Université de l'Unité Africaine (UUA)
Année universitaire : 2020 – 2021
Filière: Informatique de Gestion
Niveau: 2ème année de Licence (IG2)
Module : Gestion de base de données Oracle
Enseignant : M. SANA

Gestion de base de données Oracle

Chap 2 : Vues - Séquences - Index

Enseignant : M. SANA

Table des matières

I. Les vues

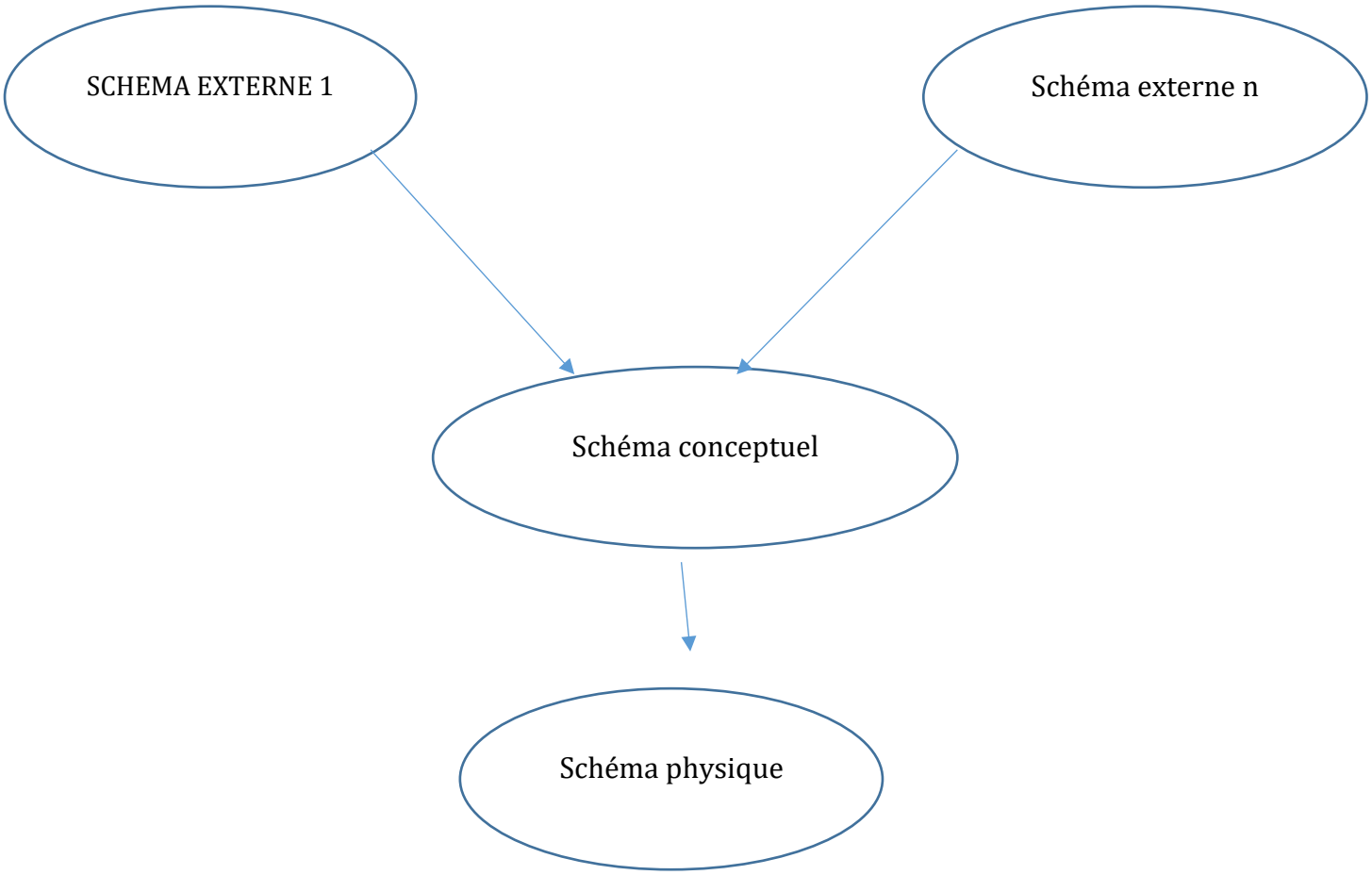
Introduction
Définition d'une vue
- Création
- Suppression
- Renommage
- Interrogation

Utilités des vues
Mise à jour au travers des vues
Cas d'oracle

I.1. Introduction

- Groupe ANSI/X3/SPARC (1975)

3 niveaux d'abstraction



I.2. Définition d'une vue

- Une vue est le résultat d'une requête à laquelle on a donné un nom :
  - Elle est construite à partir du résultat d'un SELECT.
- C'est une table virtuelle
  - Ensemble de tuples qui n'existes pas physiquement
  - Calculable à l'exécution
- La vue sera vue par l'utilisateur comme une table réelle.
- Peut-être utilise pour définir une autre vue
- Le nom d'une vue peut être utilisé partout où on peut mettre le nom d'une table :
  - SELECT, UPDATE, DELETE, INSERT, GRANT

a. Création d'une vue : Syntaxe Générale

CREATE VIEW nom-de-la-vue [Liste d'attributs]

AS <REQUETE d'interrogation SELECT>

[WITH CHECK OPTION]

- La spécification des noms des attributs de la vue est facultative.
- Par défaut, les attributs de la vue ont pour nom les noms des attributs résultant de la requête SELECT.
- La clause WITH CHECK OPTION empêche que l'utilisateur ajoute ou modifie dans une vue des enregistrements non conformes à la définition de la vue.

- **Exemple 1 :** Créer une vue des étudiants du département informatique et statistique:

CREATE VIEW etudiant_vue_par_LesProfs (num_etudiant, nom, prenom, filiere, niveau)

AS SELECT * FROM Etudiants

WHERE filiere = "SID";

- **Exemple 2 :** Créer une vue des employés du département 10 (NUM, NOM, fonction)

Emp (Num, Nom, fonction, Numsup, Embauche, SAL, Comm, DEPTNO)

CREATE VIEW empvu10

AS SELECT Num, Nom, fonction

FROM emp

WHERE deptno = 10;

- **Exemple 3 :** Créer une vue à partir de plusieurs tables :
  - Exemple : Créer une vue comportant le nom des employés, le nom du service dans lequel ils exercent leurs fonctions et le lieu du travail
  - Soit le schéma relationnel suivant :

- Emp (Num, Nom, Fonction, NumSup, Embauche, Salaire, Deptno)
- DEPT (DeptNo, Nom, Loc.)

CREATE VIEW empDept
AS SELECT e.nom, d.nom, LOC
FROM EMP e, DEPT d
WHERE e. DEPTNO = d. DEPTNO;

b. Suppression d'une Vue

Syntaxe : **DROP VIEW** nom_de_la_vue ;

- La suppression d'une vue n'entraine pas la suppression des données.
- Toutes les vues qui utilisent cette vue sont automatiquement détruites,
  Elles sont également supprimées.
- Les vues figurent dans les tables systèmes ALL_CATALOG, USER_VIEWS et ALL_VIEWS

- USER_*: Ces tables systèmes décrivent les objets qui appartiennent à l'utilisateur courant c'est-à-dire l'utilisateur qui est actuellement connecté.
- ALL_* : Ces tables systèmes décrivent les objets qui sont accessibles à l'utilisateur courant.
- DBA_* : Ces tables systèmes décrivent tous les objets (ces vues ne sont accessibles que par l'administrateur)

Exemple 1 : **DROP VIEW** xyz ;

Exemple 2 : **DROP VIEW** etudiant_vue_par_LesProfs ;

c. Renommage d'une vue

Syntaxe : **RENAME** ancien_nom **TO** nouveau_nom ;

Exemple 1 : **RENAME** xyz **TO** abc ;

Exemple 2 : **RENAME** etudiant_vue_par_LesProfs **TO** etp ;

d. Interrogation des vues:

- Pour récupérer les données d'une vue, on procédera comme si l'on était en face d'une table classique.

Exemple 1 : SELECT * FROM abc ;

- En réalité, cette table est virtuelle. Elle est reconstruite à chaque appel de la vue « abc » par exécution du SELECT constituant la définition de la vue.
- En effet, le contenu d'une vue est recalculé à chaque utilisation de la vue par SQL

Ceci est équivalent côté SGBD à rechercher dans la méta-base (le dictionnaire de données) la définition de la vue « abc » et à transformer la requête initiale.

I.3. Utilités des vues

- Des accès simplifiés aux données :
  - Effet macro : remplacer une requête compliquée nécessitant plusieurs étapes par des requêtes plus simples.
  - Masquer la complexité du schéma de la base de données.
- Une indépendance logique des données ;
- La confidentialité des données ;
- L'intégrité de la base de données.

a. Assurer la confidentialité des données

- Grâce aux vues, il est possible :
  - D'émettre des restrictions d'accès en fonction du contexte
  - De restreindre les accès à un certains attributs ou à certains tuples.

Exemple 1 : Créer une vue et octroyer les droits aux utilisateurs concernés par cette vue (un utilisateur peut visualiser leurs propre infos)

Personne1 (num_pers, nom, salaire, prime_mois, adress, tel, #numero_dep)

1) Chaque utilisateur peut visualiser un seul tuple de la relation : celui qui le concerne

On peut utiliser la variable USER : **une variable d'environnement** qui contient le nom de l'utilisateur qui démarre la session qui lance le programme d'application.

2) Donner les privilèges de lecture sur la vue à chaque utilisateur :

GRANT SELECT ON xyz **TO PUBLIC;**

- Encore mieux donner les privilèges de modification sur certaines colonnes de la vue (adresse, tel) :

- Exemple 2 : Employé (Nss, nom, prénom, adress, tel, salaire)
  - **CREATE VIEW** v_info_privee_employe
    AS SELECT * from Employé **WHERE** upper (nom) = upper (user);

GRANT SELECT, UPDATE (adresse, tel)
ON v_info_privee_employe **TO PUBLIC;**

- Restreindre l'accès à certaines colonnes personnelles ;

- **Exemple 8 :** donner au service comptable d'une entreprise la possibilité du suivi des salaires de façon anonyme (sans savoir le nom) pour les titulaires de grands salaires.

- L'utilisateur service_comptable ne peut connaitre ni le nom, ni le prénom, ni l'adresse des employés gagnant plus de 10000 d

1) **CREATE VIEW** employés_riches
   AS SELECT numero, salaire,
   FROM Emp
   WHERE salaire>10000
2) Accorder le privilège au service_comptable.
   GRANT SELECT ON employés_riches **TO** service_comptable

b. Assurer indépendence logique

- Les applications utilisant les tables de la base de données peuvent ne pas être modifiées si on change le schéma de la table.
- Grâce aux vues, les applications peuvent être insensibles au changement du schéma logique de la base de données.

I.4. MISE A JOURS A TRAVERS DES VUES

Pour qu'une vue soit modifiable, il faut respecter les conditions suivantes :

1. L'expression de la table associée à la vue doit être un simple SELECT, elle ne peut donc contenir les termes JOIN, INTERSECT, UNION ou EXCEPT/ MINUS ;
2. La clause FROM ne peut contenir qu'une seule table de base ou une vue elle-même modifiable ;
3. L'expression SELECT ne peut contenir la clause DISTINGT ;
4. La liste des colonnes du SELECT ne peut comporter d'expression ;
5. Si le SELECT contient une requête imbriquée celle-ci ne peut faire référence a la même table que la requête externe ;
6. La requête SELECT ne peut contenir ni GROUP BY, ni HAVING ;

Si une de ces conditions n'est pas remplie, la vue n'est pas modifiable :

Impossible d'utiliser les commandes INSERT INTO, DELETE ou UPDATE.

I.5. Cas de oracle

- Syntaxe de création :
  CREATE [OR REPLACE] [[NO] FORCE] VIEW [schema.] <NomVue>
  [**WITH { READ ONLY |**
  CHECK OPTION [CONSTRAINT nomContrainte] }];

    - **OR REPLACE :** remplace la vue par la nouvelle définition même si elle existait déjà (évite de détruire la vue avant de la recréer).
    - **FORCE** : pour créer la vue sans vérifier si les sources qui l'alimentent existent, ou si les privilèges adéquats (SELECT, INSERT, UPDATE, ou DELETE) sur ces objets ont acquis par l'utilisateur qui crée la vue. Par défaut c'est NO FORCE qui est utilisée.
    - **WITH READ ONLY** : déclare la vue non modifiable par INSERT, UPDATE ou DELETE
    - **WITH CHECK OPTION** : garantit que toute mise à jour de la vue par INSERT ou UPDATE s'effectuera conformément à la définition de la vue.
    - **CONSTRAINT** nomContrainte : nomme la clause CHECK OPTION sous la forme d'un nom de contrainte.

Exemple 1 :

```
SQL>  CREATE OR REPLACE VIEW  empvu20
      AS SELECT *
      FROM emp
      WHERE deptno= 20
      CHECK OPTION CONSTRAINT empvu20_ck;
```

Toute tentative de modification du numéro de département dans une ligne de la vue échouera, car elle transgresse la contrainte WITH CHECK OPTION.

Exemple 2 :

```
SQL>  CREATE  OR REPLACE VIEW  empvu10
      (employee_number, employee_name, job_title)
      AS SELECT empro, ename, job
      FROM emp
      WHERE deptno= 10
      WITH READ ONLY;
```

Toute tentative d'exécution d'un ordre du LMD sur une ligne de la vue génère l'erreur Oracle Server ORA-01752 → Refus des Ordres du LMD

I.6. Vue sous oracle

Exemple 9 :

```
SQL>  CREATE  OR REPLACE VIEW  empvu20
      AS SELECT *
      FROM emp
      WHERE deptno= 20
      CHECK OPTION [ CONSTRAINT  empvu20_ck;
View created.
```

Toute tentative de modification du numéro de département dans une ligne de la vue échouera, car elle transgresse la contrainte WITH CHECK OPTION.

Exemple 10:

```
SQL>  CREATE  OR REPLACE VIEW  empvu10
      (employee_number, employee_name, job_title)
      AS SELECT empro, ename, job
      FROM emp
      WHERE deptno= 10
      WITH READ ONLY;

       View created.
```

Toute tentative d'exécution d'un ordre du LMD sur une ligne de la vue généré l'erreur Oracle Server ORA-01752 → Refus des Ordres du LMD

- **Soit le schéma relationnel considère dans les exemples suivants :**
  - Compagnie (comp, rue, ville, nomComp)
  - Pilote (brevet, nom, nbhvol, #compa)

Compagnie

comp	ville	Nomcomp
AS	Dakar	Air Sénégal
RA	Kigali	Rwanda Airway

Pilote

Brevet	nom	Prénom	nbHVol	compa
PL-1	Kaboré	Nematou	450	AS
PL-2	Dicko	Amadou	900	AS
PL-3	Simporé	Olivier	1000	RA

- **Vues modifiables**
    - **Pour qu'une vue soit modifiable, sa définition doit respecter les critères suivants :**
    - **Pas de directive DISTINCT**
    - **Pas de fonction de calcul (AVG, COUNT, MAX, MIN, STDDEV, SUM, ou VARIANCE),**
    - **Pas de GROUP BY, ORDER BY, HAVING ou CONNECT BY.**
    - **Pas d'operations ensemblistes (MINUS, UNION[ALL], INTERSECT).**
    - **Pas de START WITH, CONNECT BY**

- **Vues monotables**

- **Exemple 11 : soit la vue état-civil définit comme suit :**

CREATE VIEW Etat_civil

AS SELECT nom, nbHvol, adresse, compa

FROM Pilote

Dites si les opérations suivantes sont autorisées ?

- Suppression des pilotes de la compagnie ASO de la vue Etat_civil

  DELETE FROM Etat_civil
  WHERE compa = 'RA' ; → autorisée

- Le pilote Yassin double ses heures

  UPDATE Etat_civil

  SET nbHvol = nbHvol * 2 → autorisée

  WHERE nom = 'Kaboré ;

- Impact de l'option WITH CHECK OPTION

- Exemple 12: Créer une vue des pilotes appartenant à la compagnie aérienne 'AF'

  CREATE OR REPLACE VIEW pilotesAF

  AS SELECT * FROM pilote

  WHERE compa = 'AS';

- Modification et suppression à travers les vues multi-tables : notion de table protégée :
    - Une table est dite protégée par sa clé (key preserved) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multi-table (peut jouer le rôle de clé primaire de la vue).
- **Résumé condition des mises à jour des vues multi-tables sous Oracle**

Pour qu'une vue multi-table soit modifiable, sa requête de définition doit respecter les critères suivants :

- La mise à jour (INSERT, UPDATE, DELETE) n'affecte qu'une seule table.
- Seuls des renseignements de la table protégée peuvent être insérés.
- Si la clause WITH CHECK OPTION est utilisée, aucune insertion n'est possible (message d'erreur : ORA-01733)
- Les colonnes virtuelles ne sont pas autorisées ici.
- Seules les colonnes de la table protégée peuvent être modifiées.
- Seuls les enregistrements de la table protégée peuvent être supprimées.

- **La table système USER-UPDATE-COLUMNS**
    - Afin de savoir dans quelle mesure les colonnes d'une vue sont modifiables (insertion ou suppression), il faut interroger la vue
        - USER-UPDATE-COLUMNS du dictionnaire des données : c'est une table système qui contient la liste des colonnes modifiables des tables et des vues.

II. Les séquences

II.1. Introduction

- Tous les SGBD offrent désormais une facilité pour obtenir des identificateurs sans avoir à accéder à une table
- Cette facilité permet d'obtenir des valeurs qui sont générées automatiquement par le SGBD.
- Cette facilité n'est malheureusement pas standardisée ;

Par exemple,

- MySQL permet d'ajouter la clause AUTO_INCREMENT à une colonne ;
- DB2 et SQL Server ont une clause IDENTITY pour dire qu'une colonne est un identifiant ;
- ORACLE et PostgreSQL utilisent des **séquences.**

II.2. Définitions et objectifs

- Une séquence est un objet de la base de données comme les tables, les vues…
- Les séquences (séries) sont un excellent moyen d'avoir une base de données qui génère automatiquement des clés primaires entières uniques qui sont incrémentés ou décrémentés par le serveur Oracle (à chaque fois que l'on consulte).
- Elles sont créées par l'utilisateur.
- Elles sont stockées et gérées indépendamment d'une table, une séquence peut être partagée par plusieurs utilisateurs.

II.3. Création de séquence

CREATE SEQUENCE nom-séquence

[INCREMENT BY (1| valeur)] → **0 n'est pas autorisé**

Si un entier négatif est spécifié, la série decroitera dans l'ordre. Un entier positif fera croitre en ordre.

[**START WITH** valeur] → par défaut valeur minimale de la séquence

[**MAXVALUE** valeur| **NOMAXVALUE**] $--10^{27}$ ou -1

[**MINVALUE** valeur| **NOMINVALUE**] -- 1 OU -10^{27}

[**CYCLE**|NOCYCLE]

[**CACHE (**valeur| 20 |**NOCACHE**];

- **CYCLE** indique que la séquence doit continuer de générer des valeurs même après avoir atteint sa limite. Au-delà de la valeur maximale, la séquence génèrera la valeur minimale et incrémentera comme cela est défini dans la clause concernée et vice versa.
- **NOCYCLE** (par défaut) : Est une option qui interdit la série de produire des valeurs au-delà des maximum ou minimum définis. C'est la valeur par défaut ;
- **NOMAXVALUE** : ceci a pour effet de définir la valeur maximale croissante à 10^{27} et la valeur maximale décroissante à -1. Cette option est l'option par défaut
- **NOMINVALUE** valeur minimale=1 pour les séries croissantes et -10^{26} pour les séries décroissantes

CREATE SEQUENCE nom-séquence

[INCREMENT BY (1| valeur)]

[**START WITH** valeur]

[**MAXVALUE** valeur| **NOMAXVVALUE**]

[**MINVALUE** valeur| **NOMINVVALUE**]

[**CYCLE**|NOCYCLE]

[**CACHE (**valeur| 20 |**NOCACHE**]

[**ORDER** | **NORDER**]

- **CACHE** est une option qui permet à des numéros de séries d'être pré-alloués et stockés en mémoire pour un accès plus rapide la valeur minimale est 2.
- Afin d'optimiser l'utilisation des séquences, on peut demander à Oracle de placer en Mémoire cache un certain nombre de valeurs de la Séquence :
    - Exp: CREATE SEQUENCE ma_sequence CACHE 100;
- La mise en cache est importante. Elles peuvent avoir un effet significatif sur les performances. On peut mettre un nombre de valeurs élevé en mémoire cache.
- **NOCACHE** indique qu'aucune valeur de la séquence n'est dans le cache.
- Si les options CACHE et NOCACHE sont absentes de l'instruction, **vingt valeurs (20)** de la séquence seront mises en cache.
- **ORDER** garantit que les valeurs de la séquence sont générées dans l'ordre des requêtes. Si vos séquences jouent le rôle d'horodatage (timestamp), vous devrez utiliser cette option.
- Pour la génération des clés primaires, cette option n'est pas importante

II.4. Utilisation d'une séquence

- Pour utiliser une séquence on utilise les pseudo-colonnes.
    - Nom_seq. CURRVAL (renvoie la valeur courante de la séquence) et
    - Nom_seq. NEXTVAL (incrémente la séquence et retourne la nouvelle valeur).
    - Lors de la première utilisation d'une séquence, il faut utiliser NEXTVAL pour l'initialiser.
- ATTENTION !
    - Ne jamais utiliser une séquence avec CYCLE pour générer des valeurs de clé primaire
    - La valeur de la séquence peut être perdue :
        - Lorsqu'un enregistrement est supprimé d la table,
        - Lors de l'insertion d'un enregistrement, s'il y'a violation d'une contrainte d'intégrité (l'enregistrement n'a pas été inséré)
- Exemple 1 : **CREATE SEQUENCE seq1**

  INCREMENT BY 10

  START WITH 10

  MAXVALUE 100 ;

- Insertion d'un enregistrement dans une table en utilisant les séquences :

INSERT INTO Etudiants (num_etudiant, nom) VALUES (seq1.nextval, "Arsène")

Num_etudiant	Nom
10	Arsène

INSERT INTO Etudiants (num_etudiant, nom) VALUES (seq1.nextval, "Mohamed")

etudiant	Nom
10	Arsène
20	Mohamed

- Exemple 2 : Soit la séquence emp_seq définie comme suit :

```
CREATE SEQUENCE emp_seq
INCREMENT BY 1
START WITH 500
[MINVALUE 100
MAXVALUE 1000
NOCYCLE;
```

- Remplir la table « employé » (num, nom, sal) par les valeurs suivantes et à partir de la séquence pour alimenter la clé primaire :

Numéro	Nom	salaire
500	Abraham	735 000
501	Sidiki	755 000

1. INSERT INTO emp (num, nom, sal)
   Values (emp_seq. nextval, 'Abraham', 735000);
2. INSERT INTO emp (num, nom, sal)
   VALUES (emp_seq. nextval, 'Sidiki', 755000);

II.5. Modification d'une sequence

```
ALTER SEQUENCE [schéma.] nomSéquence
[INCREMENT BY entier]
[{MAXVALUE entier | NOMAXVALUE}]
[{MINVALUE entier | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE entier | NOCACHE}]
[{ORDER | NOORDER}] ;
```

- La clause START WITH ne peut être modifiée sans supprimer et recréer la séquence. Des contrôles sont opérés sur les limites, par exemple MAXVALUE ne peut pas être affectée à une valeur plus petite que la valeur courante de la séquence.

- Exemple 3 :
  - Supposons qu'on ne stockera pas plus de 95 000 PASSAGERS ET PAS PLUS de 850 équipements.
  - De plus les incréments des séquences doivent être égaux à 5.
  - Les instructions SQL à appliquer sont les suivantes : Chaque appel des méthodes NEXTVAL prendra en compte désormais le nouvel incrément tout en laissant les données existantes des tables.
    ALTER SEQUENCE seqEqp INCREMENT BY 5 MAXVALUE 850;
    ALTER SEQUENCE seqPas INCREMENT BY 5 MAXVALUE 95000;

II.6. Suppression d'une séquence

DROP SEQUENCE nom_de_la_sequence;

- **Exemple**
  - **DROP SEQUENCE** seq_etudiant;

II.7. Consultation des valeurs d'une séquence

a. Utilisation de la pseudo-table DUAL

- Table DUAL
  - C'est une particularité d'Oracle
  - Elle ne contient qu'une seule ligne et une seule colonne
  - Ne peut être utilisée qu'avec une requête SELECT.
  - Elle permet de faire afficher une expression dont la valeur ne dépend d'aucune table en particulier.
- Pour voir la valeur d'une séquence, on utilise CURRVAL avec la table DUAL.
- **Exemple 4 :**

1. **CREATE** SEQUENCE ma_sequence;
2. **SELECT** ma_sequence. CURRVAL **FROM** DUAL**;**

EREUR a la ligne 1 : ORA-08002 : séquence ma_sequence.CURRVAL pas encore définie dans cette session.

- **Exemple5 :**

SQL> **CREATE** SEQUENCE ma_sequence START WITH 1 minvalue 0;

Sequence créee.

SQL> **SELECT** ma_sequence**. nextval FROM** DUAL;

NEXTVAL

1

SQL> **SELECT** 'La valeur courante est ' | | ma_sequence. currval

FROM DUAL;

' LA VALEUR COURANTE EST' | | MA_SEQUENCE.CURRVAL

La valeur courante est 1

SQL> **ALTER** sequence ma_sequence INCREMENT **BY** 20 ;

Séquence modifiée**. ; NEXTVAL ---------- 21**

SQL> **SELECT ma_sequence.NEXTVAL + ma_sequence. NEXTVAL from DUAL;**

MA_SEQUENCE.NEXTVAL + MA_SEQUENCE.NEXTVAL

---

82 (41+41)

SQL> **ALTER** sequence ma_sequene INCREMENT BY -41

MAXVALUE 100 cycle nocache ;

Sequence modifiée.

SQL> **SELECT** ma_sequence**. nextval from** DUAL;

NEXTVAL

0

SQL> **SELECT ma_sequence.nextval from DUAL;**

NEXTVAL

100

SQL> **SELECT ma_sequence. nextval from DUAL;**

NEXTVAL

59

b. Utilisation de la vue USER_SEQUENCES du dictionnaire de données

- L'utilisateur peut avoir des informations sur les séquences qu'il a créées en consultant la table USER_SEQUENCES du dictionnaire de données.
- La table ALL_SEQUENES lui donne les sequences qu'il peut utiliser (même s'il ne les a pas créées)

- Exemple 6

SQL> CREATE SEQUENCE test_seq

START WITH 10

INCREMENT BY 5

MINVALUE 10

MAXVALUE 20

CACHE 2 ORDER;

➔ Sequence created.

SQL>

SQL> SELECT * FROM user_sequences

WHERE sequence_name = 'TEST_SEQ';

SEQUENCE_NAME MIN_VALUE MAX_VALUE INCREMENT_BY CACHE_SIZE LAST_NUMBER

TEST_SEQ 10 20 5 2 10

➔ Last_number : dernier nombre généré.

Exercice

- Personne (id pers, nompers , salaire, poste, date_emb, commission, #id_serv)
- Servive (id serv, nomserv, ville, #id_chef)
- Projet (id projet, titre, #resp_proj)
- Pers Projet (#id_pers, #id_proj)
- Act ProJet (#>id_act; #id_proj, h_prevues)
- Activite (id_act, libellee, tariff, description)

1. Créer les sequences suivantes:

    a) Créer une séquence nommé « seq_personne » qui sera utilisée pour les valeurs de la clé primaire de la table Personne (sans cycle, sans cache et qui commence par la valeur 100).
    b) Créer une séquence seq_joker qui sera utilisée pour les valeurs de la clé primaire des tables projet et activité (sans cycle, sans cache et qui commence par la valeur 10).
2. Remplir les tables avec des données suivantes dans l'ordre :

Table Projet

Id_proj	titre	description	Resp_proj
Proj10	projAA	NULL	101
Proj13	projBB	azerty	103

Table activité

Id_act	libelle	Tarif_h	Description
11	activitéAZ	100	Descripactive AZ
12	activiéBZ	75	Descripactivite BZ

Table Personne

Id_pers	nompers	salaire	poste	Date_emb	Commission	Id_serv
100	Yacouba	770 000	P13	18/02/2001	NULL	Serv002
101	Roukiatou	699 000	P2	03/08/1999	500	Serv002
102	Madi	685 000	P28	20/06/2012	200	Serv001
103	Alexis	815 000	P12	29/08/1999	NULL	Serv003

III. Les index

III.1. Motivations

- Exemple 1 : Rechercher tous les renseignements des clients qui s'appellent 'Mohamed'.

SELECT * FROM Client WHERE nom = 'Mohamed'

- Un moyen pour récupérer la ou les lignes répondant à la clause nom = 'Mohamed' est de balayer toute la table.
  - Le temps de réponse sera prohibitif des que la table dépasse quelques centaines de lignes.
- Afin d'optimiser ce type de requête, on pourra indexer l'attribut « nom ».
- Un index inclut une liste d'indices pour chaque valeur distincte de Nom.
- Les indices référencent les lignes de la table client.

Ali	1, 100, 120, 134
Mohamed	5, 15, 60, 112
Said	11, 25, 40
Philippe	3, 13, 23, 53, 135

- Maintenant, pour exécuter cette requête :

SELECT * FROM Client WHERE nom = 'Mohamed'

- L'accès se fait directement à partir de l'index.

III.2. Définitions

Un index est un objet facultatif associe à une table pour accélérer les requêtes sur cette table.

- Ils peuvent être créés afin d'améliorer les performances de mise à jour et d'exécution des données.
- Mis à jour automatiquement lors de modifications de la table.
- Peut concerner plusieurs attributs d'une même table (index composé).
  - Possibilité de créer plusieurs index sur une même table.
  - Maximum 16.
  - Dans le cas où il existe plusieurs index pour une table, l'optimiseur de requête sait choisir le meilleur index en fonction de la requête à exécuter

a. **Création**

- Un index est créé

Soit automatiquement par le noyau (indexation implicite)

Soit à la demande du développeur (indexation explicite)

- Indexation implicite :

Est mise en œuvre lorsqu'une clé primaire ou une contrainte d'unicité est définie sur une table.

- Indexation Explicite :

Pour accéder plus vite aux lignes d'une table à partir d'autres attributs de la table : index crées manuellement (explicites).

- **Syntaxe**

CREATE [UNIQUE | BITMAP] INDEX nom_index

ON nom_table (nom_attribut **[ASC|DESC],** …) ;

- UNIQUE permet de créer un index qui ne supporte pas les doublons.
- BITMAP créer un index « chaine de bits »
- ASC et DESC précise l'ordre de trie des données (croissant ou décroissant).

Indexer : oui…. Mais avec vigilance car prend de l'espace disque

➔ Indexer ce qui est nécessaire

Consigne :

- On interdit que deux lignes aient la même valeur pour l'indexer
- On évite la phase de tri des lignes (si les lignes sont physiquement rangées par valeur)

b. Suppression et recréation d'index

- Suppression d'un index

DROP INDEX nom-index;

- Recreer un index

ALTER INDEX nom-index REBUILD**;**

- **Pour verifier l'existance d'index**

Consulter les vues (Oracle) dans le DICTIONNAIRE DE DONNEES

- USER_INDEXES : Indexes créer par l'utilisateur
- USER_IND_COLUMNS
- ALL_INDEXES
- ALL_IND_COLUMNS
- DBA_INDEXES

- Les vues DBA contiennent des informations sur les objets de tous les schémas.

- Les vues ALL incluent les enregistrements des vues USER et des informations sur les objets pour lesquels des privilèges ont été octroyés au groupe PUBLIC ou à l'utilisateur courant.
- Les vues USER contiennent des informations sur les objets appartenant au compte qui exécute la requête

III.3. Types d'index

- Les types d'index les plus courants sous Oracle sont les suivants :
  - **Arbre équilibrés (Balanced arbre : B-arbre) :**
    - Les index B-Tree sont le type d'index par défaut quand on ne précise rien.
    - Comme son nom l'indique l'index B-Tree est organisé en arbre,
    - Toutes les branches de l'arbre ont la même longueur ;
  - **Bitmap :**
    - Ce type d'index est particulièrement efficace lorsque le nombre de valeurs est petit ainsi pour les opérations AND et OR.

a. Index B-Arbre

- Organisation
  - Les valeurs de clé d'un index B-tree sont stockées dans une arborescence équilibrée (Balanced tree -B-tree), (toutes les branches ont la même longueur) ce qui permet d'effectuer des recherches binaires rapides.
  - Au sommet de l'index se trouve la racine, qui contient les entrées pointant vers le niveau de l'index.
  - Le niveau suivant comprend les blocs 'branches', qui pointent vers le niveau suivant de l'index.
  - Au plus bas niveau se trouvent les nœuds feuilles, qui contiennent les entrées d'index pointant vers les lignes de la table.

b. Bitmap

- Un index stocké sous forme de bitmap comporte une chaine de bits (0 ou 1) pour chaque valeur des champs indexés
- La valeur bitmap est stocke en mode compressé, ce qui rend la place occupée par l'index inférieure à celle requise par un index de type B-arbre
- Chaque bit du bitmap correspond à une ligne de la table indexée. Cela permet d'affecter des recherches rapides lorsqu'il existe un nombre restreint de valeurs distincts (on dit que la colonne indexée présente une faible cardinalité).

III.4. Quand créer des index ?

- Attributs utilisés dans des jointures,
- Attributs servant souvent pour les sélections,

- Table de gros volumes dont la majorité des interrogations sélectionnées de -15% des lignes,
- Bitmap : attribut a peu de valeurs distinctes,
- B-arbre : attribut a beaucoup de valeurs distinctes.

III.5. Quand ne pas créer un index ?

- Mais il ne faut pas créer des index à tort et à travers.
  - Attributs souvent modifiés (index à recréer) vu le coût des opérations de mise à jour d'index
  - Attributs ne faisant pas objet d'interrogations fréquentes
  - Tables de petit volume,
  - Si requêtes sur NULL car les NULL, ne sont pas stockées dans l'index. (Ex : WHERE… IS NULL).
  - Dans certaines dialectes SQL, quand la clause WHERE fait appel à une fonction (mais pas dans Oracle ou c'est possible)