

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/330662577>

Polycopié du cours: Architecture et Administration des Bases de données

Book · January 2019

CITATIONS

0

READS

2,099

1 author:



Nassim Dennouni

Hassiba Benbouali University of Chlef

15 PUBLICATIONS 14 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



ad hoc environment for the collection and capitalization of human interactions [View project](#)

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Hassiba BENBOUALI de CHLEF
Faculté des Sciences Exactes et d'Informatique



Filière : Informatique
Niveau : Master 1
Spécialités : IL & ISIA

Architecture et Administration des Bases de données

Polycopié du cours

2018-2019



Dr Nassim DENNOUNI
DEPARTEMENT D'INFORMATIQUE

Table des matières

Préambule.....	6
Introduction générale.....	7
Chapitre I. Administration élémentaire de BD.....	8
Introduction	8
I. BD et SGBD	8
I.1. Définition d'une BD	8
I.2. Objectifs des bases de données	8
I.3. Définition d'un SGBD	9
II. Oracle.....	9
II.1.Les outils d'oracle	9
II.1.1.Les outils d'administration d'Oracle	9
II.1.2.Les outils de développement	9
II.1.3.Les outils de communication	10
II.1.4.Les outils de génie logiciel	10
II.1.5.Les outils d'aide à la décision	10
II.2.Architecture fonctionnelle d'oracle	10
II.2.1.Le SQL	11
II.2.2.Le langage PL/SQL	11
II.3. Utilisation du SGBD d'oracle.....	11
II.3.1. Architecture monoposte.....	11
II.3.2. Architecture client - serveur	11
III. Administration élémentaire sous oracle	12
III.1.Création des objets d'une BD.....	13
III.1.1. Gestion des utilisateurs.....	13
III.1.2. Gestion des tables.....	14
- Création de la structure d'une table.....	14
- Modification de la structure d'une table.....	14
- Ajout de nouvelles colonnes à une table	14
- Modification de la structure d'une colonne existante	14
- Suppression de colonnes existantes.....	15
- Ajout d'une contrainte	15
- Suppression de contraintes existantes	15
- Suppression de tables	15
III.2.Les rôles et les privilèges	15

III.2.1.Assigner des privilèges système à un utilisateur	15
III.2.2.Assigner des privilèges objet à un utilisateur	16
III.2.3.Créer des rôles et leur assigner des privilèges.....	16
III.2.4.Suppression d'un rôle	19
III.2.5.Quelques astuces d'administrations	19
Conclusion du chapitre I	20
Chapitre II. Les vues & les séquences.....	21
Introduction	21
I. Les vues	21
I.1.Les privilèges de création de vue	21
I.2.Syntaxe de création de vue	21
I.3.Exemple de vue simple	22
I.4.Exemple de vue avec des opérations numériques	22
I.4.Exemple de vue en lecture seule	22
I.5.Exemple de vue avec contrainte	23
II. Les séquences	24
II.1.Les privilèges de création de séquence.....	24
II.2.La syntaxe de création de séquence.....	24
II.3.Exemple de séquence.....	24
II.4.Exemple d'utilisation de séquence	24
II.5.Autres commandes sur les séquences	25
Conclusion du chapitre II	25
Chapitre III. Les liens & les synonymes	26
Introduction	26
I. Les liens de BD	26
I.1.Les privilèges de création de lien de BD	26
I.2.Syntaxe de lien de BD.....	26
I.3.Exemple de lien de BD	26
II. Les synonymes	27
II.1.Les privilèges de création de synonyme	27
II.2.La Syntaxe de création de synonyme	28
II.2.1.Création de synonyme publique	28
II.2.2.Création de synonyme privé	28
II.3.Exemple de création de synonyme	28
II.3.1.Création d'un synonyme public	28

II.3.2.Création d'un synonyme Privé	29
II.4.Suppression d'un synonyme.....	29
II.5. Affichage des synonymes	29
Conclusion du chapitre III.....	29
Chapitre IV. Les procédures & les fonctions	30
Introduction	30
I. Le langage PL/SQL.....	30
I.1.Structure d'un bloc PL/SQL	30
I.2.Les privilèges sur les procédures et les fonctions	31
I.3.Exemple de bloc anonyme	31
II. Les cursors	31
II.1.L'instruction SELECT INTO.....	32
II.2.L'utilisation du %ROWTYPE.....	33
II.3.Exemple de boucle avec test.....	33
II.4.Exemple de cursor	33
III. Les procédures.....	35
IV. Les Fonctions	38
V. Les Triggers.....	39
V.1. Syntaxe d'un trigger	41
V.2.suppression d'un trigger	41
V.3.Exemple de trigger	41
Conclusion du chapitre IV.....	41
Chapitre V. Les tablespaces & les clusters	42
Introduction	42
I. Les Tablespaces	42
I.1. Le Tablespace parmanent.....	42
I.2. Le tablespace temporaire	42
I.4. Le tablespace UNDO	43
I.5. Tablespace transportable.....	44
I.6. Commandes de manipulation des tablespaces	44
I.6.1.Création des tablespaces	44
I.6.2.Création d'une table et de sa clé dans deux tablespaces différents.....	44
II. Les clusters	45
II.1.Les clusters indexés	46
II.1.1.Création d'un cluster de tables et d'un index associé	46

II.1.2.Création de tables dans un cluster indexé.....	46
II.2.Les clusters de hachage	47
II.2.1.Création de cluster de hachage	48
II.2.2.Requêtes de cluster de hachage	48
II.2.3.Variations de cluster de hachage	50
II.2.4.Stockage de cluster de hachage	50
II.4.CREATE CLUSTER.....	51
II.4.1.Exemple de création d'un cluster	52
II.4.2.Exemple clés de cluster	52
II.4.3.Exemple d'ajout de tables à un cluster	52
II.4.4.Exemple de création de clusters de hachage	53
II.2.5.Exemple de clusters de hachage à table unique.....	53
Conclusion du chapitre V	53
Chapitre VI. La réplication des données	54
Introduction	54
I. Commande COPY	54
II. Les Snapshots	54
II.1.Snapshots simple	55
II.2.Snapshots complexe	55
III. Vues matérialisées.....	56
IV. Les Clusters de tables.....	56
Conclusion du chapitre VI.....	56
Chapitre VII. Les architectures des BD	57
Introduction	57
I. Architecture de BD	57
II. La répartition des BD	58
II.1.Conception descendante	58
II.2.Conception ascendante	58
III. La fragmentation	59
III.1.Fragmentation Horizontale (occurrences)	59
III.2.Fragmentation verticale (attributs)	59
III.3.Fragmentation hybride (valeurs)	59
IV. Les Architectures de Systèmes Parallèles	60
IV.1.Architecture à mémoire partagée (Shared- Memory)	60
IV.2. Architecture à disque partagé (Shared-Disk ou cluster)	60

IV.3.Architecture à mémoire distribuée (Shared-Nothing).....	61
V. Trois types d'accès aux BD.....	61
V.1.Accès Client/multibase.....	61
V.2. Vue répartie	62
V.3. SGBD réparti	63
Conclusion du chapitre VII	63
Conclusion générale	64
TD 1 : Diagramme UML & Administration	65
TD 2 : Attribution des droits d'administration.....	66
TD 3 : Gestion des autorisations d'accès	67
TP 0 : LDD & LMD sous Oracle	68
TP 1 : Création d'utilisateur & connexion	70
TP 2 : Création de vues	72
TP 3 : Attribution des droits d'administration	76
TP 4 : Gestion des autorisations d'accès	78
TP 5 : Utilisation des cursors et des triggers	82
Accès e-learning au module	84
Bibliographie	85

Préambule

Ce polycopié aborde les tâches qui peuvent être réalisées par un administrateur de base de données (BD) ainsi que les différentes architectures existantes dans le monde des systèmes de gestion de base de données (SGBD).

Ce cours permet aussi d'acquérir des savoir-faire comme la création des utilisateurs, des tables, des vues, des procédures, des fonctions, des séquences,...sous le SGBD oracle.

Dans ce contexte, ce module vise à préparer les apprenants à utiliser le SGBD oracle dans leurs projets de fin d'étude ou dans leurs vies professionnelles. A l'issue de ce cours, les apprenants doivent être capables : (1) de choisir l'architecture adaptée à une entreprise donnée, (2) de maîtriser les commandes de base qui permettent la création des objets d'une BD Oracle et (3) de gérer les droits et les privilèges d'accès des utilisateurs du SGBD Oracle.

Ce polycopié est destiné aux étudiants inscrits en première année de master en informatique dans l'une des deux spécialités : Ingénierie des Logiciels (IL) ou Ingénierie des Systèmes d'Information Avancés (ISIA). Ce support concerne un cours du 1er semestre qui comporte une séance de cours et une séance de Travaux Pratique (TP) pour chaque groupe.

D'autre part, le nombre d'heures d'enseignement associé à ce module est de 42 H réparti sur 14 semaines. Chaque semaine comporte 1H30 de cours et 1H30 de TP (en plus de 1h30 de travail personnel sous forme de mini projet). Enfin, le mode d'évaluation de ce module est un examen semestriel et au moins deux contrôles continus sous forme de travaux personnels notés.

Introduction générale

Le chapitre I commence par présenter des notions élémentaires sur l'administration de BD Oracle. Ensuite, ce chapitre donne aussi un aperçu sur les différentes commandes à utiliser pour bien définir les droits et les privilèges des utilisateurs. Enfin, ce chapitre conclut par un survol de quelques astuces très pratiques dans le cadre d'une administration de BD oracle.

Le chapitre II introduit les notions de vues et de séquences. En effet, les utilisateurs peuvent avoir différents types d'accès sur une BD qui peuvent être supportés par la création de vues selon les différents schémas de conception comme les diagrammes de cas d'utilisation UML. D'autre part, les séquences sont aussi traitées dans ce chapitre pour expliquer comment peut-on personnaliser des compteurs au niveau des clés de tables ou de clusters.

Le Chapitre III présente les liens de BD comme une solution pour l'accès à distant à des objets d'une BD. Ces liens facilitent le travail de l'administrateur qui pourra se connecter à ces objets pour faire des requêtes réseaux. Dans ce chapitre aussi, nous expliquons le rôle des synonymes dans un SGBD qui permet de faciliter l'accès à des objets de la BD tout en garantissant leurs sécurités.

Le Chapitre IV aborde les traitements au niveau des BD à travers la définition de procédures, de fonctions et de triggers. Dans ce chapitre, nous expliquons aussi l'apport de la commande « select into » et des curseurs pour les requêtes de type PL/SQL.

Le Chapitre V permet de personnaliser l'espace accordé à des objets de la BD comme les tables et les indexes grâce à l'utilisation des tablespaces. Dans ce chapitre aussi, nous précisons comment les clusters peuvent être utilisés pour réduire le temps de réponse à des requêtes très couteuse à cause des jointures.

Le Chapitre VI traite le problème de réplication de données en mettant l'accent sur des techniques comme les snapshots, les vues matérialisées et les clusters.

Le chapitre VII illustre les trois types d'architecture de BD existantes : BD centralisée, BD répartie et BD distribuée. Ensuite, il aborde la question de fragmentation des données et de parallélisation des SGBD. Enfin, nous citons trois types d'accès aux BD dans un environnement réseau.

Chapitre I. Administration élémentaire de BD

Introduction

Dans ce chapitre, nous introduisons les notions de : (1) base de données (BD), (2) système de gestion de base de données (SGBD) et (3) administration élémentaire dans un environnement de BD Oracle. Ensuite, nous expliquons comment créer et manipuler des objets de la BD oracle. Enfin, nous présentons quelque exemple d'administration de BD en définissant des rôles et des privilèges comment à attribuer aux différents objets de la BD oracle.

I. BD et SGBD

I.1. Définition d'une BD

Une BD est un ensemble de données structurées modélisant un domaine précis (la gestion de stock, la gestion du personnel,...) et qui peut être partagée par plusieurs utilisateurs (administrateur, gestionnaire, utilisateur,...)

D'autre part, les BD servent à stocker des informations sur un support informatique pendant une longue période de taille importante en autorisant des accès multi-utilisateurs. Il faut donc : gérer de manière efficace les accès aux disques et proposer une définition structurée des données afin d'éviter les redondances (Sans, 2000)

I.2. Objectifs des bases de données

Les BD peuvent répondre à plusieurs objectifs parmi lesquels nous citons :

1. **Élimination** de la redondance des données.
2. **Indépendance entre les programmes et les données** : les BD peuvent être manipulés grâce au SQL qui permet de faciliter l'accès aux données sans connaître leur organisation physique sur le support de stockage.
3. **Intégration de données** : les BD doivent permettre l'intégration de toutes les données de l'entreprise dans un réservoir unique de données.

I.3. Définition d'un SGBD

Un Système de Gestion de Base de Données (SGBD) est un ensemble de programmes qui permettent aux utilisateurs de : (1) créer des BD grâce au Langage de Définition de Données, (2) manipuler les objets des BD à l'aide du Langage de Manipulation de données et (3) de contrôler les privilèges des utilisateurs et les droits d'accès en utilisant le Langage de Contrôle de Données (Dennouni, 2017).

Ex. : Oracle, PostgreSQL¹, Access, MySQL, SQL server, DB2², Informix³, etc.

II. Oracle

Oracle est un SGBD écrit en langage C et disponible sur de nombreuses plates-formes matérielles (plus d'une centaine) dont : AIX (IBM), Solaris (Sun), HP/UX (Hewlett Packard), Windows NT (Microsoft) et Linux.

II.1. Les outils d'oracle

Oracle est un environnement de travail constitué de nombreux outils classés selon diverses catégories :

II.1.1. Les outils d'administration d'Oracle

Oracle dispose de nombreux outils permettant de simplifier l'administration de la base de données. Parmi ces outils, les plus connus sont : (1) Oracle Manager (SQL*DBA), (2) NetWork Manager (SQL*Net, Net8), (3) Oracle Enterprise Manager et (4) un outil Import/Export (SQL*Plus, SQL*Loader).

Oracle intègre aussi plusieurs assistants comme (1) Oracle Universal Installer (OUI) : un outil graphique pour installer et mettre à jour les produits Oracle, (2) Oracle Enterprise Manager (OEM) : un outil d'administration de tous les objets de la base de données et (3) Oracle DataBase Assistant : un outil d'aide à la création d'une base de données.

II.1.2. Les outils de développement

Oracle propose des outils de développement permettant d'automatiser la création d'applications s'interfaçant avec la base de données. Ces outils de développement sont : Oracle Developer et SQL*Plus.

1) SQL*Plus est une interface interactive permettant d'envoyer des requêtes SQL et PL/SQL à la base de données. Elle permet notamment de paramétrer l'environnement de travail (formatage des résultats, longueur d'une ligne, nombre de lignes par page, ...)

2) Oracle Developer est un outil de développement graphique gratuit permettant de parcourir les objets, lancer des instructions SQL et mettre au point des programmes PL/SQL. Cet outil est composé de 5 applications de type client-serveur :

- Oracle Forms : un outil permettant d'interroger la base de données de façon graphique sans connaissances préalables du langage SQL. SQL*Forms permet ainsi de développer des applications graphiques (fenêtres, formulaires, ...) permettant de sélectionner, modifier et supprimer des données dans la base.
- Oracle Reports (SQL*ReportWriter) : un outil permettant de réaliser des états

¹ <https://fr.wikipedia.org/wiki/PostgreSQL>

² https://fr.wikipedia.org/wiki/IBM_DB2

³ <https://fr.wikipedia.org/wiki/Informix>

- Oracle Graphics : un outil de génération automatique de graphiques dynamiques pour présenter graphiquement des statistiques réalisées à partir des données de la base
- Procedure Builder : un outil permettant de développer des procédures, des fonctions et des packages
- Oracle Application Express : unique environnement de développement en ligne pour construire une application (utilisant uniquement un navigateur Internet).

II.1.3.Les outils de communication

Oracle Net8 Assistant est un outil qui assure le lien entre les applications et la BD via un protocole réseau. Cet assistant utilise d'une part un listener sur le serveur pour écouter les demandes de connexions à toutes les bases de données gérées par cette machine. D'autre part, sur chaque client, le fichier Tnsnames.ora définit un alias référençant l'instance administrée par le serveur. Cet alias précise le protocole réseau, la machine cible disposant de la base de données, l'identifiant de l'instance (SID) de la base de données cible, etc.

II.1.4.Les outils de génie logiciel

Oracle Designer permet de concevoir un système d'information et de générer le code pour Oracle Developer Suite. Cependant, depuis avril 2018, ce produit est en fin de vie et ne fait désormais plus l'objet de soutien. Actuellement, Oracle utilise d'autres outils de modélisation et de conception comme Oracle JDeveloper et Oracle SQL Developer Data Modeler.

II.1.5.Les outils d'aide à la décision

Oracle implémente l'option OLAP (On-Line Analytical Processing) dans un environnement de base de données. À partir d'Oracle Database 11g, les cubes peuvent remplacer les vues matérialisées multidimensionnelles, simplifiant ainsi la gestion de l'entrepôt de données Oracle et accélérant la réponse aux requêtes (wikipedia, 2018).

II.2.Architecture fonctionnelle d'oracle

Dans le cadre de notre cours, nous avons utilisé le SGBD Oracle car il se démarque des autres SGBD par son administration des BD basée sur la gestion des utilisateurs, des rôles/privilèges et des tablespaces. D'autre part, le SGBD oracle repose sur la notion d'instance de BD et possède ses propres langages : le SQL (Structured Query Language) (Tony, 2014) et le PL/SQL (Procedural Language / Structured Query Language).

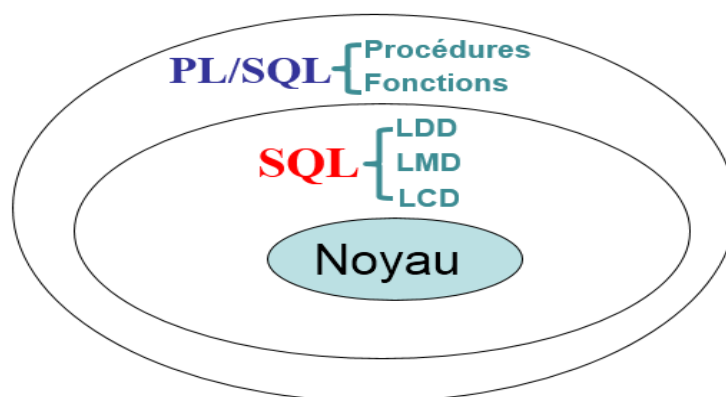


Figure 1: Architecture fonctionnelle d'oracle

II.2.1. Le SQL

Le SQL est un langage pour les BD relationnelles créé en 1970 par IBM. Ses principales caractéristiques sont :

1. **Normalisation** : SQL implémente le modèle relationnel.
2. **Standard** : la plupart des éditeurs de SGBDR intègrent SQL à leurs produits (Oracle, MS SQL Server, DB2, etc.) pour que les données, requêtes et applications soient facilement portables d'une BD à une autre.
3. **Non procédural** : SQL est un langage déclaratif non procédural permettant d'exprimer des requêtes dans un langage relativement simple. En contrepartie il n'intègre aucune structure de contrôle permettant par exemple d'exécuter une boucle itérative.

D'autre part, le SQL peut être utilisé à tous les niveaux dans la gestion d'une BD relationnelles. Le langage de définition de données (LDD) : permet la description de la structure de la base de données (tables, vues, attributs, index). Le langage de manipulation de données (LMD) : permet la manipulation des tables et des vues avec les quatre commandes : SELECT, INSERT, DELETE, UPDATE. Le langage de contrôle de données (LCD) : comprend les primitives de gestion des transactions : COMMIT, ROLLBACK et des privilèges d'accès aux données : GRANT et REVOKE.

II.2.2. Le langage PL/SQL

Le PL/SQL est un langage propriétaire créé par Oracle, utilisé dans le cadre de bases de données relationnelles. Il permet de combiner des requêtes SQL, des boucles, des tests, ... pour définir des procédures, des fonctions, des déclencheurs,... (Cabanac, 2016)

II.3. Utilisation du SGBD d'oracle

Dans le cadre de notre cours, nous nous intéressons à deux type d'architecture du SGBD oracle : l'architecture monoposte et l'architecture client/serveur.

II.3.1. Architecture monoposte

Dans cette architecture, plusieurs utilisateurs du même poste de travail (PC ou serveur) peuvent accéder à la BD locale à l'aide d'une application oracle comme SQL*plus ou autre.

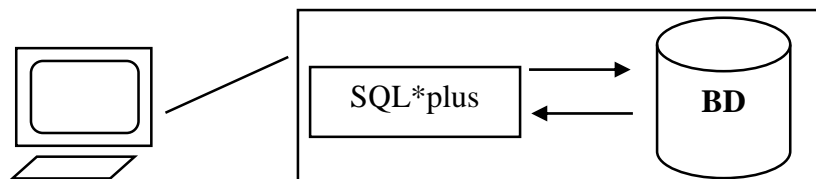


Figure 2: architecture monoposte

II.3.2. Architecture client - serveur

Dans cette architecture, l'application client et le serveur de BD oracle sont séparées. Le client exécute l'application client pour accéder aux informations de la BD. Le serveur gère les fonctions requises pour un accès simultané et partagé des utilisateurs de la BD Oracle.

Par ailleurs, les parties client et serveur peuvent être exécutées par différents ordinateurs connectés via un réseau.

Pour que l'application client (SQL plus, Oracle forms, ...) puisse se connecter à une BD Oracle qui se trouve sur un serveur, il faut fournir trois paramètres : (1) le nom d'utilisateur, (2) le mot de passe et (3) l'alias. Ce dernier regroupe plusieurs données à la fois : (a) le protocole réseau (TCP/IP) utilisé pour accéder à la machine cible, (b) le nom ou l'adresse de la machine cible sur laquelle se situe le serveur, (c) le nom global de la base et le port d'écoute du serveur.

Oracle Net permet de définir ces paramètres utilisés pour la connexion à distance entre une application client et un serveur de BD. Cette connexion est possible grâce au listener, un processus d'écoute qui s'exécute sur le serveur sur le port 1521 (par défaut)

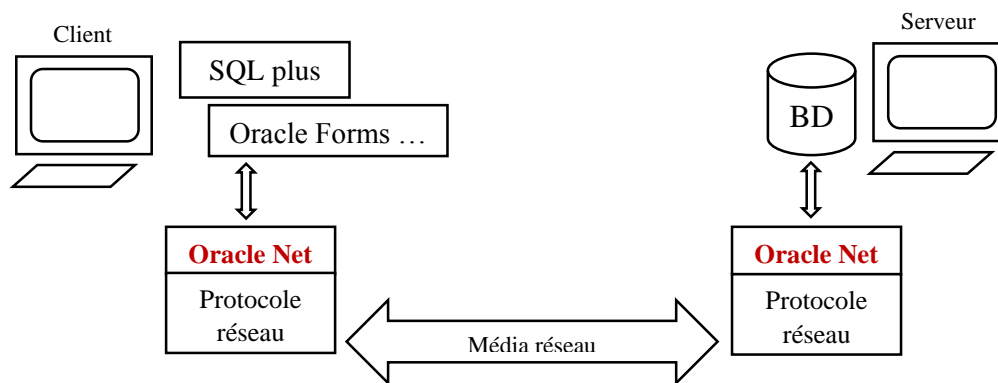


Figure 3: Architecture client - serveur

III. Administration élémentaire sous oracle

Toute BD Oracle possède : une structure logique, une structure physique et un ensemble de vues pour chaque profil d'utilisateurs. Le modèle de données (relationnel ou objet) adopté pendant la phase de conception de la BD et le dictionnaire de données choisi lors de l'étude préalable permettent la description et la correspondance entre ces trois niveaux.

La structure physique au niveau du SGBD oracle est composée de trois types de fichiers : (1) les fichiers de données (Data Files), (2) les fichiers de reprise (Redo Log Files) et (3) les fichiers de contrôle (Control Files). La spécification des Data files et des Redo Log Files se fait lors de la création ou la modification de la BD.

D'une part, les fichiers de données (Data files) assurent le stockage des objets créés par les utilisateurs : tables, index, clusters, etc. D'autre part, les Fichiers Redo Log contiennent les modifications les plus récentes des données de la BD utilisées par Oracle pour remettre la base dans un état cohérent après une panne sans perdre les opérations de mise à jour qui n'ont pas été enregistrées dans les data files. Pour répondre à ce besoin, lors du redémarrage de l'instance de la BD, Oracle applique les mises à jour décrites dans le fichier Red_Log Files sur le fichier Data_file (reprise à chaud).

Après avoir mis en place cet environnement de la BD oracle, l'administration de données consiste à (1) créer la BD, (2) définir les objets de la BD et (3) veiller à la bonne utilisation des données. Dans ce contexte, l'administrateur peut avoir un rôle organisationnel et un rôle technique. Ces deux rôles peuvent être assurés par une ou plusieurs personnes.

Le rôle organisationnel donne à l'administrateur de BD le plus haut niveau de privilège afin de lui permettre de : (1) évaluer l'espace nécessaire, (2) tester la disponibilité du disque et (3) prévoir les moyens assurant la sécurité de la BD (fichiers de reprise, archivage, sauvegarde et restauration de données, ...).

Le Rôle technique consiste à (1) Installer le SGBD et les outils associés, (2) Créer la BD et assurer son évolution, et (3) Gérer des privilèges d'accès.

III.1.Création des objets d'une BD

Le SQL est un langage supporté par la plupart des SGBD qui implémentent le modèle relationnel car il peut être utilisé aux 3 niveaux de la gestion d'une BD : LDD, LMD et LCD.

Le SQL permet la définition des objets manipulés (utilisateurs, tables, vues, index, ...) par les SGBD grâce aux commandes : **CREATE** (création des objets), **ALTER** (modification de la structure des objets), et **DROP** (suppression des objets).

III.1.1. Gestion des utilisateurs

Un utilisateur est un objet de la BD oracle. L'administrateur peut créer, modifier et supprimer un ou plusieurs utilisateurs. Dans le SGBD oracle, chaque utilisateur peut avoir des rôles, profils, des tablespaces par défaut, etc. La commande de création d'utilisateur est :

```
CREATE USER utilisateur
IDENTIFIED BY "password"
[DEFAULT TABLESPACE tbs]
[TEMPORARY TABLESPACE tbs]
[QUOTA {entier [K|M] | UNLIMITED} ON tbs]
PROFILE profil ;
```

Exemple :

```
CREATE USER Nassim
IDENTIFIED BY mohamed2016
DEFAULT TABLESPACE Example_TS QUOTA 10M ON Example_TS
TEMPORARY TABLESPACE Temp_nas QUOTA 5M ON system
PROFILE app_user
PASSWORD EXPIRE;
```

L'utilisateur « **Nassim** » a les caractéristiques suivantes :

- Le mot de passe « **mohamed2016** »
- Tablespace par défaut= « **Example_TS** » avec un quota de 10 mégaoctets
- Tablespace temporaire = « **Temp_nas** »
- Accès au Tablespace système avec un quota de 5 mégaoctets
- Limites des ressources de BD définies par le profil **app_user**
- Le mot de passe expire et doit être changé avant que **Nassim** puisse se connecter à la BD.

Cette commande possède un domaine de privilèges utilisateurs vide. La modification d'un utilisateur se fait par :

```
ALTER USER utilisateur
IDENTIFIED "password"
[DEFAULT TABLESPACE tbs]
[TEMPORARY TABLESPACE tbs]
[QUOTA {entier [K|M] | UNLIMITED} ON tbs]
PROFILE profil
[DEFAULT RÔLE {rôle,... | [ALL][EXCEPT rôle,...] | NONE};
```


Exemple:

```
ALTER USER Nassim  
  IDENTIFIED BY second_2nd_pwd  
  DEFAULT TABLESPACE example;
```

La commande de suppression d'un utilisateur est :

DROP USER utilisateur [**CASCADE**];

L'option **CASCADE** permet de supprimer l'utilisateur et tous ses objets (tables, vues, contraintes etc.)

En l'absence de **CASCADE**, l'utilisateur ne peut pas être supprimé s'il possède des objets. Cette option doit être manipulée avec prudence.

Exemple:

```
Drop USER Nassim cascade ;
```

III.1.2. Gestion des tables

Pour créer une nouvelle table dans votre schéma, vous devez disposer du privilège système **CREATE TABLE**. Pour créer une table dans le schéma d'un autre utilisateur, vous devez disposer du privilège système **CREATE ANY TABLE**. De plus, le propriétaire de la table doit avoir un quota pour l'espace de table contenant la table ou le privilège système **UNLIMITED TABLESPACE**.

- Création de la structure d'une table

Exemple :

```
CREATE TABLE Produit  
(Numprod number (6) not null,  
  Desprod varchar (15) unique,  
  Couleur char,  
  Poids number (8,3),  
  Qte_stk number (7,3),  
  Qte_seuil number (7,3),  
  Prix number (10,3),  
  CodMag number(5,3),  
  Constraint Ck1_Produit CHECK (Poids >=0),  
  Constraint PK_Produit primary key (NumProd),  
  Constraint FK_Produit Foreign Key (CodMag) references Magasin (NumMag));
```

- Modification de la structure d'une table

La table doit figurer dans votre propre schéma ou vous devez disposer du privilège objet **ALTER** sur la table, ou encore vous devez disposer du privilège système **ALTER ANY TABLE**.

- Ajout de nouvelles colonnes à une table

```
ALTER TABLE CLIENT ADD type_clt char(3) ;
```

- Modification de la structure d'une colonne existante

```
ALTER TABLE CLIENT MODIFY type_clt char(5) ;
```


- Suppression de colonnes existantes

```
ALTER TABLE Magasin DROP ville ;
```

- Ajout d'une contrainte

```
ALTER TABLE Magasin ADD Constraint ck1_magasin check (surface between 10 and 100) ;
```

- Suppression de contraintes existantes

```
ALTER TABLE magasin DROP PRIMARY KEY CASCADE ;  
ALTER TABLE produit DROP CONSTRAINT Ck4_Produit ;
```

- Suppression de tables

La table doit figurer dans votre propre schéma ou vous devez disposer du privilège système DROP ANY TABLE.

```
DROP TABLE nom_table ;
```

III.2.Les rôles et les privilèges

Chaque utilisateur d'une BD Oracle qui dispose d'un nom et d'un mot de passe peut posséder des tables, des vues et d'autres ressources qu'il a créées. Les Rôles et les privilèges sont définis pour sécuriser l'accès aux données de la BD. Ces concepts sont mis en œuvre pour protéger les données en accordant (ou retirant) des privilèges à un utilisateur ou un groupe d'utilisateurs

Au niveau de la BD oracle, il existe deux types de privilèges : (1) des privilèges systèmes et (2) des privilèges objets.

Les privilèges au niveau **système** donnent le droit d'exécuter une action particulière sur **n'importe quel objet**. Pour cette raison, les privilèges de niveau système permettent la création, modification, suppression, exécution de **groupes d'objets**. Par exemple, les privilèges CREATE TABLE, CREATE VIEW, CREATE SEQUENCE permettent à l'utilisateur qui les a reçus de créer des tables, des vues et des séquences. D'autre part, le privilège GRANT ANY PRIVILEGE permet d'accorder des privilèges à d'autres utilisateurs.

Les privilèges au niveau **objet** donnent le droit d'exécuter une action donnée sur un **objet spécifique**. Le privilège SELECT, par exemple, permet d'exécuter une opération SELECT sur une table, une vue, une séquence ...

Un rôle **regroupe** un ensemble de privilèges. L'administrateur peut assigner des privilèges spécifiques à des rôles, puis assigner ces rôles aux utilisateurs appropriés ou à un autre rôle. Par exemple, les privilèges SELECT, INSERT, UPDATE, DELETE sur la table SCOTT.EMP permettent à l'utilisateur qui les a reçus de sélectionner, ajouter, modifier et supprimer des lignes dans la table EMP appartenant à l'utilisateur SCOTT.

III.2.1.Assigner des privilèges système à un utilisateur

Lorsqu'un utilisateur est créé avec l'instruction CREATE USER, il ne dispose encore d'aucun droit car aucun privilège ne lui a encore été assigné donc il ne peut même pas se connecter à la base. Pour cette raison, il faut donc lui assigner les privilèges nécessaires pour pouvoir se connecter, créer des tables, des vues, des séquences... Pour atteindre cet objectif, il faut lui assigner ces privilèges de niveau système à l'aide de l'instruction GRANT.

Pour que l'utilisateur puisse simplement se connecter à la base, il doit bénéficier du privilège système CREATE SESSION.

```
GRANT CREATE SESSION TO nom_utilisateur ;
```

Ensuite il faut lui assigner des droits de création de table

```
GRANT CREATE TABLE TO nom_utilisateur ;
```

Puis les droits de création de vues

```
GRANT CREATE VIEW TO nom_utilisateur ;
```

L'ensemble de ces privilèges peuvent être assignés au sein d'une même commande :

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW TO nom_utilisateur;
```

III.2.2. Assigner des privilèges objet à un utilisateur

Pour assigner à l'utilisateur X le droit de sélectionner, insérer, modifier et supprimer des lignes dans la table EMP de l'utilisateur SCOTT, il faut utiliser la commande :

```
GRANT SELECT, INSERT, UPDATE, DELETE ON SCOTT.EMP TO nom_utilisateur _X ;
```

Une liste de colonnes peut être indiquée dans l'instruction afin de restreindre davantage les droits sur une table

```
GRANT UPDATE (JOB, MGR) ON SCOTT.EMP TO nom_utilisateur ;
```

III.2.3. Créer des rôles et leur assigner des privilèges

Lorsque le rôle est créé, il ne contient rien et il faut l'alimenter à l'aide d'instructions GRANT

```
CREATE ROLE comptabilite ;  
GRANT SELECT, INSERT, UPDATE, DELETE ON CPT.FACTURE TO comptabilite ;  
GRANT SELECT, INSERT, UPDATE, DELETE ON CPT.LIG_FAC TO comptabilite ;  
GRANT SELECT, INSERT, UPDATE, DELETE ON CPT.JOURNAL TO comptabilite ;
```

Une fois le rôle créé, il peut être assigné à un utilisateur ou à un autre rôle.

```
GRANT comptabilite TO nom_utilisateur ;  
GRANT comptabilite TO autre_role ;
```

Trois rôles existent en standard : **CONNECT, RESOURCE et DBA.**

Le rôle CONNECT permet l'ouverture (CREATE SESSION) et la modification (ALTER SESSION) d'une session. La création de tables, vues, clusters, séquences, synonymes et liens de bases de données comme le montre la figure suivante :

```
SQL> select * from DBA_SYS_PRIVS where grantee='CONNECT' ;
```

GRANTEE	PRIVILEGE	ADM
CONNECT	CREATE VIEW	NO
CONNECT	CREATE TABLE	NO
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE SESSION	NO
CONNECT	CREATE SYNONYM	NO
CONNECT	CREATE SEQUENCE	NO
CONNECT	CREATE DATABASE LINK	NO

8 ligne(s) sélectionnée(s).

Figure 4: les privilèges du rôle connect

L'option ADMIN OPTION positionnée à NO indique que ces privilèges ne peuvent pas être assignés à un autre utilisateur ou rôle.

Le rôle RESOURCE permet de créer des types, tables clusters, opérateurs, séquences, index et procédures. Il accorde un privilège UNLIMITED QUOTA à l'utilisateur. Les privilèges système assignés au rôle RESOURCE sont indiqués dans la figure qui suit :

```
SQL> select * from DBA_SYS_PRIVS where grantee='RESOURCE' ;
```

GRANTEE	PRIVILEGE	ADM
RESOURCE	CREATE TYPE	NO
RESOURCE	CREATE TABLE	NO
RESOURCE	CREATE CLUSTER	NO
RESOURCE	CREATE TRIGGER	NO
RESOURCE	CREATE OPERATOR	NO
RESOURCE	CREATE SEQUENCE	NO
RESOURCE	CREATE INDEXTYPE	NO
RESOURCE	CREATE PROCEDURE	NO

8 ligne(s) sélectionnée(s).

Figure 5: les privilèges du rôle RESOURCE

Le rôle DBA contient tous les privilèges nécessaires au travail d'un administrateur de BD. Dans ce qui suit, la figure 6 illustre les privilèges systèmes assignés au rôle DBA

```

1* select * from DBA_SYS_PRIVS where grantee='DBA' order by PRIVILEGE
SQL> /

```

GRANTEE	PRIVILEGE	ADM
DBA	ADMINISTER DATABASE TRIGGER	YES
DBA	ADMINISTER RESOURCE MANAGER	YES
DBA	ALTER ANY CLUSTER	YES
DBA	ALTER ANY DIMENSION	YES
DBA	ALTER ANY EVALUATION CONTEXT	YES
DBA	ALTER ANY INDEX	YES
DBA	ALTER ANY INDEXTYPE	YES
DBA	ALTER ANY LIBRARY	YES
DBA	ALTER ANY OUTLINE	YES
DBA	ALTER ANY PROCEDURE	YES
DBA	ALTER ANY ROLE	YES
DBA	ALTER ANY RULE	YES
DBA	ALTER ANY RULE SET	YES
DBA	ALTER ANY SEQUENCE	YES
DBA	ALTER ANY SNAPSHOT	YES
DBA	ALTER ANY TABLE	YES
DBA	ALTER ANY TRIGGER	YES
DBA	ALTER ANY TYPE	YES
...		
...		
DBA	SELECT ANY DICTIONARY	YES
DBA	SELECT ANY SEQUENCE	YES
DBA	SELECT ANY TABLE	YES
DBA	UNDER ANY TABLE	YES
DBA	UNDER ANY TYPE	YES
DBA	UNDER ANY VIEW	YES
DBA	UPDATE ANY TABLE	YES

139 ligne(s) sélectionnée(s).

Figure 6: les privilèges du rôle DBA

Pour voir la liste des rôles assignés à un utilisateur pendant sa connexion à la BD, il faut utiliser la commande suivante :

```

SQL> select * from SESSION_ROLES ;

```

ROLE
CONNECT
RESOURCE

Figure 7: affichage des rôles associés à un utilisateur

Pour voir la liste des privilèges assignés à l'utilisateur au cours de sa session, il faut consulter la vue SESSION_PRIVS comme suit :

```
SQL> select * from SESSION_PRIVS ;

PRIVILEGE
-----
CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
CREATE INDEXTYPE
```

Figure 8: affichage des privilèges associés à un utilisateur

III.2.4. Suppression d'un rôle

Le rôle spécifié ainsi que tous les privilèges qui lui sont associés sont supprimés de la base et également retiré à tous les utilisateurs qui en bénéficiaient grâce à la commande suivante :

```
DROP ROLE nom_role ;
```

III.2.5. Quelques astuces d'administrations

La commande ci-dessous accorde tous les privilèges sur l'objet « employee » de l'utilisateur « SCOTT » :

```
GRANT SELECT, INSERT, UPDATE, DELETE ON employee TO SCOTT;
```

Cette commande peut être remplacée par la commande suivante :

```
GRANT ALL ON employee TO SCOTT;
```

La commande ci-après donne un privilège donné (select sur employee) à tous les utilisateurs

```
GRANT SELECT ON employee TO PUBLIC;
```

La commande suivante retire à l'utilisateur SCOTT tous les privilèges sur l'objet employee :

```
REVOKE SELECT, INSERT, UPDATE, DELETE ON employee FROM SCOTT;
```

La commande ci-dessus est équivalente à la commande qui suit :

```
REVOKE ALL ON employee FROM SCOTT;
```

Dans ce qui suit, tous les privilèges sur l'objet « employee » sont retirés à l'utilisateur « SCOTT » grâce à la commande suivante :

```
REVOKE SELECT ON employee FROM PUBLIC;
```

Par la suite, nous présentons quelques exemples de gestion des droits d'accès sur les procédures et les fonctions :

- 1) Donner le privilege EXECUTE à SCOTT pour executer la procédure 'create_new_employee_proc' ⇒

```
GRANT EXECUTE ON create_new_employee_proc TO SCOTT;
```

- 2) Donner le privilege EXECUTE on à tous les utilisateurs pour exécuter la procédure 'create_new_employee_proc' ⇒

```
GRANT EXECUTE ON create_new_employee_proc TO PUBLIC;
```

- 3) Retirer le privilege execute à SCOTT pour l'exécution de la procédure create_new_employee_proc

```
REVOKE EXECUTE ON create_new_employee_proc FROM SCOTT;
```

- 4) Retirer le privilege execute à tous les utilisateurs de la procedure create_new_employee_proc

```
REVOKE EXECUTE on create_new_employee FROM PUBLIC;
```

Conclusion du chapitre I

Après avoir fait un tour d'horizon sur le SGBD oracle, ses avantages et ses atouts pour l'administration des BD. Le prochain chapitre va mettre l'accent sur la création des vues pour les différents utilisateurs d'une base de données oracle.

Chapitre II. Les vues & les séquences

Introduction

Dans ce chapitre, nous nous focalisons sur deux objets de la BD oracle : les vues et les séquences. Nous présentons (1) leurs définitions, (2) les commandes en SQL qui permettent de les créer et les privilèges associés à la manipulation de ces deux objets. Nous allons répondre à des questions comme (a) comment créer une vue dans Oracle ? (b) à quoi sert une vue ? (c) combien de colonnes dans une vue Oracle ? (d) dans quel cas utilise-t-on une séquence ?

I. Les vues

Une Vue est une **table logique pointant** sur une ou plusieurs **tables ou vues** et ne contient physiquement pas de données. La structure d'une Vue dans le SGBD oracle est stockée dans le dictionnaire de données et peut contenir 1000 colonnes. Nous pouvons utiliser une Vue pour (1) limiter l'accès à des données dans la base de données, (2) faciliter la création de requêtes complexes et présenter des données issues d'une table sous de différents formats.

I.1. Les privilèges de création de vue

Pour créer une vue dans son propre schéma, l'utilisateur doit disposer du privilège système CREATE VIEW. Pour créer une vue dans le schéma d'un autre utilisateur, l'utilisateur doit disposer du privilège système CREATE ANY VIEW.

Le propriétaire du schéma contenant la vue doit disposer des privilèges nécessaires pour sélectionner, insérer, mettre à jour ou supprimer des lignes de toutes les tables ou vues sur lesquelles la vue est basée. Le propriétaire doit obtenir ces privilèges directement, plutôt que via un rôle.

I.2. Syntaxe de création de vue

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[,alias]...)]
AS SELECT ...
WITH { READ ONLY |
CHECK OPTION [ CONSTRAINT constraint ]
```

Nassim doit avoir le privilège CREATE VIEW pour exécuter les créations de vues.

```
SQL> GRANT CREATE VIEW TO Nassim;
```

I.3.Exemple de vue simple

```
SQL> CREATE VIEW v_emp_dept_10 AS
      SELECT      ename, job, sal, deptno
      FROM        emp
      WHERE       deptno = 10;
```

Vue créée.

```
SQL> DESC v_emp_dept_10;
```

Nom	NULL ?	Type
-----	-----	-----
ENAME		VARCHAR2 (10)
JOB		VARCHAR2 (9)
SAL		NUMBER (7,2)
DEPTNO		NUMBER (2)

I.4.Exemple de vue avec des opérations numériques

```
SQL> CREATE VIEW v_emp_dept_10 AS
      SELECT      ename, job, sal*1.33 SAL, deptno
      FROM        emp
      WHERE       deptno = 10;
```

Vue créée.

```
SQL> DESC v_emp_dept_10;
```

Nom	NULL ?	Type
-----	-----	-----
ENAME		VARCHAR2 (10)
JOB		VARCHAR2 (9)
SAL		NUMBER
DEPTNO		NUMBER (2)

I.4.Exemple de vue en lecture seule

Les vues créées avec l'option **WITH READ ONLY** peuvent être interrogées mais aucune opération **DML** (UPDATE, DELETE, INSERT) ne peut être effectuée sur la vue.


```
SQL> CREATE VIEW v_emp_dept_10
      (nom, metier, salaire, dpt) AS
      SELECT      ename, job, sal, deptno
      FROM        emp
      WHERE       deptno = 10
      WITH READ ONLY;
```

Vue créée.

```
SQL> SELECT column_name, updatable
      FROM user_updatable_columns
      WHERE table_name = 'V_EMP_DEPT_10';
```

COLUMN_NAME	UPD
NOM	NO
METIER	NO
SALAIRE	NO
DPT	NO

I.5.Exemple de vue avec contrainte

Les vues créées avec l'option **WITH CHECK OPTION CONSTRAINT** empêche toutes mises à jour de la vue si les conditions de la clause WHERE ne sont pas respectées.

```
SQL> CREATE VIEW v_emp_dept_10
      (nom, metier, salaire, dpt) AS
      SELECT      ename, job, sal, deptno
      FROM        emp
      WHERE       deptno = 10
      WITH CHECK OPTION CONSTRAINT check_10;
```

Vue créée.

```
SQL> INSERT INTO v_emp_dept_10
      (nom, metier, salaire, dpt)
      VALUES('Daniel','DBA',4000,78);
INSERT INTO v_emp_dept_10
      *
ERREUR à la ligne 1 :
ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE
```



```
INCREMENT BY 1;
```

```
SQL> create sequence seq_test_produit  
2 start with 1  
3 increment by 1 ;  
Séquence créée.
```

```
INSERT INTO test_produit (code, nom) VALUE (seq_table1.nextval, 'P1');
```

```
SQL> insert into test_produit (code,nom) values (seq_test_produit.nextval,'p1');  
1 ligne créée.
```

```
SQL> select * from test_produit;  
  
CODE NOM  
-----  
1 p1
```

II.5. Autres commandes sur les séquences

```
DROP SEQUENCE sequence_name ;
```

```
CREATE SEQUENCE supplier_seq MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
NOCACHE;
```

```
ALTER SEQUENCE seq_name INCREMENT BY X;
```

Conclusion du chapitre II

Après avoir expliqué les notions de vue et de séquence dans le cadre de ce chapitre, nous abordons dans le chapitre III deux autres objets de la BD Oracle. Ces objets sont les synonymes et liens de BD qui présentent un grand intérêt pour le travail de l'administrateur de BD Oracle.

Chapitre III. Les liens & les synonymes

Introduction

Dans ce chapitre, nous abordons les concepts de lien de BD et de synonyme. Le premier peut être utilisé pour faciliter l'accès à une BD distante dans le cadre d'une BD distribuée et le second permet l'intégration d'un niveau de sécurité au niveau des noms des objets de la BD.

I. Les liens de BD

Le concept central des systèmes de bases de données distribuées est un lien de base de données. Un lien de base de données est une connexion entre deux serveurs de base de données physiques qui permet à un client d'y accéder en tant que base de données logique.

Pour interroger une BD distante, il faut créer un lien de base de données. Seul l'utilisateur qui a créé un lien privé peut l'utiliser, alors qu'un lien public est utilisé par tous les utilisateurs de la base de données. Les noms de service d'instances sont stockés dans le fichier de configuration utilisé par Net8 intitulé *tnsnames.ora*. Ce fichier spécifie l'hôte, le port, et l'instance associés à chaque nom de service (Oracle, 2019).

I.1. Les privilèges de création de lien de BD

Pour créer un lien privé de BD, il faut avoir le privilège système CREATE DATABASE LINK. Pour créer un lien public de BD, l'administrateur doit attribuer le privilège CREATE PUBLIC DATABASE LINK à l'utilisateur concerné. Toutefois, pour que les liens de BD créés soit opérationnel, il faut disposer du privilège CREATE SESSION au niveau de la BD distante.

I.2. Syntaxe de lien de BD

```
CREATE [PUBLIC][PRIVATE] DATABASE LINK NomLien
CONNECT TO
CURRENT_USER IDENTIFIED BY password
USING connect_string
```

Un lien est soit **privé** ou **public**. La clause CONNECT TO active une session vers la BD distante. La clause **CURRENT_USER** crée un lien BD pour l'utilisateur courant qui doit disposer d'un compte valide dans la BD distante. La clause USING connect_string spécifie le nom de service d'une BD distante.

I.3. Exemple de lien de BD

```
CREATE PUBLIC
```

```
DATABASE LINK bdddist  
CONNECT TO SCOTT IDENTIFIED BY 'tiger'  
USING 'sidbdddist';
```

```
select * from tab1@bdddist;
```

```
select * from USEREX.tab1@bdddist;
```

```
create table tab1 as select * from tab1@bdddist;
```

Avec sysdba :

```
GRANT CREATE DATABASE LINK TO USER;  
select * from dba_db_links;
```

II. Les synonymes

Un synonyme est un **Alias** sur un Objet de la base ou Schéma, une sorte de raccourcis. L'Objet peut être une Table, une Vue, une Séquence, une Procédure, une Fonction, un Package.

Le synonyme peut être public ou privé. Un synonyme public est accessible à partir de tous les schémas des utilisateurs. Le synonyme privé est accessible uniquement à partir du schéma dans lequel il a été créé.

L'administrateur crée des synonymes pour : (1) masquer le vrai nom des objets et leur localisations, (2) simplifier les noms des objets et (3) éviter le pré-fixage dans les requêtes avec le nom de son propriétaire.

II.1. Les privilèges de création de synonyme

Pour créer un synonyme privé dans votre schéma, on utilise CREATE SYNONYM

```
SQL> GRANT CREATE SYNONYM TO User ;
```

Pour créer un synonyme Privé dans n'importe quel schéma, on utilise CREATE ANY SYNONYM :

```
SQL> GRANT CREATE ANY SYNONYM TO User ;
```

Pour créer un synonyme Public CREATE PUBLIC SYNONYM , on utilise CREATE PUBLIC SYNONYM

```
SQL> GRANT CREATE PUBLIC SYNONYM TO User ;
```

II.2.La Syntaxe de création de synonyme

II.2.1.Création de synonyme publique

```
SQL> CREATE [OR REPLACE] [PUBLIC] SYNONYM NomSynonyme FOR [schéma.]nomObjet [@lienBaseDonnées];
```

II.2.2.Création de synonyme privé

```
SQL> CREATE [OR REPLACE] SYNONYM [schéma.]NomSynonyme FOR [schéma.]nomObjet [@lienBaseDonnées];
```

II.3.Exemple de création de synonyme

Tout d'abord vous devez avoir le privilège « CREATE SYNONYM » pour créer un synonyme privé dans votre schéma.

```
SQL> GRANT CREATE SYNONYM TO User ;
```

CREATE ANY SYNONYM pour créer un synonyme Privé dans n'importe quel schéma.

```
SQL> GRANT CREATE ANY SYNONYM TO User ;
```

CREATE PUBLIC SYNONYM pour créer un synonyme Public

```
SQL> GRANT CREATE PUBLIC SYNONYM TO User ;
```

La création de synonyme ne donne pas de droit sur l'objet. Oracle regardera si l'utilisateur accédant au synonym possède les privilèges nécessaires (SELECT, INSERT, UPDATE, DELETE, EXECUTE).

II.3.1.Création d'un synonyme public

```
SQL> CREATE PUBLIC SYNONYM site FOR sales.industrie;  
SQL> CREATE PUBLIC SYNONYM societe FOR sales.industrie;  
SQL> CREATE PUBLIC SYNONYM agence FOR sales.industrie;
```

3 synonymes Public différents (site, société, agence) pour la même table industrie du schéma sales

II.3.2.Création d'un synonyme Privé

```
SQL> CREATE SYNONYM sales.taux FOR sales.salaire;  
SQL> CREATE SYNONYM sales.primes FOR sales.salaire;
```

2 synonymes privés différents (taux, primes) dans le schéma sales pour la même table salaire du schéma sales.

II.4.Suppression d'un synonyme

```
SQL> DROP [PUBLIC] SYNONYM [schéma.]nomSynonyme [FORCE];
```

```
SQL> DROP PUBLIC SYNONYM site;  
SQL> DROP PUBLIC SYNONYM societe;  
SQL> DROP PUBLIC SYNONYM agence;  
SQL> DROP SYNONYM sales.taux;  
SQL> DROP SYNONYM sales.primes;
```

Pour pouvoir supprimer les 3 synonymes publics et les 2 synonymes privés créé précédemment, il faut avoir le privilège **DROP PUBLIC SYNONYM** et/ou **DROP ANY SYNONYM**.

II.5. Affichage des synonymes

Pour connaître la situation actuelle des synonymes, il faut interroger la vue **DBA_SYNONYMS** avec un owner <> 'PUBLIC' ou = 'VotreUser' afin de voir les synonymes existants.

Nous pouvons voir aussi les vues **ALL_SYNONYMS** et **USER_SYNONYMS** pour avoir une idée sur les synonymes disponibles dans la BD.

```
SQL> SELECT * FROM dba_synonyms WHERE owner <>'PUBLIC';
```

Conclusion du chapitre III

Après avoir expliqué les notions de lien de BD et de synonyme, nous avons donné des exemples d'utilisation de ces objets dans la BD oracle. Ensuite, nous avons montré leurs avantages et leurs limites dans le cadre du travail d'un administrateur. Dans le chapitre suivant, nous abordons les procédures et les fonctions qui présentent un bon moyen pour la maîtrise des traitements au niveau de la BD.

Chapitre IV. Les procédures & les fonctions

Introduction

Dans ce chapitre, nous nous focalisons sur l'aspect traitement au niveau de la BD oracle à travers la mise en place de bloc PL/SQL. Ces blocs peuvent être anonyme ou sous forme de procédure et de fonction. Ces trois moyens peuvent être utilisés pour implémenter des triggers et des packages pour faciliter les traitements PL/SQL au niveau de la BD.

I. Le langage PL/SQL

Le PL/SQL est Langage procédural et orienté objet et représente une extension au langage SQL. C'est un Langage pour les procédures stockées, les fonctions et les déclencheurs.

I.1. Structure d'un bloc PL/SQL

```
DECLARE (optional)
    - variable declarations
BEGIN (required)
    - SQL statements
    - PL/SQL statements or sub-blocks
EXCEPTION (optional)
    - Actions to perform when errors occur
END; (required)
```

Il existe trois types de bloc PL/SQL :

- 1) Les blocs anonymes

```
DECLARE
BEGIN
    -statements
EXCEPTION
END;
```

- 2) Les procédures

```
PROCEDURE <name>
IS
BEGIN
    -statements
EXCEPTION
END;
```


3) Les fonctions

```
FUNCTION <name>
RETURN <datatype>
IS
BEGIN
    -statements
EXCEPTION
END;
```

I.2.Les privilèges sur les procédures et les fonctions

Pour qu'un utilisateur puisse exécuter un objet de type procédure ou fonction, l'administrateur doit lui assigner le privilège EXECUTE en utilisant l'instruction suivante :

```
GRANT EXECUTE ON object TO user;
```

Pour retirer le droit d'exécution d'un objet (procédure ou fonction), l'administrateur doit utiliser l'instruction suivante :

```
REVOKE EXECUTE ON object FROM user;
```

I.3.Exemple de bloc anonyme

```
Set serveroutput on
DECLARE
    v_inv_value number(10,2);
    v_price      number(8,2) := 10.25;
    v_quantity   number(8,0) := 400;
BEGIN
    v_inv_value := v_price * v_quantity;
    dbms_output.put('The value is: ');
    dbms_output.put_line(v_inv_value);
END;
/
```

II. Les cursors

Pour exécuter une requête sur plusieurs lignes, Oracle ouvre une zone de travail sans nom qui stocke les informations de traitement. Un curseur vous permet de nommer la zone de travail, d'accéder aux informations et de traiter les lignes individuellement.

Les instructions de type SELECT ... INTO ... manquent de souplesse car elles ne fonctionnent que sur des requêtes retournant une et une seule valeur. Ne serait-il pas intéressant de pouvoir placer dans des variables le résultat d'une requête retournant plusieurs lignes ? Pour répondre à cette question, on peut utiliser un curseur qui est un objet contenant le résultat d'une requête (0, 1 ou plusieurs lignes) (Meslé, 2011).

Un curseur se déclare dans une section DECLARE :

```
CURSOR /* nom curseur */ IS /* requête */ ;
```

Par exemple, si on tient à récupérer tous les employés de la table EMP, on déclare le curseur suivant.

```
CURSOR emp_cur IS SELECT * FROM EMP;
```

Lors de l'ouverture d'un curseur, la requête du curseur est _évaluée, et le curseur contient toutes les données retournées par la requête. On ouvre un curseur dans une section BEGIN :

```
OPEN /* nomcurseur */ ;
```

Par exemple,

```
DECLARE
```

```
CURSOR emp_cur IS SELECT * FROM EMP;
```

```
BEGIN
```

```
OPEN emp_cur ;
```

```
/* Utilisation du curseur */
```

```
END ;
```

Une fois ouvert, le curseur contient toutes les lignes du résultat de la requête. On les récupère une par une en utilisant le mot-clé FETCH :

```
FETCH /* nomdu curseur */ INTO /* liste de variables */;
```

La liste de variables peut être remplacée par une structure de type nom curseur %ROWTYPE.

Si la lecture de la ligne échoue, parce qu'il n'y a plus de ligne à lire, l'attribut %NOTFOUND prend la valeur vraie.

```
DECLARE
```

```
CURSOR emp_cur IS SELECT * FROM EMP;
```

```
ligne emp_cur%rowtype
```

```
BEGIN
```

```
OPEN emp_cur ;
```

```
LOOP
```

```
FETCH emp_cur INTO ligne ;
```

```
EXIT WHEN emp_cur%NOTFOUND ;
```

```
DBMS_OUTPUT.PUT_LINE ( ligne . ename ) ;
```

```
END LOOP;
```

```
/* ... */
```

```
END;
```

II.1.L'instruction SELECT INTO

L'instruction SELECT INTO extrait les données d'une ou de plusieurs tables de la base de données et affecte les valeurs sélectionnées aux variables ou aux collections. Dans son utilisation par défaut (SELECT ... INTO), cette instruction récupère une ou plusieurs colonnes d'une seule ligne.

```
DECLARE
```

```
n NUMBER(2);
```

```
Begin
```

```
SELECT COUNT(*) INTO n FROM etudiants1;
```

```
IF n=0 THEN dbms_output.put('table vide ');
```

```
ELSE dbms_output.put_line('table contient');dbms_output.put_line(n);
```

```
END IF;
```

```
End;
```

II.2.L'utilisation du %ROWTYPE

```

Set serveroutput on
DECLARE
  v_student students%rowtype;
BEGIN
  select * into v_student
    from students
   where sid='123456';
  DBMS_OUTPUT.PUT_LINE (v_student.lname);
  DBMS_OUTPUT.PUT_LINE (v_student.major);
  DBMS_OUTPUT.PUT_LINE (v_student.gpa);
END;

```

II.3.Exemple de boucle avec test

```

DECLARE
  x NUMBER := 100;
BEGIN
  FOR i IN 1..10 LOOP
    IF MOD(i,2) = 0 THEN
      INSERT INTO temp VALUES (i, x, 'i is even');
    ELSE
      INSERT INTO temp VALUES (i, x, 'i is odd');
    END IF;
    x := x + 100;
  END LOOP;
  COMMIT;
END;

```

L'exécution de ce bloc donne le résultat suivant :

NUM_COL1	NUM_COL2	CHAR_COL
1	100	i is odd
2	200	i is even
3	300	i is odd
4	400	i is even
5	500	i is odd
6	600	i is even
7	700	i is odd
8	800	i is even
9	900	i is odd
10	1000	i is even

II.4.Exemple de cursor

Soit la requête suivante :

```
SQL> SELECT ename, empno, sal FROM emp ORDER BY sal DESC;
```

ENAME	EMPNO	SAL
KING	7839	5000
SCOTT	7788	3000
FORD	7902	3000
JONES	7566	2975
BLAKE	7698	2850
CLARK	7782	2450
ALLEN	7499	1600
TURNER	7844	1500
MILLER	7934	1300
WARD	7521	1250
MARTIN	7654	1250
ADAMS	7876	1100
JAMES	7900	950
SMITH	7369	800

```
DECLARE
  CURSOR c1 is SELECT ename, empno, sal FROM emp;
  my_ename VARCHAR2(10);
  my_empno NUMBER(4);
  my_sal NUMBER(7,2);
BEGIN
  OPEN c1;
  FOR i IN 1..5 LOOP
    FETCH c1 INTO my_ename, my_empno, my_sal;
    EXIT WHEN c1%NOTFOUND;
    INSERT INTO temp VALUES (my_sal, my_empno, my_ename);
    COMMIT;
  END LOOP;
  CLOSE c1;
END;
```

La manipulation de ce cursor donne le résultat suivant :

```
SQL> SELECT * FROM temp ORDER BY col1 DESC;
```

NUM_COL1	NUM_COL2	CHAR_COL
5000	7839	KING
3000	7902	FORD
3000	7788	SCOTT
2975	7566	JONES
2850	7698	BLAKE

III. Les procédures

Une procédure stocke du PL / SQL stocké dans la base de données et exécute ce code lorsqu'il est appelé par l'utilisateur. Une procédure est un bloc nommé, éventuellement paramétré, qu'on peut exécuter à la demande (Tounsi, 2018).

Exemple 1 : procédure bonjour

```
Create or replace procedure bonjour
is
begin
    dbms_output.put_line ('Hello');
End;
/
```

```
SQL> create or replace procedure bonjour
2  is
3  begin
4      dbms_output.put_line('Hello');
5  end;
6  /

Procédure créée.
```

On appelle une procédure PLSQL par son nom. On peut le faire directement depuis SQLPlus avec la commande execute

```
SQL> execute bonjour;
```

```
SQL> SET SERVEROUTPUT ON
SQL> execute bonjour;
Hello

Procédure PL/SQL terminée avec succès.
```

ou dans un bloc PLSQL

```
SQL> begin
    bonjour;
end;
/
```

```
SQL> begin
  2  bonjour;
  3  end;
  4  /
Hello

Procédure PL/SQL terminée avec succès.
```

Exemple 2 : passage de paramètre comme donnée

```
create or replace procedure carre (a in number)
is
begin
  dbms_output.put_line('Carré = '||a*a);
end;
/
```

```
SQL> create or replace procedure carre (a in number)
  2  is
  3  begin
  4      dbms_output.put_line('Carré = '||a*a);
  5  end;
  6  /

Procédure créée.
```

```
SQL> execute carre(5);
Carré = 25

Procédure PL/SQL terminée avec succès.
```

Un paramètre in est pris comme une donnée et ne doit pas être modifié dans la procédure.

Exemple 3 : Procédure qui calcule le maximum de deux nombres et le renvoie en résultat

```
declare
  a number;
  procedure max (a in number, b in number, x out number)
  is
  begin
    if a>b then x := a;
    else x:=b;
    end if;
  end;

begin
  max(2,5,a);
  dbms_output.put_line('max = '||a);
end;
/
```

```

SQL> declare
2   a number;
3   procedure max (a in number, b in number, x out number)
4   is
5   begin
6       if a>b then x := a;
7       else x:=b;
8       end if;
9   end;
10
11 begin
12     max(2,5,a);
13     dbms_output.put_line('max = '||a);
14 end;
15 /
max = 5
Procédure PL/SQL terminée avec succès.

```

Noter sur cet exemple que la procédure max a été déclarée à l'intérieur d'un bloc PLSQL dans la zone declare.

Exemple 4 : Procédure qui permute deux nombres

Les paramètres in out sont modifiables dans la procédure. Ce sont des paramètres données (en entrée) et résultat (en sortie).

```

create or replace procedure
permute (a in out number, b in out number) is
    w number;
begin
    w := a;
    a := b;
    b := w;
end;
/

```

```

SQL> create or replace procedure
2   permute (a in out number, b in out number) is
3   w number;
4   begin
5       w := a;
6       a := b;
7       b := w;
8   end;
9   /
Procédure créée.

```

Pour tester cette procédure, nous pouvons imaginer le bloc PL/SQL suivant :

```

SQL> declare
2   x number := 1;
3   y number := 2;
4   begin
5       permute (x,y);
6       dbms_output.put_line(x || ' ' || y);
7   end;
8   /
2 1
Procédure PL/SQL terminée avec succès.

```

IV. Les Fonctions

Une fonction définie par l'utilisateur contient du code PL / SQL stocké dans la base de données et exécutée lorsqu'un appel de fonction est effectué. Les fonctions sont semblables aux procédures, mais retournent une valeur résultat. Une fonction diffère d'une procédure par le fait qu'on peut l'utiliser dans une expression. La fonction suivante convertit un montant en Dinar vers un montant en Euro.

```
create function DAtoEuro(montant in number) return number
is
begin
return montant / 200;
end;
/
```

```
SQL> create function DAtoEuro(montant in number) return number
2  is
3  begin
4  return montant / 200;
5  end;
6  /

Fonction créée.
```

Pour tester cette fonction, nous suivons les étapes suivantes :

```
SQL> create table employee ( code number, nom varchar(20), salaire number);

Table créée.
```

```
SQL> insert into employee values (1, 'dennouni', 10000);
1 ligne créée.

SQL> insert into employee values (2, 'slimane', 20000);
1 ligne créée.

SQL> insert into employee values (3, 'hadjhenni', 30000);
1 ligne créée.
```

```
SQL> select DAtoEuro(salaire) from employee;

DATOEURO(SALAIRE)
-----
                50
                100
                150
```

L'usage d'une fonction personnalisée dans une requête de type select est très intéressant pour effectuer des calculs et les afficher sous forme de résultat d'interrogation. Pour améliorer le fonctionnement de cette fonction, nous pouvons rajouter le cas le paramètre d'entrée n'est pas un nombre. Pour résoudre ce problème, nous ajoutons l'exception qui traite cette d'erreur de conversion.


```

create or replace function DAtEuro(montant in varchar2) return number is
begin
    return to_number (montant) / 200;
exception
    when others then      -- si erreur SQL
        return null;
end;
/

```

Pour tester cette fonction, nous suivons ces étapes :

```

SQL> delete from employee where code <=4;
5 ligne(s) supprimé(s).
SQL> alter table employee
  2  modify salaire varchar(20);
Table modifiée.
SQL> insert into employee values (4, 'tayehi', 'a');
1 ligne créée.
SQL> insert into employee values (1, 'denouni', 10000);
1 ligne créée.
SQL> insert into employee values (2, 'slimane', 20000);
1 ligne créée.
SQL> insert into employee values (3, 'hadjhenni', 30000);
1 ligne créée.
SQL> select DAtEuro(salaire) from employee;
DATOEURO(SALAIRE)
-----
          50
         100
         150
SQL> select DAtEuro(salaire) from employee where code=4;
DATOEURO(SALAIRE)
-----
SQL> select code,nom, DAtEuro(salaire) from employee where code=4;

```

CODE	NOM	DATOEURO(SALAIRE)
4	tayehi	

V. Les Triggers

Les *Triggers* sont des procédures qui sont lancées automatiquement (on dit déclenchées) après une opération de mise à jour. Ils servent à répercuter automatiquement d'autres actions.

Exemple 1 : Le trigger suivant imprime un message **après l'insertion** d'un employé

```
create or replace trigger monTrigger1
after
  insert on employee
begin
  dbms_output.put_line('Ligne insérée correctement');
end;
/
```

```
SQL> create or replace trigger monTrigger1
2   after
3     insert on employee
4   begin
5     dbms_output.put_line('Ligne insérée correctement');
6   end;
7   /
```

Dúclencheur crúú.

```
SQL> insert into employee values (7,'allali',4500);
Ligne insérée correctement

1 ligne crúúe.
```

Où on voit que le trigger est lancé (affichage 'OK') suite à la mise à jour des données de la base (1 row created). C'est un effet de bord de la mise à jour.

Exemple 2 : Un *trigger* de mise à jour

```
create or replace trigger monTrigger2
after
  update on employee
begin
  dbms_output.put_line('ligne mise à jour correctement');
end;
/
```

Où on voit que le trigger est lancé pour la mise à jour d'une donnée quelconque de la relation employee (adresse ou salaire etc.)

```
SQL> create or replace trigger monTrigger2
2   after
3     update on employee
4   begin
5     dbms_output.put_line('ligne mise à jour correctement');
6   end;
7   /
```

Dúclencheur crúú.

```
SQL> update employee set salaire = salaire + 100 where code=1;
ligne mise à jour correctement

1 ligne mise ó jour.
```

V.1. Syntaxe d'un trigger

```
CREATE OR REPLACE TRIGGER <trigger_name>
[BEFORE/AFTER][DELETE/INSERT/UPDATE of <column_name |, column_name... |>
ON <table_name>
|FOR EACH ROW|
|WHEN <triggering condition>|
|DECLARE|
BEGIN
    trigger statements
    .....
END;
```

V.2.suppression d'un trigger

```
DROP TRIGGER <trigger_name>;
```

V.3.Exemple de trigger

```
CREATE OR REPLACE TRIGGER UPPERCASE
BEFORE INSERT OR UPDATE ON STUDENTS
FOR EACH ROW
DECLARE
BEGIN
    :new.lastname:=UPPER(:new.lastname);
    :new.firstname:=UPPER(:new.firstname);
END UPPERCASE;
```

Conclusion du chapitre IV

Dans les chapitres précédents, nous avons fait un tour d'horizon sur des objets courants d'une BD Oracle comme les tables, les vues, les liens, les séquences, les procédures, les fonctions et les triggers. La manipulation de ce type d'objet permet une administration de premier niveau de la BD oracle. Dans ce qui suit, nous abordons des notions plus avancées de la BD oracle comme les tablespaces et les clusters.

Chapitre V. Les tablespaces & les clusters

Introduction

Dans ce chapitre, nous faisons un survol des notions de tablespaces et de clusters. Nous abordons les commandes SQL sous Oracle qui permettent de manipuler ces deux éléments de la BD oracle.

I. Les Tablespaces

Un tablespace est un espace de table qu'on peut utiliser en lecture/écriture. L'instruction ALTER TABLESPACE pour mettre l'espace de table en mode hors connexion ou en ligne, pour y ajouter des fichiers de données ou des fichiers temporaires, ou pour en faire un espace de table en lecture seule.

Pour créer les tablespaces, il faut avoir le privilège :

```
CREATE TABLESPACE
```

Pour détailler les tablespaces existants, il faut consulter la vue :

```
Select * from USER_TABLESPACES
```

I.1. Le Tablespace permanent

Ce **tablespace** est un espace **logique** qui contient les objets stockés dans la BD comme les **tables** ou les **indexes**. Il est composé d'au moins un **datafile** qui est **physiquement** présent sur le serveur de BD (Chaque **BD** possède au moins un tablespace appelé **SYSTEM**).

I.2. Le tablespace temporaire

Ce tablespace contient des objets de schéma uniquement pour la durée d'une session. Ces objets sont stockés dans des fichiers temporaires. Ce tablespace peut être utilisé dans des opérations de tri pour lesquelles la SORT_AREA_SIZE n'est pas suffisamment grande.

Paramétrage du tablespace temporaire de la base

```
CREATE DATABASE <SID>...  
  DEFAULT TEMPORARY TABLESPACE temp;
```

Création d'un tablespace temporaire

```
CREATE TEMPORARY TABLESPACE temp
  TEMPFILE 'g:\oracle\oradata\orafrance\temp01.dbf'
  SIZE 20M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 10M;
```

Assignment d'un tablespace temporaire à un utilisateur

```
ALTER USER orafrance
  TEMPORARY TABLESPACE temp;
```

Modification du tablespace temporaire de la base

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp2;
```

I.4. Le tablespace UNDO

Le **tablespace UNDO** comme son nom l'indique, est réservé exclusivement à l'**annulation** des commandes DML (UPDATE, INSERT, ...). Ce **tablespace** est unique à une instance, donc il y a autant d'UNDO que d'instance de base de données (System Identifier - SID).

Lorsqu'on exécute l'ordre DELETE par exemple, Oracle commence par copier les lignes à supprimer dans le tablespace UNDO. Ensuite, Oracle indique que les blocs contenant les données dans le tablespace d'origine sont libres. Un ROLLBACK permet de revenir en arrière alors que le COMMIT supprimera les lignes du tablespace UNDO.

Paramétrage du tablespace UNDO de la base

```
CREATE DATABASE <SID>...
  UNDO TABLESPACE undotbs
  DATAFILE 'g:\oracle\oradata\orafrance\undotbs.dbf' size 100M;
```

Création du tablespace UNDO

```
CREATE UNDO TABLESPACE undotbs
  DATAFILE 'g:\oracle\oradata\orafrance\undotbs.dbf' size 100M;
```

Modification du tablespace UNDO utilisé

```
ALTER SYSTEM
  SET UNDO_TABLESPACE=undotbs2;
```

I.5. Tablespace transportable

Le tablespace transportable sert à copier les données entre deux bases de données. Par l'exemple, nous pouvons copier les tablespaces DATA_TBS et INDEX_TBS de la base dvp1 vers dvp2. Pour qu'un tablespace puisse être transporté, il doit contenir tous les objets soit interdépendants. On ne pourra pas transporter un tablespace qui contient une table dont les indexes seraient créés dans un autre tablespace.

Pour qu'un tablespace soit transportable, il faut que les BD source et cible doivent être sur des plateformes identiques : il est impossible donc de transporter un tablespace d'un serveur Sun Solaris à un serveur Windows 2000. Ensuite, la source et la cible doivent utiliser le même jeu de caractère. Enfin, Il est évidemment impossible de transporter un tablespace dans une base qui contiendrait un tablespace de même nom.

I.6. Commandes de manipulation des tablespaces

I.6.1. Création des tablespaces

```
SQL> create tablespace DATA_TBS
  2 datafile 'd:\dvp1\oradata\data_tbs.ora' size 1M reuse
  3 AUTOEXTEND OFF
  4 ONLINE
  5 default storage
  6 (initial 32 k next 32 k
  7 minextents 2 maxextents unlimited
  8 pctincrease 1);

Tablespace created.

SQL> create tablespace INDEX_TBS
  2 datafile 'd:\dvp1\oradata\index_tbs.ora' size 1M reuse
  3 AUTOEXTEND OFF
  4 ONLINE
  5 default storage
  6 (initial 32 k next 32 k
  7 minextents 2 maxextents unlimited
  8 pctincrease 1) ;

Tablespace created.
```

I.6.2. Création d'une table et de sa clé dans deux tablespaces différents

```
SQL> create table orafrance.matable (col1 NUMBER, col2 NUMBER )
  2 tablespace DATA_TBS;

Table created.

SQL> alter table orafrance.matable add
  2 (constraint matable_pk primary key (col1)
  3 USING INDEX TABLESPACE INDEX_TBS
  4 );

Table altered.
```

I.6.3. Assignment de tablespaces à un user

```
CREATE USER IL_ISIA  
IDENTIFIED BY '123'  
DEFAULT TABLESPACE example  
QUOTA 10M ON example  
TEMPORARY TABLESPACE temp3  
QUOTA 5M ON system  
PASSWORD EXPIRE;
```

L'utilisateur « IL_ISIA » dispose d'un quota of 10 megabytes de Default tablespace example, d'un espace « temp » de type Temporary tablespace. D'autre part, l'utilisateur « IL_ISIA » accède à un quota de 5 megabytes du tablespace SYSTEM et doit changer le password avant sa connexion à la BD.

II. Les clusters

Un cluster de tables est un groupe de tables qui partagent des colonnes communes et stockent les données associées dans les mêmes blocs. Par exemple, lorsque les tables sont en cluster, un seul bloc de données peut contenir des lignes provenant de plusieurs tables. Par exemple, un bloc peut stocker des lignes des tables « employés » et « départements ».

La clé du cluster (cluster key) est la ou les colonnes que les tables en cluster ont en commun. Par exemple, les tables « employés » et « départements » partagent la colonne department_id. Pour cette raison, la spécification de la clé de cluster doit se faire lors de la création du cluster de tables et lors de la création de toutes les tables ajoutées au cluster de tables.

La valeur de la clé de cluster est la valeur des colonnes de clé de cluster pour un ensemble de lignes particulier. Toutes les données qui contiennent la même valeur clé de cluster, comme department_id = 20, sont physiquement stockées ensemble. Chaque valeur de clé de cluster n'est stockée qu'une seule fois dans le cluster et dans l'index de cluster, quel que soit le nombre de lignes de différentes tables contenant la valeur.

Pour une analogie, supposons qu'un responsable des ressources humaines ait deux casiers : l'un avec des boîtes de dossiers d'employés et l'autre avec des boîtes de dossiers de département. Les utilisateurs demandent souvent les dossiers de tous les employés d'un service particulier. Pour faciliter la récupération, le gestionnaire réorganise toutes les cases dans une même bibliothèque. Elle divise les boîtes par identifiant de département. Ainsi, tous les dossiers des employés du département 20 et le dossier du département 20 lui-même sont regroupés dans une boîte. Les dossiers des employés du département 100 et ceux du département 100 se trouvent dans une autre boîte, etc.

Par ailleurs, il est recommandé de regrouper des tables lorsqu'elles sont principalement interrogées (mais non modifiées) et que les enregistrements des tables sont fréquemment interrogés ensemble ou joints. Étant donné que les clusters de tables stockent des lignes associées de tables différentes dans les mêmes blocs de données.

Les clusters de tables correctement utilisés offrent les avantages suivants par rapport aux tables non clustérisées : (1) les E / S de disque sont réduites pour les jointures de tables en cluster, (2) le temps d'accès est amélioré pour les jointures des tables en cluster et (3) il faut moins de

stockage pour stocker les données de table et d'index associées, car la valeur de la clé de cluster n'est pas stockée de manière répétée pour chaque ligne. Généralement, la mise en cluster de tables n'est pas appropriée dans le cas où les tables sont fréquemment mises à jour.

II.1. Les clusters indexés

Un cluster d'index est un cluster de table qui utilise un index pour localiser des données. L'index de cluster est un index B-tree sur la clé de cluster. Un index de cluster doit être créé avant que des lignes puissent être insérées dans des tables en cluster.

II.1.1. Création d'un cluster de tables et d'un index associé

Supposons que vous créez le cluster `employee_departments_cluster` avec la clé de cluster `department_id`, comme indiqué dans l'exemple suivant :

```
CREATE CLUSTER employees_departments_cluster (department_id NUMBER(4))  
SIZE 512;
```

```
CREATE INDEX idx_emp_dept_cluster ON CLUSTER employees_departments_cluster;
```

Puisque la clause `HASHKEYS` n'est pas spécifiée, `employee_departments_cluster` est un cluster indexé. Cet exemple crée un index nommé `idx_emp_dept_cluster` sur la clé de cluster `department_id`.

II.1.2. Création de tables dans un cluster indexé

Vous créez les tables d'employés et de services dans le cluster, en spécifiant la colonne `department_id` en tant que clé de cluster, comme suit (les points de suspension marquent l'emplacement où la spécification de colonne est placée) :

```
CREATE TABLE employees (...  
    CLUSTER employees_departments_cluster (department_id);  
CREATE TABLE departments ( ... )  
    CLUSTER employees_departments_cluster (department_id);
```

Supposons que vous ajoutiez des lignes aux tables des employés et des départements. La base de données stocke physiquement toutes les lignes de chaque service à partir des tables des employés et des services dans les mêmes blocs de données. La base de données stocke les lignes dans un segment et les localise avec l'index.

La figure 9 illustre le cluster de tables `employee_departments_cluster`, qui contient les employés et les services. La base de données stocke les lignes des employés du département 20 ensemble, du département 110 ensemble, etc. Si les tables ne sont pas en cluster, la base de données ne garantit pas que les lignes associées soient stockées ensemble.

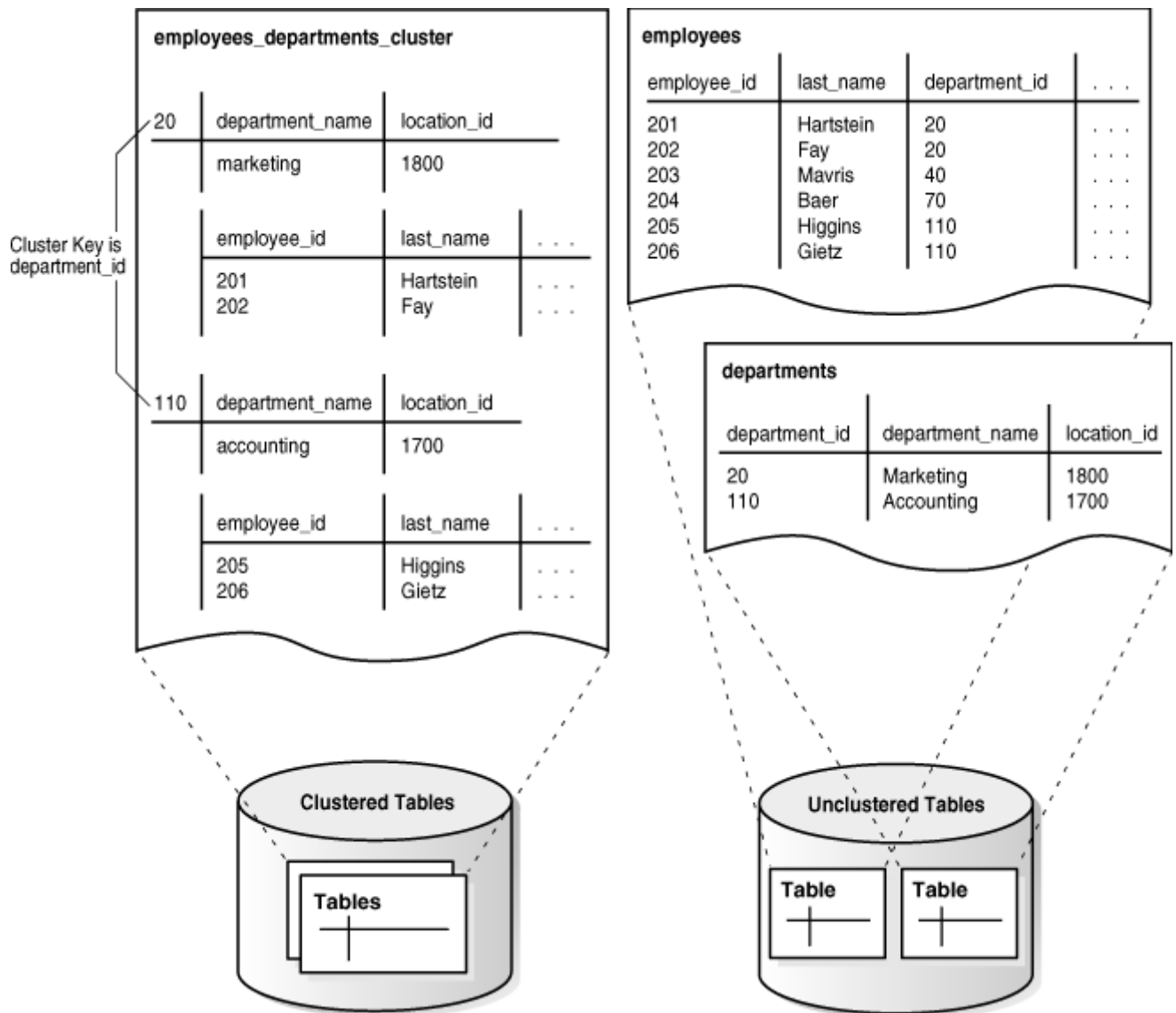


Figure 9: Données de la table en cluster

Dans la figure 9, l'index de cluster B-tree associe la valeur de clé de cluster à l'adresse de bloc de base de données du bloc contenant les données. Par exemple, l'entrée d'index de la clé 20 indique l'adresse du bloc contenant les données pour les employés du département 20. L'index de cluster est géré séparément, comme un index sur une table non clustérisée, et peut exister dans un espace de table distinct du cluster de tables.

II.2. Les clusters de hachage

Un cluster de hachage est similaire à un cluster indexé, à la différence que la clé d'index est remplacée par une fonction de hachage. Aucun index de cluster séparé n'existe. Dans un cluster de hachage, les données sont l'index.

Avec une table indexée ou un cluster indexé, Oracle Database localise les lignes de table à l'aide de valeurs de clé stockées dans un index séparé. Pour rechercher ou stocker une ligne dans une table indexée ou un cluster de tables, la base de données doit effectuer au moins deux E / S : (1) une ou plusieurs E/S pour rechercher ou stocker la valeur de clé dans l'index et (2) une autre E/S pour lire ou écrire la ligne dans la table ou le cluster de tables.

Pour rechercher ou stocker une ligne dans un cluster de hachage, Oracle Database applique la fonction de hachage à la valeur de clé de cluster de la ligne. La valeur de hachage résultante correspond à un bloc de données dans le cluster, que la base de données lit ou écrit pour le compte de l'instruction émise.

Le hachage est un moyen facultatif de stockage des données de table pour améliorer les performances de récupération des données. Les groupes de hachages peuvent être bénéfiques lorsque les conditions suivantes sont remplies :

- Une table est interrogée beaucoup plus souvent que modifiée.
- La colonne de clé de hachage est interrogée fréquemment avec des conditions d'égalité, par exemple, WHERE department_id = 20. Pour de telles requêtes, la valeur de la clé de cluster est hachée. La valeur de la clé de hachage pointe directement sur la zone du disque qui stocke les lignes.
- Vous pouvez raisonnablement deviner le nombre de clés de hachage et la taille des données stockées avec chaque valeur de clé.

II.2.1.Création de cluster de hachage

Pour créer un cluster de hachage, vous utilisez la même instruction CREATE CLUSTER que pour un cluster indexé, avec l'ajout d'une clé de hachage. Le nombre de valeurs de hachage pour le cluster dépend de la clé de hachage.

La clé de cluster, comme la clé d'un cluster indexé, est une clé unique ou une clé composite partagée par les tables du cluster. Une valeur de clé de hachage est une valeur réelle ou possible insérée dans la colonne de clé de cluster. Par exemple, si la clé de cluster est department_id, les valeurs de clé de hachage peuvent être 10, 20, 30, etc.

Oracle Database utilise une fonction de hachage qui accepte un nombre infini de valeurs de clé de hachage en entrée et les trie dans un nombre fini de compartiments. Chaque compartiment a un identifiant numérique unique appelé valeur de hachage. Chaque valeur de hachage est mappée sur l'adresse du bloc de base de données pour le bloc qui stocke les lignes correspondant à la valeur de la clé de hachage (départements 10, 20, 30, etc.).

Dans l'exemple suivant, le nombre de départements susceptibles d'exister est 100. HASHKEYS a donc la valeur 100:

```
CREATE CLUSTER employees_departments_cluster  
  
  (department_id NUMBER(4))  
  
SIZE 8192 HASHKEYS 100;
```

Une fois que vous avez créé le cluster employés_départements_cluster, vous pouvez créer les tables « employés » et « département » avec ce cluster. Vous pouvez ensuite charger des données dans le cluster de hachage.

II.2.2.Requêtes de cluster de hachage

Dans les requêtes d'un cluster de hachage, la base de données détermine comment hacher les valeurs de clé saisies par l'utilisateur.

Par exemple, les utilisateurs exécutent fréquemment des requêtes telles que les suivantes, en entrant différents numéros d'ID de service pour p_id:

```
SELECT *
FROM employees
WHERE department_id = :p_id;

SELECT *
FROM departments
WHERE department_id = :p_id;

SELECT *
FROM employees e, departments d
WHERE e.department_id = d.department_id
AND d.department_id = :p_id;
```

Si un utilisateur interroge des employés dans department_id = 20, la base de données peut alors hacher cette valeur au compartiment 77. Si un utilisateur interroge des employés à département_id = 10, la base de données pourrait alors hacher cette valeur au compartiment 15. La base de données utilise la valeur de hachage générée en interne pour localiser le bloc contenant les lignes d'employé pour le département demandé.

La figure 10 décrit un segment de cluster de hachage sous forme d'une rangée horizontale de blocs. Comme indiqué dans le graphique, une requête peut extraire des données dans une seule E / S.

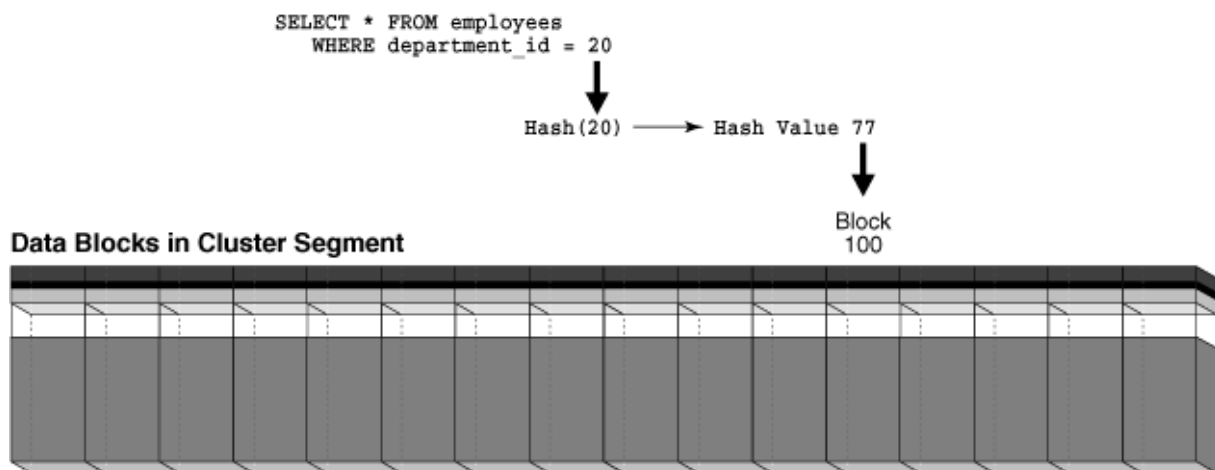


Figure 10 : Récupération de données d'un cluster de hachage

Une limitation des clusters de hachage est l'indisponibilité d'analyses de plages sur des clés de cluster non indexées (voir "Analyse de plages d'index"). Supposons qu'il n'existe pas d'index séparé pour le cluster de hachage créé lors de la création du cluster de hachage. Une requête

pour des départements avec des ID compris entre 20 et 100 ne peut pas utiliser l'algorithme de hachage, car il ne peut pas hacher toutes les valeurs possibles comprises entre 20 et 100. Etant donné qu'aucun index n'existe, la base de données doit effectuer une analyse complète.

II.2.3. Variations de cluster de hachage

Un cluster de hachage à une seule table est une version optimisée d'un cluster de hachage qui prend en charge une seule table à la fois. Un mappage un à un existe entre les clés de hachage et les lignes.

Un cluster de hachage à table unique peut être avantageux lorsque les utilisateurs ont besoin d'un accès rapide à une table par clé primaire. Par exemple, les utilisateurs recherchent souvent un enregistrement d'employé dans la table des employés par `employee_id`.

Un cluster de hachage trié stocke les lignes correspondant à chaque valeur de la fonction de hachage de manière à ce que la base de données puisse les renvoyer efficacement dans un ordre trié. La base de données effectue le tri optimisé en interne. Pour les applications qui consomment toujours des données dans un ordre trié, cette technique peut permettre une récupération plus rapide des données. Par exemple, une application peut toujours trier sur la colonne `order_date` de la table `orders`.

II.2.4. Stockage de cluster de hachage

Oracle Database alloue de l'espace pour un cluster de hachage différemment d'un cluster indexé. Dans l'exemple de Création d'un cluster de hachage, `HASHKEYS` spécifie le nombre de départements susceptibles d'exister, alors que `SIZE` spécifie la taille des données associées à chaque département. La base de données calcule une valeur d'espace de stockage en fonction de la formule suivante :

$$\text{HASHKEYS} * \text{SIZE} / \text{database_block_size}$$

Ainsi, si la taille de bloc est de 4096 octets dans l'exemple présenté dans Création d'un cluster de hachage, la base de données alloue au moins 200 blocs au cluster de hachage.

Oracle Database ne limite pas le nombre de valeurs de clé de hachage que vous pouvez insérer dans le cluster. Par exemple, même si `HASHKEYS` vaut 100, rien ne vous empêche d'insérer 200 départements uniques dans la table des départements. Cependant, l'efficacité de la récupération du groupe de hachage diminue lorsque le nombre de valeurs de hachage dépasse le nombre de clés de hachage.

Pour illustrer les problèmes d'extraction, supposons que le bloc 100 de la figure 10 soit complètement rempli de lignes pour le service 20. Un utilisateur insère un nouveau service avec l'ID de service 43 dans la table des services. Le nombre de départements dépassant la valeur `HASHKEYS`, la base de données hache `department_id` 43 à la valeur de hachage 77, qui correspond à la même valeur de hachage utilisée pour `department_id` 20. Le hachage de plusieurs valeurs d'entrée vers la même valeur de sortie est appelé une collision de hachage.

Lorsque les utilisateurs insèrent des lignes dans le cluster pour le service 43, la base de données ne peut pas stocker ces lignes dans le bloc 100, qui est plein. La base de données relie le bloc 100 à un nouveau bloc de dépassement de capacité, par exemple le bloc 200, et stocke les lignes insérées dans le nouveau bloc. Les blocs 100 et 200 sont désormais éligibles pour stocker des données pour l'un ou l'autre département. Comme le montre la figure 11, une requête du service 20 ou 43 nécessite désormais deux E / S pour extraire les données : le bloc 100 et son bloc 200 associé. Vous pouvez résoudre ce problème en recréant le cluster avec une valeur différente du HASHKEYS.

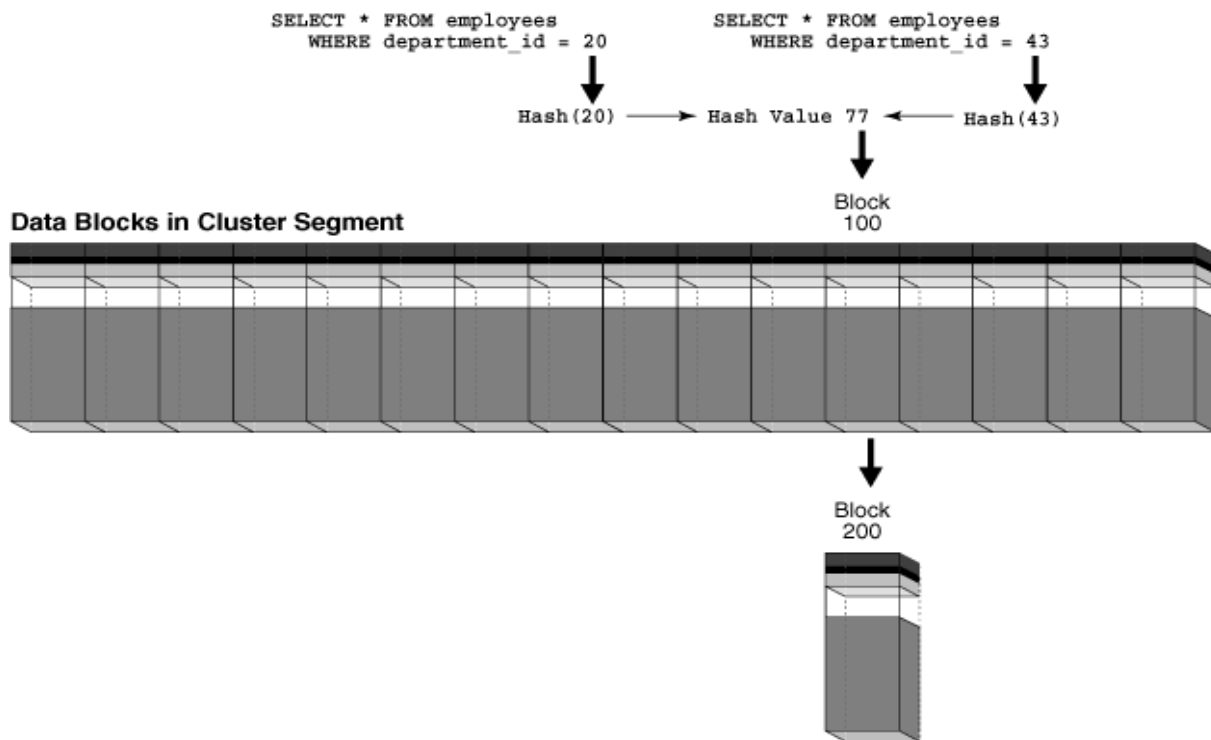


Figure 11: Récupération de données d'un cluster de hachage lorsqu'une collision de hachage se produit

II.4.CREATE CLUSTER

Un cluster est un objet de schéma contenant les données d'une ou de plusieurs tables, qui ont toutes une ou plusieurs colonnes en commun. Oracle Database stocke toutes les lignes de toutes les tables partageant la même clé de cluster. L'instruction `CREATE CLUSTER` est utilisée pour créer un cluster. Pour avoir des informations sur les clusters existants, interrogez les vues de dictionnaire de données `select * from USER_CLUSTERS`, `select * from ALL_CLUSTERS` et `select * from DBA_CLUSTERS`.

Pour créer un cluster dans votre propre schéma, vous devez disposer du privilège système `CREATE CLUSTER`. Pour créer un cluster dans le schéma d'un autre utilisateur, vous devez disposer du privilège système `CREATE ANY CLUSTER`. En outre, le propriétaire du schéma devant contenir le cluster doit disposer d'un quota d'espace sur l'espace de table contenant le cluster ou du privilège système `UNLIMITED TABLESPACE`. La base de données Oracle ne crée pas automatiquement d'index pour un cluster lors de sa création initiale. Les instructions de langage de manipulation de données (DML) ne peuvent pas être émises sur les tables de cluster d'un cluster indexé tant que vous n'avez pas créé d'index de cluster avec une instruction `CREATE INDEX`.

II.4.1.Exemple de création d'un cluster

L'instruction suivante crée un cluster nommé personnel avec le service de colonne de clé de cluster, une taille de cluster de 512 octets et des valeurs de paramètre de stockage :

```
CREATE CLUSTER personnel (department NUMBER(4))
SIZE 512
STORAGE (initial 100K next 50K);
```

```
SQL> CREATE CLUSTER personnel2
2 (department NUMBER(4))
3 SIZE 512 ;
Cluster créé.
```

II.4.2.Exemple clés de cluster

L'instruction suivante crée l'index de cluster sur la clé de cluster du personnel :

```
CREATE INDEX idx_personnel ON CLUSTER personnel;
```

```
SQL> CREATE INDEX idx_personnel2 ON CLUSTER personnel2;
Index créé.
```

Après avoir créé l'index de cluster, vous pouvez ajouter des tables à l'index et effectuer des opérations DML sur ces tables.

II.4.3.Exemple d'ajout de tables à un cluster

Les instructions suivantes créent des tables départementales à partir de la table exemple emp et les ajoutent au cluster de personnel créé précédemment.

```
CREATE TABLE dept_10
    CLUSTER personnel (department_id)
    AS SELECT * FROM employees WHERE department_id = 10;
CREATE TABLE dept_20
    CLUSTER personnel (department_id)
    AS SELECT * FROM employees WHERE department_id = 20;
```

```
SQL> CREATE TABLE departement_20
2 CLUSTER personnel (id_department)
3 AS SELECT * FROM emp WHERE id_department = 20;
Table créée.
SQL> CREATE TABLE departement_10
2 CLUSTER personnel (id_department)
3 AS SELECT * FROM emp WHERE id_department = 10;
Table créée.
```

II.4.4.Exemple de création de clusters de hachage

L'instruction suivante crée un cluster de hachage nommé language avec la colonne de clé de cluster cust_language, un maximum de 10 valeurs de clé de hachage, dont chacune est allouée sur 512 octets, ainsi que des valeurs de paramètre de stockage :

```
CREATE CLUSTER language (cust_language VARCHAR2(3))

SIZE 512 HASHKEYS 10

STORAGE (INITIAL 100k next 50k);
```

```
SQL> CREATE CLUSTER language2 (cust_language VARCHAR2(3))
  2  SIZE 512 HASHKEYS 10
  3  STORAGE (INITIAL 100k next 50k);
Cluster créé.
```

Étant donné que l'instruction précédente omet la clause HASH IS, Oracle Database utilise la fonction de hachage interne pour le cluster.

L'instruction suivante crée une adresse de cluster de hachage nommée avec la clé de cluster composée des colonnes code postal et code_pays, et utilise une expression SQL contenant ces colonnes pour la fonction de hachage :

```
CREATE CLUSTER address

(postal_code NUMBER, country_id CHAR(2))

HASHKEYS 20

HASH IS MOD(postal_code + country_id, 101);
```

```
SQL> CREATE CLUSTER address2
  2  (postal_code NUMBER, country_id CHAR(2))
  3  HASHKEYS 20
  4  HASH IS MOD(postal_code + country_id, 101);
Cluster créé.
```

II.2.5.Exemple de clusters de hachage à table unique

L'instruction suivante crée un cluster de hachage à une seule table nommé cust_orders avec la clé de cluster customer_id et un maximum de 100 valeurs de clé de hachage, dont chacune se voit attribuer 512 octets :

```
SQL> CREATE CLUSTER cust_orders2 (customer_id NUMBER(6))
  2  SIZE 512 SINGLE TABLE HASHKEYS 100;
Cluster créé.
```

Conclusion du chapitre V

Dans ce chapitre nous avons montré l'intérêt que présentent les clusters dans un contexte de BD Oracle surtout pour l'optimisation des requêtes. Nous nous sommes aussi focalisés sur le concept de tablespace et son utilité pour la meilleure gestion des ressources du SGBD oracle.

Chapitre VI. La réplication des données

Introduction

Afin de réduire la quantité de données transmises sur le réseau, et améliorer par conséquent les performances des requêtes, plusieurs options de réplication peuvent être envisagées : (1) la commande COPY, (2) les Snapshots, (3) les vues matérialisées et les Clusters (HAMDANE, 2018).

I. Commande COPY

Cette option consiste à répliquer régulièrement les données sur le serveur local au moyen de la commande **COPY de SQL*Plus**.

Exemple:

```
COPY FROM RH/PUFFINSTUFF@LOC  
  
CREATE Employes  
  
USING  
  
SELECT * FROM Employes WHERE Etat = 'NM';  
  
COMMIT ;
```

La base est identifiée par le service nommé **LOC**. durant la connexion, une session devrait être initiée par le compte **RH** avec le mot de passe **PUFFINSTUFF**.

L'inconvénient est que les données **ne peuvent pas être mises à jour**. Pour cette raison, la commande REPLACE est utilisée pour remplacer le contenu des tables.

II. Les Snapshots

Cette option utilise des *snapshots* pour répliquer les données depuis une source maître vers plusieurs cibles. Les *snapshots* peuvent être en lecture seule (ang. *read-only*) ou mis à jour (ang. *updateable*). Avant de créer un *snapshot*, il faut d'abord créer **un lien vers la base de données source**. Deux types de *snapshots* peuvent être créés : simples et complexes.

Un *snapshot simple* ne contient pas de clause distinct, group by, de jointure multitable ou d'opérations set.

Par contre, un REFRESH COMPLETE est obligatoire pour les *snapshots* **complexes**.

```
CREATE PUBLIC DATABASE LINK bdddist
CONNECT TO

SCOTT IDENTIFIED BY 'tiger'

USING 'sidbdddist';
```

```
select * from scott.tab1 @bdddist;
```

II.1.Snapshots simple

```
CREATE SNAPSHOT LocalEmp

TABLESPACE data2

STORAGE (INITIAL 100K next 100K PCTINCREASE 0)

REFRESH FAST

START with SysDate

NEXT SysDate+7

AS SELECT * FROM Employes@RH_Lien;
```

Ce *snapshot* est défini de façon à extraire les données maîtres et renouveler l'opération 7 jours plus tard.

Un REFRESH FAST utilise un snapshot log, pour actualiser le snapshot. Ainsi, pour chaque mise à jour, seules les modifications sont envoyées, et non l'ensemble des données.

II.2.Snapshots complexe

```
CREATE SNAPSHOT LOG ON Employes

TABLESPACE DATA

STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 0)

PCTFREE 5 PCTUSED 90;
```

Le *snapshot log* est à créer avant le *snapshot*:

Une utilisation classique des *snapshots* en mise à jour est le cas du contrôle technique automobile. Tous les centres de contrôle stockent des données concernant les véhicules qu'ils ont contrôlés durant la journée.

Chaque nuit, les données sont déversées dans la base nationale qui centralise les données de l'ensemble du parc automobile du pays.

III. Vues matérialisées

```
CREATE MATERIALIZED VIEW Ventes-par-Mois  
  
TABLESPACE DATA_AGG  
  
REFRESH COMPLETE  
  
START WITH sysdate  
  
NEXT sysdate+1  
  
ENABLE QUERY REWRITE  
  
AS  
  
SELECT mois, SUM(montant)  
  
FROM Ventes  
  
GROUP BY mois;
```

La clause `ENABLE QUERY REWRITE` permet à l'optimiseur de rediriger les requêtes émises sur la table vers la vue matérialisée et la clause `NEVER REFRESH` empêche tout type d'actualisation de la vue matérialisée. D'autre part, une vue matérialisée ne peut pas contenir les opérations `UNION`, `MINUS`, `INTERSECT`.

IV. Les Clusters de tables

Un cluster de tables est un regroupement de tables souvent interrogées ensemble qui partagent des colonnes communes et stockent les données associées dans les mêmes blocs.

Lorsque les tables sont en cluster, un seul bloc de données peut contenir des lignes provenant de plusieurs tables. Par exemple, un bloc peut stocker des lignes des tables « employés » et « départements »

La clé de cluster est la colonne que les tables en cluster ont en commun, par exemple les tables « employés » et « départements » partagent la colonne « id_department ».

Les clusters de tables correctement utilisés offrent les avantages suivants par rapport aux tables non clustérisées : (1) les E /S de disque sont réduites pour les tables en cluster, (2) le temps d'accès est amélioré car on peut éviter les jointures et (3) moins de stockage car la clé de cluster n'est pas stockée de manière répétée pour chaque ligne.

Généralement, la mise en cluster de tables n'est pas appropriée dans le cas où les tables sont fréquemment mises à jour.

Conclusion du chapitre VI

Ce chapitre a montré que l'intérêt de la réplication des données à travers quatre techniques différentes. Dans le chapitre qui suit, nous allons nous intéresser aux architectures des BD oracle.

Chapitre VII. Les architectures des BD

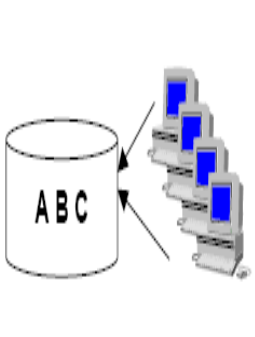
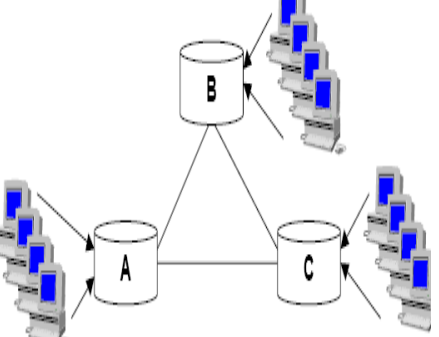
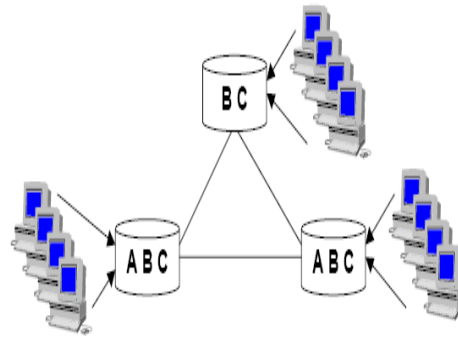
Introduction

Dans un système d'information, les objets d'une BD peuvent ne pas se trouver au même endroit ou sur la même machine. Pour cette raison, nous allons présenter dans le cadre de ce chapitre trois types d'architecture de BD (centralisée, répartie et distribuée). Dans ce contexte de répartition des données, des techniques de fragmentation horizontale ou verticale peuvent être utilisées pour répondre aux contraintes de localisation des sites des entreprises (Godin & Desrosiers, 2011) (Abiteboul, Nguyen, & Rigaux, 2016).

I. Architecture de BD

Dans ce qui suit, nous illustrons les avantages et les inconvénients de trois types d'architectures (centralisée, répartie et distribuée) d'une même BD qui contient trois tables : A, B et C. La BD centralisée regroupe les trois tables dans un seul site pour qu'elles soient accessibles par l'ensemble des utilisateurs à partir de tous les sites à l'aide d'applications client/serveur. La BD répartie associe à chaque site une BD selon le trafic des transactions effectué par les utilisateurs dans le but de garantir un meilleur temps de réponse aux différentes requêtes locales. La BD distribuée se base sur le principe de duplication de données au niveau des sites où se déroulent les traitements afin de réduire le temps d'exécution des requêtes (J.Chevance, 2003).

Tableau 1: Comparaison entre les 3 architectures de BD

		
Égalité des accès Facilité de gestion	Rapidité d'accès aux données locales Autonomie locale de chaque site	Disponibilité des données Rapidité d'accès aux données locales
Contention sur la BD	Gestion globale de la BD	Coordination des MAJ

II. La répartition des BD

Les BDR ont une architecture plus adaptée à l'organisation des entreprises décentralisées. En effet, les BDR ont souvent des données répliquées pour plus de fiabilité et disponibilité. Dans un contexte de BDR, la panne d'un site n'est pas très importante pour l'utilisateur, qui s'adressera à autre site. D'autre part, les entreprises bénéficient de meilleures performances car elles peuvent réduire le trafic sur le réseau en rapprochant les traitements de l'endroit où les données sont accédées. Enfin, une architecture de BD décentralisée facilite l'accroissement des entreprises car son extension se fait simplement par l'ajout de machines sur le réseau.

La définition du schéma de répartition est la partie la plus délicate de la phase de conception d'une BDR car il n'existe pas de méthode miracle pour trouver la solution optimale. L'administrateur doit donc prendre des décisions en fonction de critères techniques et organisationnels avec pour objectif de minimiser le nombre de transferts entre sites, les temps de transfert, le volume de données transférées, les temps moyens de traitement des requêtes, le nombre de copies de fragments, etc...

II.1. Conception descendante

Cette approche est intéressante quand on part d'aucun existant donc nous définissons un schéma conceptuel global de la BDR puis nous distribuons ce schéma sur les différents sites en des schémas conceptuels locaux. La répartition se fait donc en deux étapes : (1) la fragmentation, et (2) l'allocation de ces fragments aux sites (voir la figure 12).

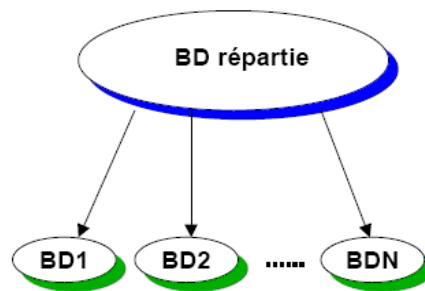


Figure 12: conception descendante d'une BDR

II.2. Conception ascendante

Cette approche suppose que la répartition des BD est déjà existante, mais il faut réussir à intégrer les différentes BDs existantes en une seule BD globale. Donc, les schémas conceptuels locaux existent et il faut réussir à les unifier dans un schéma conceptuel global comme le montre la figure 13 :

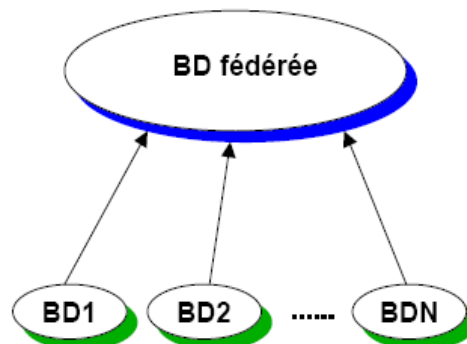


Figure 13: conception ascendante d'une BDR

III. La fragmentation

La fragmentation est le processus de décomposition d'une BD en un ensemble de sous-bases de données. Cette décomposition doit être sans perte d'information. La fragmentation peut être coûteuse s'il existe des applications qui possèdent des besoins opposés. Les règles de fragmentation sont les suivantes (Moussa, 2006) :

1. La complétude : pour toute donnée d'une relation R , il existe un fragment R_i de la relation R qui possède cette donnée.
2. La reconstruction : pour toute relation décomposée en un ensemble de fragments R_i , il existe une opération de reconstruction.

III.1.Fragmentation Horizontale (occurrences)

Les occurrences d'une même classe peuvent être réparties dans des fragments différents en utilisant deux opérateurs. L'opérateur de partitionnement est la *sélection* (σ) et l'opérateur de recomposition est l'*union* (\cup).

Par exemple, la relation *Compte* peut être fractionnée en *Compte1* et *Compte2* :

$\text{Compte1} = \sigma [\text{TypeCompte} = \text{'courant'}] \text{Compte}$

$\text{Compte2} = \sigma [\text{TypeCompte} = \text{'dépôt'}] \text{Compte}$

La recomposition de *Compte* est $\text{Compte1} \cup \text{Compte2}$.

III.2.Fragmentation verticale (attributs)

Toutes les valeurs des occurrences pour un même attribut se trouvent dans le même fragment. Une fragmentation verticale est utile pour distribuer les parties des données sur le site où chacune de ces parties est utilisée. L'opérateur de partitionnement est la *projection* (π) et l'opérateur de recomposition est la *jointure*.

Par exemple, la table *Client* peut être fragmentée en deux relations :

$\text{Cli1} = \pi [\text{NoClient}, \text{NomClient}] \text{Client}$

$\text{Cli2} = \pi [\text{NoClient}, \text{Prénom}, \text{Age}] \text{Client}$

III.3.Fragmentation hybride (valeurs)

C'est la combinaison des deux fragmentations horizontale et verticale. Les occurrences et les attributs peuvent donc être répartis dans des partitions différentes. L'opération de partitionnement est une combinaison de projections et de sélections et l'opération de recomposition est une combinaison de jointures et d'unions.

Exemple :

La relation *Client* est obtenue avec : $(\text{Cli3} \cup \text{Cli5}) * \text{Cli4} * \text{Cli6}$

Relation **Cli3** $\pi [\text{NoClient}, \text{NomClient}] (\sigma [\text{Age} < 38] \text{Client})$

Relation **Cli5** $\pi [\text{NoClient}, \text{NomClient}] (\sigma [\text{Age} \geq 38] \text{Client})$

Relation **Cli4** $\pi [\text{NoClient}, \text{Prénom}] \text{Client}$

Relation **Cli6** $\pi [\text{NoClient}, \text{Age}] \text{Client}$

IV. Les Architectures de Systèmes Parallèles

Trois architectures parallèles sont définies selon le critère de partage de ressources (Donez, 2003) (Dupire, 2004).

IV.1. Architecture à mémoire partagée (Shared-Memory)

Dans ce type d'architecture, les disques et les mémoires centrales sont partagés par les processeurs du système comme cela est indiqué dans la figure 14.

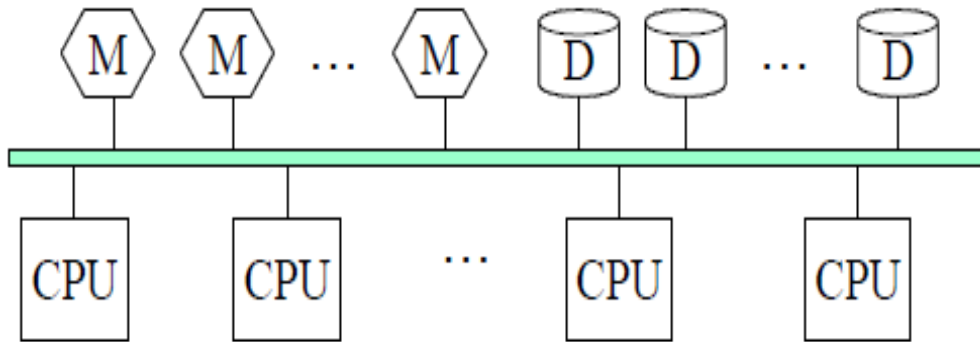


Figure 14: Exemples de SGBD : XPRS (U. de Berkeley), DBS3 (Bull).

Dans ce qui suit, le tableau 2 explique les avantages et les inconvénients de l'architecture à mémoire partagée.

Tableau 2: les avantages et les inconvénients de l'architecture à mémoire partagée

Avantages	Inconvénients
La communication inter-processeurs est rapide grâce à l'accès partagé aux MC.	Accès conflictuels aux mémoires centrales peuvent dégrader les performances.
L'échec d'un processeur n'entraîne pas la non-possibilité d'accès à données.	Architecture non scalable.

IV.2. Architecture à disque partagé (Shared-Disk ou cluster)

Chaque processeur a sa mémoire centrale privée, mais les disques sont partagés par tous les processeurs du système.

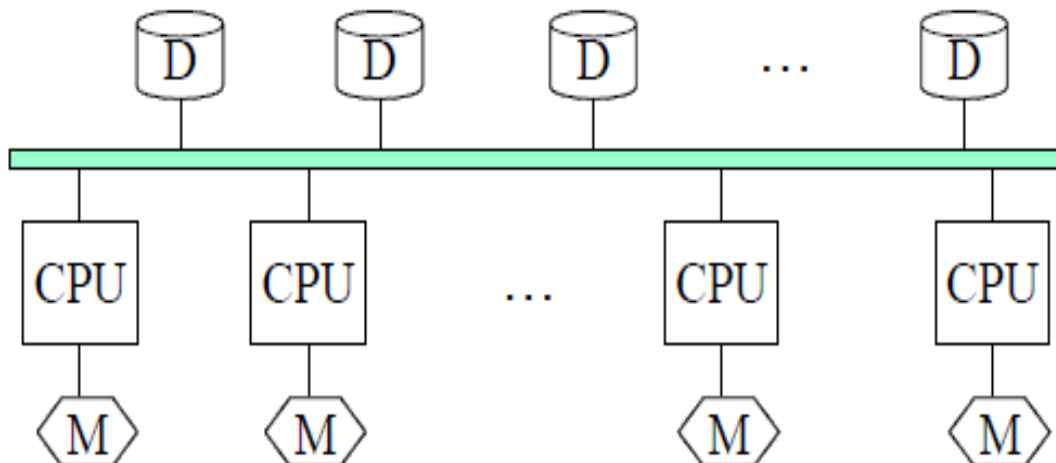


Figure 15: Exemples de SGBD : IMS/VS (IBM), VAX DBMS (DEC) ...

Dans ce qui suit, le tableau 3 explique les avantages et les inconvénients de l'architecture à disque partagé.

Tableau 3: les avantages et les inconvénients de l'architecture à disque partagé.

Avantages	Inconvénients
Equilibre de charge facile à réaliser.	La complexité du maintien de la cohérence des caches des processeurs.
L'échec d'un processeur n'entraîne pas la non-disponibilité des données.	L'accès aux disques est limité par de la capacité du bus

IV.3.Architecture à mémoire distribuée (Shared-Nothing)

Chaque processeur a sa propre mémoire centrale et disque.

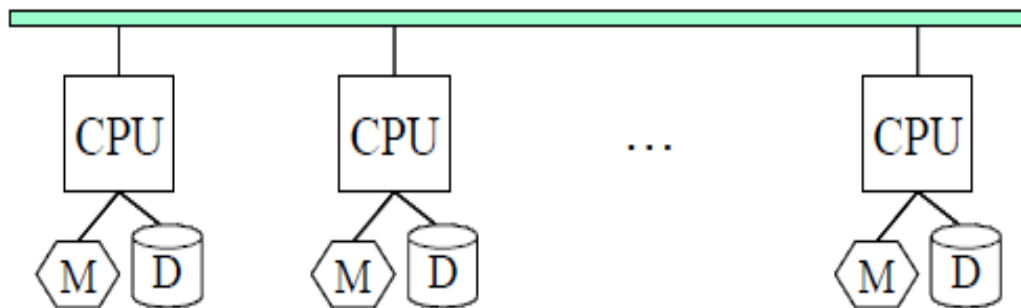


Figure 16: Exemples de SGBD : GAMMA (U. de Wisconsin), BUBBA (MCC),...

Dans ce qui suit, le tableau 4 explique les avantages et les inconvénients de l'architecture à mémoire distribuée.

Tableau 4: les avantages et les inconvénients de l'architecture à mémoire distribuée.

Avantages	Inconvénients
Coût abordable, vu que le système est une collection de PCs.	L'échec d'un processeur rend l'accès aux données impossibles. D'où la nécessité de techniques de haute disponibilité → maintien des miroirs ou des disques de redondance.
	Coût de transfert sur le réseau de grand volume de données d'une requête.

V. Trois types d'accès aux BD

V.1.Accès Client/multibase

ODBC (Open Database Connectivity) permet de manipuler plusieurs BD qui sont mises à disposition par des SGBD ayant chacun un son propre pilote. C'est une spécification contrôlée par Microsoft et supportée par les principaux fournisseurs de SGBD (voir la figure 17).

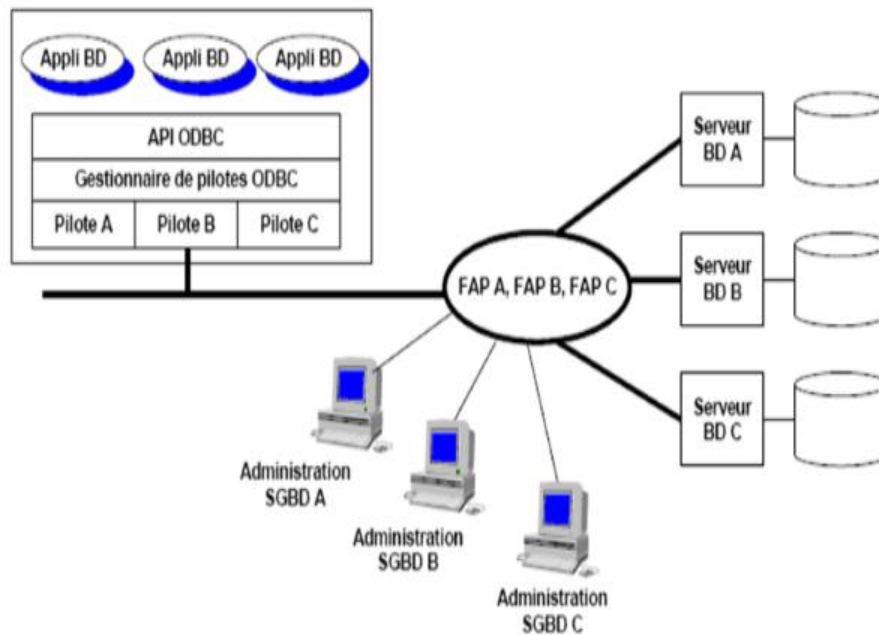


Figure 17: Accès Client/multibase en utilisant ODBC

JDBC (Java Database Connectivity) permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC disponibles pour tous les SGBD connus (voir la figure 18).

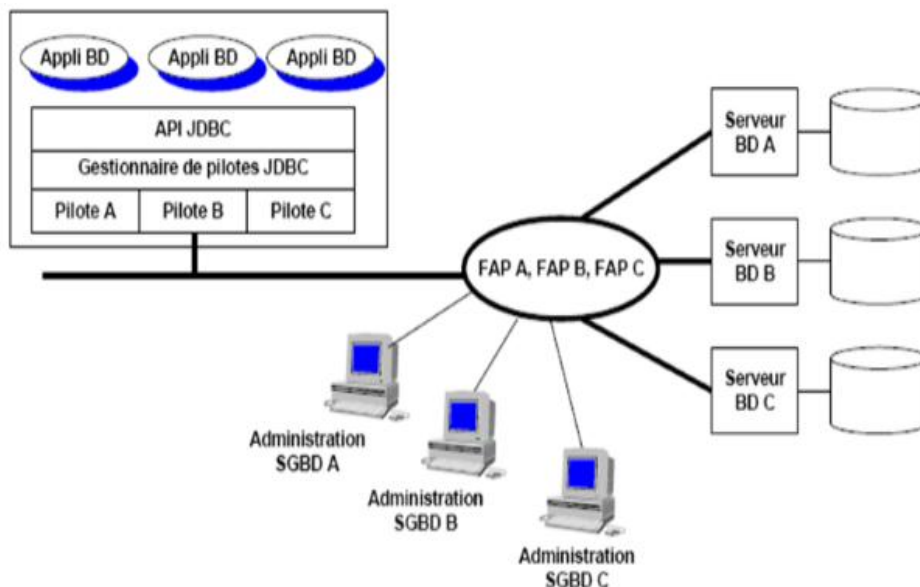


Figure 18: Accès Client/multibase en utilisant JDBC

V.2. Vue répartie

Une vue matérialisée est un stockage statique (dans une table) d'un résultat de requête. Il permet donc d'anticiper des requêtes complexes en pré-calculant tout ou partie du résultat. Dans ce qui suit, la figure 19 explique comment peut-on répartir les calculs à l'aide de vues.

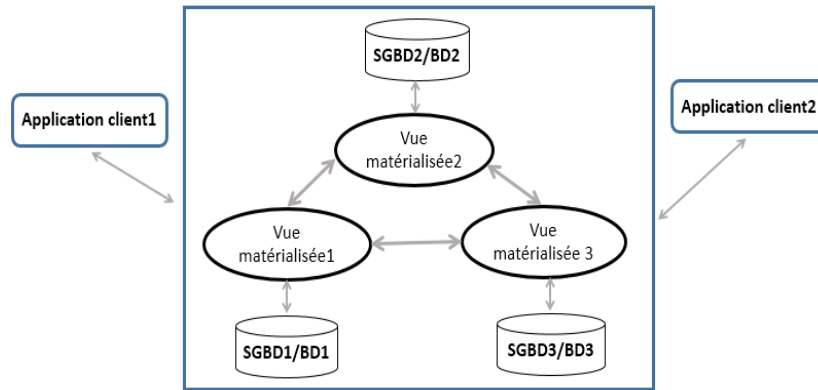


Figure 19: exemple de gestion des vues réparties

La localisation des sites et la coordination des mises à jour est prise en compte par les vues distribuées et un protocole de validation en deux phases.

V.3. SGBD réparti

Une base de données centralisée est gérée par un seul SGBD et elle est stockée dans sa totalité à un emplacement physique unique avec une seule et même unité de traitement. Par opposition, une base de données distribuée est gérée par plusieurs processeurs, sites ou SGBD.

La localisation est assurée pendant la définition de la BDR et les opérations de mise à jour sont supportées par les différents SGBD comme cela est indiqué dans la figure 20.

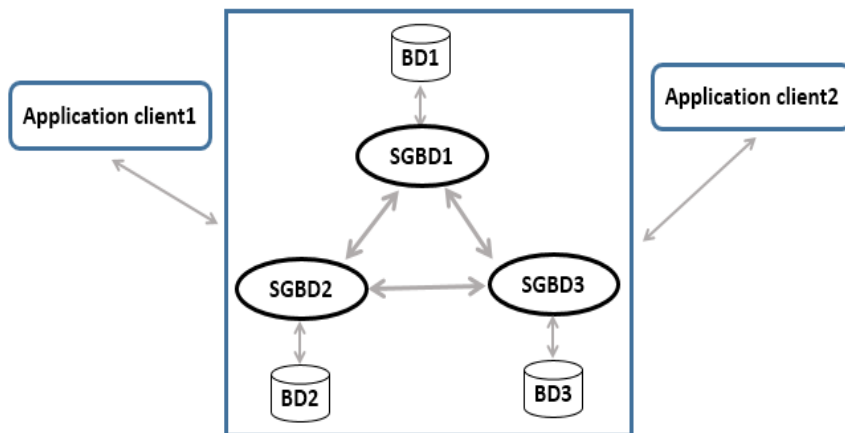


Figure 20: exemple de SGBD réparti

Conclusion du chapitre VII

Dans ce chapitre, nous avons expliqué les différentes architectures de BD, les techniques de fragmentation, les systèmes de BD parallèles ainsi que trois types de méthodes d'accès aux BD.

Conclusion générale

Ce polycopié doit permettre de préparer les apprenants à l'administration d'une BD relationnelle. Le but étant de les initier à utiliser une bonne stratégie de sécurité pour les objets d'une BD oracle grâce à la gestion des droits et des privilèges des utilisateurs.

Pour cette raison, ce document a abordé les concepts liés à une administration de premier niveau de BD oracle. Ce type d'administration concerne la création des objets de la BD ainsi que la définition des droits et des privilèges. Les objets traités dans ce document sont les tables, les utilisateurs, les synonymes, les vues, les séquences, les liens de BD, les tablespaces, les clusters, etc. Les traitements au niveau de la BD sont aussi abordés dans le cadre de ce polycopié à travers la définition des procédures, des fonctions, des triggers, etc. D'autre part, la réplication des données est une technique très utilisée pour réduire le temps de réponse des requêtes de BD. Pour cette raison, ce document fait un tour d'horizon sur les snapshots, les vues matérialisés et les clusters. Enfin, pour répondre aux exigences des entreprises trois types d'architectures de BD sont présentés dans la fin de ce manuscrit.

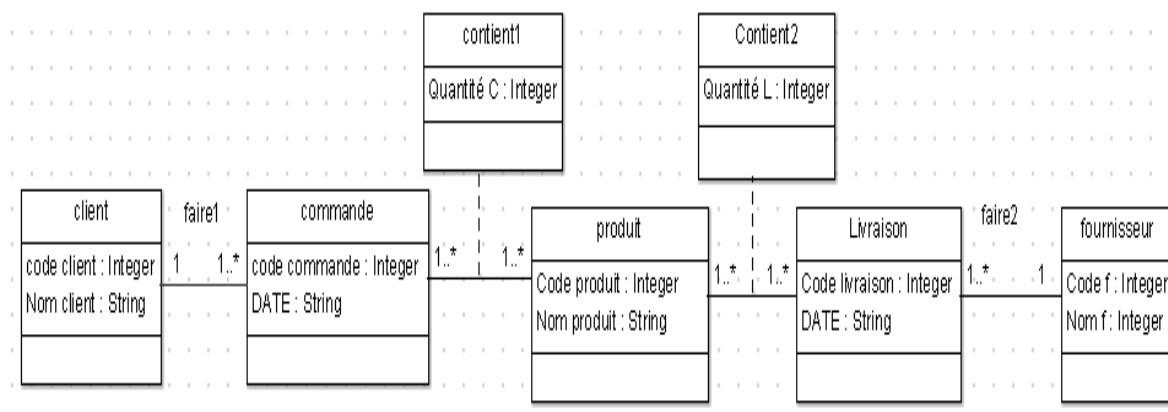
A travers ce polycopié, les apprenants peuvent acquérir des savoir-faire au niveau des tâches à réaliser par un administrateur de BD comme la création des profils d'utilisateurs, la manipulations des tablespaces/clusters et la recommandation d'une architecture de BD à une entreprise. Ces compétences permettent aux apprenants de mettre en œuvre d'une base de données Oracle en s'appuyant sur les standards SQL et PL/SQL.

Comme future perspective à nos lecteurs, nous recommandons de faire des TP de BD réparties pour se familiariser avec l'architecture distribuée qui est facile à mettre en place grâce au SGBD Oracle.

TD 1 : Diagramme UML & Administration

Exercice 1 :

La figure ci-dessous représente une partie du diagramme de classes d'une application de gestion de stock.



Selon le diagramme des cas d'utilisation associé à ce diagramme de classes :

- (1) le client peut passer des commandes.
- (2) le fournisseur peut faire des livraisons
- (3) le Gestionnaire des stocks réceptionne les produits livrés et peut valider ou non les commandes des clients.

1. Quels sont les privilèges à attribuer à l'utilisateur Mohamed pour jouer le rôle du client ?
2. Quels sont les privilèges à attribuer à l'utilisateur ALI pour jouer le rôle du fournisseur ?
3. Quels sont les privilèges à attribuer à l'utilisateur MHAMED pour jouer le rôle du gestionnaire du stock ?

Exercice 2 :

1. Proposez un diagramme de cas d'utilisation qui permet de modéliser les besoins suivants :

- (A) L'**étudiant** peut faire la mise à jour de ses **propres informations** et consulter sa **propre note pour tous les examens de chaque module**.
- (B) Le **responsable de scolarité** peut ajouter, modifier ou supprimer les notes des **examens** effectués durant les deux **semestres** de l'année universitaire pour chaque **étudiant** et pour chaque **module**.
- (C) L'**enseignant** peut ajouter, modifier et supprimer que les notes des examens de son **propre module**.

2. Proposer le diagramme de classe associé à ce diagramme de cas d'utilisation.
3. Traduire ce diagramme de classe en un modèle logique relationnel.
4. Ecrire les instructions SQL pour implémenter ce modèle logique relationnel.
5. Créer les vues correspondantes à chaque type d'utilisateurs (étudiant, enseignant, responsable).
6. A partir du diagramme de cas d'utilisation, donner les instructions SQL qui permettent de décrire les privilèges associés aux différents rôles (les privilèges peuvent concerner des tables et/ou des vues).

TD 2 : Attribution des droits d'administration

Exercice 1

Pour faciliter la proposition, l'affectation et la gestion des projets de fin d'étude (PFE) au niveau de notre département, nous envisageons de réaliser une base de données sous Oracle qui devra nous permettre de :

- (a) Archiver toute les informations qui concernent les Nouveau et les ancien PFE
- (b) Rechercher les données relatives à un PFE donné
- (c) Administrer les droits et les privilèges des différents utilisateurs de la BD.

1. Elaborer le diagramme de classe UML relatif à la gestion des PFE (ce diagramme doit contenir au moins les informations suivantes : enseignants, étudiants, spécialités, domaines de recherche, PFE,...).

2. Etablir le diagramme des cas d'utilisation associé à cette gestion.

3. Ecrire les commandes SQL qui permettent de définir les privilèges associés aux rôles enseignant et étudiant selon les deux diagrammes de la question (1) et la question (2).

Exercice 2

On considère le schéma relationnel suivant : **Enseigne** (id-ens, id-cours)

Enseignant (id-ens, nom, prénom, grade, domaine, niveau)

Etudiant (id-et, nom, prénom, niveau, statut) et **Cours** (id-cours, titre, salle, jour)

a) Quels sont les privilèges nécessaires à attribuer à l'utilisateur **Mohamed** pour exécuter la requête suivante ?

SELECT nom, prenom FROM Enseignant UNION

SELECT nom, prenom FROM Etudiant where niveau = 'doctorat' ;

b) Quels sont les privilèges nécessaires à attribuer à l'utilisateur **Ali** pour exécuter la requête suivante ?

SELECT domaine FROM Enseignant E1, Enseigne E2 WHERE E1.id-ens=E2.id-ens

AND E2.id-cours IN (SELECT id-cours FROM Cours WHERE jour = 'vendredi');

c) Quels sont les privilèges nécessaires à attribuer à l'utilisateur **Mourad** pour exécuter la requête suivante ?

INSERT INTO Enseignant (nom) SELECT DISTINCT nom FROM Etudiant WHERE statut = 'moniteur' AND nom NOT IN (SELECT nom FROM Enseignant)

Exercice 3

Soit le schéma relationnel suivant :

Produit (num_prod, nom_prod, prix_vente),

Stock (num_prod, num_mag, mois, quantiteS),

Vente (num_prod, num_mag, mois, quantiteV)

Magasin (num_mag, nom_mag)

1. Donnez les instructions SQL qui créent les schémas de relation pour les **tables Produit, Stock, Vente et Magasin** avec les contraintes de clés primaires et de clés étrangères.
2. Donnez les instructions SQL qui permettent de créer un **utilisateur Mohamed** et de lui donner la possibilité de **consulter** seulement les **quantités stockées et vendues par produit et par mois**.
3. Donner les instructions SQL qui permettent de créer un **utilisateur Ali** et lui donner la possibilité de **modifier** les **quantités stockées et vendues par produit et par magasins**.

TD 3 : Gestion des autorisations d'accès

Exercice 1

Pour sa gestion, une banque dispose d'une base de données comprenant les relations suivantes :

Clients (id-client, nom, prénom, datenaiss, ville)

Comptes (id-compte, solde)

Client-Comptes (id-client, id-compte)

Écrivez en SQL les commandes que doit effectuer :

1) Le gestionnaire de la base (qui possède tous les droits sur toutes les relations, et la possibilité de transmettre ces droits) pour gérer chacun des cas suivants :

i. Accorder à Jean le droit de lire et mettre à jour les trois relations de la base et de transmettre ces droits ;

ii. Accorder à Paul le droit de mettre à jour les comptes des clients ;

2) Jean, pour accorder à Max le droit de lire la relation Clients ;

3) Le gestionnaire de la base, pour changer les droits de Jean. Il n'a plus le droit d'accéder à la relation Clients. Écrivez en SQL la commande que doit effectuer le gestionnaire pour refléter cette situation. Que se passe-t-il pour les droits accordés à Max ?

Exercice 2

Le graphe des autorisations :

- Un nœud représente un couple (utilisateur, privilège) ;
- Un arc représente une autorisation transmise :
 - Si l'autorisation est transmise avec l'option transmissible (WITH GRANT OPTION), on marque le nœud avec une étoile ;
 - Si l'utilisateur est propriétaire de la relation, on marque le nœud avec 2 étoiles ;

On considère la séquence d'autorisation suivante (Max est propriétaire de R) :

1. Max GRANT INSERT ON R TO Luc, Léa WITH GRANT OPTION
2. Luc GRANT INSERT ON R TO Zoé
3. Luc GRANT INSERT ON R TO Jules WITH GRANT OPTION
4. Léa GRANT INSERT ON R TO Zoé
5. Jules GRANT INSERT ON R TO Léa WITH GRANT OPTION
6. Max REVOKE INSERT ON R FROM Luc

Donner l'état du graphe à l'étape 5 puis à l'étape 6.

On considère maintenant la séquence d'autorisations suivante (A est propriétaire des objets R et S) :

1. A GRANT INSERT ON R, S TO B, C WITH GRANT OPTION
2. B GRANT INSERT ON R TO D
3. C GRANT INSERT ON S TO D, E WITH GRANT OPTION
4. D GRANT INSERT ON S TO B
5. B GRANT INSERT ON S TO E
6. C GRANT INSERT ON R TO E
7. A REVOKE INSERT ON S FROM C
8. A REVOKE INSERT ON R FROM B

Donner l'état des autorisations à l'étape 6 puis à l'étape 8.

TP 0 : LDD & LMD sous Oracle

1. Connexion à la base de données Oracle grâce au compte administrateur :

```
SQL> connect system/inchalah  
Connected.
```

2. Création d'un nouvel utilisateur :

```
SQL> create user omar1 identified by 123;  
User created.
```

3. Assignment du privilège Ressource à l'utilisateur Omar1 :

```
SQL> grant resource to omar1;  
Grant succeeded.
```

4. Connexion de l'utilisateur Omar1 à son schéma de la base de données Oracle

```
SQL> connect omar1/123;  
Connected.
```

5. Création de la table relationnelle employé :

```
SQL> create table employe (code number primary key, nom varchar2(20), DN date);
```

6. Création d'une vue employé

```
SQL> connect system/inchalah  
Connected.  
SQL> grant create view to omar1;  
Grant succeeded.  
  
SQL> connect omar1/123;  
Connected.  
SQL> create view v_employe (code,nom)  
2 as select code,nom  
3 from employe  
4 where code=1;  
View created.
```

7. Insertion des données dans la table employé :

```
SQL> insert into employe values (1,'a','01/01/2012');  
1 row created.  
  
SQL> insert into employe values (2,'b','02/01/2012');  
1 row created.  
  
SQL> insert into employe values (3,'c','03/01/2012');  
1 row created.
```

8. Description de la table employé et de la vue employé

```
SQL> desc employe;
```

Name	Null?	Type
CODE	NOT NULL	NUMBER
NOM		VARCHAR2(20)
DN		DATE

```
SQL> desc v_employe;
```

Name	Null?	Type
CODE	NOT NULL	NUMBER
NOM		VARCHAR2(20)

9. Manipulation des vues relationnelles

```
SQL> insert into v_employe values (4,'c');
1 row created.

SQL> insert into v_employe values (5,'c');
1 row created.

SQL> select * from v_employe;

   CODE NOM
-----
      1  a

SQL> create view vemploye (code,nom)
  2 as select code,nom
  3 from employe
  4 ;

View created.

SQL> select * from vemploye;

   CODE NOM
-----
      1  a
      2  b
      3  c
      4  c
      5  c

SQL> create view vemp (code,nom)
  2 as select code,nom
  3 from employe;

View created.
```

```
SQL> create view vemp2 (code,nom)
  2 as select code,nom
  3 from employe
  4 with read only;

View created.

SQL> insert into vemp2 values (5,'c');
insert into vemp2 values (5,'c')
*
ERROR at line 1:
ORA-42399: cannot perform a DML operation on a read-only view
```

TP 1 : Création d'utilisateur & connexion

L'objectif de ce TP est de créer un utilisateur et lui associer le privilège de création de table puis de tester ce privilège.

1. Avec SYSTEM, créer un utilisateur nommé ilisia et donnez-lui les privilèges nécessaire à sa connexion à la BD.
2. Avec SYSTEM, donner le privilège de création de table à l'utilisateur nommé ilisia et essayez de créer une table pour tester ce privilège.
3. Avec SYSTEM, donner le rôle « resource » à l'utilisateur nommé ilisia et essayez de créer une table pour tester le privilège de création de table.

Correction

1. Avec SYSTEM, créer un utilisateur nommé ilisia et donnez-lui les privilèges nécessaire à sa connexion à la BD.

```
SQL> connect system/inchalah;
Connecté.
SQL> create user isiaail identified by "123";
Utilisateur créé.
```

```
SQL> grant create session to isiaail;
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> connect isiaail/123;
Connecté.
```

2. Avec SYSTEM, donner le privilège de création de table à l'utilisateur nommé ilisia et essayez de créer une table pour tester ce privilège.

```
SQL> connect system/inchalah;
Connecté.
SQL> grant create table to isiaail;
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> connect isiaail/123;
Connecté.
SQL> create table test (code number);
create table test (code number)
*
ERREUR ó la ligne 1 :
ORA-01950: pas de privilèges sur le tablespace 'SYSTEM'
```



```
SQL> grant resource to isiail;  
grant resource to isiail  
*  
ERREUR ó la ligne 1 :  
ORA-01031: privilèges insuffisants
```

3. Avec SYSTEM, donner le rôle « resource » à l'utilisateur nommé isia et essayez de créer une table pour tester le privilège de création de table.

```
SQL> connect system/inchallah;  
Connecté.  
SQL> grant resource to isiail;  
  
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> connect isiail/123;  
Connecté.
```

```
SQL> create table test (code number);  
  
Table créée.
```

TP 2 : Création de vues

L'objectif de ce TP est de créer et manipuler des vues simples à partir d'une table.

1. Avec SYSTEM, créer une table nommée emp(code number(3),nom varchar(20), prenom varchar(30), gain number (6,2), departement varchar(30));
2. Créer un utilisateur nommé « (il/isia) + tp3 » puis connectez le.
3. Donner à cet utilisateur la possibilité d'insérer les lignes dans la table emp.
4. Donner à cet utilisateur la possibilité de consulter la table emp.
5. Avec l'utilisateur il/isia+tp3, créer la vue G_emp (code,nom,prenom,gain) en lecture seule et vérifier que cette vue n'autorise pas des insertions.
6. Avec l'utilisateur il/isia+tp3, créer la vue G_emp2 (code,nom,prenom, departement) et vérifier que cette vue autorise des insertions.
7. Avec l'utilisateur il/isia+tp3, créer la vue G_emp3 qui permet de calculer le montant global des augmentations de gains (taux_augmentation = 5%)
8. Avec l'utilisateur il/isia+tp3, créer la vue G_emp4 qui permet de visualiser le nouveau gain de chaque employé (taux_augmentation = 5%)
9. Avec SYSTEM, afficher le contenu de la vue G_emp. Que remarquez-vous ?

Correction

1. Avec SYSTEM, créer une table nommée emp(code number(3),nom varchar(20), prenom varchar(30), gain number (6,2), departement varchar(30));

```
SQL> connect system/inchalah;
Connecté.
```

```
SQL> create table emp
2  (code number(3),
3   nom  varchar(20),
4   prenom varchar(30),
5   gain number(6,2),
6   departement varchar(30));
```

```
Table créée.
```

2. Créer un utilisateur nommé « (il/isia) + tp03 » puis connectez le.

```
SQL> connect system/inchalah;
Connecté.
```

```
SQL> create user ilisia identified by "123";
Utilisateur créé.
```

```
SQL> grant create session to ilisia;
Autorisation de privilèges (GRANT) acceptée.
```

2. Donner à cet utilisateur la possibilité d'insérer des lignes dans la table emp.

```
SQL> connect system/inchalah;
Connecté.
SQL> insert into emp values (1,'n1','p1',5,'dpt1');
1 ligne créée.
SQL> insert into system.emp values (2,'n2','p2',10,'dpt2');
1 ligne créée.
```

```
SQL> grant insert on emp to ilisia;
Autorisation de privilèges (GRANT) acceptée.
SQL> insert into emp values (3,'n3','p3',10,'dpt3');
1 ligne créée.
SQL> insert into system.emp values (1,'n1','p1',5,'dpt1');
1 ligne créée.
```

4. Donner à cet utilisateur la possibilité de consulter la table emp.

```
SQL> connect ilisia/123;
Connecté.
SQL> select * from emp;
select * from emp
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> select * from system.emp;
select * from system.emp
      *
ERREUR ó la ligne 1 :
ORA-01031: privilèges insuffisants

SQL> connect system/inchalah;
Connecté.
SQL> grant select on emp to ilisia;
Autorisation de privilèges (GRANT) acceptée.

SQL> connect ilisia/123;
Connecté.
SQL> select * from emp;
select * from emp
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> select * from system.emp;
```

5. Avec l'utilisateur il/isia+tp3, créer la vue G_emp (code,nom,prenom,gain) en lecture seule et vérifier que cette vue n'autorise pas des insertions.

```
SQL> connect ilisia/123;
Connecté.
SQL> create view G_emp as
  2 select code,nom,prenom,gain
  3 from emp
  4 with read only;
from emp
*
```

```
ERREUR ó la ligne 3 :
ORA-00942: Table ou vue inexistante
```

```
SQL> create view G_emp as
  2 select code,nom,prenom,gain
  3 from system.emp
  4 with read only;
```

Vue cr  e.

```
SQL> insert into G_emp values (5,'n5','p5',5);
insert into G_emp values (5,'n5','p5',5)
*
ERREUR ó la ligne 1 :
ORA-01733: les colonnes virtuelles ne sont pas autoris  es ici
```

```
SQL> select * from G_emp;
```

CODE	NOM	PRENOM	GAIN
1	n1	p1	5
2	n2	p2	10
3	n3	p3	10
1	n1	p1	5

6. Avec l'utilisateur il/isia+tp3, cr  er la vue G_emp2 (code,nom,prenom, departement) et v  rifier que cette vue autorise des insertions.

```
SQL> create view G_emp2 as
  2 select code,nom,prenom,departement
  3 from system.emp
  4 ;
```

Vue cr  e.

```
SQL> insert into G_emp2 values (5,'n5','p5','dep5');
1 ligne cr  e.
```

```
SQL> connect ilisia/123;
Connect  .
SQL> select * from G_emp2;
```

CODE	NOM	PRENOM	DEPARTEMEN
1	n1	p1	dpt1
2	n2	p2	dpt2
3	n3	p3	dpt3
1	n1	p1	dpt1
5	n5	p5	dep5

7. Avec l'utilisateur il/isia+tp3, créer la vue G_emp3 qui permet de calculer le montant global des augmentations de gains (taux augmentation = 5%)

```
SQL> create view G_emp3 as
  2 select code,nom,prenom,gain * 0.05 GAIN_OBTENU
  3 from system.emp;
```

Vue créée.

```
SQL> select * from G_emp3;
```

CODE	NOM	PRENOM	GAIN_OBTENU
1	n1	p1	,25
2	n2	p2	,5
3	n3	p3	,5
1	n1	p1	,25
5	n5	p5	

```
SQL> select sum(GAIN_OBTENU) from G_emp3;
```

```
SUM(GAIN_OBTENU)
-----
1,5
```

8. Avec l'utilisateur il/isia+tp3, créer la vue G_emp4 qui permet de visualiser le nouveau gain de chaque employé (taux augmentation = 5%)

```
SQL> create view G_emp4 as
  2 select code,nom,prenom,gain * 1.05 GAIN_OBTENU
  3 from system.emp;
```

Vue créée.

```
SQL> select * from G_emp4;
```

CODE	NOM	PRENOM	GAIN_OBTENU
1	n1	p1	5,25
2	n2	p2	10,5
3	n3	p3	10,5
1	n1	p1	5,25
5	n5	p5	

9. Avec SYSTEM, afficher le contenu de la vue G_emp. Que remarquez-vous ?

```
SQL> connect system/inchalah;
Connecté.
```

```
SQL> select * from G_emp2;
select * from G_emp2
*
```

```
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante
```

```
SQL> select * from ilisia.G_emp2;
```

CODE	NOM	PRENOM	DEPARTEMEN
1	n1	p1	dpt1
2	n2	p2	dpt2
3	n3	p3	dpt3
1	n1	p1	dpt1
5	n5	p5	dep5

TP 3 : Attribution des droits d'administration

A la fin de ce TP, vous serez capable d'attribuer et retirer des privilèges à des objets de la BD.

1. Créer la table dept (numdpt,nomdpt).
2. Accorder le droit de SELECT sur la table DEPT à tous les utilisateurs.
3. Accorder les droits INSERT et UPDATE sur DEPT à l'utilisateur MOMO.
4. Accorder tous les droits sur TEMPLOYE (numéro, nom, titre, numéro de département) MOMO.
5. Créer une vue relative v_t_employe (nom, titre, numéro de département) des employés. Puis accorder les droits de SELECT sur la vue à l'utilisateur MOMO.
6. Retirer le droit INSERT à l'utilisateur MOMO, sur la table DEPT.
7. Retirer le droit SELECT sur la table DEPT à l'utilisateur momo.
8. Retirer le droit SELECT sur la table DEPT à tous les utilisateurs.
9. Retirer le droit select à l'utilisateur MOMO, sur la vue v_t_employe.
10. Retirer le droit de suppression sur la table département à tous les utilisateurs.

Correction :

```
SQL> connect system/inchalah;
Connecté.
SQL> create table Dept
  2  (numdept number(2) primary key,
  3  nomdept varchar(20));
Table créée.
```

```
SQL> grant select on dept to public;
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> create user momo identified by "123";
Utilisateur créé.
```

```
SQL> grant insert,update on dept to momo;
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> create table t_employe
  2  (numemploye number(3) primary key,
  3  nomemploye varchar(25),
  4  titre varchar(20),
  5  numdept number(3) references dept(numdept));

SQL> grant all on employe to momo;

Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> create view v_t_employe
  2  as select nomemploye,titre,numdept
  3  from t_employe;

Vue créée.
```

```
SQL> grant select on v_t_employe to momo;

Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> revoke SELECT on dept from public;

Suppression de privilèges (REVOKE) acceptée.
```

```
SQL> REVOKE INSERT on EMPLOYE from MOMO;

Suppression de privilèges (REVOKE) acceptée.
```

```
SQL> revoke DELETE on DEPT from public;
revoke DELETE on DEPT from public
*
ERREUR ó la ligne 1 :
ORA-01927: impossible de révoquer (REVOKE) des privilèges non accordés
```

TP 4 : Gestion des autorisations d'accès

A la fin de ce TP, les apprenants vont se familiariser avec l'attribution et la révocation des autorisations d'accès aux différents objets de la BD.

Pour sa gestion, une banque dispose d'une BD comprenant les relations suivantes :

Clients (id-client, nom, prénom, datenaiss, ville)

Comptes (id-compte, solde)

Client-Comptes (id-client, id-compte)

Écrivez en SQL les commandes que doit effectuer :

1) Le gestionnaire de la base (qui possède tous les droits sur toutes les relations, et la possibilité de transmettre ces droits) pour gérer chacun des cas suivants :

i. Accorder à Jean le droit de **lire et modifier les trois relations** de la base et de **transmettre ces droits** ;

ii. Accorder à Paul le droit de **modifier les comptes des clients** ;

2) Jean, pour accorder à **Max** le droit de **lire** la relation **Clients** ;

3) Le gestionnaire de la base retire le droit d'accéder à la relation **Clients** à **Jean**. Que se passe-t-il pour les **droits accordés à Max** ?

Correction

```
SQL> grant select,update on clients to jean with grant option ;
Autorisation de privilèges (GRANT) acceptée.

SQL> grant select,update on comptes to jean with grant option ;
Autorisation de privilèges (GRANT) acceptée.

SQL> grant select,update on clientcomptes to jean with grant option ;
Autorisation de privilèges (GRANT) acceptée.

SQL> grant update on clients to Paul;
Autorisation de privilèges (GRANT) acceptée.
```



```

SQL> connect jean/123;
Connecté.
SQL> grant select on clients to MAX;
grant select on clients to MAX
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> grant select on system.clients to MAX;
Autorisation de privilèges (GRANT) acceptée.
SQL> select * from session_privs;

PRIVILEGE
-----
CREATE SESSION

SQL> connect system/inchalah
Connecté.
SQL> revoke select,update on clients from jean;
Suppression de privilèges (REVOKE) acceptée.

SQL> connect max/123;
Connecté.
SQL> select * from clients;
select * from clients
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> select * from system.clients;
select * from system.clients
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

SQL> connect paul/123;
Connecté.
SQL> select * from system.clients;
select * from system.clients
      *
ERREUR ó la ligne 1 :
ORA-01031: privilèges insuffisants

SQL> update clients
      2 set ville = "tlemcen";
update clients
      *
ERREUR ó la ligne 1 :
ORA-00942: Table ou vue inexistante

```

```
SQL> update system.clients  
2 set ville = 'tlemcen';
```

1 ligne mise á jour.

```
SQL> connect system/inchalah;  
Connecté.
```

```
SQL> create table clients123 (  
2 idclient number,  
3 nom varchar(20),  
4 prenom varchar(30),  
5 datenaiss date,  
6 ville varchar(30));
```

Table créée.

```
SQL> create user jean123 identified by "123";
```

Utilisateur créé.

```
SQL> create user paul123 identified by "123";
```

Utilisateur créé.

```
SQL> create user max123 identified by "123";
```

Utilisateur créé.

```
SQL> grant create session to jean;
```

Autorisation de privilèges (GRANT) acceptée.

```
SQL> grant create session to paul;
```

Autorisation de privilèges (GRANT) acceptée.

```
SQL> grant create session to max;
```

Autorisation de privilèges (GRANT) acceptée.

```
SQL> grant select,update on clients123 to jean with grant option;
```

Autorisation de privilèges (GRANT) acceptée.

```
SQL> connect jean/123;
```

Connecté.

```
SQL> select * from system.clients123;
```

aucune ligne sélectionnée

```
SQL> connect jean/123;
```

Connecté.

```
SQL> grant select on system.clients123 to max;  
Autorisation de privilèges (GRANT) acceptée.
```

```
SQL> connect max/123;  
Connecté.  
SQL> select * from system.clients123;  
  
aucune ligne sélectionnée
```

```
SQL> connect system/inchallah;  
Connecté.  
SQL> revoke select on clients123 from jean;  
Suppression de privilèges (REVOKE) acceptée.
```

```
SQL> connect max/123;  
Connecté.  
SQL> select * from system.clients123;  
select * from system.clients123  
*  
ERREUR à la ligne 1 :  
ORA-00942: Table ou vue inexistante
```

```
SQL> connect jean/123;  
Connecté.  
SQL> select * from system.clients123;  
select * from system.clients123  
*  
ERREUR à la ligne 1 :  
ORA-01031: privilèges insuffisants
```

TP 5 : Utilisation des cursors et des triggers

A la fin de ce TP, les apprenants seront capables de bien utiliser un curseur et un trigger dans une BD.

Soit la table test001 (num, name, Sal en DA)

NUM	NAME	SAL
1	dennouni	800
2	slimane	1000
3	hadjheni	1000
3	aridj	2000
3	loukam	2000

- 1) Afficher à l'aide du PL/SQL les infos (num, nom et salaire en euro) relatives à un employé identifié par code donné
- 2) Créer un trigger qui convertit le salaire en euro après l'insertion d'un nouvel employé.

Correction

1. Les infos (num, nom et salaire en euro) relatives à un employé identifié par code donné.

```
declare
x test001%rowtype;
BEGIN
SELECT * into x from test001 where num=:m ;
  dbms_output.put_line (x.num);
  dbms_output.put_line (x.name);
  dbms_output.put_line (x.sal/200||'euro');
END;
```

127.0.0.1:8081/apex/f?p=4500:138:2589866483412522::

Soumettre

:M 1

Résultats Expliquer Décrire SQL enregistré Historique

```
1
dennouni
4euro
```

Instruction traitée.

127.0.0.1:8081/apex/f?p=4500:138:2589866483412522::

Soumettre

:M 3

Résultats Expliquer Décrire SQL enregistré Historique

ORA-01422: l'extraction exacte ramène plus que le nombre de lignes demandé

```

DECLARE
  CURSOR c1 is select * from test001 where num=:m ;
  v_num NUMBER(4); v_name VARCHAR2(10); v_sal  NUMBER(7,2);
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_num, v_name, v_sal;
    EXIT WHEN c1%NOTFOUND;
    dbms_output.put_line (v_num);
    dbms_output.put_line (v_name);
    dbms_output.put_line (v_sal/200||'euro');
    COMMIT;
  END LOOP;
  CLOSE c1;
END;

```

2. Créer un trigger qui convertit le salaire en euro après l'insertion d'un nouvel employé.

```

CREATE OR REPLACE TRIGGER Trig001
  BEFORE INSERT ON test001
  FOR EACH ROW
DECLARE
BEGIN
  :new.sal:=:new.sal/200;
END Trig001;

```

```
insert into test001 values (3,'tayebi',2500);
```

```
select * from test001 ;
```

Résultats Expliquer Décrire SQL enregistré Historique

NUM	NAME	SAL
1	dennouni	800
2	slimane	1000
3	hadjheni	1000
3	aridj	2000
3	loukam	2000
3	tayebi	12,5

Accès e-learning au module

Dans ce qui suit, nous présentons les instructions qui permettent aux apprenants d'accéder aux ressources et aux activités relatives au module AABD « Architecture et administration des BD »

1. Accédez au site de l'université <http://elearning.univ-chlef.dz/fr/login/>



2. Introduire votre mot de passe en connexion anonyme

Vous possédez déjà un compte ?

Connectez-vous ici en utilisant votre nom d'utilisateur et mot de passe
(Votre navigateur doit supporter les cookies)🔗

Nom d'utilisateur

n.denouni

Mot de passe

Connexion

Vous avez oublié votre nom d'utilisateur et/ou votre mot de passe ?

Les visiteurs anonymes peuvent accéder à certains cours

➡ Connexion anonyme

3. Après identification, les interfaces suivantes vous permettent de choisir le cours AABD.



Catégories de cours:

Faculté des Sciences Exactes et d'Informatique

Sous-catégories

Tronc-Commun Sciences de la Nature et de la Vie
Tronc-Commun Informatique et Mathématique
Tronc-Commun Sciences de la Matière
Département Sciences Biologiques
Département Informatique
Département Mathématique
Département Physique
Département Chimie

Administration des Bases de données

Enseignant: Nassim DENNOUNI

3

Ce cours vise à introduire les notions de base pour l'administration des Base de données dans un environnement réseau sous Oracle



Bibliographie

- Abiteboul, S., Nguyen, B., & Rigaux, P. (2016). Bases de données Distribuées.
- Cabanac, G. (2016). Programmation et administration des bases de données.
- Dennouni, N. (2017, avril). *Base de données avancées*. Récupéré sur www.researchgate.net:
https://www.researchgate.net/profile/Nassim_Dennouni/publication/324504264_Support_de_cours_Base_de_donnees_avancees/links/5ad0c7fe0f7e9b285930e0ba/Support-de-cours-Base-de-donnees-avancees.pdf
- Donez, D. (2003). Les SGBD Parallèles.
- Dupire, J. (2004). Architectures multiprocesseurs.
- Godin, R., & Desrosiers, C. (2011). LOG660 - Bases de données de haute performance BD parallèles et réparties. Département de génie logiciel et des TI.
- HAMDANE, M. E. (2018). Bases de données avancées : Du relationnel au NoSQL: Cours et travaux pratiques.
- J.Chevance, R. (2003, mars). base de données distribuées et fédérées .
- Meslé, A. (2011). Introduction au PL/SQL Oracle.
- Moussa, R. (2006). *Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle*. Carthage: Ecole Supérieure de Technologie et d'Informatique à Carthage.
- Oracle. (2019). <https://docs.oracle.com/>. Récupéré sur <https://docs.oracle.com/>:
<https://docs.oracle.com>
- Sans, V. (2000). *BASE DE DONNÉES OBJET*. Université de RENNE 1.
- Tony, A. (2014). Cours SQL.
- Tounsi, N. (2018, 10 20). <http://www.emi.ma/ntounsi/COURS/>. Récupéré sur <http://www.emi.ma>:
<http://www.emi.ma/ntounsi/COURS/>
- Wikipédia. (2016). *Base de données relationnelle*. Récupéré sur
http://fr.wikipedia.org/w/index.php?title=Base_de_donn%C3%A9es_relationnelle&action=history:
http://fr.wikipedia.org/w/index.php?title=Base_de_donn%C3%A9es_relationnelle&oldid=123523239
- wikipedia. (2018, 11 23). *Oracle OLAP*. Récupéré sur wikipedia:
https://en.wikipedia.org/wiki/Oracle_OLAP