Agrégation (GROUP BY)

Table des matières

I. Contexte

Durée : 2h

Environnement de travail : DB Fiddle

Pré-requis : Aucun

Contexte

Lorsque l'on récupère des données d'une ou de plusieurs tables, il est souvent nécessaire de les combiner.

Les jointures permettent par exemple de combiner des données issues de plusieurs tables.

Mais lorsque ce ne sont plus des tables qu'ils faut combiner, mais des enregistrements issus d'une même table, comment faire ? On peut par exemple penser à des opérations classiques, telles que la moyenne d'un attribut ou le nombre d'enregistrements répondant à une certaine condition.

En SQL, ce sont les agrégations qui s'occupent de répondre à toutes ces questions.

II. Définition de l'agrégation (GROUP BY)

Mise en situation

Imaginez que gériez une base de donnée qui stocke les visites sur un site web. Vous gérez, par exemple, le temps passé sur chaque page par utilisateur. Chaque durée, prise individuellement, a peu d'intérêt.

En revanche, ce qui serait intéressant, c'est de pouvoir calculer le temps moyen passé sur chaque page.

Une première stratégie pourrait consister à récupérer toutes les données et à effectuer une moyenne *a posteriori*. Mais cette stratégie présente un gros inconvénient : si vous gérez un million d'enregistrements, tout ce volume doit être transféré alors que la seule information qui vous intéresse est la moyenne.

En SQL, ce genre de calculs peut s'effectuer directement à l'aide des **agrégations.**

Définition **Agrégation**

Une **agrégation** est un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'**un ou plusieurs attributs de partitionnement**, suivi de l'application éventuelle de fonctions de calcul sur d'autres attributs des sous-tables obtenues.

Définition **Agrégat**

Un **agrégat** le résultat d'une agrégation.

Synonyme : Regroupement

Syntaxe **Expression générale d'une agrégation**

```
SELECT liste ordonnée d'attributs de partionnement, liste d'application de fonctions sur d'utres attributs
FROM liste de relations
WHERE condition à appliquer avant calcul de l'agrégat
GROUP BY liste ordonnée d'attributs de partitionnement
```

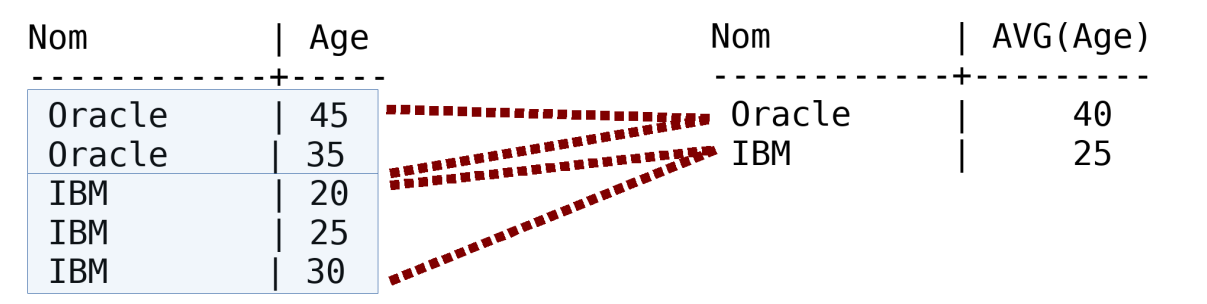
La table est divisée en sous-ensembles de lignes, avec un sous-ensemble pour chaque valeur différente des attributs de partitionnement dans le `SELECT`.

Les fonctions d'agrégation sont appliquées sur les attributs suivants pour chaque sous-ensemble.

Exemple **Âge moyen dans des entreprises**

```sql
SELECT nom_soc as nom, AVG(age)
FROM personne
GROUP BY nom_soc;
```

`p.nom_soc` est un ici le seul attribut de partitionnement, donc un sous-ensemble est créé pour chaque valeur différente de `p.nom_soc`, puis la moyenne (fonction `AVG`) est effectuée pour chaque sous-ensemble.



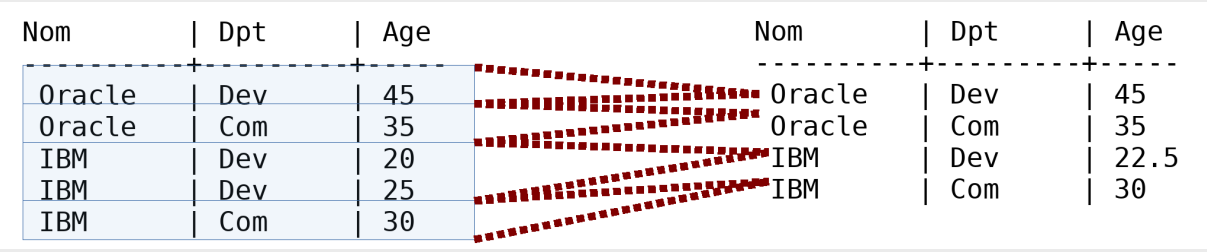
Agrégation avec un attribut et une fonction

Cette requête calcule l'âge moyen du personnel pour chaque société.

Exemple **Âge moyen dans des départements d'entreprises**

```sql
SELECT nom_soc AS nom, nom_dpt AS dpt, AVG(age) AS age
FROM personne
GROUP BY nom, dpt;
```

`nom_soc` et `nom_dpt` sont les deux attributs de partitionnement, donc un sous-ensemble est créé pour chaque valeur différente du couple (`nom_soc`, `nom_dpt`).



Agrégation avec deux attributs et une fonction

Cette requête calcule l'âge moyen du personnel pour chaque département de chaque société.

Remarque **Requête d'agrégation sans attributs de partitionnement**

Si la requête d'agrégation est définie sans attributs de partitionnements, les fonctions s'appliquent sur un seul groupe : tous les enregistrements disponibles.

Exemple **Âge moyen général**

```sql
SELECT AVG(age) AS age
FROM personne;
```

```
| age               |
```

```
| ------------------- |
| 31.0000000000000000 |
```

Remarque **Requête d'agrégation sur des attributs de partitionnement uniquement**

Si la requête d'agrégation est définie sans application de fonction d'agrégation, on obtient la liste des attributs de partitionnements.

Complément **Valeurs uniques sur le nom de société**

```sql
SELECT nom_soc AS nom
FROM personne
GROUP BY nom;
```

Ce qui est équivalent à

```sql
SELECT DISTINCT nom_soc AS nom
FROM personne;
```

```
| nom    |
| ------ |
| Oracle |
| IBM    |
```

Complément **Éléments de documentation**

On se référera aux sections suivantes de la documentation PostGreSQL :

- Sur la clause `GROUP BY` : https://docs.postgresql.fr/current/sql-select.html#sql-groupby
- Sur l'expression d'agrégats : https://docs.postgresql.fr/current/sql-expressions.html#syntax-aggregates

III. Exercice : Appliquer la notion

Soient les données représentant la population de villes européennes :

```sql
CREATE TABLE country (
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  population NUMERIC(3),
  PRIMARY KEY (countrycode)
);

CREATE TABLE city (
  citycode CHAR(3) NOT NULL,
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (citycode),
  FOREIGN KEY (countrycode) REFERENCES country(countrycode)
);

CREATE TABLE city_info (
  year_ INTEGER,
  citycode CHAR(3),
  population NUMERIC(2,1),
  PRIMARY KEY (year_, citycode),
  FOREIGN KEY (citycode) REFERENCES city(citycode)
```

```
);

INSERT INTO country VALUES ('ES', 'Spain', 46);
INSERT INTO country VALUES ('FR', 'France', 67);
INSERT INTO country VALUES ('DE', 'Germany', 82);

INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona');
INSERT INTO city VALUES ('MAD', 'ES', 'Madrid');
INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza');

INSERT INTO city_info VALUES(2014, 'BAR', 0.7);
INSERT INTO city_info VALUES(2015, 'BAR', 2.0);
INSERT INTO city_info VALUES(2016, 'BAR', 1.9);

INSERT INTO city_info VALUES(2014, 'MAD', 3.3);
INSERT INTO city_info VALUES(2015, 'MAD', 3.4);
INSERT INTO city_info VALUES(2016, 'MAD', 3.3);

INSERT INTO city_info VALUES(2014, 'ZAR', 0.7);
INSERT INTO city_info VALUES(2015, 'ZAR', 0.6);
INSERT INTO city_info VALUES(2016, 'ZAR', 0.7);

INSERT INTO city VALUES ('PAR', 'FR', 'Paris');
INSERT INTO city VALUES ('LYO', 'FR', 'Lyon');
INSERT INTO city VALUES ('LLL', 'FR', 'Lille');
INSERT INTO city VALUES ('AMN', 'FR', 'Amiens');

INSERT INTO city_info VALUES(2014, 'PAR', 2.0);
INSERT INTO city_info VALUES(2015, 'PAR', 2.1);
INSERT INTO city_info VALUES(2016, 'PAR', 2.2);

INSERT INTO city_info VALUES(2014, 'LYO', 0.4);
INSERT INTO city_info VALUES(2015, 'LYO', 0.6);
INSERT INTO city_info VALUES(2016, 'LYO', 0.5);

INSERT INTO city_info VALUES(2014, 'LLL', 0.2);
INSERT INTO city_info VALUES(2015, 'LLL', 0.2);
INSERT INTO city_info VALUES(2016, 'LLL', 0.2);

INSERT INTO city_info VALUES(2014, 'AMN', 0.1);
INSERT INTO city_info VALUES(2015, 'AMN', 0.1);
INSERT INTO city_info VALUES(2016, 'AMN', 0.1);


```

Question 1

On veut connaître le nombre de villes total dans la base.

Indice :

Écrire une requête SQL sans partitionnement qui utilise la fonction d'agrégation `COUNT` permettant d'obtenir cette information.

Question 2

On veut connaître le nombre de villes par pays.

Indice :

Utiliser à présent l'attribut de partitionnement `country`.

IV. Fonctions d'agrégation

Mise en situation

Dès que vous avez besoin de combiner des informations issues de plusieurs enregistrements, le bon réflexe est d'utiliser les agrégations.

Imaginez une base de donnée qui contient une table gérant les commandes sur un site de vente en ligne. Des informations intéressantes peuvent être extraites des enregistrements, comme le nombre de commandes pour un utilisateur, le montant total ou moyen de ses commandes, ou encore le montant minimum de ses commandes.

Toutes ces informations peuvent être calculées grâce aux fonctions d'agrégation SQL.

Définition **Fonctions d'agrégation**

Une **fonction d'agrégation** (ou fonction de regroupement) est une fonction qui s'applique aux valeurs des attributs des sous-ensembles d'un groupement.

Elle a pour résultat une valeur atomique (entier, chaîne, date, etc).

Exemple **Fonction AVG : moyenne**

```sql
SELECT AVG(population) AS "Population Moyenne"
FROM city_info;
```

Dans cet exemple la fonction d'agrégation est `AVG`. Elle produit comme valeur atomique la moyenne. Ici la moyenne est produite sur l'ensemble des valeurs de `population` de la table `city_info`.

Syntaxe

Les cinq fonctions prédéfinies sont :

- **`COUNT (Relation.Propriété)`**

  Renvoie le nombre de valeurs non nulles d'un attribut pour tous les tuples d'une relation ;
- **`SUM (Relation.Propriété)`**

  Renvoie la somme des valeurs d'un attribut des tuples (numériques) d'une relation ;
- **`AVG (Relation.Propriété)`**

  Renvoie la moyenne des valeurs d'un attribut des tuples (numériques) d'une relation ;
- **`MIN (Relation.Propriété)`**

  Renvoie la plus petite valeur d'un attribut parmi les tuples d'une relation ;
- **`MAX (Relation.Propriété)`**

  Renvoie la plus grande valeur d'un attribut parmi les tuples d'une relation.

Exemple **Âge minimum, maximum et moyen**

```sql
SELECT MIN(age), MAX(age), AVG(age)
FROM personne
WHERE qualification='Ingénieur';
```

Remarque **Fonctions de calcul sans partitionnement**

Si une ou plusieurs fonctions de calcul sont appliquées sans partitionnement, le résultat de la requête est un tuple unique.

Remarque **Compter le nombre d'enregistrement d'une relation**

Pour effectuer un comptage sur tous les tuples d'une relation, on applique la fonction `COUNT` sur les enregistrements.

```sql
SELECT COUNT(*)
FROM personne
WHERE qualification='Ingénieur';
```

Complément **Éléments de documentation sur les fonctions d'agrégation**

- Pour PostgreSQL : https://docs.postgresql.fr/current/functions-aggregate.html
- Pour MySQL : https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html

V. Exercice : Appliquer la notion

Soient les données représentant la population de villes européennes :

```sql
CREATE TABLE country (
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  population NUMERIC(3),
  PRIMARY KEY (countrycode)
);

CREATE TABLE city (
  citycode CHAR(3) NOT NULL,
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (citycode),
  FOREIGN KEY (countrycode) REFERENCES country(countrycode)
);

CREATE TABLE city_info (
  year_ INTEGER,
  citycode CHAR(3),
  population NUMERIC(2,1),
  PRIMARY KEY (year_, citycode),
  FOREIGN KEY (citycode) REFERENCES city(citycode)
);

```

```sql
INSERT INTO country VALUES ('ES', 'Spain', 46);
INSERT INTO country VALUES ('FR', 'France', 67);
INSERT INTO country VALUES ('DE', 'Germany', 82);

INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona');
INSERT INTO city VALUES ('MAD', 'ES', 'Madrid');
INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza');

INSERT INTO city_info VALUES(2014, 'BAR', 0.7);
```

```sql
INSERT INTO city_info VALUES(2015, 'BAR', 2.0);
INSERT INTO city_info VALUES(2016, 'BAR', 1.9);

INSERT INTO city_info VALUES(2014, 'MAD', 3.3);
INSERT INTO city_info VALUES(2015, 'MAD', 3.4);
INSERT INTO city_info VALUES(2016, 'MAD', 3.3);

INSERT INTO city_info VALUES(2014, 'ZAR', 0.7);
INSERT INTO city_info VALUES(2015, 'ZAR', 0.6);
INSERT INTO city_info VALUES(2016, 'ZAR', 0.7);

INSERT INTO city VALUES ('PAR', 'FR', 'Paris');
INSERT INTO city VALUES ('LYO', 'FR', 'Lyon');
INSERT INTO city VALUES ('LLL', 'FR', 'Lille');
INSERT INTO city VALUES ('AMN', 'FR', 'Amiens');

INSERT INTO city_info VALUES(2014, 'PAR', 2.0);
INSERT INTO city_info VALUES(2015, 'PAR', 2.1);
INSERT INTO city_info VALUES(2016, 'PAR', 2.2);

INSERT INTO city_info VALUES(2014, 'LYO', 0.4);
INSERT INTO city_info VALUES(2015, 'LYO', 0.6);
INSERT INTO city_info VALUES(2016, 'LYO', 0.5);

INSERT INTO city_info VALUES(2014, 'LLL', 0.2);
INSERT INTO city_info VALUES(2015, 'LLL', 0.2);
INSERT INTO city_info VALUES(2016, 'LLL', 0.2);

INSERT INTO city_info VALUES(2014, 'AMN', 0.1);
INSERT INTO city_info VALUES(2015, 'AMN', 0.1);
INSERT INTO city_info VALUES(2016, 'AMN', 0.1);


```

Question 1

On veut connaître la population minimum et la population maximum sur l'ensemble des villes et des années de la base.

Indice :

Écrire une requête SQL sans partitionnement mais qui utilise deux fonctions d'agrégation permettant d'obtenir ces informations.

Indice :

On utilise les fonctions `MIN` et `MAX`.

Question 2

On veut connaître la population minimum et la population maximum pour chaque ville, indépendamment des années.

Indice :

Écrire une requête SQL avec un partitionnement sur `citycode`.

Question 3

Ajouter le nom de la ville au résultat de la requête précédente.

Indice :

Il faut ajouter une jointure avec la table `city`.

VI. Single-Value Rule

Mise en situation

L'utilisation des fonctions d'agrégation n'est pas toujours triviale et comporte des pièges.

Imaginez par exemple une table qui gère les relations familiales. Les enregistrements sont sous la forme suivante : Anna est la cousine de Djamil, c'est aussi la mère de Namata, etc.

Vous vous posez maintenant la question suivante : combien de cousins a Anna ? Et qui sont-ils ? Vous pensez immédiatement à fonction d'agrégation `COUNT` : compter tous les enregistrements qui font mention des cousins d'Anna, et afficher leur nom.

Si cette idée paraît cohérente, il est interdit d'exécuter ce genre de requêtes en SQL.

Fondamental **Principe de la "Single-Value Rule" établi par le standard SQL**

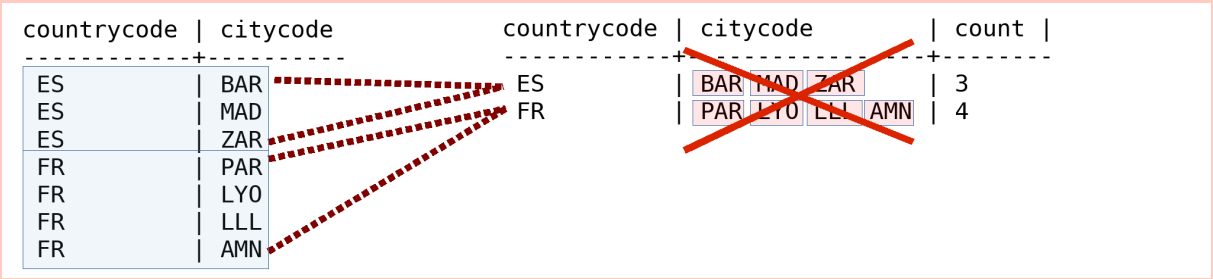
Tout attribut de la clause `SELECT` doit :

- Soit être un attribut d'agrégation (et être également présent dans la clause `GROUP BY`) ;
- Soit être un attribut de calcul (présent dans une fonction d'agrégation).

Attention **Requête illégale**

```
1 SELECT countrycode, citycode, COUNT(citycode)
2 FROM city
3 GROUP BY countrycode;
```

```
1 ERROR:  column "city.citycode" must appear in the GROUP BY clause or be used in an aggregate
  function;
```



Tentative de GROUP BY illégal

La requête est non standard et non logique car on cherche à mettre regrouper plusieurs données de l'attribut `citycode` après le `GROUP BY` (il y a plusieurs `citycode` par `countrycode`).

Elle sera refusée par tous les SGBD.

Néanmoins, il existe des requêtes non standards qui sont tolérées dans certains cas.

Exemple **Requête non standard tolérée par certains SGBD**

```
1 SELECT c.citycode, c.countrycode, AVG(i.population)
2 FROM city c JOIN city_info i
3 ON c.citycode = i.citycode
4 WHERE i.year_ = 2015
5 GROUP BY c.citycode;
```

```
1 | citycode | countrycode | avg                    |
```

```
| -------- | ---------- | ---------------------- |
| AMN      | FR         | 0.10000000000000000000 |
| BAR      | ES         | 2.0000000000000000     |
| LLL      | FR         | 0.20000000000000000000 |
| LYO      | FR         | 0.60000000000000000000 |
| MAD      | ES         | 3.4000000000000000     |
| PAR      | FR         | 2.1000000000000000     |
| ZAR      | ES         | 0.60000000000000000000 |
```

La requête est non standard, néanmoins elle estlogiquement acceptable car `citycode` est une clé et car il n'y a qu'un `countrycode` par `citycode`.

Certains SGBD comme PostgreSQL acceptent donc cette syntaxe, en ajoutant implicitement l'attribut qui manque au `GROUP BY`, car il n'y a pas d'ambiguïté. Mais le standard impose d'être explicite, car le SGBD n'a pas à inférer cela.

Exemple **Requête légale**

```
SELECT ci.countrycode, co.name, COUNT(*)
FROM city ci JOIN country co
ON ci.countrycode=co.countrycode
GROUP BY ci.countrycode, co.name;
```

```
 countrycode |  name   | count
-------------+---------+-------
 FR          | France |     4
 ES          | Spain  |     3
```

Exemple **Requête non standard tolérée par certains SGBD**

```
SELECT co.countrycode, co.name, COUNT(*)
FROM city ci JOIN country co
ON ci.countrycode=co.countrycode
GROUP BY co.countrycode;
```

La requête est non standard, car `co.name` est dans le `SELECT` mais pas dans le `GROUP BY`. Comme dans l'exemple précédent, certains SGBD accepteront cette requête.

Attention **Requête non standard non tolérée**

Si on utilise à présent `ci.countrycode` à la place de `co.countrycode` :

```
SELECT ci.countrycode, co.name, count(*)
FROM city ci JOIN country co
ON ci.countrycode=co.countrycode
GROUP BY ci.countrycode;
```

On obtient une erreur sous PostgreSQL:

```
ERROR:  column "co.name" must appear in the GROUP BY clause or be used in an aggregate
function
```

La requête est non standard, `co.name` doit être mentionné dans la clause `GROUP BY` sous PostgreSQL : le SGBD n'arrive pas ici à faire l'inférence.

Conseil **Être explicite**

Il est donc conseillé d'appliquer la *Single-Value Rule* et de toujours expliciter tous les attributs dans le `GROUP BY`, pour éviter la dépendance au SGBD ou bien les erreurs liées à des variations mineures de syntaxe.

Complément **Éléments de documentation**

Pour mieux découvrir le principe de *Single-Value Rule*, on pourra lire cette section de la documentation de *MariaDB* :https://mariadb.com/kb/en/sql-99/the-single-value-rule[1]

VII. Exercice : Appliquer la notion

Soient les données représentant la population de villes européennes :

```sql
CREATE TABLE country (
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  population NUMERIC(3),
  PRIMARY KEY (countrycode)
);

CREATE TABLE city (
  citycode CHAR(3) NOT NULL,
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (citycode),
  FOREIGN KEY (countrycode) REFERENCES country(countrycode)
);

CREATE TABLE city_info (
  year_ INTEGER,
  citycode CHAR(3),
  population NUMERIC(2,1),
  PRIMARY KEY (year_, citycode),
  FOREIGN KEY (citycode) REFERENCES city(citycode)
);

```

```sql
INSERT INTO country VALUES ('ES', 'Spain', 46);
INSERT INTO country VALUES ('FR', 'France', 67);
INSERT INTO country VALUES ('DE', 'Germany', 82);

INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona');
INSERT INTO city VALUES ('MAD', 'ES', 'Madrid');
INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza');

INSERT INTO city_info VALUES(2014, 'BAR', 0.7);
INSERT INTO city_info VALUES(2015, 'BAR', 2.0);
INSERT INTO city_info VALUES(2016, 'BAR', 1.9);

INSERT INTO city_info VALUES(2014, 'MAD', 3.3);
INSERT INTO city_info VALUES(2015, 'MAD', 3.4);
INSERT INTO city_info VALUES(2016, 'MAD', 3.3);

INSERT INTO city_info VALUES(2014, 'ZAR', 0.7);
```

[1] https://mariadb.com/kb/en/sql-99/the-single-value-rule/

```sql
INSERT INTO city_info VALUES(2015, 'ZAR', 0.6);
INSERT INTO city_info VALUES(2016, 'ZAR', 0.7);

INSERT INTO city VALUES ('PAR', 'FR', 'Paris');
INSERT INTO city VALUES ('LYO', 'FR', 'Lyon');
INSERT INTO city VALUES ('LLL', 'FR', 'Lille');
INSERT INTO city VALUES ('AMN', 'FR', 'Amiens');

INSERT INTO city_info VALUES(2014, 'PAR', 2.0);
INSERT INTO city_info VALUES(2015, 'PAR', 2.1);
INSERT INTO city_info VALUES(2016, 'PAR', 2.2);

INSERT INTO city_info VALUES(2014, 'LYO', 0.4);
INSERT INTO city_info VALUES(2015, 'LYO', 0.6);
INSERT INTO city_info VALUES(2016, 'LYO', 0.5);

INSERT INTO city_info VALUES(2014, 'LLL', 0.2);
INSERT INTO city_info VALUES(2015, 'LLL', 0.2);
INSERT INTO city_info VALUES(2016, 'LLL', 0.2);

INSERT INTO city_info VALUES(2014, 'AMN', 0.1);
INSERT INTO city_info VALUES(2015, 'AMN', 0.1);
INSERT INTO city_info VALUES(2016, 'AMN', 0.1);


```

Question 1

Soit la requête suivante :

```sql
SELECT ci.citycode, ci.name, MAX(cii.population)
FROM city_info cii JOIN city ci
ON cii.citycode=ci.citycode
GROUP BY ci.citycode;
```

Pourquoi cette requête ne respecte-t-elle pas la *Single Value Rule* ?

Question 2

Corriger la requête pour qu'elle s'exécute correctement ;

Question 3

Modifier la requête pour calculer la **moyenne** de la population de chaque ville en fonction des années.

On arrondira à deux chiffres après la virgule.

Indice :

On utilise la fonction de regroupement AVG pour calculer la moyenne.

Indice :

On utilise la fonction mono-ligne `ROUND(nombre, précision)` pour arrondir le résultat.

Question 4

Ajouter le nom du pays au résultat de la requête précédente.

Indice :

Il faut ajouter une jointure avec la table `country`.

VIII. Restriction après agrégation (HAVING)

Mise en situation

Supposez que vous meniez une étude sur la répartition des richesses dans le monde. On vous fournit pour ce faire une table qui enregistre le salaire annuel de chaque habitant, pour chaque pays.

Une question intéressante serait par exemple de comparer le salaire moyen des pays avec le niveau de santé général dans le pays.

Vous vous intéressez plus particulièrement aux pays où le salaire moyen annuel est inférieur à 20.000€, que vous souhaitez afficher.

Or ici, la condition sur les données concerne le résultat d'une agrégation : il faut d'abord effectuer la moyenne, puis filtrer le résultat.

SQL fournit une syntaxe pour filtrer les résultats après agrégation : il s'agit de la clause `HAVING`.

Définition

La clause `HAVING` permet d'effectuer une seconde restriction sur les résultats de l'agrégation selon le résultat d'une fonction d'agrégation appliquée sur les enregistrements de chaque sous-groupe.

Syntaxe

```
SELECT liste d'attributs de partionnement à projeter et de fonctions de calcul
FROM liste de relations
WHERE condition à appliquer avant calcul de l'agrégat
GROUP BY liste ordonnée d'attributs de partitionnement
HAVING condition sur les fonctions de calcul
```

Exemple **Seuillage des populations moyennes calculées**

On calcule la population moyenne des villes pour chaque année ainsi :

```
SELECT year_, AVG(population) AS populationMoyenne
FROM city_info
GROUP BY year_;
```

```
| year_ | populationmoyenne      |
| ----- | ---------------------- |
| 2016  | 1.2714285714285714     |
| 2014  | 1.0571428571428571286 |
| 2015  | 1.2857142857142857     |
```

On peut filtrer ces résultats grâce à la clause `HAVING` avec une condition **sur les valeurs calculées** par AVG.

```
SELECT year_, AVG(population) AS populationMoyenne
FROM city_info
GROUP BY year_
HAVING AVG(population) > 1.2;
```

```
| year_ | populationmoyenne  |
| ----- | ------------------ |
| 2016  | 1.2714285714285714 |
| 2015  | 1.2857142857142857 |
```

Utiliser `HAVING populationmoyenne > 1.2` ne fonctionne pas.

Remarque **Non support standard des alias d'attributs**

On ne peut pas utiliser les alias d'attributs dans la clause `HAVING` : il faut nécessairement les indiquer de nouveau entièrement.

Remarque

`HAVING` ne s'applique pas nécessairement sur les attributs de calculs présents dans la clause `SELECT` : on peut en spécifier d'autres qui n'apparaissent pas dans les résultats de requêtes.

Exemple **Filtrage des populations moyennes calculées**

On peut afficher les résultats de moyennes uniquement pour les sous-groupes ayant une population maximale observée donnée.

```
SELECT year_, AVG(population) AS populationMoyenne
FROM city_info
GROUP BY year_
HAVING MAX(population) > 3.3;
```

```
| year_ | populationmoyenne  |
| ----- | ------------------ |
| 2015  | 1.2857142857142857 |
```

Attention **Différence avec WHERE**

- `WHERE` introduit des restrictions sur des enregistrements individuels avant agrégation.
- `HAVING` introduit des restrictions sur des résultats d'agrégation. Ainsi avec `HAVING` il faut nécessairement avoir utilisé préalablement une fonction d'agrégation.

Exemple **Explication des différences avec WHERE**

Cette requête effectue une restriction avant agrégation :

```
SELECT year_, AVG(population) AS populationMoyenne
FROM city_info
WHERE population > 0.4
GROUP BY year_;
```

On calcule ici les moyennes sur les populations supérieures à 0.4 million d'habitants seulement.

```
| year_ | populationmoyenne  |
| ----- | ------------------ |
| 2016  | 1.7200000000000000 |
| 2014  | 1.6750000000000000 |
| 2015  | 1.7400000000000000 |
```

IX. Exercice : Appliquer la notion

Soient les données représentant la population de villes européennes :

```
CREATE TABLE country (
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  population NUMERIC(3),
  PRIMARY KEY (countrycode)
);

CREATE TABLE city (
  citycode CHAR(3) NOT NULL,
  countrycode CHAR(2) NOT NULL,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (citycode),
```

```
  FOREIGN KEY (countrycode) REFERENCES country(countrycode)
);

CREATE TABLE city_info (
  year_ INTEGER,
  citycode CHAR(3),
  population NUMERIC(2,1),
  PRIMARY KEY (year_, citycode),
  FOREIGN KEY (citycode) REFERENCES city(citycode)
);

```

```
INSERT INTO country VALUES ('ES', 'Spain', 46);
INSERT INTO country VALUES ('FR', 'France', 67);
INSERT INTO country VALUES ('DE', 'Germany', 82);

INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona');
INSERT INTO city VALUES ('MAD', 'ES', 'Madrid');
INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza');

INSERT INTO city_info VALUES(2014, 'BAR', 0.7);
INSERT INTO city_info VALUES(2015, 'BAR', 2.0);
INSERT INTO city_info VALUES(2016, 'BAR', 1.9);

INSERT INTO city_info VALUES(2014, 'MAD', 3.3);
INSERT INTO city_info VALUES(2015, 'MAD', 3.4);
INSERT INTO city_info VALUES(2016, 'MAD', 3.3);

INSERT INTO city_info VALUES(2014, 'ZAR', 0.7);
INSERT INTO city_info VALUES(2015, 'ZAR', 0.6);
INSERT INTO city_info VALUES(2016, 'ZAR', 0.7);

INSERT INTO city VALUES ('PAR', 'FR', 'Paris');
INSERT INTO city VALUES ('LYO', 'FR', 'Lyon');
INSERT INTO city VALUES ('LLL', 'FR', 'Lille');
INSERT INTO city VALUES ('AMN', 'FR', 'Amiens');

INSERT INTO city_info VALUES(2014, 'PAR', 2.0);
INSERT INTO city_info VALUES(2015, 'PAR', 2.1);
INSERT INTO city_info VALUES(2016, 'PAR', 2.2);

INSERT INTO city_info VALUES(2014, 'LYO', 0.4);
INSERT INTO city_info VALUES(2015, 'LYO', 0.6);
INSERT INTO city_info VALUES(2016, 'LYO', 0.5);

INSERT INTO city_info VALUES(2014, 'LLL', 0.2);
INSERT INTO city_info VALUES(2015, 'LLL', 0.2);
INSERT INTO city_info VALUES(2016, 'LLL', 0.2);

INSERT INTO city_info VALUES(2014, 'AMN', 0.1);
INSERT INTO city_info VALUES(2015, 'AMN', 0.1);
INSERT INTO city_info VALUES(2016, 'AMN', 0.1);


```

Question 1

Écrire une requête SQL qui retourne pour les villes ayant **toujours** eu plus de 1 millions d'habitants au cours des années.

Indice :

*« Qui retourne pour les villes ayant **toujours** eu plus de 1 millions d'habitants »* signifie dont la population **minimale** a toujours été supérieur à 1.

Question 2

Écrire une requête SQL qui retourne, pour les villes ayant toujours eu entre 2 et 3 millions d'habitants, leur population moyenne observée au cours des années.

X. Essentiel

Les agrégations sont un outil extrêmement puissant en SQL : elles permettent de synthétiser les informations issues de plusieurs enregistrements.

Les fonctions d'agrégations permettent d'effectuer un calcul sur une ou plusieurs colonnes, comme la somme des dépenses sur un site de vente en ligne.

La clause `GROUP BY` permet de spécifier que les enregistrements vont d'abord être regroupés selon un attribut, par exemple pour calculer le montant moyen des dépenses par client.

Enfin, la clause `HAVING` permet de filtrer le résultat d'une agrégation, par exemple pour connaître les clients dont le montant moyen de commandes n'excède pas 20€.

Attention toutefois, il est interdit de sélectionner des attributs d'une table qui ne sont pas agrégés ou groupés : c'est la *Single Value Rule*.

XI. Auto-évaluation

A. Exercice final

Exercice : Quiz - Culture

Exercice

Une agrégation en SQL est :

- ○ Un type de restriction qui s'exécute à la toute fin d'une requête de sélection.
- ○ Un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'un ou plusieurs attributs de partitionnement, suivi de l'application éventuelle de fonctions de calcul sur d'autres attributs des sous-tables obtenues.
- ○ L'union de jointures externes appelées agrégats, suivis d'une restriction.

Exercice

En SQL, les mots clé utilisés spécifiquement pour réaliser des agrégations sont :

- ☐ `INSERT`
- ☐ `EXPLAIN`
- ☐ `GROUP BY`
- ☐ `ORDER BY`
- ☐ `HAVING`
- ☐ `WHERE`

Exercice

Parmi les fonctions suivantes, lesquelles sont des fonctions d'agrégation ?

- ☐ `CAST`
- ☐ `MEAN`
- ☐ `SUM`
- ☐ `MIN`
- ☐ `COALESCE`
- ☐ `COUNT`

Exercice

Une fonction d'agrégation... :

- ☐ Peut être utilisée dans la clause `SELECT`.
- ☐ Peut être utilisée dans la clause `WHERE`.
- ☐ Peut être utilisée dans la clause `GROUP BY`.
- ☐ Peut être utilisée dans la clause `ORDER BY`.
- ☐ S'applique sur plusieurs valeurs d'un attribut.
- ☐ S'applique sur une seule valeur d'un attribut.

Exercice : Quiz - Code

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
3 GROUP BY year_;
```

- ○ Elle retourne pour chaque année la population moyenne des villes.
- ○ Elle retourne pour chaque ville la population moyenne sur les années concernées.

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
3 WHERE population > 1.2
4 GROUP BY year_;
```

- ○ Elle retourne pour chaque année la population moyenne des villes, pour celles dont la population excède 1.2 millions d'habitants.
- ○ Elle retourne pour chaque année la population moyenne des villes à condition que cette moyenne excède 1.2 millions d'habitants.

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
3 GROUP BY year_
4 HAVING AVG(population) > 1.2;
```

```
5
```

- Elle retourne pour chaque année la population moyenne des villes, pour celles dont la population excède 1.2 millions d'habitants.
- Elle retourne pour chaque année la population moyenne des villes à condition que cette moyenne excède 1.2 millions d'habitants.

Exercice

Quelle valeur renvoie la dernière instruction SQL de la liste ci-dessous ?

```sql
CREATE TABLE t (
  a INTEGER,
  b INTEGER,
  c INTEGER
);

INSERT INTO t VALUES (0, 0, 0);
INSERT INTO t VALUES (1, 0, 0);
SELECT SUM(a) + COUNT(b) FROM t;
```

Exercice

Quelles sont les requêtes qui respectent la *Single Value Rule* ?

- ☐
```sql
SELECT classe, annee, COUNT(eleve)
FROM inscription
GROUP by classe, annee;
```
- ☐
```sql
SELECT destinataire, mois, MAX(montant)
FROM virement
GROUP BY destinataire, mois;
```
- ☐
```sql
SELECT ci.countrycode, COUNT(*)
FROM city ci JOIN country co
ON ci.countrycode=co.countrycode
GROUP BY ci.countrycode, co.name;
```
- ☐
```sql
SELECT medicament, dosage, MAX(valeur)
FROM composition
GROUP BY medicament
```

B. Exercice : Défi

Soit le schéma relationnel suivant mis en place pour stocker les informations de ventes sur un marché :

```
Representant (#nr: integer, nomr:varchar, ville: varchar)
Produit (#np: integer, nomp: varchar, coul: varchar, poids: varchar)
Client (#nc: integer, nomc: varchar, ville: varchar)
Vente (#nr=>Representant(nr), #np=>Produit(np), #nc=>Client(nc), quantite: real)
```

```sql
CREATE TABLE representant (
  nr INTEGER PRIMARY KEY,
  nomr VARCHAR(20),
  ville VARCHAR(20)
);

CREATE TABLE produit (
```

```
  np INTEGER PRIMARY KEY,
  nomp VARCHAR(20),
  couleur VARCHAR(20),
  poids INTEGER
);

CREATE TABLE client (
  nc INTEGER PRIMARY KEY,
  nomc VARCHAR(20),
  ville VARCHAR(20)
);

CREATE TABLE vente (
  nr INTEGER REFERENCES representant(nr),
  np INTEGER REFERENCES produit(np),
  nc INTEGER REFERENCES client(nc),
  quantite INTEGER,
  PRIMARY KEY (nr, np, nc)
);

INSERT INTO representant VALUES (1, 'Stephane', 'Lyon');
INSERT INTO representant VALUES (2, 'Benjamin', 'Paris');
INSERT INTO representant VALUES (3, 'Leonard', 'Lyon');
INSERT INTO representant VALUES (4, 'Antoine', 'Brest');
INSERT INTO representant VALUES (5, 'Bruno', 'Bayonne');

INSERT INTO produit VALUES (1, 'Aspirateur', 'Rouge', 3546);
INSERT INTO produit VALUES (2, 'Trottinette', 'Bleu', 1423);
INSERT INTO produit VALUES (3, 'Chaise', 'Blanc', 3827);
INSERT INTO produit VALUES (4, 'Tapis', 'Rouge', 1423);

INSERT INTO client VALUES (1, 'Alice', 'Lyon');
INSERT INTO client VALUES (2, 'Bruno', 'Lyon');
INSERT INTO client VALUES (3, 'Charles', 'Compiègne');
INSERT INTO client VALUES (4, 'Denis', 'Montpellier');
INSERT INTO client VALUES (5, 'Emile', 'Strasbourg');

INSERT INTO vente VALUES (1, 1, 1, 1);
INSERT INTO vente VALUES (1, 1, 2, 1);
INSERT INTO vente VALUES (2, 2, 3, 1);
INSERT INTO vente VALUES (4, 3, 3, 200);
INSERT INTO vente VALUES (3, 4, 2, 190);
INSERT INTO vente VALUES (1, 3, 2, 300);
INSERT INTO vente VALUES (3, 1, 2, 120);
INSERT INTO vente VALUES (3, 1, 4, 120);
INSERT INTO vente VALUES (3, 4, 4, 2);
INSERT INTO vente VALUES (3, 1, 1, 3);
INSERT INTO vente VALUES (3, 4, 1, 5);
INSERT INTO vente VALUES (3, 1, 3, 1);
INSERT INTO vente VALUES (3, 1, 5, 1);
```

Question 1

Écrire en SQL une requête permettant d'obtenir le nombre de clients.

Question 2

Écrire en SQL une requête permettant d'obtenir le nombre de produits de couleur rouge.

Question 3

Écrire en SQL une requête permettant d'obtenir le nombre de clients par ville.

Question 4

Écrire en SQL une requête permettant d'obtenir la quantité totale des produits rouges vendus par chaque représentant.

Question 5

Écrire en SQL une requête permettant d'obtenir la quantité totale de produits rouges vendus pour chaque représentant ayant vendu plus de 5 fois des produits rouges, c'est-à-dire ayant réalisé plus de 5 ventes différentes de produits rouges.

Solutions des exercices

Exercice p. Solution n°1

```
SELECT COUNT(*)
FROM city;
```

```
| COUNT(*) |
| -------- |
| 7        |
```

Exercice p. Solution n°2

```
SELECT countrycode, COUNT(*)
FROM city
GROUP BY countrycode;
```

```
 countrycode | count
-------------+-------
 ES          |     3
 FR          |     4
```

Exercice p. Solution n°3

```
SELECT MIN(population) AS "Population Minimum", MAX(population) AS "Population Maximum"
FROM city_info;
```

```
| Population Minimum | Population Maximum |
| ------------------ | ------------------ |
| 0.1                | 3.4                |
```

Exercice p. Solution n°4

```
SELECT citycode, MIN(population) AS "Population Minimum", MAX(population) AS "Population
Maximum"
FROM city_info
GROUP BY citycode;
```

```
 citycode | Population Minimum | Population Maximum
----------+--------------------+--------------------
 ZAR      |                0.6 |                0.7
 LLL      |                0.2 |                0.2
 PAR      |                2.0 |                2.2
 BAR      |                0.7 |                2.0
 AMN      |                0.1 |                0.1
 MAD      |                3.3 |                3.4
 LYO      |                0.4 |                0.6
```

Exercice p. Solution n°5

```
SELECT ci.citycode, c.name, MIN(ci.population) AS "Population Minimum", MAX(ci.population) AS
"Population Maximum"
FROM city_info ci JOIN city c
ON ci.citycode = c.citycode
GROUP BY ci.citycode, c.name;
```

```
 citycode |    name    | Population Minimum | Population Maximum
```

```
----------+-----------+--------------------+--------------------
 MAD      | Madrid    |                3.3 |                3.4
 AMN      | Amiens    |                0.1 |                0.1
 ZAR      | Zaragoza  |                0.6 |                0.7
 BAR      | Barcelona |                0.7 |                2.0
 LYO      | Lyon      |                0.4 |                0.6
 PAR      | Paris     |                2.0 |                2.2
 LLL      | Lille     |                0.2 |                0.2
```

Exercice p. Solution n°6

Elle ne respecte pas la `single-rule value` car `ci.name` n'est ni défini comme un attribut de regroupement, ni comme une fonction de regroupement.

Exercice p. Solution n°7

```sql
SELECT ci.citycode, ci.name, MAX(cii.population)
FROM city_info cii JOIN city ci
ON cii.citycode=ci.citycode
GROUP BY ci.citycode, ci.name;
```

```
 citycode |   name    | max
----------+-----------+-----
 ZAR      | Zaragoza  | 0.7
 PAR      | Paris     | 2.2
 BAR      | Barcelona | 2.0
 MAD      | Madrid    | 3.4
 LYO      | Lyon      | 0.6
 LLL      | Lille     | 0.2
 AMN      | Amiens    | 0.1
```

Exercice p. Solution n°8

```sql
SELECT ci.citycode, ci.name, ROUND(AVG(cii.population),2) as avg
FROM city_info cii JOIN city ci
ON cii.citycode=ci.citycode
GROUP BY ci.citycode, ci.name;
```

On obtient :

```
 citycode |   name    | avg
----------+-----------+------
 ZAR      | Zaragoza  | 0.67
 PAR      | Paris     | 2.10
 BAR      | Barcelona | 1.53
 MAD      | Madrid    | 3.33
 LYO      | Lyon      | 0.50
 LLL      | Lille     | 0.20
 AMN      | Amiens    | 0.10
```

Exercice p. Solution n°9

```sql
SELECT ci.citycode, ci.name, co.name, ROUND(AVG(cii.population),2) AS avg
FROM city_info cii
JOIN city ci ON cii.citycode=ci.citycode
JOIN country co ON ci.countrycode=co.countrycode
GROUP BY ci.citycode, ci.name, co.name;
```

On obtient :

```
 citycode |   name    |  name  | avg
----------+-----------+--------+------
 LLL      | Lille     | France | 0.20
 ZAR      | Zaragoza  | Spain  | 0.67
 LYO      | Lyon      | France | 0.50
 AMN      | Amiens    | France | 0.10
 PAR      | Paris     | France | 2.10
 BAR      | Barcelona | Spain  | 1.53
 MAD      | Madrid    | Spain  | 3.33

```

Exercice p. Solution n°10

```sql
SELECT citycode, MIN(population) AS "Population Minimum"
FROM city_info
GROUP BY citycode
HAVING MIN(population) > 1;
```

On obtient :

```

| citycode | Population Minimum |
| -------- | ------------------ |
| PAR      | 2.0                |
| MAD      | 3.3                |
```

Exercice p. Solution n°11

On spécifie les conditions avec les fonctions `MIN` et `MAX`.

```sql
SELECT citycode, AVG(population) AS "Population Moyenne"
FROM city_info
GROUP BY citycode
HAVING MIN(population) >= 2 AND MAX(population) <= 3;
```

On obtient :

```
| citycode | Population Moyenne |
| -------- | ------------------ |
| PAR      | 2.1000000000000000 |
```

Exercice p. 17 Solution n°12

Exercice

Une agrégation en SQL est :

- ○ Un type de restriction qui s'exécute à la toute fin d'une requête de sélection.
- ◉ Un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'un ou plusieurs attributs de partitionnement, suivi de l'application éventuelle de fonctions de calcul sur d'autres attributs des sous-tables obtenues.
- ○ L'union de jointures externes appelées agrégats, suivis d'une restriction.

Exercice

En SQL, les mots clé utilisés spécifiquement pour réaliser des agrégations sont :

- ☐ `INSERT`
- ☐ `EXPLAIN`
- ☑ `GROUP BY`
- ☐ `ORDER BY`
- ☑ `HAVING`
- ☐ `WHERE`

Exercice

Parmi les fonctions suivantes, lesquelles sont des fonctions d'agrégation ?

- ☐ `CAST`
- ☐ `MEAN`
- ☑ `SUM`
- ☑ `MIN`
- ☐ `COALESCE`
- ☑ `COUNT`

Exercice

Une fonction d'agrégation... :

- ☑ Peut être utilisée dans la clause `SELECT`.
- ☐ Peut être utilisée dans la clause `WHERE`.
- ☐ Peut être utilisée dans la clause `GROUP BY`.
- ☑ Peut être utilisée dans la clause `ORDER BY`.
- ☑ S'applique sur plusieurs valeurs d'un attribut.
- ☐ S'applique sur une seule valeur d'un attribut.

Exercice p. 18 Solution n°13

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
```

```
3 GROUP BY year_;
```

- ⦿ Elle retourne pour chaque année la population moyenne des villes.
- ○ Elle retourne pour chaque ville la population moyenne sur les années concernées.

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
3 WHERE population > 1.2
4 GROUP BY year_;
```

- ⦿ Elle retourne pour chaque année la population moyenne des villes, pour celles dont la population excède 1.2 millions d'habitants.
- ○ Elle retourne pour chaque année la population moyenne des villes à condition que cette moyenne excède 1.2 millions d'habitants.

La clause `WHERE` s'applique avant ce groupement sur les valeurs individuelles de populations : les enregistrement retenus sont ceux dont la population est supérieure à 1.2.

On groupe ensuite sur les années et on calcule enfin, pour chaque année, la moyenne des populations sur les enregistrement retenus.

Exercice

Que retourne la requête suivante ?

```
1 SELECT year_, AVG(population)
2 FROM city_info
3 GROUP BY year_
4 HAVING AVG(population) > 1.2;
5
```

- ○ Elle retourne pour chaque année la population moyenne des villes, pour celles dont la population excède 1.2 millions d'habitants.
- ⦿ Elle retourne pour chaque année la population moyenne des villes à condition que cette moyenne excède 1.2 millions d'habitants.

La clause `HAVING` s'applique après le groupement sur la valeur calculée. Seules les années pour lesquelles la moyenne est supérieure à 1.2 sont retenues.

Exercice

Quelle valeur renvoie la dernière instruction SQL de la liste ci-dessous ?

```
1 CREATE TABLE t (
2   a INTEGER,
3   b INTEGER,
4   c INTEGER
5 );
6
7 INSERT INTO t VALUES (0, 0, 0);
8 INSERT INTO t VALUES (1, 0, 0);
9 SELECT SUM(a) + COUNT(b) FROM t;
```

3

Contenus de la table :

- Après Create : Ø
- Après Insert n°1 :

0	0	0

- Après Insert n°2 :

0	0	0
1	0	0

On a `SUM(a) = 1` et `COUNT(b) = 2` donc La requête renvoie **3**.

Exercice

Quelles sont les requêtes qui respectent la *Single Value Rule* ?

- [x]
```
SELECT classe, annee, COUNT(eleve)
FROM inscription
GROUP by classe, annee;
```

- [x]
```
SELECT destinataire, mois, MAX(montant)
FROM virement
GROUP BY destinataire, mois;
```

- [x]
```
SELECT ci.countrycode, COUNT(*)
FROM city ci JOIN country co
ON ci.countrycode=co.countrycode
GROUP BY ci.countrycode, co.name;
```

- [ ]
```
SELECT medicament, dosage, MAX(valeur)
FROM composition
GROUP BY medicament
```

Exercice p. Solution n°14

```
SELECT COUNT(*)
FROM client ;
```

```
| COUNT(*)  |
| --------- |
| 5         |
```

On aurait aussi pu utiliser :

```
SELECT COUNT(nc)
FROM client ;
```

Exercice p. Solution n°15

On utilise :

```
SELECT COUNT(*)
FROM produit p
WHERE p.couleur ='Rouge' ;
```

On obtient :

```
| COUNT(*) |
| -------- |
| 2        |
```

Exercice p. Solution n°16

On utilise :

```
SELECT ville, count(*)
FROM client
GROUP BY ville ;
```

On obtient :

```
| ville       | count(*) |
| ----------- | -------- |
| Compiègne   | 1        |
| Lyon        | 2        |
| Montpellier | 1        |
| Strasbourg  | 1        |
```

Exercice p. Solution n°17

On utilise :

```
SELECT v.nr, SUM(v.quantite)
FROM vente v, produit p
WHERE v.np = p.np
AND p.couleur = 'Rouge'
GROUP BY v.nr ;
```

On obtient :

```
| nr  | SUM(v.quantite) |
| --- | --------------- |
| 1   | 2               |
| 3   | 442             |
```

Exercice p. Solution n°18

On utilise :

```
SELECT v.nr, SUM(v.quantite)
FROM vente v, produit p
WHERE v.np = p.np
AND p.couleur = 'Rouge'
GROUP BY v.nr
HAVING COUNT(*) > 5 ;
```

On obtient :

```
| nr  | SUM(v.quantite) |
```

```
| --- | ---------- |
| 3   | 442        |
```