Le langage de requête SQL

1-Langage de Manipulation de Données (LMD)

L2

H.BELLEILI
h.belleili@gmail.com
Université Badji Mohktar Annaba
Département informatique

SQL

- SQL (Structured Query Langage) dérivé de l'Algèbre relationnelle
- C'est un langage non procédural dans sa formulation
- Actuellement, c'est le langage standard des SGBDs relationnels commerciaux

Les trois facettes de SQL

1. ***DDL*** (***Data Definition Language***), pour la spécification des informations concernant les relations, telles que :
   - ✕ Le schéma de chaque relation;
   - ✕ Le domaine des values associées à chaque attribut;
   - ✕ Les contraintes d'intégrité;
   - ✕ D'autre informations comme l'ensemble d'index à maintenir pour chaque relation, les information d'autorisation et de sécurité pour chaque relation.
2. ***DML*** (***Data Manipulation Language***), pour la manipulation des tables et plus précisément les manipulations des tuples de relations.
3. ***DCL*** (***Data Control Language***), pour gérer la définition physique des accès (index), la spécification des fichiers physiques et la validation des opérations exécutées dans un contexte multiposte.

Notations utilisées pour la présentation de la syntaxe de SQL

- Basées sur ***BNF*** **(*Backus-Naur Form*)** avec les extensions suivantes:
  - Les crochet ([]) indiquent des éléments optionnels;
  - Les pointillés (é) suivent un élément qui peut être répété plusieurs fois;
  - Les accolades ({}) groupent comme un seul élément une séquence d'élément;
  - Un exposant plus ($^{+}$) indique que l'élément qui précède peut être répété n fois (n >0), chaque occurrence est séparée de la précédente par une virgule;
  - Un exposant (*) indique que l'élément qui précède peut être répété n fois (n ≥0), chaque occurrence est séparée de la précédente par une virgule.

La commande SELECT

- ***SELECT*** est la requête de recherche de données qui est à la base de SQL
- L'objectif: rechercher et afficher les données à partir d'une ou plusieurs tables ou vues d'une base de données
- Extrêmement puissante: capable d'effectuer, en une seule instruction, l'équivalent des opérations de *restriction*, *projection et jointure* de l'algèbre relationnelle.
- C'est la commande de SQL la plus fréquemment utilisée
- Sa syntaxe est la suivante :

La commande SELECT

SELECT *[**DISTINCT** | **ALL**] {* | <expression de colonne> [**AS** <nouveau nom>]*
[, . . .]}
FROM *<nom de table> [<alias>] [, . . .]*
*[[**WHERE**] < condition de recherche>]*
*[**GROUP BY** <liste de colonnes>[**HAVING** <condition de recherche>]*
*[**ORDER BY** <liste de colonnes>]*

- L'ordre des différentes clauses dans une commande ***SELECT*** ne peut être changé.
- Les deux clauses obligatoires sont les deux premières : ***SELECT*** et ***FROM*** ; les autres sont optionnelles.
- L'opération ***SELECT*** est ***fermée*** (i.e., le résultat d'une requête sur une table est une autre table)
- Il y a plusieurs variantes de la commande ***SELECT***

Exemple

Usine(NU,NOMU,Adesse,tel,fax)

Fournisseur(NUMF, NOMF,Adresse,Tel,Fax)

Produit (NP, NOMP,Couleur,Poids)

PUF(NU,NF,NP, Date,QTE)

Exemples

Usine(NU,NOMU,Adresse,tel,fax)
Fournisseur(NUMF, NOMF, Adresse,Tel,Fax)
Produit (NP, NOMP,Couleur,Poids)
PUF(NU,NF,NP, Date,QTE)

- (Q1) : Lister les noms et les couleurs des produits

```
SELECT NOMP, COULEUR
FROM PRODUIT;
```

```
SELECT ALL NOMP, COULEUR
FROM PRODUIT;
```

- (Q2) : Lister toutes les informations des produits

```
SELECT NP, NOMP, COULEUR
FROM PRODUIT;
```

```
SELECT *
FROM PRODUIT;
```

- (Q3) : Lister les noms et les poids des produits de couleur "Rouge"

```
SELECT NOMP,Poids
FROM PRODUIT
WHERE COULEUR='Rouge';
```

- (Q4) : Lister toutes les couleurs (éventuellement en doubles) des différents produits dont le poids > 10

```
SELECT COULEUR
FROM PRODUIT
WHERE Poids > 10;
```

Exemple

Usine(NU,NOMU,adresse,tel,fax)

Fournisseur(NUMF, NOMF, Adresse,Tel,Fax)

Produit (NP, NOMP,Couleur,Poids)

PUF(NU,NF,NP, Date,QTE)

- (Q5) : Lister toutes les couleurs (sans doubles) des différents produits dont le poids > 10

```
SELECT DISTINCT COULEUR
FROM PRODUIT
WHERE poids > 10;
```

- (Q6) : Lister les désignations et les adresses de fournisseurs n'ayant pas de numéro de fax

```
SELECT NOMF, ADRESSE
FROM FOURNISSEUR
WHERE FAX IS NULL;
```

- (Q7) : Lister les désignations et les adresses de fournisseurs ayant un numéro de fax

```
SELECT NOMF, ADRESSE
FROM FOURNISSEUR
WHERE FAX IS NOT NULL;
```

Condition de recherche

Forme générale d'une condition de recherche

*<condition de recherche> ::= [**NOT**]*

<nom_colonne> θ constante | <nom_colonne>

*θ **::=** > | < | = | <> | >= |<=*

*<nom_colonne> **LIKE** <modèle_de_chaîne>*

*<nom_colonne> **IN** <liste_de_valeurs>*

*<nom_colonne> θ(**ALL** | **ANY** | **SOME**) <liste de valeurs>*

*<nom_colonne> **BETWEEN** constante **AND** constante*

*<condition de recherche > **AND** | **OR** <condition de recherche >*

Exemple

Usine(NU,NOMU,adresse,tel,fax)

Fournisseur(NUMF, NOMF, Adresse,Tel,Fax)

Produit (NP, NOMP,Couleur,Poids)

PUF(NU,NF,NP, Date,QTE)

- (Q8) : Lister **les noms des produits** dont **la couleur est soit Rouge, Bleu, Vert ou Marron**

  SELECT NOMP

  ***FROM** PRODUIT*

  ***WHERE** COULEUR IN ('Rouge', 'Bleu', 'Vert', 'Marron');*
- (Q9) : Lister les noms des produits dont la couleur est différente de Rouge et Bleu

  ***SELECT** NOMP*

  ***FROM** PRODUIT*

  ***WHERE** COULEUR **NOT IN** ('Rouge', 'Bleu');*
- (Q10) : Lister les noms des produits dont le poids est >= 100 et <=200

  ***SELECT** NOMP*

  ***FROM** PRODUIT*

  ***WHERE** Poids **BETWEEN** 100 **AND** 200;*

Modèle de Chaine

Å constante de chaîne de caractères pouvant contenir le caractères génériques:

- ï "%" :signifie une quelconque sous-chaîne de caractères,
- ï "_" (soulignement) : signifie un quelconque caractère

Exemple

Usine(NU,NOMU,Adresse,tel,fax)

Fournisseur(NUMF, NOMF, Adresse,Tel,Fax)

Produit (NP, NOMP,Couleur,Poids)

PUF(NU,NF,NP, Date,QTE)

- (Q11) : Lister les noms et les poids des produits dont le nom commence par 'B'.

  SELECT *NOMP, poids*
  FROM *PRODUIT*
  WHERE *NOMP* ***LIKE*** *'B%';*

- (Q12) : Lister les noms des produits contenant la chaine "SAPIN"

  SELECT *NOMP*
  FROM *PRODUIT*
  WHERE NOMP LIKE *'%SAPIN%';*

- (Q13) : Lister les noms des produits dont le 1ère et le 3ème caractères sont respectivement 'C' et 'd'

  SELECT *NOMP*
  FROM *PRODUIT*
  WHERE NOMP LIKE *"C_d%';*

Qualification des attributs

- **Un attribut qualifié** est un attribut accompagné du nom de la table auquel il appartient

- Notation attribut qualifié:
  - <nom_table>.<nom_attribut>

- La qualification est nécessaire lorsque le **nom_attribut** existe dans plusieurs tables utilisées dans la même requête.

Recherche sur plusieurs tables jointure

Å Forme générale

SELECT A1, A2,…..

FROM T1,T2,…

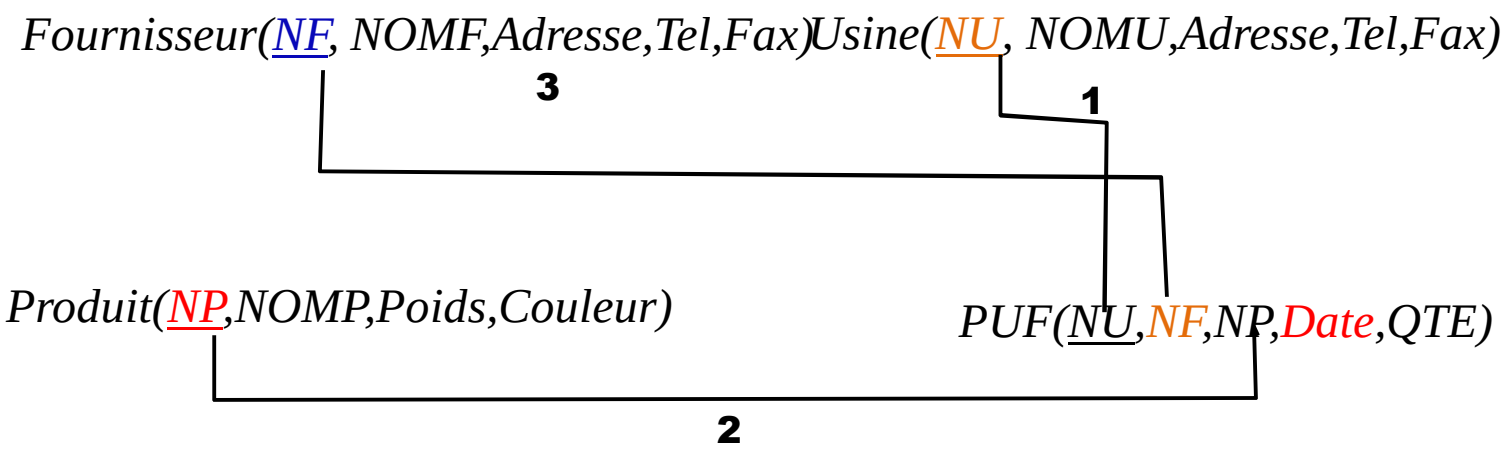
WHERE <condition de jointure> and

[<condition de la requête>]

Å Lister les **Noms des Usines** qui ont **acheté** un **produit de couleur 'Rouge'** **fourni** par **un fournisseur 'Algérois**ô

SELECT DISTINCT *U.NOMU*

FROM *Usine U,* ***PUF****, PRODUIT P, FOURNISSEUR* ***AS*** *F*



WHERE

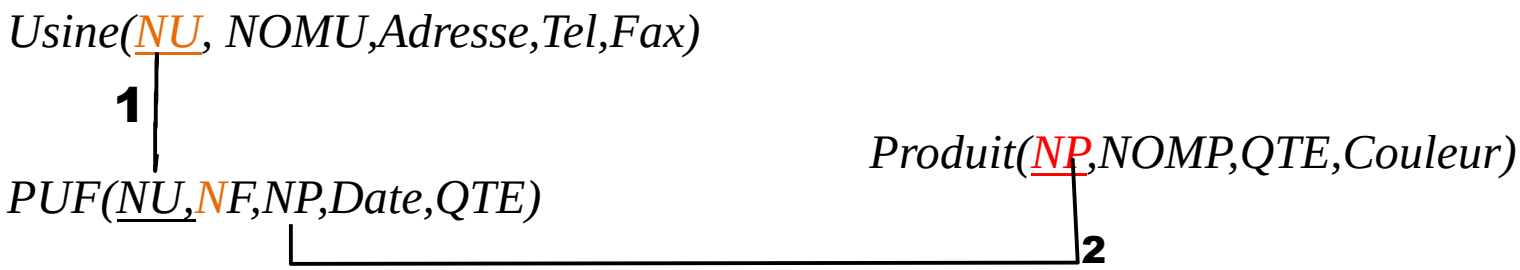
1 *U.NU=PUF.NU* ***AND***
2 *P.NP=PUF.NP* ***AND***
3 *F.NF=PUF.NF*

Condition de jointure

AND *COULEUR='Rouge'* ***AND*** *F.ADRESSE* ***LIKE*** *'%Alger%'*;

Condition de La requête

Lister les **NOMS et les adresses des Usines avec les noms et les couleurs des produits** achetés, en triant les résultats suivant **l'ordre décroissant** des Noms des usines et **l'ordre croissant** des noms et des couleurs des produits.



SELECT *NOMU, ADRESSE, NOMP as NOM_PRODUIT, COULEUR*
FROM *Usine , PUF, PRODUIT*

WHERE

1 *Usine.NU=PUF.NU* ***AND***
2 *PRODUIT.NP= PUF.NP*

Condition de jointure

ORDER BY NOMU DESC, NOMP, COULEUR ASC;

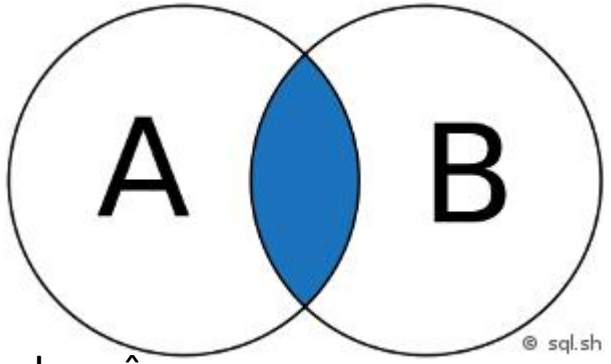
Ordre d'apparition

Variantes jointure

- SQL offre d'autres variantes de la jointure:
- INNER JOIN équivalente La jointure déjà vue mais avec une autre syntaxe,
- NATURAL JOIN
- LEFT JOIN
- RIGHT JOIN
- CROSS JOIN
- FULL JOIN
- SELF JOIN
- Nous allons présenter seulement les 2 premières jointures

INNER JOIN et NATURAL JOIN

```
SELECT *
FROM table1 INNER JOIN table2 ON table1.id=table2.fk_id
```



NATURAL JOIN s'effectue à la condition qu'il y ai des colonnes du même nom et de même type dans les 2 tables. Il n'ya pas de condition de jointure, elle est faite implicitement (égalité des colonnes de même nom)

```
SELECT *
FROM table1 NATURAL JOIN table2
```

Sous Requêtes

- Dans le langage SQL une **sous-requête** (aussi appelé “**requête imbriquée**” ou “**requête en cascade**”) consiste à exécuter une requête à l’intérieur d’une autre requête.

Sous requête: syntaxe

- Il y a plusieurs façons d'utiliser les sous-requêtes.
  - **Requête imbriquée qui retourne un seul résultat**
  - **Requête imbriquée qui retourne une colonne**

Requête imbriquée qui retourne un seul résultat

Produit(NP,NOMP,Couleur,poids)

SELECT <attributs>
FROM `table`
WHERE `nom_colonne` = (SELECT `valeur`
FROM `table2`
WHERE <condition>)

Exemple: donner les numéros de produits ayant un poids égale au produit "1"

SELECT NP
FROM PRODUIT
WHERE poids **=(SELECT poids**
FROM PRODUIT
WHERE NP="1");

Cette sous requête retourne une valeur qui est la quantité en stock du produit « 1»

À la place de '**=**', on peut aussi utiliser les autres opérateurs de comparaison **(>, <, >=, <=)**

Exemple: donner les numéros de produits ayant un poids égale au produit "1"

Produit(NP,NOMP,Couleur,poids)

```
SELECT NP
FROM PRODUIT
WHERE poids  =  (SELECT poids
                 FROM PRODUIT
                 WHERE NP="1");
```

NP	NomP	Couleur	Poids
1	PP1	Rouge	23
2	PP2	Vert	25
3	PP3	Noir	20
4	PP4	Rouge	22
5	PP5	Rouge	33
6	PP6	Noir	21
7	PP7	Rouge	65
8	PP8	Vert	34
9	PP9	Noir	32
10	PP10	Rouge	33
11	PP11	Blanc	43
12	PP12	Rouge	23
13	PP13	Vert	21
14	PP14	Noir	45

```
(SELECT poids
 FROM PRODUIT
 WHERE NP="1");
```

Poids

23

```
SELECT NP
FROM PRODUIT
WHERE Poids = 23
```

NP
1
12

Requête imbriquée qui retourne une colonne (ensemble de valeurs)

PUF(NU,NF,NP, Date,QTE)

SELECT <attributs>
FROM `table`
WHERE `nom_colonne1` **IN** (SELECT 'nom_colonne2'
FROM `table2`
WHERE <condition>)

Exemple: Donner les numéros d'usine ayant acheté un produit fourni par le fournisseur numéro "3"

SELECT NU
FROM PUF
WHERE NP **IN**

SELECT NP
FROM PUF
WHERE NF="3");

Cette sous requête retourne une colonne qui sont les produits fournis par le fournisseur « 3 »

▪

Exemple: Donner les numéros d'usine ayant acheté un produit fourni par le fournisseur numéro "3"

PUF(NU,NF,NP, Date,QTE)

```
SELECT DISTINCT NU
FROM PUF
WHERE NP IN (SELECT DISTINCT NP
             FROM PUF
             WHERE NF="3");
```

```
(SELECT DISTINCT NP
 FROM PUF
 WHERE NF="3");
```

NP
2
5
10
11

```
SELECT DISTINCT NU
FROM PUF
WHERE NP IN ('2','5','10','11')
```

NU
1
2
3
4
6

NP	NU	NF	Quantité
10	2	1	200
11	2	1	21
2	3	1	123
4	3	1	123
10	3	1	343
14	2	2	232
13	2	2	0
5	3	2	211
2	1	3	33
5	1	3	21
10	2	3	65
11	3	3	323
10	4	3	0
5	4	3	66
5	6	4	43
13	2	2	12
10	4	3	32
1	1	3	11
1	1	5	34

Requête imbriqué avec EXISTS

- Dans le langage SQL, la commande **EXISTS** s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête.
- **A noter :** cette commande n'est pas à confondre avec la clause **IN**:
  - La commande **EXISTS** vérifie si la sous-requête retourne un résultat ou non (retourne VRAI ou FAUX),
  - Tandis que **IN** vérifie la concordance d'une à plusieurs données.

EXISTS Syntaxe

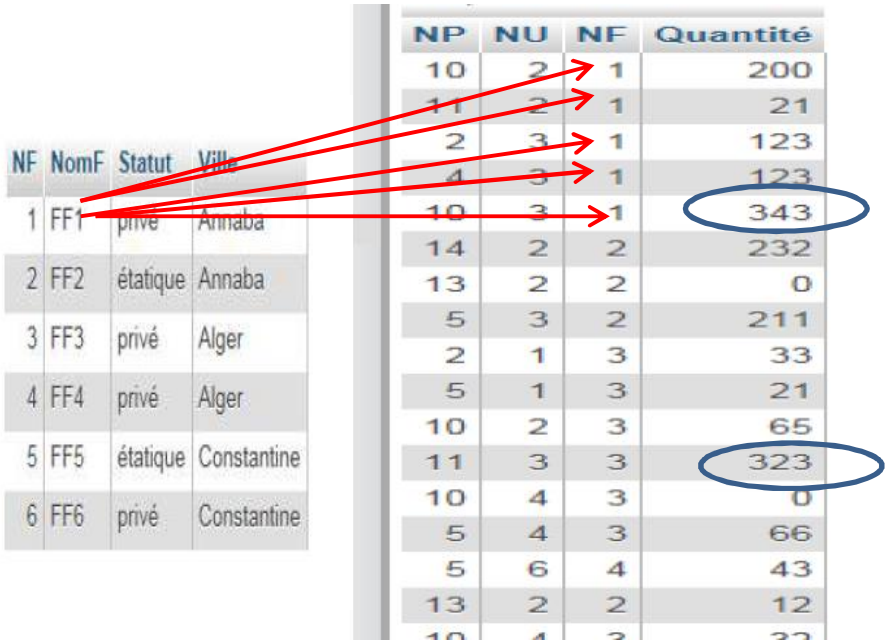
Å **SELECT** *column_name(s)*
FROM *table_name*
WHERE EXISTS (**SELECT** *column_name*
FROM *table_name*
WHERE *condition*);

SQL: requêtes imbriquées

Quels sont les noms des fournisseurs qui fournissent des produits en quantité >300

```
SELECT   NOMF
FROM F
WHERE  EXISTS  (SELECT  NP
                FROM PUF
                WHERE F.NF=PUF.NF and
                QTE>300 )
```

NomF
FF1
FF3



NF	NomF	Statut	Ville
1	FF1	privé	Annaba
2	FF2	étatique	Annaba
3	FF3	privé	Alger
4	FF4	privé	Alger
5	FF5	étatique	Constantine
6	FF6	privé	Constantine

NP	NU	NF	Quantité
10	2	1	200
11	2	1	21
2	3	1	123
4	3	1	123
10	3	1	343
14	2	2	232
13	2	2	0
5	3	2	211
2	1	3	33
5	1	3	21
10	2	3	65
11	3	3	323
10	4	3	0
5	4	3	66
5	6	4	43
13	2	2	12

Sous requête avec ALL

- La commande **ALL** permet de comparer une valeur dans l'ensemble de valeurs d'une sous-requête.
- Cette commande permet de s'assurer qu'une condition est "=", "!=", "<", ">", "<=" ou ">=" pour **tous** les résultats retournés par une sous-requête.

Syntaxe

SELECT *

FROM table1

WHERE Colonne θ ALL (SELECT *

FROM table2

WHERE condition2)

$\theta \in$ {“=“,”>”,”<“,…}

Donner les noms des produits dont le **poids** est **supérieur à tous** les **produits verts**

```
SELECT NOMP
FROM P
WHERE Poids > ALL(SELECT Poids
                  FROM P
                  WHERE couleur='vert')
```

Produit(NP,NOMP,Couleur,poids)

NP	NomP	Couleur	Poids
1	PP1	Rouge	23
2	PP2	Vert	25
3	PP3	Noir	20
4	PP4	Rouge	22
5	PP5	Rouge	33
6	PP6	Noir	21
7	PP7	Rouge	65
8	PP8	Vert	34
9	PP9	Noir	32
10	PP10	Rouge	33
11	PP11	Blanc	43
12	PP12	Rouge	23
13	PP13	Vert	21
14	PP14	Noir	45

```
(SELECT Poids
FROM P
WHERE couleur='vert')
```

Poids
25
34
21

```
SELECT NOMP
FROM P
WHERE Poids > ALL (25,34,21)
```

NOMP
PP7
PP11
PP14

Requête imbriquée (SOME/ANY)

- La commande ANY (ou SOME) permet de comparer une valeur avec le résultat d'une sous-requête.
- Il est ainsi possible de vérifier si une valeur est "=", "!=", ">", ">=", "<" ou "<=" pour **au moins une des valeurs** de la sous-requête.

syntaxe

```
SELECT *
FROM table1
WHERE  condition > ANY ( SELECT *
                         FROM table2
                          WHERE condition2 )
```

sélectionner toutes les colonnes de table1, où la condition est supérieure à **au moins** un résultat de la sous-requête.

Donner les noms des produits dont le **poids** est **égal à au moins un produits verts**

```
SELECT NOMP
FROM P
WHERE Poids = ANY (SELECT Poids
                   FROM P
                   WHERE couleur='vert')
```

Produit(NP,NOMP,Couleur,poids)

NP	NomP	Couleur	Poids
1	PP1	Rouge	23
2	PP2	Vert	25
3	PP3	Noir	20
4	PP4	Rouge	22
5	PP5	Rouge	33
6	PP6	Noir	21
7	PP7	Rouge	65
8	PP8	Vert	34
9	PP9	Noir	32
10	PP10	Rouge	33
11	PP11	Blanc	43
12	PP12	Rouge	23
13	PP13	Vert	21
14	PP14	Noir	45

```
(SELECT Poids
FROM P
WHERE couleur='vert')
```

Poids
25
34
21

```
SELECT NOMP
FROM P
WHERE Poids = ANY (25,34,21)
```

NOMP
PP6
PP13

Opérateurs ensemblistes

- Les opérations d'union, d'intersection et de différence de l'algèbre relationnelle s'expriment en SQL par :
  *<Clause Select> <Opération> [**ALL**] <Clause Select>*
  *<Opération> ::= '**UNION**' | '**INTERSECT**' | '**MINUS**' | '**EXCEPT**'*

- Si le mots clé ***ALL*** est spécifié, le résultat de l'opération *<Opération>* peut inclure les tuples en doubles.
- Certains dialectes de SQL ne supportent pas l'opération d'intersection et de différence; d'autres utilisent **'MINUS'** à la place de ***'EXCEPT'***.

Fonction d'Agrégation

Il existe des possibilités de calcul de fonctions. Les fonctions implantés sont :

- ***COUNT*** qui permet de compter le nombre de valeurs d'un ensemble;
- ***SUM*** qui permet de sommer les valeurs d'un ensemble;
- ***AVG*** qui permet de calculer la valeur moyenne d'un ensemble;
- ***MAX*** qui permet de calculer la valeur maximum d'un ensemble;
- ***MIN*** qui permet de calculer la valeur minimum d'un ensemble.

Å Ces fonctions peuvent être utilisées dans la clause SELECT. Elles sont aussi utilisables pour effectuer des **calculs d'agrégats**.

Å **Un agrégat** est un partitionnement horizontale d'une table en sous-tables en fonction des valeurs d'un ou de plusieurs attributs de partitionnement, suivi de l'application d'une fonction de calcul à chaque attribut des sous-tables obtenus.

GROUP BY & HAVING

- le partitionnement s'exprime par la clause :
  GROUP BY *<Spécification de Colonne>*$^{+}$
- La clause **GROUP BY** précise les attributs de partitionnement,
- Les fonctions de calculs doivent être appliquées aux ensembles générés par GROUP BY
- Ces fonctions sont appliquées sur les colonnes après ***SELECT***.
- Une restriction (sélection) peut être appliquée:
  - avant le calcul de **l'agrégat** au niveau de clause **WHERE**,
  - mais aussi après calcul de **l'agrégat** sur les résultats de ce dernier en utilisant une clause spéciale:

  HAVING *<Expression de Valeurs>*$^{+}$

Syntaxe

SELECT **colonne1**, Fonction_Calcul(colonne2)

FROM nom_table

[WHERE condition sur les tuples]

GROUP BY **colonne1**

[HAVING Fonction_Calcul(colonne2) θ **valeur**
(condition sur les groupements]

Fonction_Calcul="COUNT","AVG","SUM"….

Avec $\theta \in$ {"=","<",">",…}

SQL: Fonctions d'agrégation

Calculer le nombre de couleurs distinctes des produits

SELECT COUNT(DISTINCT *COULEUR) AS Nombre-couleurs*
FROM *PRODUIT*

Nombre-couleurs
4

NP	NomP	Couleur	Poids
1	PP1	Rouge	23
2	PP2	Vert	25
3	PP3	Noir	20
4	PP4	Rouge	22
5	PP5	Rouge	33
6	PP6	Noir	21
7	PP7	Rouge	65
8	PP8	Vert	34
9	PP9	Noir	32
10	PP10	Rouge	33
11	PP11	Blanc	43
12	PP12	Rouge	23
13	PP13	Vert	21
14	PP14	Noir	45

Calculer le nombre de produits par **couleur**.

SELECT COULEUR, COUNT(*) *AS Nombre*
FROM *PRODUIT*
GROUP BY COULEUR;

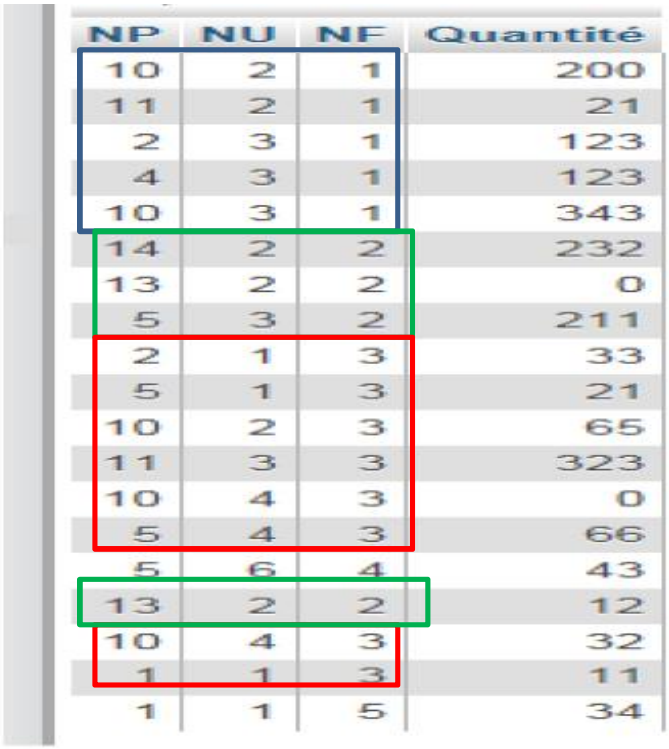
Couleur	Nombre
Rouge	6
Vert	3
Noir	4
Blanc	1

	rouge	Vert	Noir	Blanc
	PP1, PP4, PP5, PP7, PP10, PP12	PP2, PP8, PP13	PP3, PP6, PP9,PP14	PP11
COUNT	6	3	4	1

Calculer le nombre de produits distincts vendus par Fournisseur

```
SELECT NF, COUNT(DISTINCT NP)
FROM PUF
GROUP BY NF;
```

NF	Count (NP)	Count (distinct NP)
1	5	4
2	4	3
3	8	4
4	1	1
5	1	1



NP	NU	NF	Quantité
10	2	1	200
11	2	1	21
2	3	1	123
4	3	1	123
10	3	1	343
14	2	2	232
13	2	2	0
5	3	2	211
2	1	3	33
5	1	3	21
10	2	3	65
11	3	3	323
10	4	3	0
5	4	3	66
5	6	4	43
13	2	2	12
10	4	3	32
1	1	3	11
1	1	5	34

Calculer la moyenne des quantités vendues par fournisseur ayant des quantités minimales supérieur à 30

```
SELECT NF, AVG(QTES)
FROM PUF
GROUP BY NF
HAVING MIN(QTES) >30
```

NF	MIN(Qte)	Average(qte)
1	21	
2	0	
3	0	
4	43	
5	34	

NF	Average
4	43
5	34

NP	NU	NF	Quantité
10	2	1	200
11	2	1	21
2	3	1	123
4	3	1	123
10	3	1	343
14	2	2	232
13	2	2	0
5	3	2	211
2	1	3	33
5	1	3	21
10	2	3	65
11	3	3	323
10	4	3	0
5	4	3	66
5	6	4	43
13	2	2	12
10	4	3	32
1	1	3	11
1	1	5	34

Opérations de mise à jour

- INSERT INTO: permet d'ajouter un tuple ou un ensemble de tuples dans une table
- Update: permet de modifier une valeur dans un/plusieurs tuple (s). Elle est accompagnée de la clause WHERE
- DELETE: supprime les lignes indiquées dans une requête (clause where), ou bien toutes les lignes de la tables lorsque la clause where est absente.

INSERT INTO

INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)

INSERT INTO table (nom_colonne_1, nom_colonne_2, ... VALUES ('valeur 1', 'valeur 2', ...)

INSERT INTO client (prenom, nom, ville, age) VALUES ('Rébecca', 'Armand', 'Saint-Didier-des-Bois', 24), ('Aimée', 'Hebert', 'Marigny-le-Châtel', 36), ('Marielle', 'Ribeiro', 'Maillères', 27), ('Hilaire', 'Savary', 'Conie-Molitard', 58);

UPDATE

- UPDATE table SET nom_colonne_1 = 'nouvelle valeur' WHERE ***condition***

- UPDATE table SET colonne_1 = 'valeur 1', colonne_2 = 'valeur 2', colonne_3 = 'valeur 3' WHERE ***condition***

DELETE

Å DELETE FROM `table` [WHERE condition]

Å DELETE FROM table

Merci
cours suivant SQL LDD