SGBD

Ch.1

BD : collection de données concernant un sujet enregistrées sur un support permanent accessible par l'ordinateur.
SGBD : logiciel qui permet à un utilisateur d'exploiter une BD

Propriétés d'une base de données :

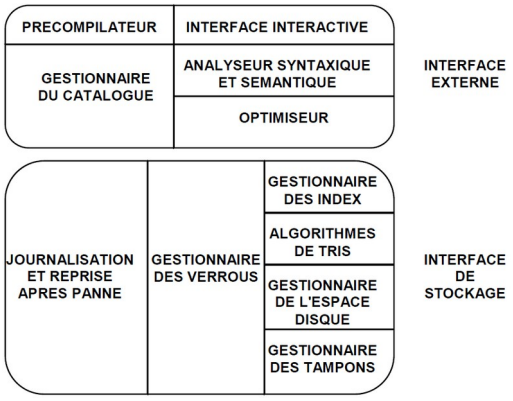
- Être un ensemble organisé/structuré
- Être un ensemble intégré
- Correspondre fidèlement à la réalité
- Contenir les données opérationnelles sur un sujet donné
- Être multi-utilisateurs
- Être non-redondante ou à redondance contrôlée

Les Fonctions d'un SGBD

1. Description et définition
2. Manipulation
3. Intégrité
4. Confidentialité
5. Concurrence d'accès

Architecture d'un SGBD :

- Le Catalogue ou Méta base ou Dictionnaire ou Tables Système constitue une mini base de données contenant des informations sur la BD
- L'interface de stockage s'occupe de tout ce qui concerne l'accès aux données stockées sur disques
- Le gestionnaire des verrous assure la fonction de concurrence d'accès.



Objectifs fondamentaux d'un système base de données :

- L'indépendance des données (LOGIQUE et PHYSIQUE) par rapport aux programmes de traitements
- La prise en compte des associations entre les différentes données
- Le partage simultané des données entre plusieurs utilisateurs

Pas de changement pour	Pgm	niv log	Org mém
Ajout d'1 nv pgm utilisant des données existantes	*	*	*
1 pgm utilise une nvelle représentation de données existantes	*	*	*
Ajout d'un nv pgm utilisant de nvelles données	*		
Description log glob améliorée / ajout nvelles assoc entre données	*		
Fusion de 2 BD	*	*	
Organisation physique améliorée, éventuellement nvelle représentation de données	*	*	
Méthodes d'accès modifiées	*	*	
Données déplacées sur d'autres volumes	*	*	
Logiciel est changé (nvelle version)	*	*	
Matériel est changé	*	*	

L'indépendance des données au niveau LOGIQUE signifie que l'on peut changer la structure logique globale sans devoir changer les programmes d'applications.

L'indépendance des données au niveau PHYSIQUE signifie que la couche physique et l'organisation des données peuvent changer sans devoir changer la structure logique globale ou les programmes d'applications.

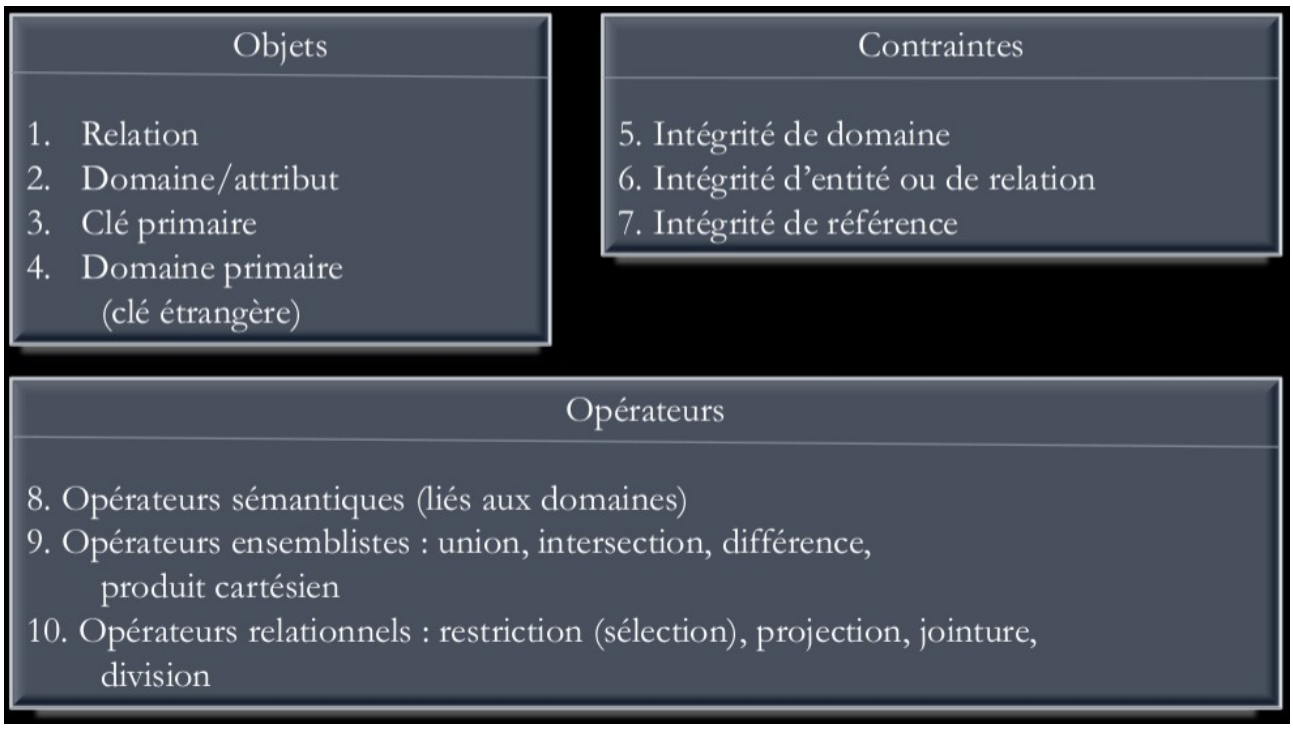
Ch.2

Le modèle relationnel s'appuie sur une base formelle : la théorie des ensembles ou la théorie des prédicats.

- Un ensemble est une collection, un regroupement d'objets, de nombres, d'identités concrètes ou abstraites.
- Les objets particuliers qui appartiennent à un ensemble sont appelés les éléments de cet ensemble.
- On peut représenter graphiquement les ensembles et les opérations sur les ensembles par des diagrammes de Venn.
- Un prédicat est une phrase qui peut comporter des paramètres et qui peut être vraie ou fausse.
- Une relation est un ensemble, au sens mathématique, qui va être visualisé sous la forme d'une table.
- Les opérateurs relationnels sont des opérateurs ensemblistes.

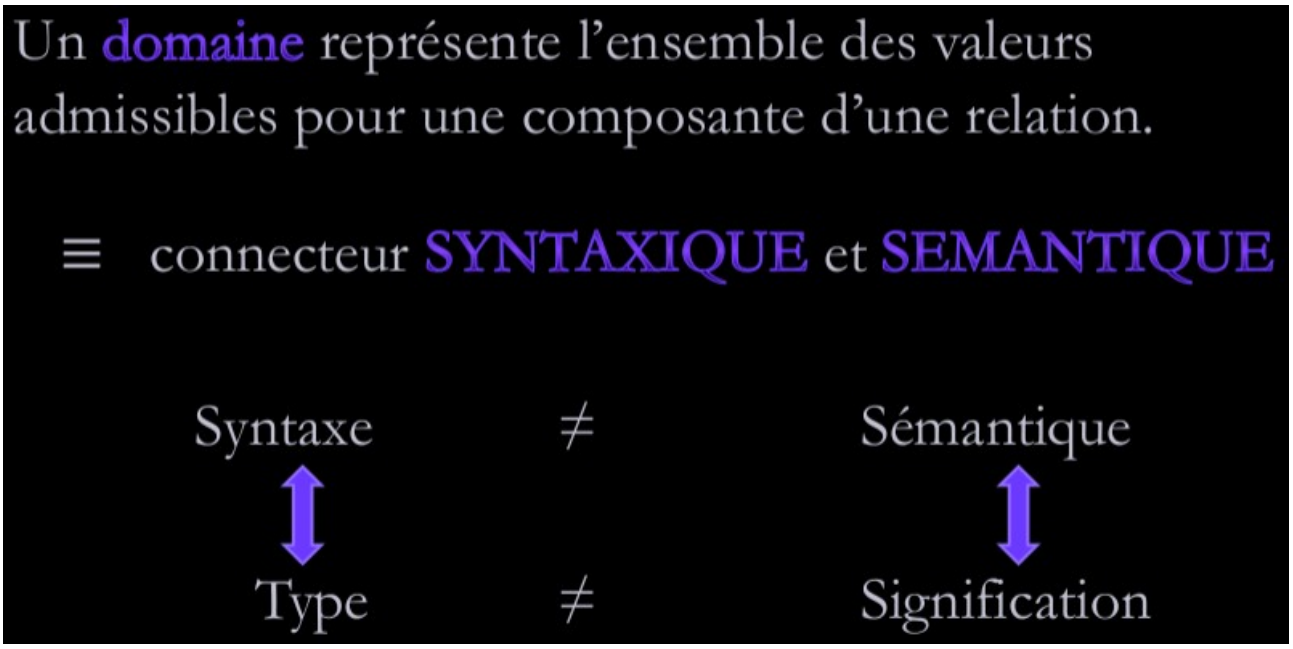
Le modèle relationnel de Codd repose sur 3 piliers :

- Les objets : les éléments des base
- Les règles d'intégrité : qui permettent de faire respecter le modèle des données
- Les opérateurs : qui offrent la possibilité de manipuler la base de données



Une relation étant un ensemble, elle peut être définie de manière :

- Extensive : en donnant la liste de tous les tuples la composant :
  AEcrit = { (1, 2), (2, 1), (3, 3), (3, 4) }
- Intensive : en donnant le prédicat d'appartenance d'un tuple à R :
  AEcrit = { (x, y) : l'auteur x a écrit le livre y }



Deux domaines sont déclarés compatibles s'ils sont sémantiquement comparables, c'est-à-dire si les ensembles qui les définissent ne sont pas disjoints.

Une clé primaire est un ensemble d'attributs, K, vérifiant la double propriété :

- Unicité : les valeurs de clés primaires sont uniques et non nulles;
- Minimalité : aucun attribut composant K ne peut être enlevé sans perdre la propriété d'unicité.

Un domaine primaire est un domaine sur lequel une clé primaire est définie.

Un attribut qui n'est pas clé primaire mais qui est défini sur un domaine primaire est appelé une clé étrangère.

L'intégrité de domaine porte sur le contrôle syntaxique et sémantique des valeurs présentes dans un attribut : seules les valeurs appartenant au domaine de l'attribut sont autorisées.

L'intégrité d'entité ou de relation concerne les valeurs de la clé primaire d'une relation qui doivent être uniques et toujours définies (non nulles).

Soit l'attribut A de R_1 définie sur le domaine primaire D. Alors, à chaque valeur v de A dans R_1, on doit avoir, soit v valeur inconnue (c'est-à-dire nulle dans le jargon des bases de données), soit v valeur d'une clé primaire définie sur D dans la relation R_2.

L'algèbre relationnelle ou langage algébrique est caractérisé par les propriétés suivantes :

- Fermeture
- Ensembliste
- Non procédural
- Universel
- Indépendance

Deux relations sont union-compatibles si :

- Elles ont le même nombre d'attributs (le même degré)
- Les attributs associés deux à deux sont définis sur des domaines compatibles.

Opérateurs ensemblistes :

- Union
- Différence
- Intersection (à partir de la différence : $X = R_1 \setminus (R_1 \setminus R_2)$)
- Produit cartésien

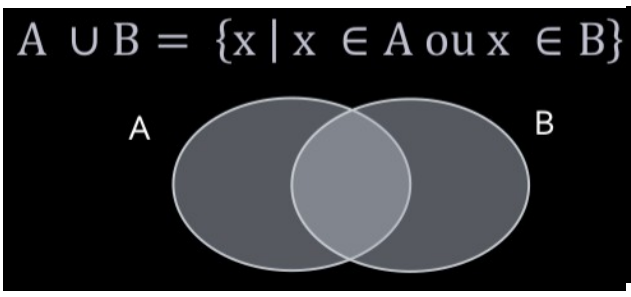


Figure 2: Union

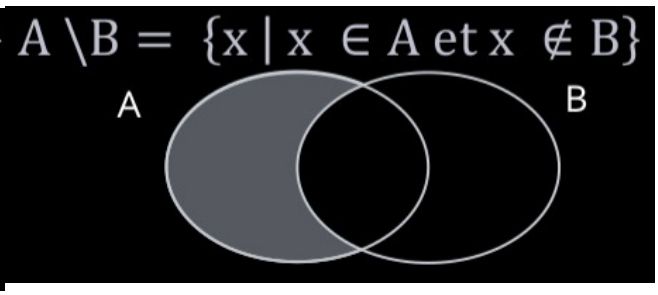


Figure 1: Différence

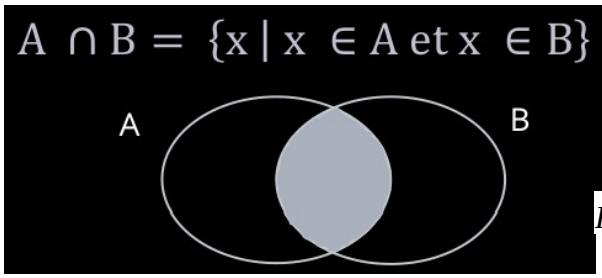


Figure 3: Intersection

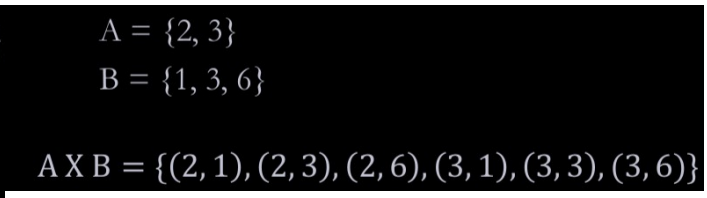


Figure 4: Produit cartésien

Opérateurs relationnels :

- Projection
- Sélection

Opérateurs additionnels :

- Intersection
- Jointure
- Jointure externe
- Division

Ch.3

La forme de Backus-Naur (BNF : Backus-Naur Form) est une notation permettant de décrire les règles syntaxiques des langages de programmation.

Symbole	Signification
::=	Définit comme suit
|	"ou" logique
<>	Concept (nom d'objet, valeur, …)
[ ]	Option possible, non obligatoire
{ }	Élément à choisir, liste d'éléments à choisir

CREATE DOMAIN nom type [valeur] ;

- valeur ::= DEFAULT constante | USER | NULL | CURRENT_DATE | CURRENT_TIME | CURRENT_TIMESTAMP
- type ::= CHAR [(n)] | VARCHAR [(n)] | SMALLINT | INTEGER | NUMERIC [(p [, q])] | DECIMAL [(p [, q])] | FLOAT [(n)] | DATE [ANSI | VMS] | TIME frac | TIMESTAMP frac | INTERVAL type_intervalle

SQL2 a apporté une réelle extension au type DATE qui est subdivisé en deux sous-types :

- Le type datetime qui englobe les types
- Le type interval qui représente une période de temps (et qui peut être négatif)

CREATE TABLE nom_table (liste_def_colonne [liste_contrainte_table]);

- def_colonne ::= nom_colonne type | nom_domaine [val_par_defaut] { contrainte_colonne }

- contrainte_colonne ::= CONSTRAINT nom_contrainte type_contrainte_col mode_contrainte
- type_contrainte_col ::= PRIMARY KEY | NOT NULL | UNIQUE | CHECK (condition) | REFERENCES nom_table (liste_colonne) [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}] [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

contrainte_table ::= CONSTRAINT nom_contrainte type_contrainte_table mode_contrainte

- type_contrainte_table ::= PRIMARY KEY (liste_colonne) | UNIQUE (liste_colonne) | CHECK (condition) | FOREIGN KEY (liste_colonne) REFERENCES nom_table (liste_colonne) [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}] [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
- mode_contrainte ::= [NOT] DEFERRABLE

Plusieurs types de contraintes :

- Sur les valeurs directement : DEFAULT / NOT NULL / UNIQUE
- Sur les intégrités : PRIMARY KEY, FOREIGN KEY, SK (secondary key)
- Sur des conditions à remplir : CHECK

Il est vivement conseillé d'utiliser systématiquement la clause CONSTRAINT

Valeur	Signification	Opérateur	Signification
CURRENT_DATE	Date courante (courte)	BETWEEN	Plage de valeurs (bornes comprises dans l'intervalle)
CURRENT_TIME	Heure courante	LIKE	Comparaison partielle de chaines de caractères (%, _)
CURRENT_TIMESTAMP	Date/Heure	IN	Liste de valeurs possibles
USER	Utilisateur courant	CASE	Branchement de différentes valeurs
CURRENT_USER	Idem	OVERLAPS	Recouvrement de périodes
SESSION_USER	Utilisateur qui a initié la session	SIMILAR	Expression rationnelle

Une base de données Oracle est constituée d'une collection de schémas (users).

Un schéma contient des objets du monde relationnel.

Ces schémas sont répartis dans des zones logiques appelées tablespaces constituées de segments composés de différentes extension (extents) organisées en blocs (blocks).

```
CREATE DATABASE NomBase ... ;
CREATE TABLESPACE NomTableSpace ... ;
CREATE ROLLBACK SEGMENT NomSegment ... ;
CREATE USER sgbdTest IDENTIFIED BY sgbdTest
        DEFAULT TABLESPACE users
        TEMPORARY TABLESPACE temp
        PROFILE DEFAULT ACCOUNT UNLOCK;
```

Méthodes d'accès plus rapides :

- Index hiérarchiques
- Clusters
- Techniques de hachage (hash-coding)
- Index à matrices binaires

supprimer_objet ::= DROP DOMAIN nom_domain | TABLE nom_table [CASCADE | RESTRICT] | INDEX nom_index | DATABASE FILENAME nom_base ;

DROP SCHEMA|USER nom_schema {RESTRICT | CASCADE}

modifier_structure_table ::= ALTER TABLE nom_table ajouter_déf | modifier_déf | supprimer_déf ;

- ajouter_déf ::= ADD COLUMN déf_colonne | CONSTRAINT contrainte_table
- modifier_déf ::= ALTER déf_colonne
- supprimer_déf ::= DROP COLUMN nom_colonne | CONSTRAINT nom_contrainte

Ch.4

Commande d'interrogation :

- SELECT

Commandes de modification :

- ajout (INSERT),
- mise à jour (UPDATE)
- suppression (DELETE)

select_de_base ::= SELECT [ALL | DISTINCT] clause_de_sélection FROM nom_table [WHERE condition] ;

- clause_de_sélection ::= * | nom_table.* | liste_colonne
- condition ::= [NOT] condition_de_base | condition_between | condition_in | condition_like | condition_null | condition AND | OR condition | (condition)
  - condition_de_base ::= expression oper_comp expression
  - oper_comp ::= = | <> | < | <= | > | >=
  - expression ::= expression_numérique | expression_caractère | expression_date_temps | expression_intervalle

nom_col [NOT] LIKE 'modèle de chaîne' [ESCAPE escape-car]

- '_' remplace un seul caractère
- '%' remplace un nombre quelconque (éventuellement null) de caractères

NULL est une absence de valeur et le résultat de toute comparaison concernant NULL est inconnu.

COALESCE recherche la première valeur non NULL dans une liste de valeurs.

A UNION [ALL] B [CORRESPONDING [BY (liste_colonne)]]

A MINUS [ALL] B [CORRESPONDING [BY (liste_colonne)]]

A INTERSECT [ALL] B [CORRESPONDING [BY (liste_colonne)]]

- Jointure croisée, le produit cartésien : CROSS JOIN
- Jointure prédicative ou jointure manuelle : FROM A, B WHERE A.id = B.id ;
- Auto-jointure : FROM A, B WHERE COALESCE (A.idChef, A.id) = B.id
- Jointure naturelle : NATURAL JOIN (USING colonnes avec même nom)
- Jointure interne : INNER JOIN (ne considère pas les non-correspondances)

- Jointure externe : [RIGHT OUTER | LEFT OUTER | FULL OUTER] JOIN
- Jointure d'union : UNION JOIN (concatène)

RIGHT OUTER :
On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes de la table TableDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.
LEFT OUTER :
On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes de la table TableGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.
FULL OUTER :
On cherche tous les tuples satisfaisant la condition de jointure précisée dans le prédicat, puis on rajoute toutes les lignes des tables TableGauche et TableDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

expression_numérique ::= terme | terme { + | - } terme
- terme ::= facteur | terme {* | /} facteur
- facteur ::= [+ | -] constante | [+ | -] nom_colonne | [+ | -] fonction_de_calcul | [+ | -] fonction_de_conversion | (expression_numérique)
  - fonction_de_calcul ::= COUNT(*) | AVG([ALL|DISTINCT] expression) | MAX([ALL|DISTINCT] expression) | MIN([ALL|DISTINCT] expression) | SUM([ALL|DISTINCT] expression) | COUNT([ALL|DISTINCT] expression)
  - fonction_de_conversion ::= CAST(expression AS type_de_donnée | domaine)

expression_caractère ::= 'chaîne_constante' | nom_colonne | UPPER (expression_caractère) | LOWER (expression_caractère) | CHARACTER_LENGTH (expression_caractère) | USER | CAST (expression AS type_de_donnée|domaine) | SUBSTRING (expression_caractère FROM début FOR long) | expression_caractère {|| expression_caractère } | POSITION(expression_caractère IN expression_caractère) | TRIM(ltb, [pad,] FROM expression_caractère)

concat(ch1, ch2) : cette fonction est identique à ||.
initcap(ch) donne la chaîne ch dont le premier caractère a été converti en majuscules et les autres caractères en minuscules.
lower(ch) : cette fonction est identique à celle de la norme.
lpad(ch1, x [, ch2]) construit une chaîne de x caractères en complétant le début de ch1 par le nombre adéquat de fois la chaîne ch2. Par défaut, ch2 est le caractère blanc.
ltrim(ch1 [, ch2]) retire du début de ch1 toutes les occurrences de ch2. Par défaut ch2, est le caractère blanc.
replace(ch1, ch2 [, ch3]) donne la chaîne ch1 dans laquelle toutes les occurrences de ch2 ont été remplacées par ch3
rpad(ch1, x [, ch2]) construit une chaîne de x caractères en complétant la fin de ch1 par le nombre adéquat de fois la chaîne ch2. Par défaut ch2, est le caractère blanc.
rtrim(ch1, ch2) retire de la fin de ch1 toutes les occurrences de ch2. Par défaut ch2, est le caractère blanc.
substr(ch, x [, y]) extrait de ch, à partir de la position x, une sous-chaîne de longueur y. Si y est omis, substr extrait la sous-chaîne jusqu'à la fin de ch.
translate(ch1, ch2, ch3) donne la chaîne ch1 dans laquelle toutes les occurrences de chaque caractère de ch2 ont été remplacées par le caractère correspondant de ch3.

upper(ch) : cette fonction est identique à celle de la norme.
instr(ch1, ch2 [, x] [, y]) donne la position de ch2 dans ch1. La chaîne ch1 est parcourue à partir du x ème caractère. Si y est précisé, instr donne la position de la y ème occurrence de ch2 dans ch1.
length(ch) est identique à character_length.

```
CASE expression
        WHEN valeur1 THEN resultat1
        WHEN valeur2 THEN resultat2
        ...
        [ ELSE resultatn ]
END
ou
CASE
        WHEN condition1 THEN resultat1
        WHEN condition2 THEN resultat2
        ...
        [ ELSE resultatn ]
END
```

expression_date_temps ::= constante | nom_colonne | CAST (expression AS type_de_donnée | domaine) | CURRENT_DATE | CURRENT_TIME [précision] | CURRENT_TIMESTAMP [précision] | EXTRACT (champ FROM source)

OVERLAPS permet de tester si deux périodes de temps se recouvrent.

En Oracle, NULL > *.

Le résultat de la clause GROUP BY est une table intermédiaire constituée d'un ensemble de groupes de lignes réarrangées en groupes. Dans chaque groupe toutes les lignes possèdent la même valeur pour la combinaison de colonnes indiquées.

HAVING permet d'indiquer une conditions qui sera évaluée pour chaque groupe.

condition ::= [NOT] condition_élémentaire | condition AND | OR condition | (condition)

- condition_élémentaire ::= condition_de_base | condition_between | condition_in | condition_like | condition_null | condition_all_any | condition_exists
  - condition_in ::= expression [NOT] IN (sélection_une_colonne)
  - condition_all_any ::= expression oper_comp [ALL | ANY | SOME] (sélection_une_colonne)
  - condition_exists ::= EXISTS (sélection_une_colonne)

instruction_d_ajout ::= INSERT INTO nom_table [(liste_colonne)] VALUES (liste_valeur) | expression_de_sélection ;

instruction_de_modification ::= UPDATE nom_table SET liste_colonne_valeur [WHERE condition] ;

instruction_de_suppression ::= DELETE FROM nom_table [WHERE condition] ;

Ch.5

Une transaction est une unité de traitement cohérent et sûr.

Une base de données est dans un état cohérent si les valeurs contenues dans la base vérifient toutes les contraintes d'intégrité définies sur la base.

Une transaction est cohérente si elle fait passer la base d'un état cohérent à un autre état cohérent. Cependant, la base peut passer par des états intermédiaires incohérents.

Le terme atomicité signifie qu'une transaction doit être traitée comme une seule opération. Autrement dit, le gestionnaire des transactions doit assurer que toutes les actions de la transaction sont exécutées, ou bien qu'aucune ne l'est.

Trois issues sont possibles pour une transaction :

- Une vie sans histoire
- Un arrêt interne
- Un arrêt externe

La durabilité est la propriété qui assure que, lorsqu'une transaction a exécuté "valider", ses effets deviennent permanents et ne peuvent plus être effacés de la base. Ils doivent survivre à toute espèce de panne.

La commande SQL qui valide et termine une transaction est COMMIT.
La commande SQL qui défait et termine une transaction est ROLLBACK.

On dit que deux transactions sont concurrentes si elles accèdent en même temps aux mêmes données.

Trois types d'anomalies :

- Pertes de mise à jour
  - Ecrasement.
- Lecture impropre
  - Référence fantôme : ligne supplémentaire par une autre transaction.
- Lecture non reproductible
  - Résultat différent pour une même requête car modification par une autre transaction.

A Atomicité
C Cohérence
I Isolation
D Durabilité

L'isolation est la propriété des transactions qui exige que chaque transaction perçoive à tout instant la base dans un état cohérent.
En d'autres termes, une transaction en cours d'exécution ne peut pas dévoiler ses effets aux autres transactions concurrentes avant d'atteindre son point de confirmation.

Verrous courts => niveau instruction
Verrous long => niveau transaction

Degré	Perte de mise à jour	Lecture impropre	Données non reprod	Référence fantôme
0 - Read Uncommitted V court excl en écriture Pas de V en lecture	NON	OUI	OUI	OUI
1 - Read Committed **V lg excl en écriture** **V court ptg en écriture**	**NON**	**NON**	**OUI**	**OUI**
2 - Repeatable Read V lg ptg en lecture	NON	NON	NON	OUI
3 - Serializable V lg excl en lecture	NON	NON	NON	NON

SQL-agent : exécution d'un programme d'applications contenant une ou plusieurs requêtes SQL.
SQL-client : l'agent démarre l'exécution sous le contrôle d'un client
SQL-connection : pour pouvoir faire des accès à la base, l'agent doit forcer le client à établir une connexion avec un server
SQL-server : a pour rôle d'effectuer les accès à la base de données
SQL-environnement : le client et le serveur constituent les deux composantes d'un environnement

Une transaction est une unité logique de travail (suite atomique de commandes SQL)
Deux transactions ne peuvent pas être imbriquées
Une transaction est démarrée implicitement par un agent lorsqu'il exécute certaines commandes SQL appelées transaction-initiating statement. (on peut résumer en disant que toute commande du LDD et du LMD peut démarrer une transaction)
Les transactions sont terminées explicitement par COMMIT ou ROLLBACK.

L'instruction SET TRANSACTION est utilisée pour définir les caractéristiques de la prochaine transaction :

- Le mode d'accès (lecture seule (read only) ou lecture écriture (read write)),
- La taille de la zone de diagnostic (diagnotics area)
- Le niveau d'isolation (ISOLATION LEVEL READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE)

Dans Oracle, une transaction se termine dans un des cas suivants :

- exécution d'un COMMIT ou d'un ROLLBACK
- exécution d'une commande du LDD : cela provoque d'abord la validation de la transaction en cours, l'exécution de la commande du LDD suivie immédiatement d'un nouveau COMMIT
- déconnexion à Oracle : la transaction courante est validée
- fin anormale d'un processus utilisateur : la transaction courante est annulée

Un point de sauvegarde intermédiaire se définit par : SAVEPOINT point;

En Oracle : une lecture ne bloque pas une modification et vice versa.

Verrouillage explicite : SELECT ... FOR UPDATE [NOWAIT] ou LOCK TABLE
- usage du WHERE => verrou ROW SHARE

Trois cas de figure peuvent se présenter lors de la lecture du tuple avec pose de verrou :
- La ressource est occupée (-54)
- La ressource n'existe plus (NO_DATA_FOUND)
- La ressource est disponible.

Ch.6
4 grandes classes de techniques propres à assurer la confidentialité d'un système manipulant des données :
- Contrôle du flux des données,
- Contrôle d'inférences
- Encryptage des données
- Contrôle des autorisations d'accès

Les privilèges sont de deux types :
- niveau système : création, modification, suppression de groupes d'objets.
- niveau objet : manipulations sur des objets spécifiques.

accorder_privilege ::= GRANT privilege ON objet TO utilisateur [WITH GRANT OPTION];
- privilege ::= SELECT | INSERT | INSERT(x) | UPDATE | UPDATE(x) | DELETE | ALL
- objet ::= TABLE | VIEW + tables
- utilisateur ::= users | PUBLIC

retirer_privilege ::= REVOKE privilege ON objet FROM utilisateur;

Ch.7
On distingue 2 types de tables dérivées :
- les photographies
- les vues

Une photographie est une table contenant des données déduites à partir des tables de base (ou même d'autres photographies). Une photographie est stockées physiquement dans la base.

Tables temporaires => TEMPORARY

Une vue est une table dérivée dynamique. Elle n'est pas stockée physiquement dans la base. Le problème de cohérence ne se pose donc pas. Par contre, elle peut entraîner un ralentissement des traitements si la vue doit être générée souvent.

Créer_vue ::= CREATE VIEW nom_vue [(Liste_colonne)] AS expression_de_sélection [WITH CHECK OPTION];

Rien ne permet à un utilisateur de faire la distinction entre une table de base et une vue. Toute modification sur les tables source est reportée à la vue.

Une vue peut être utile pour :

- Augmenter l’indépendance logique
- Préserver la confidentialité des données

Ch.8
Le contrôle sémantique assure la cohérence des informations.

Contraintes inhérentes au modèle relationnel
- Intégrité de domaine
- Intégrité de relation
- Intégrité de référence

Contraintes liées à une application particulière
- Il s'agit d'une condition que doit vérifier un sous-ensemble de la base afin que l'on puisse affirmer que les informations sont cohérentes.

Pour Oracle, une contrainte déclarée DEFERRABLE pourra être différée pendant la durée d'une transaction à condition de le préciser explicitement au moyen de la commande SET CONSTRAINT

- INITIALLY IMMEDIATE correspond à NOT DEFERRABLE de la norme et est le comportement par défaut
- INITIALLY DEFERRED est équivalent à DEFERRABLE de la norme

Un déclencheur (trigger) permet de définir un ensemble d'actions qui sont déclenchées automatiquement par le SGBD lorsque certains phénomènes se produisent.

CREATE TRIGGER nom_déclencheur BEFORE | AFTER DELETE | INSERT | UPDATE | OF liste_colonne ON nom_table [FOR EACH ROW] [WHEN condition] [bloc PL/SQL];

Table mutante : table en cours de modification. Pour un déclencheur, il s'agit de la table sur laquelle il est défini.

Table contraignante : table qui peut éventuellement être accédée en lecture afin de vérifier une contrainte de référence.

PL-SQL

Ch.1

PL/SQL est un langage procédural qui permet de traiter de manière structurée (conditionnelle ou itérative) les données retournées par une instruction SQL.

PL/SQL n'a aucun aspect normatif contrairement à SQL. Mais avec SQL3, la norme SQL a prévu les éléments de langage procédural normatif propre au langage SQL.

SQL/PSM (Persistent Stored Modules) : Le langage normalisé des modules et procédures stockées.

```
DECLARE
/* section déclarative : variables, types et routines locales */
BEGIN
/* section exécutable : les instructions procédurales et SQL sont placées ici, c'est la seule section du bloc qui est obligatoire */
END;
```

Les instructions SQL incluses dans des blocs PSM sont considérées comme du SQL intégré (embedded SQL).

Les instructions de type SELECT ne peuvent alors retourner qu'une seule ligne : celles qui n'en retournent aucune ou plusieurs généreront une erreur (si elles ne sont pas utilisées dans un curseur).

Le clause INTO est donc obligatoire dans les blocs PSM, les valeurs retournées doivent être stockées dans des variables.

Meilleure performance en écrivant des blocs de programmation PL/SQL :

- Les ordres SQL ne sont plus transmis un à un au moteur de base de données Oracle mais par bloc de programmation => moins de trafic réseau
- Le moteur PL/SQL a été optimisé ce qui améliore encore les performances globales des applications

Pour un développeur :

- Programmation SQL dynamique, déclencheurs, méthodes des TAD (types abstraits de données)
- Intégration aisée dans les langages classiques OO grâce à sa forte coloration OO

Pour un administrateur, possibilité :

- d'écrire des routines d'administration
- de définir des jobs récurrents

PL/SQL permet la création de procédures stockées : terme générique qui signifie qu'une portion de code peut être stockée sur le serveur de base de données dans la base de données elle-même.

On peut voir sur ce schéma différents avantages :

- Réutilisabilité : une même procédure ou fonction peut être appelée par plusieurs applications différentes
- Optimisation : le DBA peut utiliser toutes les techniques d'optimisation mises à sa disposition pour les procédures les plus utilisées
- Efficacité : une procédure stockée est compilée une fois et stockée sous une forme exécutable. Les appels peuvent être efficaces et se font sous forme de RPC (Remote Procedure Call)

En résumé :

- On obtient une meilleure performance et moins de trafic réseau ainsi que de meilleurs temps de réponse
- Les procédures stockées font partie de la mémoire cache de la base de données et peuvent être partagées entre les multiples utilisateurs
- L'indépendance données-programmes s'en trouve renforcée ! (Les détails du MRD peuvent être cachés aux différents clients. Il suffit de communiquer au développeur la procédure à appeler ainsi que ses différents paramètres et codes d'erreur)

Ch.2

Quatre catégories :

- Scalaires : Ces types sont atomiques : ce sont les types utilisés pour définir une colonne d'une table
- Composés : Ces types comprennent plus d'un élément ou composant
- Références : Ces types permettent de définir des références vers d'autres types
- Grands Objets (LOB = Large Binary Object) : Ces types de données spécifient la localisation d'un "grand objet" binaire comme une image stockée dans la base de données ou un fichier externe.

1. Types de données numériques
   - NUMBER[(P, S)]
   - BINARY_INTEGER
   - PLS_INTEGER
   - BINARY_FLOAT et BINARY_DOUBLE

2. Types de données "caractères"
   - CHAR [(maximum size [CHAR | BYTE])]
   - VARCHAR2 (maximum size [CHAR | BYTE])
   - LONG et LONG RAW
   - RAW (maximum size)
   - ROWID ou UROWID
   - NCHAR[(maximum size)] et NVARCHAR2 (maximum size)

3. Types de données booléen
   - PL/SQL connaît les valeurs TRUE, FALSE et NULL
   - SQL ne permet pas de colonne booléenne

4. Types de données DATE, TIME et INTERVAL
   - Type DATE
   - Type TIMESTAMP [(précision)]

- Type INTERVAL DAY [(précision)] TO SECOND [(précision)]
- Type INTERVAL YEAR [(précision)] TO MONTH

Nom_Variable [CONSTANT] TYPE [NOT NULL] [{DEFAULT | :=] VALEUR];

- Nom_variable : ce nom de variable doit être unique dans le bloc
- TYPE : le type de la variable qui peut être un type scalaire ou un type composite
- CONSTANT : La valeur de la variable est une constante qui ne sera pas modifiable dans le bloc
- NOT NULL : la variable doit obligatoirement être assignée, sinon une erreur est générée
- := VALEUR : la variable est initialisée avec la valeur

Le type de données RECORD permet de déclarer des types de variables composites.

```
TYPE TupleEmploye IS RECORD (
        EmpNo       NUMBER(4),
        Ename       VARCHAR2(10),
        Job         CHAR(9),
        Mgr         NUMBER(4),
        HireDate    DATE,
        Sal         NUMBER(7,2),
        Comm        NUMBER(7,2),
        DeptNo      NUMBER(2));

UnEmploye TupleEmploye;
```

L'attribut %TYPE permet de déclarer une variable dont le type est basé sur le type d'une colonne ou d'une autre variable

CAST (expression AS type)

Opérateur	Opération
**	Exponentiation
-	Opposé
*, /	Multiplication, division
+, -, ||	Addition, soustraction concaténation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparaison
NOT	Négation logique
AND	Conjonction (ET Logique)
OR	OU Logique

Le PL/SQL reconnaît les pseudo-colonnes de SQL telles que ROWID, CURRVAL, NEXTVAL, LEVEL et ROWNUM

Ch.3

Structures conditionnelles :

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF
- CASE

```
[ <<label>> ]
CASE expression
        WHEN valeur THEN [,...]
        [ELSE ...]
END CASE [label];
ou
[ <<label>> ]
CASE
        WHEN condition THEN [,...]
        [ELSE ...]
END CASE [label];
```

Structures itératives :

- LOOP (boucle infinie, à éviter ! La boucle se finit avec EXIT)
- WHILE
- FOR

```
[ <<label>> ]
WHILE condition LOOP
        ...
END LOOP[label];
```

```
FOR compteur IN [REVERSE] borne_inférieure..borne_supérieure LOOP
        ...
        [EXIT [WHEN ...]]
        ...
END LOOP[label];
```

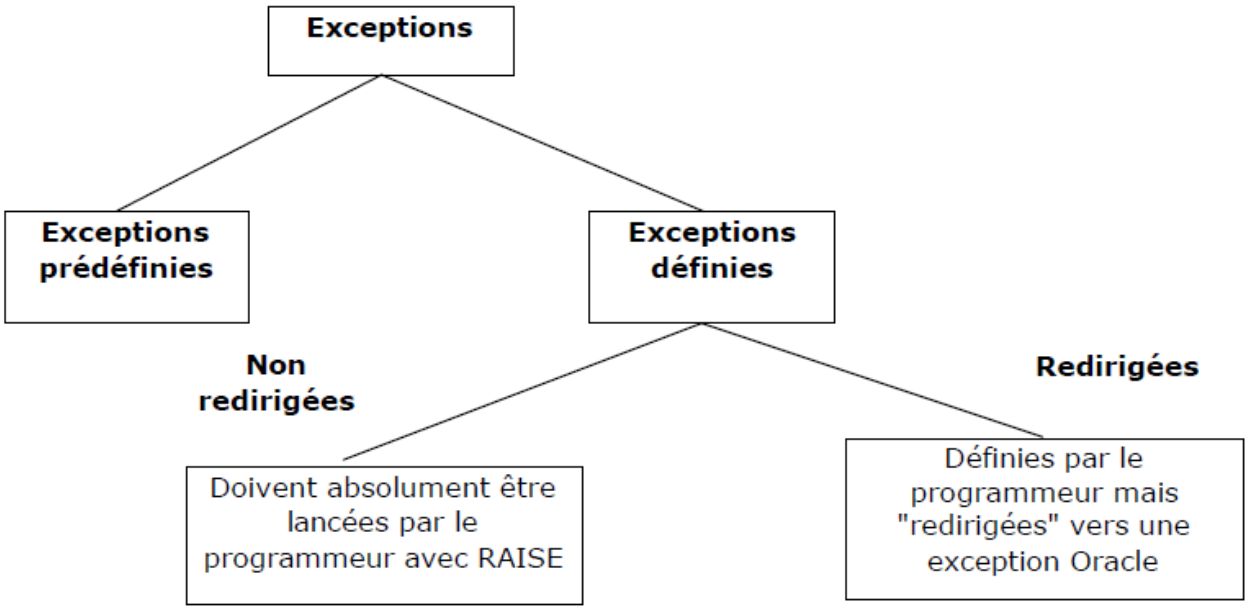
Ch.4

Déclaration des exceptions : Nom_de_l_exception EXCEPTION;

Une exception est lancée :

- Explicitement par un RAISE
- Implicitement par Oracle (exceptions prédéfinies)

La clause WHEN OTHERS dans un gestionnaire d'exceptions permet d'intercepter n'importe quel type d'exceptions non géré par ailleurs.




```
CREATE SEQUENCE SequenceEmp START WITH 1000;
DECLARE
        Ename Emp.Ename%TYPE;
        ExcEnameNULL EXCEPTION;
BEGIN
        IF Ename IS NULL THEN RAISE ExcEnameNULL; END IF;
        INSERT INTO Emp (Empno,Ename,HireDate) VALUES
                (SequenceEmp.NEXTVAL, Ename,CURRENT_DATE);
        COMMIT;
EXCEPTION
        WHEN ExcEnameNULL THEN
                DBMS_OUTPUT.PUT_LINE('Nom de l''employé est inconnu');
```

```
        WHEN OTHERS
                THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
```

Techniques des exceptions définies "redirigées" :

- Déclarer l'exception : NameExc EXCEPTION
- Associer le nom de l'exception à un numéro d'erreur Oracle : PRAGMA EXCEPTION_INIT (NameExc, Oracle_error);
- Référencer l'exception dans la section des gestions d'exceptions

Ch.5

Une collection est un ensemble, éventuellement ordonné, d'éléments de même type. Chaque élément est repéré au moyen d'un indice.

Le PL/SQL possède 3 types de collections :

- Les tableaux associatifs (associative arrays ou index-by tables)
- Les tables imbriquées (nested tables)
- Les tableaux prédimensionnés (variable-size arrays)

TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY [PLS_INTEGER | BINARY_INTEGER | VARCHAR2 (size_limit)];

EXISTS : Tester l'existence ou non d'un élément
COUNT : Compter le nombre d'éléments d'une collection
FIRST / LAST : Déterminer le premier et le dernier indice des éléments d'une collection
NEXT : Parcourir une collection
DELETE | DELETE(n) | DELETE(m, n) : Supprimer des éléments dans une collection

EXCEPTION	Déclenchée lorsque …
COLLECTION_IS_NULL	On utilise une collection *atomicaly null*
NO_DATA_FOUND	Un indice désigne un élément supprimé ou un élément qui n'existe pas dans une table PL/SQL
SUBSCRIPT_BEYOND_COUNT	Un indice dépasse le nombre d'éléments de la collection
SUBSCRIPT_OUTSIDE_LIMIT	Un indice est en dehors de la fourchette permise
VALUE_ERROR	Un indice est null ou ne peut être converti dans le type de l'indice

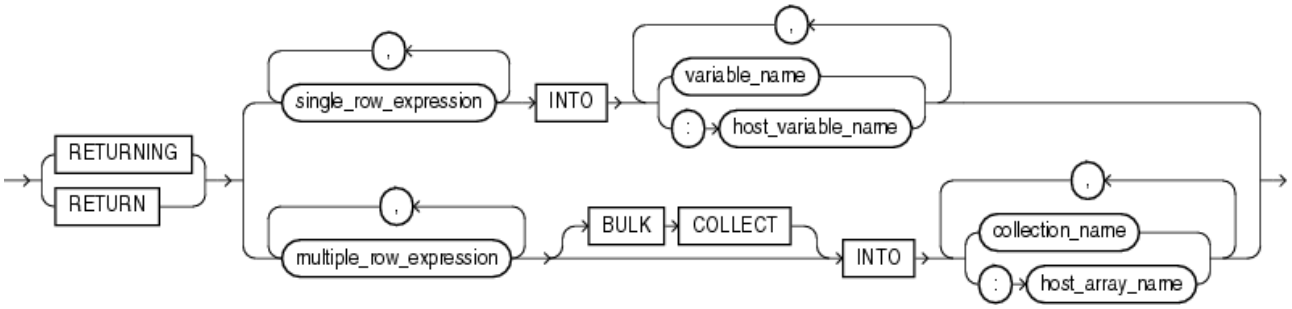
Ch.6

Les records permettent de définir des structures de données hétérogènes.

TYPE TypeDept IS RECORD (DeptNo NUMBER, Dname VARCHAR2(14), Loc VARCHAR2(13));
UnDepartement TypeDept;

ou
TYPE TypeDept IS RECORD (DeptNo Dept.DeptNo%TYPE, Dname Dept.Danme%TYPE, Loc Dept.Loc%TYPE);
UnDepartement TypeDept;
ou
UnDepartement Dept%ROWTYPE;

Pour comparer deux records, il faut les comparer champ par champ. Ne pas oublier d'utiliser COALESCE.



Dans une instruction FORALL, si l'exécution d'une instruction SQL provoque le déclenchement d'une exception non gérée, toutes les modifications réalisées par les instructions précédentes seront annulées.
Si l'exception est gérée, les modifications accomplies par les instructions précédentes NE seront PAS annulées.

Le nombre de lignes impliquées dans chaque instruction LMD d'un FORALL peut être déterminé au moyen de l'attribut composé SQL%BULK_ROWCOUNT

SELECT ... INTO ... FROM ... WHERE ... ;
- Le tuple est trouve : l'exécution continue
- Le tuple n'est pas trouvé : exception NO_DATA_FOUND
- La sélection renvoie plus d'un tuple : exception TOO_MANY_ROWS

La boucle FOR :
FOR VariableImplicite IN (SELECT ... FROM ... WHERE ...)
LOOP
    Traitement d'un tuple
END LOOP;
Un ou plusieurs tuples sélectionnés => exécution LOOP
Aucun tuple sélectionné => pas exécution LOOP, on continue l'exécution après LOOP
PAS d'exception NO_DATA_FOUND déclenchée !!!

SELECT * BULK COLLECT INTO VariableTable FROM ... WHERE ... ;
Un ou plusieurs tuples sélectionnés => initialisation de la VariableTable indicée de 1 à n
Aucun tuple sélectionné => on continue l'exécution après le SELECT
PAS d'exception NO_DATA_FOUND déclenchée !!!

INSERT INTO Emp (Empno,Ename,HireDate) VALUES (1234,'Thiry',CURRENT_DATE);
COMMIT;
ou

```
...
UnEmploye Emp%ROWTYPE;
...
INSERT INTO Emp VALUES UnEmploye;
COMMIT;
```

Exceptions:

- Intégrité d'entité : DUP_VAL_ON_INDEX
- Intégrité référentielle :

```
        ExcCleEtrangere EXCEPTION;
        PRAGMA EXCEPTION_INIT(ExcCleEtrangere,-2291);
```

- Contraintes applicatives

```
INSERT INTO dept2 (SELECT * FROM dept);
```

```
FORALL index IN LowerBound..UpperBound SqlStatement;
```

Avec la clause SAVE EXCEPTIONS, toutes les exceptions détectées pendant l'exécution de FORALL sont sauvées dans l'attribut %BULK_EXCEPTIONS (table PL_SQL).

- SQL%BULK_EXCEPTIONS.COUNT : nombre d'exceptions rencontrées pendant l'exécution du FORALL
- SQL%BULK_EXCEPTIONS(i).ERROR_INDEX : indice de l'itération qui a provoqué l'exception
- SQL%BULK_EXCEPTIONS(i).ERROR_CODE : code d'erreur d'Oracle

Le $n^{ième}$ élément de SQL%BULK_ROWCOUNT contient le nombre de lignes traitées dans la $n^{ième}$ instruction LMD.

Ch.7

CREATE PROCEDURE :
Chaque procédure ou fonction peut comprendre des paramètres. Pour chaque paramètre, on doit spécifier :

- Son nom
- Son mode d'accessibilité (IN, OUT ou IN OUT)
- Son type (pas de précision ni de longueur)
- Éventuellement sa valeur par défaut

```
CREATE OR REPLACE PROCEDURE Afficher
        (NumSecu IN Employes.NumSecu%TYPE)
AS
        UnEmploye Employes%ROWTYPE;
BEGIN
        SELECT * INTO UnEmploye FROM Employes WHERE NumSecu =
Afficher.NumSecu;
        DBMS_OUTPUT.PUT_LINE ('Nom : ' || UnEmploye.Nom);
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('Aucun employé trouvé');
        WHEN OTHERS THEN
```

```
            DBMS_OUTPUT.PUT_LINE ('ERREUR : '|| SQLCODE || SQLERRM);
END Afficher;
```

Compilation sous SQLPLUS :
SQL> START ProcAfficher.sql
Ou
SQL> @ ProcAfficher.sql

Lors de la compilation d'un objet, le moteur PL/SQL génère les messages d'erreurs dans la table ERROR$. Sous SQLPLUS, la commande SHOW ERRORS permet de visualiser les erreurs de compilation.

Le code est stocké dans le dictionnaire de données. Les sources des objets (procédures, fonctions, packages) sont mémorisés dans la table SOURCE$ (propriétaire SYS).

```
CREATE OR REPLACE FUNCTION Rechercher
        (NumSecu IN Employes.NumSecu%TYPE)
        RETURN Employes%ROWTYPE
AS
        UnEmploye Employes%ROWTYPE;
BEGIN
        SELECT * INTO UnEmploye FROM Employes WHERE NumSecu =
Rechercher.NumSecu;
        RETURN UnEmploye;
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE ('Aucun employé trouvé');
                RETURN NULL;
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('ERREUR : '|| SQLCODE || SQLERRM);
                RETURN NULL;
END Rechercher;
```

Par défaut, les paramètres OUT et IN OUT sont passés par valeur. Pour éviter des ralentissements lors de la copie de la valeur, il est possible de spécifier NOCOPY qui permet au moteur PL/SQL de passer les paramètres OUT et IN OUT par référence.

RAISE_APPLICATION_ERROR(code_erreur, message);

- Définie dans DBMS_STANDARD
- Plage de codes d'erreur de -20000 à -20999

Avec les notations nommées, on peut appeler une procédure de 3 manières différentes :

- Search_Client ('DELMAL', 'HUY');
- Search_Client (p_NomClient=>'DELMAL', p_AdresseClient => 'HUY');
- Search_Client(p_AdresseClient => 'HUY', p_NomClient => 'DELMAL');

Ch.8

Un package est composé de deux parties de code bien distinctes : la spécification (specification) et le corps (body);

- La spécification du package contient les prototypes des procédures et fonctions et la déclaration des variables publiques.
- Le corps du package contient l'implémentation des éléments définis dans la spécification. Le corps peut également contenir et définir des éléments privés.

```
CREATE OR REPLACE PACKAGE PublicVarPersistante IS vPublicVar CHAR := 'Y';
END PublicVarPersistante;
```

Avec la persistance de la session, il est possible :
- D'établir une session
- D'exécuter un programme qui assigne une valeur à une variable d'un package qui persistera jusqu'à la fin de la session

```
BEGIN
	DBMS_OUTPUT.PUT_LINE ('Valeur : ' || INPRES.PublicVarPersistance.VPublicVar)
END;
```

Si la variable est privée :

```
CREATE OR REPLACE PACKAGE BODY Session_Persistante
IS
	V_PrivateVar CHAR;

	PROCEDURE AssigneVariable (P_Valeur CHAR) IS
	BEGIN
		V_PrivateVar := P_Valeur;
	END AssigneVariable;

	FUNCTION AfficheVariable RETURN CHAR IS
	BEGIN
		RETURN V_PrivateVar;
	END AfficheVariable;
END SessionPersistante;
```

Principaux avantages des packages :
- Un package permet de regrouper plusieurs unités de programmation dans un même container => réduction du nombre d'objets stockés à maintenir
- Le fait de séparer l'implémentation de l'interface permet de maintenir ou modifier l'implémentation sans toucher à l'interface ni aux programmes appelants.
- Il n'est même pas nécessaire de recompiler les programmes appelants si l'interface n'est pas elle-même recompilée

Spécification du package contenant 2 fonctions polymorphes :

```
CREATE OR REPLACE PACKAGE Pkgclients IS
	FUNCTION RechercheClient (p_refClient IN clients.refClient%TYPE)
		RETURN Clients%ROWTYPE;
	FUNCTION RechercheClient (p_nomClient IN clients.nomClient%TYPE)
		RETURN Clients%ROWTYPE;
END PkgClients;
```

Ch.9

Un curseur peut être considéré comme une espèce de pointeur qui peut être utilisé pour parcourir un ensemble ordonné de tuples (active set)

```
DECLARE
        CURSOR C
                IS SELECT nom, prenom FROM Clients WHERE codepostal = '4000';
```

1. OPEN nom_curseur;
2. FETCH nom_curseur INTO Liste_variable | record;
   ou
   FETCH nom_curseur BULK COLLECT INTO collection;
3. CLOSE nom_curseur;

Chaque curseur possède 4 attributs :

- %FOUND vaut TRUE si un FETCH a pu extraire un tuple
- %NOTFOUND vaut TRUE si un FETCH n'a pas pu extraire de tuple
- %ISOPEN vaut TRUE si le curseur est ouvert
- %ROWCOUNT vaut le nombre de tuples qui ont déjà été extraits par des FETCH

		%FOUND	%NOTFOUND	%ISOPEN	%ROWCOUNT
OPEN	Avant	Exception	Exception	FALSE	Exception
	Après	NULL	NULL	TRUE	0
1er FETCH	Avant	NULL	NULL	TRUE	0
	Après	TRUE	FALSE	TRUE	1
Nème FETCH	Avant	TRUE	FALSE	TRUE	*
	Après	TRUE	FALSE	TRUE	*
Dernier FETCH	Avant	TRUE	FALSE	TRUE	*
	Après	FALSE	TRUE	TRUE	*
CLOSE	Avant	FALSE	TRUE	TRUE	*
	Après	Exception	Exception	FALSE	Exception

Curseur for implicite :

```
FOR Employe IN (SELECT * FROM Emp WHERE job LIKE 'A%')
LOOP
        DBMS_OUTPUT.PUT_LINE(Employe.nom);
END LOOP;
```

Curseur for explicite :

```
DECLARE
        CURSOR lesEmployes IS SELECT * FROM Emp WHERE sal > 2500;
BEGIN
        FOR UnEmploye IN lesEmployes
        LOOP
```

```
        DBMS_OUTPUT.PUT_LINE(lesEmployes%ROWCOUNT
                || ' ' || unEmploye.ename
                || ' ' || unEmploye.job);
        END LOOP;
EXCEPTION
        WHEN OTHERS
                THEN DBMS_OUTPUT.PUT_LINE (SQLERRM);
END;
```

Curseur avec paramètres :
CURSOR lesEmployes (x IN Emp.Job%TYPE) IS SELECT * FROM Emp WHERE job LIKE x || '%';

Curseur for update : (permet de modifier le tuple extrait par FETCH)
SELECT ... FROM ... FOR UPDATE [OF liste_colonnes] [WAIT/NOWAIT]

Le COMMIT DOIT être placé en dehors de la boucle car il ferme les curseurs ouverts.
Solution : plutôt qu'un curseur for update, utiliser un curseur "classique" et se servir de la clé primaire pour simuler la clause CURRENT OF

PL/SQL utilise un curseur implicite pour chaque opération du LMD de SQL (INSERT, UPDATE, DELETE). Ce curseur porte le nom SQL et il est exploitable après avoir exécuté l'instruction.

Ch.10

Un déclencheur (trigger) permet de définir un ensemble d'actions qui sont déclenchées automatiquement par le SGBD lorsque certains phénomènes se produisent.

```
CREATE TRIGGER nom_déclencheur
        BEFORE | AFTER
                DELETE | INSERT | UPDATE [OF liste_colonne]
                        ON nom_table | nom_vue
        [FOR EACH ROW]
        [WHEN condition]
[bloc PL/SQL];
```

Deux restrictions sur les commandes du bloc PL/SQL d'un déclencheur :

- Un déclencheur ne peut contenir de COMMIT ni de ROLLBACK
  - Il est impossible d'exécuter une commande du LDD dans un déclencheur
- Un déclencheur de niveau de ligne ne peut pas :
  - Lire ou modifier le contenu d'une table mutante (table en cours de modification. Pour un déclencheur, il s'agit de la table sur laquelle il est défini)
  - Lire ou modifier les colonnes d'une clé primaire, unique ou étrangère d'une table contraignante (table qui peut éventuellement être accédée en lecture afin de vérifier une contrainte de référence)

```
CREATE TRIGGER nom_déclencheur
        BEFORE | AFTER
                instruction_DDL | événement_système ON Schema | DATABASE
```

[bloc PL/SQL];

SERVERERROR	AFTER
LOGON	AFTER
LOGOFF	BEFORE
STARTUP	AFTER
SHUTDOWN	BEFORE

DBMS_STANDARD contient un ensemble de fonctions prédéfinies très utiles (type d'objet concerné, nom de l'objet, propriétaire de l'objet, ...)