Avec solutions

- **L3 MIASHS/Ingémath**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

Fonctions fenêtres/Window functions

Une *fonction fenêtre* effectue un calcul sur un ensemble de lignes de table qui sont liées d'une manière ou d'une autre à la ligne actuelle. Ceci est comparable au type de calcul qui peut être effectué avec une *fonction d'agrégation*. Toutefois, les *fonctions de fenêtre* n'entraînent pas le regroupement des lignes en une unique ligne de sortie, comme le feraient des appels d'agrégats sans fenêtre. Au contraire, avec les *fonctions fenêtres* les lignes conservent leurs identités distinctes.

Pour chaque ligne du résultat, les fonctions fenêtres calculent sur un ensemble de lignes. Cet ensemble de lignes est défini par l'adverbe `OVER`.

OVER (PARTITION BY ...)

Cette construction peut être combinée avec n'importe quelle fonction d'aggrégation.

Cette requête indique pour chaque ville, la "population moyenne" des villes de ce pays (cette moyenne n'a aucun sens).

```
SELECT id, name_city, population_city,
       avg(population_city) OVER (PARTITION BY countrycode)  AS avg_pop
FROM world.city;
```

Sans le mécanisme de fenêtrage, on pourrait obtenir le résultat au prix d'une jointure

```
 WITH tmp AS (
  SELECT c.countrycode, AVG(c.population_city) AS avg_pop
  FROM world.city c
  GROUP BY c.countrycode
 )
  SELECT cc.id, cc.name_city, cc.population_city, tmp.avg_pop
  FROM world.city cc NATURAL JOIN tmp;
```

OVER (PARTITION BY ... ORDER BY)

On peut partitioner et trier.

Cette construction est très pratique pour ranger les tuples d'un sous-groupe.

Dans `world`, si on veut ranger les langues parlées dans un pays par popularité décroissante, on peut procéder ainsi.

```
SELECT countrycode, LANGUAGE,
RANK() OVER (PARTITION BY countrycode ORDER BY percentage DESC)  AS rnk
FROM world.countrylanguage ;
```

RANK() OVER ()

On peut ne pas partitionner en utilisant l'expression `OVER ()`. Par exemple, si on veut obtenir le rang des tuples d'une table sur un tri particulier.

```
SELECT countrycode, name_country,
       RANK() OVER (ORDER BY population_country DESC)  AS rnk
FROM world.country ;
```

Autres types de fenêtres

La construction `OVER ( ...)` n'est pas utilisée exclusivement avec des partitions (peut-être triviales), on peut aussi définir des fenêtres glissantes.

Les *fonctions fenêtre* ne sont autorisées que dans la liste `SELECT` et la clause `ORDER BY` de la requête. Elles sont interdites ailleurs, par exemple dans les clauses `GROUP BY`, `HAVING` et `WHERE`.

Fonctions en langage SQL

Forme générale

```
CREATE OR REPLACE FUNCTION  schema_name.func_name(p_arg1 p_arg1_datatype, ..., [OUT o_arg1 o_arg1_d
RETURNS some type | RECORD | SETOF some type | TABLE (..)
LANGUAGE SQL AS
$$
SQL statement ;
SQL statement ;
...
SQL statement ;
$$ ;
```

Fonctions qui retournent un type simple

Dans le schéma `world`, on veut écrire une fonction qui prend en argument une région et renvoie la population maximale parmi les capitales de la région (voir td2 requête 1).

```
CREATE OR REPLACE FUNCTION username.taille_max_capitale_region(p_region text)
RETURNS INTEGER LANGUAGE SQL AS
$$
SELECT MAX(population_city) AS max_pop
FROM world.country c JOIN world.city cc ON (c.capital=cc.id)
WHERE c.region=p_region AND cc.population_city IS NOT NULL;
$$ ;
```

Fonctions qui retournent un type composé

Dans le schéma `world`, on veut écrire une fonction qui prend en argument une région et renvoie le nom et la population de la capitale la plus peuplée de cette région (voir td2 requête 1).

On peut utiliser le *qualifiant* `OUT` pour désigner des paramètres de sortie.

```
CREATE OR REPLACE FUNCTION username.capitale(p_region text, OUT o_name_capital TEXT, OUT o_populati
RETURNS RECORD  LANGUAGE SQL AS
$$
WITH r AS(
  SELECT cc.*, RANK() OVER (PARTITION BY c.region
                            ORDER BY cc.population_city DESC) AS rnk
  FROM world.country c JOIN world.city cc ON (c.capital=cc.id)
  WHERE c.region=p_region)
SELECT r.name_city,  r.population_city
FROM r
WHERE r.rnk = 1 ;
$$ ;
```

Fonctions qui retournent un type composé défini par les lignes d'une table

Dans le schéma `world`, on veut écrire une fonction qui prend en argument une région et renvoie la description de la capitale la plus peuplée de cette région (voir td2 requête 1).

```
CREATE OR REPLACE FUNCTION username.capitale(p_region text)
RETURNS world.city  LANGUAGE SQL AS
```

```
$$
WITH r AS(
  SELECT cc.*, RANK() OVER (PARTITION BY c.region
                            ORDER BY cc.population_city DESC) AS rnk
  FROM world.country c JOIN world.city cc ON (c.capital=cc.id)
  WHERE c.region=p_region
)
SELECT r.id, r.name_city, r.countrycode, r.district, r.population_city
FROM r
WHERE r.rnk = 1 ;
$$ ;
```

Fonctions qui retournent une table

Dans le schéma `world`, on veut écrire une fonction qui prend en argument une région et renvoie la table des capitales de la région (voir td2 requête 1).

Table de schéma explicite

On se contente d'abord de renvoyer le nom de la capitale. On explicite le schéma de la table résultat

```
CREATE OR REPLACE FUNCTION username.capitales_region(p_region text)
RETURNS TABLE (name_capital text) LANGUAGE SQL AS
$$
SELECT cc.name_city
FROM world.country c JOIN world.city cc ON (c.capital=cc.id)
WHERE c.region=p_region ;
$$ ;
```

Table de même schéma qu'une autre table

Documentation

On veut maintenant récupérer une table de même schéma que `city`.

> La solution est très simple. On profite de ce qu'à chaque table correspond un type de même nom et on utilise le mot-clé `SETOF`.

```
CREATE OR REPLACE FUNCTION username.capitales_region_large(p_region text)
RETURNS SETOF world.city  LANGUAGE SQL AS
$$
SELECT cc.*
FROM world.country c JOIN world.city cc ON (c.capital=cc.id)
WHERE c.region=p_region ;
$$ ;
```

Schéma `babynames`

Les données sont construites et mises à disposition par l'INSEE https://www.insee.fr/fr/accueil, disponibles ici : https://www.insee.fr/fr/statistiques/fichier/2540004/nat2021_csv.zip

Cet ensemble de données s'est développé depuis un certain temps. Il a été pris en compte par les des chercheurs en sciences sociales depuis des décennies. Les prénoms sont censés donner un aperçu d'une variété de phénomènes, y compris l'observance religieuse. phénomènes, y compris l'observance religieuse.

Un aperçu de l'ensemble des travaux peut être trouvé dans *L'archipel français* de Jérome Fourquet, Le Seuil, 2019.

https://www.seuil.com/ouvrage/l-archipel-francais-jerome-fourquet/9782021406023.

```
SET search_path TO babynames ;

\d  bebes

Table "babynames.bebes"
Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
sexe   | text    |           |          |
prenom | text    |           |          |
annee  | integer |           |          |
nombre | integer |           |          |
```

Chaque tuple nous indique le `nombre` de nouveaux-nés de genre `sexe`, prénommés `prenom` mis au monde durant l'année `annee` en France.

Exercices

- Pour chaque année, chaque sexe, donner les dix prénoms les plus populaires et leur rang de popularité.

```
#| unilur-solution: true

WITH R AS (
  SELECT annee, sexe, prenom, RANK() OVER(PARTITION BY (annee, sexe)
                                          ORDER BY nombre DESC) AS rnk
  FROM babynames.bebes
)
SELECT annee, sexe, prenom, rnk
FROM R
WHERE rnk <= 10 ;
```

- Créer dans votre schéma personnel une vue `top_10` qui, interrogée, renvoie le résultat de la requête précédente

Solution

```
CREATE OR REPLACE VIEW babynames.top_10 AS                                    ①
  WITH R AS (
    SELECT annee, sexe, prenom, RANK() OVER(PARTITION BY (annee, sexe) ORDER BY nombre DESC) A
    FROM babynames.bebes
  )

  SELECT annee, sexe, prenom, rnk
  FROM R
  WHERE rnk <= 10
;
```

① Remplacer `babynames.top_10` par `username.top_10`.

- Écrire une fonction qui prend en argument un prénom et renvoie une table de schéma `(annee, sexe, rang)` et indique le rang du prénom pour chaque année et chaque sexe.

Solution

```
CREATE OR REPLACE FUNCTION babynames.fonc_td4bis_1(p_prenom TEXT)
RETURNS TABLE(annee INT4, sexe TEXT, rang INT4)
LANGUAGE SQL AS
$$
WITH R AS (
  SELECT annee, sexe, prenom, RANK() OVER(PARTITION BY (annee, sexe) ORDER BY nombre DESC) AS
  FROM babynames.bebes
)
SELECT annee::int4, sexe::text, rnk::int4
FROM R
WHERE prenom = p_prenom ;
$$ ;
```

Remplacer `babynames.fonc_td4bis_1` par `username.fonc_td4bis_1`.

- Pour chaque année et sexe, calculez le nombre total de naissances.

Solution

```
# unilur-solution: true

SELECT b.annee, b.sexe, SUM(nombre) AS nb
FROM babynames.bebes b
GROUP BY b.annee, b.sexe  ;
```

- Créer dans votre schéma personnel une vue `naissances_sexe_annee` qui, interrogée, renvoie le résultat de la requête précédente

Solution

```
# unilur-solution: true

CREATE OR REPLACE VIEW babynames.naissances_sexe_annee AS(
    SELECT b.annee, b.sexe, SUM(nombre)  AS nb
    FROM babynames.bebes b
    GROUP BY b.annee, b.sexe
) ;
```

- Pour chaque année, calculez le rapport entre nombre total de naissances féminines et nombre total de naissances masculines.

On effectue avec les outils SQL une opération qui s'appelle un *pivot en forme élargie*.

Solution

```
WITH R AS (
 SELECT b.annee, b.sexe, SUM(nombre)  AS nb
 FROM babynames.bebes b
 GROUP BY b.annee, b.sexe
)
SELECT r1.annee, r1.nb::float4/r2.nb::float4 AS ratio
FROM R r1 JOIN R r2 ON(r1.annee=r2.annee)
WHERE r1.sexe=2 AND r2.sexe=1 ;
```

- Créer dans votre schéma personnel une vue `naissances_sexe_annee` qui, interrogée, renvoie le résultat de la requête précédente.

Solution

```
CREATE OR REPLACE VIEW babynames.naissances_f_h_annee AS (
SELECT r1.annee, r1.nb AS nb_f, r2.nb AS nb_h
FROM babynames.naissances_sexe_annee r1 JOIN babynames.naissances_sexe_annee r2 ON(r1.annee=r2
WHERE r1.sexe='2' AND r2.sexe='1'
) ;
```

- Pour chaque année, calculez le rapport entre nombre total de naissances féminines et nombre total de naissances

Solution

```
WITH R AS (
 SELECT b.annee, b.sexe, SUM(nombre)  AS nb
 FROM babynames.bebes b
 GROUP BY b.annee, b.sexe
)
SELECT r1.annee, r1.nb::float4/(r1.nb::float4 + r2.nb::float4) AS ratio    ①
FROM R r1 JOIN R r2 ON(r1.annee=r2.annee)
WHERE r1.sexe='2' AND r2.sexe='1' ;
```

① Pourquoi est-il utile de réaliser une coercition de type sur `nb` avec `nb::float4` ?

- Créer dans votre schéma personnel une vue ‘naissances_sexe_annee qui, interrogée, renvoie le résultat de la requête précédente

Solution

```
CREATE OR REPLACE VIEW babynames.naissances_sexe_annee AS (
    SELECT b.annee, b.sexe, SUM(nombre)  AS nb
    FROM babynames.bebes b
    GROUP BY b.annee, b.sexe
) ;
```

- Écrire une fonction qui prend en argument une année `yyyy` et renvoie une table de schéma `(annee, sexe, prenom, prop)` où `prenom` est un des dix prénoms les plus populaires donné en l’aneee `yyyy` et calculer la proportion de nouveaux-nés portant ce prénom année par année.

Tenez compte du fait que certains noms peuvent avoir une occurrence nulle pendant certaines années.

Solution

```
CREATE OR REPLACE FUNCTION babynames.fonc_td4bis_2(p_annee int4)
RETURNS TABLE(annee int4, sexe TEXT, prenom TEXT, prop float4) LANGUAGE SQL AS
$$
WITH r AS (
  SELECT sexe, prenom
  FROM babynames.top_10 t
  WHERE t.annee=p_annee
), s AS (
  SELECT b.annee, r.sexe, r.prenom, b.nombre
  FROM r  LEFT OUTER JOIN babynames.bebes b
  ON(r.sexe=b.sexe AND r.prenom=b.prenom)
)
SELECT s.annee, s.sexe, s.prenom, s.nombre::float4/nsa.nb::float4 AS prop
FROM s JOIN babynames.naissances_sexe_annee nsa
ON (s.annee=nsa.annee  AND  s.sexe=nsa.sexe) ;
$$ ;
```

https://www.postgresql.org/docs/15/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS