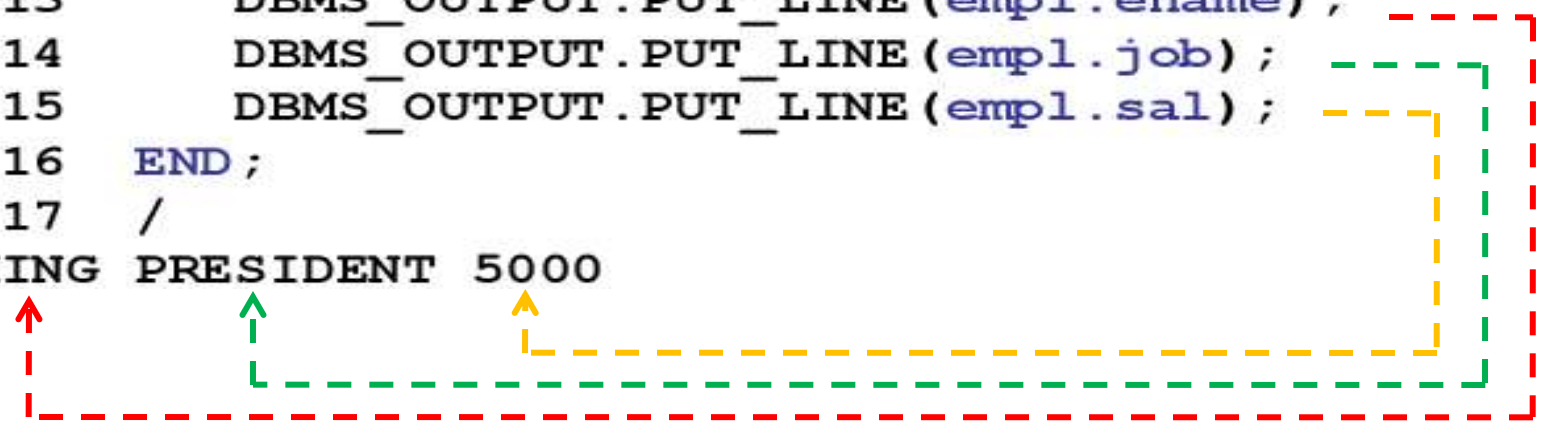
- Un type composé est
  - un type "record" : lorsque tous les attributs sont d'un type SQL
    - une variable de type record peut représenter une ligne d'une table relationnelle
  - un type collection : TABLE, VARRAY utilisés en relationnel objet
  - un type objet : modèle relationnel objet

Le type RECORD

- Peut contenir un ou plusieurs champs de type scalaire, RECORD ou TABLE
- Similaire à la structure d'enregistrement utilisée dans les L3G
- Traite un ensemble de champs comme une unité logique
- Pratique pour récupérer et traiter les données d'une table

Le type RECORD : Exemple

```
SQL> DECLARE
  2      TYPE employe IS RECORD
  3          ( ename      VARCHAR2( 25 ),
  4            job        VARCHAR2( 25 ),
  5            sal        NUMBER( 7,2 )
  6          );
  7      empl employe;
  8  BEGIN
  9      SELECT ename, job, sal
 10            INTO empl
 11            FROM emp
 12            WHERE empno = 7839;
 13      DBMS_OUTPUT.PUT_LINE(empl.ename);
 14      DBMS_OUTPUT.PUT_LINE(empl.job);
 15      DBMS_OUTPUT.PUT_LINE(empl.sal);
 16  END;
 17  /
KING PRESIDENT 5000
```



L'attribut %TYPE

- Exemple:

```
DECLARE
        nom             scott.emp.ename%TYPE;
        job             emp.job%TYPE;
        balance         NUMBER( 7, 2 );
        min_balance     balance%TYPE := 10;
```

L'attribut %ROWTYPE

- Déclarer une variable à partir d'un ensemble de colonnes d'une table ou d'une vue
- Préfixer %ROWTYPE avec la nom de la table de la base de données
- Les champs dans le RECORD ont les mêmes noms et les mêmes types de données que les colonnes de la table ou de la vue associées

Exploitation des requêtes SQL

Exploitation des requêtes SQL

- Les instructions INSERT, DELETE, UPDATE s'écrivent telles quelles dans un programme
  - Elles peuvent utiliser les variables du programme
  - Il faut donc donner des noms différents aux variables du programme et aux colonnes des tables manipulées par le programme
- Pour une requête dont le résultat est constitué d'une unique ligne, on peut utiliser la syntaxe SELECT ... INTO....
- Pour une requête qui ramène un nombre quelconque de lignes, il faut utiliser **un curseur**

Exemple

```
create or replace procedure ajouterEmp(
      leNom.employee.emp_name%type,
      lePrenom.employee.emp_firstname%type) is

     nouveauNum NUMBER ;
begin

select nvl(max(emp_no),0)+1 into nouveauNum  from
employee ;

insert into employee(emp_no, emp_name, emp_firstname)
values(nouveauNum, leNom, lePrenom);

end ajouterEmp;
```

SELECT … INTO

```
select emp_name, emp_firstname into le_nom, le_prenom
from Employee where emp_no = 346;
```

Ou bien

```
select * into emp_rec
from Employee where emp_no = 346;
```

NB : **emp_rec de type Employee%rowtype**

Les Curseurs

- Un curseur est une structure de données séquentielle avec une position courante
- On utilise un curseur pour parcourir le résultat d'une requête SQL dans un programme PL/SQL.

- On déclare un curseur en associant un nom à une requête :

```
CURSOR nom_curseur IS une_requête;
```

Les Curseurs (2)

- On peut définir des paramètres en entrée utilisés dans la requête

```
CURSOR nom_curseur(p1, ..., pn) IS une_requête ;
```

Exemple

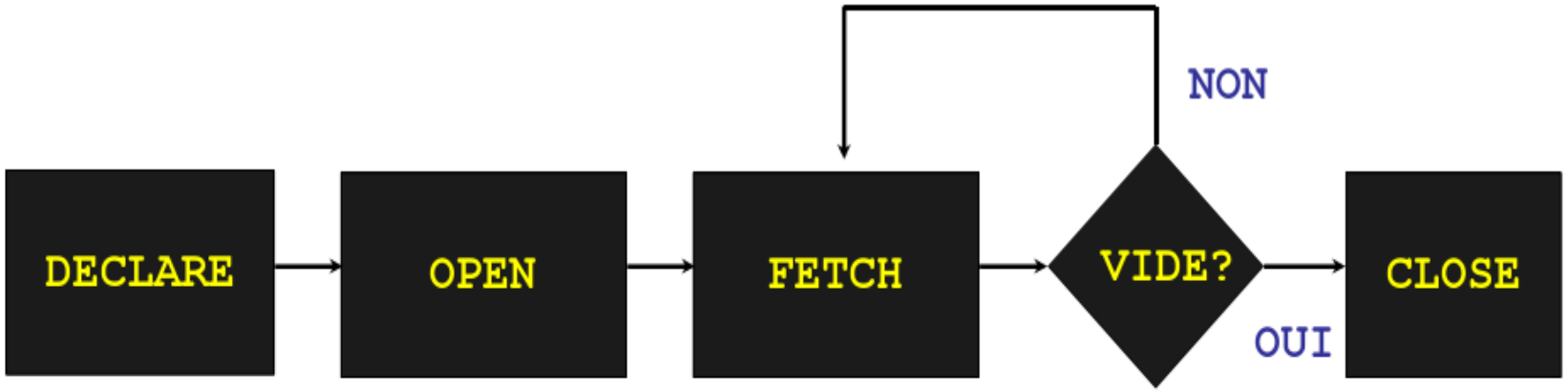
```
cursor emp_cursor(dnum NUMBER) is select salary, comm
    from Employee where deptno = dnum ;
```

Si c est un curseur, la ligne courante est de type c%rowtype, c'est à dire du type de la ligne de la requête associée à c.

Les Curseurs (3)

- Deux types de curseur :
  - Curseurs implicites : déclarés pour toutes les instructions LMD et les SELECT en PL/SQL
  - Curseurs Explicites : déclarés et nommés au sein du code source (par le développeur)

Les Curseurs explicites



Créer une zone nommée SQL

Ouvrir & exécuter

Extraire la ligne courante dans des variables déclarées dans le bloc

Test pour l'existence de lignes

Reprise de la phase d'extraction

Fermer

Les Curseurs explicites (2)

- Instructions :
  - OPEN : initialise le curseur
  - FETCH : extrait la ligne courante et passe à la suivante (pas d'exception si plus de ligne)
  - CLOSE : invalide le curseur
  - Si on veut parcourir toutes les lignes : boucle FOR
- Attributs du curseur :
  - %found vrai si le dernier fetch a ramené une ligne
  - %notfound vrai si le dernier fetch n'a pas ramené de ligne
  - %isopen vrai ssi le curseur est ouvert
  - %rowcount le nombre de lignes déjà ramenées

Les Curseurs explicites : Exemple 1

```
declare
      cursor   c_proj   is
          select proj_name, proj_budget
             from project  order by proj_budget ;

        proj_rec    c_proj%rowtype ;
 begin
      open c_proj ;
      loop -- parcours des lignes du curseur
         fetch  c_proj   into   proj_rec ;
         exit when   c_proj%notfound ;
       ... on utilise la ligne courante rangée dans proj_rec ...
      end loop ;
     close   c_proj ;
 end ;
```

Les Curseurs explicites : Exemple 2

```
declare

 cursor    c_proj     is
       select proj_name, proj_budget
       from project    order by proj_budget ;

begin

     for proj_rec in c_proj loop -- parcours des lignes du curseur

      ... on utilise la ligne courante rangée dans proj_rec ...

   end loop ;
 end ;
```

Les Curseurs implicites

- Quelque fois, PL/SQL déclare implicitement un curseur :
  - pour les instructions du DML qui modifient la base (INSERT, DELETE, UPDATE)
  - pour les requêtes de la forme SELECT INTO.
- Avec un curseur implicite, on peut obtenir des informations sur la requête réalisée, grâce aux attributs.
  - En effet SQL%attribut applique l'attribut sur la dernière requête SQL exécutée

Les Curseurs implicites : Exemple

```
delete from Employee where ...

    if SQL%rowcount > 10 then
      -- on a supprimé plus de 10 lignes
       ...
    end if ;
```

Partie II

Les exceptions
Les modules stockés

Introduction

- En PL/SQL, la gestion des erreurs se fait grâce aux **exceptions**
- Le mécanisme de déclenchement / traitement d'exceptions est identique à celui du langage ADA
- Il existe un certain nombre d'exceptions prédéfinies mais le programmeur peut bien sûr définir ses propres exceptions

Syntaxe

```
BEGIN
   ...corps du bloc...
EXCEPTION
   when exception1 [or exception2 ...] then
        instructions ;
   when exception3 [or exception4 ...] then
        instructions ;
        ...
   [when others then instructions ;]
END;
```

Exécution des exceptions

- Si l'exception existe dans une clause When, alors les instructions de cette clause sont exécutées et le programme est terminé
- Si l'exception n'existe pas dans une clause When :
  - Soit il existe une clause When Others et dans ce cas les instructions de cette clause sont exécutées et le programme est terminé
  - Soit il n'y a pas de clause When Others et l'exception est propagée au bloc englobant ou au programme appelant

Exemple

```
DECLARE
    pe_ratio NUMBER(3,1);
BEGIN
    SELECT     prix / gains       INTO  pe_ratio
    FROM stocks WHERE symbol = 'XYZ';
 -- pourrait provoquer une erreur de division par zéro

INSERT INTO stats (symbol, ratio) VALUES ('XYZ', pe_ratio);

EXCEPTION
WHEN ZERO_DIVIDE THEN
INSERT INTO stats (symbol, ratio) VALUES ('XYZ', NULL);
END;
```

Exceptions prédéfinies

- **TOO_MANY_ROWS** : instruction select ... into qui ramène plus d'une ligne.
- **NO_DATA_FOUND** : instruction select ... into qui ne ramène aucune ligne.
- **INVALID_CURSOR** : ouverture de curseur non valide.
- **CURSOR_ALREADY_OPEN** : ouverture d'un curseur déjà ouvert.
- **VALUE_ERROR** : erreur arithmétique (conversion, taille, ...) pour un NUMBER.
- **ZERO_DIVIDE** : division par 0 ;
- **STORAGE_ERROR** : dépassement de capacité mémoire.
- **LOGIN_DENIED** : connexion refusée

Déclaration des exceptions

On déclare l'exception *nomException* grâce à l'instruction suivante :

nomException **EXCEPTION** ;

Les exceptions et les codes d'erreur

- Les exceptions prédéfinies sont associées à des codes d'erreur Oracle
- Lorsqu'une exception n'est pas traitée dans le programme PL/SQL, le client qui a appelé ce programme reçoit le code d'erreur associé
- Pour lier un nom d'exception à un code d'erreur, on peut utiliser la directive :

```
PRAGMA EXCEPTION_INIT(nomException, codeErreur)
```

Exemple

```
create procedure detruitCompagnie...
    restePilote EXCEPTION ;
    PRAGMA Exception_init(restePilote, -2292);
begin
    delete from compagnie where comp = 'MaCompagnie' ;
            ...
    exception
    when restePilote then
            ... traitement de l'erreur ...
end ;
```

Exemple de programme PL/SQL

```
create procedure detruitCompagnie(ma_comp VARCHAR2(20)) is
    restePilote EXCEPTION ;
    PRAGMA Exception_init(restePilote, -2292);
    compagnieInexistante EXCEPTION ;
     -- ce n'est pas nécessaire de la lier à un code d'erreur
begin
    delete from compagnie where comp = ma_comp ;
    if SQL%NOTFOUND then
        raise compagnieInexistante ;
    end if ;
exception
    when restePilote then
        ... traitement de l'erreur ...
    when compagnieInexistante then
        ...
    when others then
        ... on traite les autres erreurs
end ;
```

Les exceptions et les codes d'erreur

- **NB**: il est aussi possible de déclencher une erreur, sans la lier à une exception :

```
raise_application_error(codeErreur,messageErreur);
```

- le code d'erreur est un entier négatif compris entre -20999 et -20000 (codes réservés aux erreurs non prédéfinies)
- le message d'erreur est celui communiqué au client (par exemple, affiché sous SQL*Plus)

Les modules stockés

Définition

- Un module stocké est un programme rangé dans la base de données
- On peut ainsi définir en PL/SQL :
  - des procédures
  - des fonctions
  - des paquetages
- Ces programmes peuvent être appelés par les programmes clients, et sont exécutés par le serveur

Procédure / Fonction

```
create or replace procedure  p_name [ les_parametres ] is
        declarations
begin
        code PL/SQL
end ;
```

```
create or replace function f_name [ les_parametres ] return datatype
is
        Declarations
begin
        code PL/SQL
end ;
```

Les paramètres

- Pour déclarer un paramètre, la syntaxe est :

  nom_param [mode] datatype [:= valeur_defaut]

- Il y a trois modes de passage de paramètre :

  mode IN : paramètre en entrée (par défaut)

  mode OUT : paramètre en sortie

  mode IN OUT : paramètre en entrée et sortie

Exemple 1

```
create or replace function nom_dept(numero
    dept.dept_no%type)
return VARCHAR2     is
    nom dept.dept_name%type ;
begin
    select dept_name into nom from dept
    where dept_no = numero ;
    return nom ;
end;
```

Exemple 2

```
create or replace procedure nom_dept2 (
    numero IN dept.dept_no%type,
    nom OUT dept.dept_name%type)     is

begin

    select dept_name into nom
    from dept
    where dept_no = numero ;

end ;
```

Les packages

- Un paquetage permet de regrouper un ensemble des procédures, exceptions, constantes...
- Un paquetage est composé de :
  - une spécification
  - un corps
- La spécification du paquetage contient des éléments que l'on rend accessibles à tous les utilisateurs du paquetage
- Le corps du paquetage contient l'implémentation et ce que l'on veut cacher

Les packages : Spécification

- La spécification contient :
  - des signatures de procédures et fonctions
  - des constantes et des variables
  - des définitions d'exceptions
  - des définitions de curseurs

Les packages : Corps

- Le corps contient :
  - Les corps des procédures et fonctions de la spécification (obligatoire)
  - D'autres procédures et fonctions (cachées)
  - Des déclarations que l'on veut rendre privées
  - Un bloc d'initialisation du paquetage si nécessaire

Les packages : Exemple

```
-- spécification
create or replace package mon_paq as
    procedure p ;
    procedure p(i NUMBER) ;
    function p(i NUMBER) return NUMBER ;
    cpt NUMBER ;
    function get_cpt return NUMBER ;
    mon_exception EXCEPTION ;
    PRAGMA EXCEPTION_INIT(mon_exception, -20101);
end ;
/
Package créé.
```

Les packages : Exemple (suite)

```
 -- corps
create or replace package body mon_paq as
      procedure p is
      begin
            dbms_output.put_line('toto');
      end ;
      procedure p(i NUMBER) is
      begin
            dbms_output.put_line(i);
      end ;
      function p(i NUMBER) return NUMBER is
      begin
            if (i > 10) then raise mon_exception ; end if ;
            return i ;
      end ;
      function get_cpt return NUMBER is
      begin     return cpt ;      end ;
end ;
/
```

Partie III

Les Triggers

Définition

- Un trigger (déclencheur) est un programme qui se déclenche **automatiquement** suite à un événement

- Il fait partie du schéma (comme les modules stockés) mais que l'on n'appelle pas explicitement, à la différence d'une procédure stockée

Syntaxe

```
CREATE [OR REPLACE] TRIGGER nom_trigger
instant liste_evts
ON nom_table  [FOR EACH ROW]
[WHEN ( condition ) ]

--Corps

instant ::= AFTER | BEFORE
liste_evts ::= evt {OR evt}
evt ::= DELETE | INSERT | UPDATE [OF { liste_cols }]
liste_col ::= nom_col { , nom_col }

--corps de pgme PL/SQL
```

Entête du trigger

On définit :

- la table à laquelle le trigger est lié,
- les instructions du DML qui déclenchent le trigger
- le moment où le trigger va se déclencher par rapport à l'instruction DML (avant ou après)
- si le trigger se déclenche
  - une seule fois pour toute l'instruction (i.e. trigger instruction),
  - une fois pour chaque ligne modifiée/insérée/supprimée. (i.e. trigger ligne, avec l'option FOR EACH ROW)
- et éventuellement une condition supplémentaire de déclenchement (clause WHEN)

After ou Before ?

- Si le trigger doit déterminer si l'instruction DML est autorisée, utiliser BEFORE
- Si le trigger doit "fabriquer" la valeur d'une colonne pour pouvoir ensuite la mettre dans la table : utiliser BEFORE
- Si on a besoin que l'instruction DML soit terminée pour exécuter le corps du trigger : utiliser AFTER

Trigger ligne ou instruction ?

- Un trigger **instruction** se déclenche **une fois**, suite à une instruction DML

- Un trigger **ligne** (FOR EACH ROW) se déclenche **pour chaque ligne modifiée** par l'instruction DML

Trigger ligne

- Dans un trigger ligne, on peut faire référence à la ligne courante, celle pour laquelle le trigger s'exécute
- Pour cette ligne, on a accès à la valeur avant l'instruction DML (nommée :old) et à la valeur après l'instruction (nommée :new)

	:old	:new
insert	null	ligne insérée
delete	ligne supprimée	null
update	ligne avant modif	ligne après modif

La clause When

On peut définir une condition pour un trigger ligne

le trigger se déclenchera pour chaque ligne vérifiant la condition

Exemple

```
Create or replace trigger journal_emp
    after update of salary on EMPLOYEE
    for each row
    when (new.salary < old.salary)
    -- attention, ici on utilise new et pas :new

begin

    insert into EMP_LOG(emp_id, date_evt, msg)
    values (:new.empno, sysdate, 'salaire diminué');

end ;
```

Ordre d'exécution des triggers

- Les triggers se déclenchent dans l'ordre suivant :
  1. Triggers instruction BEFORE
  2. Triggers ligne BEFORE (déclenchés n fois)
  3. Triggers ligne AFTER (déclenchés n fois)
  4. Triggers instruction AFTER