

## SQL Avancé



P.Mathieu

LP DA2I Lille  
<http://www.iut-a.univ-lille.fr>  
[prenom.nom@univ-lille.fr](mailto:prenom.nom@univ-lille.fr)

23 septembre 2019

### 1 Les valeurs NULL

### 2 Tuples

### 3 La fonctionnalité UPSERT

### 4 Les tables dérivées (derived tables)

### 5 Les common table expression (CTE)

### 6 Fonctions OLAP

### 7 Window functions

### 8 Tables virtuelles

## Les valeurs NULL

Définition de tuples dans les requêtes



- le test :
  - IS NULL**
  - NULLIF**(argument\_1, argument\_2);
- le calcul : pas prises en compte dans les fonctions d'agrégat
- COALESCE**(expr1, expr2, ..., expr\_n) renvoie la première expr non nulle

```
SELECT coalesce(note,0) FROM notes WHERE netu=11;
```

```
SELECT AVG(coalesce(note,0)) FROM notes WHERE netu=11;
```

```
SELECT (CASE WHEN note IS NULL THEN 0 ELSE note END)  
FROM notes  
WHERE netu=11;
```

### 1 Les valeurs NULL

### 2 Tuples

### 3 La fonctionnalité UPSERT

### 4 Les tables dérivées (derived tables)

### 5 Les common table expression (CTE)

### 6 Fonctions OLAP

### 7 Window functions

### 8 Tables virtuelles

## Tuples

Définition de tuples dans les requêtes

```
SELECT * FROM notes WHERE (mat, ncont) = ('bdd',1);
```

## Tuples

Définition de tuples dans les requêtes

*Effacer de notes les couples (nom,prenom) présents dans t*

```
DELETE FROM notes
WHERE nom IN (SELECT nom FROM t)
AND prenom IN (SELECT prenom FROM t);
```

CETTE SOLUTION EST INCORRECTE !

## Tuples

Définition de tuples dans les requêtes

*Effacer de notes les couples (nom,prenom) présents dans t*

```
DELETE FROM etudiants
WHERE (nom,prenom) IN (SELECT nom, prenom FROM t);
```

Ce qui pourrait aussi s'écrire ...(en corrélatif)

```
DELETE FROM etudiants
WHERE EXISTS (SELECT *
              FROM t
              WHERE etudiant.nom=t.nom
              AND   etudiant.prenom=t.prenom
```

1 Les valeurs NULL

2 Tuples

3 La fonctionnalité UPSERT

4 Les tables dérivées (derived tables)

5 Les common table expression (CTE)

6 Fonctions OLAP

7 Window functions

8 Tables virtuelles

## La fonctionnalité UPSERT

upsert (update or insert)

SQL :2003. Autorise à insérer ou mettre à jour une ligne selon qu'elle existe ou pas , en une seule instruction.

- merge oracle, db2, Firebird
- on duplicate key update mysql
- insert or replace into sqlite
- on conflict do postgresql
- etc ...

## La fonctionnalité UPSERT

avec Postgres

La clause ON CONFLICT target DO action est ajoutée à l'ordre INSERT

```
CREATE TABLE etu(nom text PRIMARY KEY, age int);  
INSERT INTO etu VALUES ('paul',10);
```

```
INSERT INTO etu VALUES ('paul',20)  
ON CONFLICT (nom) DO NOTHING;
```

## La fonctionnalité UPSERT

upsert (update or insert)

target peut être :

- une colonne unique
- ON CONSTRAINT nom contrainte unique
- WHERE predicat

action peut être :

- DO NOTHING
- DO UPDATE SET ...affectations

```
INSERT INTO etu VALUES ('paul',20)  
ON CONFLICT (nom) DO UPDATE SET age=EXCLUDED.age;
```

1 Les valeurs NULL

2 Tuples

3 La fonctionnalité UPSERT

4 Les tables dérivées (derived tables)

5 Les common table expression (CTE)

6 Fonctions OLAP

7 Window functions

8 Tables virtuelles

## Les tables dérivées (derived tables)

Sous-requête dans la clause FROM

Depuis SQL :1999, la clause from peut contenir une sous-requête (table virtuelle)

*Liste ordonnée des moyennes par étudiant*

```
SELECT *  
FROM (SELECT netu, AVG(note) AS moy  
      FROM notes GROUP BY netu) AS t  
WHERE t.moy > 10;
```

Evidemment, dans ce cas ci la table dérivée est inutile

## Les tables dérivées (derived tables)

Sous-requête dans la clause FROM

*Moyenne du nombre de notes de chaque étudiant*

```
SELECT AVG(COUNT(*)) FROM notes GROUP BY netu ;
```

NE FONCTIONNE PAS

*Version correcte*

```
SELECT AVG(nb)  
FROM (SELECT netu, COUNT(*) AS nb  
      FROM notes  
      GROUP BY netu) AS r;
```

## Les tables dérivées (derived tables)

exemple avec jointures

*Il est bien sûr possible d'y mettre des jointures*

```
SELECT etu, nb, moy  
FROM (SELECT etu, COUNT(*) AS nb FROM notes GROUP BY etu) AS r  
INNER JOIN  
(SELECT etu, AVG(note) AS moy FROM notes GROUP BY etu) AS s  
USING (etu)  
WHERE moy < 10  
;
```

1 Les valeurs NULL

2 Tuples

3 La fonctionnalité UPSERT

4 Les tables dérivées (derived tables)

5 Les common table expression (CTE)

6 Fonctions OLAP

7 Window functions

8 Tables virtuelles

## Les common table expression (CTE)

Tables virtuelles

SQL :1999 offre la possibilité de définir une vue virtuelle temporaire à la requête : la clause **WITH**

```
WITH query_name1 AS (SELECT ... )
, query_name2 AS (SELECT ...
                  FROM query_name1
                  ...
                  )
SELECT ...
```

## Les common table expression (CTE)

Tables virtuelles

*Mooyenne des étudiants pour ceux qui ont une moyenne au dessus de 10*

```
WITH cte(netu,moy) AS (SELECT netu,AVG(note)
                       FROM notes GROUP BY netu)
SELECT netu, moy
FROM cte
WHERE moy > 10;
```

## Les common table expression (CTE)

Tables virtuelles

*Etudiants à la moyenne la plus élevée (maximum)*

```
SELECT MAX(AVG(note)) FROM notes GROUP BY etu ;
```

NE FONCTIONNE PAS

*Version correcte*

```
WITH cte AS (SELECT netu, AVG(note) AS moy
             FROM notes
             GROUP BY netu)
SELECT netu, moy
FROM cte
WHERE moy = (SELECT MAX(moy) FROM cte);
```

## Les common table expression (CTE)

Tables virtuelles

*Etudiants à la moyenne la plus élevée (maximum)*

```
SELECT MAX(AVG(note)) FROM notes GROUP BY etu ;
```

NE FONCTIONNE PAS

*ou à l'ancienne manière*

```
SELECT netu, AVG(note)
FROM notes
GROUP BY netu
HAVING AVG(note) >=ALL (SELECT AVG(note)
                       FROM notes
                       GROUP BY netu);
```

## Les common table expression (CTE)

### Les différentes approches

- Une vue est
  - ▶ référencée au catalogue,
  - ▶ persiste hors de la session
  - ▶ peut être utilisée par plusieurs utilisateurs
- Une table temporaire
  - ▶ ne persiste pas hors de la session
  - ▶ est propre à l'utilisateur
  - ▶ peut être utilisée dans plusieurs requêtes
- Une CTE
  - ▶ est à usage unique dans une requête
  - ▶ Peut être récursive

## Les common table expression (CTE)

### Une CTE récursive

s'exprime du coup avec le mot clé **WITH RECURSIVE** en faisant en sorte que la table virtuelle fasse référence à elle-même !

l'idée est la suivante :

- 1 le **WITH RECURSIVE** permet de définir la vue virtuelle initiale.
- 2 L'opérateur **UNION** (ou **UNION ALL** selon les cas) permet de définir la clause récursive
- 3 la partie récursive est exécutée sur les nouvelles données ajoutées à chaque tour
- 4 recommencer 3 tant qu'il y a des nouvelles données (et donc bien s'assurer que ça ne boucle pas et qu'à un moment il n'y aura plus de données)
- 5 la requête se termine avec un **select \*** de la table virtuelle

## Les common table expression (CTE)

### Une CTE récursive

*la suite des puissances de 2 jusque 100*

```
WITH RECURSIVE t(nombre) AS (  
    VALUES (2)  
    UNION ALL  
    SELECT 2 * nombre FROM t WHERE 2 * nombre < 100  
)  
SELECT * FROM t;
```

## Les common table expression (CTE)

### Une CTE récursive

Le cas classique `employe(id,nom,superieur)`

*Arbre hiérarchique de l'employé 10*

```
WITH RECURSIVE hierarchie(id, nom, superieur) AS (  
    SELECT * FROM employes WHERE id = 10  
    UNION ALL  
    SELECT e.id, e.nom, e.superieur  
    FROM hierarchie AS h, employes AS e  
    WHERE h.superieur = e.id  
)  
SELECT * FROM hierarchie;
```

## Les common table expression (CTE)

Une CTE récursive

Le cas classique `employe(id, nom, superieur)`

*nombre d'employés sous les ordres du 1*

```
WITH RECURSIVE souslesordres(id, nom, superieur) AS (  
  SELECT * FROM employes WHERE id = 1  
  UNION ALL  
  SELECT e.id, e.nom, e.superieur  
  FROM souslesordres AS s, employes AS e  
  WHERE e.superieur = s.id  
)  
SELECT COUNT(*) - 1 FROM souslesordres;
```

## Les common table expression (CTE)

En résumé

Différentes formes et emplacements de sous-requêtes :

- Imbriquées
- Corrélées
- Dans la clause SELECT
- Dans la clause FROM
- Dans la clause WHERE
- Dans la clause HAVING
- CTE (Common Table Expression)
- Requêtes récursives via CTE

1 Les valeurs NULL

2 Tuples

3 La fonctionnalité UPSERT

4 Les tables dérivées (derived tables)

5 Les common table expression (CTE)

6 Fonctions OLAP

7 Window functions

8 Tables virtuelles

## Fonctions OLAP

Le GROUP BY classique

```
SELECT mat, ncont, AVG(note) AS moy  
FROM notes  
GROUP BY mat, ncont;
```

Fournit la valeur agrégée pour chaque tuple distinct issu des colonnes du group by

mat	ncont	moy
bdd	c1	8,5
bdd	c2	12,1
bdd	c3	14,7
syst	c1	7,2
syst	c2	9,4
syst	c3	11,1
gestion	c1	15,9
gestion	c2	13,2
gestion	c3	10,7

	c1	c2	c3
bdd	8,5	12,1	14,7
syst	7,2	9,4	11,1
gestion	15,9	13,2	10,7

## Fonctions OLAP

### Le ROLLUP

SQL 1999 : Extension des fonctions d'agrégation

```
SELECT mat, ncont, groupe, AVG(note)
FROM notes
GROUP BY ROLLUP(mat, ncont, groupe);
```

- Ajoute tous les sous-ensembles ordonnés ainsi qu'un total général
- Ici, avg pour (mat, ncont, groupe), puis avg pour (mat, ncont), puis avg pour (mat), puis avg général ()
- Si n est le nombre de colonnes dans le rollup, il y aura  $n + 1$  sous-totaux supplémentaires

## Fonctions OLAP

### Le CUBE

```
SELECT mat, ncont, groupe, AVG(note)
FROM notes
GROUP BY CUBE(mat, ncont, groupe);
```

- Ajoute toutes les combinaisons de sous-totaux
- Ici, avg pour (mat, numcont), puis avg pour mat, puis avg général
- Si n est le nombre de colonnes dans le cube, il y aura  $2^n$  sous-totaux supplémentaires

## Fonctions OLAP

### En résumé ...

**GROUP BY(a,b,c)**  
(a,b,c)

**ROLLUP (a,b,c)**  
(a,b,c)  
(a,b)  
(a)  
( )

**CUBE (a,b,c)**  
(a, b, c)  
(a, b)  
(a, c)  
(a)  
(b, c)  
(b)  
(c)  
( )

## Fonctions OLAP

### GROUPING SET

- Le calcul de CUBE est couteux et fournit en général plus d'agrégations que nécessaire.
- La clause `GROUPING SET` permet de spécifier uniquement ceux que l'on souhaite.

```
SELECT mat, ncont, groupe, AVG(note)
FROM notes
GROUP BY GROUPING SETS (mat, ncont);
```

Regroupe sur mat, puis sur ncont



## Fonctions OLAP

### GROUPING SET

- Plusieurs GROUPING SET sont possibles
- Utilise tous les produits cartésiens de chaque ensemble

**GROUP BY GROUPING SETS** (a,b)

(a)

(b)

**GROUP BY GROUPING SETS** (a), **GROUPING SETS** (b)

(a,b)

**GROUP BY GROUPING SETS** (a,b), **GROUPING SETS** (c,d)

(a,c)

(a,d)

(b,c)

(b,d)

1 Les valeurs NULL

2 Tuples

3 La fonctionnalité UPSERT

4 Les tables dérivées (derived tables)

5 Les common table expression (CTE)

6 Fonctions OLAP

7 Window functions

8 Tables virtuelles

## Window functions

### La clause OVER

- SQL :2003 et SQL :2008 introduisent le mot clé **OVER** à mettre dans le **SELECT** pour agréger des informations sans faire de **GROUP BY**
- Contrairement à un **GROUP BY** classique, toutes les lignes sélectionnées sont affichées

**SELECT** netu, note, **SUM**(note) **OVER** () **FROM** notes ;

Equivalent à

**SELECT** netu, note, (**SELECT SUM**(note) **FROM** notes) **FROM** notes ;

## Window functions

### La clause PARTITION BY

Le calcul précédent se faisait sur la totalité de la table. Grâce à la clause **PARTITION BY**, il peut se faire sur une partie de la table

**SELECT SUM**(note) **OVER** (**PARTITION BY** groupe), netu,note  
**FROM** notes ;

Equivalent à

**SELECT** (**SELECT SUM**(note)  
**FROM** notes **GROUP BY** groupe), netu,note  
**FROM** notes;

## Window functions

### La partition ordonnée

La partition calculée peut être ordonnée. Elle permet de ce fait des calculs spécifiques à l'ordre de la ligne sélectionnée.

```
SELECT *, rank() OVER (ORDER BY note) FROM notes ;
```

```
SELECT *, rank() OVER (PARTITION BY groupe  
                        ORDER BY note)  
FROM notes ;
```

## Window functions

### La partition ordonnée

Ce dernier aspect permet d'introduire de nouveaux opérateurs liés à l'ordre

rank()	rang avec gaps en cas d'égalité
dense_rank()	rang sans gap
percent_rank()	rang relatif à la totalité
ntile()	quantiles
...	...

## Window functions

### Calcul de quantiles

*Numérotation des quantiles issus des notes (ici médiane)*

```
SELECT ntile(2) OVER (ORDER BY note), * FROM notes
```

*Numérotation des quantiles issus des notes par groupe (ici médiane)*

```
SELECT ntile(2) OVER (PARTITION BY groupe  
                        ORDER BY note), *  
FROM notes
```

*Calcul du dernier décile*

```
SELECT ntile(10) OVER (PARTITION BY groupe  
                        ORDER BY note), *  
FROM notes
```

## Window functions

### Et pour filtrer ....

*Etudiants du premier décile*

```
WITH cte AS (SELECT ntile(10) OVER (PARTITION BY groupe  
                                    ORDER BY note) AS quantile, *  
              FROM notes)  
SELECT *  
FROM cte  
WHERE quantile=1;
```

- 1 Les valeurs NULL
- 2 Tuples
- 3 La fonctionnalité UPSERT
- 4 Les tables dérivées (derived tables)
- 5 Les common table expression (CTE)
- 6 Fonctions OLAP
- 7 Window functions

## 8 Tables virtuelles

P.Mathieu (LP DA2I Lille1)

SQL Avancé

23 septembre 2019

41 / 42

## Tables virtuelles

Fonctions qui renvoient plusieurs lignes

```
SELECT * FROM generate_series(2,4);
```

```
generate_series
-----
                2
                3
                4
```

```
SELECT * FROM generate_series(5,1,-2);
```

```
generate_series
-----
                5
                3
                1
```

```
SELECT * FROM generate_series('2018-03-01 00:00'::TIMESTAMP,
                                '2018-03-04 12:00', '10 hours');
```

P.Mathieu (LP DA2I Lille1)

SQL Avancé

23 septembre 2019

42 / 42