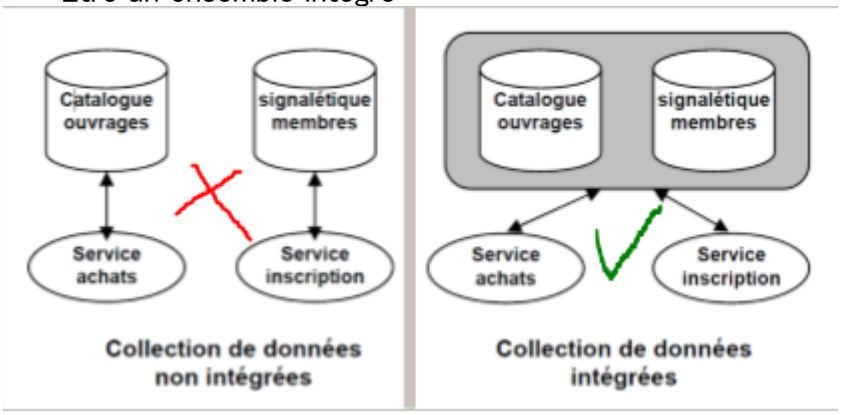
CHAPITRE 1 CONCEPTES DE BASE

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 :

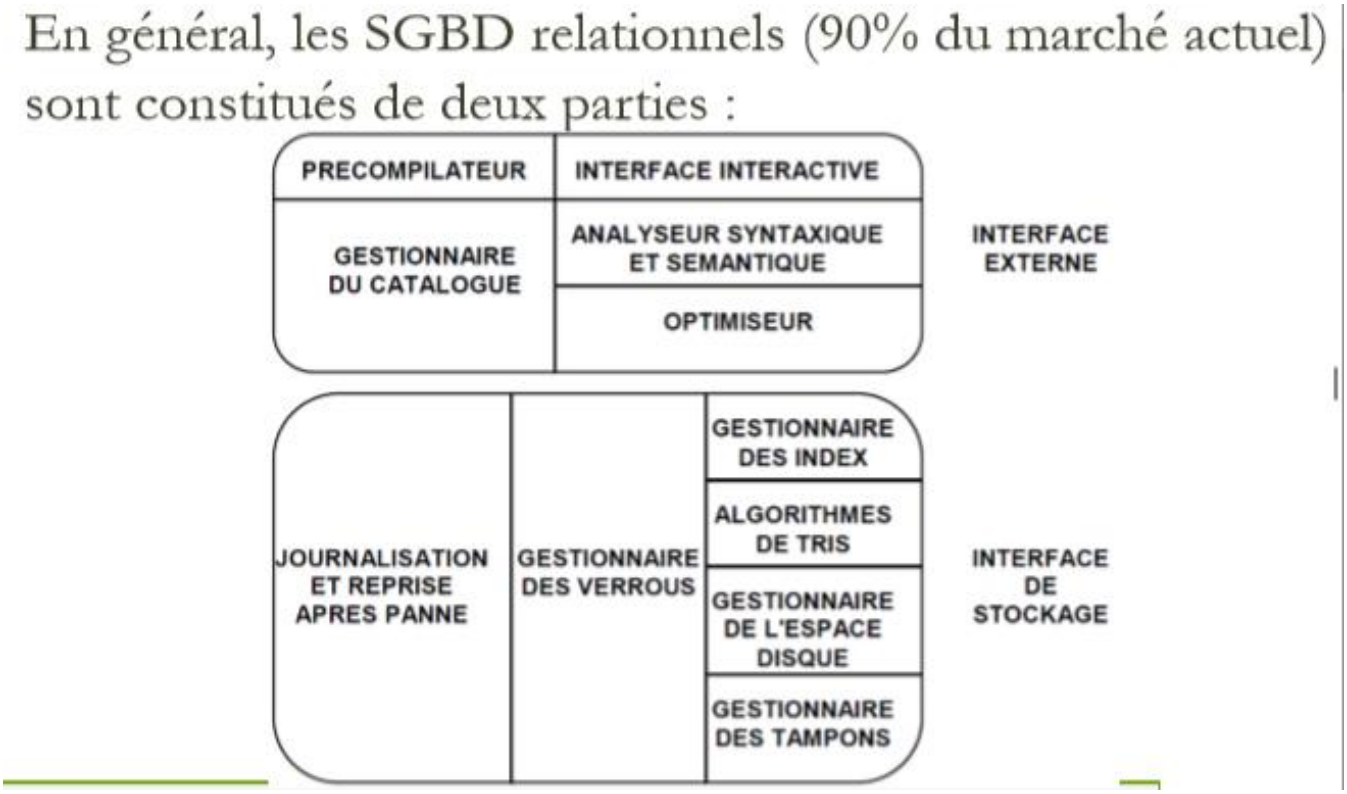
- Être un ensemble organisé/structuré (Fichier des prêts : lecture et écriture; Biblio : lecture...)
- Être un ensemble intégré



- Correspondre fidèlement à la réalité (Contraintes intégrités)
- Contenir les données opérationnelles sur un sujet donné (Entité + association)
- Être multi-utilisateurs
- Être non-redondante ou à redondance contrôlée (Trigger = contrôle de la redodance)

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

Voir Chapitre 5 : A C I D



- Interface de stockage : s'occupe de tout ce qui concerne l'accès aux données stockées sur disques
- La journalisation et reprise après pannes : assure la fonction de sécurité de fonctionnement
- Le gestionnaire des verrous : assure la fonction de concurrence d'accès

Le CATALOGUE ou METABASE ou DICTIONNAIRE ou TABLES SYSTEMES (VOIR SCHEMA AU DESSUS)

- Est mis à jour automatiquement par le SGBD
- Constitue une mini base de données contenant des informations sur la BD
- Est stocké dans la BD elle-même et peut être interrogé comme les données de la BD elle-même
- Contient la description de tous les objets présents dans la base
- Elle est dans la base

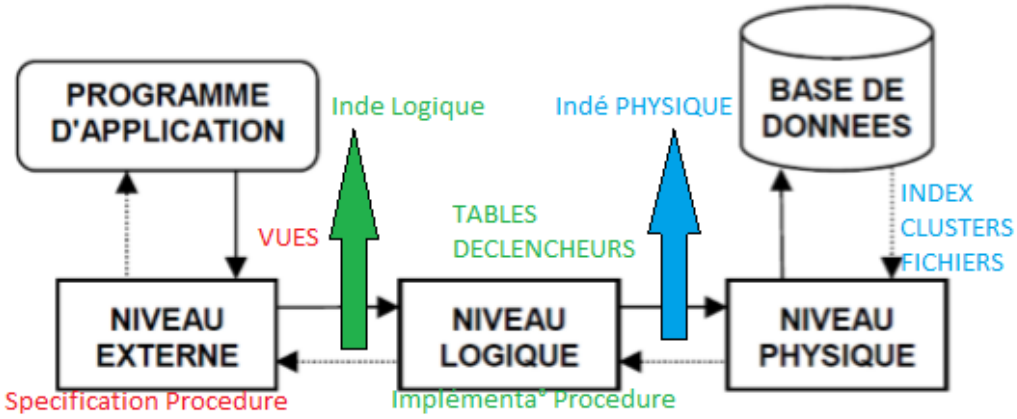
Objectif d'une BD :

- L'indépendance des données par rapport aux programmes de traitements (Niveau logique et physique)

Logique : on peut changer la structure logique globale sans devoir changer les programmes.

Physique : la couche physique et l'organisation des données peuvent changer sans devoir changer la structure logique globale ou les programmes.

3. Indépendance des données et des programmes



- La prise en compte des associations entre les différentes données
- Le partage simultané des données entre plusieurs utilisateurs

3. Indépendance données - programmes

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	*	*	

CHAPITRE 2 LE MODELE RELATIONNEL

Un **PREDICAT** est une phrase qui peut comporter des paramètres et qui peut être vraie ou fausse

Les opérateurs relationnels sont des opérateurs ensemblistes.

Enorme gain de temps 25% à 75% avec un BD plutôt que des fichiers.

1. Introduction

Objets	Contraintes
1. Relation 2. Domaine/attribut 3. Clé primaire 4. Domaine primaire (clé étrangère)	5. Intégrité de domaine 6. Intégrité d'entité ou de relation 7. Intégrité de référence

Opérateurs
8. Opérateurs sémantiques (liés aux domaines) 9. Opérateurs ensemblistes : union, intersection, différence, produit cartésien 10. Opérateurs relationnels : restriction (sélection), projection, jointure, division

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 }

Les **relations** sont définies à partir de domaines.

Pour définir une base de données relationnelle, on commence par définir les domaines.

Exemple :

Domaine **NumeroAuteur** = entier compris entre 1 et 100

Domaine **NomAuteur** = chaîne de caractères

Il est logique que la colonne NumAuteur de AEcrit soit définie sur le même domaine que la colonne NumAuteur de Auteurs.

Le domaine NumeroAuteur joue le rôle de **connecteur sémantique** entre les deux relations

Exemple : les domaines VilleEurope et VilleBelge,

Les villes belges etant en Europe le domaine est compatible !
ATTENTION : Domaine n'exite pas en ORACLE

Les Clefs Primaires :

- ➢ Une valeur de clé primaire permet d'identifier de manière unique un tuple d'une relation.

Une clé primaire doit être :

- Unique et NON NULL
- Minimaliste

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

Les contraintes :

Il existe deux grandes classes de contraintes d'intégrité :

- Les **contraintes structurelles** dépendant du modèle de données (intégrité de domaine, d'entité ou de relation et de référence)
- Les **contraintes applicatives** liées à l'univers réel modélisé.

INTEGRITE DE DOMAINE :

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.

INTEGRITE DE RELATION :

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).

INTEGRITE DE REFERENCE :

l'**intégrité référentielle** est une situation dans laquelle pour chaque information d'une table A qui fait référence à une information d'une table B, l'information référencée existe dans la table B.

Les Opérateurs ensemblistes :

Les opérateurs ensemblistes de base sont binaires : à partir de deux relations, ils en génèrent une troisième.

Les opérateurs que nous allons étudier :

- Union
- Différence
- Intersection (qui peut aussi être définie à partir de la différence)
- Produit cartésien

L'union, la différence et l'intersection ne s'appliquent qu'à des relations "**union-compatibles**".

C'est à dire : Même nombres d'attribut (colonnes + même domaines).

L'union :

Exemple :

$A = \{2, 3, 4, 5\}$

$B = \{1, 3, 6, 8\}$

$C = \{2, 5, 10\}$

$$A \cup B = \{1, 2, 3, 4, 5, 6, 8\}$$

$$A \cup C = \{2, 3, 4, 5, 10\}$$

La Difference : A/B = On garde tout ce qu'il y a dans A qui n'est pas dans B (il n'y a pas de valeur de B)

Exemple :

$A = \{2, 3, 4, 5\}$

$B = \{1, 3, 6, 8\}$

$C = \{2, 5, 10\}$

$$A \setminus B = \{2, 4, 5\}$$

$$B \setminus C = \{1, 3, 6, 8\}$$

Le Produit cartésiens :

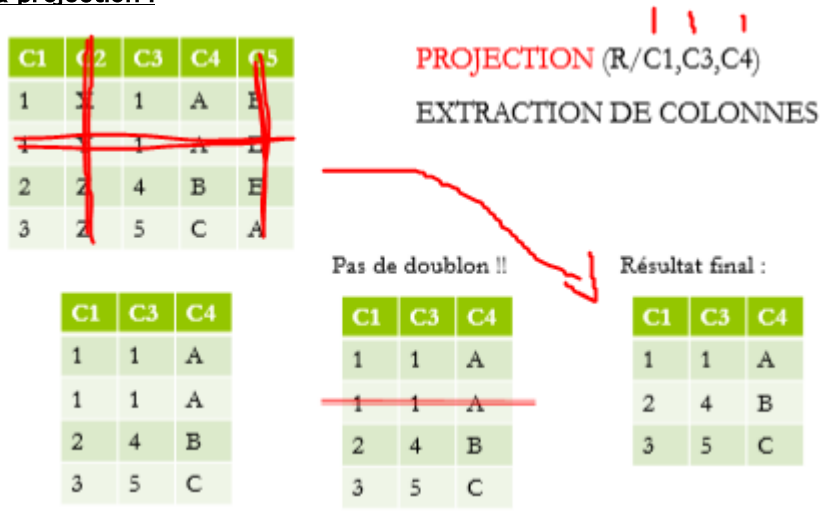
Exemple :

$A = \{2, 3\}$

$B = \{1, 3, 6\}$

$$A \times B = \{(2, 1), (2, 3), (2, 6), (3, 1), (3, 3), (3, 6)\}$$

La projection :



Intersectiion :

Exemple :

$A = \{2, 3, 4, 5\}$

$B = \{1, 3, 6, 8\}$

$C = \{2, 5, 10\}$

$A \cap B = \{3\}$

$B \cap C = \{\}$

VOIR LES JOINTURES :

CHAPITRE 3 LDD

Créer un domaine :

CREATE DOMAIN *nom* **type** [*valeur*] ;

Exemples :

```
CREATE DOMAIN NomAuteur CHAR (20);
CREATE DOMAIN StatutEmploye SMALLINT DEFAULT 10;
```

Les contraintes : CONSTRAINT :

- Syntaxe :

```
<contrainte de valeur> ::=
    [CONSTRAINT nom_contrainte]
    {[NOT] NULL | DEFAULT <expression_defaut>
        | CHECK (expression_validation)}
```

♥ ♥

```
CREATE TABLE Membres
( NumMembre          VARCHAR2(10)
      CONSTRAINT CPMemb PRIMARY KEY NOT DEFERRABLE,
  NomMembre          VARCHAR2(30)
      CONSTRAINT NNMembNom NOT NULL,
  PrenomMembre       VARCHAR2(30)
      CONSTRAINT NNMembPrenom NOT NULL,
  AnneeNaiss         NUMBER(4)
      CONSTRAINT NNMembAnnNaiss NOT NULL,
  Sexe               CHAR(1)
      CONSTRAINT NNMembSexe NOT NULL
      CONSTRAINT CKMembSexeMF CHECK (Sexe IN ('M', 'F')),
  CONSTRAINT CKMembNomPrenUnique UNIQUE (Nom, Prenom)
);
```

♥ ♥

Les Types de données : (les plus frequents)

Type	Description
CHAR (n [BYTE | CHAR])	Chaîne fixe de n caractères ou octets
VARCHAR2 (n [BYTE | CHAR])	Chaîne variable de n caractères ou octets
NCHAR (n)	Chaîne fixe de n caractères Unicode

Type	Description
NUMBER	7456123,89
NUMBER (9)	7456124
NUMBER (9,2)	7456123,89

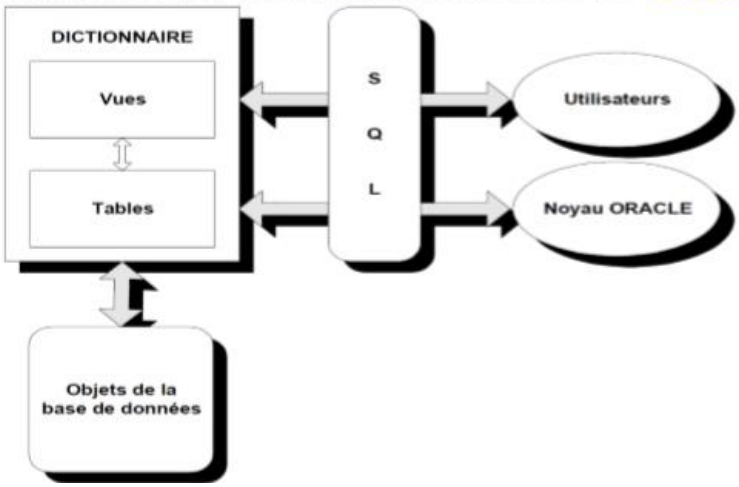
Les Schemas :

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

Un **schéma** contient des objets du monde relationnel: table, vue, index, ainsi que des déclencheurs, fonctions, procédures stockées et packages …

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

Structure et utilisation du dictionnaire de données en **Oracle** :



Les Index :

But et caractéristiques des index :

Augmenter la rapidité d'accès aux tuples en proposant un accès plus direct que l'accès séquentiel.

Questions :

- Un index peut être défini sur plusieurs colonnes V/F
- Un index ne porte que sur une seule relation V/F
- Une relation peut posséder plusieurs index V/F

Les techniques d'index sont essentielles pour construire des bases de données efficientes.

Résumé des principales caractéristiques de chaque méthode :

- Les index hiérarchiques (B-tree)
  - offrent des recherches rapides en fonction d'une valeur précise de la clé ou d'une plage de valeurs pour la clé
  - conviennent pour les colonnes qui ont un grand nombre de valeurs différentes et qui interviennent dans les critères de recherches ou de jointures
  - en pratique, on en définit toujours pour les clés primaires et les clés étrangères
- Les clusters
  - En regroupant physiquement les lignes de plusieurs tables, améliorent les temps de jointures.
  - MAIS pénalisent les parcours séquentiels
- Les index hachés
  - permettent en une seule opération d'entrée/sortie, de retrouver un tuple dont on donne la valeur de la clé.
- Les index à matrices binaires
  - sont utiles pour les requêtes de comptage en fonction de critères portant sur plusieurs colonnes contenant peu de valeurs différentes
  - sont utilisés pour de grandes tables

Supprimer une table :

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

CHAPITRE 4 CONTROLE DES DONNES

- L'opérateur AS sert à donner un nom à une colonne sélectionnée ou calculée dans une requête

  Ex : `SELECT COUNT(*) AS Nbre FROM Employes;`

Attention pour une comparaison :

```
CREATE TABLE Chaines
(  NomChar4          CHAR(4),
   NomChar10         CHAR(10),
   NomVarChar4       VARCHAR(4),
   NomVarChar10      VARCHAR(10)   );

INSERT INTO Chaines VALUES ('SQL', 'SQL', 'SQL', 'SQL');
COMMIT;

SELECT * FROM Chaines WHERE NomChar4 LIKE NomChar10;
```

- Aucun résultat !!!
- LIKE sans "joker"
- La comparaison tient compte de la longueur du type de la donnée

ATTENTION !
UNE VALEUR NULL DONNE UN RESULTAT INCONNU PAS VRAI NI FAUX

Coalesce :

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

Ex :

```
SELECT nom,
     COALESCE (numchef, 'Président') AS Chef
FROM employes;
```

Elle remplace les valleurs NULL de la colonne Numchef par “President”.

La recherche de nom ou autre (remplacement de caractère) :

Le caractère % et ___ :

Le caractère % remplace toute une partie de la chaine de caractere alors que __ remplace 1 seul carac.

Ex ; SELECT nom from Clients

Where nom LIKE ‘ %a’ ;

On va selection tout les noms qui se finissent pas a;

Alors que ‘_a%’ selection tout les noms qui ont pour deuxieme lettre a.

Les SELECTIONS IMBRIQUEES :

Les sélections imbriquées représentent un concept du SQL qui contribue grandement à sa puissance et à sa souplesse : la sélection imbriquée ou sous-question.

Une sélection imbriquée n'est rien d'autre qu'un bloc de qualification SFW encapsulé à l'intérieur d'un autre bloc de qualification.

Exemple : Rechercher le nom et le prénom des employés qu travaillent sur le projet p10347

```
SELECT Nom, Prenom
FROM Employes
WHERE NumSecu IN (SELECT NumSecu
                  FROM EmpPro
                  WHERE NumPro = 'p10347');
```

Le système commence par évaluer le bloc interne.

La requête initiale est alors équivalente à :

```
SELECT Nom, Prenom
FROM Employes
WHERE NumSecu IN ('192356', '123457', '121212');
```

Manipulation de date (voir chap 4 page 106) :

- **Expressions d'intervalles**

Exemple : afficher l'âge des employés.

```
SELECT nom, DateNais,
  EXTRACT(YEAR FROM CURRENT_DATE)
    - EXTRACT (YEAR FROM DateNais) Age
FROM Employes;
```

ATTENTION : le nombre d'années entre le 31/12/2018 et le 01/01/2019 vaut 1 !!!!

➢ L'opérateur OVERLAPS

SQL2 possède un opérateur spécial : **OVERLAPS.**

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

Ces périodes de temps peuvent être représentées de deux manières différentes : un temps de départ et un temps de fin ou un temps de départ et un intervalle.

Pour plus d'info, voir les documents de référence.

Comment trier :

Exemple : Afficher, trié par âge décroissant, le nom des employés du départements 'Application telecom'

```
SELECT Nom, DateNais,
  (CURRENT_DATE - DateNais) YEAR TO MONTH Age
FROM Employes, Departements
WHERE Employes.NumDep = Departements.NumDep
  AND NomDep = 'Applications telecom'
ORDER BY Age DESC;

Ou ORDER BY 3 DESC
```

Le terme GROUP BY :

expression_de_sélection ::=

SELECT [**ALL** | **DISTINCT**] *clause_de_sélection*
FROM *liste_table_ref*
[**WHERE** *condition*]
[*clause_grouper*]
[*clause_sélection_groupes*];

clause_grouper ::= **GROUP BY** *liste_colonne*

clause_sélection_groupes ::= **HAVING** *condition*

L' operateur EXISTS:

8. Utilisation de "EXISTS"

L'argument du EXISTS contient une référence externe (Employes.NumSecu). L'argument de EXISTS contient presque toujours au moins une référence externe.

L'opérateur EXISTS est de loin l'opérateur le plus puissant de SQL

- On peut éviter l'utilisation de ANY et ALL
- On peut exprimer des jointures
- On peut réaliser des intersections, différences, divisions

Instertion :

Exemple : On souhaite ajouter un nouveau département. Il s'agit du département "Infographie", dont on ne connaît pas le chef.

```
INSERT INTO departements (numdep,nomdep,numsecu)
     VALUES ('d00006', 'Infographie', NULL);

INSERT INTO departements (numdep,nomdep)
     VALUES ('d00006', 'Infographie');
```

Remarque : les colonnes pour lesquelles on ne spécifie pas explicitement une valeur sont initialisées à leur valeur par défaut.

Mise à jour :

Exemple : Attribuer à DE NIRO le salaire maximum du département auquel il est attaché

```
UPDATE employes emp1
   SET bareme = (SELECT MAX(bareme)
                 FROM employes
                 WHERE numdep = emp1.numdep)
WHERE nom = 'DE NIRO'
  AND prenom ='Robert';
```

Suppression :

Exemple : Supprimer toutes les attributions de Clémentine CELARIE à un projet

```
DELETE FROM EmpPro
WHERE numsecu
    IN ( SELECT numsecu
         FROM employes
         WHERE nom = 'CELARIE'
           AND prenom ='Clémentine') ;
```

CHAPITRE 5 CONTROLE DES DONNES

I° Problèmes qui peuvent survenir lors de l'exploitation d'une base de données

➢ Deux utilisateurs tentent de modifier la même donnée au même moment

➢ Une panne se produit pendant l'exécution d'une transaction

- Ces deux problèmes sont résolus grâce au concept de ***Transaction.***
- Une ***Transaction*** est une unité de traitement cohérente et sûr.

  Toute requête doit être exécutée dans le contexte d'une transaction.

II °Notions de BASE

Une base de données est dans un ♥ ***Etat 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, lors des mises à jour par des transactions cohérentes, la base peut passer par des ***états non cohérent.***

Les actions (de la transaction) sont des unités de traitement indivisibles.

- Une des grandes propriétés des transactions est ♥ **l'Atomicité** :

Le gestionnaire des transactions doit assurer que **TOUTES** les actions de la transaction sont exécutées, ou bien qu'**AUCUNES** ne l'est.

- Si un problème survient lors de la transaction :

Completer la transaction en exécutant les actions restantes.

Défaire les actions qui avaient été exécutées avant la panne.

Le début d'une transaction, noté **DEBUT** , la fin : **Valider** ou **Annuler**

A°) Trois issues possibles :

- *Vie sans histoire* : (tout se passe bien jusqu'au point de validation/confirmation).

La ♥ **Durabilité** ,lorsqu'une transaction a exécuté ″valider″, ses effets deviennent permanents sur la BD.

i. La commande SQL qui valide et termine une transaction est **COMMIT**

- *Un arrêt interne* : (Erreur de conversion. la BD annule toute la **transaction**).
    i. La commande SQL qui défait et termine une transaction est **ROLLBACK**

- *Un arrêt externe* : (action utilisateur/interblocage choix d'une transaction victime.

    i. mécanisme de **REPRISE APRES PANNE** (ou processus) de **RECOVERY**

III° Reprise après panne

➢ Il doit assurer que les effets d'une transaction validée apparaîtront dans la base après la reprise

➢ Il doit assurer que les effets d'une transaction annulée n'apparaîtront pas dans la base après la panne

IV° Transactions concurrentes

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

A°) Trois types d'anomalies :

➢ Pertes de mise à jour : (deux lectures T1 et T2 et deux actions T1 et T2 donc resultats ne tenant pas compte de l'action sur T1).

➢ Lecture impropre : (T1 modifie la donné sans COMMIT, T2 lit la donné non commit, lecture impropre).

➢ Lecture non reproductible (T2 puisse ses données dans une base incohérente (entre deux états).

B°) D'autres type d'anomalies (qui découlent des premières) :

Une lecture ou référence fantôme peut découler d'une lecture impropre.

Ex : T1 lit; T2 **INSERT** un tuple; T1 relit; un tuple sortie de nulle part : **Tuples/reference fantomes.**

V° Jusqu'à présent, nous avons vu 3 propriétés des transactions

- **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. On introduit donc la notion de **Verrou ♥**.

1. Verrou courts --> Niveau instruction.
2. Verrou longs --> Niveau transaction.

◂♥ 4 niveaux d'isolation standardisés par SQL2 (Set Transaction Isolation Level): ♥▸

- **DEGRE 0 : READ UNCOMMITTED.**

- **DEGRE 1 : READ COMMITED. (PAR DEFAUT ☺)**
- **DEGRE 2 : REPEATABLE READ.**
- **DEGRE 3 : SERIALIZABLE.**

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

VII° Spécification SQL 2

L'établissement d'une connexion entre le client et le serveur est réalisé au moyen de la commande **CONNECT** . Cette commande démarre une session (**SQL-session**) . Quand la connexion est établie et la session initialisée, l'agent peut exécuter des transactions (**SQL-transaction**) jusqu'à ce qu'il exécute l'instruction **DISCONNECT** qui arrête la connexion.

Une **transaction est une unité logique de travail** .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.** Les transactions sont terminer explicitement par **COMMIT** ou **ROLLBACK** .

L'instruction 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

VIII° Etudes de cas dans Oracle 10

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

Commit/Rollback ou exécution d'une commande de LDD ou deconnexion d'Oracle, fin anormale.

Oracle implémente également la notion de **SAVEPOINT (nom de la save)** et du **ROLLBACK (nom de la save).**

Par défaut, Oracle garantit la cohérence en lecture au niveau opération mais ne garantit pas une lecture cohérente au niveau transaction. Pour réaliser des lectures cohérentes au niveau transaction, il faut démarrer une transaction read only au moyen de la commande set transaction read only ou utiliser des verrous explicites.

◄♥En Oracle : Une lecture ne bloque pas une modif et vice versa ♥►

Rappel : par défaut, en Oracle, les lectures ne sont pas reproductibles .

En résumé, en Oracle , lorsque l'on travaille avec les options :

- Il n'y a pas de lecture impropre
- Il n'y a pas de pertes de mises à jour
- Il est possible de provoquer une référence fantôme
- Les lectures ne sont pas forcément reproductibles

Oracle empêche toute lecture impropre.

- Le niveau est le niveau **READ COMMITED ♥ (Niveau 1)** par défaut.
- Oracle ne supporte pas directement le niveau REPEATABLE READ (il ne le supporte qu'on mode READ ONLY)
- En écriture, ce niveau est inclus dans le niveau SERIALIZABLE.
- Le niveau d'isolation est spécifié au moyen de la clause ISOLATION LEVEL de la commande SET TRANSACTION

<u>Oracle permet au programmeur de placer des verrous explicitement. Ceci se fait au moyen des commandes :</u>

- **SELECT ... FOR UDAPTE [NOWAIT]**
- **LOCK TABLE**

Verrouillage explicite dans Oracle

```
SELECT  ...
FROM ...
WHERE ...
FOR UPDATE [NOWAIT];
```

Cette commande verrouille explicitement les tuples répondant à la condition de la clause WHERE

⟶ **verrou ROW SHARE**

Les tuples sont verrouillés en vue d'être modifiés. Une autre transaction pourra lire ces tuples mais pas les modifier.

Accès/transaction double :

Exemple : Un étudiant a changé d'adresse et va au service étudiant pour le faire savoir.

- Lecture de la donnée à modifier (recherche de l'étudiant à modifier)
- Encodage de la nouvelle valeur
- Mise à jour de cette valeur

Trois cas de figure peuvent se présenter lors de la lecture du tuple avec pose de verrou :

- La ressource est occupée (−54) (On attend avant de faire une tentative, on se limite à 3 tentatives).
- La ressource n'existe plus (NO_DATA_FOUND) (Donnée supprimer on arrête tout)
- La ressource est disponible.

Il faut vérifier qu'elle n'a pas été modifiée entretemps par un autre utilisateur On va alors comparer l'ancien tuple à celui que l'on vient de lire. La comparaison se fait champ par champ. Si les 2 tuples sont identiques, on va pouvoir insérer le tuple modifié et valider S'ils sont différents, le tuple a été modifié entretemps par un autre utilisateur, la modification courante est alors abandonnée. Ne pas oublier de libérer le verrou

CHAPITRE 6 CONFIDENTIALITE DES DONNES

La sécurité des données est un terme général contenant trois grands types de contrôle :

- Le contrôle d'accès au système par des utilisateurs non identifiés; (L'os prend en charge l'authentification, mdp window etc).
- Le contrôle de l'accès illégal aux données ou confidentialité; HORS COURS ☻
- Le contrôle de la modification invalide des données ou intégrité.

Dans ce chapitre, nous allons donc nous attarder sur la **CONFIDENTALITE** . Plus précisément, no nous limiterons au contrôle des autorisations d'accès

4 grandes classes de techniques propres à assurer la confidentialité d'un système manipulant des données :

- Contrôle du flux des données, HORS COURS ☻
- Contrôle d'inférences HORS COURS ☻
- Encryptage des données HORS COURS ☻
- Contrôle des autorisations d'accès

Les privilèges sont de deux types :

- Les privilèges de niveau **Système**

Permettent la création, modification, suppression de groupes d'objets. o Exemple : CREATE TABLE CREATE VIEW, CREATE SEQUENCE, permettent, à l'utilisateur qui les a reçus de créer des tables, vues et des séquences.

- Les privilèges de niveau **Objet**

Permettent les manipulations sur des objets spécifiques. Les privilèges SELECT, INSERT, UPDATE, DELETE sur la table INFOSOFT.employes permettent à l'utilisateur qui les a reçus de sélectionner, ajouter, modifier et supprimer des lignes dans la table EMPLOYES appartenant à l'utilisateur INFOSOFT.

Lorsqu'un utilisateur est créé avec l'instruction **CREATE USER**, il ne dispose d'aucun droit. Il ne peut même pas se connecter à la base **!**

Il doit pouvoir se connecter, créer des tables, des vues, des séquences. Pour lui assigner ces privilèges de niveau système, il faut utiliser l'instruction **GRANT.**

Exemple :

- **GRANT CREATE SESSION TO nom_utilisateur;**
- **GRANT CREATE TABLE TO nom_utilisateur;**
- **GRANT CREATE VIEW TO nom_utilisateur;**

SESSION_PRIVS pour voir les privileges en cours sur la session._

A°) Controles des autorisation (niveau objet)

C'est le dico des données (méta-base) qui enregistre les privilèges d'accès (octroi/annulation et d contrôle des autorisations).

Le mécanisme octroi/annulation et de contrôle des autorisations permet à l'ADB **d'accorder** ou d **retirer les privilèges** à des **sujets** sur des **objets** .

Sujet : un utilisateur, un groupe d'utilisateur, tous les utilisateurs .

Objet : la base de données, les tables, les vues, les index, les procédures stockées,...

Privilèges : varient d'un SGBD à l'autre, mais au minimum SELECT, INSERT, DELETE, UPDATE.

Le mécanisme de contrôle d'autorisation regarde la méta-base pour voir les droits de l'utilisateur.

Principes généraux :

➢ Un utilisateur possède automatiquement tous les privilèges sur un objet qui lui appartient

➢ Un utilisateur ne peut pas donner plus de privilèges qu'il n'en a reçus

➢ S'il n'a pas reçu le privilège avec l'option WITH GRANT OPTION, un utilisateur ne peut pas donn son tour ce privilège à un autre utilisateur

Accorder un privilège : GRANT

```
accorder_privilege ::=
     GRANT privilege ON objet TO utilisateur
     [ WITH GRANT OPTION];
```

privilege : SELECT, INSERT, INSERT(x), UPDATE, UPDATE(x), DELETE, ALL (= tous les privilèges que le donneur peut accorder sur l'objet)

objet : liste de tables, vues ou colonnes précédée de TABLE ou VIEW suivant qu'il s'agit d'une table ou d'une vue

utilisateur : liste d'utilisateurs ou PUBLIC

WITH GRANT OPTION : permet au donneur d'indiquer que le receveur pourra transmettre les privilèges qu'il reçoit.

Exemple :

```
GRANT ALL ON TABLE resultats TO directeur
WITH GRANT OPTION;
```

Supprimer un privilège : REVOKE

```
REVOKE SELECT ON TABLE t FROM Obelix;
```

CHAPITRE 7 CONFIDENTIALITE DES DONNES

Dans les SGBD relationnels, la notion de schéma externe correspond au concept de table dérivée

II° Les Vues ♥

Une **vue** est une table dérivée dynamique. Elle n'est pas physique. Le problème de cohérence ne pose donc pas. Par contre, elle peut entraîner un ralentissement des traitements si la vue doit êtr générée souvent.

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

Cette commande définit une table virtuelle à partir des tables de la clause FROM de l'*expression_de_sélection*. Les tables présentes dans la clause FROM sont appelées tables sources. Il peut s'agir de tables de base ou de vues.

La commande **CREATE VIEW n'extrait aucune ligne de la base**, elle inscrit la définition de la vue dans les tables de la méta-base (information schema) : TABLES, VIEWS, VIEW_TABLE_USAGE et VIEW_COLUMN_USAGE.

Exemple :

Vue reprenant uniquement les étudiants de 1ère année :

```
CREATE VIEW eleves_de_premiere
   (num_eleve, nom, prenom, date_naissance,
    poids, annee)
AS SELECT *
    FROM eleves
    WHERE annee = 1;
```

Dès qu'une vue est définie, on peut l'utiliser comme s'il s'agissait d'une table de base. (On ne peut pas les différenciés quand on est utilisateur).

Lorsqu'on utilise une **vue dans un requete** la BD va consulté dans la méta–base la **définition** de la **vue**. Le BD transforme la requête initiale : Cette transformation est appelée composition de vues.

Une vue peut être utile pour :

- Masquer la complexité du schéma de la base
- Aider à la formulation d'une requête
- Jouer le rôle de table intermédiaire dans la résolution d'une requête complexe nécessitant plusieurs étapes

Rôles plus profonds :

- **Augmenter l'indépendance logique** en définissant des schémas externes
- **Préserver la confidentialité des données**

Augmenter l'indépendance logique

Les vues permettent d'assurer la confidentialité des données.

Il est possible d'interdire l'accès à certaines lignes (contenu) et/ou à certaines colonnes (contenant) d'une table.

Grâce aux **vues**, il est possible d'émettre des restrictions d'accès en fonction du contexte.

III° Mise à jours au travers des vues

Il n'est pas toujours possible, d'effectuer des mises à jour au travers des vues.

Si la vue est telle qu'il existe une correspondance biunivoque entre les lignes de la vue et celle d'une seule table source, les modifications et les suppressions ne posent pas de problèmes et peuvent être propagées.

SQL2 a défini un ensemble de conditions que doit respecter une vue pour être modifiable : ➢ La définition ne peut contenir les mots réservés : JOIN, UNION, INTERSECT ou EXCEPT (une seule table source)

➢ La clause SELECT ne peut contenir DISTINCT

➢ La clause SELECT ne peut contenir que des références aux colonnes de la table source (pas de SUM, COUNT,…)

➢ La clause FROM contient une seule table ou une vue elle-même modifiable

➢ La clause WHERE ne peut contenir une sous-question dont la clause FROM possède une référence à la table de la requête principale

➢ La vue ne peut contenir de GROUP BY ou HAVING

CHAPITRE 8 Déclencheurs et contraintres :

Ce chapitre s'inscrit dans le cadre général de l'**intégrité des données**. Il a pour objectif d'en décrire la partie **contrôle sémantique**.

Le contrôle sémantique des donnés : **assure la cohérence** des informations stockées par rapport à leur signification dans la réalité.

Exemples :

- La cote d'un élève doit être comprise entre 0 et 20
- La valeur d'un stock ne peut être négative

Contraintes d'intégrité que doivent respecter les données

Ref des contraintes page 7

- SQL 2 fait la distinction entre
  - contraintes générales qui peuvent faire intervenir plusieurs colonnes de plusieurs tables
  - contraintes attachées aux tables de base

- Remarque : une contrainte attachée à une table ne peut exister sans la table et est donc effacée en même temps que la table

Not Defferable :

Oracle possède la clause **[NOT] DEFERRABLE**

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

Exemple :

```
CREATE TABLE t (
    c1    integer,
    c2    integer,
    CONSTRAINT c CHECK (c1 < c2) DEFERRABLE);
```

La contrainte N'EST PAS différée !

Pour qu'elle le soit :

```
SET CONSTRAINT c DEFERRED;
```

Comment le mettre par defaut :

Pour que le comportement par défaut d'une contrainte Oracle soit différé :

```
CREATE TABLE t (
        c1      integer,
        c2      integer,
        CONSTRAINT c CHECK (c1 < c2)
                INITIALLY DEFERRED);
```

Pour inverser le comportement d'une contraint INITIALLY DEFERRED, on utilise la commande :

```
SET CONSTRAINT c IMMEDIATE;
```

→ LES DECLENCHEURS :

6.1 Introduction

- Un déclencheur (trigger) permet de définir un ensemble d'actions qui sont déclenchées automatiquement par le SGBD lorsque certains phénomènes se produisent.
- Les actions sont enregistrées dans la base et non plus dans les programmes d'application.

Les déclencheurs permettent de définir des contraintes dynamiques, ils peuvent être utilisés pour :

- Générer automatiquement une valeur de clé primaire,
- Résoudre le problème des mises à jour en cascade
- Enregistrer les accès à une table
- Gérer automatiquement la redondance
- Empêcher la modification par des personnes non autorisées (dans le domaine de la confidentialité)
- Mettre en œuvre des règles de fonctionnement plus complexes

La syntaxe d'un déclencheur :

6.2 Déclencheurs dans Oracle

Syntaxe :

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

FOR EACH ROW : déclencheur du niveau tuple : sera exécuté pour toutes les lignes provoquant l'activation du déclencheur. En son absence, le déclencheur est du niveau table et le bloc PL/SQL n'est exécuté qu'une seule fois.

Les restrictions :

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
  - Lire ou modifier les colonnes d'une clé primaire, unique ou étrangère d'une table contraignante