Université de Lorraine
FST, Campus Aiguillettes
Département Informatique, Licence Informatique

Support de cours de l'UE BDD3 : SGBD

Egalement disponible sur :

http://www.loria.fr/~nacer/L3-BDD3-Sommaire.html

Nacer.Boudjlida@loria.fr

Document délibérément INCOMPLET, les compléments importants étant donnés en cours

Contenu du document

SGBD : Introduction

Nacer Boudjlida
http://www.loria.fr/~nacer

Université de Lorraine
FST, MIAE

Chapitre I : Introduction aux SGBD Relationnels

1. Rôle/Fonctions d'un SGBD
2. Typologie des SGBD
3. Situation dans l'architecture des applications
4. Langages Relationnels
5. Architecture d'un SGBD
6. "'Métiers du domaine"' et compétences
7. Contenu de l'UE

1- Rôle/Fonctions d'un SGBD

1. Décrire des "objets", leurs relations, des contraintes, etc.
2. Manipuler les données
3. Confidentialité
4. Intégrité
5. Accès concurrents et sécurité de fonctionnement

2- Typologie des SGBD

- Modèle de représentation et de manipulation de données $\longrightarrow$ Classe de SGBD
  1. Hiérarchique, Réseau $\longrightarrow$ CODASYL
  2. Relationnel $\longrightarrow$ SGBD Relationnel
  3. A objets $\longrightarrow$ SGBDOO
  4. [Relationnel "étendu" $\longrightarrow$ "Objet-Relationnel]
  5. Logique $\longrightarrow$ SGBD Déductif
  6. Données non ou semi-structurées $\longrightarrow$ SGBD pour XML
  7. NoSQL: Not Only SQL (Grandes masses de données)

3- Situation dans l'architecture des applications

4- Langages Relationnels

Type	**Fondement**
Algébrique	Théorie des ensembles
Prédicatifs a) à variable n-uplet b) à variable domaine	Logique du 1er ordre

- SQL $\simeq$ "dialecte" fondé sur l'algèbre et le calcul prédicatif à variable n-uplet

4.1- Algèbre relationnelle

- Caractéristiques :
  - Opérande(s) : Relation(s)
  - Résultat : Relation
  - Opérateur : Opérateur du calcul relationnel
- Ensemble minimal d'opérateurs :
  1. Restriction/sélection (σ)
  2. Projection (Π)
  3. Produit cartésien (X)
  4. Union ($\cup$)
  5. Différence ($\setminus$)
- Remarque : Jointure ($\bowtie = \sigma(X(\ldots))$)

4.2- Syntaxe d'un langage algébrique

- Si R est un nom de relation alors R est une expression algébrique (ea)
-
-
-

4.3- De l'algèbre à SQL

- R $\longrightarrow$
- $\sigma_C(R)$ $\longrightarrow$
- $\Pi_L(R)$ $\longrightarrow$
- $R1\ X\ R2)$ $\longrightarrow$
- $R1 \setminus R2$ $\longrightarrow$
- $R1 \cup R2$ $\longrightarrow$
- En réalité, dans un SGBD : de SQL vers l'algèbre!

5- Architecture d'un SGBD

6. Les "'métiers du domaine"' et compétences

1. Application
   - Développeur : Algèbre, SQL, SQL/xxx, Outils de connectivité (Php, jdbc, ...)
   - Concepteur : Compétences du développeur + Méthodes de conception (BdD, Logiciels)
2. Système
   - Administrateur :
     - Données : droits, intégrité, normes, ...
     - Serveurs de données : sécurité, performances, ...
   - Concepteur/Développeur de SGBD

"Métiers du domaine" et Compétences

3. Environnements de dévelopement : Conception/dévelopement d'outils
   - de conception
   - d'aide au dévelopement
   - d'exploitation de données (exemple : Fouille de données, Data mining)

Contenu de l'UE

1. Chapitre 1 : Introduction
2. Chapitre 2 : Organisation et stockage des données
3. Chapitre 3 : Accès concurrents et sécurité de fonctionnement
4. Chapitre 4 : Contraintes d'intégrité et Confidentialité
5. Chapitre 5 : Traitement des requêtes dans les SGBDR

Chapitre III : Organisation et stockage des données

Contenu du chapitre :

I- Organisation et Stockage des données

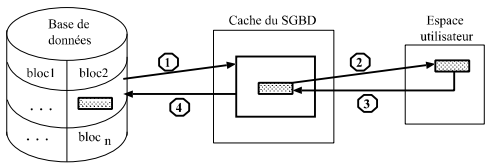
- Comprendre :
  1. Stockage et accès sans index
  2. Index : Organisation, structuration, accès
- Pour savoir :
  1. Gérer les espaces ;
  2. Forme à donner aux requêtes pour maximiser l'utilisation des index ?
  3. Utiliser les outils du serveur pour examiner les choix de son optimiseur.

Organisation et Stockage des données : Sommaire

1. Organisation et stockage : pages et *extents*

- Hypothèse : données organisées et stockées "en vrac" (*en tas* ou *heap*)
- Espace (données, index, log) : blocs (*pages*) de même taille
- *Page* : Unité de lecture/écriture *physique*
- Lecture/écriture *via* des tampons (cache) du système
- *Lecture/Ecriture logique* : Lecture/écriture dans les pages des tampons
- *Défaut de page* : Absence d'une page dans le cache
- *Extent* :
  - Nombre, fixe ou variable, de pages contiguës
  - Attribué généralement à un seul objet

Organisation et stockage en pages



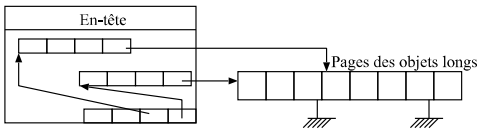
Structure générale des pages

1. *En-tête* :
   - Identification de l'objet contenu dans la page
   - Chaînage page suivante, page précédente de l'objet, etc. ;
2. *Corps* : Valeurs des tuples ;
3. *Table des déplacements* :
   - Localisation des tuples dans la page
   - Sybase : en fin de page
   - Oracle : En début de page

2. Représentation des tuples dans les pages

- En-tête :
  - identification du tuple,
  - nombre de colonnes à valeurs inconnues (*NULL*)
  - nombre de colonnes de longueur variable, etc.)
- Valeur effective du tuple :
  - Colonnes de type “ordinaire”
  - ou pointeurs vers les valeurs des objets longs

3. Stockage des objets de grande taille

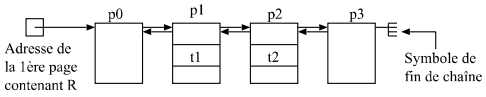


4. Organisation “en tas” (*heap*)

- Comportement :
  1. Recherche,
  2. Insertion,
  3. Suppression,
  4. Modification de tuples.

4.1- Organisation “en tas” et recherche de tuples

- Recherche de tuples de R
- Adresse de la première page contenant R : dans une des tables de la méta-base
- Parcours des pages de R



4.2- Organisation “en tas” et insertion

- Dans la dernière page de la relation, si espace disponible
- Sinon, dans une page vide de l’*extent* courant
- Si *extent* saturé :
  - Allocation d’un nouveau
  - Insertion dans sa première page
- Note : Adresse de la dernière page

4.3- Organisation “en tas” et suppression

- Localisation du tuple
- Effacement
- Décalage des tuples (éviter “l’effet gruyère”)
- Mise à jour des déplacements des tuples
- Cas page vide après suppression ?
- Cas extent vide après suppression ?

4.4- Organisation “en tas” et mise à jour

- Parcours des pages ;
- *Cas taille du tuple modifié* :
  1. = taille(ancien) : Modification du tuple ;
  2. < taille(ancien) :
     (a) Modification du tuple et “décalage” vers le haut ;
     (b) Modification des pointeurs de tuples.
  3. > taille(ancien) et place disponible dans la page : cf. cas 2, mais avec “décalage” vers le bas ;
  4. Sinon (*Migration de tuple*)
     (a) Suppression du tuple de la page ;
     (b) Insertion dans la dernière page du tas.

Organisation “en tas” et mise à jour

- Gérer au mieux les migrations de tuples
- Contrôle du remplissage *(densité)* des pages
  - Sybase : *max_rows_per_page*
  - Oracle : *PCTUSED, PCTFREE* (voir plus loin)

4. Organisation “en tas” : Conclusion

- Quand utiliser des “tas” ?
  - Petites relations utilisant peu de pages ;
  - Pas d’accès direct à un tuple ;
  - Pas d’ordre sur les ensembles résultats ;
  - Relation ayant des tuples non uniques + ce qui précède ;
  - Relations avec peu de modifications et d’insertions.
- Maintenance périodique

5. Gestion de la mémoire cache

- $\simeq$ mémoire paginée dans les systèmes d’exploitation
- Recherche d’une donnée
  1. Dans une page du cache ?
  2. Si absente (*défaut de page*) : page la contenant $\longrightarrow$ cache
  3. Si toutes les pages du cache sont occupées: en “vider” une
- Stratégies de choix de la page du cache à remplacer :
  1. Dernière page utilisée (*Most Recently Used*, MRU)
  2. La moins récemment utilisée (*Less Recently Used*, LRU)

Stratégies de “caching”

- Cache géré comme une chaîne de pages Most Recently Used/Less RU
- “Âge” d’une page : MRU $\longrightarrow$ LRU
- Quand des pages modifiées atteignent un point dans la chaîne (*wash marker*) : Ecriture asynchrone de la page [ou *checkpoint*] par le serveur (i.e. quand une page arrive en fin de chaîne, elle est “propre” et peut alors être ré-utilisée)
- *Stratégies de remplacement de pages* :
  1. LRU
  2. MRU (*fetch and discard*)

1- Stratégie LRU de remplacement de pages

- Lecture séquentielle de pages en tête de la chaîne MRU
- “Pousser” éventuellement des pages vers la LRU
- Quand une page modifiée “passe” le *wash marker* : écriture
- Si nouveau besoin/accès à cette page : la remettre en tête de MRU.
- $\Longrightarrow$ Recherche d’une page p_i et
  - p_i non modifiée et $p_i \in$ cache : p_i
  - p_i modifiée et $p_i \in$ cache : p_i en tête MRU
  - $p_i \neg \in$ cache : lecture disque (1 buffer) en tête de MRU.

2- Stratégie MRU de remplacement de pages

- Pages mises juste avant le *wash marker*
- Si $p_i \in$ cache : mettre p_i avant le marqueur
- Sinon :
  1. Lire p_i
  2. Mettre p_i avant le marqueur

II- Organisation et stockage des index

- Localisation des tuples
  - Parcours séquentiel de l'instance d'une relation
  - "Directement" via des index.
- Index : structure de données associant
  - valeur d'un attribut ou une liste d'attributs (*clé de l'index*)
  - et "adresses" des tuples contenant cette valeur

Typologie des index

1/4) *Index dense*

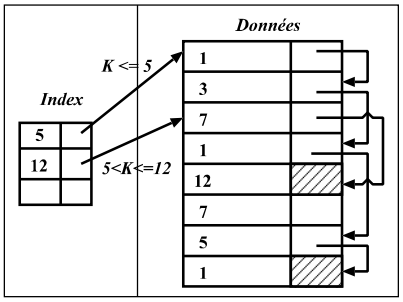
- Une entrée dans l'index par valeur de la clé
- Entrée : $< V_i,\ P_i >$ où
  - V_i : valeur d'une clé d'index C
  - P_i : tête d'une liste d'adresses de tuples
- Si Clé d'index = clé de la relation : pas de chaînage
- Rangement des tuples pas nécessairement contigu
- Exemple :

Typologie des index

2/4) *Index creux*

- Moins d'entrées que de valeurs de la clé
- Entrée : $< V_i,\ P_i >$ où
  - V_i : valeur d'une clé d'index C
  - P_i : tête d'une liste d'adresses de tuples t t.q. $\Pi_C(t) \leq V_i$

Typologie des index : Index Creux



Typologie des index

3/4) *Index primaire*

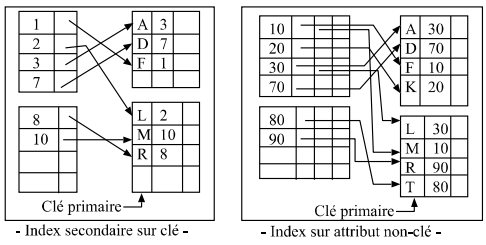
- Défini sur une clé primaire de relation
- Valeurs ordonnées dans la relation
- Ordre logique = ordre physique.
- Exemple :

Typologie des index

4/4) *Index secondaire*

- Construit sur un attribut dont les valeurs devraient être ordonnées.
- Mais relation déjà ordonnée sur sa clé primaire
- $\Longrightarrow$ ordre logique $\neq$ ordre physique

Typologie des index : Index secondaire



- Index secondaire sur clé -

- Index sur attribut non-clé -

Représentation des index

- Arbres équilibrés (*Balanced trees*, B-arbres)
- Feuilles toujours à égale distance de la racine
  1. Arbres B^+
  2. B-arbres

Représentation des index : (1) Arbres B^+

- Chaque nœud a entre n et $n/2$ nœuds fils
- n fixé pour un arbre donné
- *Nœud feuille* :
  - au plus $(n-1)$ valeurs et n pointeurs
  - Pas moins de $\lceil (n-1)/2 \rceil$ valeurs
  - Dernier pointeur : $\longrightarrow$ feuille suivante
- *Nœud racine et Nœuds internes* :
  - Entre $\lceil n/2 \rceil$ et n pointeurs
  - Premier Pointeur P_1 : clés $K_i,\ K_i < K_1$
  - Dernier pointeur $P_p,\ p \leq n$: clés $K_j, K_i - 1 \leq K_j < K_i$

Représentation des index : (1) Arbres B^+

- Recherche
  - $\simeq$ Recherche par dichotomie
  - longueur du chemin de la racine jusqu'aux feuilles $\leq \log_{\lceil n/2 \rceil} \mathcal{N}$
  - $\mathcal{N}$ étant le nombre de valeurs de la clé
  - Exemple : $n = 3$

Représentation des index : (1) Arbres B^+ et mise à jour

- Efficacité en recherche
- Activité supplémentaire en mise à jour :
  - Maintenir la propriété "B"
  - Garantir la structure des nœuds
- Insertion : cf. exemple de la page 63
  - Ajout d'un tuple de clé C.
- Suppression de la valeur P (sur l'arbre de la page 63)

Représentation des index : (2) B-arbres

- Similaires aux arbres B^+
- Différence : toutes les clés de l'index n'apparaissent pas aux feuilles
- Nombre de nœuds du B-arbre $<$ ceux du B^+
- Ajout de pointeurs dans les nœuds non-feuilles pour les valeurs de la clé de l'index qui n'apparaissent pas aux feuilles
- Exemple Correspondant à l'exemple d'arbre B^+, page 64

Organisation et stockage : III. Application à Oracle

- Espace de tables (*tablespace*) : ensemble d'unités logiques de stockage des objets d'une base
- Base : occupe une ou plusieurs *tablespaces*
- *Tablespace* : contenu rangé dans un plusieurs "fichiers de données" (*Datafile*)
- Un fichier physique : associé à une seule *tablespace* et à une seule base
- **Note :** *Datafile* n'accueille pas exclusivement des données

Application au SGBD Oracle

- Outre les *datafiles*, une base est dotée :
  - d'un fichier de paramètres de configuration (*control file*)
  - d'un fichier de paramètres d'initialisation (*init.ora*)
  - de fichiers de journalisation des transactions (*redo log* files, *rollback segments/undo tablespaces*)
- [redacted]
- Espaces physiques persistants (disque) structurés en :
  - Pages (appelés *blocs*),
  - *extents* et
  - *segments*

Application au SGBD Oracle

- Bloc = l'unité par défaut de lecture/écriture physique
- *Extent* : nombre déterminé de blocs contigus
- Segment : ensemble d'*extents* non nécessairement contigus

[redacted]

1. Introduction aux *tablespaces*

- *Tablespace SYSTEM*
  - automatiquement créée lors de la création de chaque base
  - contient, dans son premier *datafile* :
    - dictionnaire de la base
    - unités de programmes stockés (procédures, fonctions, paquetages et triggers)
- Recommandé : créer des *tablespaces* pour
  - contrôler le placement des objets
  - affecter des ressources aux utilisateurs

Introduction aux *tablespaces*

- Exemples de *tablespaces* "dédiés" :
  - Images avant modification (*undo tablespaces*)
  - Pour stockage exclusif des données ou des index
  - Espaces temporaires (pour les tris, par exemple)
- Durée de vie d'une *tablespace* : création (*create tablespace*) $\longrightarrow$ suppression
  - explicite (*drop tablespace*)
  - implicite : *tablespace temporaire* ($\longrightarrow$ fin de session de création)

Statuts et états d'une *tablespace*

1. Temporaire/permanente : changement par :
   alter tablespace . . . temporary/permanent
2. Accessible (*online"*)/Inacessible (*offline"*) :
   "alter tablespace Nom *online/offline"*
   - Pas de mise hors ligne de *SYSTEM*
   - Pas de mise hors ligne de *tablespaces* contenant des segments d'annulation (*rollback segments*) en cours d'utilisation
   - Pendant la mise hors ligne : données des transactions non achevées journalisées dans *segments d'annulation différée* (cf. chapitre Sécurité et reprise).

Etats et statuts d'une *tablespace*

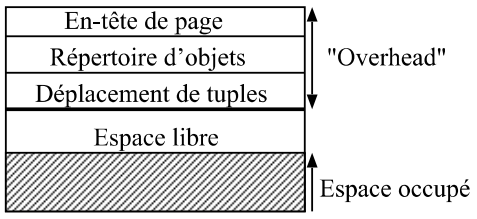
3. Lecture seule (*read only*) :
   - Par défaut, *tablespace* en lecture/écriture
   - Mise en lecture seule provisoire ou définitive
     - *alter tablespace . . . read only*
     - Changement d'état effectif à l'issue des transactions actives
     - Seules les opérations de consultation et de suppression d'objets (index, tables) restent autorisées.
   - Exemples :
   - *dba_tablespaces* : informations sur toutes les *tablespaces*
   - *user_tablespaces* : sur celles accessibles par un utilisateur.

Application au SGBD Oracle

1. Notions sur les *tablespaces*
   - Compléments dans le chapitre Bases de données et leurs objets
2. Gestion des blocs
3. Gestion des *extents*
4. Gestion des segments
5. Gestion des espaces physiques non persistants (cache)

2. Blocs Oracle

- Taille multiple de la taille des blocs du système hôte



Blocs Oracle : *Descripteur* (*"overhead"*)

- 84 à 120 octets

1. *En-tête* : adresse, type du segment d'appartenance, etc.
2. *Répertoire de tables* : informations sur les objets contenus dans le bloc
3. *Table des déplacements* des tuples :
   - 2 octets par entrée
   - Entrées non libérées lors de la suppression de tuples
   - Ré-utilisables lors d'insertions dans le bloc

Gestion et contrôle du remplissage d'un bloc

- Paramètres de stockage : Contrôler le remplissage de blocs de tables, groupements de tables (*cluster*) et index :
  1. *PCTFREE* : pourcentage minimum d'espace libre dans un bloc
  2. *PCTUSED* : pourcentage maximum pouvant être occupé
- Spécifiés lors de la création ou de la modification d'une table, d'un *cluster* ou d'un index (*create/alter table, cluster* ou *index*)
- Si taux de remplissage d'un bloc >= *PCTFREE* :
  - Insertions interdites dans le bloc ;
  - Suppressions et modifications autorisées ;
  - Reprise des insertions si taux de remplissage < à *PCTUSED*.

PCTFREE, *PCTUSED* : Exemple

- *PCTFREE* = 30% et *PCTUSED* = 50% :
  1. Insertions autorisées dans le bloc jusqu'à ce qu'il n'y reste que 30% d'espace libre ;
  2. Seules les mises à jour sont alors autorisées dans le bloc ;
  3. Insertions de nouveau permises lorsque le taux d'occupation du bloc descend au-dessous de 50%.

3. *Extents* Oracle

- Groupe de blocs contigus,
- Unité d'allocation d'espace pour un objet dans un segment,
- *Segment* :
  - Un ou plusieurs *extents* ;
  - Bloc d'en-tête d'un segment : répertoire des *extents*.

[redacted]

4. Gestion des Segments Oracle

- Blocs des *extents* : contigus,
- *extents* d'un segment : pas nécessairement contigus
- Segment : alloué
  - à un objet de la base,
  - dans une *tablespace*
  - éventuellement dans des fichiers physiques différents.
- Quatre types importants de segments :
  1. Données [redacted]
  2. Index [redacted]
  3. Temporaires [redacted]
  4. Annulation (cf. chapitre sécurité et reprise)

Oracle : Segments temporaires

- = Espaces disque
- Utilisés lors de l'analyse et de l'exécution
  - de *create index*,
  - de *select distinct* avec/sans *order by* et/ou *group by*
  - de requêtes avec $\cup$, $\cap$, $\setminus$
  - de jointures sans index.
- Utilisables pour des index construits pour des tables temporaires

[redacted]

- Segments dans *tablespaces* temporaires : *create temporary tablespace*
- Affectation utilisateur-espace temporaire :
  '*create* ou *alter user* . . . *temporary tablespace* NomEspace"

5. Gestion des caches Oracle

- Structures mémoire d'une instance Oracle incluent :
  1. Zone globale système (SGA, cf. chapitre installation)
  2. Zones globales de programmes (*Program Global Area, PGA*) ou de processus (serveur et arrière-plan)
  3. Zones mémoire partageables pour code exécutable :
     (a) code de l'instance Oracle
     (b) code utilisateur
  4. Zones de tri, etc.

5. Mémoires cache Oracle

- Plan :
  1. SGA
     (a) Taille
     (b) Cache de données
     (c) Cache du *log*

[redacted]

5.1.1 Mémoires cache : Taille de la SGA

- Déterminée lors du lancement d'une instance Oracle
- Affichée lors du lancement de *Oracle Enterprise Manager*
- Peut être obtenue par *SHOW SGA* (sous *SQL*Plus*) :

```
SQL> show sga;

Total System Global Area   73701404 bytes
Fixed Size                    75804 bytes
Variable Size              56770560 bytes
Database Buffers           16777216 bytes
Redo Buffers                  77824 bytes
```


5.1.2 Mémoires cache : cache de données

- Taille fonction de :
  - *db_block_size* (généralement 2 ou 4KO) et *db_block_buffers*
- Cache comportant :
  1. *write list* : liste des blocs modifiés en attente d'écriture physique ;
  2. liste *LRU* (*"Least Recently Used"*) :
     - blocs libres,
     - + blocs modifiés non encore transférés vers la *write list*
     - + blocs en cours d'utilisation (*"pinned blocks"*).

5.1.2 Cache de données : Remplacement de blocs

- Par défaut, parcours d'une table "en *fetch and discard"* (MRU)
- Contrôle de la stratégie :
  1. clause *"CACHE| NOCACHE"* de *create* ou *alter table* ou *index*
  2. Utilisation de pools de blocs :
     (a) Configurer les pools
     (b) Associer/affecter objets aux pools

5.1.2.a) Configuration de pools de blocs de données

- Pool = 1, n ensembles de buffers
- 3 types de pools : *KEEP*, *REUSE* et *DEFAULT*

1. *KEEP* : Taille spécifiée par *buffer_pool_keep* (paramètre de configuration)
2. *REUSE* : Taille spécifiée par *buffer_pool_reuse*
3. *DEFAULT* :
   - Emplacement, par défaut, des objets
   - Taille = *db_block_buffers* - (*buffer_pool_keep* + *buffer_pool_reuse*)

5.1.2.b) Association pools-objets

- Objets : tables, *clusters*, index, partitions
- Type de pool → Stratégie de remplacement de blocs
- Association pool d'objets-stratégie :
  - Dans {*alter* | *create*} {*table* | *index* | *cluster*}
  - Options *KEEP* et *RECYCLE* de la *clause de stockage*.
- *KEEP* : maintien des blocs en mémoire après utilisation ;
- *RECYCLE* : pas de maintien.

5.1.2 Configuration de caches de données : Exemple

create table R(x *int*, ...) *storage (buffer_pool recycle)*;
create index Idx2R *on* R(x) *storage (buffer_pool keep)*;

- Gestion des blocs de *R* : *"fetch and discard"* ;
- Gestion des blocs de *Idx2R* : *LRU* ;
- Si dans le fichier de configuration de la base :
  - *db_block_buffers = 2048*
  - *buffer_pool_keep = (buffers: 300, ...)*
  - *buffer_pool_reuse = 100*,
- Pool *KEEP* : 300 blocs ;
- Pool *REUSE* : 100 ;
- Pool par défaut : (2048 - (300 + 100)).

5.1.3 Mémoires cache du journal (*redo log*)

- Enregistrements du journal : *redo entries*
- Taille du cache du *log* :
  - Par défaut : 4*taille maximum d'une page du système hôte ;
  - ou *log_buffer* octets (paramètre de configuration de la base)
- *LGWR* : écritures physiques dans le fichier ou le groupe de fichiers *redo log actifs* (cf. chapitre sécurité et reprise) ;
- Gestion circulaire des blocs
- Exemple : Cache de 4 blocs

Organisation et stockage des données et des index : Conclusion

- Généralement transparente pour un utilisateur naïf (*indépendance données-programmes*) ;
- Compréhension nécessaire pour
  - Gérér les espaces des bases ;
  - Comprendre les choix de l'optimiseur ;
  - Configurer le serveur ;
  - ...
  - Concevoir et implémenter des SGBD.

Systèmes de Gestion des Bases de Données

Accès concurrents et Sécurité de fonctionnement

Nacer Boudjlida

Introduction à la notion de transaction

- Partage des ressources dans les systèmes d'exploitation
- Base de données accédée par plusieurs utilisateurs (ou programmes)
- *Sous-système de gestion des accès concurrents*
- *Techniques* : Verrouillage, exclusion mutuelle
- *Ressources partagées* : Données et méta-données
- *Transaction* $\simeq$ Programme (accès, modification)
- *Accès concurrents* $\longrightarrow$ Problèmes

Transactions : problème 1 : (Non) Atomicité

- *Virement de compte à compte* : {C1 = C1-m; C2 = C2 + m}
  1)
  2)
  3)
  4)
  5)
  6)
- Si arrêt du programme avant *Ecrire(y)* : Incohérence
- *Atomicité* : *"Tout ou rien"*
- *"Tout"* $\Longrightarrow$ *Durabilité* (Permanence des modifications)
- *"Rien"* $\Longrightarrow$ Annulation des modifications

Transactions : problème 2 : (Non) Isolation

- *Transaction T1* : Débiter X de N
- *Transaction T2* : Créditer X de M

T1	*T2*
$x \longleftarrow Lire(X)$	$y \longleftarrow Lire(X)$
$x \longleftarrow x - N$	$y \longleftarrow y + M$
$X \longleftarrow Ecrire(x)$	$X \longleftarrow Ecrire(y)$

Problème 2 : (Non) Isolation

- X = 100; N = 10; M = 50

Exécution avec entrelacement	X_{base}	x_{T1}	y_{T2}
T1 :	100		
T1 :			
T2 :			
T2 :			
T1 :			
T2 :			

- Perte de mise à jour : X = X-N
- $\Longrightarrow$ *Isolation* : pas d'interférence pendant l'exécution

Problème 3 : (Non) Cohérence

- *Contrainte* : A = B

T1 : $\{A = A + 1; B = B + 1\}$	*T2* : $\{A = A \times 2; B = B \times 2\}$
t11 : $A_{T1} \longleftarrow Lire(A)$	t21 : $A_{T2} \longleftarrow Lire(A)$
t12 : $A_{T1} \longleftarrow A_{T1} + 1$	t22 : $A_{T2} \longleftarrow A_{T2} \times 2$
t13 : $A \longleftarrow Ecrire(A_{T1})$	t23 : $A \longleftarrow Ecrire(A_{T2})$
t14 : $B_{T1} \longleftarrow Lire(B)$	t24 : $B_{T2} \longleftarrow Lire(B)$
t15 : $B_{T1} \longleftarrow B_{T1} + 1$	t25 : $B_{T2} \longleftarrow B_{T2} \times 2$
t16 : $B \longleftarrow Ecrire(B_{T1})$	t26 : $B \longleftarrow Ecrire(B_{T2})$

- <t11; t12; t13; t21; t22; t23; t14; t15; t16; t24; t25; t26> : Correct
- <t21; t22; t11; t12; t23; t13; t14; t15; t16; t24; t25; t26> : Incorrect

Problème 4 : (Non) Permanence des modifications

a) *(Non) Répétabilité des lectures*

Transaction T1	Transaction T2
$a \longleftarrow Lire(A)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)
$a \longleftarrow a + 100$	
$A \longleftarrow Ecrire(a)$	
	$b \longleftarrow Lire(A)$
	Imprimer(b)

- Impressions de T2 : valeurs différentes

Problème 4 : (Non) Permanence des modifications

b) *Tuples "fantômes"*

Transaction T1	Transaction T2
	1. $V \longleftarrow Lire(A = \{a < 4000\})$
	2. *Imprimer(V)*
3. Supprimer tout a tel que $a < 4000$	
	4. $V \longleftarrow Lire(A = \{a < 4000\})$
	5. *Imprimer(V)*

- Deuxième impression de V vide

Transactions et niveaux d'isolation : SQL 92

- *Niveau 0* : lectures "sales" permises
  - D en cours de modification par T
  - Transactions autres que T, avant validation par T :
    * bloquées en modification
    * autorisées à lire D
- *Niveau 1* : lectures "sales" interdites
- *Niveau 2* : non répétabilité des lectures avant validation
- *Niveau 3* (Par défaut) : empêcher les tuples fantômes

Transactions et reprise

- Nécessité de contrôler les accès concurrents
- *Causes d'échec des transactions* :
  - Erreur logiciel ou matériel
  - Avortement par le sous-système de gestion de la concurrence
  - "Catastrophe naturelle", etc.
- Nécessité de rétablir la cohérence : *Reprise*
  - Revenir à l'état cohérent le plus proche de l'instant de l'incident
  - Mécanisme : *Journal ou Log* ("Histoire des transactions")

Schéma d'exécution des transactions

- Transactions de mise à jour : séquences
  1. Lecture (Base $\longrightarrow$ Mémoire centrale)
  2. Traitement (en mémoire centrale)
  3. Écriture (Mémoire centrale $\longrightarrow$ Base)
- *Politique "du tout ou rien"* (**A**tomicité) :

  $Action_1$
  ...
  $Action_n$
  Si toutes les actions se sont bien passées
  Alors Valider la transaction (*COMMIT*)
  Sinon Annuler ses effets (*ROLLBACK*)
  Finsi

Schéma d'exécution des transactions

- *Validation* : rendre effectives les modifications
- *Annulation* : défaire les actions de la transaction
- *Diagramme d'états d'une transaction*

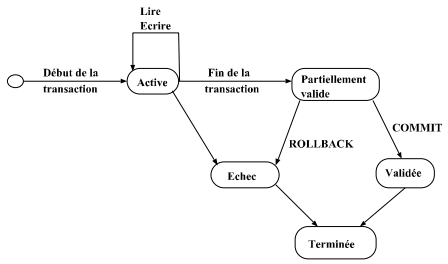


Schéma d'exécution des transactions

- *Validité partielle* :
  - Des techniques testent l'absence d'interférences
  - Des protocoles de reprise s'assurent qu'ils peuvent effectivement enregistrer les modifications
- Parfois : Annulation/Ré-exécution (UNDO/REDO) d'une opération

Journalisation, points de validation et de contrôle

- *Journal* :
  - Trace des opérations effectuées sur la base
  - Support non volatile + Archivage périodique
- *Types d'informations du journal*, pour chaque transaction :
  - <début_transaction, identification de T> ;
  - <écriture, identification de T, donnée concernée, ancienne valeur, nouvelle valeur> ;
  - <lecture, identification de T, donnée concernée>[a] ;
  - <COMMIT, identification de T>.

[a] Certains systèmes ne conservent pas de trace des opérations de lecture.

Journalisation, points de validation et de contrôle

- *Point de validation* :
  - Instant de journalisation de <COMMIT, identification de T>
  - Les actions de T ont été réussi
  - + Écritures dans le journal réussies
  - *Au-delà de ce point* : T est valide et ses MaJ sont permanentes
  - *En cas d'incident* : le système sait quoi
    - annuler ("histoire en arrière")
    - relancer, éventuellement ("histoire en avant")

Journalisation, points de validation et de contrôle

- *Points de contrôle*
  - Inscrits périodiquement dans le *Log*
  - *Période* : intervalle de temps, nombre donné de mise à jour ou combinaison des deux
  - *Prise de point de contrôle* :
    1.
    2.
    3.
    4.

  $\Longrightarrow$ les actions des transactions dont le point de validation dans le journal précède un point de contrôle pourront ne pas être ré-exécutées en cas d'incident

Points de validation et Points de contrôle

Points de validation et Points de contrôle

Transactions : Sérialisabilité et plan d'exécution

- Gestion de la concurrence fondée sur la *sérialisabilité* :

  L'exécution concurrente de n transactions doit être équivalente à (avoir le même effet pour chaque transaction que) leur exécution séquentielle

- *Propriétés (ACID)* à assurer par un gérant de transactions :
  1. *Atomicité* : politique du "tout ou rien" ;
  2. *Cohérence* : satisfaction des contraintes d'intégrité ;
  3. *Isolation* : pas d'interférence entre les transactions ;
  4. *Durabilité* : permanence des modifications effectuées.

Sérialisabilité et plan d'exécution

- A et I : par la méthode de reprise
- C : par le programmeur et le sous-système d'intégrité
- D : par les mécanismes de reprise et de contrôle de la concurrence
- *Sérialisabilité* : par le contrôle de la concurrence
- *Plan d'exécution* $\mathcal{P}$ (ou *schedule*) de n transactions :

  Ordre sur les opérations des transactions tel que pour toute transaction T_k de $\mathcal{P}$, si une opération O_i précède une opération O_j dans T_k alors O_i précède aussi O_j dans $\mathcal{P}$

- Possibilité d'entrelacement des transactions $\Longrightarrow$ $\mathcal{P}$ non unique
  $\Longrightarrow$ n'autoriser que les plans d'exécution corrects

Sérialisabilité et plan d'exécution

- Exécution séquentielle = *Succession* = Plan correct

  Un *plan d'exécution* $\mathcal{P}$ des transactions $T_1, \cdots, T_n$ est une *succession* (ou *serial schedule*) s'il existe une permutation Π de (1, ..., n) telle que P $= < T_{\Pi(1)}; \cdots; T_{\Pi(n)} >$

- *Exécution sérialisable* : Plan correct

  Une *exécution* de $T_1, \cdots, T_n$ est *sérialisable* si et seulement si elle donne, pour chaque T_i, le même résultat qu'une succession

- *Problème* : n'autoriser que les exécutions sérialisables
  - Protocoles pessimistes : verrouillage, estampillage
  - Protocoles optimistes : contrôle à la fin d'une transaction

Contrôle de la concurrence : Verrouillage

- *Verrou* : Variable associée à un *granule* et dont la valeur indique le type d'opération possible sur un granule
- *Granule* : unité de verrouillage
  - Base, Relation, Partie de relation, etc.
- *Petite taille* :
  (+) $\nearrow$ degré concurrence
  (-) $\nearrow$ temps de son contrôle
- *Grande taille* :
  (-) $\nearrow$ temps d'attente

Contrôle de la concurrence : Types de verrous

1. *Verrou binaire* :
   - Deux états : Libre | Occupé
   - (-) Une seule transaction détient un granule
2. *Verrou partagé (Share) et verrou exclusif (Exclusive)* :
   - Accès multiples en lecture
   - En mise à jour : Un accès [+ des attentes]
   - Compatibilité des verrous :

	Partagé	*Exclusif*
Partagé		
Exclusif		

Contrôle de la concurrence : Types de verrous

3. *Verrou d'intention* :
   - Transaction demande un verrou de mise à jour $\simeq$ verrou de lecture avec *intention* d'écriture
   - Compatibilité :

	Partagé	*Exclusif*	*Mise à jour*
Partagé	Oui	Non	Non
Exclusif	Non	Non	Non
Mise à jour	Oui	Non	Non

Gestion des transactions : Verrouillage à deux phases

- *Two Phase Locking Protocol* : le plus souvent implanté
  1. *Phase 1* (expansion) : acquisition
  2. *Phase 2* (rétrécissement) : libération et plus aucune autre acquisition
- Garantie de la sérialisabilité
- Mais verrouillage $\Longrightarrow$ Risques :
  1. Interblocage *(Deadlock)*
  2. Famine *(Livelock)*
- **Note :** *Two Phase Locking strict*

Verrouillage et Interblocage

- Attentes mutuelles
- *Prévention* : Imposer à toute transaction de verrouiller en avance tous les éléments dont elle a besoin et n'apposer aucun verrou si un de ces éléments n'est pas libre
- *Détection* : Cycle dans un graphe d'attente
  - *Nœuds* : Transactions
  - *Arc* $T1 \longrightarrow T2$: T1 *attend* un granule détenu par T2
  - Présence de cycle :
    1. *Tuer* une transaction
    2. Défaire ses actions
    3. Libérer ses ressources

Verrouillage, Interblocage et Famine

- *Exemple d'interblocage* :
- *Famine* : Attente infinie
  - *Exemple* : Priorité toujours aux mêmes transactions
  - *Solutions* : Priorités dynamiques ou *FIFO*

Contrôle de la concurrence : Estampillage

- *Estampille* : Identifiant de transaction (date de début)
- Technique fondée sur un ordre sur les *estampilles* : sérialisabilité sans interblocage
  - Plan d'exécution sérialisable s'il se conforme à l'ordre
  - *Succession* équivalente = transactions ordonnées sur les estampilles
- Granule G "marqué" par la date t de la dernière transaction qui y a accédé
- Actions sur G estampillées t', $t' \geq t$: autorisées
- Actions estampillées t'', $t'' < t$: violation de la sérialisabilité $\Longrightarrow$ Rollback

Contrôle de la concurrence : Techniques "optimistes"

- Pas de contrôle pendant l'exécution des transactions
- Mises à jour locales à la transaction
- Fin de transactions : test de non violation de la sérialisabilité
- Adaptées aux transactions ayant peu d'interférence

Reprise en cas d'incident (*Recovery*)

- Reconstruire un état cohérent à partir "du passé" (*Log*)
- État le plus proche possible de l'instant de l'incident
- Stratégie de reprise dépend de la gravité de l'incident

1. Reprise "à foid" : si dégâts importants
   (a) Charger une sauvegarde de la base
   (b) Re-exécuter les transactions valides des journaux
2. Reprise "à chaud" : Défaire [et refaire] des actions

Reprise "à chaud", Technique 1 : Mise à jour différée

- Mise à jour effective après le point de validation de la transaction
- $\simeq$ Modifications de données *"au brouillon"* (copies)
- *Cas échec* : base inchangée
- *Cas succès* : Substitution Copie/"Original" (*Shadow paging*)
- *Cas incident au moment de la substitution* : Refaire certaines actions de transactions validées

Reprise "à chaud", Technique 1 : Mise à jour différée

Reprise "à chaud" : Technique 2 : Mise à jour immédiate

- Journalisation des modifications avant écriture dans la base avant le point de validation
- *Write Ahead Log Protocol* :
  - Implanté dans la plupart des SGBD
  - Permet la reprise même en cas d'incident entre l'écriture dans le journal et l'écriture dans la base

Write Ahead Log Protocol

WAL et reprises

- *Cas reprise à chaud* : Annuler les transactions non validées (cf. Log)
- *Cas reprise à froid* :
  1. Charger une base cohérente (exemple : B1)
  2. Charger les journaux adéquats (exemple : J2, J3)
  3. Ré-exécuter automatiquement les transactions validées
- *Remarque* : cascade de ROLLBACK
  - Rollback T1
  - T2 a utilisé des valeurs modifiées par T1

Concurrence et Reprise : Conclusion

- Sérialisabilité et ACID"-ité"
- Verrouillage à deux phases (Interblocage)
- *Write Ahead Log Protocol*
- *Shadow Paging*
- Reprise en cas d'incident
- Sécurité par duplication (*Mirroring*)
- Organisation de l'exploitation des bases (*officier de sécurité*)
- Transactions plates : même comportement sur tous les SGBD
- Transactions imbriquées : pas de modèle d'exécution universel

2.2. Transactions, sauvegardes et reprises

- Reprise en cas d'incident (*recovery*) : remettre une base en fonctionnement
  - le plus rapidement possible
  - dans un état le plus proche possible de celui dans lequel elle était avant l'incident.
- Incidents : de l'erreur de programmation à la détérioration des unités physiques de stockage.
- Reprendre : reconstruire une base saine à partir de son *histoire* (cf. *log*).

2.2. Transactions et reprises sous Oracle : Plan

1. Gestion des *redo logs* (p. 180)
2. Gestion des *rollback segments* (p. 186)
3. Validation/Annulation (p. 194)
4. Isolation, verrouillage et points de contrôle (p. 197)
5. ~~Sauvegardes (p. 202)~~
6. ~~Restauration (p. 209)~~
7. ~~Reprise (p. 212)~~

2.2.1- Journaux ou *redo logs*

- Minimum deux fichiers *redo logs* par base
- Gestion circulaire
- Possibilité d'archivage automatique (*archivelog*)
- [+ sauvegardes archives et *redo logs actifs*]

1. Archivage automatique
2. Gestion des *redo logs* archivés

2.2.1.a- Archivage automatique des *redo logs*

- Activation du mécanisme : clause *archivelog* de la commande *create/alter database*
- Si *alter database archivelog* : *log_archive_start = true* (*init.ora*)
  - Edition fichier initialisation
  - ou *alter system*
- Note : modification d'un fichier d'initialisation d'une base ouverte prise en compte seulement lors du prochain lancement de l'instance Oracle associée.

2.2.1.a- Activation/Désactivation archivage

- *log_archive_start = true* dans *init.ora* : mode archivage automatique effectif dès le lancement (*startup*) de l'instance associée.
- Sinon, activation après le startup par *"alter system archive log start"*
- Désactivation :
  1. *log_archive_start = false* dans *init.ora*
  2. ou *"alter system archive log stop"*.

2.2.1.a- Principe de l'archivage

- *LGWR* : processus d'écriture dans le journal
- *ARCn* : processus d'archivage
- Fichier indisponible pendant son archivage.
- Si plusieurs processus d'archivage, plusieurs tampons, etc. : voir les paramètres d'initialisation *log_archive_max_processes*, *log_archive_buffers* et *log_archive_buffer_size*.
- Mécanisme :

2.2.1.a- Principe de l'archivage

- Forcer un archivage :
  1. du log courant et ceux non archivés : *alter system archive log current*
  2. d'un groupe : *alter system archive log group* i
  3. des logs non courants : *alter system archive log all*
  - 1 sur base ouverte
  - 2, 3 sur base montée, ouverte ou fermée.

2.2.1.b- Gestion des redo logs archivés

- *Emplacement physique* :
  - Paramètre *log_archive_dest* ou *alter system archive log ... to*).
- *Nommage* des logs archivés :
  - Paramètre *log_archive_format*
  - Utilisation possible d'un numéro de séquence qu'Oracle associe aux fichiers archives (identique pour tous les membres d'un même groupe)

2.2.2- Gestion des *rollback segments*

- Alternative : *Tablespaces* d'annulation (Version 9)
- Valeurs de données avant leur modification
- Données de transactions non encore validées
- Peuvent servir pour effectuer des lectures "non sales" (*consistent read*), pour :
  - annuler les effets d'une transaction
  - effectuer une reprise sur la base.
- Segment d'annulation = {"entrées d'annulation"} (*rollback entries*)

2.2.2- *Rollback segments* et transactions

- Entrée d'annulation :
  - Identification du fichier de données (*datafile*)
  - Identification du bloc contenant la donnée
  - Valeur de la donnée avant modification
- Chaînage des entrées d'une transaction ;
- En cas d'annulation : Parcours de la chaîne "en arrière"
- Si segment partagé
  - une table T de transactions par segment
  - T[i] = lien vers la liste des entrées de la transaction P_i

Rollback segments et transactions

- *Affectation d'une transactions* à un segment d'annulation :
  - en début de la transaction,
  - lors de la rencontre de la première instruction de mise à jour ou de définition de données,
  - par le système
  - ou par le programmeur (spécification, dans la transaction, de segment à utiliser).
- *Pas d'association* si transaction d'interrogation i.e.
  - Déclarée comme telle (*set transaction read only*)
  - ou ne faisant que de la consultation
- Distribution "équitable" transactions/segments

Types de *Rollback segments*

1. *SYSTEM* :
   - créé lors de la création d'une base dans la *tablespace SYSTEM*
   - avec les paramètres de stockage par défaut de celle-ci
   - Ne peut pas être supprimé ni mis hors ligne.
2. Segments publics/privés : en nombre quelconque (cf. *create rollback segment/undo tablespace*) ;

Types de *Rollback segments*

3. *Segment d'annulation différée* (*defferred rollback segment*) :
   - crée automatiquement dans la *tablespace SYSTEM*
   - contient, pour un espace de données (*tablespace*) mis hors ligne, les entrées d'annulation qui le concernent et qui, du fait de son état hors ligne, n'ont pas pu lui être appliquées lors d'annulations de transactions.
   - Au retour en ligne de la *tablespace* : utilisation du ou des segments d'annulation différée pour éventuellement y annuler les effets de certaines transactions.

Rollback segments* vs *Undo tablespaces

- A partir de la version 9i : choix lors de la création de la base
- Utilisation de *rollback segments* : *UNDO_MANAGEMENT = MANUAL* dans init.ora
- Utilisation de *Undo tablespaces* : dans init.ora, paramètres
  1. *UNDO_MANAGEMENT = AUTO*
  2. *UNDO_TABLESPACE* = Nom de la *tablespace*
- Création d'une *undo tablespace* :
  1. lors du *create database* : argument *undo tablespace*
  2. ou commande *create undo tablespace*
- Extension d'une *undo tablespace* : *alter tablespace ... add datafile ...*

***Undo tablespaces* : Exemple**

create database ...
undo tablespace "MonUndoTs"
datafile '...\UndoTs01.dbf' *size* 50M,
'...\UndoTs02.dbf' *size* 50M *reuse*
autoextend on next 5120K *max size* 1000M;

Rollback segments* vs *Undo tablespaces

- Changement de *undo tablespace*
  1. Créer une *undo tablespace*
  2. Arrêter la base
  3. Modifier le fichier d'initialisation de la base
  4. Démarrer la base
  5. Supprimer l'ancienne *undo tablespace* : *drop undo tablespace*.
- *Durée de rétention des images avant* :
  - Pour "interroger le passé"
  - Paramètre *dynamique UNDO_RETENTION*
  - Paquetage PL/SQL *dbms_flashback*

2.2.3- Validation et annulation de transactions

- Début transaction = première instruction SQL exécutable
- Fin si :
  1. *commit* ou *rollback* "total", (i.e. pas *rollback to savepoint*).
  2. Rencontre d'une instruction de définition de données comme (*create, drop alter table, ...*) :
     (a) Validation de la transaction si elle comportait des instructions de modification
     (b) Exécution de l'instruction de définition de données comme une transaction.
  3. Déconnexion par l'utilisateur.
  4. Arrêt en échec du processus utilisateur : transaction avortée.

Validation de transactions : Les effets

1.
2.
3.
4.
5.
6.

2.2.3- Cas Echec des transactions

1. Restaurer les anciennes valeurs des données à partir des segments d'annulation
2. Libérer les ressources verrouillées.

2.2.4.a- Isolation, verrouillage et points de contrôle

- Trois *niveaux d'isolation* :
  1. *read committed*,
  2. *serializable*,
  3. *read only*.
- *read committed* = niveau par défaut
- *read only* ne fait pas partie du standard SQL'92
- Pour une transaction, les trois niveaux d'isolation sont applicables (*set transaction isolation level read committed*, *serializable* ou *read only*)
- Pour une session, seuls les deux premiers (*alter session set isolation_level read committed* ou *serializable*).

2.2.4.b- Isolation, Verrouillage et points de contrôle

- Verrouillage de niveau tuple
- Différents types de verrous (*Share*), *eXclusive*), partagés ou exclusifs de niveau tuple (*Row Share*, *Row EXclusive* respectivement), etc.
- Choix par le système du plus petit niveau de verrouillage lors de la rencontre d'une instruction de modification ou de définition de données.
- Verrous maintenus pendant toute la durée de la transaction
- Demandes explicites de verrous :
  - *set transaction isolation level*,
  - *lock table*,
  - *select for update*
  - *alter session set isolation_level* : pour la durée d'une session.

2.2.4.b- Isolation, Verrouillage et points de contrôle

- *Déverrouillage* :
  - Fin de la transaction (annulation ou validation).
  - Cas d'une annulation partielle (*rollback to savepoint*) : Déverrouillage des ressources acquises entre le point de sauvegarde (*savepoint)* et le point d'annulation.
- *Interblocage* : Détection par Oracle
  - Fine granularité du verrouillage $\longrightarrow$ situations devraient être rares.
  - Cas interblocage :
    - Annulation des seuls effets d'une des instructions impliquées
    - Envoi d'un message à la transaction
    - Utilisateur : annuler toute la transaction ou réexécuter ultérieurement l'instruction annulée.

2.2.4.c- Isolation, Verrouillage et Points de contrôle

- Processus *CHKPT* : met à jour des informations de l'en-tête des fichiers de données (*datafiles*) en y inscrivant des informations relatives à la prise des points de contrôle (*checkpoint*).
- Point de contrôle explicite : *alter system checkpoint*
- Point de contrôle implicite : périodiquement par le système.

2.2.4.c- Isolation, Verrouillage et Points de contrôle

- Paramètres de contrôle de la fréquence :
  1. *log_checkpoint_interval* : Périodicité en termes de nombre de blocs physiques (système hôte et non Oracle) du journal,
  2. *log_checkpoint_timeout* :
     - spécifié en nombre de secondes,
     - limite le temps écoulé entre l'enregistrement du *log* le plus récent et le point de contrôle
     - $\simeq$ temps maximum qu'un tampon modifié du cache peut y séjourner avant d'être inscrit sur disque.

Systèmes de Gestion de Bases de Données

Chapitre

Contraintes d'intégrité et Confidentialité

Nacer Boudjlida

Confidentialité et Contraintes d'intégrité

- *Sécurité, protection, confidentialité* : empêcher l'accés à une base ou à une de ses parties par des personnes non autorisées
- *Intégrité* : maintenir la base dans un état cohérent vis-à-vis de contraintes données
- *Similitudes* :
  - Spécification de contraintes
  - Conservation (dictionnaire)
  - Préservation (SGBD ++)
- *Contenu du chapitre* :
  1. Confidentialité et administration
  2. Contraintes d'intégrité

Confidentialité et Protection de données

- *Aspects* : légaux, éthiques, techniques
- Assurées par un *sous-système d'autorisation*
- *Techniques* :
  - Numéros de compte, mots de passe
  - Allocation/Révocation de droits
  - Encryptage/Décryptage
  - "ad hoc" : BdD Statistiques

Sécurité et Administration

- *Utilisateur* : Numéro de compte, Mot de passe
- *Audit trail* : Fichier "espion", trace, par numéro de compte, de toute action effectuée ou tentée
- *Administrateur* : Utilisateur privilégié
  - Gestion des utilisateurs, de leurs droits
  - Gestion des espaces
  - Sécurité, performances, etc
- *Types de Droits* :
  - Compte : types de commandes/compte
  - "Objet" : relation, vue, procédure, etc.

Privilèges de niveau compte

- Création d'objets (*CREATE TABLE, VIEW, PROC, . . .*)
- Suppression d'objets (*DROP TABLE, VIEW, etc.*)
- Modification de schéma (*ALTER*)
- Consultation (*SELECT*), Insertion (*INSERT*)
- Modification (*UPDATE*), Suppression (*DELETE*)

Privilèges de niveau objet d'une base

- Contrôler accès, utilisation des objets
- Spécifier les types de commandes autorisées
- Compatibilté avec ceux de niveau compte
- Alloués par le *propriétaire*
- *Rappel* : Nommage des objets ([NomPropriétaire.]NomObjet)
- *Types* : *SELECT*, *UPDATE*, *ALTER*, *EXECUTE*, etc.
- *Privilèges parfois transmissibles* (Option GRANT)

Notion de rôle

- *Rôle* : Ensemble nommé de privilèges (assignable à un utilisateur ou à un rôle)
- Création, allocation, suppression de rôles :

Oracle : Gestion des utilisateurs

- Versions < 7 : gestion des utilisateurs et des privilèges : *grant* et *revoke* ;
- Versions ≥ 7 :
  - Notion de rôle ;
  - Commandes explicites de gestion des utilisateurs.
- La suite :
  1. Gestion des comptes et des utilisateurs ;
  2. Ressources et profils ;
  3. Gestion des rôles ;
  4. Gestion des privilèges.

2.1. Gestion des comptes et des utilisateurs

- Six types d'utilisateurs Oracle :
  1. Administrateurs (rôle *DBA*),
  2. Responsables sécurité (rôle *OPER*),
  3. Développeurs d'applications,
  4. Administrateurs d'applications,
  5. Utilisateurs d'une base,
  6. Administrateurs réseau.

2.1.1- Les administrateurs

- AU moins deux comptes, par défaut, créés lors de la création d'une base :
  1. *sys*/*change_on_install*
  2. *system*/*manager*.
- *dba* : rôle prédéfini
  - Créé lors de la création d'une base ;
  - Octroyable à d'autres utilisateurs par *sys* ou *system*.
- Authentification des utilisateurs bénéficiant de ces rôles :
  - Fichier de mots de passe ou
  - Système d'exploitation.

2.1.1- Les administrateurs : Exemple

- [redacted]
- Exemple :

2.1.2- Les utilisateurs d'une base

- *licence_max_sessions* (*init.ora*) : Nombre maximum de sessions simultanées
- *licence_max_users* : Nombre maximum d'utilisateurs qu'on peut créer dans une base ;
- Création utilisateur :
  - dans la base,
  - droit administrateur ou *create user*,
  - Nom, mot de passe, mode d'identification
- Autorisation de connexion : privilège *"connect"* ou *"create session"*
- Modification des caractéristiques d'un utilisateur : *"alter user"*

2.1.2- Les utilisateurs d'une base

- [redacted]
- Affectation d'espace par défaut :
  - *create user* ou *alter user* ;
  - *default tablespace* ;
  - *temporary tablespace*
- Suppression : *drop user*
- *drop user… "cascade"* : Suppression de l'utilisateur et de ses objets

2.1.2- Les utilisateurs d'une base : Exemples

2. Oracle : Gestion des utilisateurs

- La suite :
  1. Comptes et utilisateurs ;
  2. Ressources et profils ;
  3. Gestion des rôles ;
  4. Gestion des privilèges.

2.2. Limitations de ressources et notion de profil

- *Profil* : ensemble nommé de limites de ressources,
- Exemples de ressources :
  - Nombre de sessions (*sessions_per_user*),
  - Temps unité centrale (*cpu_per_session*, *cpu_per_call*),
  - Temps de connexion (*connect_time*) par session,
  - Nombre de lectures logiques (*logical_reads_per_session* ou *per_call*).
- Autres éléments possibles d'un *profil* :
  - durée de vie d'un mot de passe (*password_life_time*)
  - Nombre de tentatives de connexion infructueuses au delà duquel le compte est verrouillé (*failed_login_attempts*).

2.2. Limitations de ressources et notion de profil

- Permettre la limitation de ressources :
  - *resource_limit = true* (*init.ora*)
  - ou pendant la session :
    alter system set resource_limit = true
- *Profil par défaut* :
  - Dans chaque base,
  - Ressources illimitées (*unlimited*),
  - Hérité par tout utilisateur sans profil explicite.

  $\Longrightarrow$ définir des profils avec les "bonnes limites".

2.2. Limitations de ressources et notion de profil

- Opérations sur les profils :
  - Créer, modifier, supprimer : *create, alter, drop profile* ;
  - *Droits* : administrateur ou *create, alter, drop profile*.
- Affectation de profils :
  - {*create, alter*} *user … profile…*
  - Limites du profil remplacent celles du profil par défaut ;
  - Limites non spécifiées par le profil gardent leurs valeurs par défaut!
- [redacted]

2.2. Notion de profil : Exemple

1. Définition,
2. Restauration valeur par défaut,
3. Affectation à un utilisateur.

2. Oracle : Gestion des utilisateurs

- La suite :
  1. Comptes et utilisateurs ;
  2. Ressources et profils ;
  3. Gestion des rôles ;
  4. Gestion des privilèges.

2.3. Gestion des rôles

- Rôles prédéfinis : *sysdba* (*connect, dba, resource* : compatibilité versions $<$ V8)
- Autres rôles prédéfinis installables (Scripts fournis) :
  - Rôles *exp_full_database* et *imp_full_database* : Import/Export données
  - Script *catexp.sql*
- Création de rôles (*create role*) avec/sans mot de passe
- Affectation à des rôles et/ou des utilisateurs (*grant role*).

2.3. Gestion des rôles : Exemples

2.3. Gestion des rôles

- Affectation rôle par défaut : *alter user . . . default role . . .*
- A la connexion, privilèges = privilèges octroyés $\cup$ ceux du rôle par défaut
- Bénéficier des privilèges d'un rôle : *set role*
- Nombre maximum de rôles endossables pendant une session $\leq$ *max_enabled_roles* (*init.ora*).

2. Oracle : Gestion des utilisateurs

- La suite :
  1. Comptes et utilisateurs ;
  2. Ressources et profils ;
  3. Gestion des rôles ;
  4. Gestion des privilèges.

2.4. Gestion des privilèges sous Oracle

- Types de privilèges :
  1. Objets
  2. Système

2.4.1- Les privilèges sur des objets

- Qui : propriétaire objet
- Quoi :
  - *select*, *update, delete, insert* : vues ou relations,
  - Utiliser une relation dans une contrainte (*references*),
  - Exécuter (*execute*) des procédures, des fonctions ou des paquetages.
  - privilège *update* sur certaines colonnes.
- Droit de propager des privilèges acquis : *grant ... with admin option*
- Révocation en cascade : *revoke ... cascade constraints*.

2.4.1- Privilèges sur des objets : Exemple

2.4.2- Privilèges système

- Une centaine ! (cf. Manuel de référence)
- Créer/supprimer/modifier objets système ou base de données (session, user, profils, rôles, index, tables, vues, triggers, etc.)
- Agir sur :
  - Base : *alter database*
  - Système : *alter system*
  - *Audit* : *audit system*
- Egalement transmissibles.

2.4.2- Privilèges système : Exemple

2.4. Gestion des utilisateurs : Eléments du dictionnaire

- *all_users*, *user_users*, *dba_users*,
- *user_ts_quotas*, *dba_ts_quotas*,
- *user_password_limits*, *user_resource_limits*,
- *dba_profiles*,
- *v$session*, *v$sesstat*, *session_roles*,
- *session_privs*,
- *all_tab_privs_recd*, *user_tab_privs_recd* (privilèges reçus),
- *all* et *user_tab_privs_made* (privilèges octroyés),
- *all* et *user_col_privs_recd* (ou *made*),
- *user* et *dba_role_privs*, ...

2.4. Gestion des utilisateurs : Eléments du dictionnaire

- *"select * from session_privs"* : Privilèges en vigueur pour la session courante ;
- *select table_name, grantee, grantor, privilege from user_tab_privs_made*
  - noms des objets (*table_name*),
  - nom du bénéficiaire (*grantee*),
  - désignation du privilège (*privilege*)
  - nom de l'utilisateur qui l'a octroyé (*grantor*).

Les Contraintes d'Intégrité (CI)

- Schéma : description des données ET des contraintes
- *Contraintes d'Intégrité* : Assertions devant être vraies durant la vie de la base ou en des instants déterminés
- *Cohérence d'une base* : satisfaction de ses contraintes
- *Structure de la présentation* :
  1. Typologie des contraintes
  2. Spécification et stratégies d'implantation

Une typologie des contraintes d'intégrité

1. *CI Individuelles* : sur un type de donnée
   - Domaine (entier, réel, intervalle ou liste de valeurs, ...),
   - Valeur obligatoire (NOT NULL), etc
2. *CI Intra-relation* : sur un ensemble de tuples ou sur les valeurs des attributs d'un tuple au sein d'une relation
   - Unicité de valeur, clés,
   - Cardinalité : *dépôt numéro 3 ne stocke que les produits P1, P2 et P3*
   - Dépendances : *Produits P4 et P5 ne doivent pas être stockés dans le même dépôt*

Typologie des CI (suite)

3. *CI Inter-relations* : dépendances référentielles (existentielles ou d'inclusion)
   $\Pi_{No\_Prod}(stock) \subseteq \Pi_{No\_Prod}(Produit)$
4. *Contraintes Dynamiques* : Valeurs dans des états différents
   - Pas de diminution de salaires
   - Situation matrimoniale : pas de retour à célibataire
5. *Contraintes Temporelles* : commandes de la semaine courante à traiter avant le premier jour ouvrable de la semaine suivante.

CI : les Problèmes

1. *Spécification* : parfois dans le LDD
2. *Implantation* : SGBD ou programmation
3. *En cas de violation* ?
   - Annuler des actions (cf. gestion des transactions)
   - Rétablissement par inférence
4. *Cohérence* : Non contradiction (techniques de preuve)

CI : Spécification

1. *Spécification procédurale* : au sein des programmes
2. *Spécification déclarative* : Langage de type CP1
   - Stockage dans le dictionnaire
   - Préservation : *Sous-système de contrôle de l'intégrité*
   - Exemple (System-R, 1976) :

```
ASSERT contrainte_sur_salaire
ON personne p c , equipe e :
            p.salaire < c.salaire
     AND    p.#equ = e.#equ
     AND    e.#resp_equ = c.#id
```

CI : Spécification

3. *Utilisation des triggers*
   - Partie événement déclencheur
   - Partie Condition
   - Partie Action

   - Exemple (System-R, 1976) :

```
DEFINE TRIGGER contrainte_sur_salaire
ON personne p c , equipe e :
                p.salaire > c.salaire
      AND       p.#equ = e.#equ
      AND       e.#resp_equ = c.#id
ACTION Crime-de-Lese-Majeste(p.#id, c.#id)
```


CI : Exemples

1. *CI exprimables dans le LDD* (Create table)

   - Unicité : UNIQUE, PRIMARY KEY
   - Inclusion : REFERENCES
   - Valeur Obligatoire : [NOT] NULL
   - Valeur par Défaut : DEFAULT *Expression_Constante*
   - Autres : CHECK *(Expression logique)*
   - Remarques :

   (a) Contraintes nommées
   (b) Modification pendant la durée de vie de la base
   (c) "Partage" de contraintes
   (d) Activation/Désactivation (ENABLE/DISABLE CONSTRAINT)

CI et LDD : Exemples

```
CREATE TABLE produit
     (prod# int PRIMARY KEY CONSTRAINT C_NoProd,
      libelle VARCHAR(30)
     )
CREATE TABLE stock
     (prod# int,
      dep# int CHECK (dep# BETWEEN 22 AND 500)
      qte int DEFAULT 0 CONSTRAINT C_qte
      PRIMARY KEY (prod#, dep#),
      FOREIGN KEY (prod#) REFERENCES produit(prod#)
      CHECK (qte >= 0) CONSTRAINT C_stock)
```


CI et LDD : Exemples (suite)

```
CREATE TABLE depot
     (no_dep int NOT NULL,
      adr_dep VARCHAR(50),
      capacite int),
      PRIMARY KEY no_dep CONSTRAINT C_NoDep
      CHECK (capacite > 0))
```


CI Sybase : Triggers

2. *Déclencheurs ou Triggers*
   - Evénement : *insert, update, delete*

   - Objet : *une* relation

   - Action : Programme Transact-SQL (≃ PL/SQL Oracle) sur relations de base et *relations logiques*
     - *inserted* : tuples modifiés (update) ou insérés
     - *deleted* : tuples avant modification (update) ou supprimés

   - Un trigger au plus par type d'événement par relation

CI Sybase : Triggers (suite)

(a) *Création* : propriétaire de relation ou ayant droit

```
CREATE TRIGGER delprod
ON produit FOR DELETE
AS  IF ( SELECT COUNT(*) FROM deleted, stock
            WHERE deleted.#prod = stock.#prod) > 0
       BEGIN
            ROLLBACK TRAN
            PRINT "– Produit existe en stock: Suppression refusee"
       END
       ELSE PRINT "– Suppression effectuee"
```

(b) *Suppression* : DROP TRIGGER delprod

Triggers sous Oracle

- Trigger $\simeq$ Sorte de règle ECA :

  Si l'événement E survient
  Alors Si la condition C est vraie
  Alors exécuter l'action A

- **Définition d'un trigger sous ORACLE :** CREATE OR REPLACE TRIGGER

Triggers sous Oracle : Composants

1.
2.
3.
4.
5.
6. .
   -
   -

Triggers Oracle

- Exemple :

  AFTER UPDATE OF qte_en_stock ON Inventaire
  — Evénement (UPDATE), Instant (AFTER) et Objet (Inventaire)
  WHEN (new.qte_en_stock < new.seuil) — Condition
  FOR EACH ROW — Nombre de déclenchements
  <Bloc PL/SQL> — Action

- Action d'un trigger v.s. Procédure ou Fonction
  - Procédure ou Fonction : sur invocation explicite
  - Trigger : sur arrivée de l'événement et si condition satisfaite

Triggers : Cas d'utilisation (exemples)

1. Générer des valeurs dérivées (calculées)
2. Empêcher l'exécution de transactions (programmes) non valides
3. Implanter des contraintes d'intégrité dynamiques
4. Implanter des fonctions de surveillance (audit)
5. Synchroniser des tables répliquées
6. Implanter des contraintes référentielles dans une base de données distribuées
7. etc.

Triggers Oracle : Types d'événements

1.
2.
3.

- La partie action peut :
  1. Contenir du SQL, PL/SQL, Java, C
  2. Définir des "objets" PL/SQL (Variables, constantes, curseurs, etc.)
  3. Appeler des procédures stockées

Triggers Oracle

- **Notes :**
  1. Corrélation (ancienne/nouvelle valeur)
     NEW.NomDeColonne, **OLD**.NomDeColonne
  2. Renommage de **NEW** et de **OLD** : cf. clause **REFERENCING OLD/NEW AS** dans CREATE TRIGGER
  3. "Cascade" de triggers (et terminaison)
- Typologie des triggers (et plan de la suite) :
  1. Sur instruction, sur ligne (FOR EACH)
  2. Avant (BEFORE)/Après (AFTER) exécution de l'événement déclenchant
  3. "Au lieu de" l'instruction déclenchante (INSTEAD OF)
  4. Sur événement système ou utilisateur

Typologie des triggers : 1) Sur instruction, sur ligne

- Lors de la déclaration du trigger, indication du nombre de fois où il doit être exécuté
  1. Autant de fois que de lignes concernées (FOR EACH ROW)
  2. Une seule fois (par défaut)

Typologie des triggers : 1.1) Trigger sur ligne

- Déclenché chaque fois qu'une table est concernée
- Exemple : update ... where ...
- Cas d'utilisation : Quand l'action du trigger **dépend** de la valeur des données de l'instruction déclenchante ou de celle des tuples concernés

Typologie des triggers : 1.2) Sur instruction

- Activé **une seule fois**, quel que soit le nombre de tuples concernés
- Cas d'utilisation : Quand l'action du trigger **ne dépend pas** de la valeur des données de l'instruction déclenchante ou de celle des tuples concernés
- Exemples d'usage :
  - Surveillance (*auditing*)
  - Tests complexes sur le temps ou sur un utilisateur

Typologie des triggers : 2) AVANT/APRES

- S'appliquent pour les deux types précédents
- Si trigger sur instruction du LMD :
  - S'applique sur des tables, **pas sur des vues**
- Si trigger sur instruction du LMD :
  - S'applique sur la base ou sur un schéma

Typologie des triggers : 2) AVANT/APRES

1. Trigger AVANT
   - Exécuté avant l'exécution de l'instruction (update, delete)
   - Exemple 1 : Vérifier que l'instruction est autorisée
   - Exemple 2 : Dériver des valeurs de colonnes avant modification
2. Trigger APRES : Exécuté après l'exécution de l'instruction

Triggers Sur instruction, Sur ligne : Conclusion

- 4 compositions possibles
- Possiblité de définir **plusieurs triggers d'un même type** pour toute instruction du LMD (insert, update, delete)
- 4 combinaisons possibles :

1. **Trigger de type AVANT instruction** :
   - Attention : Lignes verrouillées (cf. accès concurrents)
2. **Trigger de type AVANT chaque ligne** :

   Si la condition gardant le trigger est vraie
   alors exécuter l'action du trigger AVANT modification des lignes
   ET AVANT vérification des contraintes d'intégrité

Triggers Sur instruction, Sur ligne : Conclusion

3. **Trigger de type APRES instruction** :

   Exécuter l'action du trigger APRES modification des lignes
   ET APRES vérification des contraintes d'intégrité

4. **Trigger de type APRES chaque ligne** :

   Même modèle d'exécution qu'un trigger AVANT
   mais réalisé APRES l'instruction ET pour chaque ligne

Typologie des triggers : 3) De substitution (INSTEAD OF)

- Exécution de l'action du trigger à la place de l'instruction
- Cas d'utilisation : Mise à jour **"à travers une vue"**
  - Pour mettre à jour les tables sous-jacentes
  - **Quand la mise à jour à travers la vue** n'est pas possible
- Vues non modifiables : Leur définition comporte :
  - des opérateurs ensemblistes, des jointures
  - des fonctions d'agrégation (somme, moyenne, etc.)
  - GROUP BY, START WITH, CONNECT, clause DISTINCT
- Vues modifiables
  - Celles non modifiables
  - **et quand** la mise à jour n'est pas ambigüe

Typologie des triggers : 4) Sur événement système ou utilisateur

- Pour publication d'information sur la base à l'intention "d'abonnés"
- Abonnement d'applications à des événements sur la base (package *DBMS_AQ*)
- Evénements Système : sur base ou schéma
  1. Lancement/Arrêt (startup/shutdown)
  2. Guardes sur les transitions de rôle
  3. Messages d'erreurs du serveur de données

Typologie : 4) Sur événement système ou utilisateur

- Evénements Utilisateurs :
  1. Connexion/Déconnexion (logon/logoff) : sur base ou schéma
  2. Instructions du LDD (CREATE, ALTER, DROP) : sur base ou schéma
  3. Instructions du LMD (INSERT, UPDATE, DELETE) : sur table ou vue
- Publication/Abonnement (*publish/subscribe*) à des événements : Mécanismes de *Oracle Streams Advanced Queuing*. Voir :
  - *Oracle Streams Advanced Queuing User's Guide and Reference*
  - *Oracle Database PL/SQL Packages and Types Reference*

Typologie : 4) Sur événement système ou utilisateur

- Exemple :

```
CREATE TRIGGER Si_Arret
  ON DATABASE SHUTDOWN
  BEGIN
    ...
    DBMS_AQ.ENQUEUE(...);
    ...
  END;
```


Triggers : Modèles d'exécution

1. Cas triggers multiples sur la même instruction : dans l'ordre,
   (a) Triggers de niveaux instruction
   (b) Triggers de niveaux ligne
2. Cas triggers multiples d'un même type : Ordre aléatoire
3. Triggers et Test de contraintes : une instruction SQL peut déclencher 4 types de triggers
   (a) Before row
   (b) Before instruction
   (c) After row
   (d) After instruction
   - L'instruction déclenchante et l'action du trigger peuvent mettre en cause des contraintes d'intégrité : quel modèle d'exécution?

Triggers et CI : Modèles d'exécution

1.
2. .
   (a)
   (b)
   (c)
   (d)
   (e)
3.
4.

Triggers et CI : Modèles d'exécution

- **Modèle d'exécution récursif** :
- Exemple de récursion : Scenario
  1. Instruction déclenche un trigger "BEFORE ROW" **T-before-R** + Test CI
  2. **T-before-R** fait une mise à jour qui :
     (a) Nécessite un test de CI
     (b) Déclenche un trigger "APRES instruction" **T-after-I**

Triggers et CI : Exemple d'exécution récursive

Triggers : Remarques

1. Privilèges requis pour la création :
   (a) Sur des objets possédés : CREATE TRIGGER
   (b) Sur ceux non possédés : CREATE ANY TRIGGER, ALTER ANY TABLE
   (c) Sur une base : ADMINISTER DATABASE TRIGGER
2. Autoriser (ENABLE)/Inhiber (DISABLE) le déclenchement :
   - cf. ALTER TABLE ... ENABLE/DISABLE ALL TRIGGERS ou ALTER TRIGGER ... ENABLE/DISABLE
3. Accès concurrents aux données : Mêmes règles de verrouillage que pour une transaction

Triggers : Remarques

4. Atomicité : Tout ou rien, effets de l'instruction déclenchante compris
5. Exécution : $\simeq$ procédures stockées
6. Stockage : Sous forme PseudoCode compilé (Pas de stockage du source)
7. Information : cf. USER_TRIGGERS, ALL_TRIGGERS et DBA_TRIGGERS dans le dictionnaire
8. Cas où un trigger est gardé par plusieurs instructions : prédicats INSERTING, DELETING et UPDATING

Triggers : Remarques

Triggers : Remarques

9. Cas où un UPDATE ne concerne que certaines colonnes :

Triggers : Remarques

10. Arrêt de l'exécution d'un trigger : Lever une erreur
    - Procédure *raise_application_error (No-Erreur, Message-Erreur)* :
      - $No - Erreur \in [-20999.. -20000]$
      - Message-Erreur : chaîne d'au plus 500 caractères
    (a) Arrêt de l'action du trigger
    (b) Annulation des effets de la transaction
    (c) Emission du No Erreur et du message vers l'application
11. Triggers et Exceptions :
    - Si exception non "programmée" : Annulation des effets du trigger et de ceux de l'instruction déclenchante
    - Exceptions "programmées" : cf. PL/SQL

Triggers : Remarques

- Quelques exceptions prédéfinies (Rappels) :
  NO_DATA_FOUND : levée quand un SELECT INTO ne retourne pas de lignes
  DUP_VAL_ON_INDEX : tentative d'insertion d'une ligne "en double" dans un index UNIQUE
  ZERO_DIVIDE : division par zéro

Exemples

Triggers Oracle : Conclusions

- Utiliser des triggers pour garantir que, lorsqu'une opération spécifique est effectuée, les actions liées ont aussi réalisées.
- N'utiliser des triggers que s'il n'y a pas d'autre possibilité ((exemple des contraintes déclaratives).
- Eviter des triggers récursifs (un trigger AFTER UPDATE faisant un UPDATE sur la même table).

CI et Triggers : les problèmes

1. *Déclenchement* : Immédiat, différé, etc
   - Sybase : Immédiat
   - Oracle : Avant/Après/Pour chaque tuple
2. *Propagation et terminaison* (Cascading)
   - Oracle : à la charge de l'utilisateur
   - Sybase :
     - Paramètre de configuration (profondeur 16, sa)
       sp_configure "allow nested triggers", 0 | 1
     - Récursion : Action d'un trigger T déclenche T
       set self_recursion ON | OFF
3. *Atomicité* : cf. transactions imbriquées

Conclusion

- *Protection, Confidentialité* : Vues, privilèges
- *Contrainte d'Intégrité* :
  - Pas de typologie universelle
  - Utilisables pour :
    1. Qualité de l'information
    2. Déduction (BdD Déductives)
    3. Optimisation sémantique
- *Problèmes non résolus* (BdD Actives) :
  - Evénements "user-defined"
  - Evénements composites
  - Terminaison, etc.

Bases de Données relationnelles et leurs systèmes de Gestion

RAPPELS

- **Contraintes d'intégrité sous Oracle**
- **Notion de vue**

III.1- Définition de schémas

1. Typage des attributs
2. Contrainte d'intégrité
   1. Intra-relation
   2. Inter-relations (CI Référentielle)
3. Conduite à tenir si violation

Syntaxe de CREATE TABLE (1/2)

```
CREATE TABLE nomTable
      (définitionDattribut
      [,définitionDattribut]…
      [,définitionDeContrainte]…));
```

- *définitionDattribut*

```
nomAttribut  [typeDeDonnées|domaine]
Contraintes { [DEFAULT valeur][NULL|NOT NULL]
              [UNIQUE|PRIMARY KEY]
              [REFERENCES nomTableBis(nomAttributBis)]
              [[CONSTRAINT nomContrainte]
                           CHECK (condition)]
```

NULL Autorisé ou pas

Typage (domaine)

Syntaxe de CREATE TABLE (2/2)

Clé Primaire

Nommer une contrainte

Inclusion/Référence

```
[CONSTRAINT nomContrainte]
{PRIMARY KEY listeAttributs |
 FOREIGN KEY listeAttributs REFERENCES
    nomTableBis[listeAttributs]
       [ON DELETE {NO ACTION|CASCADE|SET NULL|
                                SET DEFAULT}]
     [ON UPDATE {NO ACTION|CASCADE|SET NULL|
                                SET DEFAULT}] |
 CHECK (condition)
 }
```

Que faire si…

Autres contraintes

III.1.2- Contraintes d'intégrité (CI)

- Plusieurs types de contraintes statiques déclarables avec **`CREATE TABLE`**
  - Contraintes sur le **domaine** d'un attribut
  - Contraintes de **clés** (primaire, candidate)
  - Contraintes **d'intégrité référentielle** (contrainte d'**inclusion** et **clé étrangère**)
- D'autres contraintes peuvent être programmées et/ou porter sur des changements d'états de la BD, sur plusieurs tables…
  - **`CREATE TRIGGER`** (non traité ici)

CI sur le domaine d'un attribut (1/5)

- Type de données (number, char…)
- Contrainte **NOT NULL**
  - Valeur obligatoire pour l'attribut (si pas de DEFAULT)

```
CREATE TABLE Client
 (noClient      INTEGER        NOT NULL,
  nom           VARCHAR(50)    NOT NULL,
  ddn           VARCHAR(15)    NULL,
  téléphone     VARCHAR(15) )
```

- Par défaut : NULL
  - Insertion de la valeur NULL si absence de valeur et de default

CI sur le domaine d'un attribut (2/5)

- Contrainte CHECK sur un attribut
- CHECK (Expression logique)
- Exemple: numéro de client ∈ [0..1000]

```
CREATE TABLE Client
 (noClient      INTEGER        NOT NULL
   CHECK (noClient>0 AND noClient<1000),
  nom   VARCHAR(50)    NOT NULL,
  ddn   VARCHAR(15)    NULL )
```


CI sur le domaine d'un attribut (3/5)

- Contrainte CHECK sur plusieurs attributs du même tuple (Contrainte intra-relation)
- *Les produits dont le numéro est supérieur à 100 ont un prix supérieur à 15 €.*

```
CREATE TABLE Produit
 (noProduit      INTEGER  NOT NULL,
  libellé        VARCHAR(50),
  prixUnitaire   NUMBER(10,2) NOT NULL,
  CHECK (noProduit<=100 OR prixUnitaire>15) )
```


CI sur le domaine d'un attribut (4/5)

- Création d'un domaine spécifique (`CREATE DOMAIN`)
- *Un employé/client est un homme ou une femme.*

```
CREATE DOMAIN domaineSexe AS
            CHAR(1) CHECK (VALUE IN ('M','F'))

CREATE TABLE employé
        (numEmp        INTEGER,
        genre    domaineSexe, … );
CREATE TABLE client
        (noClientINTEGER,
        genre    domaineSexe, … )
```

N.B. **Les domaines ne sont pas supportés dans Oracle.**

CI sur le domaine d'un attribut (5/5)

- Valeur d'un attribut par défaut (DEFAULT)
  - Si pas de valeur lors d'une insertion (NULL)
  - Ou attribut mis à NULL lors d'une modification
- Exemple: *numéro de téléphone : présence obligatoire et valeur "confidentiel" par défaut*

```
CREATE TABLE employé
 (numEmp       INTEGER,
  numTel       VARCHAR(15)   NOT NULL
        DEFAULT 'confidentiel', … )
```


III.1.2- Contraintes d'intégrité (CI)

- Plusieurs types de contraintes statiques déclarables avec **`CREATE TABLE`**
  - Contraintes sur le domaine d'un attribut
  - Contraintes de clé
    - Primaire
    - Candidate
  - Contraintes d'intégrité référentielle (contrainte d'inclusion et clé étrangère)

Contrainte de clé primaire

- Deux tuples ne peuvent pas avoir la même valeur de la clé
- Le(s) attribut(s) clé(s) ne peuvent être NULL
- *noClient: clé primaire de la table Client* (clé atomique)

```
CREATE TABLE Client
 (noClient  INTEGER     PRIMARY KEY,
  nom       VARCHAR(50) NOT NULL,
  ddn       VARCHAR(15) NULL );


CREATE TABLE Client
 (noClient  INTEGER ,
  nom       VARCHAR(50) NOT NULL,
  ddn       VARCHAR(15) NULL,
  PRIMARY KEY (noClient))
```

Contrainte de clé : clé composée

- *noClient x noProduit x dateCde = clé primaire de la table Commande*

```
CREATE TABLE Commande
        (noClientINTEGER,
        noProduit          INTEGER,
        dateCommande DATE,
        quantité INTEGER,
        PRIMARY KEY     (noClient, noProduit, dateCommande)
        )
```

- Lorsque la clé est composée de plusieurs attributs, définir la contrainte PRIMARY KEY au niveau de la table

Clé Candidate et Unicité

- Une seule clé primaire par table mais d'autres attributs peuvent avoir des valeurs uniques pour chaque tuple (exemple: clés candidates)
- Clause **UNIQUE**

```
CREATE TABLE Citoyen
        (numSécu        INTEGER PRIMARY KEY ,
        nom             VARCHAR(50),
        prénom          VARCHAR(50),
        ddn             DATE  NULL,    –date de naissance
        noPassport      INTEGER NULL  UNIQUE )
```


Contrainte d'intégrité référentielle (FOREIGN KEY / REFERENCES)

Contrainte inter-relations

- *Émetteur d'une Commande doit être un Client connu*

```
CREATE TABLE Commande
        (noCommande     INTEGER PRIMARY KEY,
        noClient INTEGER,          noProduit  INTEGER,
        dateCommande DATE,         quantité   INTEGER,


        );
```

- Tables ***Client*** et ***Produit*** déjà définies : pas de référence en avant
- Cible d'une référence : PRIMARY KEY ou UNIQUE
- Ajout de commandes autorisée : si le client et le produit existent

III.1- Définition de schémas

1. Typage des attributs
2. Contrainte d'intégrité
   1. Intra-relation
   2. Inter-relations (CI Référentielle)
3. Conduite à tenir si violation de CI référentielle

III.1.3- CI référentielle : conduite à tenir en cas de mise à jour

```
[CONSTRAINT nomContrainte]
FOREIGN KEY listeAttributs REFERENCES
      nomTableBis(listeAttributs)
[ON DELETE {NO ACTION|CASCADE|SET NULL| SET DEFAULT}]
[ON UPDATE {NO ACTION|CASCADE|SET NULL| SET DEFAULT}]
```


CI référentielle : conduite à tenir en cas de mise à jour (DELETE)

- Tentative de suppression de la clé primaire

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY, noClient INTEGER,
                    …
 FOREIGN KEY (noClient) REFERENCES Client(noClient))

DELETE FROM Client WHERE noClient=45
```

- **Que deviennent les commandes du client numéro 45 ?**
- 4 options :
  - NO ACTION
  - CASCADE
  - SET NULL
  - SET DEFAULT

CI référentielle : conduite à tenir en cas de suppression de la clé primaire

- Suppression du client 45 : que faire de ses éventuelles commandes ?
- NO ACTION : suppression du client 45 refusée
- CASCADE :
  1.
  2.
- SET NULL :
  1.
  2.
- SET DEFAULT :
  1.
  2.

CI référentielle : conduite à tenir en mise à jour (UPDATE)

- Tentative de modification de la clé primaire

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
 noClient   INTEGER,  …
 FOREIGN KEY (noClient) REFERENCES Client(noClient))

 UPDATE Client SET noClient=100 WHERE noClient=45
```

- ***Que deviennent les commandes du client numéro 45 ?***

CI référentielle : conduite à tenir en mise à jour (UPDATE)

- NO ACTION :
- CASCADE :
  1.
  2.
- SET NULL :
  1.
  2.
- SET DEFAULT :
  1. noClient = valeur par défaut si celle-ci est définie
  2. Modifier le client

CI référentielle : conduite à tenir en cas de mise à jour

```
CREATE TABLE Commande
(noCommande INTEGER PRIMARY KEY,
 noClient   INTEGER,  …
 FOREIGN KEY (noClient) REFERENCES Client(noClient)
  ON DELETE CASCADE
  ON UPDATE CASCADE )
```

- Sous Oracle
  - par défaut :
    - ON DELETE NO ACTION
    - ON UPDATE NO ACTION
  - seules possibilités dans CREATE TABLE
    - ON DELETE CASCADE
    - ON DELETE SET NULL

III.3. Modification de schémas (1/3)

- Ajouter un attribut (nom, type, défaut, NOT NULL uniquement) ou une contrainte
- Modifier ou supprimer le type ou la valeur par défaut d'un attribut
- Supprimer un attribut ou une contrainte
- Supprimer une contrainte: nommée, clé primaire, unicité
- Impact possible sur les programmes !!!!

III.3. Modification de schémas (1/3)

ALTER TABLE nomTable
ADD (attribut type [DEFAULT valeur] [contrainte]) |
MODIFY (attribut [type] [DEFAULT valeur] [contrainte]) |
DROP COLUMN attribut |
ADD [**CONSTRAINT** nom] contrainte |
DROP {{**PRIMARY KEY** | **UNIQUE**} (attribut) | **CONSTRAINT** nom}

- Renommage de table

```
RENAME ancienNomDeTable TO nouveauNomDeTable
```

III.3. Modification de schémas (2/3)

```
ALTER TABLE client ADD (TEL_PORTABLE CHAR(10))

ALTER TABLE client MODIFY (ADR_CLIENT CHAR(70))

ALTER TABLE produit ADD CONSTRAINT PRIX_POSITIF
check (PRIXUNITAIRE >= 0)

ALTER TABLE produit DROP CONSTRAINT PRIX_POSITIF

ALTER TABLE client  DROP COLUMN TEL_PORTABLE

RENAME CLIENT TO ACHETEUR
```


III.3. Modification de schémas (3/3)

- ALTER TABLE et désactivation d'une contrainte nommée

```
ALTER TABLE nom_table
DISABLE CONSTRAINT nom_contrainte;
```

- ALTER TABLE et activation d'une contrainte nommée

```
ALTER TABLE nom_table
ENABLE CONSTRAINT nom_contrainte;
```


III.4. Suppression d'un schéma (DROP TABLE)

DROP TABLE nom-relation
[**CASCADE CONSTRAINTS**]

- Suppression
  1. des **tuples** ET du **schéma** de la relation
  2. des **index**
  3. désactivation des **vues** liées
  4. des contraintes référentielles éventuelles (dans les relations où la clé primaire de la relation à supprimer est référencée)

Le langage SQL (SGBD ORACLE)

I. Introduction
II. Interrogation des données
III. Définition des données
IV. Les vues

IV. Les Vues (1/3)

- Vue : relation calculée
  - ses tuples ne sont pas stockés
  - sa requête de définition est stockée
  - ses tuples sont générés à chaque appel de la vue

```
CREATE [OR REPLACE] VIEW nom-vue[(attribut(s)]
 AS  (commande SELECT de définition)
```

- *Définir, comme une vue* ProduitsChers*, la liste des produits dont le prix dépasse 5000€. Mettre dans la vue les colonnes nomProduit et prixUnitaire*

Les Vues (2/3)

- Une fois créée, une vue est utilisable comme toute autre table
- *Liste des produits chers dont le libellé comporte 'de luxe'*

```
SELECT *     FROM ProduitsChers
WHERE        nomProduit  like '%de luxe%'

SELECT *  FROM  ProduitsChers pc, Stock s
Where ….
```

- Suppression d'une vue

DROP VIEW nom-vue

Les Vues (3/3): intérêt

- Masquage des opérations de jointure
- Sauvegarde (indirecte) de requêtes complexes
- Support de l'indépendance logique:

  si les tables sont modifiées, les vues doivent être réécrites mais les requêtes utilisant les vues ne subissent pas de changement
- Support de la confidentialité en combinaison avec des privilèges d'accès adéquats:

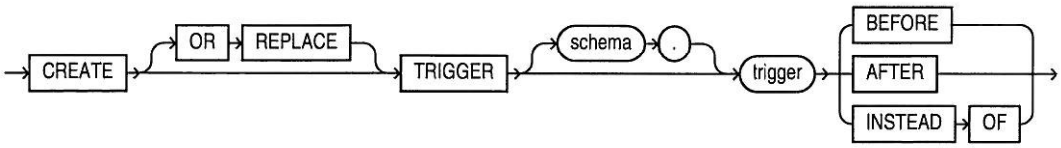
  Masquer les lignes ou colonnes pour les utilisateurs non autorisés

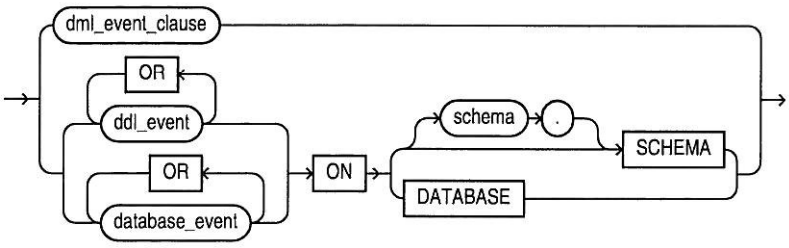
SQL : Conclusion

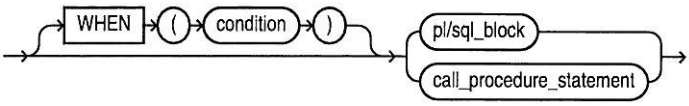
- Langage commun aux SGBDR
- Déclaratif, Orienté ensembles
- Fondement : Algèbre + Calcul relationnel
- Définition & Manipulation :
  - Schémas : CREATE, ALTER, DROP
  - Instances : SELECT, INSERT, UPDATE, DELETE
- Extensions de SQL :
  - Fonctions d'agrégation
  - Procédurales (si alors sinon, tant que, etc.)

Syntax

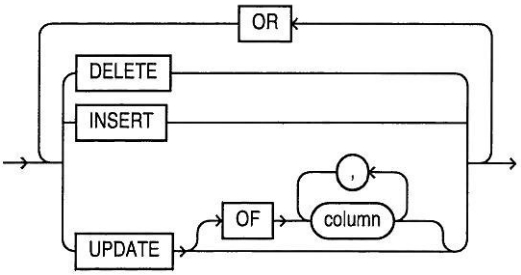
***create_trigger*::=**

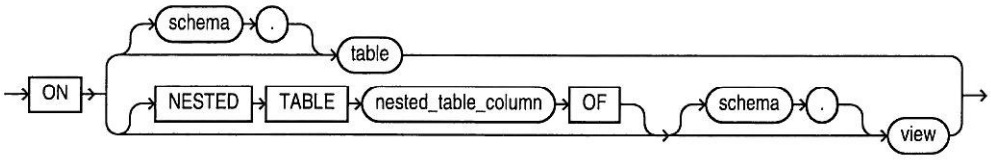


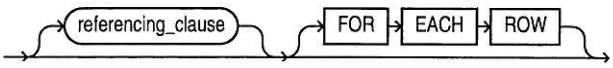




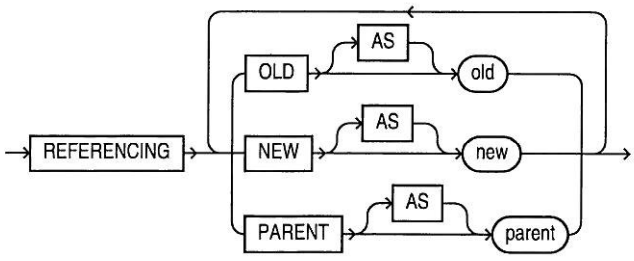
***DML_event_clause*::=**







***referencing_clause*::=**



Systèmes de Gestion des Bases de Données

Traitement des requêtes dans les SGBD-R

Nacer Boudjlida

Traitement des requêtes dans les SGBD relationnels

- Evaluation des requêtes :
  - Données de la base vers la mémoire centrale
  - Traitement
  - "Retour" dans la base si mise à jour
- Peut nécessiter la création de tables intermédiaires
- Volume des transferts et des tables de travail fonction des tailles des relations mises en jeu
- *Objectif d'un optimiseur* : Réduction des volumes et du temps

Traitement des requêtes : Besoins d'optimiser ?

- Temps d'exécution d'une requête "anormal" % taille des relations, existence d'index
- Une requête s'exécute plus lentement que des requêtes similaires
- ↗ temps d'exécution d'une requête
- Temps d'exécution d'une requête en tant que procédure > celui de son exécution en tant que requête
- Plan d'exécution utilisant un parcours de relation au lieu d'utiliser un index

Traitement des requêtes : Origines des mauvaises performances ?

- "Vieilles" statistiques sur la distribution des données
- Inexistence d'index appropriés
- Index en cours d'utilisation pour accéder à une table volumineuse
- Clause *where* conduisant au choix d'une mauvaise stratégie
- Procédure non re-compilée après changements significatifs
- etc.

Traitement des requêtes : Les méthodes

1. Syntaxique :
   - Fondement : Heuristiques + Propriétés de l'algèbre
   - Transformation d'arbres (algèbre) ou de graphes (calcul relationnel)
2. Estimation de coûts d'exécution de diverses stratégies d'évaluation
3. Sémantique : Exploitation des contraintes d'intégrité

- Méthodes non exclusives : 1 et 2 souvent combinées

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

I. Transformation d'arbres algériques

- Heuristiques : $\searrow$ taille des opérandes des opéraions les plus coûteuses
  1. "Descente" des sélections : $\searrow$ les relations "en hauteur"
  2. "Descente" des projections : $\searrow$ les relations "en largeur"
- [Phrase SQL $\longrightarrow$] Arbre algébrique :
  - *Feuilles* : Relations
  - *Racine* : Résultat de la requête
  - *Nœuds internes* : Opérations de l'algèbre
  - *Arcs "entrants"* : Opérandes
  - *Arcs "Sortants"* : Résultat de l'évaluation d'un nœud
- Evaluation de "bas en haut" (gauche-droite ou droite-gauche)

I. Transformation d'arbres algériques : Exemple

- *Personne*(id#, nom, prenom, date_naisssance, adresse, equ#)
- *Equipe*(equ#, nom_equ, resp_equ#)
- *Projet*(proj#, nom_proj, lieu_proj, equ#)
- *Travaille_sur*(id#, proj#, taux)
  - *Personne.equ#* : Equipe de rattachement
  - *Projet.equ#* : Equipe en charge du projet
  - *Travaille_sur.taux* : Part de temps sur un projet
- *Requête* :
  Personnes nées après 1962 et travaillant sur le projet "BDD"

I. Transformation d'arbres algériques : Exemple

- En SQL :

```
SELECT  nom
FROM    Projet, Travaille_sur, Personne
WHERE   Projet.nom_proj = 'BDD'
AND     Personne.id# = Travaille_sur.id#
AND     Personne.date_naissance > 'dec-31-1962'
AND     Travaille_sur.proj# = Projet.proj#
```

- *Représentation dite canonique* :
  - *Relations de la clause FROM* $\longrightarrow$ Produit cartésien (X)
  - *WHERE* $\longrightarrow$ Sélection (σ), nœud père du sous-arbre X
  - *SELECT* $\longrightarrow$ Projection (Π) en racine de l'arbre

Transformation d'arbres algériques : Arbre initial (0)

Transformation d'arbres : Arbre 1

- *"Descente" des sélections*

Transformation d'arbres : Arbre 2

- *Permutation de Personne et de Projet* (cf. notion de sélectivité) : **Si mélange méthodes syntaxique et estimation de coûts**.

Transformation d'arbres : Arbre 3

- *Introduction de jointures*

Transformation d'arbres : Arbre 4

- *Introduction de projections* : ne "faire remonter" que les attributs utiles

Transformation d'arbres : Sur l'arbre initial (Arbre 0)

- *Projet* : 20 tuples de 100 caractères
- *Travaille_sur* : 100 tuples de 50 caractères
- *Personne* : 500 tuples de 100 caractères
- *Deux produits cartésiens* : 10^6 tuples de 250 caractères !

Transformation d'arbres : Sur l'arbre final (Arbre 4)

- *1ère jointure* :
  - Une relation à une colonne et probalement à un n-uplet
  - Relation *Travaille_sur* : deux colonnes
- *2ème jointure* :
  - Une relation à une colonne (sous-arbre gauche)
  - Une relation à 2 colonnes (sous-arbre droit) réduite aux seuls n-uplets satisfaisant la sélection

Transformation d'arbres : Règles de transformation

- ***(R1)*** Décomposition des expressions de sélection.
  $Select(R,\ C_1 \wedge C_2 \wedge \cdots \wedge C_{n-1} \wedge C_n) \longrightarrow$
  $Select(Select(\cdots\ Select\ (Select(R,\ C_n),\ C_{n-1})),\ \cdots),\ C_2),\ C_1)$
- ***(R2)*** Commutativité de la sélection.
  $Select(Select(R,\ C_2), C_1) \longrightarrow Select(Select(R,\ C_1),\ C_2)$.
- ***(R3)*** Restriction de la liste de projections.
  $Proj_{L1}(Proj_{L2}(\cdots\ (Proj_{Ln}(R))\ \cdots) \longrightarrow Proj_{L1}(R)$.
- ***(R4)*** Commutation de la sélection et de la projection.
  Si C ne porte que sur les A_i :
  $Proj_L(Select(R,\ C)) \longrightarrow Select(Proj_L(R),\ C)$

Transformation d'arbres : Règles de transformation

- ***(R5)*** Commutativité de la jointure (et du produit cartésien).
  $Join_C\ (R,\ S) \longrightarrow Join_C(S,\ R)$.
- ***(R6)*** Commutation sélection-jointure (ou produit cartésien).
  - ***(R61)*** Si les attributs de la condition C de sélection n'appartiennent qu'à une des relations de la jointure, par exemple R, alors :
    $Select(Join_E(R,\ S),\ C) \longrightarrow Join_E(Select(R,\ C),\ S)$.
  - ***(R62)*** Sinon, si la condition C peut se ré-écrire en $C_1 \wedge C_2$ où :
    - C_1 ne porte que sur des attributs de R
    - $C_1\ C_2$ ne porte que sur ceux de S, alors :

    $Select(Join_E(R,\ S),\ C) \longrightarrow$
    $Join_E(Select(R,\ C_1),\ Select(S,\ C_2))$

Transformation d'arbres : Règles de transformation

- ***(R7)*** Commutation projection-jointure (produit cartésien).
  - ***(R71)*** $Proj_L(Join_C(R,\ S)) \longrightarrow Join_C(Proj_{L_1}(R),\ Proj_{L_2}(S))$
    1. Si attributs de C = attributs de L
    2. Si $L = L1||L2$ avec $L_1 = Liste(A_i),\ A_i$: attributs de R et $L_2 = Liste(B_j),\ B_j$: attributs de S
  - ***(R72)*** Si la condition de jointure comporte des sous-listes L_3 d'attributs de R et L_4 d'attributs de S n'appartenant pas à L :
    1. Les ajouter aux projections "internes"
    2. Appliquer une projection "externe" sur L

    $Proj_L(Join_C(R,\ S)) \longrightarrow Proj_L(Join_C(Proj_{L_1,\ L_3}(R), Proj_{L_2,\ L_4}(S)))$.

Transformation d'arbres : Règles de transformation

- ***(R8)*** Commutativité des opérations ensemblistes ($\cup$, $\cap$).
- ***(R9)*** Associativité : jointure, produit cartésien, union et intersection. $((R\ op_i\ S)\ op_j\ T) \longrightarrow (R\ op_j (S\ op_i\ S))$ où R, S, T sont des relations et op_i, op_j des opérateurs parmi ceux cités
- ***(R10)*** Commutation de la sélection et des opérations ensemblistes. (Select((R op S), C)) $\longrightarrow$ (Select(R, C) op Select (S, C)) où op est l'opérateur d'union, d'intersection ou de différence

Transformation d'arbres : Règles de transformation

- ***(R11)*** Commutation de la projection et des opérations ensemblistes. $(Proj_L(R\ op\ S)) \longrightarrow ((Proj_L\ (R))\ op\ (Proj_L(S)))$.
- Autres :
  - Equivalences logiques
  - Autres propriétés des opérateurs algébriques

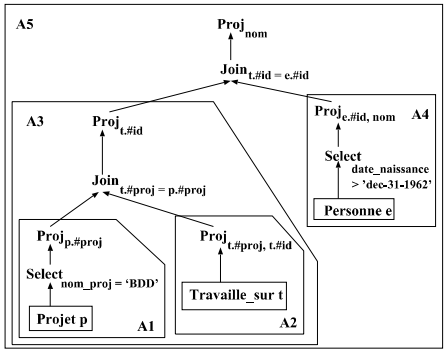
Transformation d'arbres : Ebauche d'algorithme

1. Transformer toute expression de sélection conjonctive en une "cascade" de sélections (Règle R1)
2. "Descendre" les sélections le plus bas possible ((R2), (R4), (R6), (R10)).
3. Ordonner les feuilles de l'arbre de sorte que les sélections les plus restrictives soient évaluées en premier ((R8), (R9))
   - **Etape à n'exécuter que si** combinaison méthode syntaxique et méthode par estimation de coûts ;
   - "Plus restrictives" = produisant les plus petites relations ;
   - cf. distribution des valeurs, index (dictionnaire) ;
   - cf. notion de sélectivité, plus loin.

Transformation d'arbres : Ebauche d'algorithme

4. (Produit cartésien; Sélection) $\longrightarrow$ Jointure, où la condition de jointure est la condition de sélection.
5. "Fragmenter" et "faire descendre" le plus bas possible les listes de projection, en créant de nouvelles projections, si besoin est (Règles (R3), (R4), (R7), (R11)) ,
6. Identifier et ordonner les sous-arbres qui représentent des groupes d'opérations pouvant être exécutés par une seule routine du SGBD.

Transformation d'arbres : Exemples de plan (Etape 6)



Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'optimisation

II. Optimisation de graphes de requêtes

- Technique dite de décomposition de requêtes
- Introduite dans Ingres/QUEL (Variables n-uplets)
- *Graphe de requêtes* :
  - *Sur les arcs* : Conditions de jointure des nœuds reliés
  - *Nœuds* : Variables de tuples ou constantes de la requête
  - *Arcs "'conditions de sélection"'* : relient des nœuds de constantes aux nœuds des variables impliquées dans les conditions.
- Requête à n variables $\longrightarrow$ m requêtes, plus simples, à (n-1) variables.

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

III. Optimisation par estimation de coûts

- Minimiser une fonction côut, avec comme facteurs :
  - Accès disque,
  - Coût du traitement en mémoire centrale,
  - Taille des n-uplets, des relations,
  - Index : Existence, nombre de niveaux,
  - Coût de communication (requête, résultats) entre sites (architectures client/serveur, par exemple),
  - etc.
- Utilisation effective (souvent) des coûts des accès

III. Optimisation par estimation de coûts

- *Informations du dictionnaire*
  - Nombre exact ou estimé de tuples et de blocs physiques par relation ;
  - Nombre de niveaux d'index ;
  - Nombre de valeurs distinctes par attribut, dans les cas où un index ou un cluster existe : permet d'estimer le nombre moyen d'enregistrements qui satisfont une sélection (*sélectivité* ou *cardinal de sélection* d'un attribut A).
    - A est clé : *s = 1*
    - A n'est pas clé : *s = Card(R)/Nombre de valeurs de A*.

III. Optimisation par estimation de coûts

- *Sélectivité de la jointure* : $sj = Card((R \bowtie_{Cond} S))/Card(R \times S)$
  - *sj* = 1 si pas de condition de jointure
  - *sj* = 0 si aucun tuple ne vérifie *Cond.*
- *Cas de l'équi-jointure* : (*Condition du type R.A = S.B*) Cas particuliers :
  1. A est clé de R : $Card(R \bowtie S) \leq Card(S)$
     - $sj \leq Card(S)/Card(R) \times Card(S)$
     - $sj \leq 1/Card(R)$
  2. De même, si B est clé de S : $sj \leq 1/Card(S)$.
- D'où la taille estimée de $R \bowtie S$: *sj* × *Card(R)* × *Card(S)*.

Exemples d'estimation de l'équi-jointure

- br : nombre de blocs de R,
- bs : nombre de blocs de S
- k : facteur de blocage de la relation résultat

1. *Jointure par boucles imbriquées*
   - R dans la boucle extérieure ; *sj* donné ; 2 buffers
   - Coût estimé : *br + (br * bs) + ((sj * Cardinal(R) * Cardinal(S))/k)*
   - Dernier facteur : Coût de l'écriture du résultat.

Exemples d'estimation de l'équi-jointure

2. *Jointure par tri-fusion*
   - Si relations déjà triées : *Coût* = *br* + *bs*
   - Sinon, *Coût* = $br + bs + l * (br * \log_2 br + bs * \log_2 bs))$.
     - $(l * b * \log_2 b)$: Approximation du tri
     - b : nombre de blocs
     - l : facteur constant.

IV. Optimisation sémantique

- *Contrainte d'intégrité* Formule logique CI
- *Expression de sélection* E
- *Pas d'accès à la base si* $\neg(E \wedge CI)$
- *Exemple* :
  - CI : Tous les vols long courrier partent de Paris-Roissy
  - E : Vols long courrier partant de Paris-Orly ?
- Méthode non implantée (démonstration automatique)

Optimisation de requêtes : SGBD Sybase

- Génération de plans avec/sans exécution de requête
- Exemple : Visualisation d'un plan sans exécution

  SET SHOWPLAN ON
  SET NOEXEC ON
  Requête dont on veut examiner le plan.
- Plans d'exécution et procédures stockées :
  - Exécution avec *WITH RECOMPILE* : MàJ du plan stocké
  - Création avec *WITH RECOMPILE* : Génération systématique
- *Mise à jour* des informations sur la distribution des valeurs des clés des index :

  UPDATE STATISTICS NomDeRelation [NomIndex]

Optimisation de requêtes : SGBD Oracle

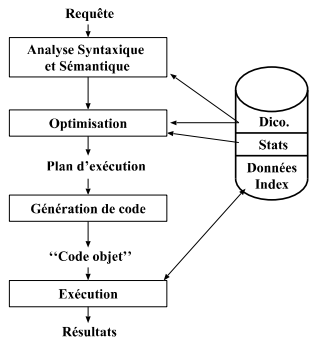
- Commande *EXPLAIN PLAN* : obtenir le plan d'exécution
- Plan rangé dans *PLAN_TABLE* ou dans une relation créée préalablement à l'exécution de *EXPLAIN PLAN* (clause *INTO* de *EXPLAIN PLAN*)

  EXPLAIN PLAN
  [*SET STATEMENT_ID* = identification_plan]
  [*INTO* [nom_utilisateur.]nom_de_relation] *FOR* requête_SQL
- ANALYZE {index| table | cluster} {COMPUTE | ESTIMATE} STATISTICS

Traitement des requêtes : Plan

1. Transformation d'arbres algériques
2. Transformation de graphes de requêtes
3. Optimisation à base de coûts
4. Optimisation Sémantique
5. Processus Général d'Optimisation

V. Optimisation : Processus général



Traitement des requêtes : Processus général

1. Normalisation des prédicats
2. Analyse sémantique (Requête = Graphe connexe)
3. Simplification des formules logiques
4. Mise sous forme d'arbre relationnel

1/4) Normalisation des prédicats

- *Forme normale conjonctive* : conjonction de disjonctions
  $(p_1 \vee p_2 \vee \cdots \vee p_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_m)$
- *Forme normale disjonctive* : disjonction de conjonctions
  $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \vee \cdots \vee (q_1 \wedge \cdots \wedge q_m)$
- *Formules sans quantificateurs* :
  - Commutativité, Associativité de la conjonction et de la disjonction
  - Distributivité $(\vee, \wedge)$, $(\wedge, \vee)$
  - $\neg(p_1 \wedge p_2) \longleftrightarrow \neg p_1 \vee \neg p_2$
  - $\neg(p_1 \vee p_2) \longleftrightarrow \neg p_1 \wedge \neg p_2$
  - $\neg(\neg p) \longleftrightarrow p$
- *Formules avec quantificateurs* : mise sous forme *prenex*

- *Forme disjonctive* :
  - Requête traitée comme une union de sous-requêtes
  - Risque de calcul redondant
- *Forme conjonctive* : généralement plus de "ET" que de "OU"
- *Exemple* :

  select libelle from produit p, stock s
  where p.prod# = s.prod# and s.adr = "Nancy"
  and (s.qte = 1000 or s.qte=200)

  - *Forme disjonctive* :

    (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 1000)
    $\vee$ (p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ s.qte = 2000)
  - *Forme conjonctive* :

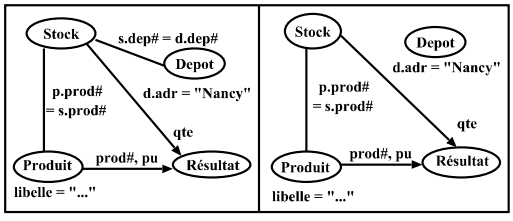
    p.prod# = s.prod# $\wedge$ s.adr = "Nancy" $\wedge$ (s.qte = 1000 $\vee$ s.qte=2000)

2/4) Analyse

- *Dictionnaire* : Relations, attributs connus
- *Typage des expressions*
- *Correction sémantique* : pas de sous-requête isolée
  - *Calcul relationnel* : impossible à determiner
  - *Requêtes sans $\vee$ ni $\neg$* : graphe connexe
- *Graphe de requête* :
  - *Nœud* : résultat ou opérande
  - *Arcs entre nœuds non résultats* : Jointure
  - *Arcs d'extrêmité nœud résultat* : Projection
  - *Nœud non résultat* peut être labellé par une expression de sélection ou une auto-jointure

- *Exemple* :

  select p.prod#, p.pu, s.qte
  from produit p, stock s, depot d
  where p. prod# = s.prod# and s.dep# = d.dep#
  and s.qte >= 0 and d.adr = "Nancy" and libelle = "..."



3/4) Simplification d'expressions logiques

- *Cas de l'utilisation de vues*

```
create view V
as      select * from produit
        where pu > 100.0 and pu < 200.0
```

- *Requête* :

```
select * from V
where pu < 200.0
```

- *Après ré-écriture* :

```
select * from produit
where pu > 100.0
   and pu < 200.0 and pu < 200.0
```


3/4) Simplification d'expressions logiques (suite)

- *Elimination de la redondance* : règles d'idempotence
  - $p \wedge p \longleftrightarrow p; p \vee p \longleftrightarrow p$
  - $p \wedge Vrai \longleftrightarrow p; p \wedge Faux \longleftrightarrow Faux$
  - $p \vee Vrai \longleftrightarrow Vrai; p \vee Faux \longleftrightarrow p$
  - $p \wedge \neg p \longleftrightarrow Faux; p \vee \neg p \longleftrightarrow Vrai$
  - $p_1 \wedge (p_1 \vee p_2) \longleftrightarrow p_1$
  - $p_1 \vee (p_1 \wedge p_2) \longleftrightarrow p_1$

3/4) Simplification d'expressions logiques (suite et fin)

- *Exemple* :

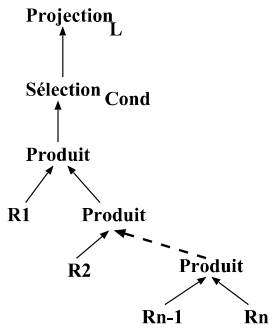
```
select libelle from produit
where NOT libelle = "K7Cr"
      and (libelle = "K7Cr" or pu = 20.0)
      and NOT pu = 20.0
      and prod# = 4
```

se simplifie en :

```
select libelle from produit
where prod# = 4
```


4/4) Mise sous forme d'arbre algébrique

SELECT L
FROM R1, R2, ..., Rn
WHERE Cond



Traitement des requêtes sous Oracle

- Etapes "classiques" dans le processus de traitement des requêtes :
  1. Analyse,
  2. Optimisation,
  3. Génération de plans d'exécution,
  4. Exécution.

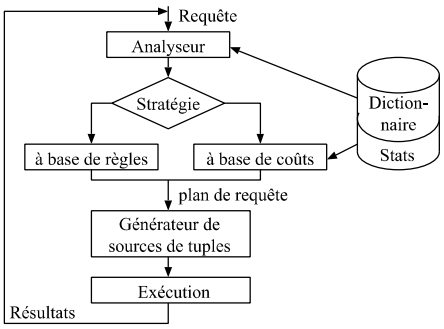
Processus de Traitement des requêtes sous Oracle

- Deux stratégies d'optimisation :
  1. A base de règles (*Rule Based Optimization*, *RBO*) : $\simeq$ optimisation d'arbres algébriques
  2. A base de coûts : *Cost Based Optimization* (*CBO*).
- Choix de la stratégie : automatique ou imposé pour une requête, une session ou une instance Oracle,
- Influencer la génération de plans : Directives d'optimisation dans les requêtes,
- Ré-utiliser des plans : cf. *outlines* (non traités ici).

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes (p. 59)
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

1/5. Architecture de l'optimiseur et plans d'exécution



1/5. Architecture de l'optimiseur et plans d'exécution

1. *Analyseur* : décomposition de la requêtes en un ensemble de composants emboîtés ou reliés (*blocs de la requête*)
   - Vue, requête imbriquée $\longrightarrow$ blocs distincts de celui de la requête (ou pas)
2. *Générateur de sources de tuples* : Engendrer un plan d'exécution étant donné le plan optimal ($\simeq$ arbre algébrique) rendu par l'optimiseur :
   - Noeuds opérations "internes" à Oracle ($\neq$ opérateurs algébriques) :
     - Provenance = mode d'accès aux tuples et type d'algorithme de jointure choisis
     - Exemples : tri (*sort*), fusion (*merge*)
   - Source de tuples $\simeq$ feuille d'un arbre ou relation résultant de l'évaluation d'un sous-arbre.

1/5. Exemple de plan d'exécution

- Hypothèse : Aucun index
- Requête :

  select p.libelle, s.qte *from* produit p, stock s
  where p.prod# = s.prod#;

- Plan d'exécution :

1/5. Plans d'exécution

- Demande de génération : *explain plan . . . for* instruction SQL
- Plan engendré :
  - par défaut, dans *plan_table*
  - ou dans une table de même schéma (*explain plan . . . into . . .*)
- *plan_table* créée par le script *utlxplan.sql* (généralement sous *$OraHome\rdbms\admin*).
- Examen du plan :
  1. *select . . . from plan_table where . . .*
  2. utilitaire *utlxpls* : traitements en série
  3. utilitaire *utlxplp* : exécutions parallèles

1/5. Plans d'exécution : Des colonnes de *plan_table*

- *id :* numéro d'étape du plan ;
- *parent_id :* numéro de l'étape qui utilise les résultats de l'étape *id* ;
- *position :* rang d'une étape parmi les étapes "filles" d'une même étape "mère" ;
- *cardinality :* nombre estimé de tuples accédés par l'opération ;
- *operation :* nom de l'opération interne réalisée dans l'étape ;
- *object_name :* objet concerné par l'opération ;
- *cost :* coût estimé de l'opération (si optimisation basée sur les coûts) ;
- *options :* "variante" d'opération utilisée pour exécuter *operation*, etc.

1/5. Plans d'exécution : Exemple

1/5. Plans d'exécution : Exemple

```
Id P_id    Operation       Objet    Options
-- ---- ----------------- -------- ---------
 0       SELECT STATEMENT
 1    0  MERGE JOIN
 2    1  SORT                        JOIN
 3    2  TABLE ACCESS       STOCK    FULL
 4    1  SORT                        JOIN
 5    4  TABLE ACCESS       PRODUIT  FULL
6 ligne(s) sélectionnée(s).
```

- cf. figure exemple p. 52

1/5. Plans d'exécution

- *plan_table* : représentation tabulaire des plans d'exécution et des choix de l'optimiseur :
  - mode de parcours des objets
  - type d'algorithme de jointure, etc.
- Choix explicités dans *plan_table.operation* et *plan_table.options*

Opération	**Options**	**Commentaires**
table access	*full, by rowid, hash, ...*	Type d'accès à une table.
sort	*unique*	Tri en éliminant les doublons.
	join	Tri avant jointure par fusion.
	order by	Tri requis par la requête.
nested loops		Jointure par boucles imbriquées.
	outer	Idem pour une jointure externe.
merge join		Jointure par tri-fusion.
	outer	Idem pour une jointure externe.
	semi	Idem pour une semi-jointure.
view		Interrogation d'une vue.

2/5. Les étapes de l'optimisation

1.
2.
3.
4.
5.
6.
7.

2.1/5- Simplification et évaluation d'expressions

- cf. Traitement des requêtes dans les SGBDR
- *Simplification d'expressions logiques* : éliminer des sous-expressions communes par la mise d'expressions composées sous forme normale conjonctive $[(p_1 \vee \cdots \vee p_n) \wedge \cdots \wedge (q_1 \vee \cdots \vee q_m)]$.
- *Transformation d'expressions logiques* : règles de réécriture syntaxique et équivalences logiques.

1. Expression de constante évaluable statiquement : la remplacer par sa valeur (Exemple : *"X > 550/10"* $\longrightarrow$ *"X > 55"*).
2. *X like 'ABCD'* $\longrightarrow$ *X = 'ABCD'*, si X est de type chaîne de caractères à longueur variable.

2.1/5- Simplification et évaluation d'expressions

3. *"x between v1 and v2"* $\longrightarrow$ *"x >= v1 and x <= v2"*.
4. *"x in (V_1, ..., V_n)"* $\longrightarrow$ (*"x = V_1 or ... or x = V_n"*).
5. *"x > all (V_1, ..., V_n)"* $\longrightarrow$ (*"x > V_1 and ... and x > V_n"*)
6. *"x > all (select y ... where condition)"* $\longrightarrow$ *"not exists (select y ... where condition and x <= y)"*.
7. Idem pour des expressions comprenant *any* ou *some* :
   Exemple1 : *"x > any (y, z)"* $\longrightarrow$ *"x > y or x > z"*
   Exemple2 : *"x > any (select y ... where Condition)"* $\longrightarrow$ *"exists(select y ... where Condition and x > y)"*.

2.1/5- Simplification et évaluation d'expressions

8. Eliminer la négation dans les expressions de sélection, si possible
   Exemple1 : *"not x = (select ...)"* $\longrightarrow$ *"x <> (select ...)"*
   Exemple2 : *"not (x < 123 or y is null)"* $\longrightarrow$ *"x >= 123 or y is not null"*.

2.2/5- Réécriture de requêtes complexes

- Transformations décidées en fonction de la complexité de la requête, de la présence de vue ou d'index, etc.
- Types de transformation :
  1. Requête simple $\longrightarrow$ Requête composée,
  2. Désimbriquer des sous-requêtes,
  3. Incorporer des définitions de vues dans une requête
  4. Introduire des prédicats de la requête dans une vue.

2.2/5- Requête simple $\longrightarrow$ Requête composée

select ... where A = Constante1 or B = Constante2

réécrite en :

(select ... where A = Constante1)
union (select ... where B = Constante2)

Si index sur A et B.

2.2/5- Désimbriquer des sous-requêtes

- Traiter la requête sous sa forme initiale
- ou désimbriquer la sous-requête en introduisant des jointures dans la requête initiale.
- Exemple : Si contrainte de clé primaire ou d'unicité sur *R2.x*

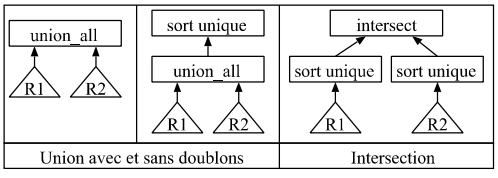
  select ... from R1
  where x in (select x from R2 where condition) ;

  réécrite en :

  select ... from R1, R2
  where condition and R1.x = R2.x.

2.2/5- Cas ded Requêtes composées

- Cas requête avec opérateurs ensemblistes ($\cup, \cap, \setminus$) :
  1. Un plan d'exécution pour chaque membre de l'opérateur
  2. Composition du résultat avec l'opérateur ensembliste
- Exemples : (triangle $\simeq$ sous-arbre)



2.3/5- Traitement des requêtes avec vues

(a) Intégrer la définition de la vue à la requête

(b) Etendre la définition de la vue par des prédicats de la requête.

2.3.a/5- Intégration de la vue à la requête

- Faisable sur les *vues* dites *intégrables* ou *fusionnables* ne comportant pas, dans la liste de projection, :
  - d'opérations ensemblistes
  - de clause *connect*,
  - de pseudo-colonne *rownum*,
  - de fonction d'agrégation (*min, avg, main, max, sum*)
- Exemple :

  "select x1 from V where Condition2"

  et *"create view V as (select x1, . . .from R where Condition1)"*

  $\longrightarrow$ *"select x1 from R where Condition1 and Condition2"*

2.3.b/5- Intégration de prédicats de la requête à la vue

- Quand échec de l'intégration de la vue à la requête

select x1, x2 *from* V
where xk = 1234
avec V définie par :
create view V(x1, ..., xn) *as*
select x1, ..., xn *from R1*
union (*select* x1, ..., xn *from R2)*

est ré-écrit en

select x1, x2
from select x1, ..., xn *from R1*
where xk = 1234
union (select x1, x2
from select x1, ..., xn *from R1*
where xk = 1234)

2/5. Les étapes de l'optimisation : Plan

2.4/5- Stratégie et but de l'optimisation

- Stratégie (rappel) : Base de coûts/de règles
- But de l'optimisation : globale ou temps de réponse
- *Optimisation globale* : minimiser la quantité totale de ressources nécessaires au traitement de tous les tuples concernés par une instruction
  - But par défaut
  - Adaptée pour les traitements en mode différé
- *Optimisation du temps de réponse* : minimiser les ressources pour accéder au premier tuple concerné par l'instruction.
  - Adaptée pour les applications conversationnelles

2.4/5- Stratégie et but de l'optimisation : Paramétrisation

1. Niveau instance Oracle : paramètre d'initialisation *optimizer_mode*
2. Niveau session : argument *optimizer_goal* de *alter session*
3. Niveau instruction : directives d'optimisation *optimizer_mode* et *optimizer_goal*

- Si paramères utilisés comme directive d'optimisation : portée = l'instruction qui les contient.
- Si *optimizer_goal* spécifié pendant une session : substitution à la valeur de *optimizer_mode* pour la durée de la session.

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

.

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

- *choose, rule, all_rows, first_rows*

1. *choose* et présence de statistiques sur au moins une table
   - Stratégie : à base de coûts
   - But : optimisation globale
2. *choose* et absence de statistiques : stratégie à base de règles.
3. *rule :*
   - = Valeur par défaut
   - Stratégie : à base de règles (avec ou sans statistiques)
   - But : optimisation globale

2.4/5- Paramétrisation : Valeurs de *optimizer_mode* et *optimizer_goal*

4. *all_rows :* même en l'absence de statistiques,
   - Stratégie : à base de coûts
   - But : optimisation globale
5. *first_rows :*
   - Stratégie : à base de règles
   - But : minimisation du temps de réponse

2/5. Les étapes de l'optimisation : Plan

1. Simplification et Evaluation d'expressions (p. 60)
2. Réécriture de requêtes complexes (p. 63)
3. Réécriture de requêtes avec vues (p. 67)
4. Choix stratégie d'optimisation ? (p. 71)
5. Traitement des jointures (p. 77)

2.5/5- Traitement des jointures

- Combinaison des facteurs :
  - **(a)** Type d'algorithme de jointure
  - **(b)** Ordre des jointures
  - **(c)** Chemins d'accès
    - Dans l'optimisation à base de coûts : page 86
    - Dans l'optimisation à base de règles : page 105

2.5.a/5- Choix de l'algorithme de jointure

- Méthodes ou algorithmes de jointure :
  1. Boucles imbriquées (*nested loops*),
  2. Tri-fusion (*sort-merge*),
  3. Hachage (*hash join*),
  4. Groupement (*cluster join*).
- Critères de choix : cf. page 79 à page 81

2.5.a/5- Choix de l'algorithme de jointure

1. Boucles imbriquées :
   - Utilisable par les deux stratégies d'optimisation
   - Choisi si taille du résultat > 10000 tuples
   - Coût = $C\_R\_ext + (C\_R\_int * Card(R\_ext))$
   - *C_R_ext* : coût de l'accès à la relation externe
   - *C_R_int* : coût de l'accès à la relation interne

2.5.a/5- Choix de l'algorithme de jointure

2. Tri fusion :
   - Utilisable par les deux stratégies d'optimisation
   - Le plus efficace si
     (i) optimisation à base de règles
     (ii) résultat de grande taille
   - Coût : Coûts d'accès aux relations [+ Coûts de leur tri]

2.5.a/5- Choix de l'algorithme de jointure

3. Hachage :
   - Tables partitionnées
   - Non utilisable dans une optimisation à base de règles
   - Le plus efficace si résultat de grande taille
   - Coût : $C\_R2 + (C\_R1 * Nb\_Partition\_R2)$,
     - *C_R1, C_R2* : coûts des accès aux relations
     - *Nb_Partition_R2* : nombre de partitions de hachage de *R2*

4. Groupement : utilisable dans une équi-jointure :
   - de tables d'un même *cluster*
   - sur la clé du cluster

2.5.b/5- Ordre des jointures

- Quelle que soit la stratégie d'optimisation :
  1. En tête de l'ordre : Table dont la jointure rend un seul tuple
  2. Poursuite de la recherche de l'ordre (variable selon les stratégies)
- cf. détails page 83 à page 84

2.5.b/5- Ordre des jointures et *Optimisation à base de coûts*

- Engendrer et évaluer le coût des plans en prenant en compte :
  - les chemins d'accès disponibles
  - l'ordre des jointures
  - le type d'algorithme de jointure.

2.5.b/5- Ordre des jointures et *Optimisation à base de règles*

- Engendrer un ensemble de plans d'exécution des jointures
- En choisir un en se laissant guider par un objectif :
  1. de maximisation du nombre de jointures par boucles imbriquées
  2. où la table interne est accessible *via* un index (*index scan*).

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

3/5. Optimisation à base de coûts

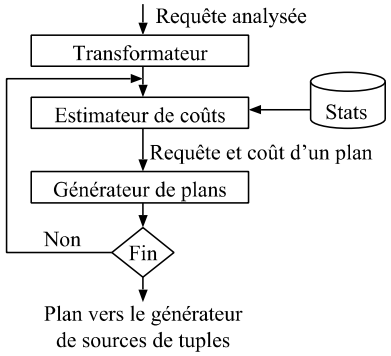
- Conditions de mise en œuvre :
  1. Disponibilité de statistiques
  2. Paramètres d'initialisation positionnés :
     (a) *optimiser_mode = choose, first_rows* ou *all_rows*
     (b) *optimiser_features_enable* = . . .
     (c) *compatible* = . . .
- *Note* : Consulter les paramètres en vigueur dans *v$parameter*
- Exemple :

```
select name, value from v$parameter
where upper(name) like upper('%optimizer%')
```


3/5. Optimisation à base de coûts

- Stratégie généralement requise pour les instructions qui utilisent :
  - des tables partitionnées et/ou organisées en index,
  - des index basés sur des fonctions.
- Processus d'optimisation :
  1. Engendrer un ensemble de plans : se fonder sur
     - les chemins d'accès existants
     - les éventuelles directives d'optimisation
  2. Estimer le coût de chaque plan : utiliser les statistiques sur
     - la distribution des données
     - les caractéristiques de stockage des objets (tables, index, partitions) impliqués
  3. Choisir le plan le moins coûteux.

3/5. Optimisation à base de coûts : Architecture



Collecte de statistiques

- Calculer/estimer (*compute/estimate statistics*) des statistiques sur les espaces ;
- *Calcul* sur la totalité de l'instance d'un objet (table, index, partition ou cluster) ;
- *Estimation* : sur un échantillon spécifié en
  - nombre de lignes (1064 par défaut)
  - ou en pourcentage de la taille de l'objet.
- Clauses de portée de la commande d'analyse (*analyze*) :
  - *for table* : informations sur la table,
  - *for columns* : une liste de ses colonnes

Collecte de statistiques : ExempleS

Collecte de statistiques : (1/2) Sur des tables

Nom de colonne	Nature de l'information
NUM_ROWS	Nombre de tuples
BLOCKS	Nombre de blocs de données
EMPTY_BLOCKS	Nombre de blocs qui n'ont jamais été occupés
AVG_SPACE	Taux moyen d'espace libre par bloc
CHAIN_COUNT	Nombre de tuples ayant subi une migration
AVG_ROW_LEN	Taille moyenne d'un tuple

Collecte de statistiques : (2/2) Sur des Index

Nom de colonne	Nature de l'information
BLEVEL	Nombre de niveaux d'index
LEAF_BLOCKS	Nombre de blocs feuilles
DISTINCT_KEYS	Nombre de valeurs distinctes de la clé de l'index
AVG_LEAF_BLOCKS_PER_KEY	Nombre moyen de blocs feuilles par valeur de la clé de l'index
AVG_DATA_BLOCKS_PER_KEY	Idem pour les blocs de données
CLUSTERING_Factor	Facteur de groupement des clés de l'index

Statistiques et tables du dictionnaire

- Stats sur tables : Voir *user_tables, all_tables* et *dba_tables*
- Sur colonnes : {*user_* | *all_* | *dba_*}*tab_columns*
- Sur clusters : (en partie) vues {*all_* | *user_* | *dba_*}*clusters*.

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation de coûts (p. 100)
  4. Génération de plans (p. 103)

3.1/5. Optimisation à base de coûts : Chemins d'accès

- (Terminologie d'Oracle) Chemin d'accès (*access path*) :
  1. Mode d'accès
  2. Mode de parcours éventuel des tuples d'un objet.
- Quelques types de chemins d'accès :
  - *Full table scans* : parcours de toute la table ;
  - *Sample table scans* : parcours d'un échantillon de tuples ;
  - *Table access by RowId* : accès à l'aide d'un identifiant de tuple ;
  - *Index scans* : mode de parcours d'index avec des variantes dont :
    - *unique scan* : lorsqu'un identifiant de tuple est rendu ;
    - *range scan* : étant donné un intervalle de valeurs de la clé ;
    - *full scan* : parcours de la totalité d'un index ;
    - *fast full scan* : l'index suffit pour satisfaire la requête.

3.1/5. Optimisation à base de coûts : Chemins d'accès

- Des facteurs influant sur le *choix d'un chemin d'accès* :
  - Liste des chemins disponibles,
  - Estimation du gain potentiel en utilisant tout ou partie des chemins,
  - *Sélectivité* des attributs et de la jointure,
  - Nombre de valeurs de chaque colonne d'une table (dans *user_tab_columns_num_distinct*),
  - Nombre de tuples d'une table (*user_tab_columns_num_rows*),
  - Statistiques disponibles (*user_tab_col_statistics* et *user_tab_columns*).

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation de coûts (p. 100)
  4. Génération de plans (p. 103)

3.2/5- Optimisation à base de coûts : Transformation de requêtes

- Objectif du transformateur de requêtes : est-il avantageux d'opérer des réécritures de requêtes ?
- Evaluation de l'opportunité
  1. d'intégrer les définitions de vues à la requête,
  2. de désimbriquer des requêtes,
  3. d'utiliser des *vues matérialisées*.
- **Note** : *Vue matérialisée* = Vue instanciée ($\simeq$ Table)

3.2/5- Optimisation à base de coûts : Transformation de requêtes

1. Intégration des vues
   - Cas vue *fusionnable* avec l'instruction : Génération d'un seul plan
   - Cas vue *non fusionnable* : Génération d'un sous-plan pour la vue
2. Sous-requêtes :
   - Certaines peuvent être désimbriquées et d'autres pas
   - Celles non désimbriquées $\longrightarrow$ Plans séparés + Ordre entre les plans.
3. Vues matérialisées : si une partie de la requête correspond à la définition d'une vue matérialisée, la remplacer par le nom de la vue.

3.3/5- Estimation des coûts

- Facteurs :
  1. *Sélectivité*,
  2. Cardinal des ensembles
  3. Coût des ressources utilisées : principale unité = nombre d'entrées-sorties physiques (blocs de données et d'index).
- *Sélectivité*, sans disponibilité de statistiques :
  - Utilisation d'une valeur interne est utilisée
  - Exemple : Sélectivité d'une d'une expression du type *"attribut = valeur"* est estimée meilleure que celle d'une expression du type *"attribut > valeur"*.

3.3/5- Optimisation à base de coûts : Estimations

- *Sélectivité* et présence de statistiques :
  - Calcul de la sélectivité
  - Exemple : Sélectivité d'une expression *"attribut = valeur"* = (1/nombre de valeurs distinctes de l'attribut).
- *Cardinal des ensembles* :
  - Tables, Vues,
  - Résultats de jointures et de requêtes avec *group by*
  - Calcul d'un *cardinal effectif* en utilisant la sélectivité
  - Si défaut de statistiques : Estimation d'un *cardinal de base* en fonction du nombre d'*extents* occupés.

3/5. Optimisation à base de coûts

- La suite :
  1. Chemins d'accès (p. 95)
  2. Transformation de requêtes (p. 98)
  3. Estimation des coûts (p. 100)
  4. Génération de plans (p. 103)

3.4/5- Optimisation à base de coûts : Génération de plans

- Un sous-plan :
  - pour chaque sous-requête désimbriquée
  - pour chaque vue non intégrée à la requête.
- Génération et optimisation du plan global de bas en haut (du bloc de requête le plus interne vers la requête)
- Arrêt lorsque l'évaluateur estime que le coût du dernier plan engendré est "acceptable"
  (i.e. rapport gain escompté-coût examen exhaustif)

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

4/5. Optimisation à base de règles

- Choix du plan d'exécution en fonction
  1. des chemins d'accès existants
  2. d'un poids associé à chaque type de chemins,
- Meilleur chemin : celui de plus faible poids
- Quinze types de chemins disponibles
- Leur choix dépend :
  1. de la forme de la requête
  2. de la structure des objets concernés

4/5. Optimisation à base de règles : Exemples de choix de chemins d'accès

Chemin d'accès	**Conditions et forme de la requête**
Single Row by RowId	*where rowid = '…'*
Single Row by cluster join	*where* R1.A = R2.A *and* R1.B = valeur et R1, R2 groupées (*cluster*) sur A et B est clé primaire de R1.
single row by unique or primary key	La clause *where* porte sur tous les attributs d'une clé primaire ou d'un index *unique*.
bounded range search or index columns	*where* A = expression ou *where* A > [=] expression1 *and* A > [=] expression2.

Traitement des requêtes sous Oracle : Plan

1. Architecture du compilateur de requêtes et structure de stockage des plans d'exécution (p. 50)
2. Processus et principales étapes de traitement des requêtes
3. Optimisation à base de coûts (p. 86)
4. Optimisation à base de règles (p. 105)
5. Directives d'optimisation (p. 108)

5/5. Directives d'optimisation

- Pour forcer certains choix de l'optimiseur
- Sous la forme : "…/* + *directives* */ …" dans une instruction
- Exemples :

5/5. Directives d'optimisation et leurs objets

1. Méthode et but de l'optimiseur : *all_rows*, *first_rows*, *choose*, *rule*.
   - *all_rows*, *first_rows*, *choose* $\longrightarrow$ invoquer l'optimiseur à base de coûts
2. Méthode d'accès : *full, rowid, index, cluster, index_join, rewrite,* etc.
3. Ordre des jointures : *ordered.*
4. Algorithme de jointure :
   - *use_nl* : boucles imbriquées,
   - *use_merge* : tri-fusion),
   - *use_hash* : fonction de hachage
   - *use_nl*, *use_merge* recommandés conjointement avec *ordered*

5/5. Directives d'optimisation et leurs objets

5. Fusion (ou pas) de vues :
   - *merge, no_merge*,
   - *push_pred*, *no_push_pred.*
6. Désimbrication (ou non) de sous-requêtes : *unnest, no_unnest*.
7. Stratégie de gestion en mémoire cache : *cache, nocache.*
8. Exécution parallèle d'instructions : *parallel, noparallel, parallel_index* et *noparallel_index*.
9. etc.

Traitement des requêtes : Conclusion

- Optimisation algébrique : Fondement = Propriété des opérateurs relationnels
- Algèbre $\simeq$ Langage de base (”Assembleur”) des SGBDR
- Optimisation à base de coûts : disponibilité de statistiques
- Concepts et techniques à connaître pour :
  - une vision non naïve des SGBD
  - amélioration des performances
  - administration des applications (et des systèmes)

Oracle et SQL*PLUS

Nacer Boudjlida
Université de Lorraine, FST,
Département informatique

Introduction à Oracle et à SQL*PLUS

1. A propos du SGBD ORACLE
2. L'interface SQL-Plus

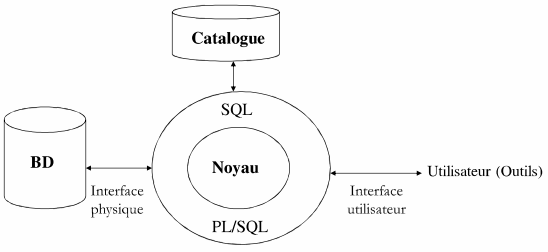
1. Introduction au SGBD ORACLE

- Edité par *Oracle Corporation* (www.oracle.com) depuis 1981 : 60% du marché
- Version actuelle : 11g (mi 2007)
- 10g = Version installée à l'ESIAL et au Département)
- Large éventail d'outils « autour »:
  - Serveurs de données et utilitaires (sql*plus, import/export loader…)
  - Environnements de développement (Web, java, pré-compilateurs, …)
  - Logiciels d'aide à la décision (data warehouse, Oracle Discoverer…)
  - Progiciels client serveurs et Internet (Oracle Application & Oracle e-business suite)
  - …

Introduction au SGBD ORACLE

a) Architecture « externe »
b) Quelques concepts
c) L'interface SQL*Plus: Concepts et mise en œuvre

a) Architecture d'Oracle data server



SQL*plus : interface pour soumettre des requêtes SQL ou PL/SQL à Oracle

b) Quelques concepts Oracle

- Serveur de données Oracle = une base + une ***instance***
- ***Instance :***
  - {Processus d'arrière-plan}
  - Zone globale système (SGA) : tampons de données/journal partagés
- Dictionnaire de données : {tables} et {vues}
  - Créé dans chaque base, dans l'espace SYSTEM
  - Appartient à sys, non modifiable (sauf sys.aud$)

Dictionnaire de données (catalogue)

- Vues de nom **USER_xxx**
  - Informations sur les objets dont l'utilisateur est propriétaire
- Vues de nom **ALL_xxx**
  - Informations sur les objets auxquels a accès l'utilisateur
- Vues de nom **DBA_xxx**
  - Informations sur les objets de tous les utilisateurs
- Exemples :
  - USER-TABLES: « mes » tables
  - ALL_TABLES: « mes » tables + celles auxquelles j'ai accès
  - DBA_TABLES : Toutes les tables

Dictionnaire de données (catalogue)

- Quelques vues du dictionnaire :
  - USER_OBJECTS, ALL_OBJECTS, DBA_OBJECTS
  - USER_TABLES, ALL_TABLES, DBA_TABLES
  - USER_TAB_COLUMNS
  - USER_TAB_COMMENTS
  - USER_VIEWS
  - …
- Vue **DICTIONARY** (ou **DICT**) : liste des tables et des vues du dictionnaire

Introduction à Oracle et à SQL*PLUS

1. Introduction au SGBD ORACLE
2. L'interface SQL-Plus

SQPL*Plus

- SQL : basé sur l'algèbre relationnelle et le calcul des prédicats à variables tuples
- SQL est un standard ANSI/ISO depuis 1986
  - SQL-86 (version préliminaire)
  - SQL-89 ou SQL1(niveau minimal supporté)
  - SQL-92 ou SQL2 (standard courant bien supporté)
  - SQL-99 ou SQL3 (extensions majeures de SQL2, peu supporté)
- SQL-Oracle proche de la norme
  - Oracle 8.i : Compatible SQL2
  - Oracle 9.i : Compatible SQL3

Généralités sur SQL (2/2)

1. **LDD** (**L**angage de **D**éfinition de **D**onnées) pour créer, modifier et supprimer des définitions d'objets (schema object)
   - *create, alter, drop {table | index | procedure | trigger…}*
2. **LMD** (**L**angage de **M**anipulation de **D**onnées) : ajouter, supprimer, modifier, rechercher, valider/annuler

   insert, delete, update, select, commit/rollback
3. **LCD** (**L**angage de **C**ontrôle de **D**onnées) pour gérer les protections d'accès.
   - *grant, revoke,*

SQL*Plus : interface SQL et PL/SQL

- SQL*Plus : interface en ligne de commande entre l'utilisateur et le SGBD Oracle.
- Mise en œuvre de SQL*Plus
  1. Lancer, Quitter
  2. Connexion, Déconnexion
  3. Exécuter des commandes SQL ou des blocs PL/SQL
  4. Editer
  5. Formater les résultats
  6. Divers

2.1. Lancer, Quitter SQL*Plus

- sqlplus *login/mot_de_passe*
  - Lancement en mode ligne de commande
  - Connexion à une BD par défaut
- sqlplus *login/mot_de_passe@chaîne2connexion*
  - Connexion au « service » (ou à la base) spécifié

Lancer, Quitter SQL*Plus

- Démarrer → Programmes → ... → SQL Plus
  - Lancement en mode menus (windows)
- exit *ou* quit
  - quitter SQL*Plus

2.2. SQL*Plus : Connexion, Déconnexion

- Disconnect (ou disc)
  - Déconnexion de la base sans quitter SQL*Plus
  - On ne peut plus soumettre de commande (PL/)SQL
- connect login[/mot_de_passe][@chaîne]
  - Connexion sous le nom `login` à la base cible ou à la base par défaut
- password
  - Changement de mot de passe

SQL*Plus : interface SQL et PL/SQL

- Mise en œuvre de SQL*Plus
  1. Lancer, Quitter
  2. Connexion, Déconnexion
  3. Exécuter des commandes SQL ou des blocs PL/SQL
  4. Editer
  5. Formater les résultats
  6. Divers

2.3. SQL*Plus : Exécuter

- **run** ou **/**: Exécuter les commandes contenues dans le tampon
- **start** *nom_fichier ou @nom_fichier*:
  Exécuter le contenu du fichier
- **spool** *nom_fichier*
  Rediriger les sorties vers *nom_fichier*
- **spool off** : Retour à l'affichage sur écran
- **spool out** : Fermer le fichier spool et l'imprimer

2.4. SQL*Plus : Editer

1. DEFINE _EDITOR=*nom_éditeur*
2. EDIT *fichier[.extension]*
   Ouvrir le fichier sous l'éditeur défini
3. SAVE *nom_fichier*[CREATE|REPLACE|APPEND]
   Sauvegarder le tampon dans le fichier
   - CREATE : créer un nouveau fichier
   - REPLACE : remplacer l'ancien
   - APPEND : étendre l'ancien

SQL*Plus : Editer

- GET *fichier* [LIST][NOLIST]
  - Charger le fichier dans le tampon avec (LIST) ou sans (NOLIST) affichage à l'écran (LIST par défaut)
  - Chargement sans exécution (cf. exécution par run ou /)

SQL*Plus : interface SQL et PL/SQL

- Mise en œuvre de SQL*Plus
  1. Lancer, Quitter
  2. Connexion, Déconnexion
  3. Exécuter des commandes SQL ou des blocs PL/SQL
  4. Editer des commandes, des fonctions, etc.
  5. Formater les résultats
  6. Divers

2.5. SQL*Plus : Formater, Afficher

- SET HEADING {ON/OFF}
  - Affichage ou non des noms de colonnes
- SET ECHO {OFF/ON}
  - Affichage des commandes d'un script à l'éxécution

SQL*Plus : Formater, Afficher

- SET LINESIZE n
  - Taille d'une ligne en nombre de caractères
- SET PAGESIZE n
  - Nombre de lignes par page
- SET TERMOUT {ON|OFF}
  - Affichage ou non des résultats

SQL*Plus : Formater, Afficher

- Largeur de colonnes à l'affichage :

```
sqlplus > column col-1 format A10
sqlplus > column col-2 format 9999
sqlplus > select col-1, col-2, col-3 from R;
```

  - col-1 : 10 caractères
  - col-2 : 4 chiffres
  - col-3 : comme définie dans *create table R*

2.6. SQL*Plus : Divers

- DESC[RIBE] *table_ou_vue*
  - Décrit la structure de la table/de la vue
- SHOW USER
  - Visualise le nom de l'utilisateur connecté
- SHOW ERROR
  - Affiche les erreurs
- HOST *commande*
  - Exécuter une commande du système hôte