III. Le Langage de Manipulation de données (LMD SQL)

III.1 Syntaxe générale d'une requête:

SELECT [ALL|DISTINCT] attribut(s) FROM table(s)
[WHERE condition] [GROUP BY attribut(s) [HAVING condition]]
[ORDER BY attribut(s) [ASC|DESC]];

- **Tous les tuples d'une table**

ex. SELECT * FROM Client;

- **Tri du résultat**

ex. *Par ordre alphabétique inverse de nom*

SELECT * FROM Client ORDER BY Nom DESC;

BD exemple

- client (**numcli** , nom, prenom, adresse, codepost, ville, tel);
- produit (**numprod**, designation, prixunit, qtestock);
- commande (**numcom**, numcli, idvendeur, datec, quantite, numprod);
- vendeur (**idvendeur**, nomvendeur, qualité, salaire, commission));

Interrogation des données

- **Calculs** ex. *Calcul de prix TTC*

SELECT PrixUni+PrixUni*0.206 prixttc FROM Produit;

- **Projection**

ex. *Noms et Prénoms des clients, uniquement*

SELECT Nom, Prenom FROM Client;

- **Restriction**

ex. *Clients qui habitent à Tunis*

SELECT * FROM Client WHERE Ville = ‘Tunis’;

Interrogation des données

- ex. *Commandes en quantité au moins égale à 3*

SELECT * FROM Commande WHERE Quantite >= 3;

- ex. *Produits dont le prix est compris entre 50 et 100 DT*

SELECT * FROM Produit

WHERE PrixUni **BETWEEN** 50 **AND** 100;

- ex. *Commandes en quantité indéterminée*

SELECT * FROM Commande WHERE Quantite IS **NULL**;

Interrogation des données

- ex. *Clients habitant une ville dont le nom se termine par* Menzel

SELECT * FROM Client

WHERE Ville **LIKE** '**%**Menzel';

- 'Menzel**%**' ⇒ commence par *Menzel*
- '**%**Men**%**' ⇒ contient le mot *Men*
  - le joker '_' peut être remplacé par n'importe quel caractère
  - le joker '%' peut être remplacé par 0, 1, 2, … caractères

Interrogation des données

- ex. *Prénoms des clients dont le nom est Mohamed, Ahmed ou Mahmoud*

SELECT Prenom FROM Client

WHERE Nom **IN** (‘Ahmed’, ‘Mahmoud’, ‘Mohamed’);

  - **spécifie une liste de constantes dont une doit égaler le premier terme pour que la condition soit vraie**
  - **pour des NUMBER, CHAR, VARCHAR, DATE**

- **NB :** Possibilité d’utiliser la négation pour tous ces prédicats : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN.

Fonctions d'agrégat

- Elles opèrent sur un ensemble de valeurs, fournissent une valeur unique
  - AVG() : moyenne des valeurs
  - SUM() : somme des valeurs
  - MIN(), MAX() : valeur minimum, valeur maximum
  - COUNT() : nombre de valeurs
- ex. *Moyenne des prix des produits*

SELECT AVG(PrixUni) FROM Produit;

Opérateur DISTINCT

- ex. *Nombre total de commandes*

```
SELECT COUNT(*) FROM Commande;
SELECT COUNT(NumCli) FROM Commande;
```

- ex. *Nombre de clients ayant passé commande*

```
SELECT COUNT( DISTINCT NumCli)
FROM Commande;
```

Exemple

- Table COMMANDE (simplifiée)

NumCli	*Date*	*Quantite*
1	22/09/99	1
3	22/09/99	5
3	22/09/99	2

- COUNT(NumCli) ⇒Résultat = 3
- COUNT(DISTINCT NumCli) ⇒ Résultat = 2

Jointure

- Consiste en un produit cartésien ou certaines lignes seulement sont sélectionnées via la clause WHERE
- ex. *Liste des commandes avec le nom des clients*

SELECT Nom, Date, Quantite FROM Client, Commande WHERE Client.NumCli = Commande.NumCli;

- ex. *Idem avec le numéro de client en plus*

SELECT C1.NumCli, Nom, Date, Quantite

FROM Client **C1**, Commande **C2** WHERE **C1**.NumCli = **C2**.NumCli ORDER BY Nom;

- **NB :** Utilisation d*'alias* (C1 et C2) pour alléger l' écriture + tri par nom.

Jointure exprimée avec le prédicat IN

- ex. *Nom des clients qui ont commandé le 23/09*

```
SELECT Nom FROM Client WHERE NumCli
IN (SELECT NumCli FROM Commande
    WHERE DateC = '23-09-2000' );
```

NB : Il est possible d'imbriquer des requêtes.

Prédicats EXISTS / NOT EXISTS

- ex. *Clients qui ont passé au moins une commande [n 'ont passé aucune commande]*

```
SELECT * FROM Client C1
WHERE [NOT] EXISTS ( SELECT * FROM
   Commande C2 WHERE C1.NumCli =
   C2.NumCli );
```

Prédicats ALL / ANY

- ex. *Numéros des clients qui ont commandé au moins un produit en quantité supérieure à chacune [à au moins une] des quantités commandées par le client n° 1.*

```
SELECT DISTINCT NumCli FROM Commande
WHERE Quantite > ALL [ANY] (
SELECT Quantite FROM Commande
WHERE NumCli = 1 );
```

Groupement

- permet de créer des groupes de lignes pour appliquer des fonctions d'agrégat sur les groupes
  - il est possible de créer des groupes sur plusieurs attributs
- ex. *Quantité totale commandée par chaque client*

```
SELECT NumCli, SUM(Quantite) FROM Commande
    GROUP BY NumCli;
```

- ex. *Nombre de produits différents commandés...*

```
SELECT NumCli, COUNT(DISTINCT NumProd)
FROM Commande GROUP BY NumCli;
```

Groupement

- ex. *Quantité moyenne commandée pour les produits faisant l'objet de plus de 3 commandes*

SELECT NumProd, AVG(Quantite)
FROM Commande GROUP BY NumProd HAVING COUNT(*)>3;

Attention : La clause HAVING ne s'utilise qu'avec GROUP BY.

Opérations ensemblistes

- INTERSECT, MINUS, UNION
  - les deux tableaux opérandes doivent avoir une description identique:
    - nombre de colonnes
    - domaines des valeurs de colonnes
  - structure générale
    - `co-requête OPERATEUR co-requête`
    - ou chaque co-requête est une instruction SELECT

ex. *Numéro des produits qui soit ont un prix inférieur à 100 DT, soit ont été commandés par le client n° 2*

SELECT NumProd FROM Produit WHERE PrixUni<100 UNION SELECT NumProd FROM Commande WHERE NumCLi=2;

Les sous-requêtes

- mécanisme très puissant mais d'un emploi subtil !
- = une requête SELECT insérée dans la clause WHERE d'une autre requête SELECT
- bien sur, une sous-requête peut contenir une sous-sous-requête !

Les sous-requêtes

- peuvent retourner
  - une seule valeur
  - une liste de valeurs
- une sous-requête constitue le second membre d'une comparaison dans la clause WHERE

Les sous-requêtes retournant une valeur

```
SELECT last_name, salary FROM s_emp
WHERE salary >= (SELECT AVG(salary) FROM
  s_emp);
```

sous-requête

```
LAST_NAME                 SALARY
------------------------- -----------------
Velasquez                        2500
Ngao                             1450
Nagayama                         1400
Quick-To-See                     1450
...
```

Les sous-requêtes retournant une liste de valeurs

```
SELECT last_name, salary FROM s_emp
 WHERE salary >= ALL (SELECT salary FROM
   s_emp WHERE dept_id = 10) ;
```

LAST_NAME	SALARY
Velasquez	2500
Ngao	1450
Quick-To-See	1450
Ropeburn	1550
Giljum	1490

Contraintes des sous-requêtes

- clause ORDER BY y est interdite
- chaque sous-requête doit être entourée de parenthèses
- clause SELECT d'une sous-requête ne peut contenir qu'un seul attribut
- les attributs définis dans la requête principale peuvent être utilises dans la sous-requête
- les attributs définis dans la sous-requête ne peuvent pas être utilises dans la requête principale

Opérateurs de comparaison et les sous-requêtes

- dans le cas de sous-requêtes retournant une seule valeur, les opérateurs classiques (<, >, …) peuvent être appliques
- dans le cas de sous-requêtes retournant une liste de valeurs, il faut utiliser les quantificateurs
  - ANY: l'expression est vraie si une des valeurs de la sous-requête vérifie la comparaison
  - ALL: l'expression est vraie si toutes les valeurs de la sous-requête vérifient la comparaison
- Note: IN équivaut à = ANY

Les sous-requêtes multiples

- la clause WHERE d'une requête principale peut contenir plusieurs sous-requêtes reliées par les connecteurs AND et OR

Mise à jour des données

- **Ajout d'un tuple**

INSERT INTO nom_table VALUES (val_att1, val_att2, …);

ex. INSERT INTO Produit VALUES (400, 'Nouveau produit', 78.90);

- **Mise à jour d'un attribut**

UPDATE nom_table SET attribut=valeur

[WHERE condition];

ex. UPDATE Client SET Nom='Dudule' WHERE NumCli = 3;

Mise à jour des données

- **Suppression de tuples**

DELETE FROM nom_table [WHERE condition];

ex. DELETE FROM Produit;

ex. DELETE FROM Client WHERE Ville = ‘Tunis’;

Les vues

- **Vue** : table *virtuelle* calculée à partir d'autres tables grâce à une requête

CREATE [OR REPLACE]
[FORCE | NOFORCE] **VIEW** nom_de_vue
[(alias_colonne1, alias_colonne2, ...)]
AS subquery
WITH CHECK OPTION [CONSTRAINT
constraint] ;

- ***Mots clés et paramètres***
- **OR REPLACE :** permet de supprimer puis de recréer la vue si elle existe
- **FORCE** : ignore les erreurs et crée la vue
- **WITH CHECK OPTION** : permet d'assurer la cohérence des informations modifiées afin de laisser dans la vue les lignes affectées par une modification
- **CONSTRAINT constraint** : permet juste de nommer la contrainte de la vue

- **Syntaxe (suite)**

Remarques :

- la modification d'une table de base affecte la vue
- le corps d'une vue ne peut contenir de clause ORDER BY ou FOR UPDATE
- on ne peut effectuer des insertions, des mises à jours et des suppressions dans une vue contenant une jointure, des opérateurs ensemblistes, des fonctions de groupe, les clauses GROUP BY, ou l'opérateur DISTINCT.
- tables systèmes: all_views, dba_views, user_views

Les vues

- **Intérêt des vues**
  - Simplification de l'accès aux données en masquant les opérations de jointure

ex. CREATE VIEW Prod_com AS SELECT P.NumProd, Dési, PrixUni, Date, Quantite FROM Produit P, Commande C WHERE P.NumProd=C.NumProd;

SELECT NumProd, Dési FROM Prod_com

WHERE Quantite>10;

Les vues

- **Intérêt des vues**
  - Sauvegarde indirecte de requêtes complexes
  - Présentation de mêmes données sous différentes formes adaptées aux différents usagers particuliers
  - Support de l'indépendance logique

ex. Si la table Produit est remaniée, la vue Prod_com doit être refaite, mais les requêtes qui utilisent cette vue n'ont pas à être remaniées.

Les vues

- **Intérêt des vues**
  - Renforcement de la sécurité des données par masquage des lignes et des colonnes sensibles aux usagers non habilités
- **Problèmes de mise à jour, restrictions**
  - La mise à jour de données via une vue pose des problèmes et la plupart des systèmes impose d'importantes restrictions.

Les transactions

Les Transactions

Exemple

Une transaction peut transférer une somme d'argent entre deux comptes d'un client d'une banque. Elle comporte deux ordres : un débit sur un compte et un crédit sur un autre compte. Si un problème empêche le crédit, le débit doit être annulé.

Les Transactions

Propriétés : il est facile de se souvenir des propriétés essentielles des transactions : elles sont ACID".

- **Atomicité** : un tout indivisible ;
- **Cohérence :** une transaction doit laisser la base dans un état cohérent ; elle ne doit pas contredire une contrainte d'intégrité ou mettre les données dans un état anormal,
- **Isolation** : les modifications effectuées par une transaction ne doivent être visibles par les autres transactions que lorsque la transaction est validée ;
- **Durabilité** : le SGBD doit garantir que les modifications d'une transaction validée seront conservées, même en ca de panne.

Les Transactions

Propriétés

- "AID" est du ressort du système transactionnel du SGBD.
- "C" est du ressort de l'utilisateur (ou du programmeur) mais il est aidé
  - par "I", car ainsi il n'a pas à considérer les interactions avec les autres transactions,
  - par la vérification automatique des contraintes d'intégrité par le SGBD.
- Le "I" est effectué par le système de contrôle de la concurrence et "AD" sont supportés par les procédures de reprise après panne.

Les Transactions

- Une transaction se termine :
  - soit par une validation qui applique les modifications de façon permanente,
  - soit par une annulation qui remet la base dans son état initial (avant l'exécution de la transaction).
- En cas **de fin anormale d'une tâche utilisateur** : il y a automatiquement annulation des transactions non terminées afin d'assurer l'intégrité de la base.

Les Transactions dans SQL

- Une transaction commence :
  - au début d'une session de travail :
    - Exemple : lors d'une ouverture de session sous `sql*plus`
  - ou juste après la fin de la transaction précédente.
- Elle se termine soit par :
  - Un ordre explicite de validation: `commit`
    - Les modifications deviennent alors définitives,et visibles par toutes les autres transactions.
  - Un ordre explicite d'annulation: `Rollback`
    - Toutes les modifications depuis le début de la transaction sont alors annulées.

    - La BD est restituée à son état avant l'exécution de transaction annulée.

Atomicité d'une transaction

- L'ordre `commit` valide toutes les modifications effectuées sur la BD lors de la transaction.
- L'ordre `Rollback` annule toutes les modifications de la transaction.
- Les instructions SQL sont atomiques : quand une instruction provoque une erreur (par exemple si une contrainte d'intégrité n'est pas vérifiée), toutes les modifications déjà effectuées par cette instruction sont annulées.
  - **Exemple**: modification du N° du depart dans la table `emp` en ajoutant `20` à l'ancien numéro.

```
sql> update emp a
 set (deptno) =(select distinct deptno + 20
    from emp b where b.deptno=a.deptno);
ERREUR à la ligne 1 :
ORA-02291: violation de contrainte (SCOTT.FK_DEPTNO)
  d'intégrité - touche parent introuvable
```

  Toutes les modifications effectuées par cet ordre sql sont annulées.

  - Mais cette erreur ne provoque pas nécessairement de `rollback` automatique de la transaction (c'est le cas si elle est exécutée par sql*Plus).

Exemple

- Soit la table `compte_client` d'une BD d'une banque : `compte_client(NCompte,nomClient,solde)`
- On désire effectuer un virement de 200 dinars du compte `1111` vers le compte `2222`.
- L'opération consiste à effectuer les 2 opérations suivantes :
  - Débiter le solde de **`1111`** de 200 dinars
  - Créditer le solde de **`2222`** de 200 dinars
- Ces deux opérations doivent former un tout indivisible c'est-à-dire
  - Soit elles sont exécutées toutes les deux,
  - Soit aucune n'est exécutée.

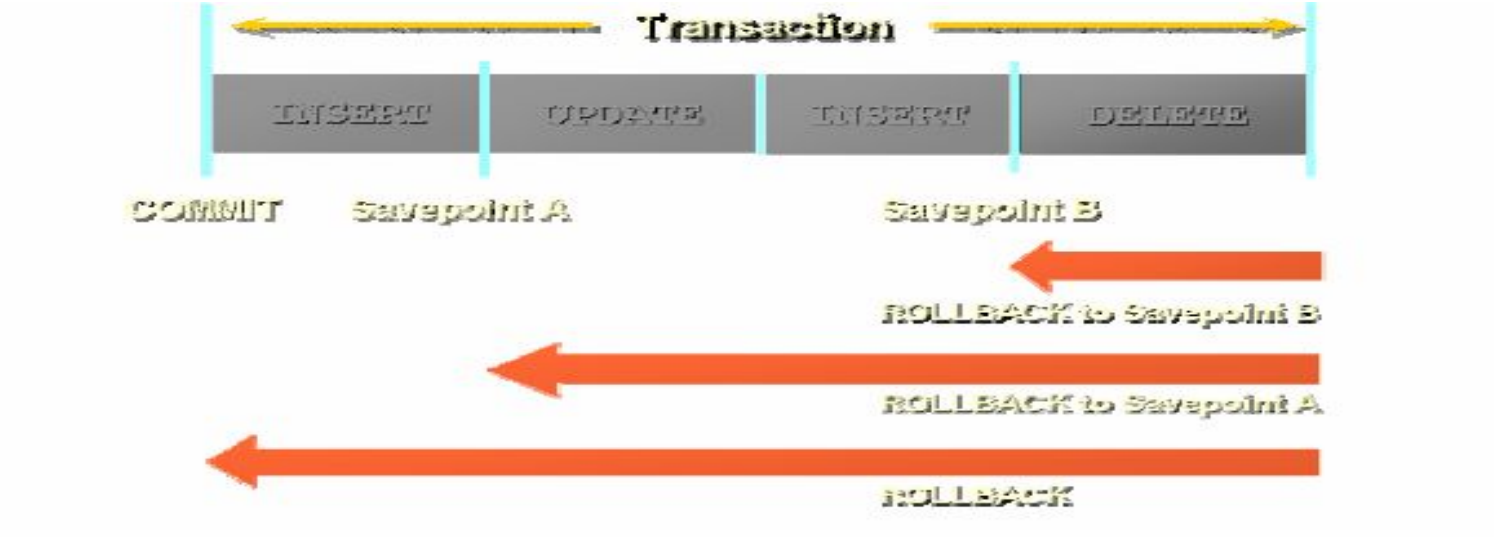
Elles doivent donc être exécutées dans la même transaction

```
…
commit;-- fin de la transaction précédente
-- nouvelle transaction
update compte_client set
          Solde = solde - 200
          where ncompe=1111;
update compte_client set
          Solde = solde + 200
          where ncompe=2222;
commit;-- validation de la transaction
…
```

Même transaction

Les transactions dans SQL

- Les points de sauvegarde:`savepoint`
  - Il peut arriver qu'il ne soit pas nécessaire d'annuler toutes les modifications effectuées. Pour cela, il est possible d'utiliser des points de sauvegarde **`SAVEPOINT`**. Ces points permettront de faire des **`ROLLBACKS`** "réduits".



Exemple

```
SQL> select sal,comm from emp where empno=7499;
 SAL       COMM
----- ----------
 1760        300
SQL> update emp set sal=nvl(sal,0) + nvl(sal,0)*0.1;
14 ligne(s) mise(s) à jour.
SQL> select sal,comm from emp where empno=7499;
       SAL       COMM
---------- ----------
      1936        300
SQL> savepoint A;
Point de sauvegarde (SAVEPOINT) créé.
SQL> update emp set comm=comm + comm*0.1 Where comm is not null;
4 ligne(s) mise(s) à jour.
SQL> select sal,comm from emp where empno=7499;
  SAL       COMM
-------- ----------
    1936        330
SQL> Rollback to A;
Annulation (ROLLBACK) effectuée.
SQL> commit;
Validation effectuée.
SQL> select sal,comm from emp where empno=7499;
 SAL       COMM
---------- ----------
    1936        300
```