



BASES DE DONNÉES

COURS:

REQUÊTES SQL

Licence 2

Présenté par:
Dr.A BOUTORH

Informatique

PLUSIEURS NIVEAUX DE NORMALISATION

➤ **SQL 1**: Norme de base

➤ **SQL 2** : Extension de SQL 1

❑ Meilleur support des règles du relationnel

❑ Types de données plus variés

➤ **SQL 3** : Intégration du modèle objet

SQL ALTER TABLE

PERMET DE MODIFIER UNE TABLE EXISTANTE

➤ Ajouter une Colonne

ALTER TABLE Table **ADD** Nom_Colonne Type_Données

➤ Renommer une Colonne

ALTER TABLE Table **CHANGE** Ancien_Nom_Colonne
Nouveau_Nom_Colonne Type_Données

➤ Modifier une Colonne

ALTER TABLE Table **MODIFY** Nom_Colonne Type_Données

➤ Supprimer une Colonne

ALTER TABLE Table **DROP** Nom_Colonne

REQUÊTES DE MISE À JOUR DES DONNÉES

➤ Ajout d'un Tuple

INSERT INTO Table **VALUES** (val₁, val₂, ..., val_n)

➤ Mise à jour d'un Attribut

UPDATE Table **SET** attribut_p=Val **WHERE** Condition

➤ Suppression de Tuples

DELETE FROM Table **WHERE** Condition

- **Exemple :Table Livre**

Cote	Titre	Auteur
17	Bases de Données	Ahmed
5	Langage SQL	Hadjer
9	SQL de Base	Yacine

- **Ajouter un Tuple à la table Livre:**

- **Cote = 20, Titre = Langage Java, Auteur = Amel**

INSERT INTO Livre **VALUES** (20, “Langage Java”, “Amel”)

- **Mettre à jour le titre de la cote 9 à « Requête SQL »**

UPDATE Livre **SET** Titre= “Requête SQL” **WHERE** Cote = 9

- **Supprimer le Tuple de l’auteur « Hadjer »**

DELETE FROM Livre **WHERE** Auteur = “Hadjer”

LES RÈGLES D'INTÉGRITÉ

- Les *règles d'intégrité* sont les **assertions** qui doivent être **vérifiées** par les données contenues dans une base.
- La gestion automatique des **contraintes d'intégrité** est l'un des outils les plus importants d'une base de données.

LES RÈGLES D'INTÉGRITÉ

➤ INTÉGRITÉ DE DOMAINE:

Les valeurs d'une colonne de relation doivent appartenir au **domaine** correspondant

➤ INTÉGRITÉ DE CLÉ:

Les valeurs de **clés primaires** doivent être : *uniques et non NULL*

➤ INTÉGRITÉ RÉFÉRENCIELLE:



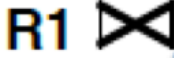
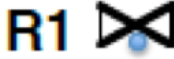

Les valeurs de **clés étrangères** sont 'NULL' ou sont des valeurs de la **clé primaire** auxquelles elles font référence.

- **Les contraintes de référence ont un impact important pour les opérations de mises à jour, elles permettent d'éviter les anomalies de mises à jour.**
- **Exemple :**
 - **Département** (**NumD**, NomD, Loc)
 - **Étudiant** (**Matricule**, Nom-E, Prénom-E, Année-Insc, **#No-Dep**)
- **Clé étrangère: No-Dep dans Étudiant**
- **Insertion** tuple **No-Dep = X** dans **Étudiant**
→ **Vérification** si **X** existe dans **Département**

- Les contraintes de référence ont un impact important pour les opérations de mises à jour, elles permettent d'éviter les anomalies de mises à jour.
- Exemple :
 - Département (**NumD**, NomD, Loc)
 - Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)
- Clé étrangère: No-Dep dans Étudiant
- Suppression tuple NumD = X dans Département
 - Soit interdire si X existe dans Étudiant
 - Soit supprimer en cascade tuple X dans Étudiant
 - Soit modifier en cascade X = NULL dans Étudiant

- Les contraintes de référence ont un impact important pour les opérations de mises à jour, elles permettent d'éviter les anomalies de mises à jour.
- Exemple :
 - Département (**NumD**, NomD, Loc)
 - Étudiant (**Matricule**, Nom-E, Prénom-E, Année-Insc, *#No-Dep*)
- Clé étrangère: **No-Dep** dans **Etudiant**
- **Modification** tuple **NumD = X** à **NumD = Y** dans **Département**
 - Soit **interdire** si **X existe** dans **Étudiant**
 - Soit **modifier** en cascade **No-Dep = Y** dans **Étudiant**

JOINTURES EN SQL

SQL 2	Opération	Algèbre
R1 CROSS JOIN R2	Produit Cartésien	R1 x R2
R1 JOIN R2 ON Condition	Jointure	R1  Condition R2
R1 LEFT / RIGHT / FULL OUTER JOIN R2 ON Condition	Jointure Externe	 Condition R2  Condition R2  Condition R2
R1 NATURAL JOIN R2	Jointure Naturelle	R1  R2 R1 * R2

JOINTURES EN SQL: **INNER JOIN**

- Commande (NumC, NomC, #NumP, Quantité)
- Fourniture (NumP, NumF, PrixA)
- **Question** : Donner les numéros, les prix d'achat et les numéros des fournisseurs des produits commandés par « Ahmed »

➤ En SQL de Base :

```
SELECT Commande.NumP, PrixA, NumF      FROM      Commande, Fourniture
WHERE Commande.NumP = Fourniture.NumP AND NonC = 'Ahmed'
```

➤ En SQL2 :

```
SELECT Commande.NumP, PrixA, NumF
FROM      Commande INNER JOIN Fourniture
ON        (Commande.NumP = Fourniture.NumP )
WHERE NonC = 'Ahmed'
```

*Remarque: **INNER** est facultatif dans la plupart des SGBDR*

JOINTURES EN SQL: **NATURAL JOIN**

- Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)
 - Département (NumD, NomD, Loc)
- **Question** : Donner les noms des départements avec les noms de leurs étudiants

➤ **En Algèbre Relationnelle** :

$\pi_{\text{NumD, Nom-E}} (\text{Département} \bowtie \text{Étudiant})$

➤ **En SQL** :

SELECT NumD, Nom-E

FROM Department **NATURAL JOIN** Étudiant

Remarque: **NATURAL JOIN** fait la jointure naturelle (sur l'attribut NumD)
L'attribut "NumD" n'apparaît qu'une seule fois dans la table résultat

JOINTURES EN SQL: **AUTO-JOINTURE**

- **Produit** (NumP, NomP, PrixV)

➤ **Question** : Donner les noms et les prix de vente des produits vendus avec un prix supérieur au prix de vente du produit numéro 6

➤ **En Algèbre Relationnelle** :

$R1 = \pi_{\text{PrixV}} (\sigma_{\text{NumP}=6} (\text{Produit}))$

$R2 = \pi_{\text{NomP}, \text{PrixV}} (\text{Produit}) \bowtie_{\text{Produit.PrixV} > R1.\text{PrixV}} R1$

➤ **En SQL** :

SELECT P1.NomP, P1.PrixV

FROM Produit P1 **JOIN** Produit P2 **ON** (P1.PrixV > P2.PrixV)

WHERE P2.NumP = 6

Remarque: P1 et P2 sont deux instances différentes de la table Produit

JOINTURES EN SQL: **AUTO-JOINTURE**

- Département (NumD, NomD, Loc)

➤ **Question** : Donner les noms des départements qui se trouvent deux à deux dans la même location

➤ **En SQL** :

SELECT Dep1.NomD, Dep2.NomD

FROM Département Dep1, Département Dep2

WHERE Dep1.Loc = Dep2.Loc AND Dep1.NumD < Dep2.NumD

*Remarque: Dep1 et Dep2 sont deux instances différentes de la table Département
La 2ème condition permet d'éliminer les paires (x,x) et éviter d'avoir en redondance les paires (x,y) et (y,x)*

JOINTURES EN SQL: JOINTURE EXTERNE PLEINE

FULL OUTER JOIN

Num	Nom	Adresse
12	Ahmed	Alger
100	Ilyes	Oran
45	Ahmed	Setif

R1

R1 ⋈ R2

Matr	Nom	Tel
23	Ahmed	057391
6	Amel	038143
90	Amel	046723

R2

➤ En SQL : R1 NATURAL FULL OUTER JOIN R2

Num	Nom	Adresse	Matr	Tel
12	Ahmed	Alger	23	057391
100	Ilyes	Oran	NULL	NULL
45	Ahmed	Setif	23	057391
NULL	Amel	NULL	6	038143
NULL	Amel	NULL	90	046723

Remarque: On garde **tous** les tuples des deux tables.

JOINTURES EN SQL: JOINTURE EXTERNE GAUCHE/DROITE LEFT/ RIGHT OUTER JOIN

R1

Num	Nom	Adresse
12	Ahmed	Alger
100	Ilyes	Oran
45	Amel	Setif

R2

Matr	Nom	Tel
23	Ahmed	057391
6	Amel	038143
90	Yacine	046723

➤ Jointure externe **Gauche** : **R1 NATURAL LEFT OUTER JOIN R2** // on garde les tuples de **R1**

Num	Nom	Adresse	Matr	Tel
12	Ahmed	Alger	23	057391
100	Ilyes	Oran	NULL	NULL
45	Amel	Setif	6	038143

➤ Jointure externe **Droite** : **R1 NATURAL RIGHT OUTER JOIN R2** // on garde les tuples de **R2**

Num	Nom	Adresse	Matr	Tel
12	Ahmed	Alger	23	057391
45	Amel	Setif	6	038143
NULL	Yacine	NULL	90	046723

EXPRESSIONS ENSEMBLISTES: UNION

R1

Matricule	Depart
1234	MI
5378	SNV
9264	MI
3274	SM

R2

Matricule	Année-Insc
1234	2018
6358	2019
9264	2015
4294	2018

- **Question :** Donner les matricules des étudiants du département « MI » ou ceux qui sont inscrits en 2018.

$$\begin{aligned} & [\pi_{\text{Matricule}} (\sigma_{\text{Depart}=\text{"MI"}} (\mathbf{R1}))] \\ \mathbf{U} & [\pi_{\text{Matricule}} (\sigma_{\text{Année-Insc}=2018} (\mathbf{R2}))] \end{aligned}$$

➤ **En SQL :**

SELECT Matricule **FROM** R1 **WHERE** Depart = "MI"

UNION

SELECT Matricule **FROM** R2 **WHERE** Année-Insc=2018

EXPRESSIONS ENSEMBLISTES: INTERSECTION

R1

Matricule	Depart
1234	MI
5378	SNV
9264	MI
3274	SM

R2

Matricule	Année-Insc
1234	2018
6358	2019
9264	2015
4294	2018

- **Question :** Donner les matricules des étudiants du département « MI » et ceux qui sont inscrits en 2018.

$$\left[\pi_{\text{Matricule}} (\sigma_{\text{Depart}=\text{"MI"}} (\mathbf{R1})) \right] \\ \cap \left[\pi_{\text{Matricule}} (\sigma_{\text{Année-Insc}=2018} (\mathbf{R2})) \right]$$

➤ **En SQL :**

SELECT Matricule **FROM** R1 **WHERE** Depart = "MI"

INTERSECT

SELECT Matricule **FROM** R2 **WHERE** Année-Insc=2018

EXPRESSIONS ENSEMBLISTES: DIFFÉRENCE

Département

NumD	NomD
1	MI
2	SM
3	SNV

Étudiant

Matricule	Nom-E	Prénom-E	Année-Insc	No-Dep
1234	Nom_3	Prénom_3	2018	1
9876	Nom_9	Prénom_9	2019	3

- **Question :** Donner les départements dont lesquels aucun étudiant n'est inscrit

$[\Pi_{\text{NumD}} (\text{Département})] - [\Pi_{\text{No-Dep}} (\text{Étudiant})]$

➤ **En SQL :**

SELECT NumD FROM Département

EXCEPT

SELECT No-Dep FROM Étudiant

IMBRICATION DE REQUÊTES

- **IN**
- **ANY**
- **ALL**
- **EXISTS**
- **Formes Equivalentes de Quantification**

REQUÊTES IMBRIQUÉES: CAS SIMPLE DE JOINTURE « IN »

- Commande (NumC, NomC, #NumP, Quantité). Fourniture (NumP, NumF, PrixA)

➤ **Question :** Donner les numéros, les prix d'achat et les numéros des fournisseurs des produits commandés par « Ahmed »

➤ **En Algèbre Relationnelle :**

$\Pi_{\text{NumP, PrixA, NumF}} (\sigma_{\text{NomC}='Ahmed'} (\text{Commande} \bowtie_{\text{C.NumP} = \text{F.NumP}} \text{Fourniture}))$

➤ **En SQL :**

SELECT NumP, PrixA, NumF

FROM Fourniture

WHERE NumP **IN** (**SELECT** NumP

FROM Commande

WHERE NonC = 'Ahmed')

Equivalent à:

SELECT NumP, PrixA, NumF
FROM Fourniture F, Commande C
WHERE F.NumP = C.NumP
AND NonC = 'Ahmed'

REQUÊTES IMBRIQUÉES: CAS SIMPLE DE DIFFÉRENCE « NOT IN »

- Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)
 - Département (NumD, NomD, Loc)
- **Question** : Donner les Départements dont lesquels aucun étudiant n'est inscrit
- **En Algèbre Relationnelle** :

$$[\Pi_{\text{NumD}} (\text{Département})] - [\Pi_{\text{no-Dep}} (\text{Étudiant})]$$

➤ **En SQL** :

```
SELECT NumD
FROM Département
WHERE NumD NOT IN ( SELECT DISTINCT No-Dep
                    FROM Étudiant)
```

Equivalent à:

```
SELECT NumD FROM Département
EXCEPT
SELECT No-Dep FROM Étudiant
```

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « ANY »

- Commande (NumC, NomC, #NumP, Quantité).
 - Fourniture (NumP, NumF, PrixA)
- **Question :** Donner les numéros des fournisseurs du produit numéro 40 à un prix d'achat supérieur au prix d'achat du produit numéro 20.

➤ **En SQL :**

SELECT NumF

FROM Fourniture

WHERE NumP= 40 **AND** PrixA > **ANY** (**SELECT** PrixA
FROM Fourniture
WHERE NumP= 20)

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « ANY »

- Commande (NumC, NomC, #NumP, Quantité).
 - Fourniture (NumP, NumF, PrixA)
- **Question** : Donner les numéros, les prix d'achat et les numéros des fournisseurs des produits commandés par « Ahmed »

➤ **En SQL :**

SELECT NumP, PrixA, NumF

FROM Fourniture

WHERE NumP = **ANY** (**SELECT** NumP

FROM Commande

WHERE NonC = 'Ahmed')

*Remarque: **IN** et **=ANY** sont utilisés de la même façon.*

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « ALL »

- Commande (NumC, NomC, #NumP, Quantité).
- **Question** : Donner les noms des clients ayant commandé la plus grande quantité du produit numéro 20

➤ **En SQL** :

SELECT NomC

FROM Commande

WHERE Quantité >= **ALL** (**SELECT** Quantité
FROM Commande
WHERE NumP= 20)

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « ALL »

- Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)
 - Département (NumD, NomD, Loc)
- **Question** : Donner les Départements dont lesquels aucun étudiant n'est inscrit

➤ **En SQL** :

SELECT NumD

FROM Département

WHERE NumD **NOT = ALL** (**SELECT DISTINCT** No-Dep
FROM Étudiant)

Remarque: NOT IN et NOT = ALL sont utilisés de la même façon.

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « EXISTS »

- Fournisseur (NumF, NomF, Statut, Ville).
 - Fourniture (NumP, NumF, PrixA)
- **Question** : Donner les numéros des fournisseurs qui fournissent au moins un produit

*Remarque: la condition EXISTS est vraie si et seulement si le résultat du bloc (SELECT * FROM) n'est pas vide*

➤ **En SQL :**

SELECT NumF

FROM Fournisseur

WHERE EXISTS (SELECT *

FROM Fourniture

WHERE Fournisseur.NumF = Fourniture.NumF)

REQUÊTES IMBRIQUÉES: CAS COMPLEXE « EXISTS »

- Fournisseur (NumF, NomF, Statut, Ville).
 - Fourniture (NumP, NumF, PrixA)
- **Question :** Donner les numéros des fournisseurs qui ne fournissent aucun produit

*Remarque: la condition **NOT EXISTS** est vraie si et seulement si le résultat du bloc (**SELECT * FROM ...**) est vide*

➤ **En SQL :**

SELECT NumF

FROM Fournisseur

WHERE NOT EXISTS (SELECT *

FROM Fourniture

WHERE Fournisseur.NumF = Fourniture.NumF)

REQUÊTES IMBRIQUÉES: FORMES ÉQUIVALENTES DE QUANTIFICATION

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- **Question** : Donner les numéros, les prix d'achat et les numéros des fournisseurs des produits commandés par « Ahmed »

➤ **En SQL :**

```
SELECT NumP, PrixA, NumF FROM Fourniture F
WHERE EXISTS ( SELECT * FROM Commande C
WHERE NonC = 'Ahmed' AND F.NumP = C.NumP )
```

➤ **Equivalent à :**

```
SELECT NumP, PrixA, NumF FROM Fourniture
WHERE NumP = ANY
( SELECT NumP FROM Commande WHERE NonC = 'Ahmed' )
```

REQUÊTES IMBRIQUÉES: FORMES ÉQUIVALENTES DE QUANTIFICATION

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- **Question :** Donner les numéros des fournisseurs qui fournissent au moins un produit avec un prix d'achat supérieur aux prix d'achat des produits fournis par le fournisseur N 30

➤ **En SQL :**

```
SELECT FN1.NumF FROM Fourniture FN1
WHERE NOT EXISTS ( SELECT * FROM Fourniture FN2
WHERE FN2.NumF = 30 AND FN1.PrixA <= FN2.PrixA )
```

➤ **Equivalent à :**

```
SELECT DISTINCT NumF FROM Fourniture
WHERE PrixA > ALL
( SELECT PrixA FROM Fourniture WHERE NumF = 30 )
```

FONCTIONS DE CALCULS

❖ **MAX ()**

❖ **MIN ()**

❖ **AVG ()**

❖ **COUNT ()**

❖ **SUM ()**

MAX

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- Produit (NumP, NomP, PrixV)

➤ **Question :** Donner le prix d'achat maximum du produit N 30

➤ **En SQL :**

SELECT MAX (PrixA)

FROM Fourniture

WHERE NumP = 30

MIN

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- Produit (NumP, NomP, PrixV)

➤ **Question :** Donner la quantité minimale commandée du produit N 30

➤ **En SQL :**

SELECT MIN (Quantité)

FROM Commande

WHERE NumP = 30

AVG

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- Produit (NumP, NomP, PrixV)

➤ **Question :** Donner le prix de vente moyen de plastique

➤ **En SQL :**

SELECT AVG (**PrixV**)

FROM Produit

WHERE NomP = 'plastique'

COUNT

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- Produit (NumP, NomP, PrixV)

➤ **Question :** Donner le nombre de fournisseurs d'Alger

```
SELECT COUNT (*) FROM Fournisseur WHERE Ville = 'Alger'
```

La fonction **COUNT (*)** compte le nombre de tuples du résultat d'une requête **sans** élimination de **tuples doubles**, ni vérification des **valeurs nulles**.

Dans le cas contraire, on utilise **COUNT (UNIQUE ...)**

➤ **Question:** Donner le nombre de fournisseurs qui fournissent des produits

```
SELECT COUNT (DISTINCT NumF) FROM Fourniture
```

SUM

- Commande (NumC, NomC, #NumP, Quantité).
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- Produit (NumP, NomP, PrixV)

➤ **Question :** Donner le prix d'achat total du produit numéro 30.

➤ **En SQL :**

SELECT SUM (**PrixA**)

FROM Fourniture

WHERE NumP = 30

OPÉRATEURS D'AGRÉGATION

❖ **GROUP BY**

❖ **ORDER BY**

❖ **HAVING**

GROUP BY

- Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)

➤ **Question** : Donner le nombre d'étudiants par département

➤ **En SQL** :

```
SELECT No-Dep, COUNT ( Nom-E) FROM Étudiant  
GROUP BY No-Dep
```

la clause **GROUP BY** permet de préciser les attributs de partitionnement des tables déclarées dans **FROM**

Exemple de BDD :
(le nom de département est donné dans l'exemple dans la colonne num pour clarification)

Nom-E	No-Dep
Ahmed	MI
Amel	SNV
Ahmed	SM
Amel	MI
Nadia	SM
Yacine	MI

Résultat de la requête :

NO-Dep	COUNT (NOM-E)
MI	3
SNV	1
SM	2

GROUP BY

- Fourniture (NumP, NumF, PrixA)

➤ **Question** : Donner pour chaque produit la moyenne de son prix d'achat.

➤ **En SQL** :

```
SELECT NumP, AVG (PrixA) FROM Fourniture GROUP BY NumP
```

*Les fonctions de calcul appliquées au résultat de regroupement sont directement indiquées dans la clause **SELECT***

Exemple
de BDD :

NumP	PrixA
34	230
2	300
25	100
34	190
25	150
34	260

Résultat de la requête :

NumP	AVG (PrixA)
34	226.67
2	300
25	125

ORDER BY

- Fourniture (NumP, NumF, PrixA).
 - Fournisseur (NumF, NomF, Statut, Ville)
- **Question** : Donner les villes, noms des fournisseurs et numéros des produits fournis triés par ordre **descendant** par ville comme 1^{er} critère de tri, numéro de produit comme second critère de tri et nom de fournisseur comme 3^{ème} critère.

➤ **En SQL** :

```
SELECT      Ville, NomF, NumP
FROM        Fourniture FN, Fournisseur FR
WHERE       FN.NumF = FR.NumF
ORDER BY    Ville, NumP, NomF DESC
```

ASC pour l'ordre **ASCENDANT**

HAVING

- Fourniture (NumP, NumF, PrixA).

➤ **Question** : Donner numéros des produits fournis par deux ou plusieurs fournisseurs avec un prix d'achat supérieur à 500 da.

➤ **En SQL** :

```
SELECT    NumP
FROM      Fourniture
WHERE     PrixA > 500
Group BY  NumP
HAVING    COUNT(*) >= 2
```

Résultat Final de
la Requête

NumP
34

La clause **HAVING** permet d'éliminer des **partitionnements**, comme la clause **WHERE** élimine des **Tuple** du résultat d'une requête

NumP	NumF	PrixA
34	16	530
2	8	30
25	23	550
34	2	612
25	8	402
34	23	620

NumP	NumF	PrixA
34	16	530
34	2	612
34	23	620

HAVING

- Fourniture (NumP, NumF, PrixA).
 - Fournisseur (NumF, NomF, Statut, Ville)
- **Question** : Donner numéros et prix d'achat moyens des produits fournis par les fournisseurs dont le siège est à **Alger**, seulement si leur prix d'achat minimum est supérieur à 500 da.

➤ **En SQL** :

```
SELECT      NumP, AVG (PrixA)
FROM        Fourniture FN, Fournisseur FR
WHERE        FN.NumF = FR.NumF AND FR.Ville = "Alger"
Group BY    NumP
HAVING      MIN (PrixA) > 500
```

DIVISION

➤ Il *n'existe pas* en **SQL** d'équivalent direct à l'opération de **la division**.

Cependant, il est toujours possible de trouver une autre solution, notamment par l'intermédiaire des opérations de **calcul** et de **regroupement**

➤ Dans l'exemple suivant, on souhaite trouver:

les personnes qui participent à toutes les compétitions scientifiques.

Table : Personne

Nom	NumComp
Ahmed	15
Nadia	40
Ahmed	20
Ahmed	8
Yacine	15
Asma	20
Ahmed	40
Nadia	20
Yacine	20

Table : Compétition

NumComp
20
40
15
8

DIVISION

➤ Requête 1 SQL :

SELECT Nom **FROM** Personne

GROUP BY Nom

HAVING COUNT(*) =

(**SELECT** COUNT (DISTINCT NumComp)

FROM Compétition)

Table : Personne

Nom	NumComp
Ahmed	15
Nadia	40
Ahmed	20
Ahmed	8
Yacine	15
Asma	20
Ahmed	40
Nadia	20
Yacine	20

Table : Compétition

NumComp
20
40
15
8

DIVISION

➤ Requête 2 SQL :

SELECT DISTINCT Nom

FROM Personne AS P1

WHERE NOT EXISTS

(SELECT * FROM Compétition AS C

WHERE NOT EXISTS

(SELECT * FROM Personne AS P2

WHERE P1.Nom = P2.Nom

AND P2.NumComp = C.NumComp)

)

Table : Personne

Nom	NumComp
Ahmed	15
Nadia	40
Ahmed	20
Ahmed	8
Yacine	15
Asma	20
Ahmed	40
Nadia	20
Yacine	20

Table : Compétition

NumComp
20
40
15
8

LES VUES

- On a vu que le résultat d'une requête SQL représente une relation. Cette dernière peut être stockée dans une **table 'virtuelle'**
- Les **tables virtuelles** ajoutées au schéma sont nommées **des vues** dans la terminologie relationnelle.
- On peut *interroger des vues* comme des tables stockées.
- Une **vue** n'induit aucun stockage puisqu'elle *n'existe pas physiquement*, et permet d'obtenir une représentation différente des tables sur lesquelles elle est basée.

LES VUES

- **Définition:** « Une vue est une table virtuelle calculée à partir des tables de base par une requête ».
- Une vue apparaît à l'utilisateur comme une table réelle, cependant les lignes d'une vue ne sont pas stockées dans la base de données.
- Les vues peuvent être utilisées pour cacher des données sensibles, ou pour montrer des données statistiques.

LES VUES : CREATION

- Une vue est essentiellement une *requête* à laquelle on a donné un *nom*. La syntaxe de création d'une vue est très simple :

CREATE VIEW <Nom-Vue>

AS <Requête SQL>

[**WITH CHECK OPTION**]

- **Exemple:** on veut créer une vue qui ne contient que les étudiants du département MI (code MI =1).

CREATE VIEW MISTUDENTS

AS SELECT * FROM Étudiant WHERE No-Dep = 1

LES VUES : CREATION

- **CREATE VIEW** : Crée la définition d'une vue.

```
CREATE VIEW <Nom-Vue> [ (Liste de colonnes) ]  
AS <Requête SQL>  
[WITH CHECK OPTION]
```

- **Exemple**: Créer une vue qui contient les étudiants en L2.

```
CREATE VIEW L2STUDENTS (Mat-S, Nom-S, Année-S)  
AS SELECT Matricule, Nom-E, Année-Insc  
FROM Étudiant  
WHERE Année-Insc = 2018
```

- Commande (NumC, NomC, #NumP, Quantité)

➤ **Question** : Créer une vue qui contient le numéro et la quantité des produits commandés par « Ahmed »

CREATE VIEW ProdQ

AS SELECT NumP, Quantité FROM Commande
WHERE NomC = 'Ahmed'

- Étudiant (Matricule, Nom-E, Prénom-E, Année-Insc, #No-Dep)

➤ **Question** : Créer une vue pour le nombre d'étudiants par département

CREATE VIEW NBE

AS SELECT No-Dep, COUNT (Nom-E) FROM Étudiant
GROUP BY No-Dep

LES VUES

- Un des intérêts des vues est de donner une représentation “dénormalisée” de la base, en regroupant des informations par des jointures. **Exemple:**
- Fourniture (NumP, NumF, PrixA)
- Fournisseur (NumF, NomF, Statut, Ville)
- On peut créer une vue **FNFR** donnant numéros des produits, prix d'achat et noms et villes des fournisseurs

```
CREATE VIEW FNFR (Num-P, Prix-A, Nom-F, Ville-F)  
AS SELECT NumP, PrixA, NomF, Ville  
FROM Fourniture FN, Fournisseur FR  
WHERE FN.NumF = FR.NumF
```

LES VUES

- Une vue peut être interrogée par des requêtes **SQL**
- **Exemple:** **FNFR** (Num-P, Prix-A, Nom-F, Ville-F)

Donner numéros des produits et noms des fournisseurs de la ville d'Alger

```
SELECT  Num-P, Nom-F
FROM    FNFR
WHERE   Ville-F = "Alger"
```

On **détruit** une vue avec la syntaxe courante **SQL** :

```
DROP VIEW FNFR
```

LES VUES : MISE À JOUR

- L'idée de **modifier** une vue s'agit de modifier la table qui sert de support à la vue.
- Il existe de **sévères restrictions** sur les droits d'**insérer** ou de **mettre-à-jour** des tables à travers les **vues**.
- **Exemple:** On souhaite insérer une ligne dans la vue **FNFR**.

INSERT INTO FNFR (Num-P, Prix-A, Nom-F, Ville-F)

VALUES (230, 500, 'Kamel', 'Oran');

- Cet ordre s'adresse à une vue issue de **deux tables** (*Fournissuer et Fourniture*). Il n'y a clairement pas **assez d'information** pour alimenter ces tables de manière cohérente.
- **L'insertion n'est pas possible** (*de même que toute mise à jour*).
- De telles vues sont dites **non modifiables**.

LES VUES : MISE À JOUR

- Les **règles** définissant les **vues modifiables** sont très strictes:
 - 1) La vue doit être basée sur une seule table.
 - 2) Toute colonne non référencée dans la vue doit pouvoir être mise à NULL ou disposer d'une valeur par défaut.
 - 3) On ne peut pas mettre-à-jour un attribut qui résulte d'un calcul ou d'une opération.
- Il est donc possible d'insérer, modifier ou détruire la table Etudiant au travers de la vue **MISTUDENTS**.

INSERT INTO MISTUDENTS

VALUES (2348, 'Nom_9','Prenom_9', 2015, 2)

Code
SM = 2

LES VUES : WITH CHECK OPTION

- **SQL2** propose l'option **WITH CHECK OPTION** permet de vérifier que les lignes insérées dans une table de base au-travers d'une vue vérifient les conditions exprimées dans la requête *i.e* les critères de sélection de la vue. Cela permet d'imposer des contraintes d'intégrité lors des mises à jour au travers la vue.

```
CREATE VIEW MISTUDENTS (Mat-S, Nom-S, Pr-S, Année-S, Dep-S)
AS SELECT *
FROM Etudiant
WHERE No-Dep = 1
WITH CHECK OPTION
```

L'insertion donnée en exemple précédent devient impossible.

```
INSERT INTO MISTUDENTS
```

```
VALUES (2348, 'Nom_9','Prenom_9', 2015, 2)
```


INTÉRÊTS DES VUES

➤ Indépendance logique

Le concept de vue permet d'assurer une indépendance des applications vis-à-vis des modifications du schéma

➤ Simplification d'accès

Les vues simplifient l'accès aux données en permettant par exemple une prédéfinition des jointures et en masquant ainsi à l'utilisateur l'existence de plusieurs tables. **Exemple** : la vue qui calcule les moyennes générales pourra être consultée par la requête **SELECT * FROM MoyGenerale**

➤ Confidentialités des données

Une vue permet d'éliminer des lignes et/ ou des colonnes sensibles dans une base

BIBLIOGRAPHIE

- GARDARIN, Georges. *Bases de données*. Editions Eyrolles, 2003.
- Bernard ESPINASSE, Le langage SQL. Aix-Marseille Université (AMU) Ecole Polytechnique Universitaire de Marseille, Janvier 2018.
- Concepts et langages des Bases de Données Relationnelles - SUPPORT DE COURS SGBD I, IUT de Nice – Département INFORMATIQUE.
- Philippe Rigaux , Cours de bases de données. Paris-Dauphine