

- [L3 MIAHS/Ingémath](#)
- [Université Paris Cité](#)
- Année 2023-2024
- [Course Homepage](#)
  
- [Moodle](#)



! Sur le serveur, dans votre schéma personnel (celui dont le nom est votre `username`), créer les fonctions SQL et vues correspondant aux cinq questions suivantes.

i Certaines questions portent sur le schéma `babynames` qui ne contient qu'une seule table `bebes`. Le schéma de `bebes` est le suivant :

Table "babynames.bebes"			
Column	Type	Nullable	
sex	integer	not null	
name	character varying(500)	not null	
year	integer	not null	
count	integer		

Indexes:

"bebes\_pk" PRIMARY KEY, btree (sex, name, year)

- `sex` représente le sexe des bébés avec le codage suivant :
  - 1 pour un garçon
  - 2 pour une fille
- `name` représente un prénom donné. C'est une chaîne de caractères de longueur variable mais inférieure à 500.
- `year` représente l'année, c'est de type entier.
- `count` représente le nombre de bébés de sexe `sex` nés pendant l'année `year` auxquels ont été attribués le prénom `name`.

👉 Une ligne de `bebes` est identifiée par les attributs `year`, `sex`, `name`.  
D'autres questions portent sur le [schéma world](#).

⚠ Pour construire les corrections, nous avons développés nos solutions à la main.  
Mais nous avons aussi transmis le schéma de `babynames` à un chatbot (`chatgpt 3.5`) et nous lui avons demandé de construire les requêtes correspondant à nos questions posées en langage naturelle.

## Popularité

### i Question

Écrire une fonction SQL nommée `top_k_names_by_year` qui prend en argument une année `yyyy` et un entier `n` et renvoie une table (`name TEXT`, `sex integer`, `rank integer`) qui contient, selon la table `babynames.bebes`, les `k` prénoms les plus populaires (attribués) de l'année `yyyy` pour chaque sexe.

💡 Réponse attendue pour `SELECT * FROM top_k_names_by_year(1950, 5); :`

```
+-----+-----+-----+
|F|MARIE|1|
|F|MARTINE|2|
|F|FRANÇOISE|3|
|F|MONIQUE|4|
|F|NICOLE|5|
|H|JEAN|1|
|H|MICHEL|2|
|H|ALAIN|3|
|H|GÉRARD|4|
|H|BERNARD|5|
```

### 💡 Solution

Première question posée au chatbot :

For every year and sex, find the 5 most popular names.

Réponse :

To find the 5 most popular names for each year and sex in the `babynames.bebes` table, you can use ``ROW_NUMBER()``. Here's a query that retrieves the top 5 names for each combination of year and sex:

```
WITH RankedNames AS (
    SELECT
        name, year, sex, count,
        ROW_NUMBER() OVER (PARTITION BY year, sex ORDER BY count DESC) AS rank
    FROM
        babynames.bebes
)

SELECT
    name, year, sex, count
FROM
    RankedNames
WHERE
    rank <= 5
ORDER BY
    year, sex, rank;
```

Explanation:

- The ``ROW_NUMBER()`` function ranks names within each ``year`` and ``sex`` partition, ordered by the ``count`` of babies in descending order.
- The ``PARTITION BY year, sex`` groups the data by each year and sex.
- The ``ORDER BY count DESC`` ensures that the most popular names (with the highest counts) are ranked first.
- In the ``WHERE`` clause, we filter for the top 5 names (``rank <= 5``) for each year and sex.

This query will return the most popular 5 names for both males and females for each year in the dataset.

### 💡 Solution (suite)

Pour définir la fonction, on peut demander de l'aide au chatbot :

Design a sql function that takes as input a year y and an integer k, and returns a table with columns name, sex and rank containing the k most popular names in year y for each sex

La réponse est

```
CREATE OR REPLACE FUNCTION top_k_names_by_year(y INT, k INT)
RETURNS TABLE(name VARCHAR(500), sex INT, rank INT) AS $$
BEGIN
    RETURN QUERY
    WITH RankedNames AS (
        SELECT
            name, sex, count,
            ROW_NUMBER() OVER (PARTITION BY sex ORDER BY count DESC) AS rank
        FROM
            babynames.bebes
        WHERE
            year = y
    )
    SELECT
        name, sex, rank
    FROM
        RankedNames
    WHERE
        rank <= k
    ORDER BY
        sex, rank;
END;
$$ LANGUAGE plpgsql;
```

### 💡 Solution (suite)

En insistant pour obtenir une fonction SQL on obtient :

Sure! If you'd prefer a **pure SQL function** (without using PL/pgSQL), here's how you can implement it. In PostgreSQL, you can create a function using SQL that performs the required query:

```
CREATE OR REPLACE FUNCTION top_k_names_by_year(y INT, k INT)
RETURNS TABLE(name VARCHAR(500), sex INT, rank INT) AS $$
    WITH RankedNames AS (
        SELECT
            name, sex, count,
            ROW_NUMBER() OVER (PARTITION BY sex ORDER BY count DESC) AS rank
        FROM
            babynames.bebes
        WHERE
            year = y
    )
    SELECT
        name, sex, rank
    FROM
        RankedNames
    WHERE
        rank <= k
    ORDER BY
        sex, rank;
$$ LANGUAGE sql;
```

### 💡 Solution (suite)

Explanation:

- **Function Definition:**
  - The function `top_k_names_by_year` accepts two parameters: `y` (year) and `k` (number of top names).
  - It returns a table with three columns:
    - `name` (name of the baby),
    - `sex` (gender: 1 for male, 2 for female), and
    - `rank` (the rank of the name by popularity).
- **WITH Clause:**
  - The `WITH RankedNames AS` clause ranks names by the count of babies for each sex in the specified year (`y`) using the `ROW_NUMBER()` window function. It partitions by `sex` and orders by `count DESC` to assign ranks starting from the most popular.
- **WHERE Clause:**
  - The main query filters for names with a rank less than or equal to `k` to get only the top `k` names for each sex.
- **Language:**
  - The function is written in **pure SQL** (as specified by `LANGUAGE sql`).

This SQL function will return the top `k` names for each sex in the specified year (`y`).

## Nombre de naissances/*sex ratio*

Données dans le schéma `babynames`.

### Question

Écrire une fonction SQL nommée `sex_ratio` qui prend en argument une année `yyyy` et renvoie le rapport entre le nombre total de naissances féminines et le nombre total de naissances masculines cette année là selon la table `babynames.bebes`.

### Réponse attendue pour `SELECT * FROM sex_ratio(1922);`

```
+-----+  
| sex_ratio |  
+-----+  
|    1.05    |
```

NB : précision demandée : 0.1

### Solution

```
WITH R AS (  
    SELECT year, sex, SUM("count") AS n  
    FROM babes  
    GROUP BY year, sex  
)  
  
SELECT r1."year",  
       (r1.n::float/r2.n::float)::decimal(3,2) AS sex_ratio  
FROM R r1 JOIN R r2 ON (  
    r1."year"=r2."year" AND  
    r1.sex < r2.sex  
) ;
```

Pour calculer une ligne du résultat final, il faut combiner deux lignes concernant la même année mais deux sexes différents de la table intermédiaire R. La jointure réalise en passant une opération de *pivot*.

### 💡 Solution (suite)

```
create or replace function sex_ratio(p_y INT)
  returns DECIMAL(3,2) as
$$
with R as (
  select
    b.sex,
    SUM(b."count") as n
  from
    babynames.bebes as b
  where
    b."year" = p_y
  group by
    b.sex
)

select
  (r1.n::float / r2.n::float)::decimal(3,1) as sex_ratio
from
  R r1
join R r2 on
  (r1.sex < r2.sex) ;
$$ language sql;
```

### 💡 Solution (suite)

Question posée au chatbot :

Can you design a pure SQL function that takes as input a year y and returns the sex ratio for that year ?

Réponse :

Yes, I can design a pure SQL function to calculate the sex ratio for a given year in the babynames.bebes table.  
The sex ratio is typically defined as the number of males (sex = 1) per 100 females (sex = 2).

### 💡 Solution (suite)

```
CREATE OR REPLACE FUNCTION get_sex_ratio(y INT)
RETURNS DECIMAL(3,2) AS $$
  SELECT
    CASE
      WHEN SUM(CASE WHEN sex = 2 THEN count ELSE 0 END) = 0 THEN NULL
      ELSE
        ((SUM(CASE WHEN sex = 1 THEN count ELSE 0 END))::float /
         SUM(CASE WHEN sex = 2 THEN count ELSE 0 END)::float)::decimal(3,2)
    END AS sex_ratio
  FROM
    babynames.bebes
  WHERE
    year = y;
$$ LANGUAGE sql;
```

Cette solution évite un GROUP BY et une JOINTURE. L'utilisation de CASE WHEN ... THEN ... ELSE ... permet de calculer les deux agrégations et de les combiner en une seule passe sur la table filtrée.

## Popularité relative et rang de popularité

Données dans le schéma `babynames`

### Question

Écrire une vue nommée `popularity` de schema  
(`sex text, name text, year integer, rank integer, popularity numeric`)  
qui donne pour chaque prénom, année, sexe, la popularité relative (nombre sur nombre total  
de naissance de ce sexe cette année là) et le rang par popularité décroissante.

### Réponse attendue pour

```
SELECT *  
FROM popularity  
WHERE name='THÉO' AND  
       sex = 1 AND  
       "year" >1999 AND  
       "year" < 2005;
```

sex	name	year	rank	popularity
1	THÉO	2000	6	0.019
1	THÉO	2001	6	0.022
1	THÉO	2002	5	0.022
1	THÉO	2003	5	0.020
1	THÉO	2004	7	0.016

NB : précision demandée pour la popularité 0.01

### 💡 solution

Question posée au chatbot :

Can you define a view with schema  
(sex text, name text, year integer, rank integer, popularity numeric)  
where popularity is the proportion of babies with given sex  
who were named by name during year and rank is the rank of the name  
when ordering by decreasing popularity.

Réponse du chatbot :

```
CREATE OR REPLACE VIEW name_popularity_view AS
WITH TotalBySexYear AS (
    -- Calculate the total number of babies by sex and year
    SELECT
        sex,
        year,
        SUM(count) AS total_babies
    FROM
        babynames.bebes
    GROUP BY
        sex, year
),
RankedNames AS (
    -- Calculate the popularity and rank the names by decreasing popularity
    SELECT
        CASE
            WHEN b.sex = 1 THEN 'Male'
            WHEN b.sex = 2 THEN 'Female'
        END AS sex,
        b.name,
        b.year,
        (b.count::numeric / t.total_babies) AS popularity,
        ROW_NUMBER() OVER (PARTITION BY b.sex, b.year
                           ORDER BY b.count DESC) AS rank
    FROM
        babynames.bebes b
    JOIN
        TotalBySexYear t ON b.sex = t.sex AND b.year = t.year
)
-- Final selection for the view
SELECT
    sex,
    name,
    year,
    rank,
    popularity
FROM
    RankedNames;
```

## Les riches et les pauvres dans world

Données dans le [schéma world](#)



### Question

Écrire une vue nommée `ricchi_e_poveri` de schéma :

```
(region TEXT,  
 min_gnp_cap NUMERIC, poorest_country_name TEXT,  
 max_gnp_per_cap NUMERIC, richest_country_name TEXT)
```

où chaque ligne indique pour une région donnée, le minimum du PIB (`gnp`) par habitant, un des pays de la région où ce minimum est réalisé, le maximum du PIB par habitant et un des pays où ce maximum est réalisé.



```
SELECT region, richest_country, poorest_country  
FROM ricchi_e_poveri  
WHERE region='Eastern Asia' ;
```

+	+	+	+
region	richest_country	poorest_country	
+	+	+	+
Eastern Asia	Japan	North Korea	



### Solution

Question posée au chatbot :

```
In the world schema, can you build a view ricchi_e_poveri  
with schema (region TEXT,  
 min_gnp_cap NUMERIC, poorest_country_name TEXT,  
 max_gnp_per_cap NUMERIC, richest_country_name TEXT)  
which, for each region, contains the minimum gnp per capita,  
 the name of the country in the region  
 with the minimum gnp per capita (poorest country),  
 the maximum gnp per capita in the region,  
 the name of the country with maximum gnp per capita  
 in the region (richest country).
```

Countries with zero or no population should not be considered.

### 💡 Solution (suite)

```
CREATE OR REPLACE VIEW world.ricchi_e_poveri AS
WITH GnpPerCapita AS (
  -- Calculate GNP per capita for countries with valid population and GNP
  SELECT
    region,
    name_country,
    gnp / population_country AS gnp_per_capita
  FROM
    world.country
  WHERE
    population_country > 0 -- Exclude countries with zero or no population
    AND gnp IS NOT NULL -- Exclude countries with no GNP data
)
SELECT
  region,
  MIN(gnp_per_capita) AS min_gnp_cap, -- Minimum GNP per capita for the region
  (SELECT name_country FROM GnpPerCapita g2
   WHERE g2.region = g1.region
   ORDER BY gnp_per_capita ASC LIMIT 1) AS poorest_country_name, -- Country with minimum GNP
  MAX(gnp_per_capita) AS max_gnp_per_cap, -- Maximum GNP per capita for the region
  (SELECT name_country FROM GnpPerCapita g2
   WHERE g2.region = g1.region
   ORDER BY gnp_per_capita DESC LIMIT 1) AS richest_country_name -- Country with maximum GNP
FROM
  GnpPerCapita g1
GROUP BY
  region;
```

## Fenêtre glissante sur les bébés

Données dans le schéma `babynames`

### i Question

Créer une vue `sliding_avg_count` de schéma (`name` TEXT, `sex` TEXT, `year` INTEGER, `avg_count` NUMERIC) qui, pour chaque prénom (`name`), sexe (`sex`) et année (`year`), donne le nombre moyen (average count, `avg_count`) d'attributions du prénom aux enfants du sexe en question durant l'intervalle formé par les deux années qui précèdent et les deux années qui suivent l'année courante (incluses).

`avg_count` est un exemple de *moyenne mobile* ou *glissante*.



```
SELECT *
FROM sliding_avg_count
WHERE "name" = 'PATRICK' AND
      "year" BETWEEN 1955 AND 1960 ;
```

name	sex	year	avg_count
PATRICK	F	1959	3.00
PATRICK	H	1955	20381.00
PATRICK	H	1956	20673.80
PATRICK	H	1957	20532.20
PATRICK	H	1958	20173.80
PATRICK	H	1959	19443.20
PATRICK	H	1960	18479.60

La manière la plus simple de répondre à cette question 10 est d'utiliser une *fenêtre glissante*. Les fenêtres glissantes étendent les fenêtres définies par `PARTITION BY ... ORDER BY ...`.

La syntaxe de l'invocation des fonctions fenêtres est décrite dans la documentation [PostgreSQL](#).

L'invocation d'une fonction opérant sur une fenêtre suit la forme :

```
function_name ([expression [, expression ... ]])
    [ FILTER ( WHERE filter_clause ) ]
    OVER ( window_definition )
```

-- OU

```
function_name ( * )
    [ FILTER ( WHERE filter_clause ) ]
    OVER ( window_definition )
```

Nous n'avons pas eu besoin d'utiliser la clause `FILTER (WHERE filter_clause)`

La définition de la fenêtre `window definition` est de la forme

```
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

Rappel : une expression entre `[]` est optionnelle.

La dernière clause optionnelle `frame_clause` permet de calculer des fenêtres glissantes et d'autres types de fenêtres. Elle peut se décliner en

```
{ RANGE | ROWS | GROUPS } frame_start [ frame_exclusion ]
{ RANGE | ROWS | GROUPS } BETWEEN frame_start AND frame_end [ frame_exclusion ]
```

où `frame_start` (début du cadre/de la fenêtre) peut prendre les formes suivantes :

```
UNBOUNDED PRECEDING -- ou
offset PRECEDING -- ou
CURRENT ROW --ou
offset FOLLOWING -- ou
UNBOUNDED FOLLOWING -- ou
```

`frame_end` se définit de la même manière.

Si on veut calculer le nombre cumulé d'attributions d'un prénom depuis l'origine des temps on pourra procéder ainsi :

```
WITH R AS (
SELECT "name", sex, "year", SUM("count")
      OVER (PARTITION BY "name" , sex
```

```
ORDER BY "year"
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
as cumule
FROM babynames.bebes
)

SELECT * FROM R
WHERE "name" IN ('PATRICK', 'MICHEL') ; -- pour limiter la taille du résultat
```

### 💡 Solution

Question posée au chatbot

In schema babynames, design a view with schema  
(name TEXT, sex TEXT, year INTEGER, avg\_count NUMERIC)  
where `avg\_count` represents the moving average of `count` for the given `name` and `sex` when

### 💡 Solution (suite)

Réponse du chatbot :

To design the view with schema  
(name TEXT, sex TEXT, year INTEGER, avg\_count NUMERIC)  
that computes a moving average of `count` for the given `name` and `sex`, including the 2 year

Steps:

Define the window: The average should be computed over the 2 years before, the current year, and the 2 years after.  
Partition by name and sex: This ensures that the moving average is calculated independently for each name and sex.  
Apply a windowing function: Use AVG() with a window frame of 2 rows preceding and 2 rows following.

### 💡 Solution (suite)

```
CREATE OR REPLACE VIEW name_avg_count AS
SELECT
    name,
    sex,
    year,
    AVG(count) OVER (
        PARTITION BY name, sex
        ORDER BY year
        ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING
    ) AS avg_count
FROM
    babynames;
```