



Département Génie des systèmes  
Filière IMSI : 4<sup>ème</sup> année ingénieur

**ARCHITECTURE  
ET  
ADMINISTRATION  
DES  
BASES DE DONNÉES**

**DR F.KABLI**  
**Maitre assistant A a**

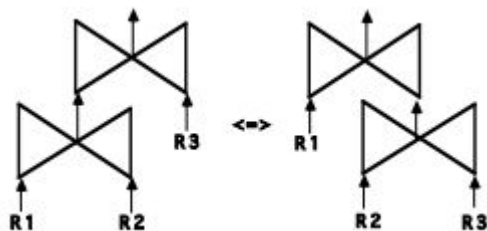
**kablifatima47@  
gmail.com**

# OPTIMISATION DES REQUÊTES

**2. Réécriture des requêtes :** Consiste à transformer logiquement les requêtes de sorte à obtenir une meilleure représentation, il existe **9** règles de transformation :

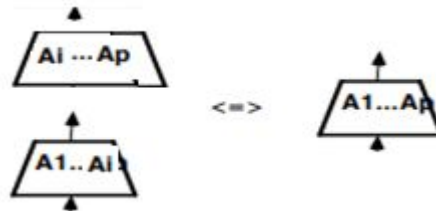
❖ **Règle 1 : Commutativité des Jointures**

❖ **Règle 2 : Associativité des Jointures**

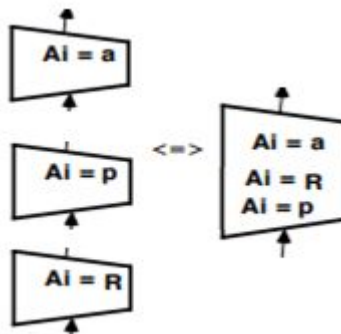


# OPTIMISATION DES REQUÊTES

## ❖ Règle 3 : Fusion des projections

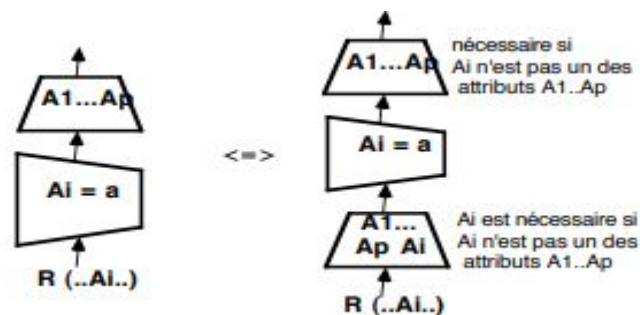


## ❖ Règle 4 : Regroupement des restrictions

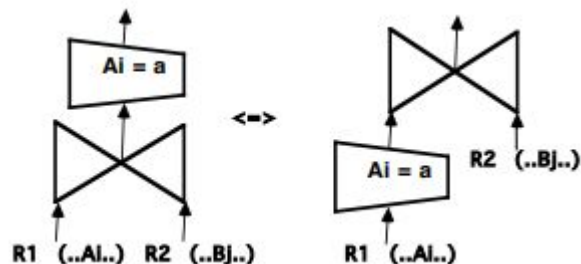


# OPTIMISATION DES REQUÊTES

## ❖ Règle 5 : Commutation des Restrictions et Projections

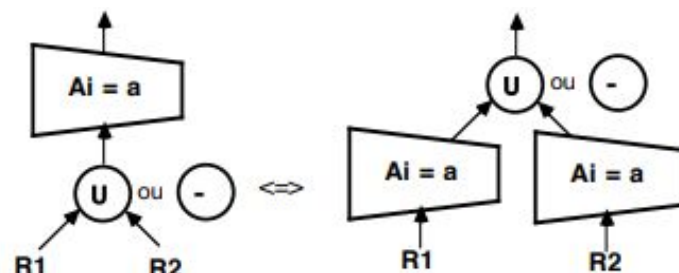


## ❖ Règle 6 : Commutation des Restrictions et Jointures

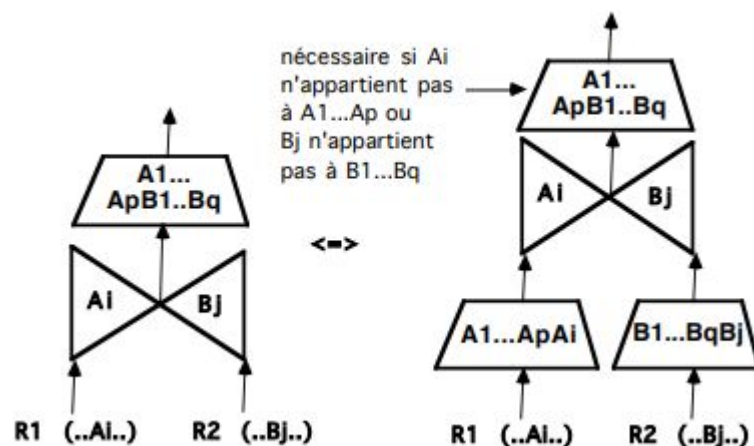


# OPTIMISATION DES REQUÊTES

- ❖ **Règle 7 : Commutation des Restrictions et Unions, ou des Restrictions et Différences**

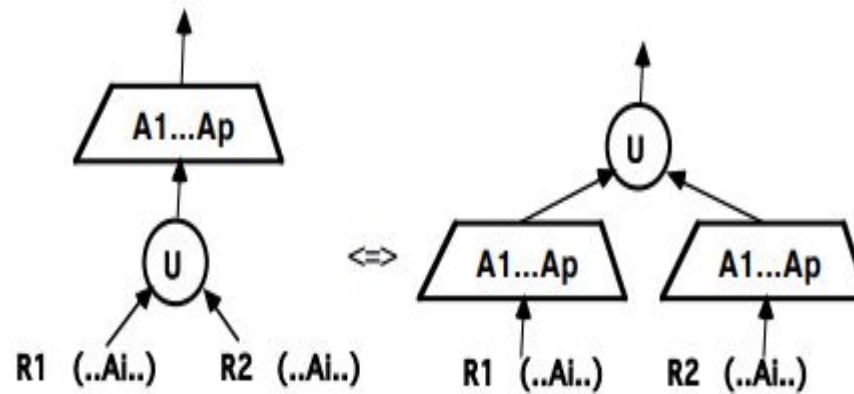


- ❖ **Règle 8 : Commutation des Projections et Jointures**



# OPTIMISATION DES REQUÊTES

## ❖ Règle 9 : Commutation des Projections et Unions



# OPTIMISATION DES REQUÊTES

## 3. Heuristique d'optimisation

- ❖ Séparer les sélections comportant plusieurs prédicats
- ❖ Descendre les sélections aussi bas que possible à l'aide des règles 5, 6, 7
- ❖ Regrouper les sélections successives portant sur la même relation à l'aide de la règle 4
- ❖ Descendre les projections aussi bas que possible à l'aide des règles 8, 9
- ❖ Regrouper les projections successive à partir de la règle 3 et éliminer les projections inutile







# OPTIMISATION DES REQUÊTES

## ❖ **Optimisation Physique :**

- ❖ Le nombre de E/S
- ❖ La taille des tampons nécessaires à l'exécution
- ❖ Le temps de CPU nécessaire.
- ❖ Coût de transfert
- ❖ etc.....



# LES VUES



# LES VUES

- ❖ Chaque utilisateur peut avoir sa description particulière de la partie de la base qu'il utilise.
- ❖ Une vue est un ensemble de relations déduites d'une base de données, c'est à dire comme le résultat d'une requête d'interrogation.
- ❖ Elles permettent de réaliser, dans le monde des SGBD relationnels, le niveau externe des SGBD selon l'architecture ANSI/SPARC.
- ❖ les vues garantissent une meilleure indépendance logique des programmes par rapport aux données
- ❖ Les vues en une phrase : une vue est une table qui est le résultat d'une requête (SELECT) à laquelle on a donné un nom



# LES VUES

- ❖ Les vues ont aussi un rôle de sécurité : l'utilisateur ne peut accéder qu'aux données des vues auxquelles il a droit d'accès.
- ❖ Une vue est donc finalement une table virtuelle calculable par une question.
- ❖ Une vue est une requête nommée qui permet de découper les traitements.
- ❖ Une vue est une définition stockée dans un SGBD relationnelle.



# LES VUES : SYNTAXE SQL

- ❖ **Création d'une vue:** `CREATE VIEW <NOM DE VUE> AS <QUESTION> [WITH CHECK OPTION]`
- ❖ **Utilisation d'une vue:** `SELECT ... FROM nom-de-vue WHERE ...`
- ❖ **Suppression d'une vue:** `DROP VIEW nom-de-vue`
- ❖ **Renommer une vue :** `RENAME ancien-nom TO nouveau-nom`
- ❖ Les vues figurent dans les tables systèmes `USER_VIEWS` et `ALL_VIEWS`

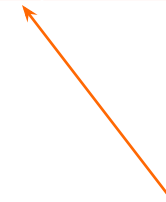


## LES VUES: EXEMPLE

- ❖ Créer la vue:
- ❖ `CREATE VIEW MAVUE`
- ❖ `SELECT * FROM MATALE`
- ❖ `WHERE J < 5;`

- ❖ **Interroger la vue:**
- ❖ `SELECT * FROM MAVUE`

I	J
1	2
2	4
3	6



MATALE

OU

`SELECT * FROM MAVUE WHERE J > 1`



## LES VUES: RAISONS D'UTILISATION

- ❖ Eviter de taper une longue requête
- ❖ Masquer certaines données à certains utilisateurs (Confidentialité)
- ❖ Remplacer une requête compliquée par des requêtes plus simples
- ❖ Augmenter l'indépendance logique (ex: changer le schéma de la BDD)



# UTILISATION DES VUES

- ❖ Création simple d'une vue.
- ❖ Création d'une vue à partir d'une autre vue.
- ❖ Suppression d'une vue utilisée par une autre vue.
- ❖ Modification de vue
- ❖ Insertion dans une vue
- ❖ Suppression d'une vue (ligne , colonne)





## EXEMPLE :

### ❖ Base de donnée R1 et R2:

La table vendeur

id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

La table commande :

ord_no	p mt	ord_date	customer_id	id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.4	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.6	2012-04-25	3002	5001

# EXEMPLE

**La table client :**

customer_id	cust_name	city	grade	id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005



## REQUÊTES :

1. Créer une vue pour afficher les vendeurs de la ville New York.
2. Créer une vue pour tous les vendeurs avec les colonnes id, name et city.
3. Ecrire une requête pour afficher l'id et la city des vendeurs de la ville New York.
4. Ecrire une requête pour trouver les vendeurs de la ville de New York qui ont réalisé la commission de plus de 0.13.
5. Créer une vue permettant de compter le nombre de clients que nous avons pour chaque grade.
6. Créer une vue indiquant pour chaque commande le vendeur et le client par nom.



## VUES MATÉRIALISÉES:

- ❖ Une vue matérialisée est **un objet qui permet de stocker le résultat d'une requête SELECT.**
- ❖ Chercher directement des données sans passer par les tables d'origine et/ou une table temporaire intermédiaire.
- ❖ Son contenu est rafraichi a la demande.
- ❖ Syntaxe:

```
CREATE  MATERIALIZED  VIEW  nom-MV  AS  
        SELECT Col1, Col2, [...], Coln FROM nom-table
```



# RAFRAICHISSEMENT DES VUES

## MATÉRIALISÉES:

### Les modes de :

- ❖ Une mise à jour sur demande .
- ❖ Une mise à jour automatique chaque fois qu'un changement est fait.

**ON COMMIT:** Dans ce mode, chaque fois qu'un changement des tables maîtres est engagé, la vue matérialisée est automatiquement actualisée pour refléter le changement.

**ON DEMAND:** Dans ce mode, vous devez exécuter une procédure pour mettre à jour la vue matérialisée.



# LES INDEXES

- ❖ **Définition** : L'utilisation d'un index simplifie et accélère les opérations de recherche, de tri, de jointure ou effectuées par le SGBD
- ❖ L'**index** placé sur une table permet au SGBD d'accéder très rapidement aux enregistrements, selon la valeur d'un ou plusieurs champs.
- ❖ Si l'index n'existe pas le moteur SGBD parcourt tous les enregistrements de la table, sinon il utilise les indexes pour rechercher rapidement les données.
- ❖ L'impact des index sur requêtes d'écriture est moins important.
- ❖ Un SGBD crée automatiquement un index sur(clé primaire, unicité)



# LES INDEXES

- ❖ Mais pourquoi ne pas simplement trier la table complète sur la base de la colonne *id* ? Pourquoi créer et stocker une structure spécialement pour l'index ?
- ❖ **Réponse** : Tout simplement parce qu'il peut y avoir plusieurs index sur une même table.



# LES INDEXES

## Désavantages

- ❖ Ils prennent de la place en mémoire.
- ❖ Ils ralentissent les requêtes d'insertion, modification et suppression, puisqu'à chaque fois, il faut remettre l'index à jour en plus de la table.

## Les types de représentation d'index :

- ❖ Un seul attribut de la table.
- ❖ Index sur plusieurs colonnes.
- ❖ Plusieurs Index sur plusieurs colonnes





# LES INDEXES

## 1. Index sur une seule colonne

**Requête:** Quelle est la catégorie de l'id 9 ? (par recherche dichotomique)

Index	ID	Catégorie produit
1	4	PC portable
2	10	Imprimante
3	2	Smartphone
4	1	Accessoires
9	15	Serveur
10	3	Tablette
15	9	Stockage




# LES INDEXES

## 2. Index sur plusieurs colonnes

Recherche sur : **Dupont Valérie C** ?

nom	prenom	init_2e_prenom		id	nom	prenom	init_2e_prenom	email
Boulian	Gérard	M		1	Dupont	Charles	T	charles.dupont@email.com
Caramou	Arthur	B		2	François	Damien	V	fdamien@email.com
Dupont	Charles	T		3	Vandenbush	Guillaume	A	guillaumevdb@email.com
Dupont	Valérie	C		4	Dupont	Valérie	C	valdup@email.com
Dupont	Valérie	G		5	Dupont	Valérie	G	dupont.valerie@email.com
François	Damien	V		6	François	Martin	D	mdmartin@email.com
François	Martin	D		7	Caramou	Arthur	B	leroiarthur@email.com
Loupiot	Laura	F		8	Boulian	Gérard	M	gebou@email.com
Sunna	Christine	I		9	Loupiot	Laura	F	loulau@email.com
Vandenbush	Guillaume	A		10	Sunna	Christine	I	chrichrisun@email.com



# LES INDEXES

## 3. Plusieurs Index sur plusieurs colonnes

Id	Id	Espèce	Sexe	Date de naissance	Nom	Commentaires	Date de naissance
1	2	chat	NULL	2010-03-24 02:23:00	Roucky	NULL	2008-09-11 15:38:00
2	1	chien	male	2010-04-05 13:43:00	Rox	Mordille beaucoup	2008-12-06 05:18:00
3	3	chat	femelle	2010-09-13 15:02:00	Schtroumpfette	NULL	2009-06-13 08:17:00
4	6	tortue	femelle	2009-06-13 08:17:00	Bobosse	Carapace bizarre	2009-08-03 05:12:00
5	9	tortue	NULL	2010-08-23 05:18:00	NULL	NULL	2010-03-24 02:23:00
6	4	tortue	femelle	2009-08-03 05:12:00	NULL	NULL	2010-04-05 13:43:00
7	7	chien	femelle	2008-12-06 05:18:00	Caroline	NULL	2010-08-23 05:18:00
8	8	chat	male	2008-09-11 15:38:00	Bagherra	NULL	2010-09-13 15:02:00
9	5	chat	NULL	2010-10-03 16:44:00	Choupi	Né sans oreille gauche	2010-10-03 16:44:00

# LES INDEXES

## ❖ Les différents types d'index

1. **Index UNIQUE** : Permet de s'assurer que jamais vous n'insérerez deux fois la même valeur et lorsque vous mettez un index **unique** sur une table, vous devez ajouter une **contrainte**.
2. **Index FULLTEXT** : Un index FULLTEXT est utilisé pour faire des recherches de manière puissante et rapide sur les données de type (**VARCHAR** ou **CHAR**).



# LES INDEXES

- ❖ **Index sur des colonnes de type alphanumérique Types (char et varchar) : on peut décomposer l'index en plusieurs lettres.**

1re lettre	2e lettre	3e lettre	4e lettre	5e lettre	6e lettre	7e lettre	8e lettre	9e lettre	10e lettre
B	o	u	l	i	a	n			
C	a	r	a	m	o	u			
D	u	p	o	n	t				
D	u	p	o	n	t				
D	u	p	o	n	t				
F	r	a	n	ç	o	i	s		
F	r	a	n	ç	o	i	s		
L	o	u	p	i	o	t			
S	u	n	n	a					
V	a	n	d	e	n	b	u	s	h



# LES INDEXES : SYNTAXE SQL

Ils peuvent être créés de deux manières :

- ❖ soit directement lors de la création de la table :

```
CREATE TABLE nom_table (  
    Att1 INT KEY,  
    Att2 VARCHAR(40) UNIQUE );
```

- ❖ Soit en les ajoutant par la suite :

```
CREATE TABLE nom_table (  
    Att1 NOT NULL AUTO_INCREMENT,  
    Att2 VARCHAR(40) NOT NULL,  
    Att3 CHAR(1),  
    Att4 DATETIME NOT NULL,  
    Att5 VARCHAR(30),  
    PRIMARY KEY (Att1),  
    INDEX ind_Att4 (Att4),  
    INDEX ind_Att5 (Att5(10)));
```



# LES INDEXES : SYNTAXE SQL

## ❖ Index FullText :

```
CREATE TABLE Livre (  
    id INT PRIMARY KEY,  
    auteur VARCHAR(50),  
    titre VARCHAR(200));
```

```
CREATE FULLTEXT INDEX ind_full_titre  
ON Livre (titre);
```

```
CREATE FULLTEXT INDEX ind_full_aut  
ON Livre (auteur);
```

```
CREATE FULLTEXT INDEX ind_full_titre_aut  
ON Livre (titre, auteur);(pour faire la recherche sur deux colonnes)
```



# LES INDEXES : TYPES

## Indexation par hachage :

1. Très performant pour les recherches d'égalité :  $=$  ,  $\neq$
2. Permet un accès directe à un enregistrement.
3. Il est basé sur une fonction de **Hachage**
4. Créer deux colonnes : référence de l'attribut et l'adresse physique.





# LES INDEXES : HACHAGE

Référence  
de produit

L'espace  
physique

lenovo i7	
Acer I5	
Acer I7	
Asus999	
HP550	

$H(\text{lenovo i7})=1$   
 $H(\text{HP550})=5$

.  
. .  
.

1	Acer I5	Pc Portable I5
2	lenovo i7	Pc Portable I7
3	HP550	Pc Portable HP
4	Acer I5	Pc Portable ACER
5	Asus999	Pc Portable Assus

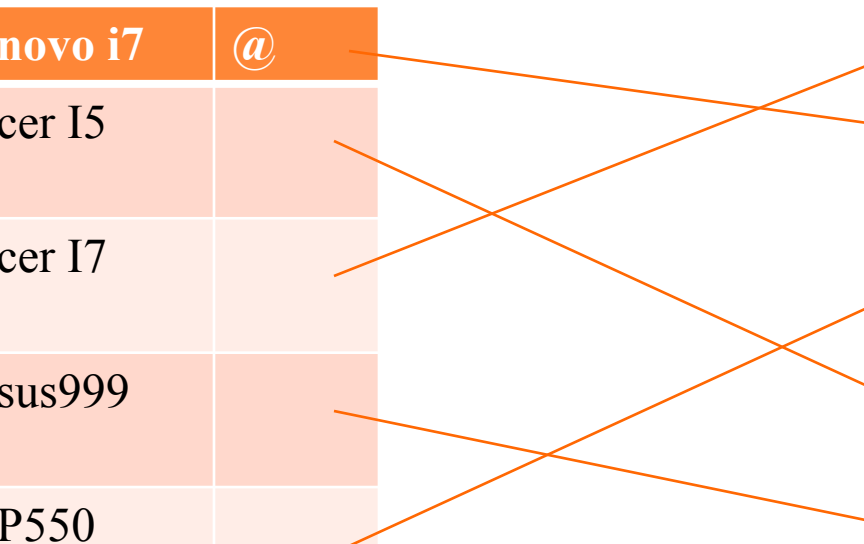


# LES INDEXES : HACHAGE

Référence  
de produit

L'espace  
physique

lenovo i7	@	1	Acer I5	Pc Portable I5
Acer I5		2	lenovo i7	Pc Portable I7
Acer I7		3	HP550	Pc Portable HP
Asus999		4	Acer I5	Pc Portable ACER
HP550		5	Asus999	Pc Portable Assus



# LES INDEXES : HACHAGE

Référence  
de produit

L'espace  
physique

lenovo i7		1	Acer I5	Pc Portable I5
Acer I5		2	lenovo i7	Pc Portable I7
Acer I7		3	HP550	Pc Portable HP
Asus999		4	Acer I5	Pc Portable ACER
HP550		5	Asus999	Pc Portable Assus



# LES SEQUENCES



# LES SÉQUENCES : DÉFINITION

## Définition :

- ❖ Une séquence est une suite de nombres entiers
- ❖ Générer des valeurs (numériques)
- ❖ Objet partagé
- ❖ L'utilisation d'une séquence permet donc d'avoir à disposition une suite de valeurs. Ceci peut permettre de :
  1. Générer des clés uniques dans des tables
  2. Avoir un compteur à titre informatif, que l'on incrémente quand on veut



# LES SÉQUENCES : SYNTAXE SQL

## **Création de séquence:**

```
CREATE SEQUENCE nom-séquence  
INCREMENT BY valeur  
START WITH valeur  
MAXVALUE valeur  
MINVALUE valeur  
CYCLE  
CACHE ( valeur | 20 )  
ORDER;
```



SYNTAXE	DESCRIPTION
<b>INCREMENT BY</b>	intervalle entre les deux valeurs de la séquence (par défaut 1).
<b>MAXVALUE</b>	valeur maximale de la séquence.
<b>MINVALUE</b>	valeur minimale de la séquence.
<b>CYCLE</b>	la séquence continuera de générer des valeurs après avoir atteint sa limite.
<b>NOCYCLE</b>	la séquence s'arrêtera de générer des valeurs après avoir atteint sa limite.
<b>CACHE</b>	les valeurs de la séquence seront mises en cache.
<b>NOCACHE</b>	les valeurs de la séquence ne seront pas dans le cache.
<b>ORDER</b>	les séquences sont générées dans l'ordre des requêtes.



# LES SÉQUENCES : SYNTAXE SQL

## ❖ **Suppression de séquence:**

**DROP SEQUENCE** nom-Séquence ;

## ❖ **Modifier une séquence :**

**ALTER SEQUENCE** nom\_Séquence  
**INCREMENT BY** Valeur  
**MAXVALUE** valeur  
**CACHE** valeur ;





# LES SÉQUENCES : UTILISATION

- ❖ **La fonction NEXTVAL** : incrémente la séquence et retourne la nouvelle valeur.

Nom\_sequence.**NEXTVAL** ;

- ❖ **Exemple:**

SELECT Nom\_sequence.**NEXTVAL** FROM **dual**;

- ❖ **La fonction CURRVAL** : retourne la valeur courante de la séquence.

- ❖ **Exemple:**

SELECT Nom\_sequence.**CURRVAL** FROM **dual**;



# LES SÉQUENCES : UTILISATION

**Utiliser la séquence lors de la création de la table:**

```
CREATE SEQUENCE S1 START WITH 1;
```

```
CREATE TABLE TAB (
```

```
  a1 NUMBER DEFAULT S1.NEXTVAL NOT NULL,
```

```
  a2 VARCHAR(10));
```

```
INSERT INTO TAB (a2) VALUES ('Mohamed');
```



## LES SÉQUENCES : UTILISATION

- ❖ **L'effet de cache** : La mise en cache des valeurs de séquence en mémoire donne un accès plus rapide à ces valeurs.
- ❖ **Les (Gaps)** dans les séquences peuvent se produire lorsque:
  1. Rollback
  2. System Crashes
  3. Séquence partagée
- ❖ **Information sur une séquence:**  

```
SELECT sequence_name, min_value,max_value  
FROM user_sequences;
```



# *LES TRANSACTIONS*



## *TRANSACTION* (DÉFINITION ):

- ❖ Une **transaction** est une séquence d'opérations de lecture ou de mise à jour sur une base de données
- ❖ Une suite d'actions demandée par un seul ' utilisateur ou programme d'application, qui lit ou met a jour le contenu de la base de données.
- ❖ Une transaction est une unité logique de travail sur la base de données
- ❖ Les transactions permettent de regrouper des requêtes dans des blocs, et de faire en sorte que tout le bloc soit exécuté en une seule fois, cela afin de préserver l'intégrité des données de la base.



## *TRANSACTION* (DÉFINITION ):

- ❖ Durant l'exécution d'une transaction, l'état de la base peut ne pas être cohérent
- ❖ Quand une transaction est validée (commit), l'état de la base doit être cohérent
- ❖ Toute la transaction est réalisée ou rien ne l'est :

**Validation** : toute la transaction est prise en compte

**Avortement ou annulation** : la transaction n'a aucun effet.

### **Le rôle d'une transaction :**

- ❖ Préserver l'intégrité des données.
- ❖ Gérer la concurrence aux accès des données



## COMMENT SE DÉROULE UNE TRANSACTION ?

- ❑ On démarre une transaction.
- ❑ On exécute les requêtes désirées une à une.
- ❑ Si une des requêtes est échoué, on annule toutes les requêtes, et on termine la transaction.
- ❑ Par contre, si à la fin des requêtes, tout s'est bien passé, on valide tous les changements, et on termine la transaction.
- ❑ Si le traitement est interrompu (entre deux requêtes par exemple), les changements ne sont jamais validés, et donc les données de la base restent les mêmes qu'avant la transaction.



# TRANSACTIONS: LES PROPRIÉTÉS **ACID**

Le système doit garantir certaines propriétés :

- ❖ **Atomicité** : Soit toutes les opérations de la transaction sont validées, ou bien aucune opération ne l'est.
- ❖ **Cohérence** : L'exécution d'une transaction préserve la cohérence de la base ( une transaction qui provoque un conflit d'intégrité doit échouer ).
- ❖ **Isolation** : (pas de vision partielle de données ), une transaction ne voit pas les données modifiées et non encore validées par les autres transactions.
- ❖ **Durabilité** : (conservation de données)  
les modifications d'une transaction validée sont enregistrées sur le disque.





## *TRANSACTION : EXEMPLE*

Une transaction qui transfert 1000 Euro du compte A vers le compte B

1. Lire(A)
2.  $A := A - 1000$
3. Ecrire(A)
4. Lire(B)
5.  $B := B + 1000$
6. Ecrire(B)



## *TRANSACTION : EXEMPLE*

- ❖ La base est cohérente si la somme  $A+B$  ne change pas suite à l'exécution de la transaction (**cohérence**)
- ❖ Si la transaction “échoué” après l'étape 3, alors le système doit s'assurer que les modifications de  $A$  ne soient pas persistantes (**atomicité**).
- ❖ Une fois l'utilisateur est informé que la transaction est validée, il n'a plus à s'inquiéter du sort de son transfert (**durabilité**).
- ❖ Si entre les étapes 3 et 6, une autre transaction est autorisée à accéder à la base, alors elle “verra” un état incohérent ( $A+B$  est inférieur à ce qu'elle doit être). **L'isolation** n'est pas assurée. La solution triviale consiste à exécuter les transactions en séquence.



## ETATS D'UNE TRANSACTION

- ❖ **Active** : la transaction reste dans cet état durant son exécution
- ❖ **Partiellement validée** : juste après l'exécution de la dernière opération
- ❖ **Echec** : après avoir découvert qu'une exécution "normale" ne peut pas avoir lieu
- ❖ **Avorté** : Après que toutes les modifications faites par la transaction soient annulées (Roll back). Deux options
  - ❑ Ré-exécuter la transaction
  - ❑ Tuer la transaction
- ❖ **Validée** : après l'exécution avec succès de la dernière opération



# TRANSACTION : SYNTAXE SQL

## ❖ Exemple :

**User 1 :**

update tab1

set NOM='mohamed '

where ID= 3;

**commit;**

**User 2 :**

**select \* from user1.tab1;**

**>sql**

1 Ahmed	1 Ahmed
2 Amina	2 Amina
3 Karim	3 mohamed

ID	PRENOM
1	Ahmed
2	Amina
3	<del>Karim</del>

mohamed



# TRANSACTION : SYNTAXE SQL

## ❖ Exemple :

**User 1 :**

```
update tab1  
set NOM='mohamed '  
where ID= 3;
```

**ROLLBACK;**

**User 2 :**

```
select * from user1.tab1;
```

```
>sql
```

```
1 Ahmed  
2 Amina  
3 Karim
```

ID	PRENOM
1	Ahmed
2	Amina
3	Karim



# TRANSACTION : MISE EN ŒUVRE SOUS ORACLE

## Début de transaction :

- ❖ Connexion.
- ❖ Fin de la transaction précédente.

## Fin de transaction :

- ❖ Commit ou Rollback.
- ❖ Instructions du DDL.
- ❖ Déconnexion

