Support de Cours : Algèbre Relationnelle & SQL

Séquence 4 : Grains 1.1-1.2

I- INTRODUCTION

L'objectif principal de ce cours est d'exploiter les fonctions d'agrégation en SQL pour synthétiser efficacement des ensembles de données et répondre à des besoins analytiques spécifiques. Dans ce contexte, SQL ne se limite pas à la manipulation basique des données (création de bases, insertion, requêtes simples), mais offre également des outils puissants pour réaliser des calculs mathématiques avancés. Les fonctions d'agrégation, en particulier, permettent de résumer des données complexes en générant des indicateurs clés tels que des totaux, des moyennes ou des décomptes, transformant ainsi des volumes d'informations brutes en insights exploitables. Nous explorons par la suite quelques fonctions usuelles d'agrégation :

II- LES FONCTIONS D'AGREGATION

EMPLOYES

ID	NOM	PRENOM	DATE_ENTREE	DEPARTEMENT	SALAIRE
S21364	JABBAD	HAMID	16/03/2020	COMPTABILITE	6824.50
E14545	KOTBI	HAYAT	09/04/2014	IT	9793.00
M11234	MAGHRAOUI	HANAE	26/09/2012	VENTES	7893.00
K15425	KETTANI	YASSINE	15/03/2018	RH	8800.50
S21512	FAID	RACHID	20/10/2014	MARKETING	11452.00
S42424	MARDI	SOUAD	26/04/2004	IT	15234.00
E17456	ZAIDI	MOHAMED	15/11/2018	MARKETING	9820.00
M25554	SEBTI	RIAD	11/01/2007	VENTES	7893.00
E14555	KADIRI	KHALID	01/03/2015	RH	10530.00
M19531	NACIRI	LEILA	09/04/2020	IT	7500.00

ABSENCES

ID	DAT_DE_AB	DAT_FIN_AB	NB_H	MOTIF
S21364	16/03/2024	16/03/2024	1.50	RETARD
E14545	09/04/2024	10/04/2024	16	ABSENCE NON JUSTIFIEE
S42424	04/11/2024	08/11/2024	40	CERTIFICAT MALADIE
K15425	09/04/2024	09/04/2024	2	RETARD
S21512	11/01/2024	11/01/2024	8	ABSENCE NON JUSTIFIEE
S42424	04/04/2024	04/07/2024	480	CONGE MATERNITE
E14545	15/11/2024	15/11/2024	1	RETARD
M25554	11/01/2024	11/01/2024	8	ABSENCE NON JUSTIFIEE
S21364	09/08/2024	09/08/2024	2	RETARD
K15425	07/03/2024	09/03/2024	24	CERTIFICAT MALADIE

NB_H (en heures) : 1 journée correspond à 8h de travail

a. La Fonction SUM ()

Rôle : Calcule la somme des valeurs d'une colonne numérique.

Exemple : Calculer la somme des salaires des employés de l'entreprise

EMPLOYES

EE	DEPARTEMENT	SALAIRE
0	COMPTABILITE	6824.50
4	IT	9793.00
2	VENTES	7893.00
8	RH	8800.50
4	MARKETING	11452.00
4	IT	15234.00
8	MARKETING	9820.00
7	VENTES	7893.00
5	RH	10530.00
0	IT	7500.00

→ SUM

POUR CALCULER LA SOMME DES SALAIRES

En SQL : `SELECT SUM (SALAIRE) FROM EMPLOYES;`

Aucun nom de colonne
95 740.00

Résultat : Une seule valeur (ex: 95 740,00 DH).

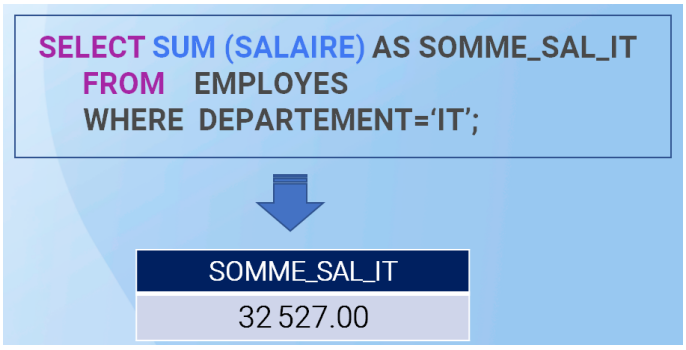
b. L'ALIAS AS :

Comme vous remarquez dans le résultat précédent on a pour le nom de la colonne : « **aucun nom de colonne** ». Pourquoi ? Tout simplement parce que c'est une nouvelle valeur. C'est pas quelque chose qui est déjà enregistrées dans la base de données c'est un calcul qui est nouvellement fait et que le SGBD ne connaît pas donc il ne sait pas quel nom lui attribuer. Dans ce cas, on a recours à l'alias « AS » pour donner un nom à ces résultats là. Cette fois ci, nous allons écrire : « **SELECT SUM (SALAIRE) AS SOMME_SALAIRE FROM EMPLOYES** » et là, on aura un tableau d'une colonne qui a le nom SOMME_SALAIRE qui lui a été attribuée grâce à l'alias « AS »

A noter : Les fonctions d'agrégation génèrent des colonnes sans nom dans les résultats, ce qui nécessite l'utilisation de l'alias AS pour attribuer un identifiant clair à ces colonnes. Cet alias permet non seulement de personnaliser le nom des colonnes calculées lors de l'exécution d'une requête SELECT, mais aussi de renommer temporairement une colonne dans l'affichage final (par exemple, pour rendre les résultats plus lisibles), assurant ainsi une interprétation cohérente et organisée des données retournées.

Une deuxième remarque : Les fonctions d'agrégation peuvent être aussi accompagnées de la clause WHERE. Par exemple, si on veut la somme des salaires de tous les employés travaillant dans le Département IT. La requête sera : « **SELECT SUM (SALAIRE) AS SOM-SAL-IT FROM EMPLYES WHERE DEPARTEMENT= IT**"

EE	DEPARTEMENT	SALAIRE
0	COMPTABILITE	6824.50
4	IT	9793.00
2	VENTES	7893.00
8	RH	8800.50
4	MARKETING	11452.00
4	IT	15234.00
8	MARKETING	9820.00
7	VENTES	7893.00
5	RH	10530.00
0	IT	7500.00



c. Les Fonctions MIN (), MAX ()

Rôle : **Min ()-Max() retourne la plus petite (grande) valeur de la colonne A**

Exemple : Calculer le salaire minimum (maximum) des employés de l'entreprise

```
SELECT MAX (salaire) AS Max_SAL FROM Employés; -- Résultat : 15 234 DH
SELECT MIN (salaire) AS Min_SAL FROM Employés; -- Résultat : 6824.50 DH
```

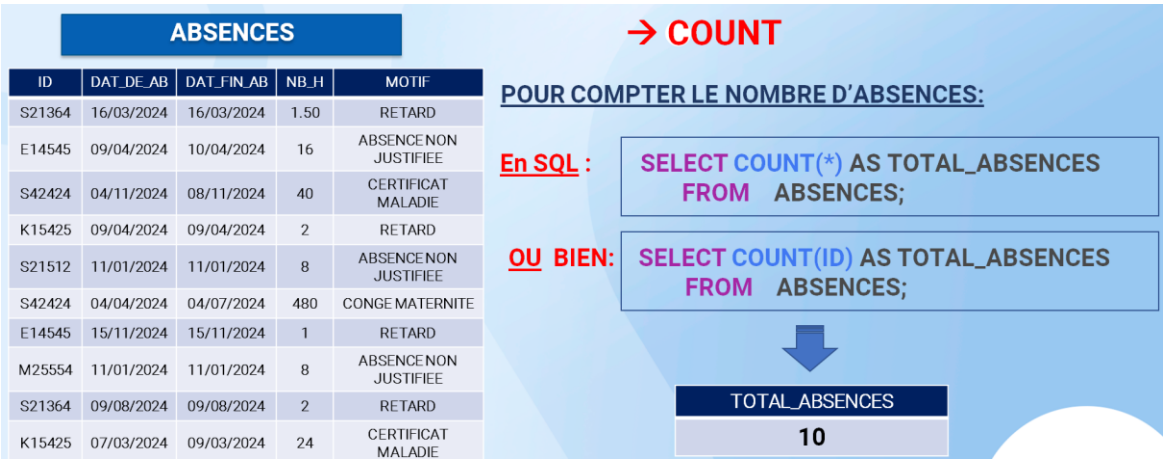
d. La Fonction AVG ()

Rôle : Calcule la valeur moyenne d'une colonne numérique.
Exemple : Calculer la moyenne des salaires, Cette opération effectue automatiquement la somme de tous les salaires, puis la divise par le nombre total d'employés de l'entreprise.

```
SELECT AVG (salaire) AS Moy_SAL FROM Employés; -- Résultat : 947.00 DH
```

e. La Fonction COUNT ()

Rôle : La fonction Count entre parenthèses colonne A permet de compter le nombre de valeurs de la colonne A.
Exemple : Compter le nombre d'absences dans la table ABSENCES



On peut également utiliser DISTINCT dans la fonction COUNT. Cela permet de retourner que le nombre de valeurs distinctes et uniques de la colonne A.

Exemple : Calculer le nombre d'employés ayant été absents, une requête simple comme : « **SELECT COUNT(*) AS TOTAL_ABSENCES FROM ABSENCES** » renverrait 10, mais ce résultat serait erroné. En effet, cette commande compte toutes les entrées de la table, y compris les absences multiples d'un même employé (comme ceux avec les ID S21364, E14545 ou S42424).

Pour obtenir uniquement le nombre d'employés distincts ayant été absents, il faut éviter les doublons en utilisant COUNT(DISTINCT ID_employé) . Ainsi, une approche correcte serait « **SELECT COUNT(DISTINCT ID_EMPLOYE) AS TOTAL_ABSENCES FROM ABSENCES** », ce qui donnerait **6**, correspondant au nombre réel d'employés absents, indépendamment du nombre de leurs absences enregistrées.

III- GROUP BY, HAVING

a. La commande GROUP BY

Rôle : L'instruction GROUP BY permet de regrouper des lignes partageant des valeurs identiques en lignes récapitulatives. Elle est généralement couplée aux fonctions d'agrégation vue précédemment comme : (COUNT(), MAX(), MIN(), SUM(), AVG(), etc..) afin d'appliquer des calculs sur chaque groupe défini par une ou plusieurs colonnes. La syntaxe est comme suit :

```
SELECT champ_names, fonction(champ2) [AS] alias1
FROM nom_table
GROUP BY champ_names;
```

Exemple 1 : Calculer la somme des salaires des employés pour chaque département

EE	DEPARTEMENT	SALAIRE
0	COMPTABILITE	6824.50
4	IT	9793.00
2	VENTES	7893.00
8	RH	8800.50
4	MARKETING	11452.00
4	IT	15234.00
8	MARKETING	9820.00
7	VENTES	7893.00
5	RH	10530.00
0	IT	7500.00

IT 9793.00 + 15234.00 + 7500.00 = 32527.00

VENTES 7893.00 + 7893.00 = 15786.00

RH 8800.50 + 10530.00 = 119330.50

MARKETING 11452.00 + 9820.00 = 21272.00

COMPTABILITE 6824.50

En SQL :

EMPLOYES

EE	DEPARTEMENT	SALAIRE
0	COMPTABILITE	6824.50
4	IT	9793.00
2	VENTES	7893.00
8	RH	8800.50
4	MARKETING	11452.00
4	IT	15234.00
8	MARKETING	9820.00
7	VENTES	7893.00
5	RH	10530.00
0	IT	7500.00

POUR CALCULER LA SOMME DES SALAIRES
POUR CHAQUE DEPARTEMENT

```
SELECT DEPARTEMENT, SUM (SALAIRE) AS SOMME_SALAIRE
FROM EMPLOYES
GROUP BY DEPARTEMENT ;
```

DEPARTEMENT	SOMME_SALAIRE
IT	32527.00
VENTES	15786.00
RH	119330.50
MARKETING	21272.00
COMPTABILITE	6824.50

Exemple 2 : Calculer le nombre des heures maximum pour chaque motif d'absence.

ABSENCES

ID	DAT_DE_AB	DAT_FIN_AB	NB_H	MOTIF
S21364	16/03/2024	16/03/2024	1.50	RETARD
E14545	09/04/2024	10/04/2024	16	ABSENCE NON JUSTIFIEE
S42424	04/11/2024	08/11/2024	40	CERTIFICAT MALADIE
K15425	09/04/2024	09/04/2024	2	RETARD
S42424	11/01/2024	11/01/2024	8	CERTIFICAT MALADIE
S42424	04/04/2024	04/07/2024	480	CONGE MATERNITE
E14545	15/11/2024	15/11/2024	1	RETARD
M25554	11/01/2024	11/01/2024	8	ABSENCE NON JUSTIFIEE
S21364	09/08/2024	09/08/2024	2	RETARD
K15425	07/03/2024	09/03/2024	24	CERTIFICAT MALADIE

RETARD — MAX(1.50 ; 2 ;1 ;2) = 2

AB. NON JUST. — MAX(16 ; 8) = 16

CERTIFICAT M. — MAX(40; 8 ; 24) = 40

CONGE MATER. — MAX(480) = 480

En SQL :

```
SELECT MOTIF, MAX (NB_H) AS MAX_NB_H
FROM ABSENCES
GROUP BY MOTIF ;
```

MOTIF	MAX_NB_H
RETARD	2
ABSENCE NON JUSTIFIEE	16
CERTIFICAT MALADIE	40
CONGE MATERNITE	480

NB. La clause GROUP BY permet aussi de regrouper les données selon plusieurs colonnes simultanément, offrant une granularité plus fine dans l'analyse. Cependant, une règle essentielle s'applique : les colonnes listées dans le GROUP BY doivent correspondre exactement à celles sélectionnées dans le SELECT (à l'exception des fonctions d'agrégation).

```
SELECT A, B, C, FCT_AGR(D)
FROM TABLE
GROUP BY A,B, X
```

Exemple 3 : Calculer le nombre des heures maximum par EMPLOYE et par MOTIF d'absence

Dans les cas précédents, le GROUP BY portait sur une seule colonne, mais il peut s'étendre à plusieurs colonnes pour des regroupements plus précis. Par exemple, pour obtenir la durée d'absence en heures par employé et par motif, nous devons définir ces deux colonnes dans le GROUP BY. Le processus consiste à regrouper les données avec une clause **GROUP BY** sur les colonnes *ID employé* et *Motif*, permettant de calculer la durée totale d'absence par employé et par type.

On commence par grouper par ID pour isoler chaque employé, puis par motif pour catégoriser les absences (ex. Retard, Certificat médical, ect).
Par exemple, un employé cumule 3,5h de retard (1,5h + 2h), un autre combine 48h de certificats médicaux et 480h de congé maternité, tandis qu'un troisième enregistre à la fois un retard (2h) et un certificat médical (24h). Un dernier employé n'a qu'une absence non justifiée.

ABSENCES

ID	DAT_DE_AB	DAT_FIN_AB	NB_H	MOTIF
S21364	16/03/2024	16/03/2024	1.50	RETARD
E14545	09/04/2024	10/04/2024	16	ABSENCE NON JUSTIFIEE
S42424	04/11/2024	08/11/2024	40	CERTIFICAT MALADIE
K15425	09/04/2024	09/04/2024	2	RETARD
S42424	11/01/2024	11/01/2024	8	CERTIFICAT MALADIE
S42424	04/04/2024	04/07/2024	480	CONGE MATERNITE
E14545	15/11/2024	15/11/2024	1	RETARD
M25554	11/01/2024	11/01/2024	8	ABSENCE NON JUSTIFIEE
S21364	09/08/2024	09/08/2024	2	RETARD
K15425	07/03/2024	09/03/2024	24	CERTIFICAT MALADIE

S21364 RETARD(1.50) + RETARD (2)
→ RETARD: 2 + 1.50 = 3.50
E14545 ABSENCE N. JUSTIFIEE(16) , RETARD (1)
→ RETARD : 1
→ ABSENCE N. JUSTIFIEE: 16
S42424 CERT. MEDICAL (40)+ CERT. MEDICAL (8) , CONGE MATE. (480)
→ CERT. MEDICAL : 40+8 = 48
→ CONGE MATERNITE = 480
K15425 RETARD (2), CERT.MEDICAL (24)
→ RETARD : 2
→ CERT. MEDICAL : 24
M25554 ABSENCE N. JUSTIFIEE(8)
→ ABSENCE N. JUSTIFIEE: 8

En SQL :

```
SELECT ID, MOTIF, SUM(NB_H) AS SOM_NB_H
FROM ABSENCES
GROUP BY ID, MOTIF;
```

ID	MOTIF	SOM_NB_H
S21364	RETARD	3.50
E14545	RETARD	1
E14545	ABSENCE NON JUSTIFIEE	16
S42424	CERTIFICAT MEDICAL	48
S42424	CONGE MATERNITE	480
K15425	RETARD	2
K15425	CERTIFICAT MEDICAL	24
M25554	ABSENCE NON JUSTIFIEE	8

b. La commande HAVING

Rôle : L'instruction HAVING indissociable du GROUP BY. Celle-ci permet de filtrer les groupes **après** leur création suite à l'application des fonctions d'agrégation. Sa syntaxe SQL s'intègre comme suit :

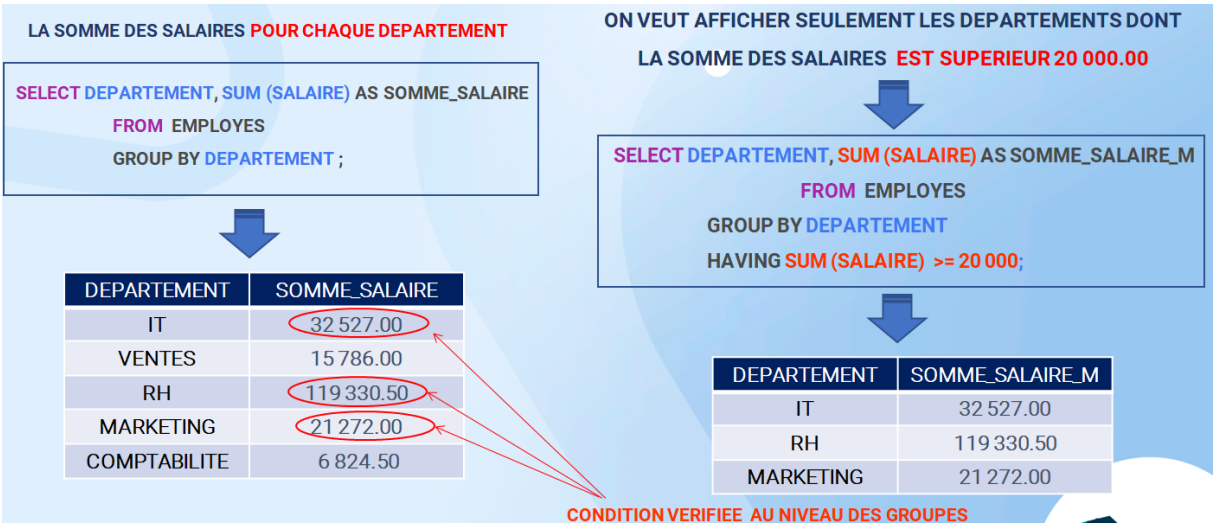
```
SELECT champ_names, fonction(champ2) [AS] alias1
FROM nom_table
GROUP BY champ_names;
HAVING fonction(champ2)  Opération_à_effectuer
```

c. WHERE vs. HAVING

Nous examinons ici une distinction clé entre WHERE et HAVING. La clause WHERE filtre les **lignes individuelles** d'une table *avant* tout regroupement ou agrégation, en s'appliquant directement après SELECT … FROM. En revanche, HAVING intervient *après* la création des groupes via GROUP BY, en filtrant les **groupes entiers** selon une condition portant sur les résultats agrégés. Ainsi, WHERE agit sur les données brutes, tandis que HAVING cible les données synthétisées, ce qui en fait des outils complémentaires mais à utiliser à des étapes distinctes de la requête.

Exemple 4 : Afficher seulement les départements dont la somme des salaires est supérieur à 20 000.00

Reprenons le cas où nous avons calculé la somme des salaires par département. Imaginons maintenant que nous souhaitions filtrer ces résultats pour n'afficher que les départements dont la somme des salaires dépasse 20 000. Pour cela, nous utilisons la requête suivante. Nous devons appliquer une condition non pas aux lignes brutes, mais aux groupes déjà formés via GROUP BY. La clause HAVING entre alors en jeu : elle permet d'utiliser un opérateur de comparaison (ex: > 20000) directement sur le résultat de la fonction d'agrégation (SUM(salaires)).



IV- ORDER BY, LIMIT

ORDER BY et **LIMIT** sont deux clauses essentielles en MySQL pour organiser et contrôler les résultats d'une requête :

a. ORDER BY : La clause ORDER BY permet de trier les résultats d'une requête selon une /(+) colonnes, soit par ordre croissant (ASC, par défaut) ou décroissant (DESC).

```
SELECT colonne1, colonne2
FROM table
ORDER BY colonne1 [ASC|DESC], colonne2 [ASC|DESC];
```

Exemple1 : Affiche les employés du salaire **le plus élevé au plus bas**.

```
SELECT nom, salaire
FROM Employés
ORDER BY salaire DESC;
```

b. LIMIT : La clause LIMIT restreint le nombre de lignes retournées par une requête, utile pour la pagination ou l'extraction d'un sous-ensemble de données.

```
SELECT colonne1, colonne2
FROM table
LIMIT nombre;
```

Exemple 2 : Retourne les 5 premiers employés.

```
SELECT nom
FROM Employés
LIMIT 5;
```

Exemple 3 : Affiche **les 3 employés les mieux payés**.

```
SELECT nom, salaire
FROM Employés
ORDER BY salaire DESC
LIMIT 3;
```

V- CONCLUSION

Pour conclure : Les fonctions d'agrégation (SUM, MIN, MAX, AVG, COUNT) permettent de synthétiser efficacement des ensembles de données complexes en fournissant des indicateurs clés (totaux, extrêmes, moyennes, décomptes), essentiels pour une prise de décision éclairée.
Ces fonctions peuvent être combinées dans une même requête et associées à des clauses comme WHERE pour segmenter l'analyse et générer des insights granulaires. L'utilisation de l'alias AS améliore la clarté des résultats en renommant les colonnes de sortie, facilitant ainsi l'interprétation des rapports.
D'autant plus, la clause GROUP BY permet d'organiser les données en groupes homogènes selon des critères spécifiques, facilitant l'analyse agrégée. La clause HAVING, quant à elle, filtre ces groupes après agrégation, contrairement à WHERE qui intervient avant.
La clause ORDER BY permet de trier les résultats d'une requête selon une /(+) colonnes, soit par ordre croissant (ASC, par défaut) ou décroissant (DESC) et La clause LIMIT restreint le nombre de lignes retournées par une requête, utile pour la pagination ou l'extraction d'un sous-ensemble de données. Ensemble, elles offrent une flexibilité puissante pour cibler des résultats précis dans des requêtes complexes impliquant des calculs sur des ensembles de données.
Nous approfondirons ces notions en classe à travers des exercices pratiques et des études de cas.