Université Ibn Tofaïl

Cours Base de données avancées

Professeur : Khalid HOUSNI

https://www.youtube.com/channel/UCaSjLXaBk4zc05XaXjK64Jg

ORACLE PLSQL

SMI 6

Classroom

Plan de cours

- **Ch. I. Principes de fonctionnement des (SGBD)**
  - Introduction aux bases de données
  - Modèle de base de données
  - Système de gestion de base de données (SGBD)
- **Ch. II. Développement d'une base de données (tables, contraintes, relations, vues)**
  - Création de tables, contraintes et relations
  - Création et gestion des vues
- **Ch. III. Procédures stockées et Déclencheurs(Triggers)**
  - Gestion des procédures et fonctions
  - Métadonnées des procédures et fonctions
  - Extensions au standard SQL (Délimiteur, contrôle flux, boucles, Curseurs)
  - Structure d'un déclencheur
  - Types d'événements déclencheurs
  - Propriétés d'un déclencheur
  - Validation des données
- **Ch. IV Pages web pilotées par une base de données**
  - Ouverture d'une connexion vers une base de données
  - Optimisation des requêtes (Partitionnement des tables et indexes, indexation des opérations en ligne et en parallèle, Maintenance et configuration des indexes).
  - Stockage de données capturées par des formulaires
  - Envoi de requêtes dynamiques à une base de données
  - Générer une page web affichant les résultats d'une requête

Chapitre I : Principes de fonctionnement des SGBD

- Introduction aux bases de données
- Modèle de base de données
- Système de gestion de base de données (SGBD)

Introduction aux bases de données

- Un système d'information est construit autour de volumes de données de plus en plus important. Ces données doivent être stockées sur des supports physiques. Les données sont stockées et organisées dans des bases de données – BD (databases – DB).
- Un utilisateur doit pouvoir les retrouver. Il faut pouvoir les interroger par des requêtes.
- Les données évoluent, il faut donc pouvoir les manipuler : ajouter, modifier, supprimer des données.
- Le logiciel de base qui permet de manipuler ces données est appelé un système de gestion de bases de données – SGBD (database management system – DBMS).
  Tout système d'information est construit autour de bases de données

Introduction aux bases de données

- **Base de données- Un ensemble organisé d'informations avec un objectif commun.**
- **Peu importe le support utilisé pour rassembler et stocker les données (papier, fichiers, etc.), dès lors que des données sont rassemblées et stockées d'une manière organisée dans un but spécifique, on parle de base de données.**
- **Enjeux**
  - les bases de données de demain devront être capables de gérer
    - plusieurs dizaines de téra-octets de données,
    - géographiquement distribuées à l'échelle d'Internet,
    - par plusieurs dizaines de milliers d'utilisateurs
    - dans un contexte d'exploitation changeant.

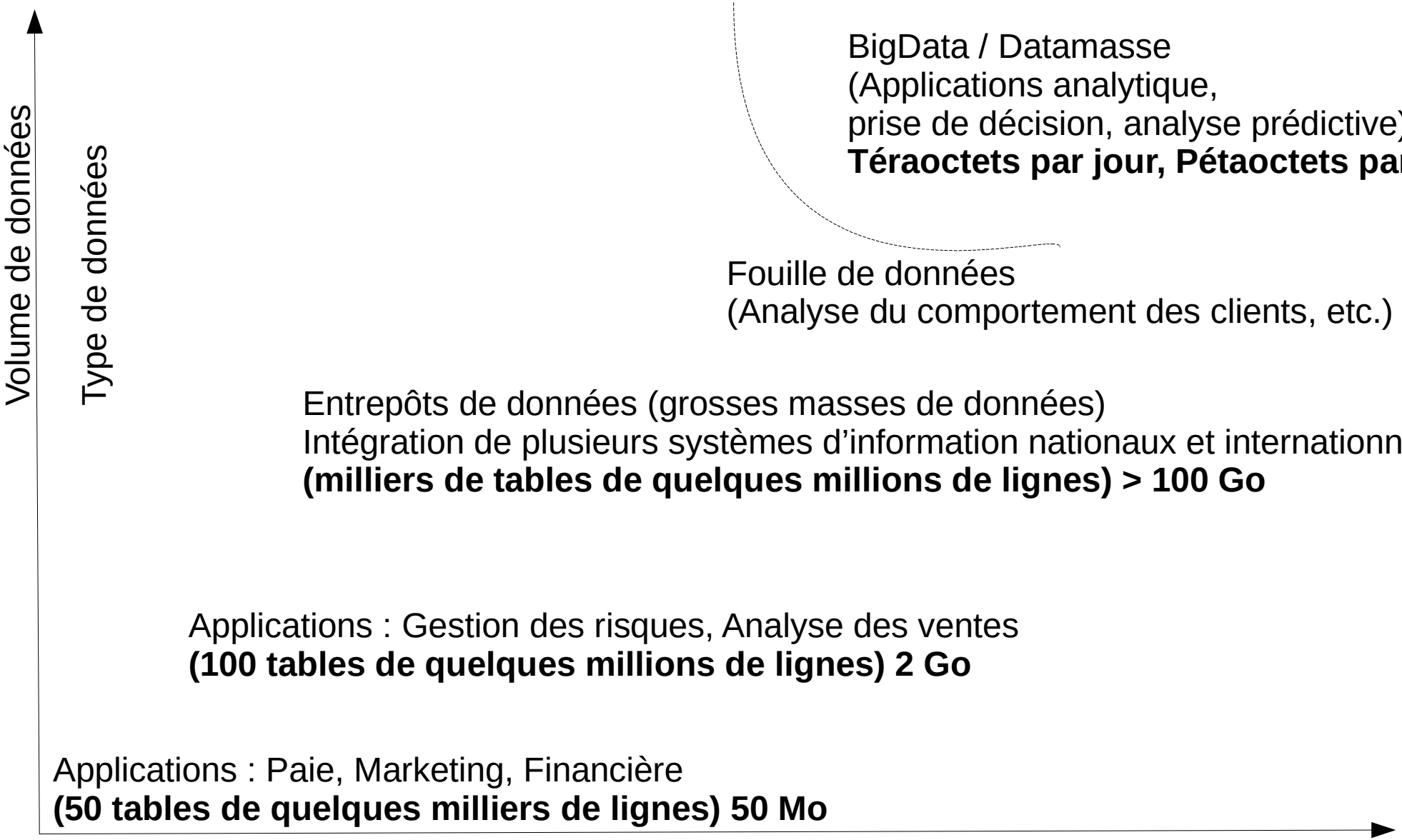
Introduction aux bases de données

- **Une base de données informatique est un ensemble de données stockées sur un support informatique, organisées et structurées de manière à pouvoir facilement consulter et modifier leur contenu.**
- **l'utilisation directe de fichiers soulève de très gros problèmes :**
  1) Lourdeur d'accès aux données. En pratique, pour chaque accès, même le plus simples, il faudrait écrire un programme.
  2) Manque de sécurité. Si tout programmeur peut accéder directement aux fichiers, il est impossible de garantir la sécurité et l'intégrité des données.
  3) Pas de contrôle de concurrence. Dans un environnement où plusieurs utilisateurs accèdent aux même fichiers, des problèmes de concurrence d'accès se posent.

Concepts de base

- **Un Système de Gestion de Base de Données (SGBD) est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données.**
- **SGBDR : Le R de SGBDR signifie "relationnel". Un SGBDR est un SGBD qui implémente la théorie relationnelle.**
- **Le langage SQL :**
  - Le SQL (Structured Query Language) est un langage informatique qui permet d'interagir avec des bases de données relationnelles.
  - Il a été créé dans les années 1970 et c'est devenu standard en 1986 (pour la norme ANSI - 1987 en ce qui concerne la norme ISO). Il est encore régulièrement amélioré.

Historique
Applications BD, ED, FD, ...

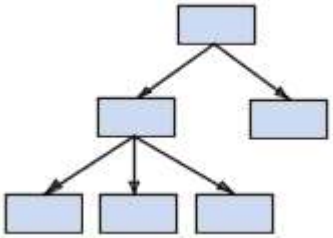


Modèles de base de données

- Les bases de données sont apparues à la fin des années 60, à une époque où la nécessité d'un système de gestion de l'information souple se faisait ressentir.
- Il existe plusieurs modèles de SGBD, différenciés selon la représentation des données qu'elle contient

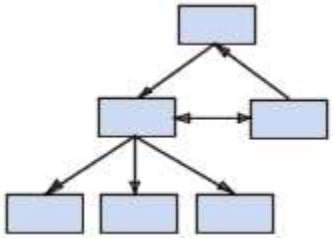
Modèles de base de données

- **le modèle hiérarchique : les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD**



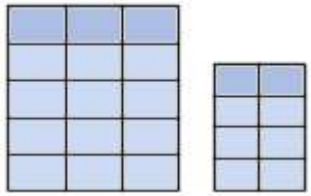
  - Limitation : problème au niveau des relations N*N

- **le modèle réseau : comme le modèle hiérarchique ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant**

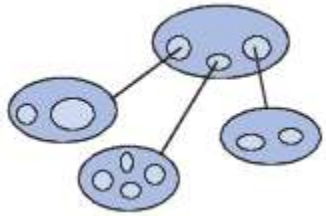


Modèles de base de données

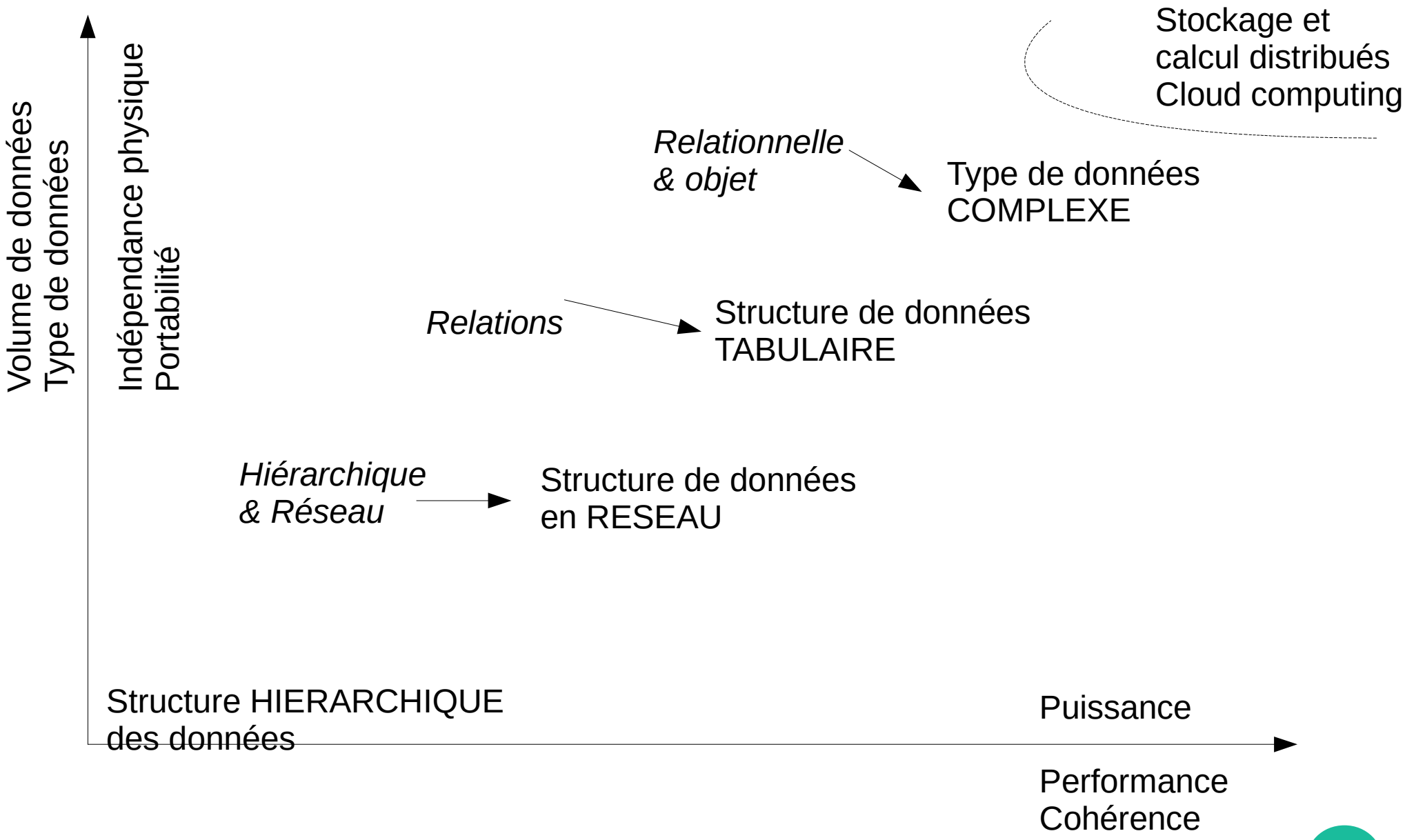
- **le modèle relationnel :** les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations



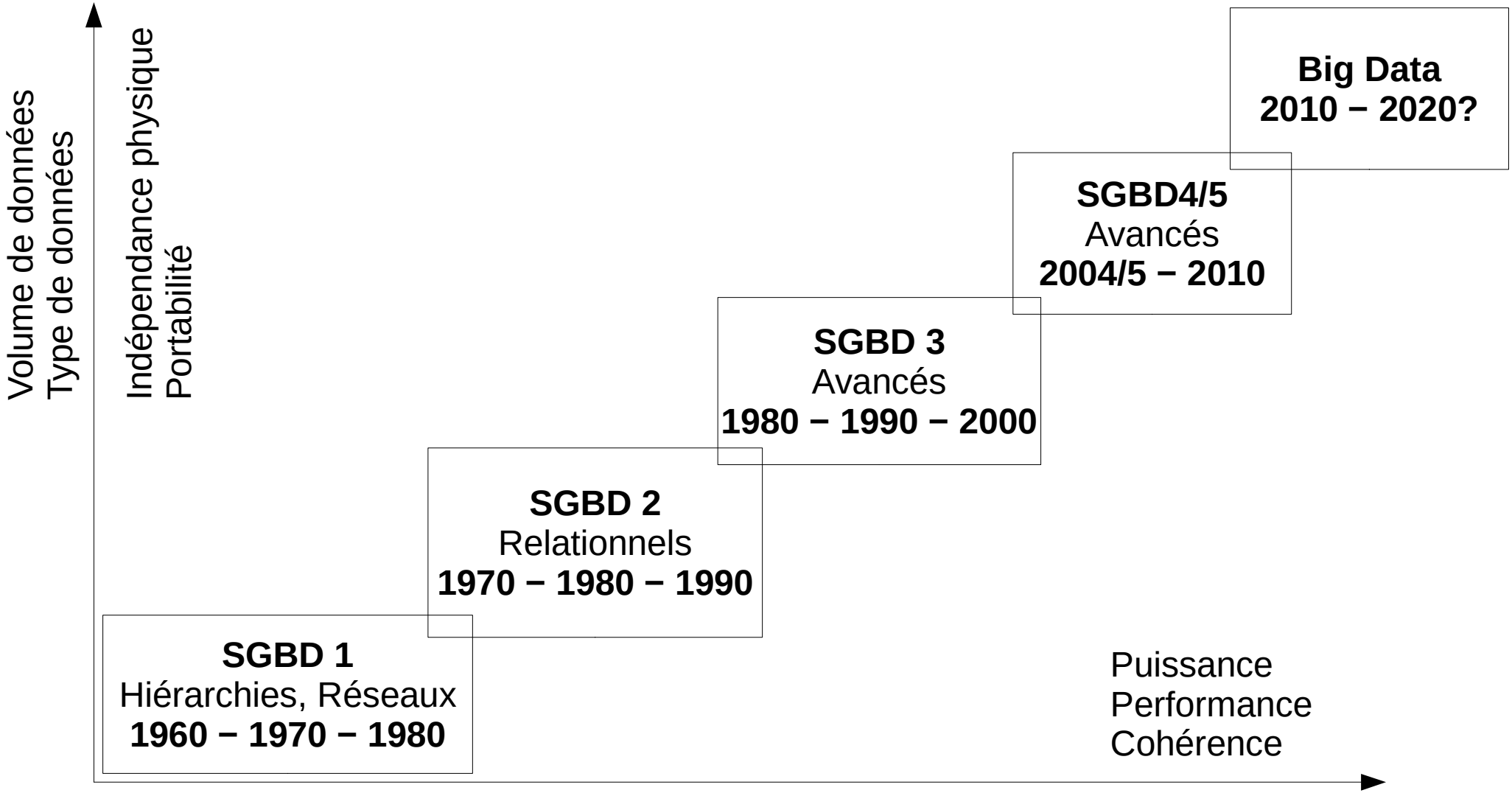
- **le modèle objet :** les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les champs sont des instances de ces classes



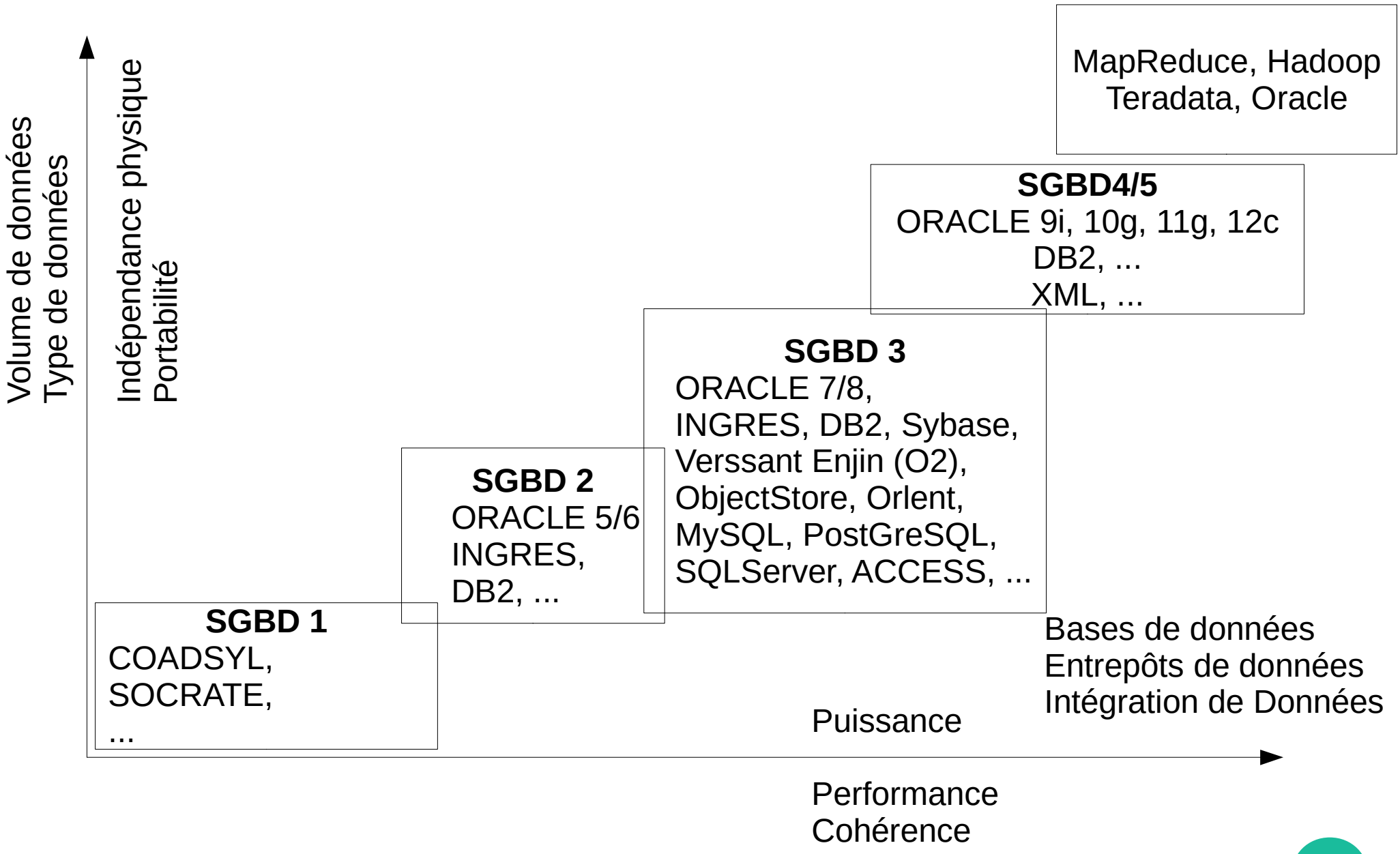
Historique
Structure et type de données



Historique
Générations de SGBD



Historique
Générations de SGBD



Objectifs des SGBD

- **Des objectifs principaux ont été fixés aux SGBD dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par le système de fichier classique.**
  - **Indépendance physique :** La façon dont les données sont définies doit être indépendante des structures de stockage utilisées.
  - **Indépendance logique :** Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrées dans une vision globale.
  - **Accès aux données :** L'accès aux données se fait par l'intermédiaire d'un Langage de Manipulation de Données (LMD).
  - **Administration centralisée des données :** Toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications.
  - **Non redondance des données**

Objectifs des SGBD

- **Cohérence des données :** Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Les contraintes d'intégrité sont décrites dans le Langage de Description de Données (LDD).
- **Partage des données**
- **Sécurité des données :** Les données doivent pouvoir être protégées contre les accès non autorisés.
- **Résistance aux pannes**

SGBD «fichier» vs «client/serveur»

- **Il existe à ce jour, deux type courant d'implémentation physique des SGBD relationnels :**
  - Ceux qui utilisent un service de fichiers associés à un protocole de réseau afin d'accéder aux données : SGBD « fichier ». ces SGBD ne proposent en général pas le contrôle des transactions, et peu fréquemment le DDL et DCL.
    - **Avantage** : **Simplicité** du fonctionnement,**coût peu élevé** voir gratuit, f**ormat des fichiers ouverts**, **administration quasi inexistante**.
    - **Inconvénient** : faible capacité de stockage, encombrement du réseau, faible nombre d'utilisateurs, droits d'accès sont gérés par le système d'exploitation ...
    - **Exemples** : Access, SQLite, MySQL

SGBD «fichier» vs «client/serveur»

- Ceux qui utilisent une application centralisée dite serveur de données : SGBD client/serveur. Ce service consiste à faire tourner sur un serveur physique, un moteur qui assure une relation indépendance entre les données et les demandes de traitement venant des différentes applications.
  - **Avantage : grande capacité de stockage**, gestion de la **concurrence** dans un SI à grand nombre d'utilisateurs, indépendance vis à vis de l'OS, gestion des **transaction** …
  - **Inconvénient : lourdeur dans le cas de solution « monoposte », coût élevé** des licences, nécessite de machines puissantes
  - **Exemples :** InterBase, Oracle, PostGreSQL, SQL Server, Sybase , ..

Exemples SGBDR

- **MySQL**
  - MySQL est un Système de Gestion de Bases de Données Relationnelles.
  - MySQL utilise le langage SQL.
  - MySQL est un logiciel Open Source. Une version gratuite de MySQL est par conséquent disponible. À noter qu'une version commerciale payante existe également.
  - Le développement de MySQL commence en 1994 par David Axmark et Michael Widenius. EN 1995, la société MySQL AB. C'est la même année où la première version officielle de MySQL est sortie
  - En 2008, MySQL AB est rachetée par la société **Sun Microsystems,** qui est elle-même rachetée par **Oracle Corporation en 2010.**

Exemples SGBDR

- **Oracle DATABASE**
  - Edité par Oracle Corporation est un SGBDR payant.
  - Son coût élevé fait qu'il est principalement utilisé par des entreprises.
  - Oracle gère très bien de grands volumes de données. Pour plusieurs centaines de Go de données Oracle sera bien plus performant.
  - Par ailleurs, Oracle dispose d'un langage procédural très puissant (du moins plus puissant que le langage procédural de MySQL) : le **PL/SQL**.

Exemples SGBDR

- **PostgreSQL**
  - Comme MySQL, **PostgreSQL est un logiciel Open Source.**
  - La première version Windows n'est apparue qu'à la sortie de la version 8.0 du logiciel, en 2005.
  - **PostgreSQL a longtemps été plus performant que MySQ**L, mais ces différences tendent à diminuer. MySQL semble être aujourd'hui équivalent à PostgreSQL en terme de performances sauf pour quelques opérations telles que l'insertion de données et la création d'index.
  - Le langage procédural utilisé par PostgreSQL s'appelle le **PL/pgSQL.**

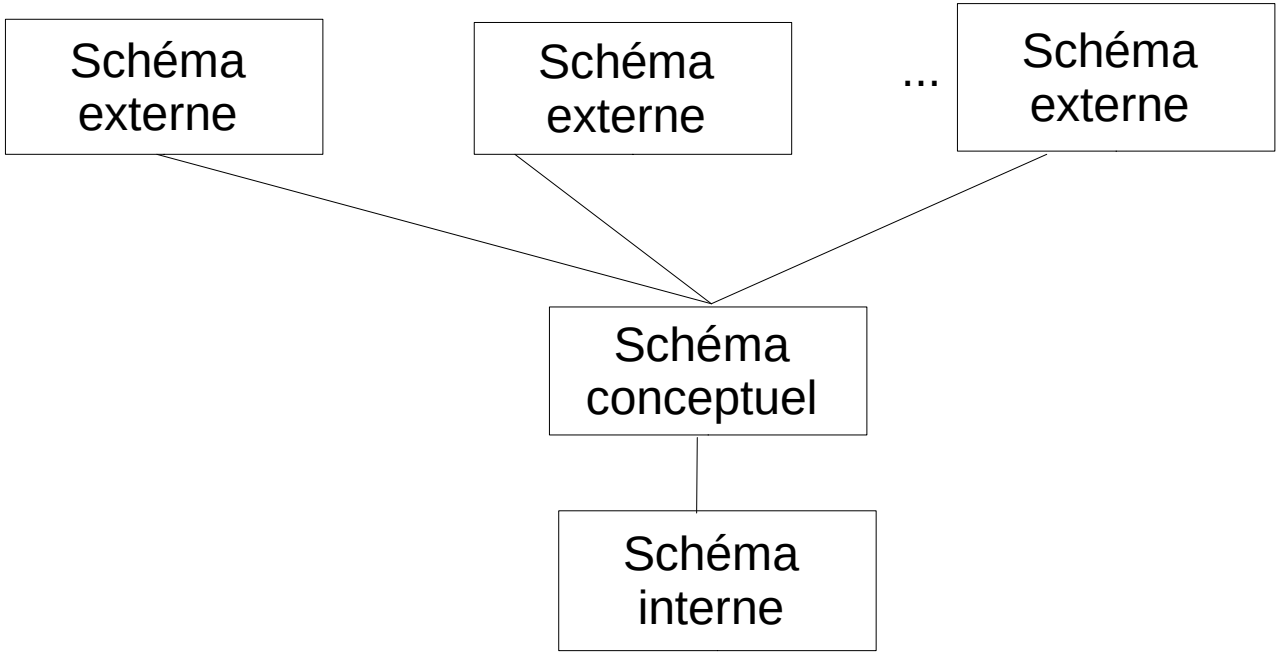
Exemples SGBDR

- **MS Access**
  - MS Access ou Microsoft Access est un logiciel édité par Microsoft
  - **c'est un logiciel payant** qui **ne fonctionne que sous Windows.**
  - **Il n'est pas du tout adapté pour gérer un grand volume** de données et a beaucoup moins de fonctionnalités que les autres SGBDR.
  - **Son avantage principa**l est l'**interface graphique** intuitive qui vient avec le logiciel.

Exemples SGBDR

- **SQLite**
  - La particularité de SQLite est de ne pas utiliser le schéma client-serveur utilisé par la majorité des SGBDR.
  - SQLite stocke toutes les données dans de simples fichiers. Par conséquent, il ne faut pas installer de serveur de base de données, ce qui n'est pas toujours possible (certains hébergeurs web ne le permettent pas).
  - Pour de très petits volumes de données, SQLite est très performant.
  - Le fait que les informations soient simplement stockées dans des fichiers rend le système difficile à sécuriser (autant au niveau des accès, qu'au niveau de la gestion de plusieurs utilisateurs utilisant la base simultanément).

Architecture à trois niveaux de description



Dans la suite de ce cours nous nous intéressons aux aux bases de données relationnelles et au SGBD Oracle version expresse

-

Ch. II. Développement d'une base de données (tables, contraintes, relations, vues)

- Méthode Merise
- Création de tables, contraintes et relations
- Création et gestion des vues

Concepts de base

Domaine : ensemble de valeurs caractérisée par un nom

Relation : sous-ensemble du produit cartésien d'une liste de domaines caractérisée par un nom

La relation est une table à deux dimensions. Les colonnes sont les attributs et lignes sont les tuples.

Attribut : colonne au niveau d'une relation caractérisé par un nom. Un attribut possède un nom, un type et un domaine.

Tuple : Ligne d'une relation appelée aussi enregistrement.

Schéma de la relation : nom de la relation suivi de la liste des attributs et de la définition de leurs domaines

Clé : Ensemble minimal d'attributs dont la connaissance des valeurs permet d'identifier un tuple unique de la relation considérée.

Valeur NULL : valeur conventionnelle dans une relation pour représenter une information inconnue

Méthode MERISE

- **MERISE est une méthode d'analyse et de conception des SI basée sur le principe de la séparation des données et des traitements.**
- **Elle possède un certain nombre de modèles (ou schémas) qui sont répartis sur 3 niveaux :**
  - Le niveau conceptuel,
  - Le niveau logique ou organisationnel,
  - Le niveau physique.

Merise – Niveau conceptuel

- **Le modèle conceptuel des données (MCD) est une représentation graphique et structurée des informations mémorisées par un SI.**
- **Le MCD est basé sur deux notions principales :**
  - les entités
  - et les associations,

  d'où sa seconde appellation : le schéma Entité/Association.
- **L'élaboration du MCD passe par les étapes suivantes :**
  - La mise en place de règles de gestion (besoins des futurs utilisateurs),
  - L'élaboration du dictionnaire des données,
  - La recherche des dépendances fonctionnelles entre ces données,
  - L'élaboration du MCD (création des **entités** puis des **associations** puis ajout des **cardinalités**).

Merise – Niveau conceptuel : Les règles de gestion

- **Les besoins des futurs utilisateurs de votre application.**
- **Et à partir de ces besoins, vous devez être en mesure d'établir les règles de gestion des données à conserver.**
- **Exemple : le SI d'une bibliothèque. Règles de gestion :**
  - Pour chaque **livre**, on doit connaître le *titre*, l'année de *parution*, un *résumé* et le *type* (roman, poésie, science fiction, ...).
  - Un livre peut être rédigé par aucun (dans le cas d'une œuvre anonyme), un ou plusieurs **auteurs** dont on connaît le *nom*, le *prénom*, la *date de naissance* et le *pays* d'origine.
  - Chaque **exemplaire** d'un livre est identifié par une *référence* composée de lettres et de chiffres et ne peut être paru que *dans une et une seule* **édition**.
  - Un **inscrit** est identifié par un *numéro* et on doit mémoriser son *nom*, *prénom*, *adresse*, *téléphone* et *adresse e-mail*.
  - Un inscrit peut faire zéro, un ou plusieurs **emprunts** qui concernent *chacun un et un seul* exemplaire. Pour chaque emprunt, on connaît la *date* et le *délai* accordé (en nombre de jours).

Merise – Niveau conceptuel :Le dictionnaire des données

- **Le dictionnaire des données est un document qui regroupe toutes les données que vous aurez à conserver dans votre base (et qui figureront donc dans le MCD).**
- **Pour chaque donnée, il indique :**
  - Le code mnémonique : il s'agit d'un libellé désignant une donnée (par exemple «titre_l» pour le titre d'un livre)
  - La désignation : il s'agit d'une mention décrivant ce à quoi la donnée correspond (par exemple «titre du livre»)
  - Le type de donnée :
    - A ou Alphabétique : lorsque la donnée est uniquement composée de caractères alphabétiques (de 'A' à 'Z' et de 'a' à 'z')
    - N ou Numérique : lorsque la donnée est composée uniquement de nombres
    - AN ou Alphanumérique : alphabétiques et numériques
    - Date : lorsque la donnée est une date (au format AAAA-MM-JJ)
    - Booléen : Vrai ou Faux
  - La taille : elle s'exprime en nombre de caractères ou de chiffres.
  - Et parfois des remarques ou observations complémentaires

Merise – Niveau conceptuel :Le dictionnaire des données

Code mnémonique	Désignation	Type	Taille	Remarque
id_i	Identifiant numérique d'un inscrit	N		
nom_i	Nom d'un inscrit	A	30	
prenom_i	Prénom d'un inscrit	A	30	
rue_i	Rue où habite un inscrit	AN	50	
ville_i	Ville où habite un inscrit	A	50	
cp_i	Code postal d'un inscrit	AN	5	
tel_i	Numéro de téléphone fixe d'un inscrit	AN	15	
tel_port_i	Numéro de téléphone portable d'un inscrit	AN	15	
email_i	Adresse e-mail d'un inscrit	AN	100	
date_naissance_i	Date de naissance d'un inscrit	Date	10	Au format AAAA-JJ-MM
id_l	Identifiant numérique d'un livre	N		
titre_l	Titre d'un livre	AN	50	
annee_l	Année de parution d'un livre	N	4	
resume_l	Résumé d'un livre	AN	1000	
ref_e	Code de référence d'un exemplaire d'un livre	AN	15	Cette référence servira également d'identifiant dans ce système
id_t	Identifiant numérique d'un type de livre	N		
libelle_t	Libellé d'un type de livre	AN	30	
id_ed	Identifiant numérique d'une édition de livre	N	6	
nom_ed	Nom d'une édition de livre	AN	30	
id_a	Identifiant numérique d'un auteur	N		
nom_a	Nom d'un auteur	A	30	
prenom_a	Prénom d'un auteur	A	30	
date_naissance_a	Date de naissance d'un auteur	Date		Au format AAAA-JJ-MM
id_p	Identifiant numérique d'un pays	N		
nom_p	Nom d'un pays	A	50	
id_em	Identifiant numérique d'un emprunt	N		
date_em	Date de l'emprunt	Date		Au format AAAA-JJ-MM
delais_em	Délai autorisé lors de l'emprunt du livre	N	3	S'exprime en nombre de jours

Merise – Niveau conceptuel :Les dépendances fonctionnelles

- Soit deux propriétés (ou données) P1 et P2.
- On dit que P1 et P2 sont reliées par une dépendance fonctionnelle (DF) si et seulement si une occurrence (ou valeur) de P1 permet de connaître une et une seule occurrence de P2.
- Cette dépendance est représentée comme ceci : P1 → P2
- On dit que P1 est la source de la DF et que P2 en est le but.
- Par ailleurs, plusieurs données peuvent être source comme plusieurs données peuvent être but d'une DF. Exemples :
  - P1,P2 → P3
  - P1 → P2,P3
  - P1, P2 → P3,P4,P5

Merise – Niveau conceptuel :Les dépendances fonctionnelles

- **Définition**
  Soit R(P1, P2, P3, P4, ... Pn) une relation
  Soit U={P1, P2, P3, P4, ... Pn} l'ensemble des attributs de R
  Soit X, Y $\subseteq$U, i, e X et Y sont deux attributs ou ensemble d'attributs de R.
  On dit qu'il existe une DF entre X et Y (ou X détermine Y ou encore que Y est déterminé par X), noté X $\rightarrow$ Y si et seulement si :
  $\forall$ t1 et t2, deux tuples (enregistrements) de R, si t1[X] =t2[X] alors t1[Y] =t2[Y]

Propriétés des DF

X et Y sont deux ensembles d'attributs

- **Réflexivité** si $Y \subseteq X$ alors $X \rightarrow Y$ (et donc $X \rightarrow X$)
  Exemple : X={P1, P2,P3}, Y={P2,P3} alors P1, P2, P3 $\rightarrow$ P2,P3
- **Augmentation** Si $X \rightarrow Y$ et $W \subseteq Z$ alors $X, Z \rightarrow Y, W$
  Exemple : X={P1}, Y={P2,P3}, P1 $\rightarrow$ P2, P3 et Z={P4, P5,P6}, W={P4,P6} alors P1, P4, P5, P6 $\rightarrow$ P2, P3, P4, P6
- **Transitivité** Si $X \rightarrow Y$ et $Y \rightarrow Z$ alors $X \rightarrow Z$.
- **Pseudo-transitivité** Si $X \rightarrow Y$ et $Y, Z \rightarrow W$ alors $X, Z \rightarrow W$
- **Union** Si $X \rightarrow Y$ et $X \rightarrow Z$ alors $X \rightarrow Y, Z$
- **Décomposition** $X \rightarrow Y, Z$ alors $X \rightarrow Y$ et $X \rightarrow Z$

Merise – Niveau conceptuel :Les dépendances fonctionnelles

- **En reprenant les données du dictionnaire précédent, les DF sont :**
  - id_em → date_em, delais_em, id_i, ref_e
  - id_i → nom_i, prenom_i, rue_i, ville_i, cp_i, tel_i, tel_port_i, email_i, date_naissance_i
  - ref_e → id_l
  - id_l → titre_l, annee_l, resume_l, id_t, id_ed
  - id_t → libelle_t
  - id_ed → nom_ed
  - id_a → nom_a, prenom_a, date_naissance_a, nom_p
- **On peut déduire les conclusions suivantes de ces DF :**
  - À partir d'un numéro d'emprunt, on obtient une date d'emprunt, un délai, l'identifiant de l'inscrit ayant effectué l'emprunt, la référence de l'exemplaire emprunté.
  - À partir d'une référence d'exemplaire, on obtient l'identifiant du livre correspondant.
  - À partir d'un numéro de livre, on obtient son titre, son année de parution, un résumé, l'identifiant du type correspondant, son numéro d'édition.

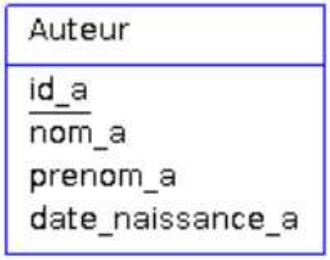
Remarque :

- **Une DF doit être :**
  - élémentaire : C'est l'intégralité de la source qui doit déterminer le but d'une DF.

    exemple si P1 → P3 alors P1,P2 → P3 n'est pas élémentaire.
  - directe : La DF ne doit pas être obtenue par transitivité. Par exemple, si P1 → P2 et P2 → P3 alors P1 → P3 a été obtenue par transitivité et n'est donc pas directe.

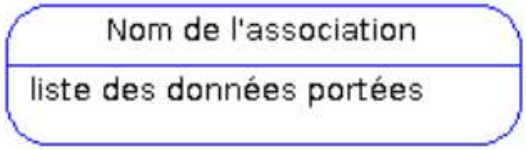
Merise – Niveau conceptuel : Les entités

- A partir du dictionnaire de données, on regroupe les données élémentaires par ‘objets reconnaissables’ appelés entités (opération de composition).
- Le nom d’une entité est souvent un nom représentant un ‘objet de gestion’
  Exemple : Client, Commande, Produit, …
- Chaque entité est unique et est décrite par un ensemble de propriétés encore appelées attributs ou caractéristiques.
- Une des propriétés de l'entité est l'identifiant. Cette propriété doit posséder des occurrences uniques et doit être source des dépendances fonctionnelles avec toutes les autres propriétés de l'entité.



Merise – Niveau conceptuel : Les associations

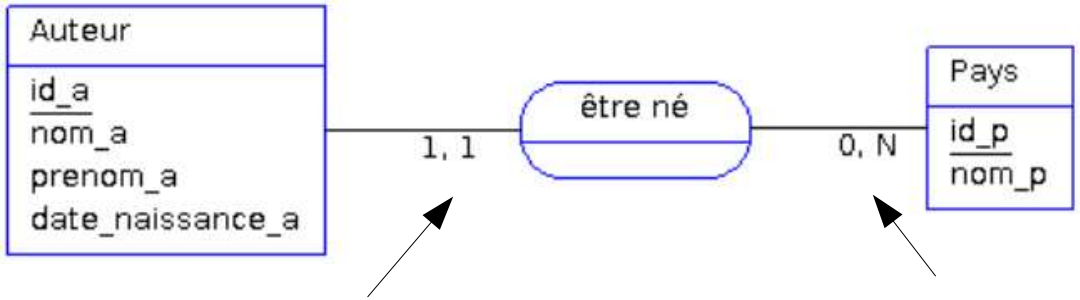
- Une association définit un lien sémantique entre une ou plusieurs entités.
- En effet, la définition de liens entre entités permet de traduire une partie des règles de gestion qui n'ont pas été satisfaites par la simple définition des entités.
- Le formalisme d'une association est le suivant :



- Généralement le nom de l'association est un verbe définissant le lien entre les entités qui sont reliées par cette dernière.

Merise – Niveau conceptuel : Cardinalité

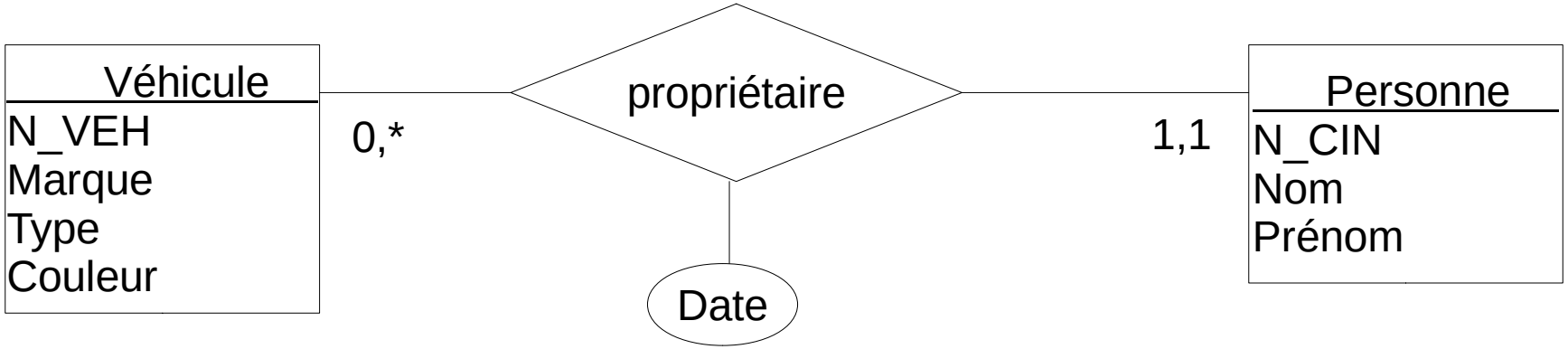
- Les cardinalités permettent de caractériser le lien qui existe entre une entité et la relation à laquelle elle est reliée.
- La cardinalité d'une relation est composée d'un couple comportant une borne maximale et une borne minimale, intervalle dans lequel la cardinalité d'une entité peut prendre les valeurs : 0,N ; 1,N ; 0,1 ; 1,1



se lit « Un auteur est né dans un et un seul pays »

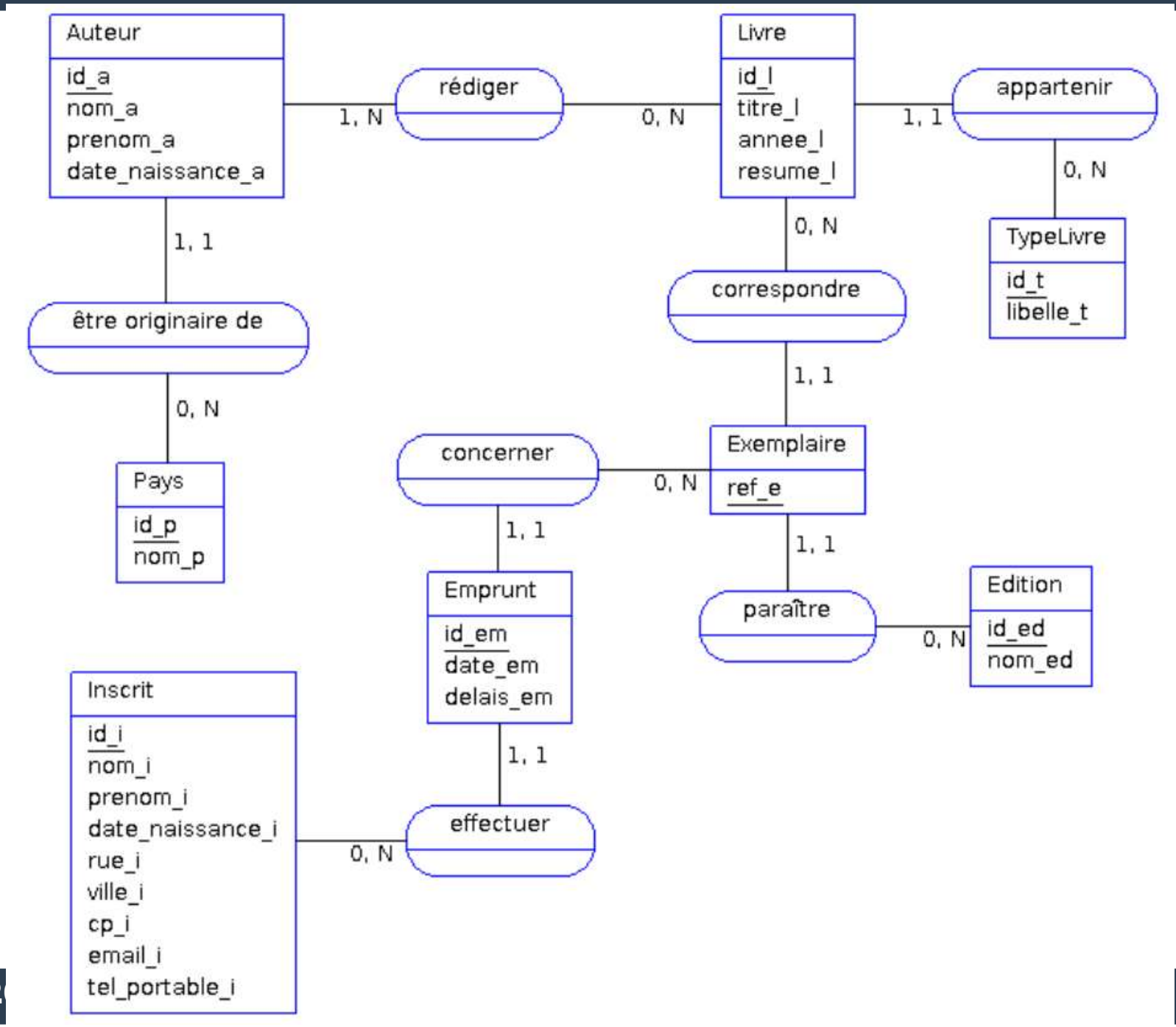
se lit « Dans un pays, sont nés aucun, un ou plusieurs auteurs »

Représentation UML



- Un véhicule est lié à une personne et seule
- Une personne est le propriétaire de zéro ou plusieurs véhicule

Merise – Niveau conceptuel



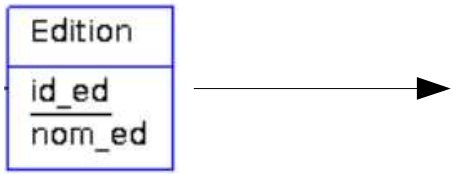
Le passage du MCD au MLD et SQL – Les relations

- Le modèle logique de données (MLD) est composé uniquement de ce que l'on appelle des relations.
- Ces relations sont à la fois issues des entités du MCD mais aussi d'associations, dans certains cas.
- Ces relations nous permettront par la suite de créer nos tables au niveau physique.
- Une relation possède un nom qui correspond en général à celui de l'entité ou de l'association qui lui correspond.
- Une relation est composée d'attributs. Ces attributs sont des données élémentaires issues des propriétés des différentes entités mais aussi des identifiants et des données portées par certaines associations.
- Elle possède aussi une clef primaire qui permet d'identifier sans ambiguïté chaque occurrence de cette relation.
- La clef primaire peut être composée d'un ou plusieurs attributs.

Le passage du MCD au MLD et SQL
Règle 1 - conversion d'une entité

- **En règle générale, toute entité du MCD devient une relation dont la clef est l'identifiant de cette entité.**
- **Chaque propriété de l'entité devient un attribut de la relation correspondante.**

- Exemple de relation (issue de l'entité «Edition» de notre précédant MCD) :



Edition (id_ed, nom_ed)

Clé primaire

Définition : Dépendance Fonctionnelle Totale (DFT)

Soit R(U) une relation, X, Y $\subset$ U et X $\rightarrow$ Y.

On a une DFT entre X et U, X $\rightarrow$ (DFT) Y, si et seulement si

$\forall$ X' $\subset$ X, X' $\rightarrow$ Y n'est pas vérifiée.

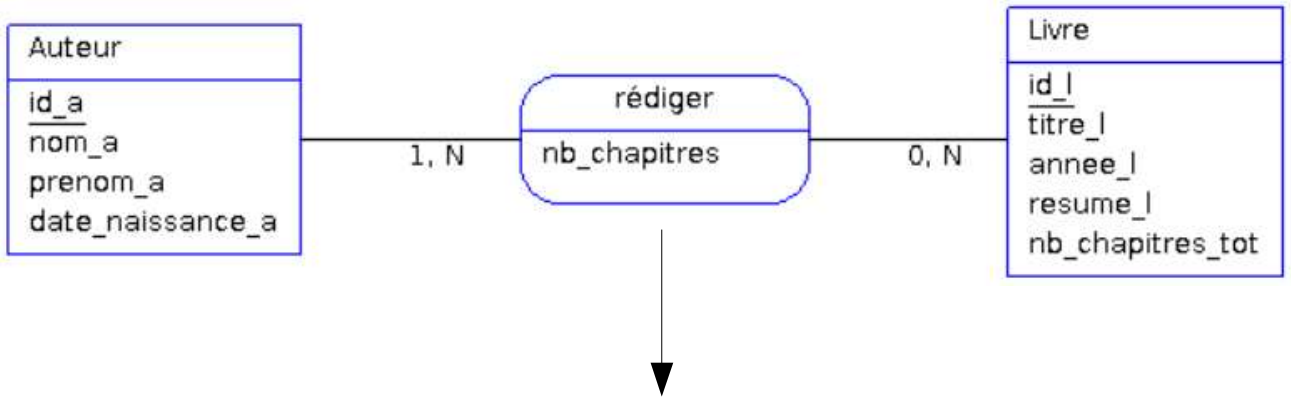
Définition : Clef minimale

Soit R(U) une relation, X $\subset$ U et X est une clef de R.

On dit que X est une clef minimale de R si et seulement si $\forall$ X' $\subset$ X, X' n'est pas une clef de R.

Règle 2 - conversion d'associations n'ayant que des cardinalités de type 0/1,N

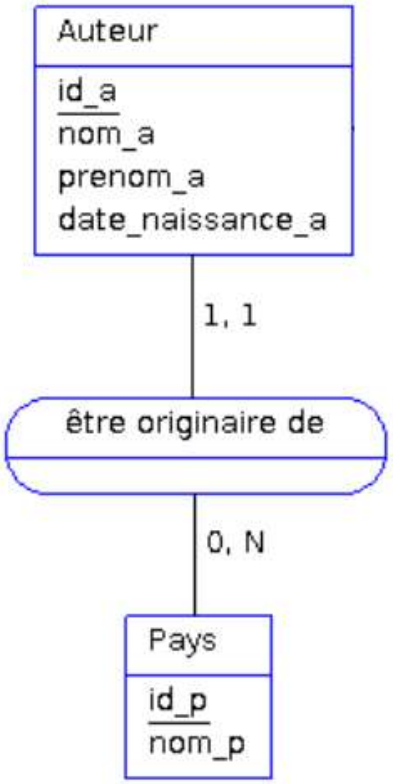
- Une association ayant des cardinalités 0,N ou 1,N de part et d'autre devient une relation dont la clef est constituée des identifiants des entités reliées par cette association.
- Ces identifiants seront donc également des clefs étrangères respectives.



Rédiger(id_a# , id_l#, bn_chapitre)

Règle 3 - conversion des associations ayant au moins une cardinalité de type 1,1

- La règle de conversion la plus répandue aujourd'hui est d'ajouter une clef étrangère dans la relation qui correspond à l'entité se situant du côté de cette cardinalité 1,1.
- Cette clef étrangère fera donc référence à la clef de la relation correspondant à la seconde entité reliée par l'association.
- Lorsque deux entités sont toutes deux reliées avec une cardinalité 1,1 par une même association, on peut placer la clef étrangère de n'importe quel côté.



Pays (id_p, nom_p)
Auteur (id_a, nom_a, prenom_a, date_naissance_a, id_p#)

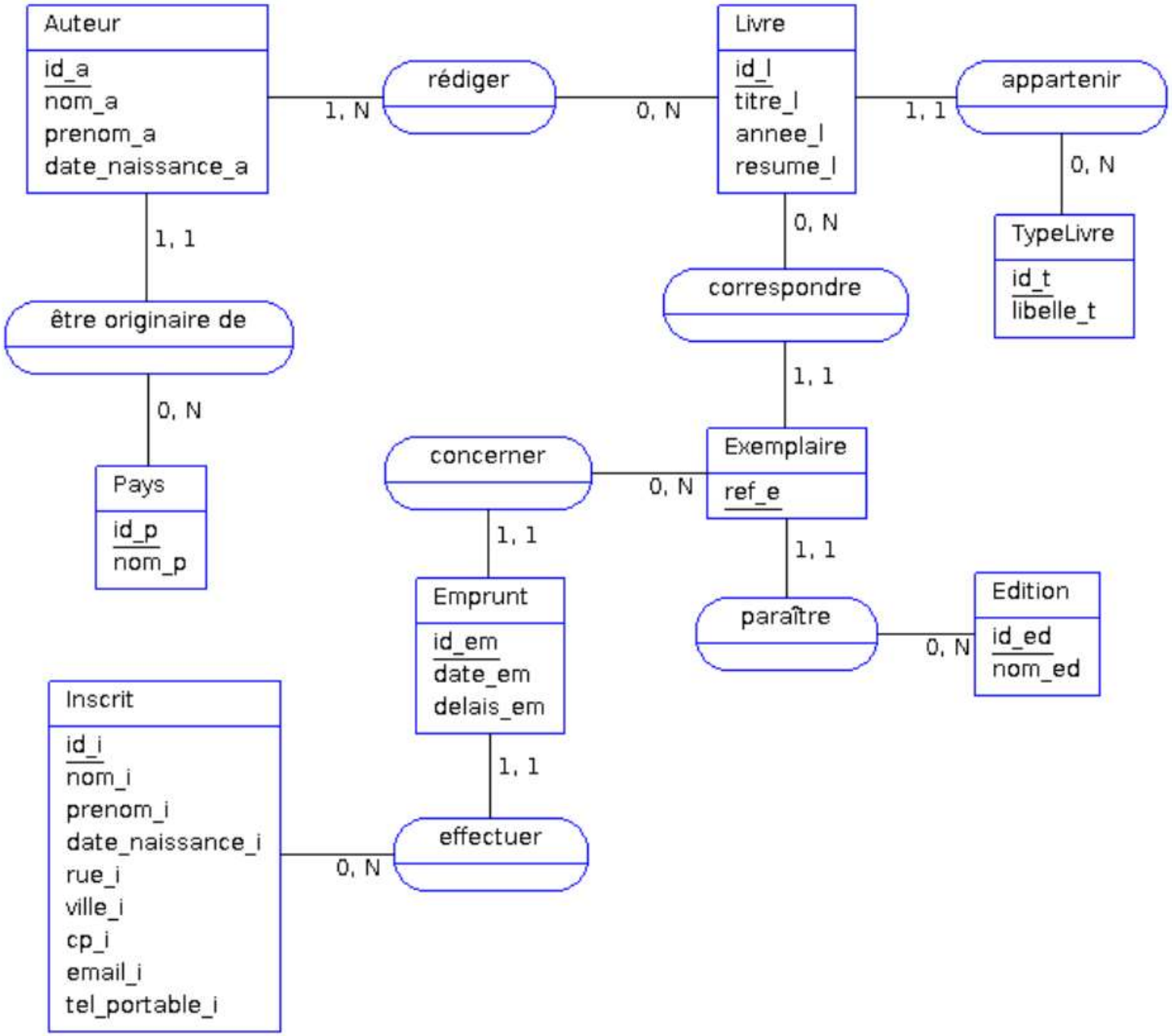
Règle 4 - conversion des associations ayant au moins une cardinalité de type 0,1

- **les deux mêmes possibilités s'offrent à nous**
  - Créer la clef étrangère dans la relation correspondant à l'entité du côté de la cardinalité 0,1. Rappelons que dans ce cas, l'association ne peut pas être porteuse de données.
  - Créer une relation associative qui serait identifié de la même façon que pour une cardinalité 0,1

Categorie (id_cat, libelle_cat)
Livre (id_l, titre_l, annee_l, resume_l, id_cat#)

Categorie (id_cat, libelle_cat)
Livre (id_l, titre_l, annee_l, resume_l)
Appartenir (id_l#, id_cat#)

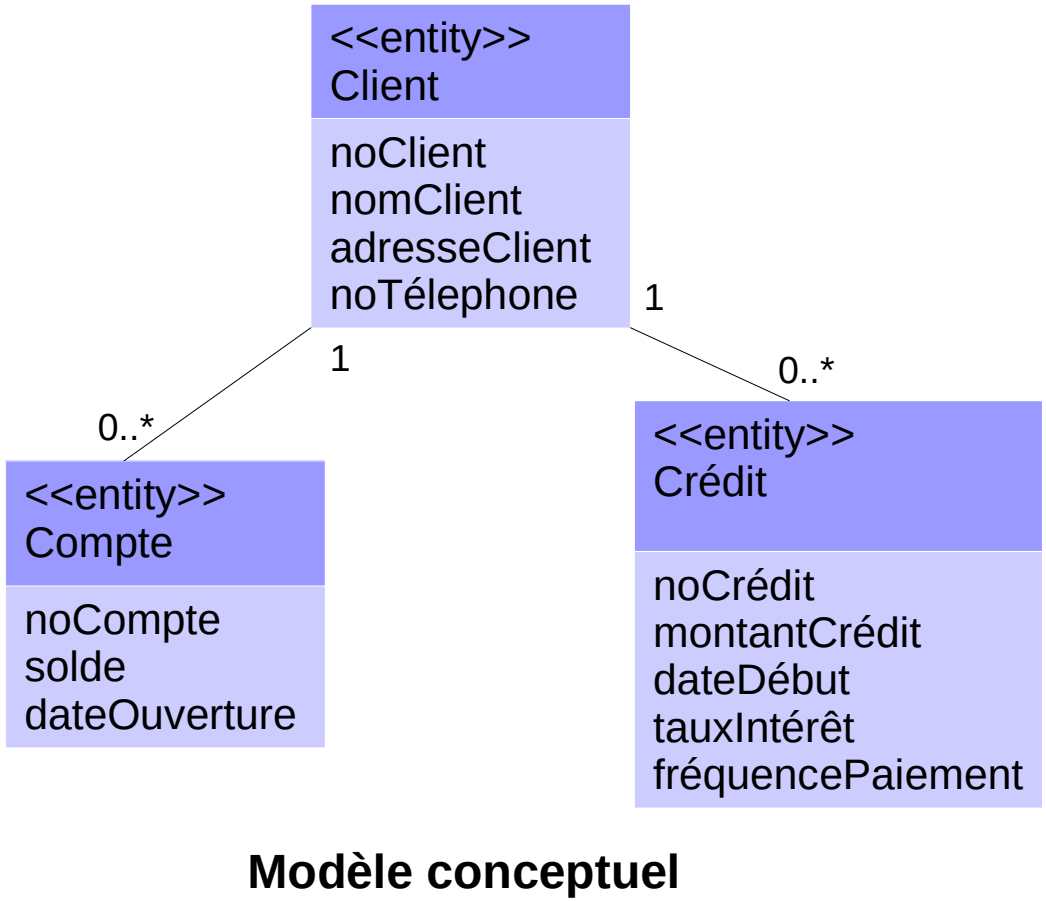
Merise – Niveau conceptuel



Merise – Élaboration du MLD

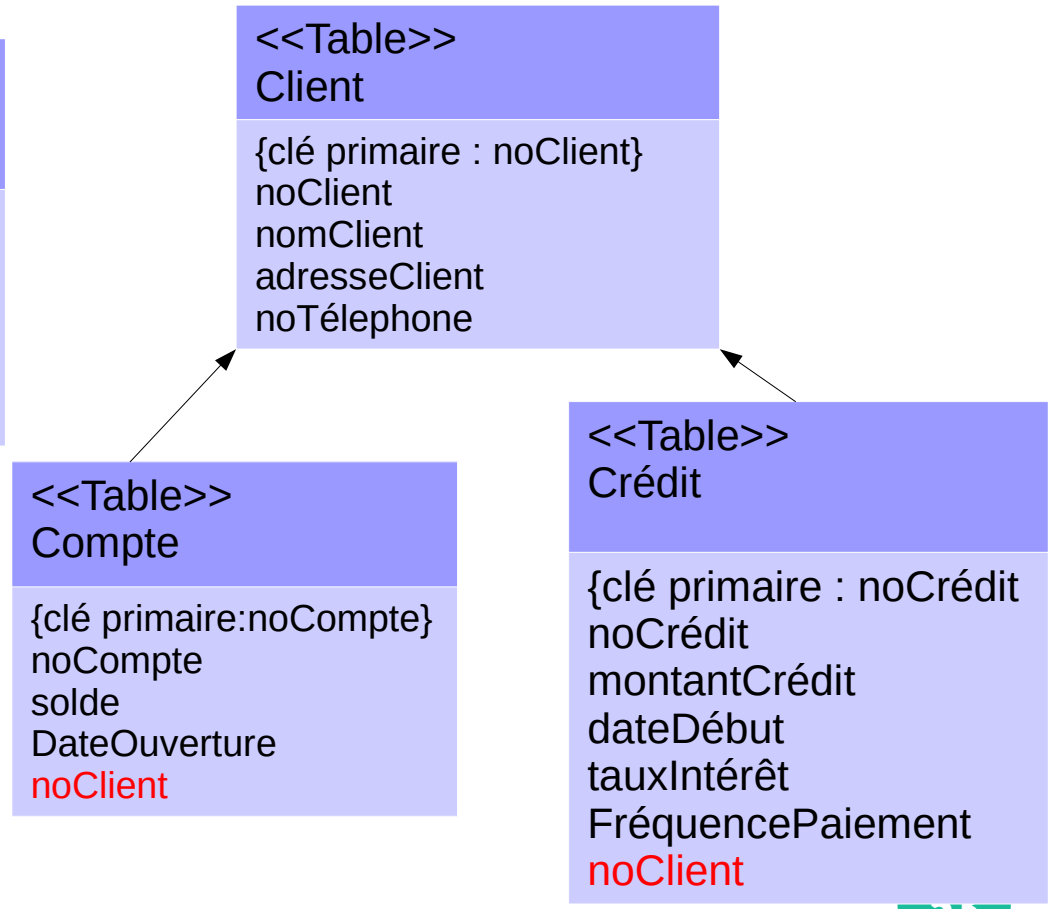
- Pays (id_p, nom_p)
- Auteur (id_a, nom_a, prenom_a, date_naissance_a, id_p#)
- TypeLivre (id_t, libelle_t)
- Livre (id_l, titre_l, annee_l, resume_l, id_t#)
- Rediger (id_a#, id_l#)
- Edition (id_ed, nom_ed)
- Exemplaire (ref_e, id_ed#, id_l#)
- Inscrit (id_i, nom_i, prenom_i, date_naissance_i, rue_i, ville_i, cp_i, email_i, tel_i, tel_portable_i)
- Emprunt (id_em, date_em, delais_em, id_i#, ref_e#)

Conception du schéma d'une base de données – représentation UML



Modèle conceptuel

Schéma relationnel



Le passage du MCD au MLD et SQL – Les relations:clé primaire

Définition : Clef possible ou clef candidate

Soit R(U) une relation et $X \subset U$.

X est une clef possible ou candidate si et seulement si on a $X \rightarrow Y$ où $Y = U - X$ (i.e. le complémentaire de X dans U).

La clef primaire d'une relation doit être choisie dans l'ensemble des clefs possibles de la relation.

S'il n'y en a aucune, tous les attributs de la relation constituent sa clef. On parle alors de relation « toute clef ».

La théorie de la normalisation

La théorie de la normalisation est l'un des points forts du modèle relationnel. Elle permet de définir formellement la qualité des tables au regard du problème posé par la redondance des données.
Formes normales caractérisant les tables relationnelles :

- Première forme normale (IFN) : si tous les attributs sont en dépendance fonctionnelle de la clé. Soit encore :si elle ne contient que des attributs atomiques.
- Deuxième forme normale (2FN) : si elle est en IFN et, si de plus, il n'existe pas de dépendance fonctionnelle entre une partie d'une clé et une colonne non clé de la table.
- Troisième forme normale (3FN) : si elle est en 2FN et si, de plus, aucune dépendance fonctionnelle entre les colonnes non clé.

Première forme normale

- **Première forme normale (1FN)**
  Une relation est première forme normale ssi
  - Tous les attributs sont en dépendance fonctionnelle de la clé,
  - Elle ne contient que des attributs atomiques.

Comment normaliser en 1FN ?

LIVRE	CODE	TITRE	AUTEUR
	100	L'art des BD	Miranda Busta

solution : Créer une nouvelle relation comportant la clef de la relation initiale et l'attribut multi-valué puis éliminer l'attributs multi-valué de la relation initiale (stockage vertical).

Exemple :

LIVRE	CODE	TITRE
	100	L'art des BD

AUTEURS	CODE	AUTEUR
	100	Miranda
	100	Busta

Deuxième forme normale

Deuxième Forme Normale (2 NF)

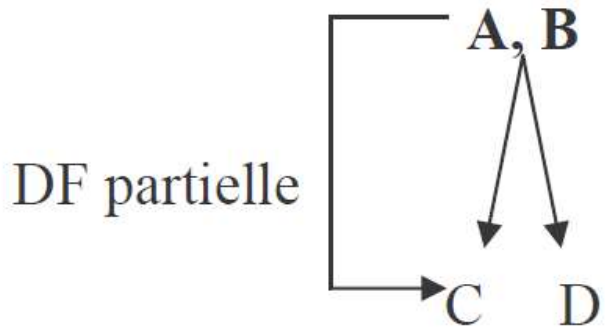
Une relation est en 2 NF si et seulement si :

- Elle est en 1 NF
- Tout attribut n'appartenant pas à la clef primaire est en DF totale avec la clef primaire.

Comment normaliser en 2FN ?

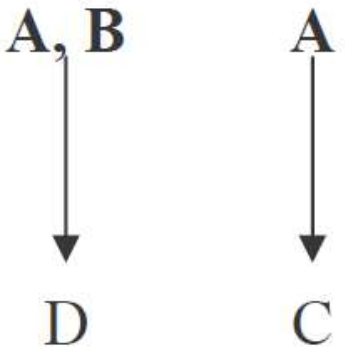
Exemple de relation non 2 NF
ENSEIGNEMENT(**NUM, CODE_MATIERE**, NOM, VOLUME_HORAIRE)
avec NUM → NOM

Graphiquement :



Comment normaliser en 2 NF

- Isoler la DF partielle dans une nouvelle relation
- Eliminer l'attribut cible de la DF de la relation initiale



Exemple :
ENSEIGNEMENT(**NUM, CODE**, VOLUME_HORAIRE)
ENSEIGANT(**NUM**, NOM)

Troisième forme Normale (3NF)

Troisième Forme Normale (3 NF)

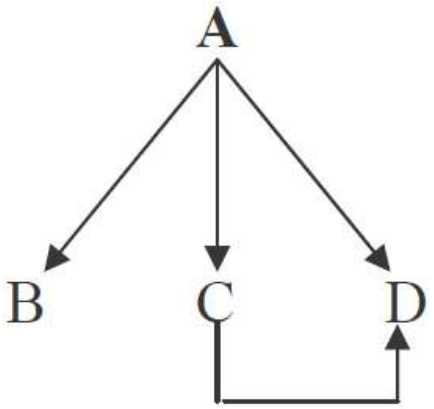
Une relation est en 3 NF si et seulement si :

- Elle est en 2 NF
- Elle ne contient pas de DF Transitive entre attributs non clef

Comment normaliser en 3FN ?

Exemple :
ENSEIGNANT(**NUM**, NOM, CATEGORIE, CLASSE, SALAIRE) avec
CATEGORIE, CLASSE → SALAIRE

Graphiquement :



Comment normaliser en 3 NF :

- Isoler la DF transitive dans la nouvelle relation
- Eliminer la cible de la DF de la relation initiale

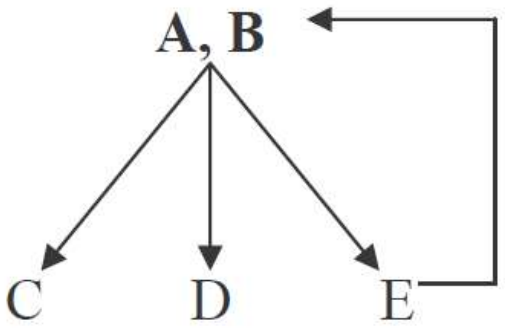
Exemple :
ENSEIGNEMENT(**NUM**, NOM, CATEGORIE, CLASSE)
SALAIRE(**CATEGORIE, CLASSE**, SALAIRE)

Forme Normale de Boyce et Codd (BCNF)

Une relation est en BCNF si et seulement si :

- Elle est en 2 NF
- Toute source de DF est une clef primaire minimale

Graphiquement : relation non BCNF



- Attributs atomiques ➔ 1 NF
- Toutes les DF de source AB sont minimales ➔ 2 NF
- Pas de transitivité entre attribut non clef ➔ 3 NF

Comment normaliser en BCNF ?

Exemple de relation non BCNF

Soit la relation ADRESSE(**Rue, Lieu-dit, Bureau-Postal**, Code Postal)

- Quelles sont les DF existant dans cette relation ?
- En quelle forme normale est la relation ?

Comment normaliser en BCNF

- Isoler la DF problématique dans une nouvelle relation
- Eliminer la cible de la DF problématique et la remplacer par sa source dans la relation initiale

Graphiquement : relation non BCNF

La théorie de la normalisation

Exemples

- **LIGNEARTICLE(numcommande, numproduit, quantité, prixproduit) dont la clé est numcommande + numproduit**
  - n'est pas en 2FN car prixproduit dépend fonctionnellement de numproduit
- **PRODUIT(numproduit, ..., numfournisseur, nomfournisseur, adressefournisseur, ...) de clé numproduit**
  - n'est pas en 3FN car nomfournisseur et adressefournisseur dépendent fonctionnellement de numfournisseur.
- **Une solution est d'éclater la relation en deux relations : la relation PRODUIT(numproduit, ..., numfournisseur) et la relation FOURNISSEUR(numfournisseur, nomfournisseur, adresse fournisseur).**

Langage SQL *Structured Query Language*

- **Norme établie pour SGBD relationnel**
- **Les subdivisions du SQL**
  - **LDD** (langage de définition des données): c'est la partie du SQL qui permet de créer des bases de données, des tables, des vues, des domaines, des index, des contraintes, …

    ALTER, CREATE, DROP
  - **LMD** (langage de manipulation de données) : c'est la partie du SQL qui s'occupe de traiter les données : mise à jour et interrogation (requêtes) INSERT, UPDATE, DELETE, SELECT
  - **LCD** (langage de contrôle de données):c'est la partie du SQL qui s'occupe de gérer les droit d'accès au tables GRANT, REVOKE
  - **LCT** (langage de contrôle de transactions) : c'est la partie du SQL chargé de contrôler la bonne exécution des transactions SET TRANSACTION, COMMIT, ROLLBACK
  - **SQL intégré** (Embedded SQL) : il s'agit d'éléments procéduraux que l'on intègre à un langage hôte (C++, Java …) SET, DECLARE CURSOR, OPEN, FETCH

Langage SQL

- **Il n'y a aucune différence entre un SQL disponible pour un SGBDR de type fichier et un autre SGBD C/S.**
- **Dans ce document, les exemples d'illustration utilisent la base de données dont le schéma relationnel est**
  - avions(No_AV, NOM_AV, CAP, LOC)
  - pilotes(No_PIL, NOM_PIL, VILLE)
  - vols(No_VOL, No_AV, No_PIL, V_d, V_a, H_d, H_a)

SQL comme LDD : Exemple

- **avions(no_AV, NOM_AV, CAP, LOC)**
  - no_AV Integer
  - NOM_AV VARCHAR2(20)
  - CAP integer(4)
  - LOC VARCHAR(15)
- **pilotes(no_PIL, NOM_PIL, VILLE)**
  - no_PIL Integer
  - NOM_PIL VARCHAR(20)
  - VILLE VARCHAR(15)
- **vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a)**
  - no_VOL VARCHAR(5)
  - V_d VARCHAR(15)
  - V_a VARCHAR(15)
  - H_d DATE
  - H_a DATE
  - no_AV integer
  - no_PIL integer

Types de données en SQL2

- **Types pour les chaînes de caractères**
  - CHAR(taille)
    - chaînes de caractères de longueur fixe
    - codage en longueur fixe : remplissage de blancs
    - taille comprise entre 1 et 2000 octets
  - VARCHAR(taille max)
    - chaînes de caractères de longueur variable
    - taille comprise entre 1 et 4000 octets
  - constantes
    - chaînes de caractères entre guillemets
  - NCHAR(taille), NVARCHAR(taille max) sont codés sur le jeu UNICODE

Types de données en SQL2

- **Types numériques**
  - types numériques pour les entiers :
    - SMALLINT pour 2 octets
    - INTEGER pour 4 octets
  - types numériques pour les décimaux à virgule flottante :
    - REAL
    - DOUBLE PRECISION
    - FLOAT
  - types numériques pour les décimaux à virgule fixe :
    - DECIMAL(nb chiffres, nb décimales)
    - NUMERIC(nb chiffres, nb décimales)
  - constantes
    - exemples : 43.8, -13, 5.3E-6

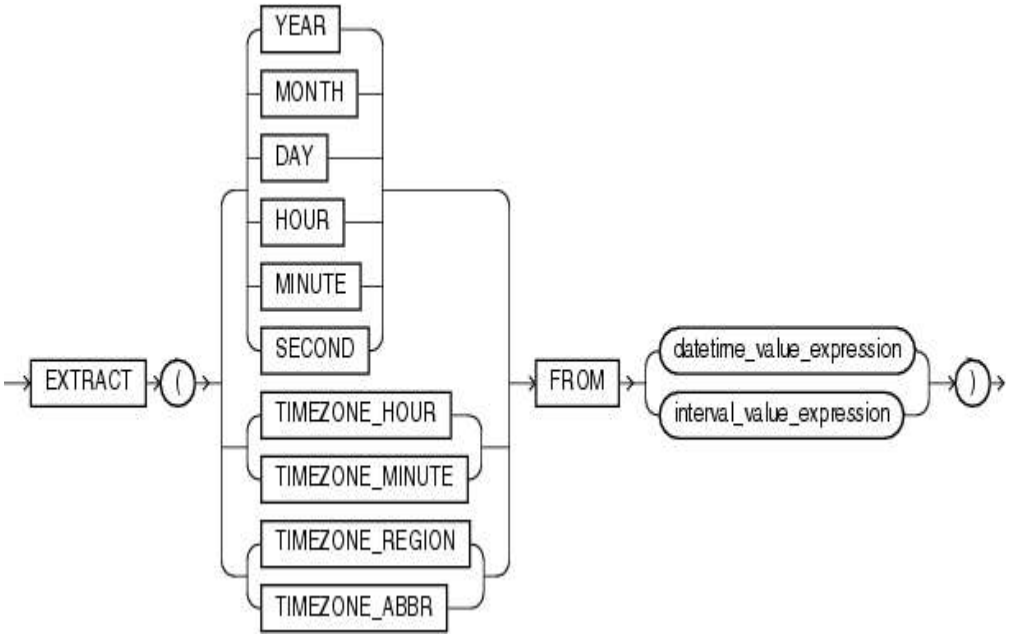
Types de données en SQL2

- **Types temporels**
  - DATE
    - pour les dates
  - TIME
    - pour les heures, minutes et secondes
  - TIMESTAMP
    - pour un moment précis : date et heure, minutes et secondes (précision jusqu' à la microseconde)
  - constantes
    - exemples : '1/05/2007' ou '1 MAY 2007'
- **Remarque : pour oracle le type DATE remplace DATE et TIME de SQL2**

Manipulation des dates

- Exemple de manipulation des dates

```
SELECT
TO_CHAR(SYSDATE, 'DD') JOUR,
TO_CHAR(SYSDATE, 'MM') MOIS,
TO_CHAR(SYSDATE, 'YYYY') ANNEE,
TO_CHAR(SYSDATE, 'HH24') HEURE,
TO_CHAR(SYSDATE, 'MI') MINUTE,
TO_CHAR(SYSDATE, 'SS') SECONDE
FROM DUAL;
```



- EXTRACT

Retourne l'année, le mois, le jour, l'heure, la minute ou la seconde d'une date :

```
SELECT  EXTRACT(YEAR FROM SYSDATE),
EXTRACT(MONTH FROM SYSDATE),
EXTRACT(DAY FROM SYSDATE)
FROM DUAL;
```

```
SELECT SUM(extract(hour from h_d)) NBR_H
FROM vols where no_pil=1;
```

Types de données en SQL2

- Type booléen
  - BIT
    - pour enregistrer la valeur d'un bit
    - exemples : BIT(1), BIT(4)
  - pas supporté par ORACLE

Nouveaux type de données SQL3

- BOOLEAN
- CHARACTER LARGE OBJECT[(n [m])] (ou CLOB) est un type littéral permettant de stocker des caractères de grande dimension dans un jeu à deux octets par caractères (ASCII). n la langueur et m l'unité (K pour kilo, M méga ou G pour géga octects.
- NATIONAL CHARACTER LARGE OBJECT[(n [m])] (ou NCLOB) pour un jeu UNICODE.
- BINARY LARGE OBJECT[(n [m])] (ou BLOB) une chaine de bits de grande dimenssion dont la langueur est fixe.

SQL comme LDD - CREATE

- création de table

  CREATE TABLE nom de table (
  liste de définition de colonne
  [, liste de contrainte de table]) ;

  définition de colonne : :=
  nom de colonne (nom de domaine ou type)
  [liste de contrainte de colonne]
  [DEFAULT valeur par défaut]

SQL comme LDD - CREATE

- création de table : contrainte de colonne
  contrainte de colonne : :=
    [CONSTRAINT nom]
    type de contrainte de colonne

  type de contrainte de colonne : :=
      PRIMARY KEY ou
      NOT NULL ou
      UNIQUE ou
      CHECK(condition sur valeur) ou
      REFERENCES nom de table(nom de colonne)

SQL comme LDD - CREATE

- création de table : contrainte de table
  contrainte de table : :=
    [CONSTRAINT nom]
    type de contrainte de table

  type de contrainte de table : :=
    PRIMARY KEY (liste de nom de colonne) ou
    NOT NULL (liste de nom de colonne) ou
    UNIQUE (liste de nom de colonne) ou
    CHECK (condition sur ligne) ou
    FOREIGN KEY liste de nom de colonne REFERENCES
    nom de de table (liste de nom de colonne)

SQL comme LDD - CREATE

- Exemple : Création de la table avions à partir du schéma :
  avions(no_AV, nom_AV, CAP, LOC)

```
CREATE TABLE avions (
no_AV Integer  CONSTRAINT avions_PK  PRIMARY KEY,
NOM_AV VARCHAR2(20),
CAP Integer CONSTRAINT Dom_CAP_avions CHECK (CAP >= 4),
LOC VARCHAR(15) ) ;
```

SQL comme LDD - CREATE

Exemple : Création de la table vols à partir du schéma :

vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a)

- Contraintes de colonnes ?
- Contraintes de table ?

SQL comme LDD – CREATE

- **Création de la tablevols(no VOL, no AV, no PIL, V d, V a, H d, H a)**

```
CREATE TABLE vols (
    no_VOL VARCHAR(5)
        CONSTRAINT vols_PK PRIMARY KEY,
    no_AV Integer
        CONSTRAINT Ref_no_AV_vols REFERENCES avions(no_AV),
    no_PIL Integer
        CONSTRAINT Ref_no_PIL_vols REFERENCES pilotes(no_PIL),
    V_d VARCHAR(15) NOT NULL,
    V_a VARCHAR(15) NOT NULL,
    H_d DATE,
    H_a DATE,
        CONSTRAINT C1_vols CHECK (v_d <> v_a),
        CONSTRAINT C2_vols CHECK (h_d < h_a) ) ;
```

SQL comme LDD - DROP

- suppression de table

DROP TABLE nom ;

Quand une table est supprimée, ORACLE :
- efface tous les index qui y sont attachés quelque soit le propriétaire
- efface tous les privilèges qui y sont attachés

MAIS les vues et les synomymes se référant à cette table ne serons pas supprimés

SQL comme LDD - ALTER

- modification de table

ALTER TABLE nom_de_table modification_de_table ;

modification_de_table : :=

ADD COLUMN définition de colonne

ADD CONSTRAINT contrainte de table

DROP COLUMN nom de colonne

DROP CONSTRAINT nom de contrainte

SQL comme LDD - ALTER

- Exemple : Ajout d'un colonne à la table vols de schéma :

vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a)

```
ALTER TABLE vols ADD COLUMN COUT_VOL Integer ;
```

le schéma devient :

vols(no_VOL, no_AV, no_PIL, V_d, V_a, H_d, H_a, COUT_VOL)

Cas particulier oracle : MODIFIER DES COLONNES MULTIPLES

- Pour MODIFIER DES COLONNES MULTIPLES dans une table existante, la syntaxe Oracle ALTER TABLE est la suivante:

```
ALTER TABLE table_name
 MODIFY (column_1 column_type,
      column_2 column_type,
      ...
      column_n column_type);
```

- Exemple de modification du type d'un champ sous oracle

```
Alter table  vols
 Modify (
          H_D   TIMESTAMP ,
          H_A   TIMESTAMP
        );
```

SQL comme LMD - INESRT

- insertion de lignes dans une table

  INSERT INTO nom de table [liste de colonnes]
  VALUES liste de valeurs ;

  ou

  INSERT INTO nom de table [liste de colonnes] requête ;

SQL comme LMD - INESRT

- **ajouter un avion dans la table avions en respectant l'ordre des colonnes**

```
INSERT INTO avions  VALUES (100, 'Airbus', 200, 'Agadir') ;
```

- **ajouter un avion dans la table avions sans connaître l'ordre**

```
INSERT INTO avions (no_AV, CAP, LOC, NOM_AV)
VALUES (101, 200, 'Agadir', 'Airbus') ;
```

- **ajouter un avion dans la table avions dont la localisation est INDEFINI**

```
INSERT INTO avions (no_AV, NOM_AV, CAP)
VALUES (102, 'Airbus', 200) ;
```

ou

```
INSERT INTO avions
VALUES (102, 'Airbus', 200, NULL) ;
```

SQL comme LMD - INESRT

Exemples de la commande insert

- **insert into vols values(1,1,1, 'Agadir', 'Rabat',**
  To_DATE('1983-04-25 8:30:00AM', 'YYYY-MM-DD HH:MI:SSAM'),
  To_DATE('1983-04-25 10:30:00AM', 'YYYY-MM-DD HH:MI:SSAM'));

- **insert into pilotes values(2, "Ayoubi", "ERRACHIDIYA");**

SQL comme LMD - DELETE

- **suppression de lignes d'une table**
  DELETE FROM **nom de table** [WHERE **condition**] ;

  Exemples :
- **vider la table avions**
  DELETE FROM avions ;
- **supprimer de la table avions tous les avions dont la capacité est inférieur à 100**
  DELETE FROM avions
  WHERE CAP < 100 ;

SQL comme LMD - UPDATE

- modification de lignes dans une table

  UPDATE nom_de_table SET liste_expression_colonne
  [WHERE condition] ;
  expression_colonne : :=

  nom_de_colonne = expression ou

  nom_de_colonne = requête

  Exemple :
- modifier la capacité de l'avion numéro 100

  UPDATE avions SET CAP = 300

  WHERE no AV = 100 ;

SQL comme LMD - SELECT

- **interrogation**
  requête : := SELECT [DISTINCT] projection
  FROM liste de (nom de table [[AS] nom]) | (requête AS nom)
  WHERE condition
  [GROUP BY liste de nom de colonne]
  [HAVING condition]
  [ORDER BY liste de ((nom de colonne | rang de colonne) (ASC | DESC)] ;
  requête : := requête (UNION | INTERSECT | EXCEPT) requête
  requête : := (requête)

SQL comme LMD - SELECT

- **projection : :=**
  *** | nom de table | liste de (terme de projection[[AS] nom])**
  terme de projection : :=
  expression | agrégation

  expression : :=
  valeur | nom de colonne | expression arithmétique | · · ·

  agrégation : :=
  COUNT(*)

  opérateur d'agrégation([DISTINCT] expression)

  opérateur d'agrégation : :=
  COUNT | SUM | AVG | MAX | MIN

SQL comme LMD - SELECT

- **comparaison : :=**

  expression (= | <>| > | < | <= | >= |) expression |

  expression (= | <>) (| SOME | ALL) requête

  expression (> | < | <= | >=) (| SOME | ALL)

  requête_mono_colonne

- **appartenance à un intervalle : :=**

  expression BETWEEN expression AND expression

- **appartenance à un ensemble : :=**

  expression (IN | NOTIN) (requête) |

  (liste de expression) (IN | NOTIN) (requête)

- **ensemble de valeurs : := (liste de valeur) | requête mono colonne**
- **existence : := EXISTS (requête)**

SQL comme LMD - SELECT

- **Sélection de lignes**
  - **toutes les lignes et toutes les colonnes**

    SELECT * FROM nom de table ;

    - pour connaître toutes les caractéristiques des avions stockés dans la table

      SELECT * FROM avions ;
  - **toutes les lignes mais seulement certaines colonnes**

    SELECT liste de nom de colonne FROM nom de table;

    - pour connaître les numéros de tous les vols

      SELECT no_VOL FROM vols ;

SQL comme LMD - SELECT

- **Sélection de lignes**
  - **suppression des lignes dupliquées**

    SELECT DISTINCT liste de nom de colonne FROM nom de table ;
    - pour connaître les numéros des pilotes qui conduisent au moins un avion
      SELECT DISTINCT no_PIL FROM vols ;
  - **colonnes calculées**

    SELECT expression [AS alias] FROM nom de table ;
    - afficher une augmentation de 5% du coût de chaque vol

      SELECT no_VOL, '5%' "%",

      COUT_VOL*0.05 AUGM, COUT_VOL*1.05 "Nouveau coût"

      FROM vols ;

SQL comme LMD - SELECT

- **sur les chaînes de caractères**
  - afficher les trajets assurés par les vols sous la forme :
    Ville de départ −− > Ville d'arrivée

```
SELECT no_VOL, V_d || ' − − > ' || V_a TRAJET
FROM vols ;
```

- **sur les dates**
  - afficher les dates de départ et d'arrivée de chaque vol en décalant les dates

```
SELECT no_VOL, D_d + 1/24 D_d, D_a + 1/24 D_a
FROM vols ;
```

  - pour connaître la durée en heures de tous les vols

```
SELECT no_VOL, 24 *(D_a -D_d) durée
FROM vols ;
```

SQL comme LMD - SELECT

- **Recherche par comparaison**

  SELECT liste de nom de colonne FROM nom de table

  WHERE expression ;

  - pour connaître tous les avions qui ont une capacité > 200 places

```
SELECT no_AV
    FROM avions
    WHERE CAP > 200 ;
```

Date * 24 ⇒ HEURES
Date *1440 ⇒ MINUTES
Date *86400 ⇒ SECONDES

  - pour connaître tous les pilotes qui effectuent des vols qui durent plus d'une heure

```
SELECT DISTINCT no_PIL
    FROM vols
    WHERE (24 *(D_a -D_d) )> 1 ;
```

SQL comme LMD - SELECT

- **Recherche par ressemblance**

  SELECT liste de nom_de_colonne FROM nom_de_table

  WHERE expression [NOT] LIKE motif [caractères spéciaux] ;

  - caractère spéciaux :
    - % : remplace 0, 1 ou plusieurs caractères
    - _ remplace 1 caractère
  - caractère d'échappement :
    - permet de traiter les caractère spéciaux comme de simples caractères
    - il n'y a pas de caractère d'échappement prédéfini

SQL comme LMD - SELECT

- **Recherche par ressemblance : exemples**
  pour connaître la capacité de tous les avions Boeing

```
SELECT no_AV, NOM_AV, CAP
    FROM avions
    WHERE NOM_AV LIKE 'Boeing%' ;
```

SQL comme LMD - SELECT

- **Recherche avec condition conjonctive**
  SELECT liste de nom_de_colonne FROM nom_de_table
  WHERE condition AND condition ;
  - pour connaître tous les avions qui sont à Agadir et dont la capacité est de 300 places

```
SELECT no_AV
    FROM avions
    WHERE LOC = 'Agadir' AND CAP = 300 ;
```

SQL comme LMD - SELECT

- **Recherche avec condition disjonctive**
  SELECT liste de nom_de_colonne FROM nom_de_table
  WHERE condition OR condition ;
  - pour connaître tous les vols qui utilisent les avions 100 ou 101

```
SELECT no VOL
  FROM vols
  WHERE no_AV = 100 OR no_AV = 101 ;
```

SQL comme LMD - SELECT

- **Recherche avec condition négative**
  SELECT liste de nom_de_colonne FROM nom_de_table
  WHERE NOT condition ;
  - pour connaître tous les pilotes qui n'habitent pas à Rabat

```
SELECT no_PIL
    FROM pilotes
    WHERE NOT VILLE = 'Rabat' ;
```

SQL comme LMD - SELECT

- **Recherche avec un intervalle**
  SELECT liste de nom_de_colonne FROM nom_de_table
  WHERE expression BETWEEN expression AND expression ;
  - pour connaître tous les avions qui ont une capacité entre 200 et 300 places

```
SELECT no AV
    FROM avions
    WHERE CAP BETWEEN 200 AND 300 ;
```

SQL comme LMD - SELECT

- **Recherche avec une liste**
  SELECT liste de nom_de_colonne FROM nom_de_table
  WHERE expression [NOT] IN liste de expression ;
  - pour connaître tous les pilotes qui habitent soit à Rabat soit à Casa

```
SELECT no_PIL
    FROM pilotes
    WHERE VILLE IN ('Rabat', 'Casa') ;
```

SQL comme LMD - SELECT

- **Recherche avec une liste**
  SELECT liste de nom_de_colonne FROM nom de table
  WHERE expression (<>| > | < | <= | >= |) ALL
  liste de expression ;
  SELECT liste de nom de colonne FROM nom de table
  WHERE expression (<>| > | < | <= | >= |) SOME
  liste de expression ;
    - pour connaître tous les vols dont le départ est à plus de 5 jours et dont la durée est moins de 5 heures

      SELECT no_VOL, D_d, 24*(D_a - D_d) Durée

      FROM vols

      WHERE D_d > ALL (sysdate +5, D_a -5/24) ;

SQL comme LMD - SELECT

- **Traitement de l'absence de valeur**
  - **sur les expressions numériques**

    un calcul numérique ou de dates exprimé avec les opérateurs +, −, ∗, / n'a pas de valeur lorsqu'au moins une des composantes n'a pas de valeur

```
SELECT no_AV, 2*CAP/3 AS CAP_RED
    FROM avions
    WHERE no_AV = 320 ;
```

SQL comme LMD - SELECT

- **Traitement de l'absence de valeur**
  - **sur les chaînes de caractères**
    - un calcul de chaînes exprimé avec l'opérateur || n'a pas de valeur lorsque toutes ses composantes n'ont pas de valeur
    - la chaîne vide et l'absence de valeur sont confondues

```
SELECT no_CL, NOM_RUE_CL || '' || VILLE_CL
AS ADR_CL
ou no_CL, NOM_RUE_CL || NULL || VILLE_CL AS ADR_CL
FROM clients
WHERE no_CL = 1035 ;
```

SQL comme LMD - SELECT

- **Traitement de l'absence de valeur**
  - **sur les comparaisons**
    - toute comparaison exprimée avec les opérateurs =, <>, >, <, <=, >=, LIKE qui comporte une expression qui n'a pas de valeur prend la valeur logique INDEFINIE
    - les comparaisons ignorent les lignes où il y a absence de valeur

      SELECT * FROM pilotes

      WHERE NAISS_PIL <> 1960 AND VILLE <> 'Agadir' ;
    - comparaisons indéfinies :

      SELECT *

      FROM avions

      WHERE NULL = NULL OR '' = '' OR '' LIKE '%'

      OR 'A' LIKE '' OR 'A' NOT LIKE '' ;

SQL comme LMD - SELECT

- **équivalences disjonctives**
  - expr NOT BETWEEN expr_1 AND expr_2

    $\Leftrightarrow$ expr < expr_1 OR expr > expr_2
  - expr IN ($\text{expr}_1 \cdots \text{expr}_N$)

    $\Leftrightarrow$ expr = expr_1 OR $\cdots$ OR expr = expr_N
  - expr op ANY ($\text{expr}_1 \cdots \text{expr}_N$)

    $\Leftrightarrow$ expr op expr_1 OR $\cdots$ OR expr op expr_N

SQL comme LMD - SELECT

- **Les expressions suivantes :**
  - expr NOT BETWEEN $expr_1$ AND $expr_2$
  - expr IN ($expr_1 \cdots expr_N$)
  - expr op ANY ($expr_1 \cdots expr_N$)

  sont vraies ssi expr a une valeur et si au moins une des expressions $expr_1$, $expr_N$ a une valeur qui satisfait les comparaisons

  SELECT NUM_PIL FROM pilotes
  WHERE VILLE IN ('Rabat', 'Fes', '') ;

SQL comme LMD - SELECT

- **équivalences conjonctives**
  - expr BETWEEN $expr_1$ AND $expr_2$

    $\Leftrightarrow$ expr >= $expr_1$ OR expr <= $expr_2$
  - expr NOT IN ($expr_1 \cdots expr_N$)

    $\Leftrightarrow$ expr <> $expr_1$ OR $\cdots$ OR expr <> $expr_N$
  - expr op ALL ($expr_1 \cdots expr_N$)

    $\Leftrightarrow$ expr op $expr_1$ AND $\cdots$ AND expr op $expr_N$

SQL comme LMD - SELECT

- **Les expressions suivantes :**
  - expr BETWEEN $expr_1$ AND $expr_2$
  - expr NOT IN ($expr_1 \cdots expr_N$)
  - expr op ALL ($expr_1 \cdots expr_N$)

  sont vraies ssi expr a une valeur et si au toutes les expressions $expr_1$, $expr_N$ a une valeur qui satisfait les comparaisons

  SELECT NUM_PIL FROM pilotes
  WHERE VILLE NOT IN ('Marseille', 'Nice', '') ;

SQL comme LMD - SELECT

- **Traitement de l'absence de valeur**
  - **recherche de l'absence de valeur**

SELECT liste de nom_de_colonne FROM nom de table
WHERE expression IS [NOT] NULL ;

- pour connaître tous les vols auxquels on n'a pas encore affecté d'avions

```
SELECT no_VOL
    FROM vols
    WHERE no_AV IS NULL ;
```

SQL comme LMD - SELECT

- **Traitement de l'absence de valeur**
  - **Donner une valeur à l'absence de valeur**

NVL (expr 1 , expr 2) = expr 1 si elle définie, expr 2 sinon
expr 1 et expr 2 doivent être de même type

- pour qu'une capacité d'avion inconnue soit considérée comme 0

```
SELECT no_VOL, NVL(CAP,0)
    FROM avions ;
```

SQL comme LMD - SELECT

- Ordonner les réponses
  SELECT liste de nom de colonne
  FROM nom de table
  [WHERE expression]
  ORDER BY { expression | position } [ASC | DESC]
  [{ expression | position } [ASC | DESC]] ;

SQL comme LMD - SELECT

- **Ordonner les réponses**
  - pour connaître les horaires des vols triés par ordre croissant des dates et heures de départ

```
SELECT no_VOL, D_d, D_a
FROM vols
ORDER BY D_d ;
```

SQL comme LMD - SELECT

- **Les fonctions de groupe**
  - les fonctions de groupe calculent les résultats à partir d'une collection de valeurs.

COUNT (*|[DISTINCT | ALL|expression]) comptage des lignes
COUNT ([DISTINCT | ALL|expression]) comptage des valeurs
MAX ([DISTINCT | ALL|expression]) maximum des valeurs
MIN ([DISTINCT | ALL|expression]) minimum des valeurs
SUM ([DISTINCT | ALL|expression]) somme des valeurs
AVG ([DISTINCT | ALL|expression]) moyenne des valeurs
STDDEV ([DISTINCT | ALL|expression]) écart-type des valeurs
VARIANCE ([DISTINCT | ALL]|expression) variance des valeurs

SQL comme LMD - SELECT

- **Les fonctions de groupe**
  - pour connaître le nombre d'avions

```
SELECT COUNT(*) NBR_AV
FROM avions ;
```

  - pour connaître le nombre d'heures de vols du pilote 4020

```
SELECT SUM(24 *(D_a - D_d)) NBR_H
FROM vols
WHERE no_PIL = 4020 ;
```

SQL comme LMD - SELECT

- **Les regroupements de lignes**
  - les fonctions de groupe calculent les résultats à partir d'une collection de valeurs.

    SELECT liste d'expressions1

    FROM nom de table

    GROUP BY liste d'expressions2 ;
  - les expressions de liste d'expressions1 doivent être des expressions formées uniquement :
    - d'expressions de liste d'expressions2
    - de fonctions de groupe
    - de constantes littérales

SQL comme LMD - SELECT

- **Les regroupements de lignes**
  - pour connaître le nombre d'avions affectés à chaque ville d'affectation d'un avion

```
SELECT LOC, COUNT(*) NBR_AV
FROM avions
GROUP BY LOC ;
```

SQL comme LMD - SELECT

- **Les regroupements de lignes**
  - pour connaître le nombre de vols qui ont la même durée

```
SELECT 24*(D_a - D_d) DUR_VOL, COUNT(*) NBR_VOL
FROM vols
GROUP BY D_a - D_d ;
```

SQL comme LMD - SELECT

- **Les regroupements de lignes**
  - regroupement de lignes sélectionnées

```
SELECT liste d'expressions1
FROM nom de table
WHERE condition
GROUP BY liste d'expressions2 ;
```

  - pour connaître le nombre d'avions différents utilisés par chaque pilote assurant un vol

```
SELECT LOC, no_PIL, COUNT(DISTINCT no_AV) NBR_AV
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL ;
```

SQL comme LMD - SELECT

- **Conditions sur l'ensemble des lignes**
  - pour savoir si le pilote 4010 assure tous les vols avec un avion différent à chaque fois

```
SELECT 'OUI' REP
FROM vols
WHERE no_PIL = 4010
HAVING COUNT(*) = COUNT(DISTINCT no_AV) ;
```

SQL comme LMD - SELECT

- **Conditions sur l'ensemble des lignes**

  SELECT liste d'expressions

  FROM nom de table

  [WHERE condition]

  GROUP BY liste d'expressions2

  HAVING condition sur lignes ;

- **les expressions de liste d'expressions et condition sur lignes doivent être formées uniquement :**
  - d'expressions de liste d'expressions2
  - de fonctions de groupe
  - de constantes littérales

SQL comme LMD - SELECT

- **Conditions sur l'ensemble des lignes**
  - pour connaître les pilotes qui conduisent au moins deux avions différents

```
SELECT no_PIL
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL
HAVING COUNT(DISTINCT no_AV) >= 2 ;
```

SQL comme LMD - SELECT

- **Opérateurs ensemblistes**

SELECT liste d'expressions1

FROM nom de table

[WHERE condition]

[GROUP BY liste d'expressions2]

UNION | UNION ALL | INTERSECT | MINUS

SELECT liste d'expressions3

FROM nom de table

[WHERE condition]

[GROUP BY liste d'expressions4] ;

SQL comme LMD - SELECT

- **Opérateurs ensemblistes**
  - pour connaître les villes qui sont soit des villes de départ soit des villes d'arrivées d'un vol

```
SELECT V_d VILL
FROM vols
WHERE V_d IS NOT NULL
UNION
SELECT V_a
FROM vols
WHERE V_a IS NOT NULL ;
```

SQL comme LMD - SELECT

- **Opérateurs ensemblistes**
  - pour connaître le nombre de vols assurés par chaque pilote

```
SELECT no_PIL, COUNT(*) NBR_VOL
FROM vols
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL
UNION ALL
    (SELECT no_PIL, 0
    FROM pilotes
    MINUS
    SELECT no_PIL, 0
    FROM vols) ;
```

SQL comme LMD - SELECT

- **Produit cartésien**

  SELECT liste d'expressions

  FROM liste de(nom de table [alias])

  [WHERE condition] ;

  - pour connaître le coût de chaque classe du vol V900 lorsqu'on les applique au vol V100

    SELECT Classe, COEF_PRIX * COUT_VOL COUT

    FROM defclasses D, vols V

    WHERE D.no_VOL = 'V900' AND V.no_VOL = 'V100' :

SQL comme LMD - SELECT

- **Opérateur de jointure naturelle**

  SELECT liste d'expressions

  FROM liste de(nom de table [alias])

  WHERE expr comp expr [AND | OR expr comp expre] ;

  ou

  SELECT liste d'expressions

  FROM

  nom de table [alias]

  INNER JOIN

  nom de table [alias]

  ON expr comp expr [AND | OR expr comp expre] ;

SQL comme LMD - SELECT

- **Opérateur de jointure naturelle**
  - pour connaître le nombre de places de chaque vol qui a été affecté à un avion

```
SELECT no_VOL, CAP
FROM vols, avions
WHERE vols.no_AV = avions.no_AV ;
```

SQL comme LMD - SELECT

table vols	
no_VOL	no_AV
V101	560
V141	101
V169	101
V631	NULL
V801	240

table avions	
no_AV	CAP
101	350
240	NULL
560	250

equi-jointure sur no_AV

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

SQL comme LMD - SELECT

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL

projection sur no_VOL, CAP

vols.no_VOL	avions.CAP
V101	250
V141	350
V169	350
V801	NULL

SQL comme LMD - SELECT

SELECT liste d'expressions

FROM nom de table1, nom de table2

WHERE expr table1 comp nom table2.col(+)

[AND expr table1 comp nom table2.col(+)]

ou

SELECT liste d'expressions

FROM

nom de table1 [alias]

LEFT JOIN | RIGHT JOIN

nom de table2 [alias]

ON expr comp expr [AND | OR expr comp expr] ;

SQL comme LMD - SELECT

- **Opérateur de semi-jointure externe**

  pour connaître le nombre de places de chaque vol (même lorsqu'aucun avion n'est affecté au vol)

```
SELECT no_VOL, CAP
FROM vols V LEFT JOIN avions A
ON V.no_AV = A.no_AV ;
```

SQL comme LMD - SELECT

table vols	
no_VOL	no_AV
V101	560
V141	101
V169	101
V631	NULL
V801	240

table avions	
no_AV	CAP
101	350
240	NULL
560	250

equi-jointure externe

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240
V631	NULL

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL
NULL	NULL

SQL comme LMD - SELECT

vols.no_VOL	vols.no_AV
V101	560
V141	101
V169	101
V801	240
V631	NULL

avions.no_AV	avions.CAP
560	250
101	350
101	350
240	NULL
NULL	NULL

projection sur no_VOL, CAP

vols.no_VOL	avions.CAP
V101	250
V141	350
V169	350
V801	NULL
V631	NULL

SQL comme LMD - SELECT

- **Sous-requêtes**
  - imbrication de sous-requêtes dans la clause WHERE

```
SELECT projection
FROM nom de table
WHERE condition
    (SELECT projection
    FROM nom de table
    WHERE condition, · · · ) ;
```

SQL comme LMD - SELECT

- **Sous-requêtes : donnant une seule ligne**
  - pour connaître les vols qui utilisent le même avion que celui utilisé par le vol V101

```
SELECT no_VOL
FROM vols
WHERE no_Av = (SELECT no_Av
               FROM vols
               WHERE no_VOL = 'V101' ) ;
```

SQL comme LMD - SELECT

- **Sous-requêtes donnant au plus une ligne**
  - pour connaître le vols qui assure le même trajet que celui du vol V101 mais 2 jours plus tard

```
SELECT no_VOL
FROM vols
WHERE (V_d, V_a, D_d, D_a) =
            (SELECT (V_d, V_ a, D_d+2, D_a+2)
             FROM vols
             WHERE no VOL = 'V101' ) ;
```

SQL comme LMD - SELECT

- **Sous-requêtes donnant 0, 1 ou plusieurs lignes**
  - pour connaître les vols qui sont assurés par un pilote qui habite Rabat

```
SELECT no_VOL
FROM vols
WHERE no_PIL IN
        (SELECT no_PIL
         FROM pilotes
         WHERE VILLE = 'Rabat' ) ;
```

SQL comme LMD - SELECT

- **Sous-requêtes donnant 0, 1 ou plusieurs lignes**
  - pour connaître les pilotes qui n'assurent aucun vol

```
SELECT no_PIL
FROM pilotes
WHERE no_PIL NOT IN
                    (SELECT no_PIL
                     FROM vols
                     WHERE no_PIL IS NOT NULL ) ;
```

SQL comme LMD - SELECT

- **Sous-requêtes d'existence**
  - pour connaître les avions qui sont conduits par au moins un pilote de Marseille

```
SELECT DISTINCT no_AV
FROM vols
WHERE EXISTS
        (SELECT *
         FROM pilotes
        WHERE no_PIL = vols.no_PIL
       AND VILLE = 'Agadir') ;
```

Suite de TP N°1

Ecrire les requêtes SQL qui permettent de :

- **Toutes les informations sur les clients.**
- **Toutes les informations sur les clients résident à "kénitra".**
- **Les commandes avec une quantité entre 3 et 6.**
- **Les numéros des clients qui sont situés dans une ville qui commence par 'k'.**
- Changer la ville du client ‘Naciri Mohammed’ de ‘Méknès’ à ‘Rabat’.
- Supprimer le client N°1. Remarque ?

Suite de TP N°1

- **Exercice N°1**

  Modifier la base de données commerce de tel sorte à gérer le cas où un client peut passer une commande contenant plusieurs produits.

COMMANDE(Numcom : Numérique, Numcli : Numérique, Datecom : Date, Datelivrai : Date) ;
LIGNE_COMMANDE (Numprod : Numérique , Numcom : Numérique ,Qtecom : Numérique) ;
PRODUIT (Numprod : Numérique, Desig : Texte (50), PU : Numérique, NumFor :Numérique QteStock : Numérique) ;

- Augmenter de 20% le prix des produits dont la quantité stockée est inférieure à 10.

Suite de TP N°1

Ecrire les requêtes SQL qui permettent de lister :

- tous les noms des clients dont le prénom est soit ‘Hamid’ soit ‘Mustapha’.
- toutes les commandes dont la date de commande est entre ‘01/06/2014’ et ‘01/10/2014’ classées dans l’ordre croissant des numéros de commandes.
- les noms des clients qui ont commandé au moins un produit de prix supérieur à 3000DH.
- les numéros des clients qui n’ont pas commandé le produit N°1.
- recherche des clients qui n'ont pas commandé depuis plus de 30 jours
- Les numéros et les noms des clients qui ont commandé tous les produits
- Les numéros des produits qui ont été commandés par le client N°1

Les vues

- **une vue est une table virtuelle résultat d'une requête**
- **rôle d'une vue**
  - réduire la complexité syntaxique des requêtes
  - définir les schémas externes.
  - définir des contraintes d'intégrité.
  - définir un niveau additionnel de sécurité en restreignant l'accès à un sous ensemble de lignes et/ ou de colonnes.

Les vues

création d'une vue de schéma externe

CREATE [OR REPLACE] [FORCE | NO FORCE]
VIEW nom de table [(liste de nom de colonne)]
AS requête [WITH CHECK OPTION | WITH READ ONLY] ;

Les vues

- FORCE VIEW permet de créer des vues lorsque la table ou les tables utilisées ne sont pas disponibles dans votre environnement, alors la Vue sera INVALIDE mais existante.
- Avec NOFORCE (valeur par défaut), si les tables n'existent pas, la vue n'est pas créée.
- Les vues créées avec l'option WITH READ ONLY peuvent être interrogées mais aucune opérations DML (UPDATE, DELETE, INSERT) peut être effectuées sur la Vue.
- L'option WITH CHECK OPTION ou WITH CHECK OPTION CONSTRAINT crée une contrainte de vérification sur la vue à partir de la clause WHERE.
- Les vues créées avec l'option WITH CHECK OPTION CONSTRAINT empêche toutes mises à jour de la Vue si les conditions de la clause WHERE ne sont pas respectées. (Visible dans USER_CONTRAINTS en type V).
- L'option OR REPLACE permet de changer la définition d'une vue sans faire un DROP/CREATE. L'avantage de l'option OR REPLACE, c'est la non suppression des privilèges accordés à la vue, cependant les objets dépendant de la vue deviennent invalides et doivent être compilés.

Les vues

- **création de la vue correspondant aux vols qui partent de Rabat**

```
CREATE VIEW vols_d_Rabat
AS SELECT no_VOL, V_a, H_d, H_a
FROM vols
WHERE V_d = 'Rabat' ;
```

- **interroger une vue**
  - interrogation de la vue correspondant aux vols qui partent de Rabat

```
SELECT * FROM vols_d_Rabat ;
```

- **supprimer une vue**

```
DROP VIEW nom_de_vue ;
```

  - suppression de la vue correspondant aux vols qui partent de Rabat

```
DROP VIEW vols_d_Rabat ;
```

Les vues – Exemples

- **pour connaître les pilotes qui assurent le plus grand nombre de vols**

```
CREATE VIEW nbre_de_vols_par_pil
AS SELECT no_PIL, count(*) AS NBR_VOLS
FROM vols V
WHERE no_PIL IS NOT NULL
GROUP BY no_PIL ;

SELECT no_PIL FROM nbre_de_vols_par_pil
WHERE NBR_VOLS =
    (SELECT max( NBR_VOLS)
     FROM nbre_de_vols_par_pil) ;
```

Donner une solution
En utilisant rownum

Les vues : mise à jour

- **opérations sur les vues**
  INSERT
  UPDATE
  DELETE
- **restrictions : Ces instructions ne s'appliquent pas aux vues qui contiennent :**
  - une jointure
  - un opérateur ensembliste : UNION, INTERSECT, MINUS
  - une clause GROUP BY, CONNECT BY, ORDER BY ou START WITH
  - la clause DISTINCT, une expression ou une pseudo-colonne dans la liste de sélection des colonnes.

Requêtes avec vues

- **création de la vue pour la personne qui définit les vols**

```
CREATE VIEW def_vols
AS SELECT no_VOL, V_d, D_d, V_a, D_a
FROM vols
WHERE no_VOL IS NULL AND no_PIL IS NULL ;
```

- **définir un nouveau vol**

```
INSERT INTO def_vols VALUES
('999', 'Agadir', to_date('01/05/07 10 :30', 'DD/MM/RR HH :MI'),
'Rabat', to_date('01/05/07 10 :30', 'DD/MM/RR HH :MI')) ;
```

Requêtes avec vues

- **supprimer un vol non affecté**

```
DELETE FROM def_vols

WHERE no_VOL = ’V998’ ;
```

- **modifier un vol non affecté**

```
UPDATE def_vols

SET D_d= D_d + 1 / 24, D_a= D_a + 1 / 24 WHERE

no_VOL = ’V998’ ;
```

- **connaître les vols non affectés**

```
SELECT * FROM def_vols ;
```

Requêtes avec vues

- **création de la vue pour la personne qui affecte un avion et un pilote à un vol**

```
CREATE VIEW affect_vols

AS SELECT no_VOL, no_AV, no_PIL

FROM vols ;
```

- **affecter un avion et un pilote à un nouveau vol**

```
UPDATE affect_vols

SET no_AV = 101, no_PIL = 5050

WHERE no_VOL = 'V999'

AND no_AV IS NUL AND no_PIL IS NULL ;
```

- **affecter un nouvel avion à un vol**

```
UPDATE affect_vols

SET no_AV = 202

WHERE no_VOL = 'V999' ;
```

Requêtes avec vues

- **permuter l'affectation des pilotes de 2 vols**

```
UPDATE affect_vols A1
SET no_PIL =
    ( SELECT no_PIL
      FROM affect_vols A2
      WHERE
          (A1.no_VOL = 'V100' AND A2.no_VOL = 'V200')
          OR
         (A1.no_VOL = 'V200' AND A2.no_VOL = 'V100')
    )
WHERE no_VOL = 'V100' OR no_VOL = 'V200' ;
```

Les vues : contrôle de mise à jour

- **vérification des contraintes de domaine (interdiction des valeurs inconnues)**

```
CREATE VIEW a_avions
AS SELECT * FROM avions
WHERE
    no_AV > 0
    AND CAP > 1
    AND NOM_AV IN ('Airbus', 'Boeing', 'Caravelle')
WITH CHECK OPTION ;
```

Requêtes avec vues

- **Vérification des contraintes de domaine (autorisation des valeurs inconnues)**

```
CREATE VIEW aa_avions
AS SELECT * FROM avions
WHERE
    no_AV > 0
    AND (CAP IS NULL OR CAP > 1)
    AND (NOM_AV IS NULL OR IN ('Airbus', 'Boeing',
    'Caravelle'))
WITH CHECK OPTION ;
```

Requêtes avec vues

- **Contraintes de référence**
  1) valider l'insertion dans la table référençant
  2) valider la suppression dans la table référencée

règle d'adéquation : les insertions et les suppressions se font toujours au travers des vues

Requêtes avec vues

- **Exemple : expression de clés étrangères de la relation vols**
  - validation des insertions dans vols

```
CREATE VIEW a_avions
AS SELECT * FROM vols
WHERE
    no_AV > 0
    AND (no_PIL IS NULL OR no_PIL IN(SELECT no_PIL FROM
    pilotes))
    AND (NOM_AV IS NULL OR IN (SELECT NOM_AV FROM avions))
WITH CHECK OPTION ;
```

Requêtes avec vues

- **Exemple : expression de clés étrangères de la relation vols**
  - validation des suppressions dans avions et dans pilotes

```
CREATE VIEW d_avions
AS SELECT * FROM avions A
WHERE NOT EXISTS ( SELECT * FROM vols V WHERE
                   A.no_AV = V.no_AV) ;

CREATE VIEW d_pilotes
sAS SELECT * FROM pilotes P
WHERE NOT EXISTS ( SELECT * FROM vols V WHERE
                   P.no_PIL = V.no_PIL) ;
```

Requêtes avec vues

- **Expression de contraintes générales**
- **Exemple : empêcher l'affectation d'un même avion à deux vols différents dont les tranches horaires se chevauchent**

```
CREATE VIEW a_vols
AS SELECT * FROM vols V1
WHERE NOT EXISTS (  SELECT * FROM vols V2
    WHERE V1.no AV = V2.no AV
   AND NVL(V2.D_a, V2.D_d) >= NVL(V1.D_d, V1.D_a)
   AND NVL(V1.D_a, V1.D_d) >= NVL(V2.D_d, V2.D_a)
  )
WITH CHECK OPTION ;
```

SQL : langage de contrôle de données (LCD)

- **Sécurité des données**
  - confidentialité

    gestion des rôles et des utilisateurs

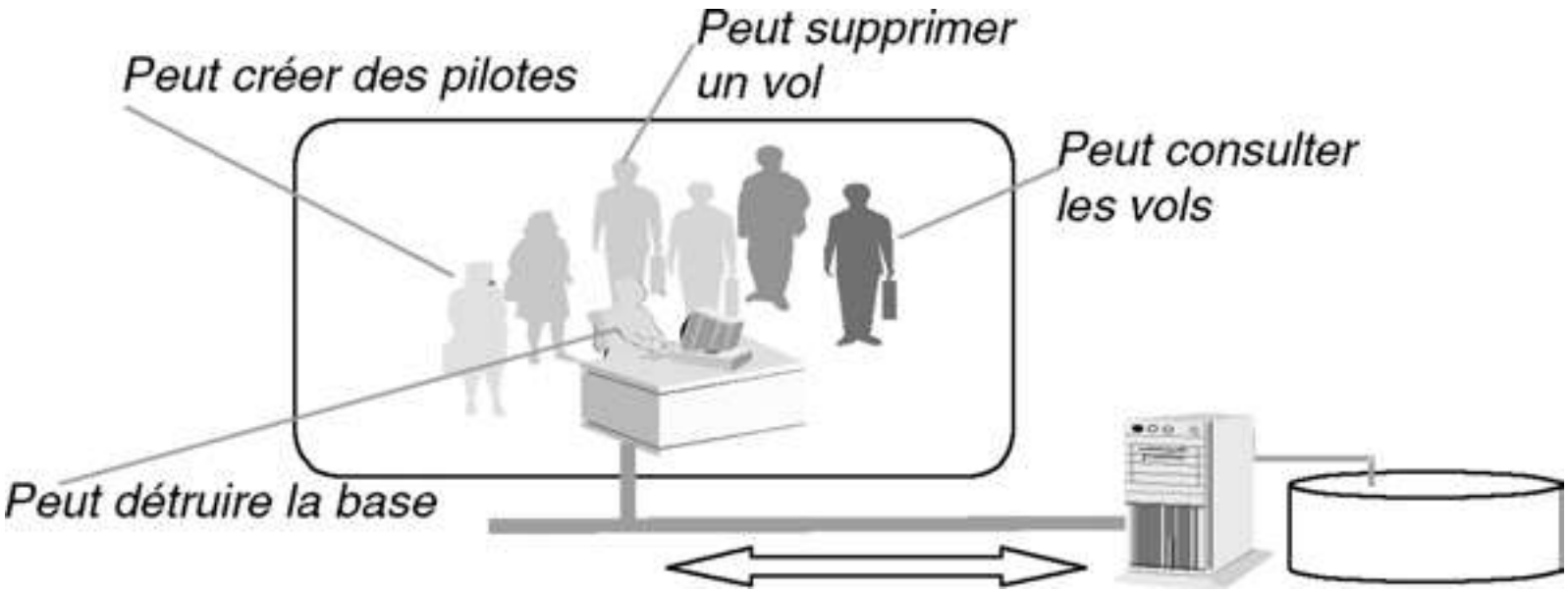
    attribution de privilèges aux rôles et aux utilisateurs

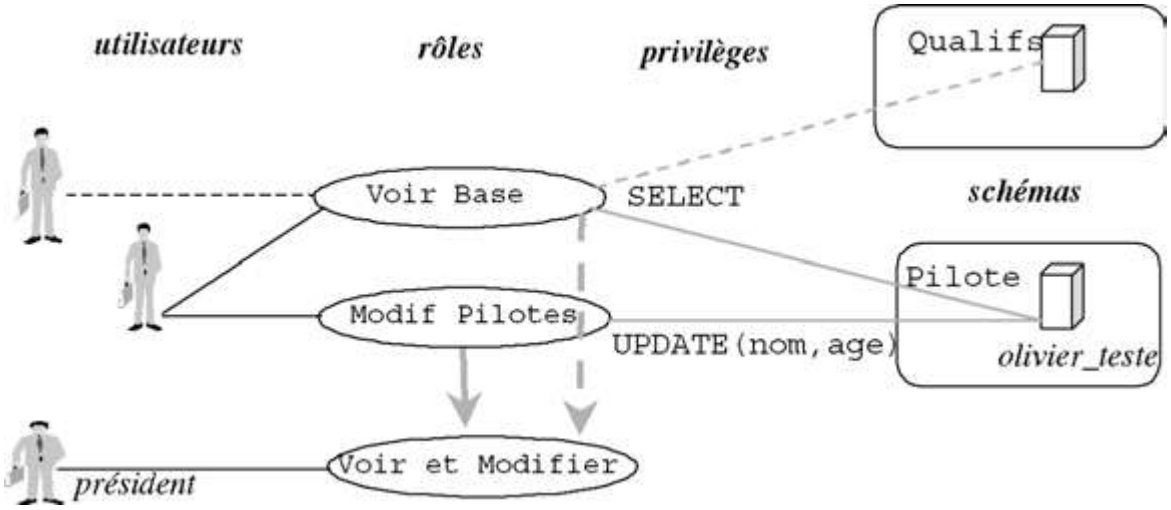
    définition de filtres (protection de données confidentielles, contrôle d'intégrité)
  - pérennité

    gestion des transactions
  - intégrité

    gestion des transactions

SQL : langage de contrôle de données (LCD)





SQL : langage de contrôle de données (LCD)

- **transaction : séquence d'opérations manipulant des données vérifient les propriétés suivantes :**
  - atomicité
  - cohérence
  - indépendance
  - permanence
- **contrôle des transactions :**
  - COMMIT : valide la transaction en cours
  - ROLLBACK : annule la transaction en cours

SQL : langage de contrôle de données (LCD)

- **On distingue typiquement six types de commandes SQL de contrôle de données :**
  - GRANT : autorisation d'un utilisateur à effectuer une action ;
  - DENY : interdiction à un utilisateur d'effectuer une action ;
  - REVOKE : annulation d'une commande de contrôle de données précédente ;
  - COMMIT : validation d'une transaction en cours ;
  - ROLLBACK : annulation d'une transaction en cours ;
  - LOCK : verrouillage sur une structure de données.

Gestion des utilisateurs et des privilèges

- **Rôles et privilèges sont définis pour sécuriser l'accès aux données de la base**
- **Ces concepts sont mis en oeuvre pour protéger les données en accordant (ou retirant) des privilèges a un utilisateur ou un groupe d'utilisateurs**
- **Un rôle est un regroupement de privilèges. Une fois créé il peut être assigné à un utilisateur ou à un autre rôle**
- **Les privilèges sont de deux types**
  - ***Les privilèges de niveau système***

    Qui permettent la création, modification, suppression, exécution de groupes d'objets

  - ***Les privilèges de niveau objet***

    Qui permettent les manipulations sur des objets spécifiques

Gestion des utilisateurs et des privilèges

- **création de rôle**

  CREATE ROLE nom-de-rôle [IDENTIFIED BY mot-de passe] ;

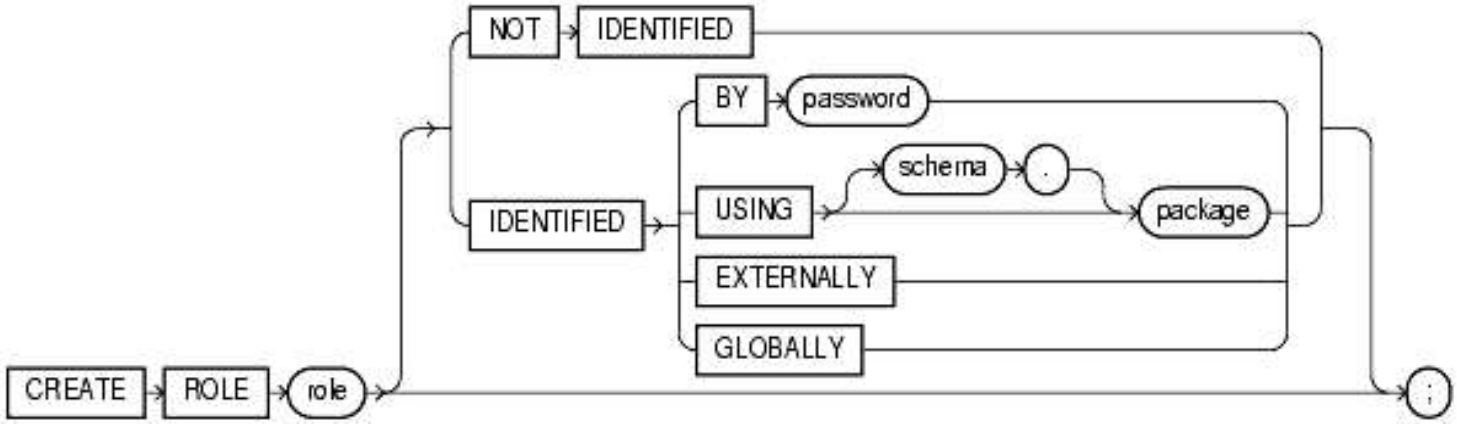
- **ajout, modification, suppression de mot de passe**

  ALTER ROLE nom-de-rôle [IDENTIFIED BY mot-de passe] ;

- **suppression de rôle**

  DROP ROLE nom-de-rôle ;

Créer des rôles et leur assigner des privilèges



- role représente le nom du rôle
- NOT IDENTIFIED (défaut) indique qu'aucun mot de passe n'est nécessaire pour activer le rôle
- IDENTIFIED BY password indique qu'un mot de passe est nécessaire pour activer le rôle
- IDENTIFIED USING package indique qu'un package va être utilisé pour fixer les droits de l'utilisateur
- IDENTIFIED EXTERNALLY indique que l'autorisation provient d'une source externe (S.E.)
- IDENTIFIED GLOBALLY pour un user GLOBAL géré par exemple par Enterprise Directory Service (système d'annuaire).

Exemples

- CREATE ROLE comptabilite ;
- GRANT SELECT, INSERT, UPDATE, DELETE ON LUS.COMMANDE TO comptabilite ;
- GRANT SELECT, INSERT, UPDATE, DELETE ON LUS.LIGNE_COMMANDE TO comptabilite ;
- GRANT SELECT, INSERT, UPDATE, DELETE ON LUS.JOURNAL TO comptabilite ;

Quelques rôles prédéfinis

Rôle	Privilèges
CONNECT	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, **CREATE SEQUENCE**, **CREATE SESSION**, CREATE SYNONYM, **CREATE TABLE**, CREATE VIEW
RESOURCE	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, **CREATE PROCEDURE**, CREATE SEQUENCE, CREATE TABLE, **CREATE TRIGGER**, CREATE TYPE
DBA	Tous les privilèges système avec WITH ADMIN OPTION
EXP_FULL_DATABASE	Privilèges requis pour des exportations : SELECT ANY TABLE, BACKUP ANY TABLE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE... avec les rôles EXECUTE_CATALOG_ROLE et SELECT_CATALOG_ROLE

Activation d'un rôle (SET ROLE)

```
SET ROLE
{ nomRôle [IDENTIFIED BY motdePasse] [,nomRôle [IDENTIFIED BY
MotdePasse]]…
| ALL [EXCEPT nomRôle [,nomRôle]…]
| NONE } ;
```

- `IDENTIFIED` indique le mot de passe du rôle.
- `ALL` active tous les rôles (non identifiés) accordés à l'utilisateur qui exécute la commande.

Cette activation n'est valable que dans la session courante. La clause `EXCEPT` permet d'exclure des rôles accordés à l'utilisateur (mais pas via d'autres rôles) de l'activation globale.

- `NONE` désactive tous les rôles dans la session courante (rôle `DEFAULT` inclus).

Gestion des utilisateurs et des privilèges

- **création d'utilisateur**

  CREATE USER nom-d'utilisateur [IDENTIFIED BY mot-de passe] ;

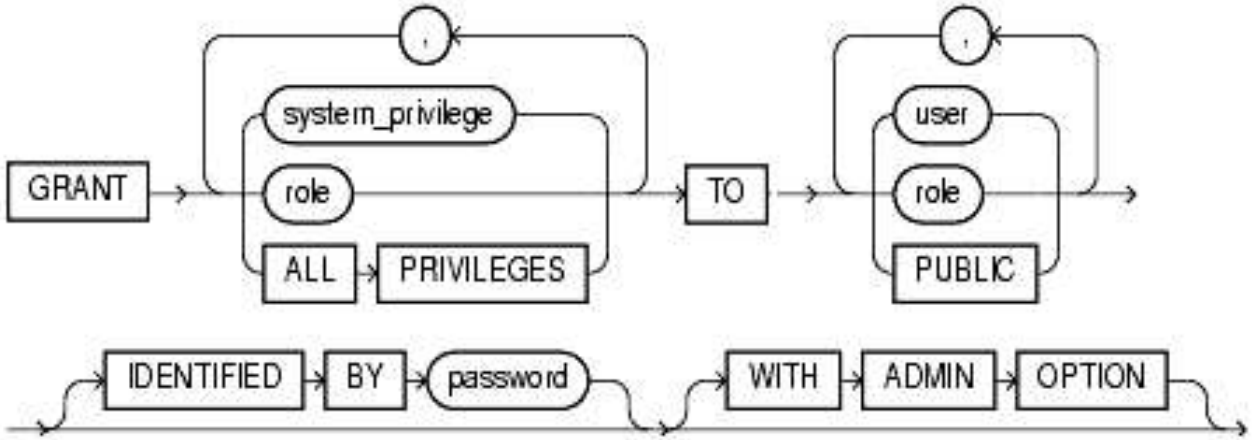
- **ajout, modification, suppression de mot de passe**

  ALTER USER nom-d'utilisateur [IDENTIFIED BY mot-de passe] ;

- **suppression de rôle**

  DROP USER nom-d'utilisateur ;

Assigner des privilèges système à un utilisateur



- **systeme_privilege** représente un privilège système
- **role** représente un rôle préalablement créé
- **ALL PRIVILEGES** représente tous les privilèges système (à l'exception de SELECT ANY DICTIONARY)
- **user** représente le nom de l'utilisateur qui doit bénéficier du privilège
- **PUBLIC** assigne le privilège à tous les utilisateurs
- **WITH ADMIN OPTION** assigne à l'utilisateur le droit d'assigner, de retirer, de modifier et de supprimer à son tour les privilèges du rôle reçus

Assigner des privilèges système à un utilisateur

- **attribution de privilèges**

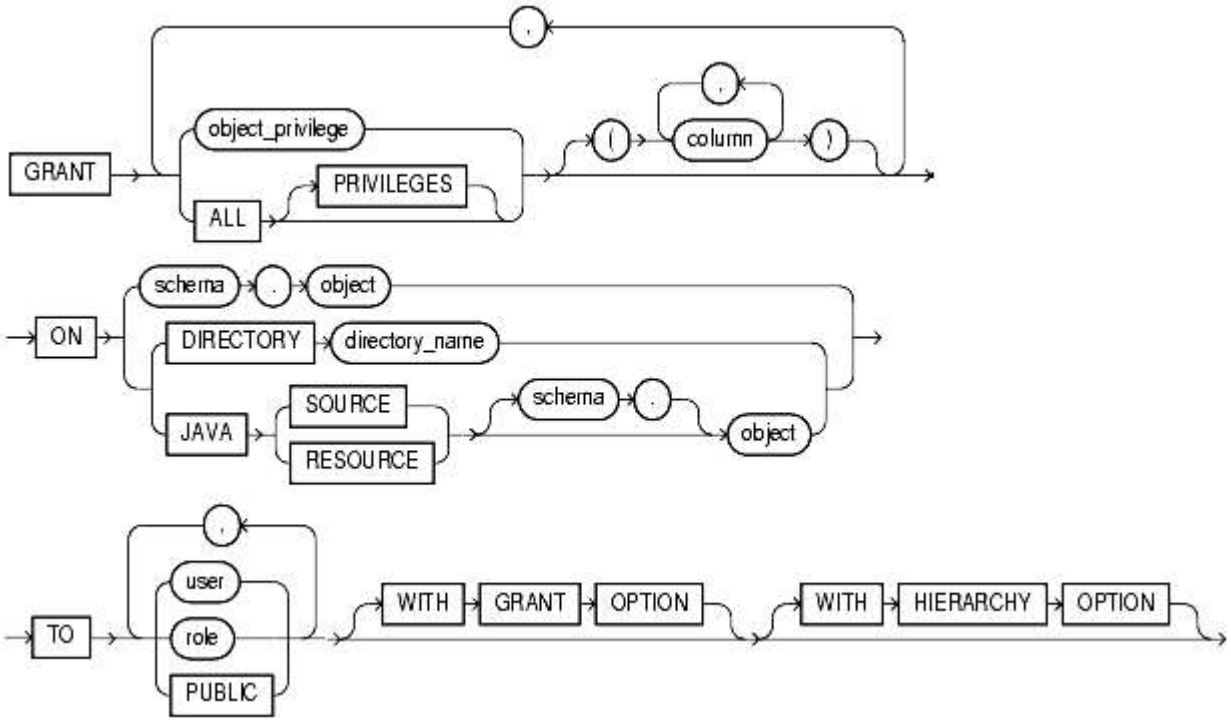
  GRANT systeme-privileges | ALL [privileges]

  TO liste-roles-utilisateurs | PUBLIC

  [WITH ADMIN OPTION] ;

- **systeme-privileges :**
  - CREATE ROLE
  - CREATE SEQUENCE
  - CREATE SESSION
  - CREATE SYNONYM
  - CREATE PUBLIC SYNONYM
  - CREATE TABLE
  - CREATE USER
  - CREATE VIEW

Assigner des privilèges objet à un utilisateur



Assigner des privilèges objet à un utilisateur

- object_privilege représente un privilège objet
- role représente un rôle préalablement créé
- ALL PRIVILEGES représente tous les privilèges assignés à l'exécuteur de l'instruction
- column représente le nom de colonne d'une table
- schema représente le nom d'un schéma
- object représente le nom d'un objet du schéma
- directory_name représente le nom d'une directory
- JAVA SOURCE représente le nom d'une source Java
- JAVA RESOURCE représente le nom d'une ressource Java
- WITH GRANT OPTION assigne à l'utilisateur de droit d'assigner à son tour le privilège reçu à un autre utilisateur
- (WITH GRANT OPTION s'applique à un utilisateur ou à PUBLIC, mais pas à un rôle)
- WITH HIERARCHY OPTION assigne le privilèges aux sous-objets

Assigner des privilèges objet à un utilisateur

- **attribution de privilèges sur des objets oracle**

  GRANT liste-droits

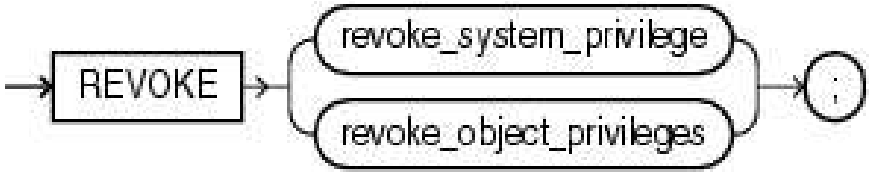
  ON nom-composant

  TO liste-roles-utilisateurs

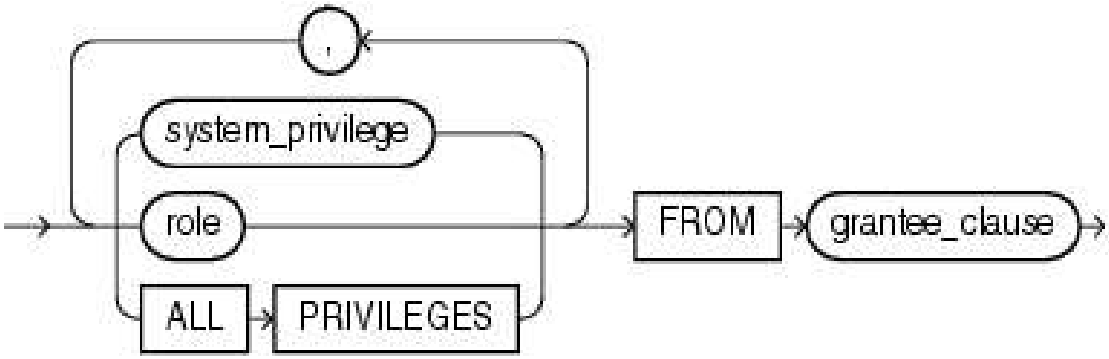
  [WITH GRANT OPTION] ;

- **liste-droits :**
  - SELECT -INSERT
  - UPDATE -DELETE
  - ALTER -REFERENCES
  - ALL [PRIVILEGES]

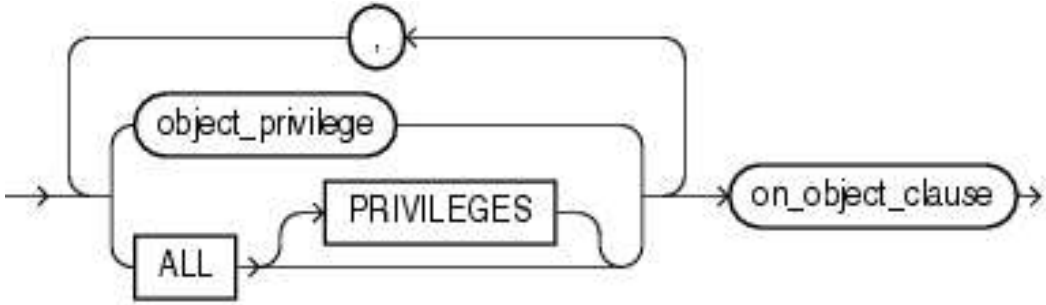
Annulation des privilèges

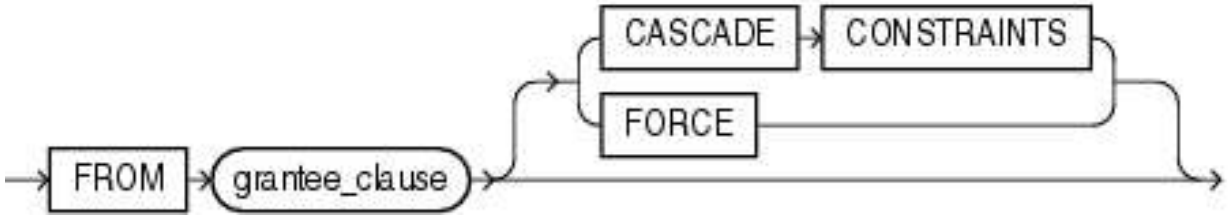


revoke_system_privileges::=



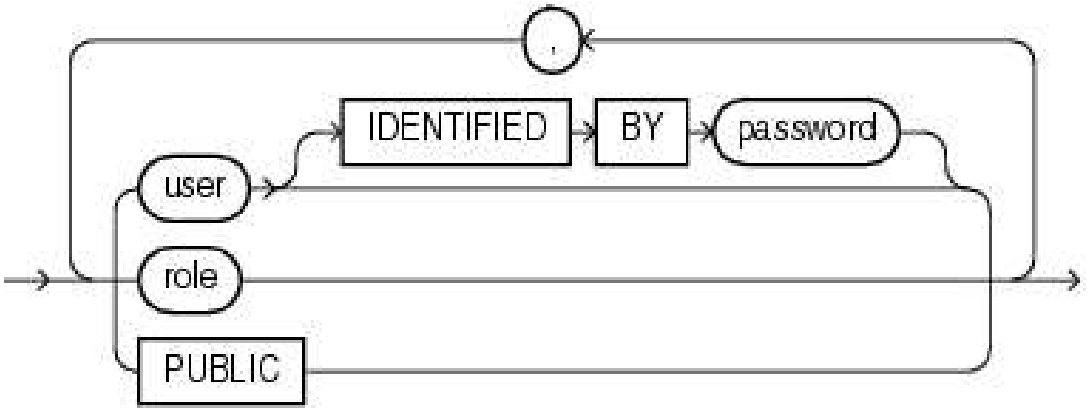
revoke_object_privileges::=



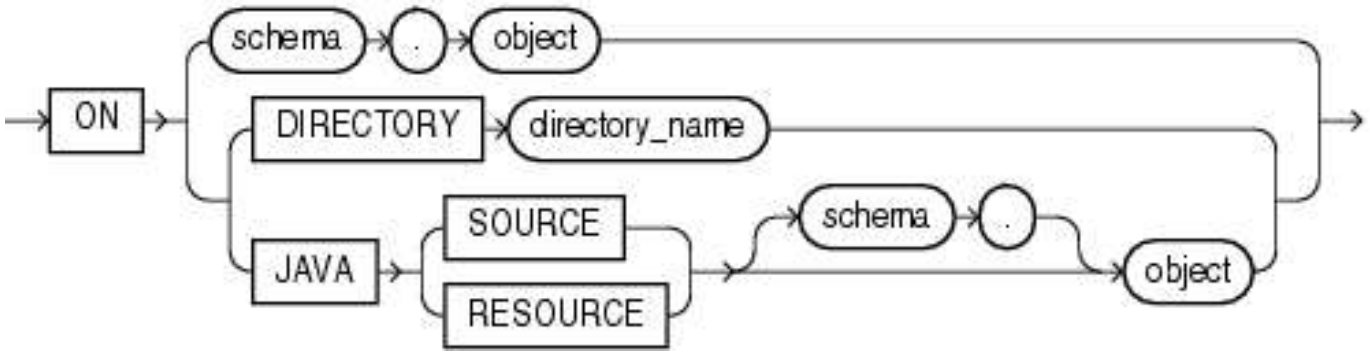


Annulation des privilèges

grantee_clause::=



on_object_clause::=



Différence entre GRANT, DENY & REVOKE

- Grant est utilisé pour autoriser spécifiquement l'accès à un objet.
- Deny est utilisé pour empêcher spécifiquement l'accès à un objet.
- Revoke est utilisé pour supprimer l'accès spécifiquement accordé ou refusé à un objet.
- La différence entre GRANT, DENY et REVOKE peut également être indiquée dans les mots ci-dessous
- Lorsque l'autorisation est accordée, l'utilisateur ou le rôle reçoit l'autorisation d'effectuer un acton, tel que la création d'une table.
- L'instruction DENY refuse l'autorisation sur un objet et empêche le principal d'obtenir l'autorisation GRANT en fonction de l'appartenance à un groupe ou à un rôle.
- L'instruction DENY refuse l'autorisation sur un objet et empêche le principal d'obtenir l'autorisation GRANT en fonction de l'appartenance à un groupe ou à un rôle.
- L'instruction REVOKE supprime une autorisation précédemment accordée ou refusée.
- Les autorisations refusées à une portée plus élevée dans un modèle de sécurité

Remarque

- DENY n'existe pas en SQL. C'est une invention de Sybase / SQL Server...
- en SQL il n'existe que GRANT et REVOKE.

```
SELECT * FROM test
Le schéma courant est user1
```

Lorsque le schéma de l'objet n'est pas précisé (ici l'objet TEST), Oracle effectue la recherche de cet objet dans l'ordre suivant :

- Recherche d'un objet appartenant au schéma courant user1
- Recherche dans les synonymes du schéma courant
- Dans les synonymes `PUBLIC`

- Pour plus de détaille sur la création, activation, modification, suppression des rôles veuillez voir le lien suivant : https://oracle.developpez.com/guide/administration/adminrole/

Bloc anonyme, Procédures stockées et Déclencheurs

- Gestion des procédures et fonctions
- Métadonnées des procédures et fonctions
- Extensions au standard SQL (Délimiteur, contrôle du flux, boucles, Curseurs)
- Structure d'un déclencheur
- Types d'événements déclencheurs
- Propriétés d'un déclencheur
- Validation des données

Introduction

- **Problématique**
  - Comme en programmation, un code SQL est souvent répété.
  - Nécessité d'exécuter plusieurs requêtes pour une seule tâche.
- **Solution :**
  - Créer (comme en programmation):
    - Des fonctions
    - Des procédures
  - Déclencheurs
- **Conséquences:**
  - Allégement maxi des traitements sur le client
  - Structuration plus propre du code SQL

Introduction

- **Une procédure stockée**
  - compilé une fois pour toutes
  - peut être exécuté plusieurs fois.

  Implique une vitesse exécution accrue.
- **Un déclencheur**
  - est un type spécifique de procédure stockée qui **n'est pas appelé directement par un utilisateur**.
  - Lorsque le déclencheur est créé, il est défini de façon à se déclencher lorsqu'un certain type de **modification de données** est effectué dans une table.
- **Dans ce cours nous nous intéressons au langage PL/SQL**

Introduction

- Le PL/SQL (Procedural Language extensions to SQL) est un langage procédural d'ORACLE.
- L'intérêt du PL/SQL est de pouvoir dans un même traitement combiner la puissance des instructions SQL et la souplesse d'un langage procédural.
- Dans l'environnement SQL, les ordres du langage sont exécutés les uns à la suite des autres.
- Dans l'environnement PL/SQL, les ordres SQL et PL/SQL sont regroupés en blocs; un bloc ne demande qu'un seul transfert exécution de l'ensemble des commandes contenues dans le bloc.

Utilisation de PL/SQL

- **Le PL/SQL peut être utilisé sous 3 formes :**
  - un bloc de code (aussi appelé bloc anonyme) , exécuté comme une unique commande SQL, via un interpréteur standard (SQLplus ou iSQL*Plus)
  - un fichier de commande PL/SQL
  - un programme stocké (procédure, fonction, trigger)

Blocs

- **Un programme est structuré en blocs d'instructions de 3 types :**
  - procédures ou bloc anonymes,
  - procédures nommées,
  - fonctions nommées.
- **Un bloc peut contenir d'autres blocs.**

Structure d'un bloc PL/SQL

DECLARE

Déclaration des variables locales, constantes, exceptions, curseurs, modules locaux

BEGIN

Commandes exécutables: Instructions SQL et PL/SQL. Possibilité de blocs imbriqués.

EXCEPTION

code de gestion des erreurs

!!! Seuls BEGIN et END sont obligatoires

END;

/

- **Les blocs se terminent par un ;**
- **Sous SQL*PLUS, il faut taper une dernière ligne contenant « / » pour compiler un bloc pl/sql**

Structure d'un bloc PL/SQL

- **Un bloc PL/SQL peut contenir:**
  - Toute instruction du LMD (SELECT, INSERT, UPDATE, DELETE)
  - Les commandes de gestion des transactions (COMMIT, ROLLBACK, SAVEPOINT)
- **Les sections DECLARE et EXCEPTION sont optionnelles.**
- **Chaque instruction se termine par ;**
- **Les blocs peuvent être imbriqués**
  - Les sous blocs ont la même structure que les blocs.
  - Une variable est visible dans le bloc où elle est déclarée et dans tous ses sous-blocs.
  - Si une variable est déclarée dans un premier bloc et aussi dans un sous bloc, la variable du bloc supérieur n'est plus visible dans le sous-bloc.

Affichage

- Pour afficher le contenu d'une variable, les procédures DBMS_OUTPUT.PUT() et DBMS_OUTPUT.PUT_LINE() prennent en argument une valeur à afficher ou une variable dont la valeur est à afficher.
- Par défaut, les fonctions d'affichage sont désactivées Il convient de les activer avec la commande SET SERVEROUTPUT ON.
- Exemple

```
SET SERVEROUTPUT ON
DECLARE
c varchar2(15) := 'Bonjour!';
BEGIN DBMS_OUTPUT.PUT_LINE(c);
END;
/
```

```
SET SERVEROUTPUT ON

DECLARE
c varchar2(15) := 'Bonjour !';
BEGIN
    DBMS_OUTPUT.PUT(c);
    DBMS_OUTPUT.NEW_LINE ;
END;
/
```

Structure d'un bloc PL/SQL

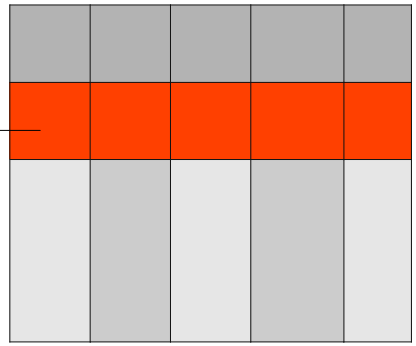
- **Variables**
  - Identificateurs Oracle :
    - 30 caractères au plus
    - commence par une lettre
    - peut contenir des lettres, des chiffres, _, $ et #
  - Pas sensible à la casse
  - Doivent être déclarées avant d'être utilisées
- **Commentaires**

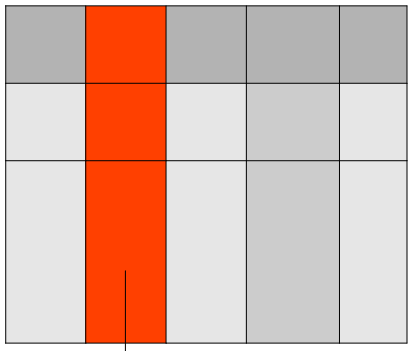
```
-- Pour un commentaire sur une ligne
/* Pour commentaire sur plusieurs
   lignes */
```

DÉCLARATION DES VARIABLES

Variables

- Scalaires
- Composites
  - Enregistrements (zone subdivisée)
  - Table PL/SQL
    Table à une dimension (une colonne)
    Appelée aussi une COLLECTION





Structure d'un bloc – types de variables

- **Types de variables**
  - Les types habituels correspondants aux types SQL2 ou Oracle : integer, varchar,...
  - Types composites adaptés à la récupération des colonnes et lignes des tables SQL : %TYPE, %ROWTYPE
  - Type référence : REF

Déclaration des variables scalaires

- **Déclaration par copie du type de donnée:**
  - La variable fait référence au même type qu'une colonne ou au même type qu'une autre variable .

Syntaxe:

Nom-de-variable nom_table.nom-colonne%type ;

ou

Nom-de-variable1 Nom-de-variable2%type ;

Exemple:

```
DECLARE
   Nom_client   client.nom%type ;
   X number (10,3) ;
   Y    X%type ;
```

Structure d'un bloc – L'attribut %ROWTYPE

- L'attribut % ROWTYPE permet de déclarer une variable de même type que l'enregistrement

  Syntaxe:

  Nom_de_variable nom_table %ROWTYPE ;

  Exemple:

  DECLARE enrg_empl empl%ROWTYPE ;
- Les éléments de la structure sont identifiés par:

  nom_variable . nomcolonne

Utilisation d'une variable de type Record

- Un record permet de définir des types composites

  Syntaxe :

  TYPE nom_record IS RECORD

  (nom_ch 1 type ,

  nom_ch2 type,

  .....);

- Déclaration d'une variable de ce type: nom_variable nom_record

  Exemple : DECLARE

  TYPE enrg_empl IS RECORD

  (nom employe. nomempl%TYPE,

  salaire employe . sal %TYPE);

  e enrg_empl ;

Déclaration des variables scalaires

- **La partie déclarative d'un bloc PL/SQL peut comporter 3 types de déclarations:**
  - Déclarations des variables, constantes,
  - Déclaration des curseurs ,
  - Déclaration des exceptions.

1. **Variables ou constantes :**

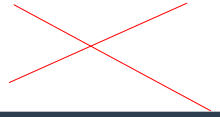
- **Nom-de-variable [CONSTANT] TYPE [NOT NULL] [:= | DEFAULT Expression] ;**

  Expression: peut être une constante ou un calcul faisant éventuellement référence à une variable précédemment déclarée

Exemple:

```
DECLARE
    Nom_duclient char (30);
    X number DEFAULT 10 ;
    PI constant number(7,5):=3.14159;
```

- Déclarations multiples interdites : i, j integer;



Déclaration des variables scalaires

- **Valorisation des variables : 3 possibilités:**
  - Par l'opérateur d'affectation **:=**
  - Directive **INTO** de la requête **SELECT**
  - Par le traitement d'un **curseur** dans la section **BEGIN**.
- **Par l'opérateur d'affectation :=**
  - Nom_variable**:=** expression
  - **Exemple** :

```
X := 0;
Y := ( X+5) * Y ;
```

- **Par la clause Select....Into**
  - **Exemple**:

```
DECLARE
    Vref NUMBER(5,0);
    Vprix produit.PU%type ;
BEGIN
   SELECT REF, PU INTO Vref, Vprix
   FROM Produit WHERE REF= 1 ;
   dbms_output.put_line('NUMPROD='||Vref||'  PU='||vprix);
END;
```

SELECT …INTO …

- **Instruction SELECT expr1, expr2, …INTO var1, var2, …**
  - Met les valeurs récupérées de la BD dans une ou plusieurs variables var1, var2, …
  - SELECT ne doit retourner qu'une seule ligne
  - Avec Oracle il n'est pas possible d'inclure un SELECT sans INTO dans le corps d'un bloc pl/sql.
- **Dans le cas d'un SELECT qui retourne plusieurs lignes, voir la suite du cours sur les curseurs.**

Déclaration des variables composites
Les collections VARRAY – définition & allocation

- **Les types tableau peuvent être définis explicitement en utilisant la syntaxe suivante**

  Syntaxe :

  TYPE nom_type IS {VARRAY | VARYING ARRAY} (size_limit) OF typeElements [NOT NULL];
  - nom_type : est le nom du type tableau crée par cette instruction
  - taille : est le nombre maximal d'éléments du tableau
  - typeElements : est le type des éléments qui vont être stockés dans le tableau.

  Exemple

```
TYPE numberTab IS VARRAY (10) OF NUMBER;
```

- **Allocation d'un tableau**

```
DECLARE
    TYPE numberTab IS VARRAY (10) OF NUMBER;  -- définition de type numberTab
    tab numberTab ;   -- déclaration de tableau tab
BEGIN
    tab := numberTab ( ) ;  -- allocation de l'espace mémoire avec constructeur de type
    /* utilisation du tableau */
END;
```

Déclaration des variables composites
Les collections VARRAY - Dimensionnement

```
DECLARE
    TYPE numberTab IS VARRAY (10) OF NUMBER;
    tab numberTab ;
BEGIN
    tab := numberTab ( ) ;
    tab.EXTEND( 4 ) ;
    /* utilisation du tableau */
END;
/
```

- **Dans cet exemple, tab.EXTEND(4) permet par la suite d'utiliser les éléments du tableau t(1), t(2), t(3) et t(4). Il n'est pas possible "d'étendre" un tableau à une taille supérieure à celle spécifie lors de la création du type tableau associé.**
- **On peut aussi utilisé le constructeur avec des valeurs d'initialisations.**

```
    tab := numberTab (1,2,3,4) ;
```

Déclaration des variables composites
Les collections TABLE

- **Les types tableau peuvent aussi être définis explicitement par la syntaxe suivante :**
  TYPE nom_type IS TABLE OF type_des_valeurs INDEX BY type_des_clés ;
  - nom_type : est le nom du type tableau crée par cette instruction,
  - type_des_valeurs : est le type des éléments qui vont être stockés dans le tableau,
  - type_des_clés est BINARY_INTEGER ou VARCHAR(10) par exemple. Ce sont les clés du tableau associatif, elles doivent être uniques !
  - Les types admis doivent être numériques (BINARY_INTEGER ou PLS_INTEGER) ou alphabétique (VARCHAR(longueur), STRING, LONG ou VARCHAR2(longueur à préciser obligatoirement).
  - Si on veut utiliser d'autre type (dates), il faudra faire une conversion.

Les collections TABLE

Déclaration d'un tableau : nom_Tableau nom_Type ;

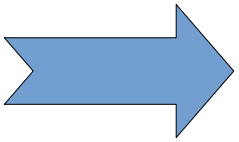
Exemple

```
declare
        -- collection de type table sans index by
        TYPE TYP_TAB  is table of  varchar2(100) ;
        -- collection de type table avec index by
        TYPE TYP_TAB_IND is table of number index by binary_integer ;

        tab1 TYP_TAB ;
        tab2 TYP_TAB_IND ;
 Begin
        tab1 := TYP_TAB('Lundi','Mardi','Mercredi','Jeudi' ) ;
        for i in 1..10 loop
                tab2(i):= i ;
        end loop ;
End;
```

Les collections TABLE

```
-- table de multiplication par 8 et par 9...
declare
   type tablemul is record ( par8   number, par9   number);
   type tabledentiers is table of tablemul index by binary_integer;
   ti    tabledentiers;
   i number;
begin
   for i in 1..10 loop
      ti(i).par9 := i*9 ;
      ti(i).par8:= i*8;
      dbms_output.put_line (i||'*8='||ti(i).par8||' '||i||'*9='||ti(i).par9 );
   end loop;
end;
```



```
1*8=8 1*9=9
2*8=16 2*9=18
3*8=24 3*9=27
4*8=32 4*9=36
5*8=40 5*9=45
6*8=48 6*9=54
7*8=56 7*9=63
8*8=64 8*9=72
9*8=72 9*9=81
10*8=80 10*9=90
```

Méthodes intégrées pour les collections

- Un constructeur de collection (constructeur) est une fonction définie par le système avec le même nom qu'un type de collection, qui renvoie une collection de ce type. La syntaxe d'un appel de constructeur est:

  collection_type ([value [, value]...])
- EXISTS - Utilisé pour déterminer si un élément spécifique d'une collection existe. EXISTS est utilisé avec des tables imbriquées.

  Tab.exists(valeur)
- COUNT - Retourne le nombre d'éléments actuellement contenus dans une collection sans inclure les valeurs NULL. Pour varrays count est égal à LAST. Pour les tables imbriquées, COUNT et LAST peuvent être différents en raison de valeurs supprimées dans les sites de données interstitielles dans la table imbriquée.

  Tab.count
- LIMIT - Utilisé pour VARRAYS pour déterminer le nombre maximum de valeurs autorisées. Si LIMIT est utilisé sur une table imbriquée, elle renverra une valeur nulle.

  Tab.LIMIT
- FIRST et LAST - Renvoie les plus petits et les plus grands numéros d'index pour la collection référencée. Naturellement, ils renvoient null si la collection est vide.

  Tab.First, Tab.Last

Méthodes intégrées pour les collections

- PRIOR et NEXT - Renvoie la valeur précédente ou suivante en fonction de la valeur d'entrée de l'index de collection. PRIOR et NEXT ignorent les instances supprimées d'une collection.

```
indice_suivant:=tab.next(indice);
```

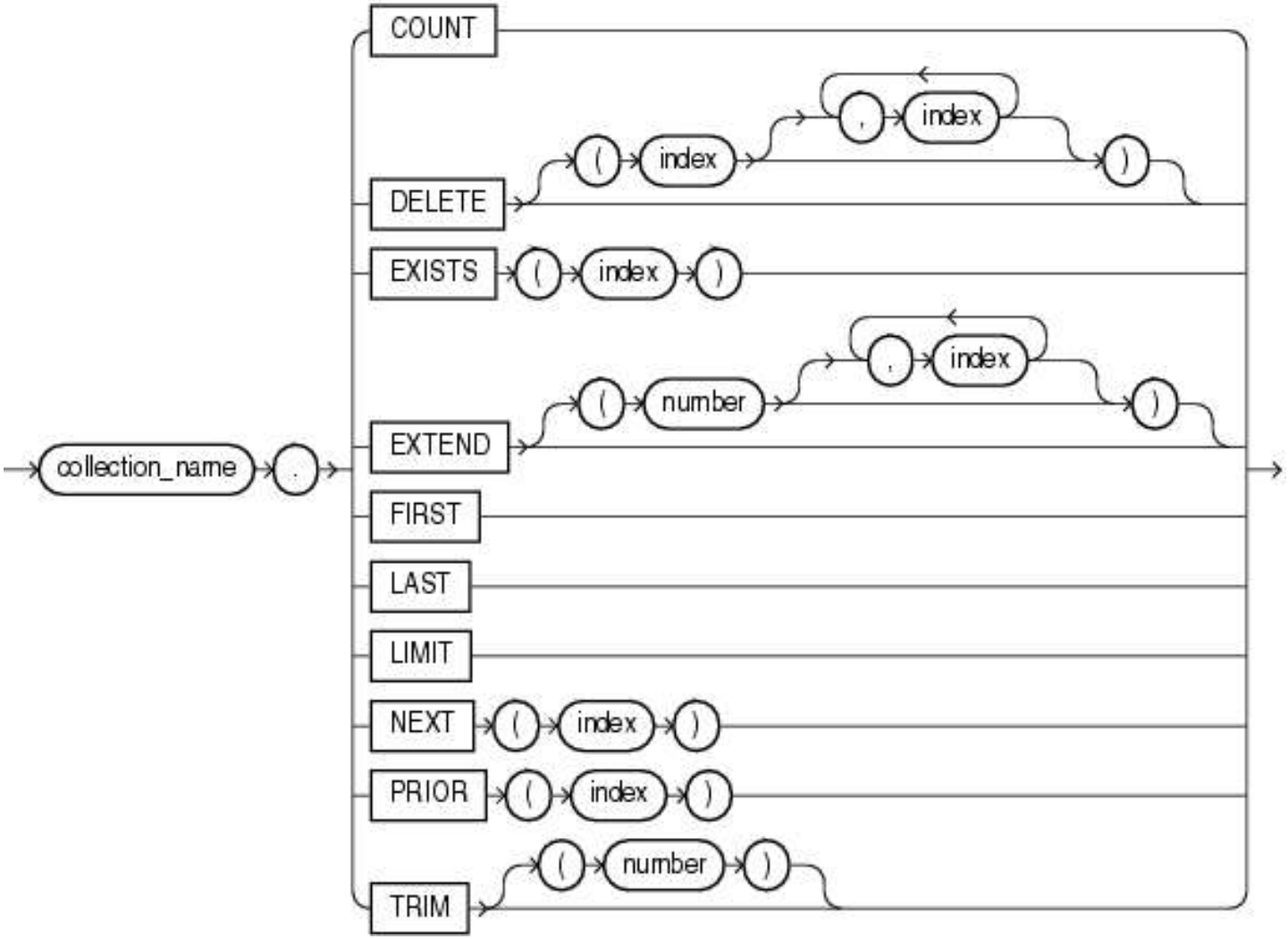
- EXTEND - Ajoute des instances à une collection. EXTEND a trois formes, EXTEND, qui ajoute une instance nulle, EXTEND (n) qui ajoute "n" instances NULL et EXTEND (n, m) qui ajoute n copies de l'instance "m" à la collection. Pour les formes de collections spécifiées non nulles, les formes un et deux ne peuvent pas être utilisées.

- DELETE - DELETE supprime les éléments spécifiés d'une table imbriquée ou d'une table

```
tab.Delete(indice)
```

- VARRAY. DELETE spécifié sans argument supprime toutes les instances d'une collection. Pour les tables imbriquées, seul DELETE (n) supprime la $n^{ième}$ instance et DELETE (n, m) supprime la $n^{ième}$ à travers les instances mth de la table imbriquée qui se rapportent à la fiche spécifiée.

Méthodes intégrées pour les collections



Exemple

```
i := tab.FIRST;
WHILE i IS NOT NULL LOOP
   traitement
  i := tab.NEXT(i);
END LOOP;
```

-
- Cette boucle s'arrête au dernier élément parce que NEXT(i) renvoie NULL au dernier i.

Collections & Exception

- **Les méthodes de collecte peuvent générer les exceptions suivantes:**
  - COLLECTION_IS_NULL - générer lorsque la collection référencée est nulle.
  - NO_DATA_FOUND - L'indice indique une instance nulle de la collection.
  - SUBSCRIPT_BEYOND_COUNT - L'indice spécifié dépasse le nombre d'instances de la collection.
  - SUBSCRIPT_OUTSIDE_LIMIT - L'indice spécifié est en dehors de la plage autorisée (généralement reçue des références VARRAY)
  - VALUE_ERROR - L'indice est nul ou n'est pas un nombre entier.

Comment parcourir un tableau associatif (table)?

- **Accées direct à une valeur en utilisant la clé**

  nom_Tableau(clé)
- **FIRST et LAST**
  - Un tableau associatif (comme toutes les autres collections en PL/SQL) a les propriétés FIRST et LAST auxquelles on accède en faisant **nom_tableau.FIRST** et **nom_tableau.LAST.**
  - FIRST renvoie **la première clé** issue d'un tri numérique ou alphabétique.
  - Pour parcourir le tableau, on peut donc faire une boucle comme ceci : **FOR i IN nom.FIRST..nom.LAST LOOP**
- **PRIOR et NEXT**

```
i := nom.FIRST;
WHILE i IS NOT NULL LOOP
      ...
   i := nom.NEXT(i);
END LOOP;
```

  - Cette boucle s'arrête au dernier élément parce que NEXT(i) renvoie NULL au dernier i.

Opérateurs PL/SQL

- Opérateurs logiques NOT, AND, OR
- Opérateurs arithmétiques + - * / ** (puissance)
- Opérateurs de comparaison =, >, !=, >, <=, >=, IS NULL, LIKE, BETWEEN
- Opérateurs de chaîne de caractères || (Concaténation)

Utiliser des parenthèses pour simplifier l'écriture et la lecture des expressions

Structures de contrôles

- Structure alternative
  - IF .... THEN.... END IF ;
  - IF .... THEN.... ELSE ... END IF ;
- Itérations
  - LOOP ..
  - FOR i.....
  - WHILE
- Branchement séquentiels
  - GOTO
  - NULL

Structures alternative

- **IF THEN**

```
IF condition THEN
    sequence_insts;
END IF ;
```

- **IF THEN ELSE**

```
IF condition THEN
    sequence_insts1;
ELSE
    sequence_insts2;
END IF ;
```

- **IF THEN ELSIF**

```
IF condition1 THEN
    sequence_insts1;
ELSIF condition2 THEN
    sequence_insts2;
ELSE
    sequence_insts3;
END IF ;
```

Structures alternative

- **Exemple:**

```
if var1>10 then
      var2:= var1+20;
elsif var1 between 7 and 8 then
            var2:= 2* var1;
  else
            var2:=var1*var1;
end if ;
```

Itérations

- **LOOP-EXIT-WHEN-END**

```
LOOP
    .....
    EXIT WHEN CONDITION ou EXIT
    .....
END LOOP
```

- **Exemple:**

```
i:= 1 ;
LOOP
    i:= i +1 ;
    EXIT WHEN i >= 10 ;
    ....
END LOOP
```

Itérations

- **WHILE-LOOP-END**

```
WHILE condition LOOP

    .....

END LOOP
```

- **Exemple:**

```
WHILE total<= 10000 LOOP

    ........

    SELECT salaire INTO S FROM employe WHERE.....

    ....

    total:= total + S ;

END LOOP
```

Itérations

- **FOR-IN-LOOP-END**

```
FOR compteur IN [ REVERSE] inf..sup LOOP

    .....

END LOOP
```

- **Exemple:**

```
FOR I IN 1..10 LOOP

    J:= J* 3 ;

    INSERT INTO T VALUES ( I, J );

END LOOP
```

```
declare
   i int;
begin
   FOR i IN REVERSE 1..10 LOOP
      DBMS_OUTPUT.PUT_LINE(i);
   END LOOP;
end;
```

NULL

- **Signifie «ne rien faire » , passer à l'instruction suivante**
- **Exemple:**

```
IF I>=10 THEN
    NULL ;
ELSE
    INSERT INTO T VALUES ( 'inférieur à 10',I );
END IF ;
```

Curseurs

- Définition d’un curseur.
- Déclaration d’un curseur.
- Utilisation d’un curseur.

Curseur - définition

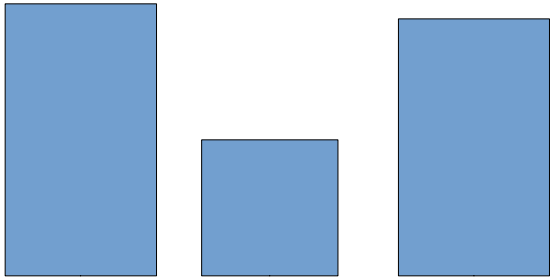
- **PL/SQL utilise des curseurs pour tous les accès à des informations de la base de données.**
- **Un curseur est pointeurs associés au résultat d'une requête.**
- **Deux types de curseurs:**
  - Implicite: créés automatiquement par Oracle pour chaque ordre SQL (select, update, delete, insert) .
  - Explicite: crée par le programmeur pour pouvoir traiter le résultat de requêtes retournant plus d'un tuple.

Étapes de gestion d'un curseur

déclarative

* 1 — La déclaration du curseur — **CURSOR** ...

Tables Oracle

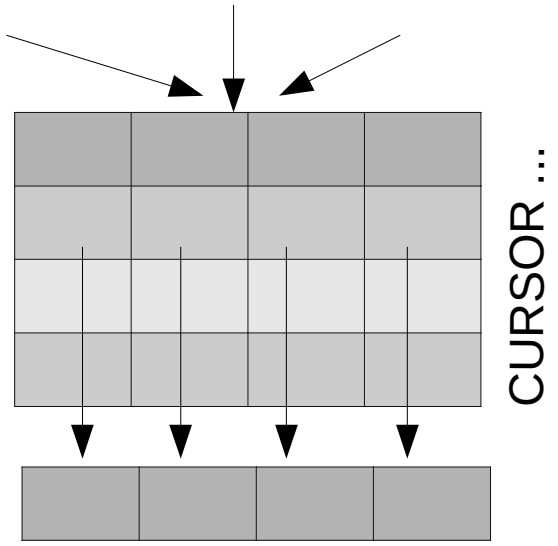


exécution

* 1 — Ouverture du curseur
Exécution de la requête SQL
Chargement en mémoire — **OPEN** ...

* n — Accès séquentiel aux lignes
via des variables
Gestion d'une boucle — **FETCH** ... **INTO**

* 1 — Fermeture du curseur
Libération de la zone mémoire acquise — **CLOSE** ...



Curseurs

- **Tous les curseurs ont des attributs que l'utilisateur peut utiliser**
  - %ROWCOUNT : nombre de lignes traitées par le curseur
  - %FOUND : vrai si au moins une ligne a été traitée par la requête ou le dernier fetch
  - %NOTFOUND : vrai si aucune ligne n'a été traitée par la requête ou le dernier fetch
  - %ISOPEN : vrai si le curseur est ouvert (utile seulement pour les curseurs explicites)

Curseur implicite

- **Les curseurs implicites sont tous nommés SQL**

```
DECLARE
    nb_lignes integer;
BEGIN
    delete from empl  where dept = 10;
    nb_lignes := SQL%ROWCOUNT;
    . . .
```

Curseur explicite

- **Les curseurs explicites**
  - sont utilisés pour traiter les select qui renvoient plusieurs lignes,
  - doivent être déclarés
- **Le code doit être utiliser explicitement avec les ordres OPEN, FETCH et CLOSE**
- **Le plus souvent on les utilise dans une boucle dont on sort quand l'attribut NOTFOUND du curseur est vrai.**

Curseurs explicite - déclaration

Syntaxe

- Déclaration d'un curseur sans paramètres :

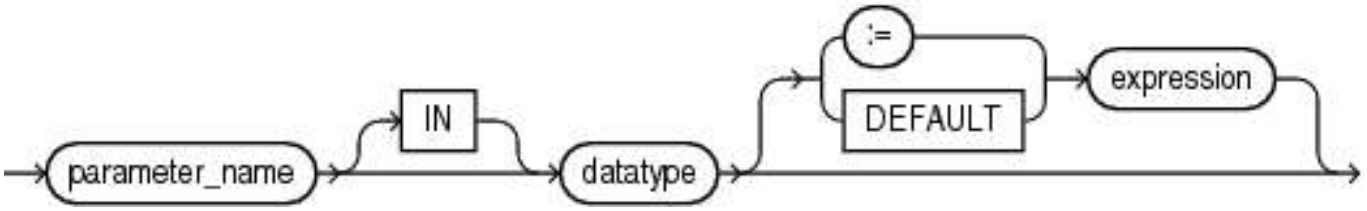
```
DECLARE
    CURSOR <nom_curseur> IS Requete_SELECT
```

Exemple: Déclarer un curseur qui permet de lister les employés de Kénitra

```
DECLARE
    v_ville employe.ville%type :='Kénitra'      --variable
    CURSOR employe_kenitra IS                   --curseur
    SELECT numemp, nomemp FROM employe WHERE ville=
    v_ville;
```

Curseurs explicite- déclaration

- Déclaration du curseur avec paramètres:
  CURSOR <nom_curseur> (para1, para2,.....) IS Requete_SELECT ;



- On doit fermer le curseur entre chaque utilisation de paramètres différents (sauf si on utilise « for » qui ferme automatiquement le curseur)

Exemple:

```
DECLARE
    CURSOR employe ( v IN VARCHAR2) IS
    SELECT numemp, nomemp FROM employe WHERE ville= v;
```

Curseur - Ouverture

- **L'ouverture du curseur se fait dans la section BEGIN du bloc par:**

  OPEN nom_curseur [(para1, para2 …)]

Exemple:

```
DECLARE
    CURSOR employe_rabat IS
    SELECT numemp, nomemp FROM employe WHERE ville= '
    Rabat';
BEGIN
    OPEN employe_rabat ;
    ……
    ……
END ;
```

Si aucune ligne n'est lue
⇒ OPEN ne déclenche pas d'exception
⇒ Le programme teste l'état du curseur après le FETCH

Curseur - accès aux données

- L'accès aux données se fait par la clause: **FETCH …. INTO**

  Syntaxe:

  FETCH nom_curseur INTO < var1, var2,.....>

- Le **FETCH** permet de récupérer un tuple de l'ensemble répnse associé au curseur et stocker les valeurs dans des variables.

- Pour traiter n lignes, il faut une boucle.

Fermeture d'un curseur

- **La fermeture du curseur se fait dans la section BEGIN du bloc par: CLOSE nom_curseur**
- **Exemple**

```
DECLARE
    CURSOR employe_rabat IS
    SELECT numemp, nomemp FROM employe WHERE ville= ' Rabat' ;
BEGIN
    OPEN employe_rabat
    .....
    CLOSE employe_rabat ;
END ;
```

Curseurs - Exemple

```
CREATE TABLE resultat ( nom varchar2, sal number )
DECLARE
    CURSOR employe_rabat IS
    SELECT numemp, sal FROM employe WHERE ville= ‘ Rabat’ ;
    nom employe.nomempl%TYPE ;
    salaire employe.sal %TYPE ;
BEGIN
    OPEN employe_rabat ;
    FETCH employe_rabat INTO nom, salaire ;
    WHILE employe_rabat%found LOOP
        IF salaire >3000 THEN
            INSERT INTO resultat VALUES ( nom, salaire) ;
        END IF ;
        FETCH employe_rabat INTO nom, salaire ;
    END LOOP;
    CLOSE employe_rabat ;
END ;
```

Curseurs explicite avec paramètres - Exemple

```
declare
    CURSOR cur_employe( v_ville varchar2) IS
    SELECT nomemp, sal FROM employe WHERE ville= v_ville ;
    v_nomemp   employe.nomemploye%type ;
    v_sal   employe.sal%type ;
begin
    open cur_employe('Kénitra');
    fetch cur_employe into v_nomemp, v_sal;
    while (cur_employe%found) loop
        DBMS_output.put_line(v_nomemp||' '||v_sal);
        fetch cur_employe into v_nomemp, v_sal;
    end loop;
end;
```

Curseurs explicite avec paramètres - Exemple

- **FOR LOOP remplace OPEN, FETCH et CLOSE.**
- **Lorsque le curseur est invoqué, un enregistrement est automatiquement créé avec les mêmes éléments de données que ceux définies dans l'instruction select.**

```
declare
    CURSOR cur_employe( v_ville varchar2) IS
    SELECT nomemp, sal FROM employe WHERE ville= v_ville ;
begin
    FOR enrg_e IN cur_employe(´rabat´) loop
        /* traitement*/
    END LOOP;
    FOR enrg_e IN cur_employe(´kenitra´) loop
        /* traitement*/
    END LOOP;
end;
```

Curseur
Exemple

- **Donner un programme PL/SQL utilisant un curseur pour calculer la somme des salaires des employés**

```
DECLARE
    CURSOR salaires IS
        select sal from emp where dept = 10;
    salaire numeric(8, 2);
    total numeric(10, 2) := 0;
BEGIN
    open salaires;
    loop
        fetch salaires into salaire;
        exit when salaires%notfound;
        if salaire is not null then
            total := total + salaire;
        end if;
    end loop;
    close salaires; -- Ne pas oublier
    DBMS_OUTPUT.put_line(total);
END;
```

Attention !

Type « rowtype » associé à un curseur

- **On peut déclarer un type « rowtype » associé à un curseur**

  Nom_enregistrement nom_cursuer%rowtype ;

Exemple :

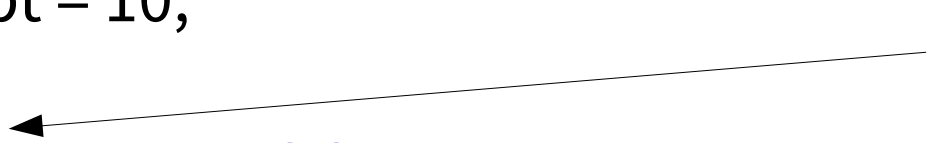
```
declare
    cursor c is
        select matr, nome, sal from emp;
        employe c%ROWTYPE;
begin
    open c;
    fetch c into employe;
    if employe.sal is null then ...
```

Boucle FOR pour un curseur

- For permet de simplifier la programmation car elle évite d'utiliser explicitement les instruction open, fetch, close.
- En plus elle déclare implicitement une variable de type «row» associée au curseur

Exemple

```
declare
    cursor c is select dept, nome from emp
    where dept = 10;
begin
    FOR employe IN c LOOP
        dbms_output.put_line(employe.nome);
    END LOOP;
end;
```



Les attributs de curseur explicite

Attributs	Informations obtenues
%ROWCOUNT	Nombre de lignes concernées par la dernière requête
%FOUND	TRUE si la dernière requête concernait une ou plusieurs lignes
%NOTFOUND	TRUE si la dernière requête ne concernait aucune ligne
%ISOPEN	TRUE si le curseur est ouvert

Format d'utilisation

Nom_curseur% ATTRIBUT

Curseur modifiable

- Un curseur modifiable est un curseur qui peut être utiliser pour modifier le contenu d'une table
- Un curseur qui comprend plus d'une table dans sa définition ne permet pas la modification des tables de BD.
- Seuls les curseurs définis sur une seule table sans fonction d'agrégation et de regroupement peuvent être utilisés dans une MAJ : delete, update.
- FOR UPDATE [OF col1, col2,…]. Cette clause bloque toute la ligne ou seulement les colonnes spécifiées
- Les autres transactions ne pourront modifier les valeurs tant que le curseur n'aura pas quitté cette ligne
- Pour désigner la ligne courante à modifier on utilise la clause CURRENT OF nom_curseur.

Curseur modifiable - exemple

```
DECLARE
    Cursor C1 is
        select nomemp, sal from employe for update of sal;
    resC1 C1%rowtype;
BEGIN
    Open C1;
    Fetch C1 into resC1;
    While C1%found Loop
        If resC1.sal < 1500 then
            update employe
            set sal = sal * 1.1
            where current of c1;
        end if;
        Fetch C1 into resC1;
    end loop;
    close C1 ;
END; /
```

Exceptions

- **Une exception est une erreur qui survient durant une exécution**
- **Deux types d'exception :**
  - prédéfinie par Oracle
  - définie par le programmeur
- **Une exception ne provoque pas nécessairement l'arrêt du programme si elle est saisie par un bloc (dans la partie «EXCEPTION»)**
- **Une exception non saisie remonte dans la procédure appelante (où elle peut être saisie)**
- **Exemple d'exceptions prédéfinies**
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - VALUE_ERROR (erreur arithmétique)
  - ZERO_DIVIDE

Traitement des exceptions

```
BEGIN
    ...
EXCEPTION
    WHEN exception1 [OR exception2 . . . ] THEN
        . . .
    WHEN exception3 [OR exception4 . . . ] THEN
        . . .
    [WHEN OTHERS THEN -- optionnel
        . . .]
END;
```

- **WHEN . . . . . Identifie une ou plusieurs exceptions**
- **WHEN OTHERS . . . . . Indique le traitement à effectuer si l'exception ne peut être appréhendée par une des clauses WHEN**

Exceptions: 2 types

- **Exceptions définies par ORACLE**
  - Nommées par oracle
    - Ex: NO_DATA_FOUND, TOO_MANY_ROWS,....
    - Se déclenchent automatiquement.
    - Nécessite de prévoir la prise en compte de l'erreur dans la section EXCEPTION.
- **Exceptions définies par l'utilisateur**
  - Nommées par l'utilisateur.
  - Arrêt de l'exécution du bloc.
  - Sont déclenchées par une instruction du programme soit automatiquement (PRAGMA).
  - Nécessite de prévoir la prise en compte de l'erreur dans la section EXCEPTION.

Exceptions

Liste d'exceptions prédéfinies

Nom Exception	Numéro erreur	Description
NO_DATA_FOUND	ORA-01403	La SELECT into ne ramène pas de ligne
TOO_MANY_ROWS	ORA-01422	La SELECT into ramène plus d'une ligne
INVALID_CURSOR	ORA-01001	Opération sur un curseur invalide
ZERO_DIVIDE	ORA-01476	Tentative de division par 0
DUP_VAL_ON_INDEX	ORA-00001	Tentative d'insertion d'une valeur présente dans une colonne ayant un index unique

Exceptions non prédéfinies - règles

DECLARE

Définition

1/ Définir un nom pour l'exception

Nom_E EXCEPTION;

2/ Associer le nom de l'exception à un numéro d'erreur Oracle

PRAGMA EXCEPTION_INIT (nom_E, NumE);

BEGIN

Déclenchement

Déclenchement automatique sous le contrôle du serveur Oracle

EXCEPTION

Interception

Utiliser le mot clé identifiant l'erreur dans la structure de choix :

WHEN nom_E THEN - - -;

Exceptions non prédéfinies - exemple

```
DECLARE
    s_enrg_fils EXCEPTION;
    PRAGMA EXCEPTION_INIT
    (s_enrg_fils ,-2292);
BEGIN
    - - -

EXCEPTION
    WHEN s_enrg_fils THEN
    DBMS_OUTPUT.PUT_LINE('Contrainte d''intégrité non respectée');
END;/
```

Définition d'un nom d'exception

Associe un nom d'erreur avec un numéro d'erreur ORACLE

Prise en compte de l'erreur définie

Gérer l'erreur -2292 qui correspond à la violation d'une contrainte d'intégrité référentielle.

Exception définie par l'utilisateur: règles

DECLARE Définition	1/ Définir un nom pour l'exception Nom_d'exception EXCEPTION;
BEGIN Déclenchement	Déclenchement sous le contrôle du programme RAISE Nom_d'exception;
EXCEPTION Interception	Utiliser le mot clé identifiant l'erreur dans la structure de choix : WHEN nom_d'exception THEN - - -;

Exception définie par l'utilisateur

```
DECLARE
    .....
    Nom_exception EXCEPTION;
BEGIN
    Instructions ;
    IF ( condition_erreur ) THEN RAISE Nom_exception ;
        ...............
EXCEPTION
    WHEN Nom_exception THEN traitements ;
END ;
```

Remarques:

- on sort du bloc après l'exécution du traitement d'erreur.
- Les règles de visibilité des exceptions sont les mêmes que celle des variables.

Propagation des exceptions

- **Traitement de la référence à une exception:**
  1) Le traitement associé à l'exception est d'abord recherché dans le bloc courant. Si l'exception n'est pas trouvée, passer à l'étape 2:
  2) Si un bloc supérieur est trouvé, le traitement associé est recherché ensuite dans ce bloc
  3) Les étapes 1 et 2 sont répétées tant qu'il y a un bloc supérieur ou jusqu'à l'identification du traitement associé à l'exception.

Exception

Exemple

```
DECLARE
    CURSOR employe_rabat IS
        SELECT numemp, sal FROM employe WHERE ville= ‘ Rabat’ ;
    num employe.numemp%TYPE ;
    salaire employe.sal %TYPE ;
    ERR_salaire EXCEPTION ;
BEGIN
    OPEN employe_rabat;
    FTECH employe_rabat INTO num, salaire ;
    WHILE employe_rabat%found LOOP
        IF salaire IS NULL THEN
            RAISE ERR_salaire ;
    .....
EXCEPTION
    WHEN ERR_salaire THEN
            INSERT INTO temp ( num || ‘ salaire non définie’ ) ;
    WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE(‘Pas d’employe');
END ;
```

Exception "others "

- **Exception prédéfinie porte le nom de others.**
- **Permet de traiter toute autre exception non prévue.**
- **Syntaxe : WHEN OTHERS THEN ...**
- **Deux fonctions permettent de récupérer des informations sur l'erreur oracle:**
  - Sqlcode: retourne une valeur numérique : numéro de l'erreur.
  - Sqlerrm: renvoit le libellé de l'erreur.

Exemple

```
DECLARE
    salaire employe.sal %TYPE ;
    SAL_nulle EXCEPTION ;
    code number ;
    message char (50);
BEGIN
    SELECT sal INTO salaire from employe;
    IF salaire= 0 THEN
        RAISE SAL_nulle;
    END IF
EXCEPTION
    WHEN SAL_nulle THEN
        -- gérer erreur salaire
    WHEN TOO_MANY_ROWS THEN
        -- gérer erreur trop de lignes
    WHEN NO_DATA_FOUND THEN
        -- gérer erreur pas de lignes
    WHEN OTHERS THEN
        -- gérer toutes les autres erreurs
        code:= sqlcode ;
        message:= sqlerrm ;
        dbms_output.put_line ( 'erreur: ' || code || message );
END ;
```

Types de blocs PL/SQL

```
[DECLARE]

BEGIN
      -- phrases
   [EXCEPTION]

END;
```

Bloc anonyme

Imbriqué dans un programme

```
PROCEDURE name IS
[Déclaration_variables_locales]

BEGIN
      -- phrases
   [EXCEPTION]

END;
```

Procédure

Effectue un traitement
Peut recevoir des paramètres

```
FUNCTION name IS
RETURN datatype
[Déclaration_variables_locales]

BEGIN
      -- phrases
   [EXCEPTION]

END;
```

Fonction

Effectue un calcul et donne un résultat
Peut recevoir des paramètres

Procédures et fonctions

- **Une procédure/fonction stockée est composée d'instructions compilées et enregistrées dans la BD. Elle est activée par des événements ou des applications. Elle comporte, outre des ordres SQL, des instructions de langage PL/SQL (branchement conditionnel, instructions de répétition, affectations,...).**
- **L'intérêt d'une procédure stockée est :**
  1) D'alléger les échanges entre client et serveur de BD en stockant au niveau du serveur les procédures régulièrement utilisées.
  2) D'optimiser les requêtes au moment de la compilation des procédures plutôt qu'à l'exécution.
  3) De renforcer la sécurité : on peut donner l'autorisation à un utilisateur d'utiliser une procédure stockée sans lui donner les droits directement sur les tables qu'elle utilise.

Procédures

- On définit une procédure de la sorte

  CREATE [OR REPLACE] PROCEDURE nom_procedure
  [(liste_argument1)] {IS|AS}

```
[ -- déclaration des variables, exceptions, procedures,...locales]
BEGIN
    -- Instructions PL/SQL
[exception
    -- gestion des exceptions
]
END [nom_procedure] ;
```

Procédures

- **Syntaxe de liste_argument :**

nom_argument [IN|OUT|IN OUT] type_données [{ := | DEFAULT} valeur]

  - IN : Paramètre d'entrée
  - OUT : Paramètre de sortie
  - IN OUT : Paramètre d'entrée/Sortie

- **L'exécution de l'ordre CREATE PROCEDURE ... déclenche :**
  - La compilation du code source avec génération de pseudo-code si aucune erreur n'est détectée.
  - Le stockage du code source dans la base même si une erreur a été détectée.
  - Le stockage du pseudo-code dans la base, ce qui évite la recompilation de la procédure à chaque appel de celle-ci.

Procédures – Exemple 1

Exemple : procédure permettant d'afficher le nom de tous les employés d'un département dont le numéro est passé en paramètres.

```
CREATE OR REPLACE PROCEDURE Get_emp_names (Dept_num IN NUMBER) IS
  Emp_name     VARCHAR2(10);
  CURSOR       c1 (Depno NUMBER) IS
                 SELECT Ename FROM Emp_tab
                   WHERE deptno = Depno;
BEGIN
  OPEN c1(Dept_num);
  LOOP
    FETCH c1 INTO Emp_name;
    EXIT WHEN C1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(Emp_name);
  END LOOP;
  CLOSE c1;
END;
```

Procédures – exemple 2

- **Exemple : Créer une procédure qui permet d'augmenter le prix d'un produit par un taux.**

```
CREATE OR REPLACE PROCEDURE augmentation_prix ( numerop IN
    NUMBER, Taux IN NUMBER ) IS
     non_trouve Exception;
BEGIN
     UPDATE produit SET PU= PU* ( 1+Taux)
     WHERE Numprod= numerop ;
     if ( SQL%NOTFOUND ) THEN
          RAISE non_trouve;
      end if ;
EXCEPTION
     WHEN non_trouve THEN
           dbms_output.put_line ( 'produit non trouvé ' );
END ;
```

Procédures – Exemple 3

- Procédure Get_emp_rec, qui permet de mettre toutes les colonnes de la table Emp_tab dans un enregistrement PL / SQL pour un employé donné:

```
CREATE OR REPLACE PROCEDURE Get_emp_rec (Emp_number  IN
Emp_tab.Empno%TYPE, Emp_ret    OUT  Emp_tab%ROWTYPE) IS
BEGIN
  SELECT *
    INTO Emp_ret
    FROM Emp_tab
    WHERE Empno = Emp_number;
END;
DECLARE
  Emp_row     Emp_tab%ROWTYPE;    -- declare a record matching a
                                  -- row in the Emp_tab table
BEGIN
  Get_emp_rec(7499, Emp_row);   -- call for Emp_tab# 7499
  .....
END;
```

Procédures – Exemple 4

- **Exemple : Créer une procédure, qui retourne les informations de l'employé dont le numéro est passé en paramètre.**
  employe (numemp, nomemp, salaire, date_embauche)

```
CREATE PROCEDURE REQ_EMPLOYE
    ( numero_e IN employe.numemp%type,
    nom_e OUT employe.nomemp%type,
    sal_e OUT employe.salaire%type,
    date_e OUT employe.date_embauche%type)
    IS
BEGIN
    SELECT nomemp, salaire, date_embauche
    into nom_e, sal_e, date_e FROM Emp_tab
    WHERE numemp = numero_e ;
END ;
```

Optimisation des procédures

- Si la base de données évolue, il faut recompiler les procédures existantes pour qu'elles tiennent compte de ces modifications.
- La commande est la suivante:

  ALTER { FUNCTION | PROCEDURE } nom COMPILE

- Exemple :

  ALTER PROCEDURE augmentation_prix COMPILE;

Suppression d’une procédure (fonction)

- Pour supprimer une procédure
  DROP { FUNCTION |PROCEDURE } nom

Fonctions

- **On définit une fonction de la sorte**

```
CREATE [OR REPLACE] FUNCTION nom_fonction [(liste_argument)]
RETURN type_données {IS|AS}
    [déclaration_variables_locales]
BEGIN
    -- Instructions PL/SQL
    return expression ;
[Gestions des exceptions]
END [nom_fonction] ;
```

Fonctions - Exemple

Exemple : Créer une fonction qui retourne le nom d'un employé.
employe (numemp, nomemp, salaire, date_embauche)

```
CREATE FUNCTION nom_employe ( numero IN NUMBER )
                                    RETURN VARCHAR2
IS
    nom employe. nomemp%type ;
BEGIN
    SELECT nomemp into nom
    FROM employe
    WHERE numemp = numero ;
    RETURN  nom;
END ;
```

Exécution et suppression

- **Sous SQL*PLUS on exécute une procédure PL/SQL avec la commande EXECUTE :**

  EXECUTE nomProcédure(param1, ...);

  Une fonction peut être utilisée dans une requête SQL

  Exemple

  ```
  select nome, sal, euro_to_dh(sal)
  from emp;
  ```

- **Sous SQL PLUS:**
  - 1) EXECUTE augmentation_prix(1, 0.2);
  - 2) VARIABLE nom varchar2(30) ;

    EXECUTE :nom= nom_employe (2) ;

    PRINT nom;

- **partir d'un bloc PL/SQL**

```
DECLARE
      V_numP NUMBER := 14 ;
BEGIN
     augmentation_prix( V_numP, 0.1);
END ;
```

Type d'appel d'un sous-programme

- **L'appel d'un sous-programme (procédure ou fonction) peut être positionnel, mot clé (nommé) ou mixte.**
  - Paramètres positionnels

    augmentation_prix(1, 0.2)
  - Paramètres mot clé

    augmentation_prix(Taux=>0.2, numerop=>1)
  - Mixtes

    augmentation_prix(1, Taux=>0.2)
- **Attention: Pour tous les appels mixtes, il faut que les notations positionnelles précèdent les notations nommées.**

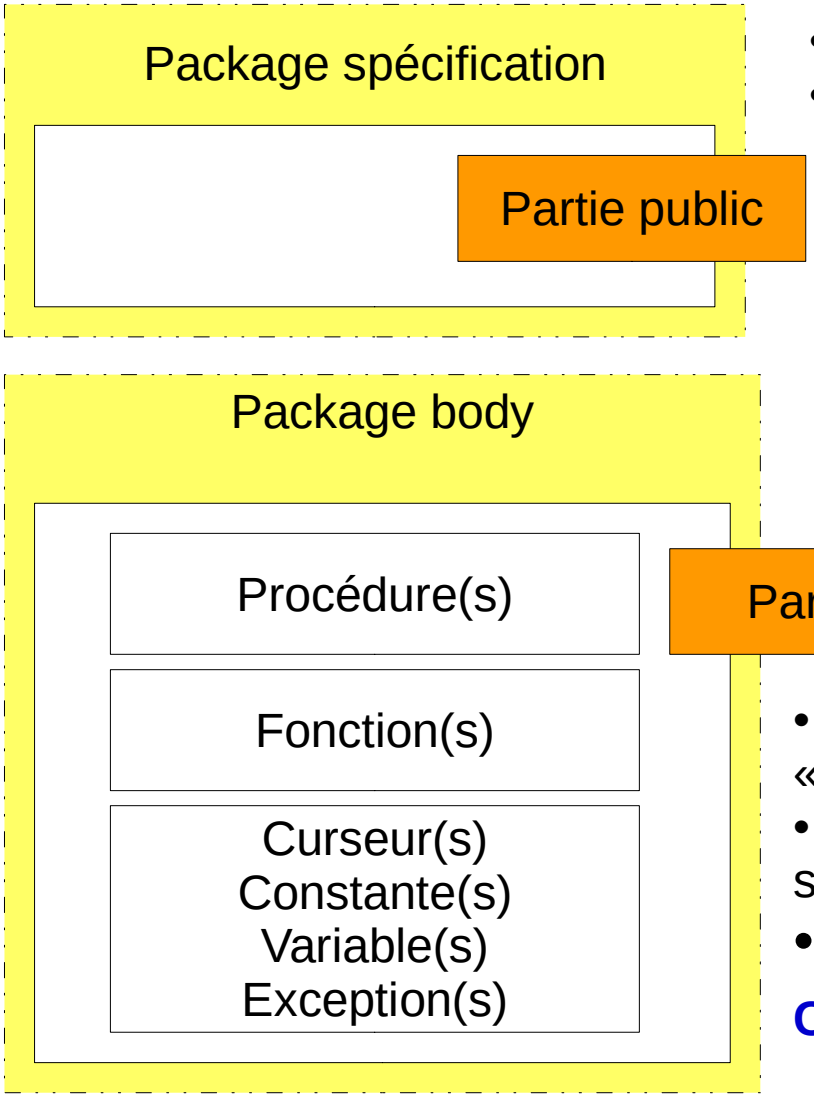
Les packages

- **Un package (paquetage) est l'encapsulation d'objets de programmation PL/SQL dans une même unité logique de traitement tels : types, constantes, variables, procédures et fonctions, curseurs, exceptions.**
- **La création d'un package se fait en deux étapes:**
  - Création des spécifications du package
    - spécifier à la fois les fonctions et procédures publiques ainsi que les déclarations des types, variables, constantes, exceptions et curseurs utilisés dans le paquetage et visibles par le programme appelant.
  - Création du corps du package
    - définit les procédures (fonctions), les curseurs et les exceptions qui sont déclarés dans les spécifications de la procédure. Cette partie peut définir d'autres objets de même type non déclarés dans les spécifications. Ces objets sont alors privés.

      Cette partie peut également contenir du code qui sera exécuté à chaque invocation du paquetage par l'utilisateur (bloc d'initialisation).

Structure d'un package

Un package est constitué de 2 composants



- Déclaration des éléments publics du package.
- Informations accessibles à tout utilisateur autorisé.

CREATE [OR REPLACE] PACKAGE … ;

DROP PACKAGE [BODY] package_name;

- Définition des éléments identifiés dans la partie « package spécification » .
- Définition de sous programmes (privés) utilisables seulement dans le package.
- Bloc d'initialisation

CREATE [or REPLACE] PACKAGE BODY … ;

Création d'une spécification du paquet

La syntaxe pour créer une spécification de paquet PL / SQL:

CREATE [OR REPLACE] PACKAGE package_name
[AUTHID { CURRENT_USER | DEFINER }]
{ IS | AS }

[definitions of public TYPES
,declarations of public variables, types, and objects
,declarations of exceptions
,pragmas
,declarations of cursors, procedures, and functions
,headers of procedures and functions]

END [package_name];

Création du corps du package

- Syntaxe:

  CREATE [OR REPLACE] PACKAGE BODY package_name **{ IS| AS }**
  specification PL/SQL
  [BEGIN
  bloc d'initialisation]
  END;

  - specification PL/SQL::= **declaration_de_variable|**
    declaration_d_enregistrement|
    declaration_de_curseur|
    declaration_d_execption|
    Définition_de_procedure|
    Définition_de_fonction...

- **L'identificateur de la spécification et du corps du package doivent être les mêmes.**

Exemple 1

```
CREATE OR REPLACE PACKAGE gest_empl IS
    -- Variables globales et publiques

    Sal EMP.sal%Type ;
    -- Fonctions publiques

    FUNCTION F_Augmentation (Numemp IN EMPLOYE.noemp%Type
            ,Pourcent IN NUMBER ) Return NUMBER ;
    -- Procédures publiques

    PROCEDURE Test_Augmentation (
       Numemp IN EMPLOYE.noemp%Type  --numéro del'employé
     ,Pourcent IN OUT NUMBER    -- pourcentage d'augmentation
 ) ;

End gest_empl;
```

Exemple 1 (suite 1)

```
CREATE OR REPLACE PACKAGE BODY gest_empl IS
    --Variables globales privées
    Emp EMPLOYE%Rowtype ;
    -- Procédure privées
    PROCEDURE Affiche_Salaires IS
        CURSOR C_EMP IS select * from EMPLOYE ;
    BEGIN
        OPEN C_EMP ;
        Loop
            FETCH C_EMP Into Emp ;
            Exit when C_EMP%NOTFOUND ;
            dbms_output.put_line( 'Employé ' || Employe.name || ' -> '|
                                  To_char( Employe.sal );
        End loop ;
       CLOSE C_EMP ;
    END Affiche_Salaires ;
```

Exemple 1 (suite 2)

```
    FUNCTION F_Augmentation ( Numemp IN EMPLOYE.noemp%Type , Pourcent IN NUMBER )
    Return NUMBER IS        -- Fonctions publiques
        Salaire EMPLOYE.sal%Type default 0 ;
    BEGIN
       Select sal Into Salaire From EMPLOYE  Where noemp = Numemp ;
       if (SQL%Found) then
          update Employe set sal=sal*Pourcent Where noemp = Numemp ;
          Salaire := Salaire * Pourcent    ;-- Affectation de la variable globale publique
          Sal := Salaire ;
        end if ;
       Return( Salaire ) ; -- retour de la valeur
    END F_Augmentation;
-- Procédures publiques
    PROCEDURE Test_Augmentation ( Numemp IN EMPLOYE.noemp%Type ,Pourcent IN OUT
    NUMBER ) IS
        Salaire EMPLOYE.sal%Type ;
    BEGIN
        F_Augmentation(Numemp, Pourcent) ;
        Affiche_Salaires ;-- appel procédure privée
    END Test_Augmentation;
END gest_empl; /
```

Exemple

- **Spécification**

```
CREATE OR REPLACE PACKAGE compteur IS
    procedure reset;
    function nextValue return number;
END compteur; /
```

- **Corps**

```
CREATE OR REPLACE PACKAGE BODY compteur IS
    cpt NUMBER := 0;
    PROCEDURE reset IS
    BEGIN
        cpt := 0;
    END;
    FUNCTION nextValue RETURN NUMBER IS
    BEGIN
        cpt := cpt + 1;
        RETURN cpt - 1;
    END;
END compteur; /
```

L'accès à un objet d'un paquetage

- **L'accès à un objet d'un paquetage est réalisé avec la syntaxe suivante :**

  nom_paquetage.nom_objet[(liste paramètres)]

- **Exemple: Appel de la fonction F_Augmentation du paquetage**

```
Declare
    Salaire emp.sal%Type ;
Begin
    Select sal Into Salaire From EMPLOYE Where noempno = 100 ;
    dbms_output.put_line( 'Salaire d'employé numero 100 avant
    augmentation ' || To_char( Salaire ) ) ;
    dbms_output.put_line( 'Salaire d'émployé numéro l100 après
    augmentation '|| To_char(gest_empl.F_Augmentation( 100, 0.2 ) ) ) ;
End ; /
```

Exemple d'utilisation d'un package

```
DECLARE
    nb NUMBER;
BEGIN
    FOR nb IN 4..8 LOOP
        DBMS_OUTPUT.PUT_LINE(COMPTEUR.nextValue()) ;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(COMPTEUR.nextValue()) ;
     COMPTEUR.RESET;
    FOR nb IN REVERSE 4..10 LOOP
        DBMS_OUTPUT.PUT_LINE(COMPTEUR.nextValue());
    END LOOP;
END; /
```

Suppression d’un package

- DROP PACKAGE [BODY] package_name;

- Si vous souhaitez supprimer uniquement le corps du package, vous devez spécifier le mot clé BODY.
- Si vous omettez le mot clé BODY, l'instruction supprime à la fois le corps et les spécifications du package.

Déclencheur (trigger)

- **Un trigger est un morceau de code PL/SQL associé à une vue ou une table :**
  - stocké dans la base,
  - déclenché lors de l'occurrence d'un événement particulier de langage de manipulation de données (LMD)
- **Syntaxe :**

```
CREATE [or REPLACE] TRIGGER <nom>
{BEFORE | AFTER l INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
ON <table>
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
[WHEN (condition)]
-- bloc PL/SQL
```

- **Avec**
  - listeEvénements : liste d'événements séparés par une virgule DELETE, INSERT, ou UPDATE
  - Si UPDATE on peut préciser les attributs concernés (UPDATE OF listeAttributs).

Déclencheur (trigger)

- **Ainsi les Triggers :**
  - permettent de synchroniser des opérations entre plusieurs tables
  - peuvent être utilisés pour implémenter certaines règles de gestion (souvent les contraintes remplissent plus efficacement ce rôle)
  - sont généralement déclenchés par la modification du contenu d'une table
  - les ordres du LDD (CREATE, ALTER, DROP, ...) et de gestion de transactions (COMMIT, SAVEPOINT,...) sont interdits dans les Triggers.

Les types de triggers

- **Il existe deux types de triggers différents :**
  - les triggers de table (STATEMENT)
  - les triggers de ligne (ROW).
- **Les triggers de table sont exécutés une seule fois lorsque des modifications surviennent sur une table (même si ces modifications concernent plusieurs lignes de la table). Ils sont utiles si des opérations de groupe doivent être réalisées (comme le calcul d'une moyenne, d'une somme totale, d'un compteur, ...).**
- **Les triggers lignes sont exécutés «séparément» pour chaque ligne modifiée dans la table. Ils sont très utiles s'il faut mesurer une évolution pour certaines valeurs, effectuer des opérations pour chaque ligne en question.**

Les types de triggers

- **Le traitement spécifié dans un trigger peut se faire :**
  - pour chaque ligne concernée par l'événement => trigger de niveau ligne (FOR EACH ROW)
  - une seule fois pour l'ensemble des lignes concernées par l'événement : => trigger de niveau table : pas de clause FOR EACH ROW
- **Quand le Trigger est déclenché ? BEFORE | AFTER | INSTEAD OF**
  - trigger de niveau table : déclenché avant ou après l'événement
  - trigger de niveau ligne : exécuté avant ou après la modification de CHAQUE ligne concernée
  - INSTEAD OF : est utilisée pour créer un trigger sur une vue.

Accès aux lignes en cours de modification

- **Dans les FOR EACH ROW triggers, il est possible avant la modification de chaque ligne, de lire l'ancienne ligne et la nouvelle ligne par l'intermédiaire des deux variables structurées old et new.**
- **exemple :**
  - Si nous ajoutons un client dont le nom est toto alors nous récupérons ce nom grâce à la variable :new.nom
  - Dans le cas de suppression ou modification, les anciennes valeurs sont dans la variable :old.nom

L'option REFERENCING

- **Seule la ligne en cours de modification est accessible par l'intermédiaire de 2 variables de type enregistrement OLD et NEW**
- **Les noms de ces deux variables sont fixés par défaut, mais il est possible de les modifier en précisant les nouveaux noms dans la clause REFERENCING**

```
CREATE [or REPLACE] TRIGGER <nom>
{BEFORE | AFTER l INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
ON <table>
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
[WHEN (...)]
-- bloc PL/SQL
```

Conditions d'un triggers

- Le trigger se déclenche lorsqu'un événement précis survient : BEFORE UPDATE, AFTER DELETE, AFTER INSERT, …. Ces événements sont importants car ils définissent le moment d'exécution du trigger.
- Pour rappel, lorsqu'un trigger ligne est mentionné à
  - INSERT Pas d'accès à l'élément OLD (qui n'existe pas)
  - UPDATE Accès possible à l'élément OLD et NEW
  - DELETE Pas d'accès à l'élément NEW (qui n'existe plus)

Triggers - Modification

- On ne peut modifier la définition du trigger, il faut la remplacer (d'où l'intérêt du CREATE OR REPLACE).
- Quand on crée le trigger, il est automatiquement activé. On peut le désactiver, puis le réactiver :
  ALTER TRIGGER nom_trigger {disable|enable|compile} pour désactiver/réactiver/recompiler
  ALTER TABLE nom_table DISABLE ALL TRIGGERS pour désactiver tous les triggers d'une table
  DROP TRIGGER nom_trigger pour la suppression d'un trigger

Exemple 1

```
CREATE OR REPLACE TRIGGER TRG_BDR_EMP -- BeforDeleteRecord
    BEFORE DELETE -- avant supression
    ON EMP       -- sur la table EMP
    FOR EACH ROW  -- pour chaque ligne
  [Declare
            %déclaration des variables]
  Begin
   dbms_output.put_line( 'Suppression de l''employé n° ' ||
   To_char( :OLD.empno )  || ' -> ' || :OLD.ename ) ;
  End ;
```

La procédure raise_application_error

- La procédure raise_application_error (error_number,error_message)
  - **error_number** doit être un entier compris entre -20000 et -20999
  - **error_message** doit être une chaîne de 500 caractères maximum.
  - Quand cette procédure est appelée, elle termine le trigger, défait la transaction (ROLLBACK), renvoie un numéro d'erreur défini par l'utilisateur et un message à l'application.

Gestion des exceptions

- Si une erreur se produit pendant l'exécution d'un trigger, toutes les mises à jour produites par le trigger ainsi que par l'instruction qui l'a déclenché sont défaites.
- On peut introduire des exceptions en provoquant des erreurs.
  - Une exception est une erreur générée dans une procédure PL/SQL.
  - Elle peut être prédéfinie ou définie par l'utilisateur.
  - Un bloc PL/SQL peut contenir un bloc EXCEPTION gérant les différentes erreurs possibles avec des clauses WHEN.
  - Une clause WHEN OTHERS THEN ROLLBACK; gère le cas des erreurs non prévues.

Exemple 2

- La DRH annonce que désormais, tout nouvel employé devra avoir un numéro supérieur ou égal à 10000 Il faut donc interdire toute insertion qui ne reflète pas cette nouvelle directive

```
CREATE OR REPLACE TRIGGER TRG_BIR_EMP
    BEFORE INSERT -- avant insertion
    ON EMP      -- sur la table EMP
   FOR EACH ROW  -- pour chaque ligne
  Begin
   If :NEW.empno < 10000 Then
      RAISE_APPLICATION_ERROR ( -20010, 'Numéro employé inférieur
à 10000' ) ;
    End if ;
  End ;
```

Les prédicats conditionnels INSERTING, DELETING et UPDATING

- Quand un trigger comporte plusieurs instructions de déclenchement (par exemple **INSERT** OR **DELETE** OR **UPDATE**), on peut utiliser des prédicats conditionnels
  - **INSERTING**,
  - **DELETING** et
  - **UPDATING**
- Pour exécuter des blocs de code spécifiques pour chaque instruction de déclenchement.

Les prédicats conditionnels INSERTING, DELETING et UPDATING

Exemple :

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE ON employe
.....
BEGIN
.....
IF INSERTING THEN ..... END IF;
IF UPDATING THEN ..... END IF;
.....
END;
```

Les prédicats conditionnels INSERTING, DELETING et UPDATING

UPDATING peut être suivi d'un nom de colonne :

CREATE TRIGGER ...

BEFORE UPDATE OF salaire, commission ON employe

.....

BEGIN

.....

IF UPDATING ('salaire') THEN ..... END IF;

.....

END;

Exemple 3

```
CREATE OR REPLACE TRIGGER TRG_BIUDR_EMP
   BEFORE INSERT OR UPDATE OR DELETE-- avant insertion,
modification ou suppression
   ON EMP      -- sur la table EMP
   FOR EACH ROW  -- pour chaque ligne
 Begin
  If INSERTING Then
    dbms_output.put_line( 'Insertion dans la table EMP' ) ;
  End if ;
  If UPDATING Then
   dbms_output.put_line( 'Mise à jour de la table EMP' ) ;
  End if ;
  If DELETING Then
    dbms_output.put_line( 'Suppression dans la table EMP' ) ;
  End if ;
 End ;
```

Quelques exemples

```
CREATE TABLE Livre (
   noLivre NUMERIC PRIMARY KEY,
   prix NUMERIC(9,2)
)
CREATE TABLE PrixLivre (
   nb NUMERIC,
   somme NUMERIC(12,2)
)
INSERT INTO PrixLivre VALUES (1,0);
```

- Créer un Trigger qui permet de vérifier avant chaque insertion d'un livre son prix :
- Si le prix est supérieur à 0
  - S'il est compris entre le prix moyen * 0.7 et le prix moyen *1.3 , mettre à jour la table PrixLivre (somme et nb)
  - Si le prix est < prix_moyen*0.7 ou >prix_moyen*1.3 soulever une exception
- Sinon mettre à jour uniquement le champ somme de la table PrixLivre

Quelques exemples (suite)

```
CREATE OR REPLACE TRIGGER moyenne_prix
  BEFORE INSERT
  ON Livre
  FOR EACH ROW
DECLARE
  prix_moyen number;
BEGIN
 SELECT SOMME/NB INTO prix_moyen
 FROM PrixLivre;
 IF (prix_moyen>0)THEN
  IF (:new.PRIX < prix_moyen*0.7 OR :new.PRIX > prix_moyen*1.3)THEN
   raise_application_error(-20001,'Prix modifiant trop la moyenne');
  END IF;
  IF (:new.PRIX > prix_moyen*0.7 AND :new.PRIX < prix_moyen*1.3)
  THEN
    UPDATE PrixLivre SET NB=NB+1,
   SOMME=SOMME+:new.PRIX;
  END IF;
 ELSE
  UPDATE PrixLivre SET SOMME=SOMME+:new.PRIX;
 END IF;
END;
```

Quelques exemples (suite)

Soit une table quelconque TABL, dont la clé primaire CLENUM est numérique.

- **Définir un trigger en insertion permettant** d'implémenter une numérotation automatique de la clé. Le premier numéro doit être 1.

Quelques exemples (suite)

```
create or replace trigger cleauto
   before insert on tabl
   for each row
declare
   n integer;
   newkey integer;
   mon_exception exception;
begin
   -- Recherche s'il existe des tuples dans la table
   select count(*) into n from tabl;
   if n=0 then
      raise mon_exception; -- Premiere insertion
   end if;
   -- Recherche la valeur de cle C la plus elevee
   -- et affecte C+1 a la nouvelle cle
   select max(clenum) into newkey from tabl;
   :new.clenum := newkey + 1;
exception
```

Quelques exemples (suite)

- CLIENT (NUMCL, NOM, PRENOM, ADR, CP, VILLE, SALAIRE, CONJOINT)
  DETENTEUR (NUMCL, NUMCP)
  COMPTE (NUMCP, DATEOUVR, SOLDE)
- **Écrire un trigger en insertion permettant de contrôler les contraintes suivantes :**
  - le département dans lequel habite le client doit être 01, 07, 26, 38, 42, 69, 73, ou 74 (sinon il n'est pas en France*)
  - le nom du conjoint doit être le même que celui du client.

Quelques exemples (suite 1/3)

```
CREATE TRIGGER INS_CLIENT
      BEFORE INSERT ON CLIENT
      FOR EACH ROW
DECLARE
     nom_conjoint CLIENT.NOM%TYPE ;
     compteur CLIENT.NUMCL%TYPE ;
     pb_dept EXCEPTION ;
     pb_conjoint1 EXCEPTION ;
     pb_conjoint2 EXCEPTION ;
BEGIN
     -- Contrainte sur le département
     IF (:NEW.CP) NOT IN (01, 07, 26, 38, 42, 69, 73, 74) THEN
          RAISE pb_dept ;
     END IF ;
```

Quelques exemples (suite 2/3)

```
-- Contrainte sur le nom du conjoint (+ test d'existence du conjoint)
   IF :NEW.CONJOINT IS NOT NULL THEN
       SELECT COUNT(*), NOM INTO compteur, nom_conjoint
           FROM CLIENT
            WHERE NUMCL = :NEW.CONJOINT
           GROUP BY NOM ;
      IF compteur = 0 THEN -- Pas de conjoint
         RAISE pb_conjoint1 ;
      END IF ;
      IF nom_conjoint != :NEW.NOM THEN
         RAISE pb_conjoint2 ;
      END IF ;
   END IF ;
```

Quelques exemples (suite 3/3)

```
EXCEPTION
    WHEN pb_dept THEN
       RAISE_APPLICATION_ERROR (-20501, 'Insertion impossible : le
client n'habite pas en France!') ;
   WHEN pb_conjoint1 THEN
     RAISE_APPLICATION_ERROR (-20502, 'Insertion impossible : le
conjoint du client n'existe pas !') ;
   WHEN pb_conjoint2 THEN RAISE_
     APPLICATION_ERROR (-20503, 'Insertion impossible : le nom du
conjoint est différent de celui du client !') ;
END ;
```

Ch. IV Pages web pilotées par une base de données

- Ouverture d'une connexion vers une base de données
- Optimisation des requêtes (Partitionnement des tables et indexes, indexation des opérations en ligne et en parallèle, Maintenance et configuration des indexes).
- Stockage de données capturées par des formulaires
- Envoi de requêtes dynamiques à une base de données
- Générer une page web affichant les résultats d'une requête

BD et Web

- **Un site dynamique est construit sur une base de données.**
  - l'utilisateur sollicite une page, la demande est envoyée au site, le serveur exécute un programme,
  - les informations utiles sont extraites de la base de données et rapatriées au serveur,
  - une feuille de style est appliquée, un document html est construit et envoyé sur le réseau,
  - le document est affiché dans le navigateur de l'utilisateur.
- **l'intérêt est de distinguer les données des traitements et de l'affichage : pour modifier le contenu du site, il suffit de mettre à jour les informations dans la base de données ; pour modifier la présentation, il suffit de modifier le style du document Web.**

Ouverture d'une connexion vers une base de données

*db.properties

```
1 jdbc.driver=com.mysql.jdbc.Driver
2 jdbc.url=jdbc:mysql://localhost:3306/DB_CONTACTS
3 jdbc.user=root
4 jdbc.password=
```

```
try {
ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
InputStream in = classLoader.getResourceAsStream("db.properties");
prop.load(in);
in.close();
// Extraction des propriétés
String driver = prop.getProperty("jdbc.driver");
String url = prop.getProperty("jdbc.url");
String user = prop.getProperty("jdbc.user");
String password = prop.getProperty("jdbc.password");
Class.forName(driver);
Connection connect = DriverManager.getConnection(url, user,
password);
} catch (Exception e) {
e.printStackTrace();
}
```

Ouverture d'une connexion vers une base de données

```
<?php
   $link = mysql_connect('hostname','dbuser','dbpassword');
   if (!$link) {
          die('Could not connect to MySQL: ' . mysql_error());
   }
      echo 'Connection OK';
      mysql_close($link);
?>
```

Stockage de données capturées par des formulaires

```
<?php
    $do_this = $_POST["do_this"];
    if(strcmp ($do_this,"Ajouter la commande")==0)
    {
        $ncom = $_POST["ncom"];
        $ncli = $_POST["ncli"];
        $datecom=$_POST["datecom"];
        $adrliv=$_POST["adrliv"];
        if(($ncom=intval($ncom))==false)
        {
            $erreur="Le numéro de la commande est mal saisi";
            require "erreur.php";
        }else  if(($ncli=intval($ncli))==false)
       {
           $erreur="Le numéro de client est mal saisi";
           require "erreur.php";
       } else
```

Stockage de données capturées par des formulaires

```
        if($time = date($datecom)==false)
        {
              $erreur="La date est mal saisi";
              require "erreur.php";
        }
        else
        {
       // traitement de la requête
              if($db=new PDO('sqlite:commande.db'))
             {
$sql="INSERT INTO commande(NCOM,NCLI, DATECOM,ADRLIV)
VALUES($ncom,$ncli,$datecom,$adrliv)";
              $infG=$db->exec($sql);
              $db=NULL;
            }
           else
           {
```

Exemple authentification

```
<?php
   // auth.php - Authentification des admins Bases Hacking
   $login = $_POST["pseudo"];
   $mdp = $_POST["mdp"];
   echo($login.'   '.$mdp);
   if ($login != "" && $mdp != "") {
         echo("Authentification..");
          @mysql_connect("localhost", "root", "") or die("Impossible de se
connecter à la base de données");
      @mysql_select_db("users") or die("Table inexistante");
      $resultat = mysql_numrows(mysql_query("SELECT * from admin
WHERE login='$login' AND mdp='$mdp';"));
      mysql_close();
      if ($resultat >= 1){
               echo("Authentification réussie, vous allez être redirigés
immédiatement.");
            header("location:./admin.php");
```

Générer une page web affichant les résultats d'une requête

```
<?php
      $ncom = $_POST["ncom"];
      $row = array();
     if ($ncom != "")  {
        if(is_numeric($ncom)){
           $ncom=intval($ncom);
       }else{
            $erreur="Le numéro de la commande est mal saisi";
            require "erreur.php";
            goto fin;
       }
       if($db=new PDO('sqlite:commande.db'))
     {
              $sql1="select C.NCLI ,C.NOM, CM.ADRLIV from COMMANDE
CM,  CLIENT C
                        where C.NCLI=CM.NCLI and CM.NCOM=$ncom";
      $infG=$db->query($sql1);
```

Générer une page web affichant les résultats d'une requête

```
                $sql="select D.NPRO, P.LIBELLE, P.Prix as 'Prix Unit',
D.QCOM, (D.QCOM*P.Prix) as Montant
                    from COMMANDE CM
                    JOIN DETAIL D on CM.NCOM = D.NCOM
                    JOIN PRODUIT P on D.NPRO = P.NPRO
                    Where CM.NCOM=$ncom";
                $resultat=$db->query($sql);
                $db=NULL;
                $row=$infG->fetch(PDO::FETCH_NUM);
                require 'Facturesucces.php';
                goto fin;
            }
        }
        else{
           die ($sqliteerror);
        }
    } else {
```

Références

- http://odile.papini.perso.esil.univmed.fr/sources/BD.html
- http://combot.univ-tln.fr/loris/admin/plsql/hash.html
- https://www.db.bme.hu/files/Manuals/Oracle/Oracle11gR2/appdev.112/e17126/composites.htm#CHDEIJHD
-