Requêtes récursives avec la clause WITH (« CTE »)

& Fonctions « Window »

Dans PostgreSQL

De la théorie à la pratique

JM Souchard
jm.souchard@gmail.com

Au programme

Introduction

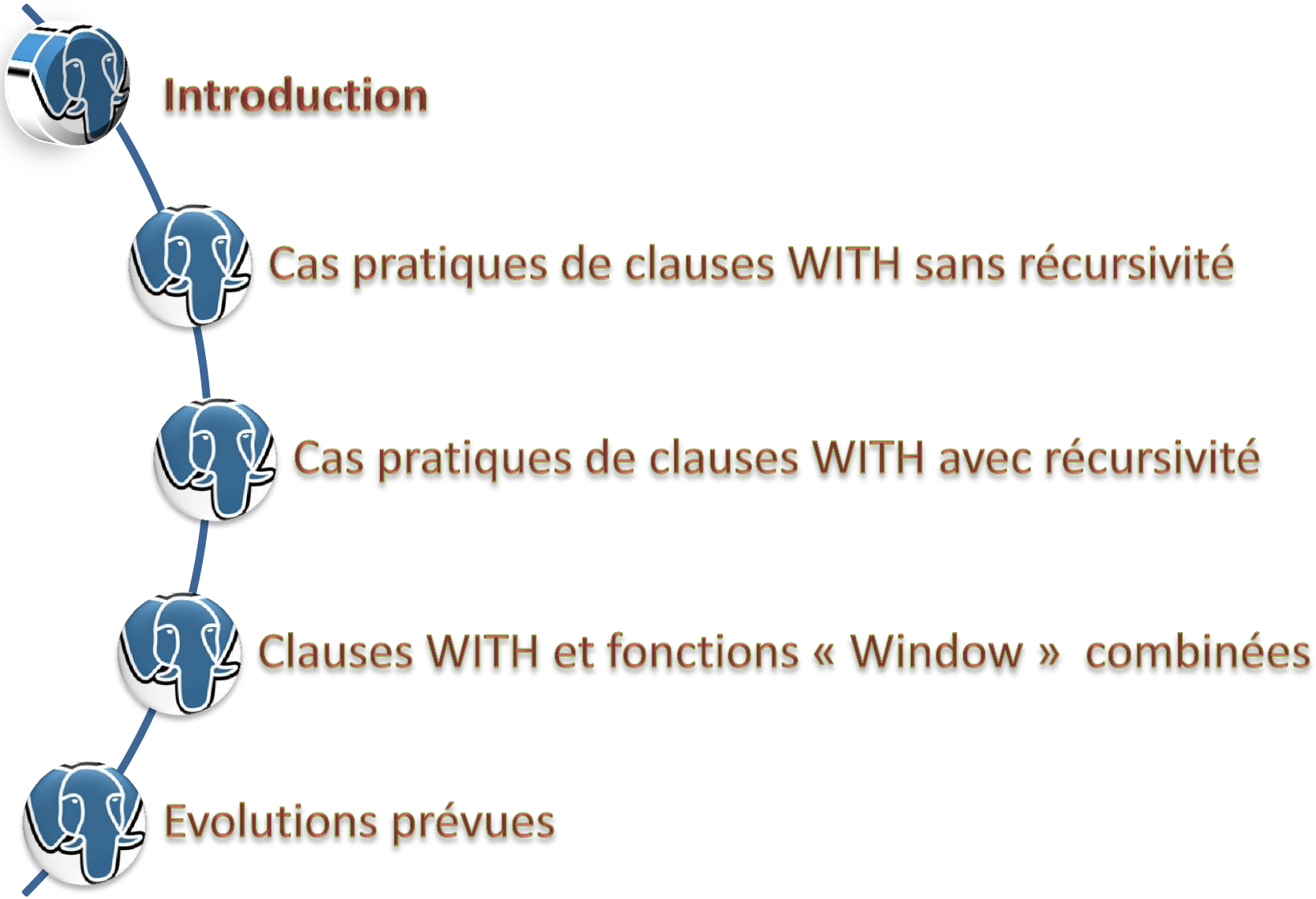
Cas pratiques de clauses WITH sans récursivité

Cas pratiques de clauses WITH avec récursivité

Clauses WITH et fonctions « Window » combinées

Les évolutions prévues

Introduction



INTRODUCTION

- Les concepts
- La norme SQL
- L'implémentation chez les éditeurs
- Syntaxe et exemples dans PostgreSQL

Les concepts

INTRODUCTION

La clause WITH est l'implémentation des « Common Table Expression »

- ***CTE ou « expression de table » en français***
- ***Vues non instanciées ou sous-requêtes utilisables par la requête dans laquelle elles figurent, et ce afin de factoriser des expressions***
- ***Permet également l'écriture de requêtes récursives***

Les fonctions « Window »

- **Fonction de fenêtrage appliquée à une partition d'un résultat de requêtes**
- **Deux syntaxes sont possibles : l'une spécifie la fenêtre de données directement après la clause OVER, l'autre utilise la clause WINDOW avant la clause ORDER BY de l'ordre SELECT**

La norme SQL

Année	Nom	Appellation	Commentaires sur les apparitions des fonctionnalités étudiées
1999	ISO/CEI 9075:1999	SQL-99 ou SQL3	*Requêtes récursives avec le mot clé RECURSIVE ainsi que des clauses de recherches et de cycle (SEARCH et CYCLE)*
2003	ISO/CEI 9075:2003	SQL:2003	*Fonctions « window »*
2008	ISO/CEI 9075:2008	SQL:2008	*Ajout de fonctions « Window » : NTILE, LEAD, LAG, first value, last value, nth value*

Implémentation chez les éditeurs

INTRODUCTION

SGBD	CTE (Clause With)	Fonctions « Window »
PostgreSQL	PostgreSQL 8.4 (2009)	PostgreSQL 8.4 (2009)
Oracle	Oracle 9i (2001 - *sans récursivité*) Oracle 11gR2 (2009 - *avec récursivité et possibilité de déterminer le mode de stockage des vues intermédiaires*)	Oracle 11g (2007 - *sans la clause WINDOW*)
DB2	DB2 UDB 8 (*Pas de mot clé RECURSIVE*)	DB2 9 (2006)
MySQL	Pas implémenté	Pas implémenté
Ingres	Pas implémenté	Pas implémenté
SQL Server	SQL Server 2005 (*Pas de mot clé RECURSIVE, une seule référence/requête*) SQL Server 2008 (*disparition de la limitation présente en version 2005 et apparition de la clause MAXRECURSION pour limiter la profondeur max. Possibilité de multiple récursions*)	SQL Server 2008
SYBASE	SQL Anywhere 9.0 (2003)	SQL Anywhere 10.0 (2006)

Un sondage révélateur…

Quelle fonctionnalité de la version 8.4 de PostgreSQL vous intéresse le plus ?

Réponses	*Nb réponses*	*Pourcentage*
Requêtes récursive & CTEs	**49**	**24.378%**
Fonctions Windowing	**49**	**24.378%**
Restauration Parallèle	20	9.950%
« Visibility Map » & moins de VACUUM	33	16.418%
Meilleures connexions SSL	9	4.478%
Permissions sur les colonnes	20	9.950%
Paramètres fonctions « Default & Variadic »	6	2.985%
Autre	15	7.463%
Total	**201**	

Syntaxe PostgreSQL de la clause WITH

INTRODUCTION

WITH [RECURSIVE]

<Nom sous requête1> **[(***<colonne>* **[***<type>***] [,…])] AS**
([SELECT | VALUES] *<1ère partie de la sous requête>*
[UNION [ALL]
SELECT *<2ème partie récursive de la sous requête>***])**

Sous requête1

[, *<sous_requête_2>*

[, *<sous_requête_n >***]]**

SELECT *<corps de la requête principale>*
;

Requête principale

Exemples avec la clause WITH

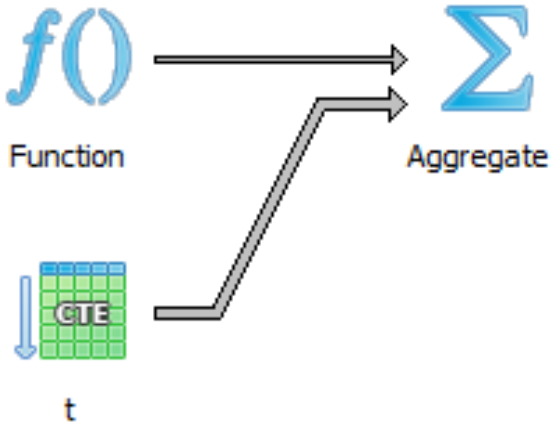
Calculer la somme des 10 premiers entiers avec les requêtes ci-dessous

INTRODUCTION

```
SELECT SUM(n)
FROM   (VALUES (1),(2),(3),(4),(5),
               (6),(7),(8),(9),(10)
        ) t(n);
```


```
SELECT SUM(GENERATE_SERIES)
FROM   GENERATE_SERIES(1,10);
```


```
WITH   t(n) AS
       (SELECT GENERATE_SERIES AS n
        FROM   GENERATE_SERIES(1,10)
        )
SELECT SUM(n) FROM t;
```

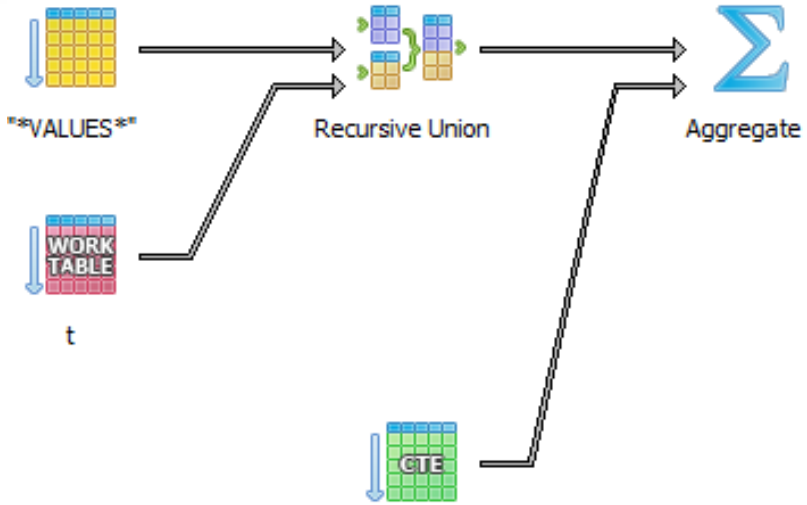


Exemples avec la clause WITH

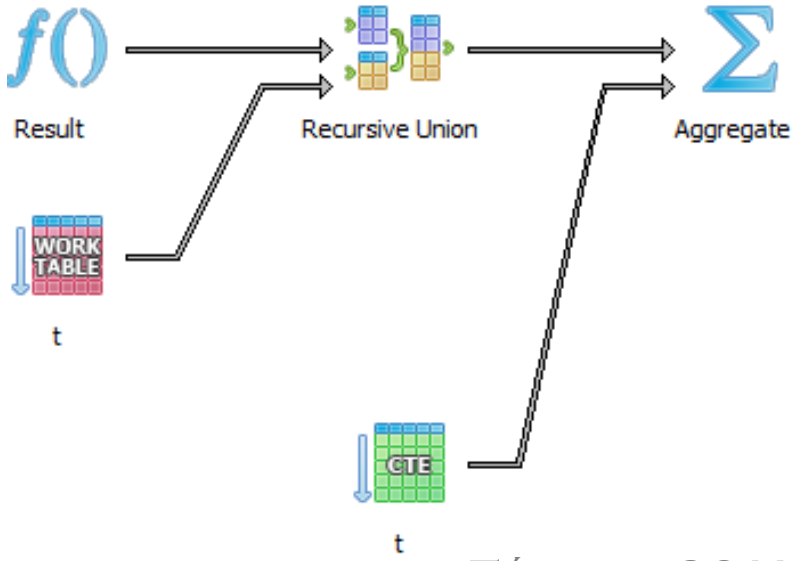
Calculer la somme des 10 premiers entiers avec les requêtes ci-dessous

INTRODUCTION

```
WITH     RECURSIVE
t(n) AS  (VALUES (1)
          UNION ALL
          SELECT n+1
          FROM t
          WHERE n < 10)
SELECT   SUM(n) FROM t;
```




```
WITH     RECURSIVE
t(n) AS  (SELECT 1
          UNION ALL
          SELECT n+1
          FROM t
          WHERE n < 10)
SELECT   SUM(n) FROM t;
```



Syntaxe PostgreSQL fonction «Window»

INTRODUCTION

SELECT [<expressions>,]

<fonction fenêtrée> OVER

[([PARTITION BY (<liste expression groupage>)]
[ORDER BY <liste expression tri>])]

| [<nom fenêtre>]

Partie commune des fonctions « Window »

Première syntaxe possible

Début de deuxième syntaxe possible

[FROM <clause from>]

[WINDOW <nom fenêtre> AS (
[PARTITION BY (<liste expression groupage>)]
[ORDER BY <liste expression tri>])

Fin de deuxième syntaxe possible

[ORDER BY <clause de tri>];

Les listes PARTITION BY et ORDER BY ont globalement les mêmes syntaxes et fonctionnalités que les clauses GROUP BY et ORDER BY de l'ordre SELECT.

Les fonctions fenêtrés PostgreSQL

INTRODUCTION

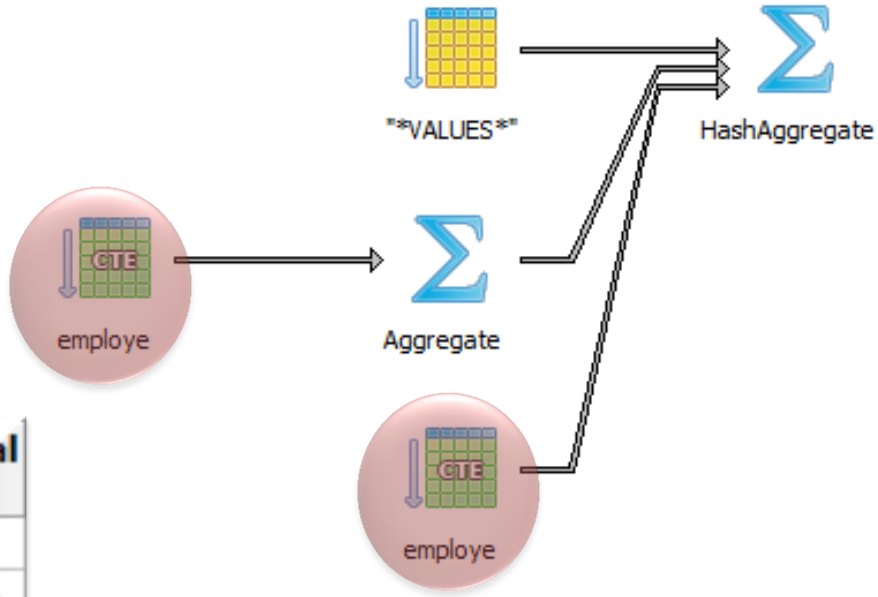
Fonction	Description
MIN, MAX, AVG, COUNT, SUM, …	Fonctions d'agrégat traditionnelles.
ROW_NUMBER()	Numéro de la ligne en cours de traitement dans sa partition, en commençant à 1
RANK()	Rang de la ligne en cours de traitement, avec des trous. Ainsi les valeurs 5,7,7,8,8,8,10 donnent les rangs 1,2,2,4,4,4,7
DENSE_RANK()	Rang de la ligne en cours de traitement, sans trous. Ainsi les valeurs 5,7,7,8,8,8,10 donnent les rangs 1,2,2,3,3,3,4
FIRST_VALUE(*val*)	Renvoie la première valeur de la fenêtre
LAST_VALUE(*value*, any)	Renvoie la dernière valeur de la fenêtre
NTH_VALUE(*val*,any, *nième*,integer)	Renvoie la nième valeur de la fenêtre (en commençant à partir de 1)
Et bien d'autres…	Cf. doc PostgreSQL 9.0 à partir de la page 182 (**9.19. Fonctions Window**)

Exemple de fonctions «Window»

INTRODUCTION

Calculer la moyenne des salaires, la masse salariale et le pourcentage de masse salariale par département avec les requêtes ci-dessous :

```
WITH   employe (matricule,nom,id_dept,fonction,salaire) AS (
VALUES (1,'Dupond',1,'CP',3000),(2,'Durand',1,'AP',5000),
       (3,'Henri',1,'CP',2500),(4,'Smith',1,'AP',2900),
       (5,'Scott',2,'SA',1800),(6,'Hidalgo',2,'CP',3500))
SELECT id_dept AS dept,
       TRUNC(AVG(salaire)) AS moy_salaire,SUM(salaire) AS mass_sal,
       TRUNC(SUM(salaire)*100.0/
       (SELECT SUM(salaire) FROM employe),2) AS pc_mass_sal
FROM   employe
GROUP  BY id_dept;
```



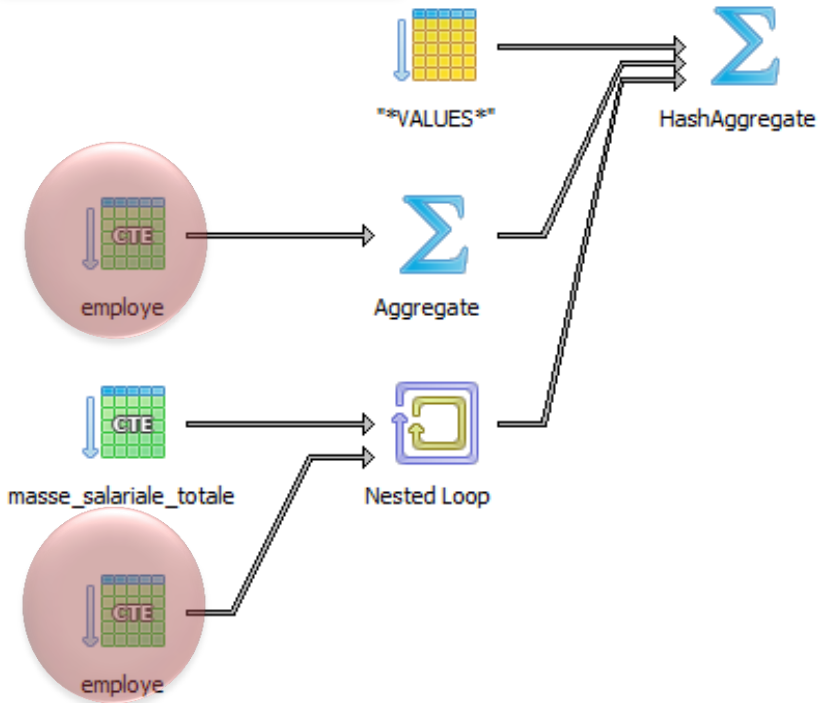
dept integer	moy_salaire numeric	mass_sal bigint	pc_mass_sal numeric
1	3350	13400	71.65
2	2650	5300	28.34

Exemple de fonctions «Window»

INTRODUCTION

Calculer la moyenne des salaires, la masse salariale et le pourcentage de masse salariale par département avec les requêtes ci-dessous :

```
WITH   employe (matricule,nom,id_dept,fonction,salaire) AS (
VALUES (1,'Dupond',1,'CP',3000),(2,'Durand',1,'AP',5000),
       (3,'Henri',1,'CP',2500),(4,'Smith',1,'AP',2900),
       (5,'Scott',2,'SA',1800),(6,'Hidalgo',2,'CP',3500)),
       masse_salariale_totale (total_salaire) AS (
SELECT SUM(salaire) FROM employe)
SELECT id_dept AS dept,TRUNC(AVG(salaire)) AS moy_salaire,
       SUM(salaire) AS mass_sal,
       TRUNC(SUM(salaire)*100.0/MAX(total_salaire),2) AS pc_mass_sal
FROM   employe
CROSS  JOIN
       masse_salariale_totale
GROUP  BY id_dept;
```



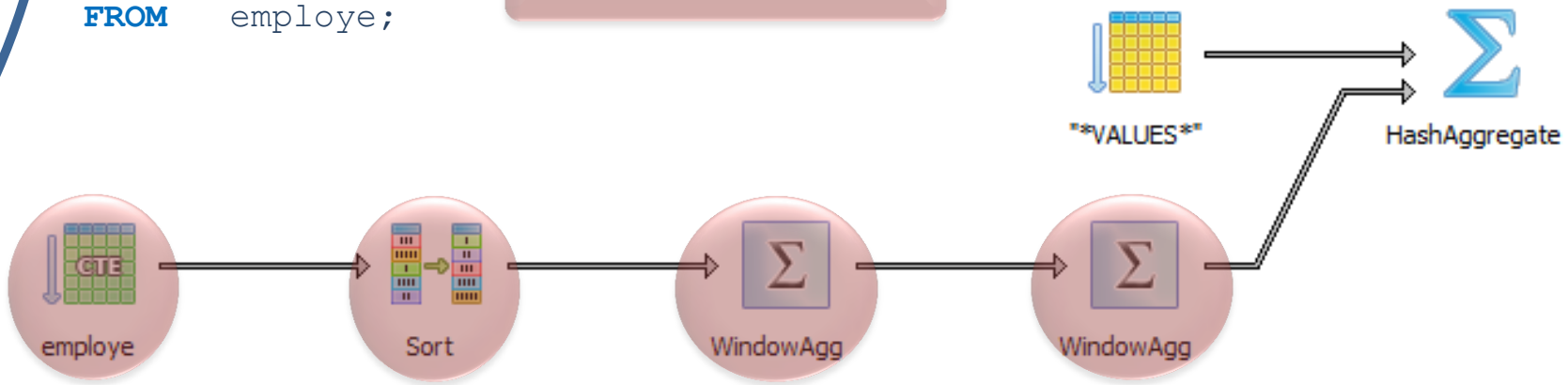
Exemple de fonctions «Window»

INTRODUCTION

Calculer la moyenne des salaires, la masse salariale et le pourcentage de masse salariale par département avec les requêtes ci-dessous :

```
WITH    employe (matricule,nom,id_dept,fonction,salaire) AS (
VALUES (1,'Dupond',1,'CP',3000),(2,'Durand',1,'AP',5000),
       (3,'Henri',1,'CP',2500),(4,'Smith',1,'AP',2900),
       (5,'Scott',2,'SA',1800),(6,'Hidalgo',2,'CP',3500))
SELECT DISTINCT id_dept AS dept,
       TRUNC( AVG(salaire)
              OVER (PARTITION BY id_dept ORDER BY id_dept)
            ) AS moy_salaire,
       SUM(salaire)
       OVER (PARTITION BY id_dept ORDER BY id_dept) AS mass_sal,
       TRUNC( SUM(salaire)
              OVER (PARTITION BY id_dept ORDER BY id_dept)

              *100.0/ SUM(salaire) OVER (),2) AS pc_mass_sal
FROM    employe;
```

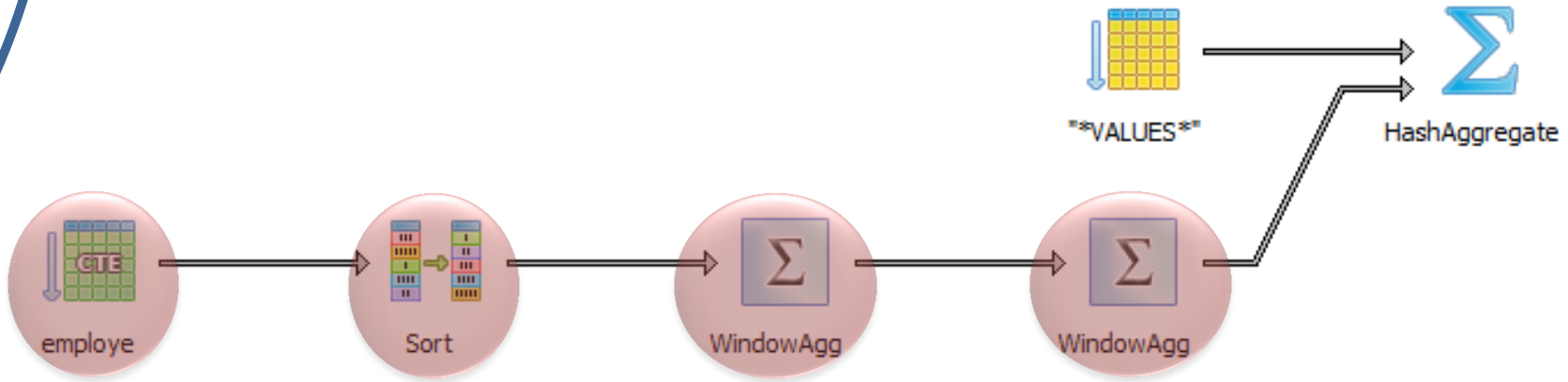


Exemple de fonctions «Window»

INTRODUCTION

Calculer la moyenne des salaires, la masse salariale et le pourcentage de masse salariale par département avec les requêtes ci-dessous :

```
WITH   employe (matricule,nom,id_dept,fonction,salaire) AS (
VALUES (1,'Dupond',1,'CP',3000),(2,'Durand',1,'AP',5000),
       (3,'Henri',1,'CP',2500),(4,'Smith',1,'AP',2900),
       (5,'Scott',2,'SA',1800),(6,'Hidalgo',2,'CP',3500))
SELECT DISTINCT id_dept AS dept,
       TRUNC(AVG(salaire) OVER regroup_dept ) AS moy_salaire,
       SUM(salaire)       OVER regroup_dept AS mass_sal,
       TRUNC(SUM(salaire) OVER regroup_dept
             *100.0/SUM(salaire) OVER (),2) AS pc_mass_sal
FROM   employe
WINDOW regroup_dept AS (PARTITION BY id_dept ORDER BY id_dept);
```



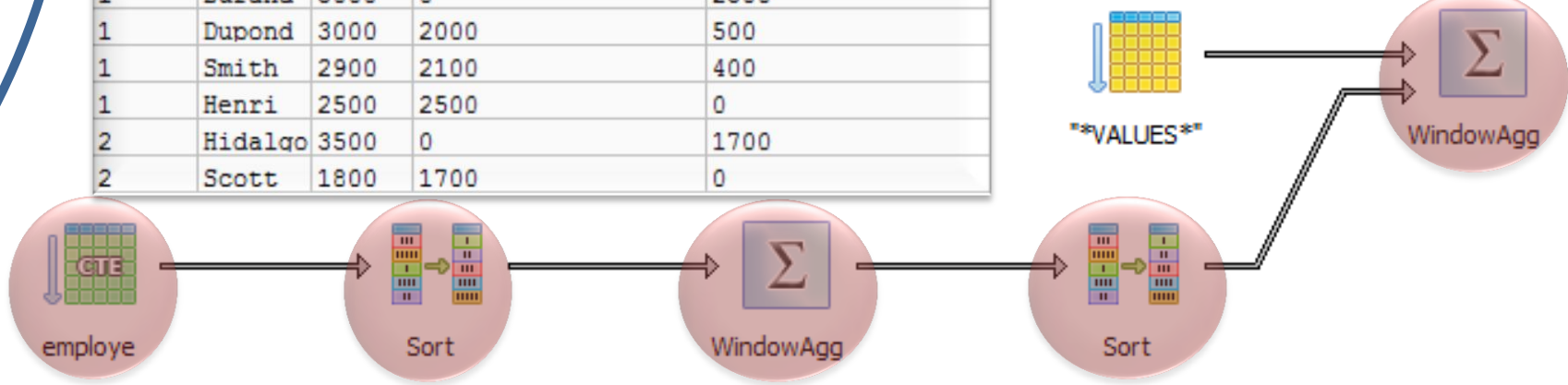
Exemple de fonctions «Window»

INTRODUCTION

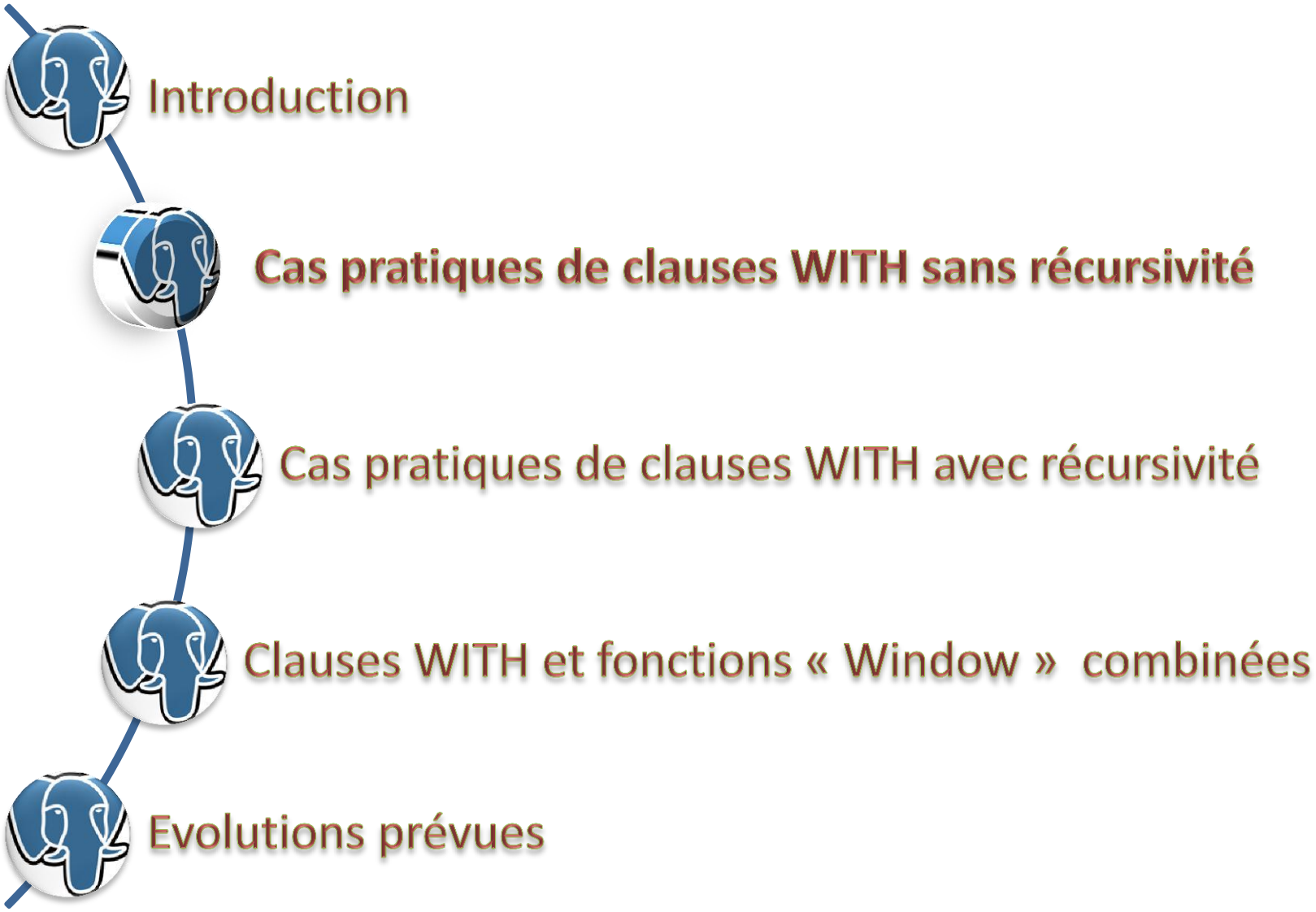
Calculer l'écart de chaque salarié par rapport au salaire minimum et au salaire maximum de chaque département :

```
WITH    employe (matricule,nom,id_dept,fonction,salaire) AS (
VALUES (1,'Dupond',1,'CP',3000),(2,'Durand',1,'AP',5000),
       (3,'Henri',1,'CP',2500),(4,'Smith',1,'AP',2900),
       (5,'Scott',2,'SA',1800),(6,'Hidalgo',2,'CP',3500))
SELECT id_dept,nom,salaire,
       MAX(salaire) OVER sal_max - salaire AS "Ecart/max salaire dept",
       salaire-MIN(salaire) OVER sal_min AS "Ecart/min salaire dept"
FROM   employe
WINDOW sal_min AS (PARTITION BY id_dept
                   ORDER BY id_dept,salaire),
       sal_max AS (PARTITION BY id_dept
                   ORDER BY id_dept,salaire DESC);
```

id_dept integer	nom text	salaire integer	Ecart/max salaire dept integer	Ecart/min salaire dept integer
1	Durand	5000	0	2500
1	Dupond	3000	2000	500
1	Smith	2900	2100	400
1	Henri	2500	2500	0
2	Hidalgo	3500	0	1700
2	Scott	1800	1700	0



WITH sans récursivité



WITH RECURSIVITE SANS

- Exploiter les logs PostgreSQL
- Rechercher de mots clés dans un forum
- Fonction de génération aléatoire de nom
- Charger une base de données
- Anonymiser des données personnelles

Exploiter les logs PostgreSQL

On peut exploiter les logs PostgreSQL en les stockant dans une table, puis en construisant une requête qui affiche les requêtes les plus fréquentes et les plus longues :

WITH RECURSIVE

Création de table → Chargement du journal → Affichage des durées et fréquences

```
CREATE  TABLE log_postgresql_a_analyser (date_heure_log TIMESTAMP(3) WITH TIME ZONE,
utilisateur TEXT,base_de_donnees TEXT,id_process INTEGER,connection_from TEXT,id_session TEXT,
num_ligne_session BIGINT,indicatif_commande TEXT,date_heure_debut_session TIMESTAMP WITH TIME
ZONE,id_transaction_virtuelle TEXT,id_transaction BIGINT,severite_erreur TEXT,code_etat_sql
TEXT,message TEXT,detail TEXT,hint TEXT,requete_interne TEXT,pos_requete_interne INTEGER,
contexte TEXT,requete TEXT,pos_requete INTEGER,localisation TEXT,nom_application TEXT,
PRIMARY KEY (id_session, num_ligne_session));
```

```
COPY    log_postgresql_a_analyser
FROM    'C:/Users/jmsou/log_postgresql1.csv' WITH csv;
```

```
WITH    recup_info_log(utilisateur,base_de_donnees,requete,duree_ms) AS (
SELECT  utilisateur,base_de_donnees,
        SUBSTR(message,POSITION('ms, instruction : ' IN message)+18) AS requete,
        CAST(REPLACE(SUBSTR(message,9,POSITION (' ms,' IN message) -9),'.','') AS BIGINT)
FROM    log_postgresql_a_analyser
WHERE   indicatif_commande='SELECT')
SELECT  base_de_donnees,utilisateur,requete,
        SUM(duree_ms) AS duree_cumulee_exec_requete,
        COUNT(*) AS frequence_exec_requete
 FROM   recup_info_log
 GROUP  BY base_de_donnees,utilisateur,requete
 HAVING SUM(duree_ms) != 0 ORDER BY base_de_donnees,SUM(duree_ms) DESC;
```

base_de_donnees text	utilisateur text	requete text	duree_cumulee_exec_requete numeric	frequence_exec_requete bigint
postgres	postgres	SELECT proname	32000	1
seminaire	postgres	WITH RECURSIVE	2919000	4

Recherche de mots clés dans un forum

Pour effectuer des recherches sur les mots clés de messages d'un forum, on peut procéder ainsi :

WITH RECURSIVITE

```
WITH message (id_message,message,liste_mot_cle)
AS   (VALUES (1,'corps mess1','sql,déclencheur,table'),
             (2,'corps mess2','sql,performance'),
             (3,'corps mess3','déclancheur'),
             (4,'corps mess4','vue,table')),
     recherche (id_recherche,liste_recherche)
AS   (VALUES (1,'sql,performance'),(2,'déclencheur'),(3,'table,performance,vue')),
     decomposition_message (id_message,message,liste_mot_cle,mot_cle)
AS   (SELECT id_message,message,liste_mot_cle,
            REGEXP_SPLIT_TO_TABLE(message.liste_mot_cle,',+') AS mot_cle
            -- Coupe les chaines de caractère en utilisant le séparateur virgule.
            -- Chaque mot est généré sur une ligne (fonction ensembliste)
      FROM   message),
     decomposition_recherche (id_recherche,liste_recherche,recherche)
AS   (SELECT id_recherche,liste_recherche,
             REGEXP_SPLIT_TO_TABLE(recherche.liste_recherche,',+') AS recherche
       FROM   recherche)
SELECT id_recherche,liste_recherche,id_message,message,liste_mot_cle,COUNT(*) AS score
FROM   decomposition_message
       JOIN decomposition_recherche ON mot_cle=recherche
GROUP  BY id_recherche,liste_recherche,id_message,message,liste_mot_cle
ORDER  BY 1,5 DESC;
```

id_recherche integer	liste_recherche text	id_message integer	message text	liste_mot_cle text	score bigint
1	sql,performance	2	corps mess2	sql,performance	2
1	sql,performance	1	corps mess1	sql,déclencheur,table	1
2	déclencheur	1	corps mess1	sql,déclencheur,table	1
3	table,performance,vue	4	corps mess4	vue,table	2
3	table,performance,vue	2	corps mess2	sql,performance	1
3	table,performance,vue	1	corps mess1	sql,déclencheur,table	1

Utilisation de WITH dans une fonction

Le cas d'utilisation consiste en la création d'une fonction SQL de génération de nom aléatoire en fonction d'une répartition stockée dans un dictionnaire. On teste ensuite la répartition aléatoire générée :

WITH SANS RECURSIVITE

```
CREATE  OR REPLACE FUNCTION genere_repart_nom() RETURNS TEXT AS $$
WITH    terme_dico (id_terme_dico,nom_terme_dico,popul,b_inf,b_sup) AS
(VALUES (1,'Martin' , 100000,     1, 100000),
        (2,'Bernard',  50000,100001, 150000),
        (3,'Dubois' ,  30000,150001, 180000),
        (4,'Thomas' ,  25000,180001, 205000),
        (5,'Robert'   ,22000,205001, 227000),
        (6,'Richard'  ,20000,227001, 247000)),
        borne_dico (borne_inf,borne_sup) AS (
SELECT  MIN(b_inf),MAX(b_sup) FROM terme_dico),
        generation_alea (valeur_alea) AS (
SELECT  TRUNC(RANDOM() * (borne_sup-borne_inf+1))+borne_inf
FROM    borne_dico)
SELECT CAST(nom_terme_dico AS TEXT)
FROM    terme_dico JOIN generation_alea ON valeur_alea>=b_inf
                                       AND valeur_alea<=b_sup;
$$      LANGUAGE SQL;

WITH    generation_nom (id_nom,nom) AS (
SELECT GENERATE_SERIES,genere_repart_nom() FROM GENERATE_SERIES(1,50000))
SELECT nom AS "Nom généré",COUNT(*) AS "Répartition générée" FROM generation_nom
GROUP  BY nom ORDER BY 2 DESC;
```

Nom généré text	Répartition générée bigint
Martin	20354
Bernard	10036
Dubois	6110
Thomas	5135
Robert	4433
Richard	3932

Charger une base de données

A partir de la fonction de génération créée précédemment, on va charger une base avec une combinaison d'ordre CREATE et d'un WITH, puis d'un ordre INSERT avec un WITH:

WITH SANS RECURSIVITE

```
CREATE  TABLE personne AS
WITH    liste_prenom (id_prenom,prenom) AS
(VALUES (1,'Enzo'),(2,'Lucas'),(3,'Mathis'),(4,'Nathan'),(5,'Thomas'),
        (6,'Lea'),(7,'Clara'),(8,'Manon'),(9,'Chloe'),(10,'Camille')),
        genere_alea (id_personne,nom_personne,id_prenom,date_naissance) AS
        (SELECT GENERATE_SERIES,genere_repart_nom(),TRUNC(RANDOM()*(10-1+1))+1,
                CAST(RANDOM()*365*70 AS INTEGER)+CAST('01/01/1940' AS DATE)
         FROM   GENERATE_SERIES(1,1000))
SELECT  id_personne,nom_personne,prenom,date_naissance
FROM    genere_alea JOIN liste_prenom ON genere_alea.id_prenom=liste_prenom.id_prenom;

INSERT  INTO personne
WITH    liste_prenom (id_prenom,prenom) AS
(VALUES (1,'Hugo'),(2,'Theo'),(3,'Tom'),(4,'Louis'),(5,'Raphael'),
        (6,'Ines'),(7,'Sarah'),(8,'Jade'),(9,'Lola'),(10,'Anais')),
        genere_alea (id_personne,nom_personne,id_prenom,date_naissance) AS
        (SELECT GENERATE_SERIES+1000,genere_repart_nom(),TRUNC(RANDOM()*(10-1+1))+1,
                CAST(RANDOM()*365*70 AS INTEGER)+CAST('01/01/1940' AS DATE)
         FROM   GENERATE_SERIES(1,1000))
SELECT  id_personne,nom_personne,prenom,date_naissance
FROM    genere_alea JOIN liste_prenom ON genere_alea.id_prenom=liste_prenom.id_prenom;

SELECT EXTRACT(CENTURY FROM date_naissance) AS "Siècle",COUNT(*) AS "Nb personnes"
FROM   personne
GROUP  BY EXTRACT(CENTURY FROM date_naissance)
ORDER  BY 1;
```

Siècle double precision	Nb personnes bigint
20	1744
21	256

id_personne integer	nom_personne text	prenom text	date_naissance date
1	Martin	Lucas	1958-03-13
2	Martin	Manon	1986-08-15
3	Dubois	Mathis	1988-01-06

Anonymiser des données personnelles

Le but ici est d'anonymiser des données personnelles :

WITH RECURSIVE

```
UPDATE  personne  SET nom_personne=genere_alea.nom_personne,
                      prenom=genere_alea.prenom,
                      date_naissance=genere_alea.date_naissance
FROM    (WITH     liste_prenom (id_prenom,prenom) AS (
          VALUES  (1,'Hugo'),(2,'Theo'),(3,'Tom'),(4,'Louis'),(5,'Raphael'),
                  (6,'Ines'),(7,'Sarah'),(8,'Jade'),(9,'Lola'),(10,'Anais')),
                  genere_alea (id_personne,nom_personne,id_prenom,date_naissance) AS
                  (SELECT GENERATE_SERIES,genere_repart_nom(),
                          TRUNC(RANDOM()*(10-1+1))+1,
                          CAST(RANDOM()*365*70 AS INTEGER)+CAST('01/01/1940' AS DATE)
                   FROM   GENERATE_SERIES(1,2000))
           SELECT  id_personne,nom_personne,prenom,date_naissance
           FROM    genere_alea JOIN liste_prenom
                   ON genere_alea.id_prenom=liste_prenom.id_prenom) genere_alea
WHERE   personne.id_personne=genere_alea.id_personne;

SELECT EXTRACT(CENTURY FROM date_naissance) AS "Siècle",COUNT(*) AS "Nb personnes"
FROM   personne
GROUP  BY EXTRACT(CENTURY FROM date_naissance) ORDER  BY 1;
```

Siècle double precision	Nb personnes bigint
20	1701
21	299

id_personne integer	nom_personr text	prenom text	date_naissan date
1	Bernard	Ines	1985-12-27
2	Martin	Raphael	1951-05-13
3	Dubois	Hugo	1947-08-08

Clause WITH avec récursivité

Introduction

Cas pratiques de clauses WITH sans récursivité

Cas pratiques de clauses WITH avec récursivité

Clauses WITH et fonctions « Window » combinées

Evolutions prévues

WITH RECURSIVITE

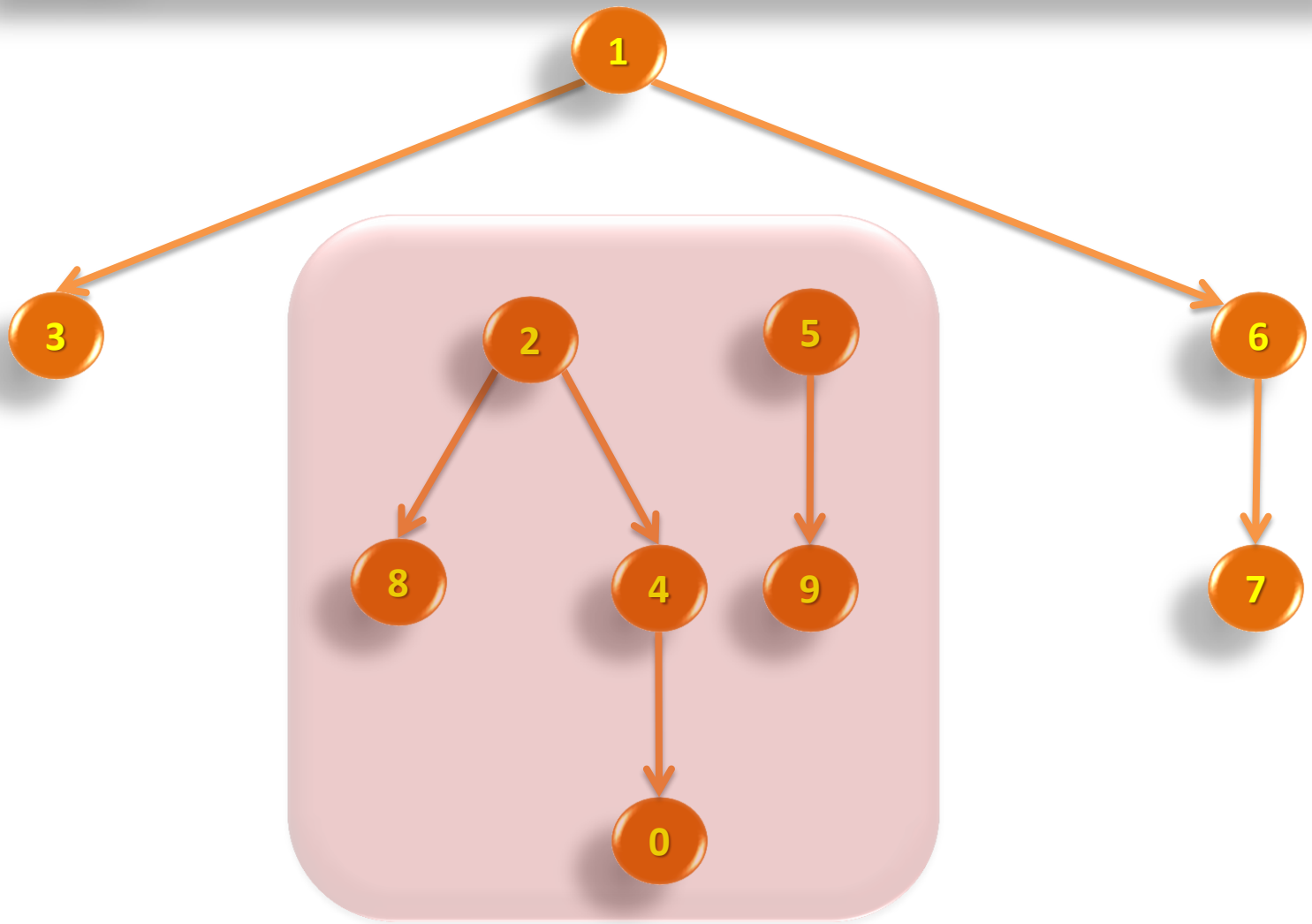
AVEC

- Recherche d'erreurs dans un arbre
- Calcul d'itinéraire avec Dijkstra
- Recherche dans un sociogramme
- Résolution de Sudoku
- Les tours de Hanoï

Recherche d'erreurs dans un arbre

La récursivité SQL est parfaitement adaptée au parcours d'arborescence. Il s'agit ici de mettre en évidence des portions d'arbres, orphelins ou avec des parents en dehors de l'arbre courant.

WITH RECURSIVITE



Recherche d'erreurs dans un arbre

La requête affiche tous les nœuds en erreur :

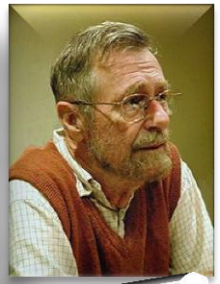
WITH RECURSIVE

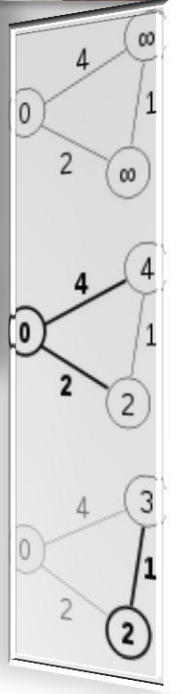
```
WITH   RECURSIVE racine_depart(id_noeud_racine) AS (VALUES (1)),
       hierarchie(id_noeud,lib_noeud,id_noeud_pere) AS (
VALUES (1,'Service1',NULL),(2,'Service2',NULL),(3,'Service3',1),(4,'Service4',2),
       (5,'Service5',99),(6,'Service6',1),(7,'service7',6),(8,'Service8',2),
       (9,'Service9',5),(0,'Service0',4)),
       parcours_hierarchie(id_noeud,lib_noeud,id_noeud_pere) AS (
SELECT id_noeud,lib_noeud,id_noeud_pere
FROM   hierarchie CROSS JOIN racine_depart WHERE  id_noeud =id_noeud_racine
UNION  ALL
SELECT hierarchie.id_noeud,hierarchie.lib_noeud,hierarchie.id_noeud_pere
FROM   hierarchie JOIN parcours_hierarchie ON
       hierarchie.id_noeud_pere=parcours_hierarchie.id_noeud),
       liste_erreur (id_noeud,lib_noeud,id_noeud_pere) AS (
SELECT id_noeud,lib_noeud,id_noeud_pere FROM hierarchie
EXCEPT
SELECT id_noeud,lib_noeud,id_noeud_pere FROM parcours_hierarchie),
       parcours_erreur (id_noeud,lib_noeud,id_noeud_pere,description,niveau,ancetre) AS (
SELECT id_noeud,lib_noeud,id_noeud_pere,
       CASE id_noeud_pere IS NULL WHEN TRUE THEN 'Arbre orphelin'
                                  ELSE 'Noeud ancêtre non existant' END,1,id_noeud
FROM   liste_erreur WHERE  id_noeud_pere IS NULL OR NOT EXISTS
       (SELECT * FROM liste_erreur lst_bis
        WHERE liste_erreur.id_noeud_pere=lst_bis.id_noeud)
UNION  ALL
SELECT liste_erreur.id_noeud,liste_erreur.lib_noeud,liste_erreur.id_noeud_pere,
       description,niveau+1,ancetre
FROM   liste_erreur JOIN parcours_erreur
                    ON parcours_erreur.id_noeud=liste_erreur.id_noeud_pere)
SELECT id_noeud AS "Id",LPAD(lib_noeud,8+3*(niveau-1),' ') AS "Désignation",
       id_noeud_pere AS "Id père",description AS "Motif rejet",niveau AS "Profondeur rejet"
FROM   parcours_erreur ORDER BY ancetre,niveau;
```

Id integer	Désignation text	Id père integer	Motif rejet text	Profondeur rejet integer
2	Service2		Arbre orphelin	1
8	Service8	2	Arbre orphelin	2
4	Service4	2	Arbre orphelin	2
0	Service0	4	Arbre orphelin	3
5	Service5	99	Noeud ancêtre non existant	1
9	Service9	5	Noeud ancêtre non existant	2

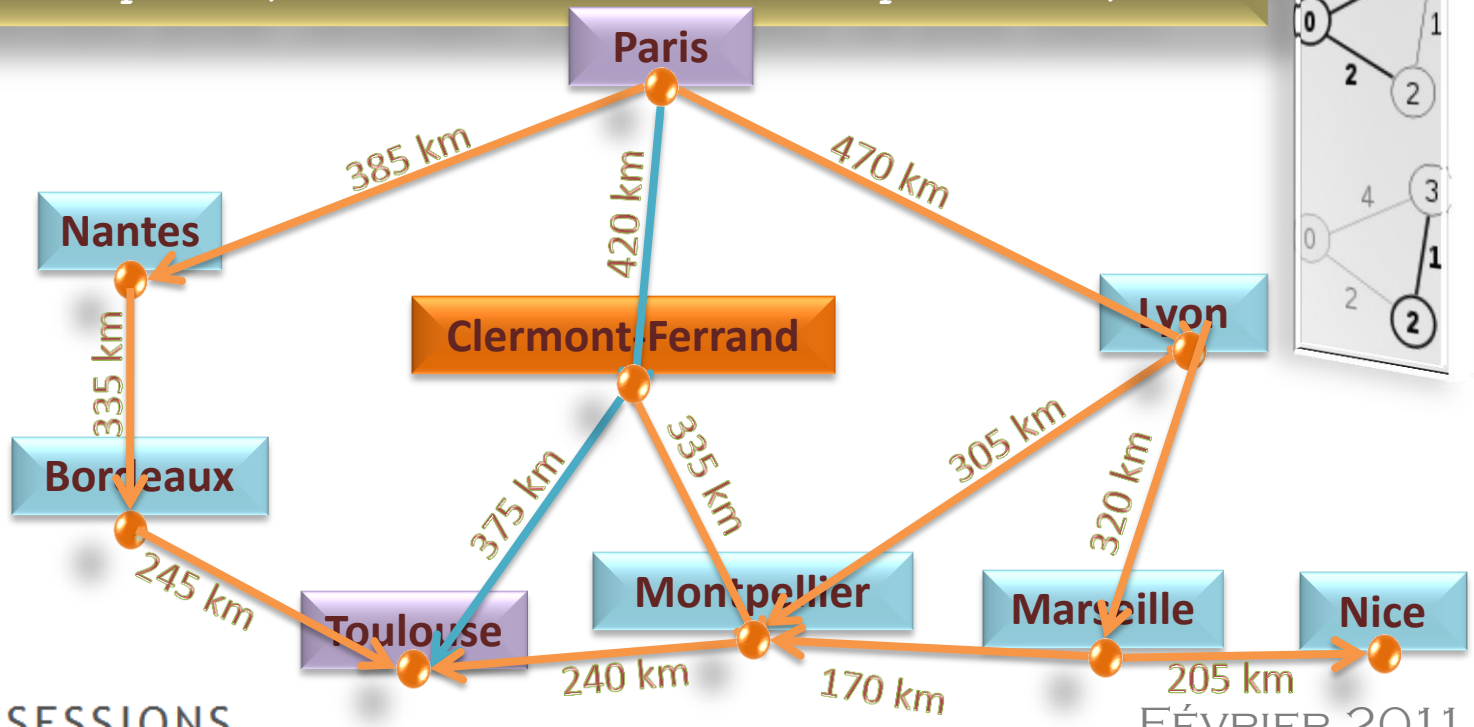
Calcul d'itinéraire avec Dijkstra

En théorie des graphes, l'algorithme de Dijkstra sert à résoudre le problème du plus court chemin. Il s'applique à un graphe connexe dont le poids lié aux arêtes est positif ou nul. L'algorithme porte le nom de son inventeur, l'informaticien néerlandais Edsger Dijkstra et a été publié en 1959. En 1972, il obtient le prix Turing (l'équivalent en informatique du prix Nobel). Le principe de l'algorithme est le suivant : Le poids du chemin entre deux sommets est la somme des poids des arêtes qui le composent. Pour une paire donnée de sommets sdeb (le sommet du départ) sfin (sommet d'arrivée) appartenant à S, l'algorithme trouve le chemin depuis sdeb vers sfin de moindre poids (autrement dit le chemin le plus court).





WITH AVEC RECURSIVITE



Calcul d'itinéraire avec Dijkstra

La requête recherche les plus courts chemins entre deux villes :

WITH RECURSIVE

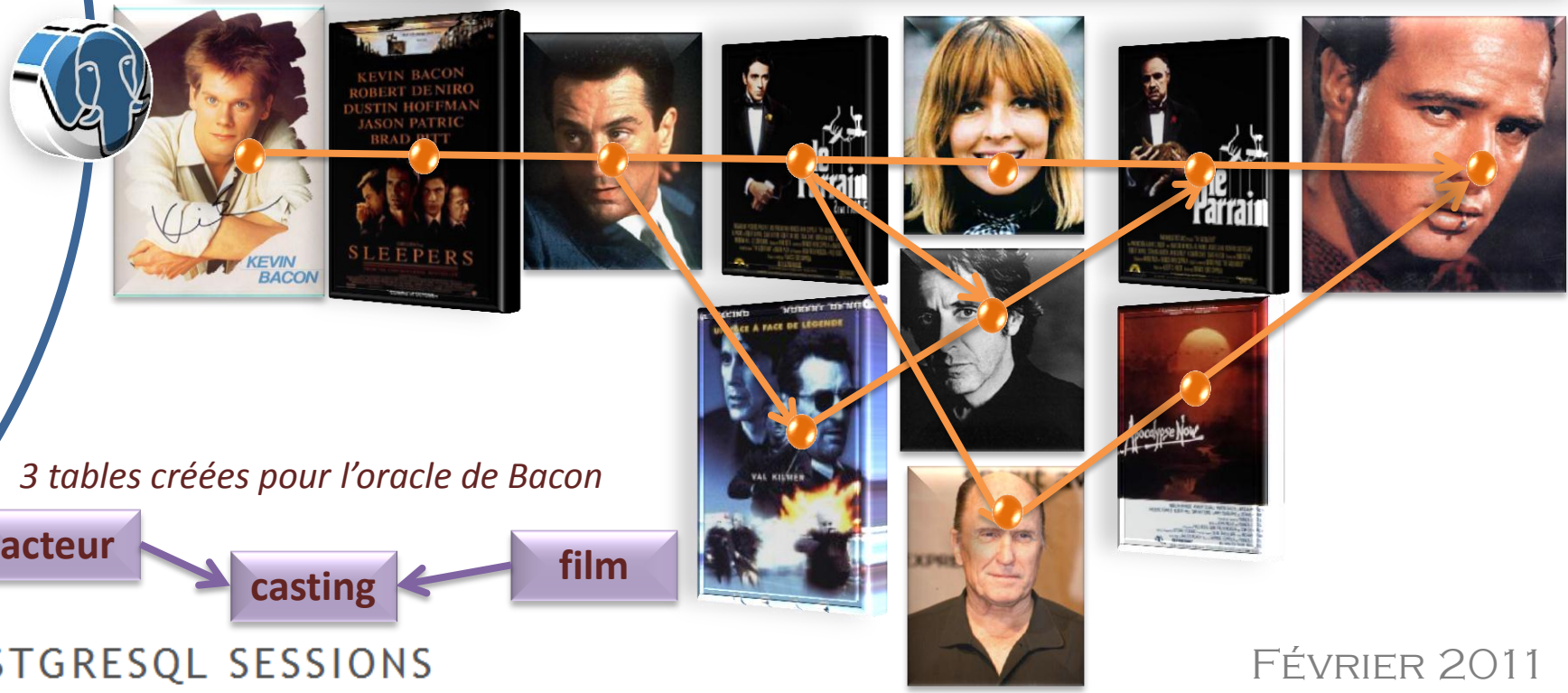
```
WITH   RECURSIVE  graphe_ville (ville_deb,ville_fin,cp_deb,cp_fin,distance) AS (
VALUES ('PARIS','NANTES','75000','44000',385),('PARIS','CLERMONT-FERRAND', '75000','63000',420),
       ('PARIS','LYON','75000','69000',470),('CLERMONT-FERRAND','MONTPELLIER','63000','34000',335),
       ('CLERMONT-FERRAND','TOULOUSE','63000','31000',375),('LYON','MONTPELLIER','69000','34000',305),
       ('LYON','MARSEILLE','69000','13000',320),('MONTPELLIER','TOULOUSE','34000','31000',240),
       ('MARSEILLE','NICE','13000','06000',205) ('NANTES','BORDEAUX','44000','33000',335),
       ('BORDEAUX','TOULOUSE','33000','31000',245),('MARSEILLE','MONTPELLIER','13000','34000',170)),
       voyage(ville_depart,cp_depart,ville_arrivee,cp_arrivee) AS (
VALUES ('PARIS','75000','TOULOUSE','31000')),
       graphe_croise (ville_deb,ville_fin,cp_deb,cp_fin,distance)  AS (
SELECT ville_deb,ville_fin,cp_deb,cp_fin,distance FROM graphe_ville
-- Pour avoir les distances entre 2 villes dans les 2 sens (sens graphe_ville et opposé)
UNION  ALL
SELECT ville_fin,ville_deb,cp_fin,cp_deb,distance FROM graphe_ville), --  Sens opposé
       dijkstra (ville_arrivee,cp_fin, nb_etapes, distance, etapes) AS (
       -- Implémentation de l'algorithme de Dijkstra
SELECT DISTINCT ville_deb,cp_deb, 0, 0, ville_deb||'('||cp_deb||')'
FROM   graphe_croise CROSS JOIN voyage
WHERE  ville_deb = ville_depart AND cp_deb=cp_depart
UNION  ALL
SELECT arrivee.ville_fin,arrivee.cp_fin, depart.nb_etapes + 1,
       depart.distance + arrivee.distance,
       depart.etapes||','|| arrivee.ville_fin||'('||arrivee.cp_fin||')'
FROM   graphe_croise AS arrivee INNER JOIN dijkstra AS depart
       ON depart.ville_arrivee = arrivee.ville_deb AND depart.cp_fin=arrivee.cp_deb
WHERE  depart.etapes NOT LIKE '%'||arrivee.ville_fin||'('||arrivee.cp_fin||')%')
       -- Evite les boucles : ne pas déjà avoir la ville dans la liste des étapes
SELECT  nb_etapes AS "Nb d'étapes",dijkstra.distance AS "Distance",
        etapes AS "Liste des étapes"
FROM    dijkstra CROSS JOIN voyage
WHERE   dijkstra.ville_arrivee = voyage.ville_arrivee AND cp_fin=voyage.cp_arrivee
ORDER   BY dijkstra.distance;
```

Nb d'étapes integer	Distance integer	Liste des étapes text
2	795	PARIS(75000),CLERMONT-FERRAND(63000),TOULOUSE(31000)
3	965	PARIS(75000),NANTES(44000),BORDEAUX(33000),TOULOUSE(31000)
3	995	PARIS(75000),CLERMONT-FERRAND(63000),MONTPELLIER(34000),TOULOUSE(31000)
3	1015	PARIS(75000),LYON(69000),MONTPELLIER(34000),TOULOUSE(31000)

Recherche dans un sociogramme

Les sociogrammes permettent de mettre en évidence les relations entre personnes. Voici un exemple avec l'oracle de Bacon : Connaissez-vous le paradoxe de Milgram, c'est l'hypothèse que chaque être humain est relié à une autre personne par une courte chaîne humaine. Appliquée au cinéma, cette théorie tiendrait à certains acteurs qui ont tourné dans beaucoup de films à travers le Monde et le temps, comme Bud Spencer ou Kevin Bacon. Ce dernier a d'ailleurs donné l'idée de créer un jeu : « six degrees ». Dans ce jeu, vous devez trouver un lien entre n'importe quel acteur et Kevin Bacon en passant par le moins de liens possibles. Le résultat donne le « Bacon Number » de cet acteur (oracleofbacon.org).

WITH AVEC RECURSIVITE

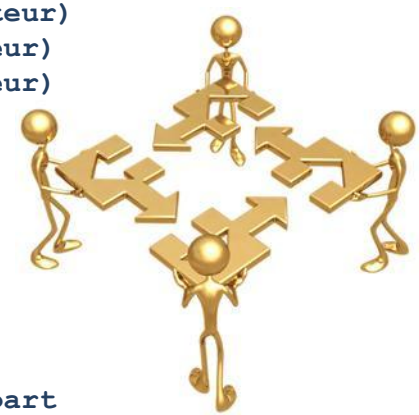


3 tables créées pour l'oracle de Bacon

Recherche dans un sociogramme

La requête recherche les plus courts chemins entre Kevin Bacon et Marlon Brando :

WITH AVEC RECURSIVITE



```
WITH    RECURSIVE meme_tournage
        (id_acteur_orig,nom_acteur_orig,id_acteur_dest,nom_acteur_dest,film) AS (
SELECT  casting1.id_acteur,acteur1.nom_acteur,casting2.id_acteur,
        acteur2.nom_acteur,titre||'('||annee||')'
FROM    casting casting1 JOIN casting casting2 ON (casting1.id_film=casting2.id_film
                        AND casting1.id_acteur != casting2.id_acteur)
        JOIN acteur acteur1 ON (casting1.id_acteur=acteur1.id_acteur)
        JOIN acteur acteur2 ON (casting2.id_acteur=acteur2.id_acteur)
        JOIN film ON (casting1.id_film=film.id_film)),
        distance_acteur (id_acteur_dest,distance,chemin)  AS (
SELECT DISTINCT id_acteur_orig,0,CAST('' AS TEXT)
FROM    meme_tournage WHERE nom_acteur_orig='Kevin Bacon'
UNION   ALL
SELECT  arrivee.id_acteur_dest,depart.distance + 1,
        depart.chemin||'['||arrivee.id_acteur_orig||']'||
        arrivee.nom_acteur_orig||'->'||film||'->'
FROM    meme_tournage AS arrivee INNER JOIN distance_acteur AS depart
                                 ON depart.id_acteur_dest = arrivee.id_acteur_orig
WHERE   depart.chemin NOT LIKE '%['||arrivee.id_acteur_dest||']%' AND distance <7),
        plus_courte_distance (distance) AS (
SELECT MIN(distance) FROM   distance_acteur WHERE  id_acteur_dest = 85),
        liste_distance (distance,chemin) AS (
SELECT distance_acteur.distance, chemin||'[85]Marlon Brando'
FROM    distance_acteur JOIN plus_courte_distance
                        ON distance_acteur.distance = plus_courte_distance.distance
WHERE   id_acteur_dest = 85)
SELECT CAST(distance AS TEXT) AS "Bacon number",chemin AS "Sociogramme"
FROM    liste distance;
```

Bacon number text	Sociogramme text
3	[46]Kevin Bacon->Sleepers(1996)->[3]Robert De Niro->Le Parrain II(1974)->[44]Robert Duvall->Le Parrain(1972)->[85]Marlon Brando
3	[46]Kevin Bacon->Sleepers(1996)->[3]Robert De Niro->Le Parrain II(1974)->[94]Diane Keaton->Le Parrain(1972)->[85]Marlon Brando
3	[46]Kevin Bacon->Sleepers(1996)->[3]Robert De Niro->Heat(1995)->[12]Al Pacino->Le Parrain(1972)->[85]Marlon Brando
3	[46]Kevin Bacon->Sleepers(1996)->[3]Robert De Niro->Le Parrain II(1974)->[12]Al Pacino->Le Parrain(1972)->[85]Marlon Brando
3	[46]Kevin Bacon->Sleepers(1996)->[3]Robert De Niro->Le Parrain II(1974)->[44]Robert Duvall->Apocalypse Now(1979)->[85]Marlon Brando

Résolution de Sudoku

WITH AVEC RECURSIVITE

La requête SQL suivant permet de résoudre des grilles de Sudoku. Le mot SuDoku signifie nombre (Su) unique (Doku) en japonais. Le principe semble en avoir été découvert en 1979 par Howard Games sous le nom de Number place. Les premières grilles ont été publiées dans la revue « Math, puzzles and logic problems » avec un modeste succès. Sudoku s'est fortement propagé depuis 1984 au Japon puis dans le monde en 2005.

En étudiant la requête, vous vous rendrez compte qu'ici la vision ensembliste couplée à la récursivité de la clause WITH font merveille en recherchant toutes les solutions et en ne retenant que la solution finale qui permet de répondre aux exigences des règles de remplissage : Le but du jeu est de remplir ces cases avec des chiffres allant de 1 à 9 en veillant toujours à ce qu'un même chiffre ne figure qu'une seule fois par colonne, une seule fois par ligne, et une seule fois par carré de neuf cases.

5	9	7	3	2	8	1	6	4
6	2	3	5	1	4	8	9	7
4	8	1	9	7	6	5	2	3
8	3	5	1	6	2	7	4	9
9	4	6	7	3	5	2	1	8
1	7	2	8	4	9	6	3	5
2	5	8	4	9	1	3	7	6
7	1	9	6	8	3	4	5	2
3	6	4	2	5	7	9	8	1

Résolution de Sudoku

La requête recherche toutes les solutions d'une grille de Sudoku :

WITH RECURSIVE

```
WITH   RECURSIVE probleme_sudoku (enonce) AS (
VALUES ('5  3 8    235 48  4 1  6   8      4   6   2   7      '||
        '5   4  3 6  96 345    2 7  1')),
       chiffres AS (SELECT  generate_series chiffre FROM generate_series(1,9)),
       resolution_sudoku(solution, nb_case_vide) AS (
SELECT enonce,POSITION(' ' IN enonce) FROM   probleme_sudoku
       -- Position dans la chaine de la sous-chaine
       UNION ALL
SELECT SUBSTR(solution,1,nb_case_vide-1)||CAST(chiffre AS TEXT)||
       SUBSTR(solution,nb_case_vide+1),
        -- Complète la grille avec le chiffre trouvé
        CASE POSITION( ' ' IN SUBSTR(solution,nb_case_vide+1))
             WHEN 0 THEN 0
             ELSE POSITION(' ' IN SUBSTR(solution,nb_case_vide+1))+nb_case_vide END
FROM   resolution_sudoku CROSS JOIN chiffres chiffres_sup
WHERE  nb_case_vide > 0
AND    NOT EXISTS
       (SELECT NULL FROM chiffres
        WHERE  CAST(chiffres_sup.chiffre AS TEXT)=
               SUBSTR(solution,CAST(TRUNC((nb_case_vide-1)/9)AS INTEGER)
                      *9+chiffres.chiffre,1)
        -- Pas de chiffre identique dans la ligne de la case vide en cours d'examen
        OR     CAST(chiffres_sup.chiffre AS TEXT)=
               SUBSTR(solution,MOD(nb_case_vide-1,9)-8+chiffres.chiffre*9,1)
        -- Pas de chiffre identique dans la colonne de la case vide en cours d'examen

        OR     CAST(chiffres_sup.chiffre AS TEXT)=
               SUBSTR(solution,MOD(CAST(TRUNC((nb_case_vide-1)/3) AS INTEGER),3)*3
               + CAST(TRUNC((nb_case_vide-1)/27)AS INTEGER)*27+chiffres.chiffre
               + CAST(TRUNC((chiffres.chiffre-1)/3)AS INTEGER)*6,1)))
        -- Pas de chiffre identique dans le carré 3x3 de la case vide en cours d'examen
SELECT  resolution_sudoku.solution
FROM  probleme_sudoku CROSS JOIN resolution_sudoku WHERE  nb_case_vide=0;
```

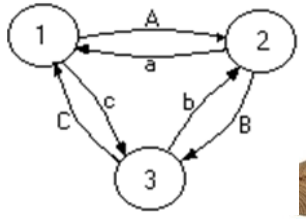
WITH AVEC RECURSIVITE

Les tours de Hanoï

WITH RECURSIVITE AVEC

Le problème des tours de Hanoï est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas en 1892. Le dispositif se compose de N disques troués de diamètres variables et de trois tiges sur lesquelles il est possible d´empiler les disques (cf. photo ci-jointe). Etant donné que ces disques sont empilés du plus grand au plus petit sur la tige de gauche, le but du jeu est de les empiler dans le même ordre sur une des deux tiges plus à droite. Le déplacement des disques est régi par trois règles : Vous ne pouvez déplacer qu´un seul disque à la fois ; Vous pouvez placer le disque sur la tige que vous voulez ; Vous ne pouvez pas mettre un disque sur un plus petit que lui.
Pour résoudre ce problème, un algorithme bien connu existe. Cependant pour implémenter cet algorithme, on se heurte à une limitation des requêtes SQL récursives (clause WITH RECURSIVE) : on ne peut utiliser qu'un niveau de récursivité. Or, le programme contient un double appel récursif avec des paramètres différents :

```
PROCEDURE hanoi (n,depart,arrivee,intermedaire)
DEBUT
   SI n>0 ALORS
          hanoi (n-1,depart,intermédiaire,arrivee)
          deplacer(depart,arrivee)
          hanoi (n-1,intermédiaire,arrivée,depart)
   FIN SI
FIN
```



Pour contourner cette limitation, on peut implémenter un algorithme itératif qui détermine le mouvement en fonction du numéro de mouvement où on se trouve(cf. graphique ci-contre pour le codage).

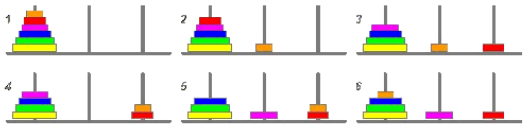
Les tours de Hanoï

L'amusant c'est que pour implémenter l'algorithme itératif en SQL, il faut introduire de la récursivité pour géré chaque mouvement …

WITH RECURSIVE AVEC RÉCURSIVITÉ

```
WITH    RECURSIVE nb_disque (nb_disque) AS (VALUES (4)),
        param_mvt (code_mouvement,designation_mouvement) AS (
VALUES ('A','Déplacer un disque de 1 vers 2'),('a','Déplacer un disque de 2 vers 1'),
       ('B','Déplacer un disque de 2 vers 3'),('b','Déplacer un disque de 3 vers 2'),
       ('C','Déplacer un disque de 3 vers 1'),('c','Déplacer un disque de 1 vers 3')),

       hanoi(nb_disque,num_mvt,cumul_mouvement) AS (
      -- Implémentation de l'algorithme de résolution des Tours de Hanoi en itératif
SELECT nb_disque,0,'A' FROM nb_disque
UNION  ALL
SELECT nb_disque,num_mvt+1,cumul_mouvement||
       CASE MOD((num_mvt+1),4)
       -- On déduit le type de mouvement en fonction du num ordre du mouvement
       WHEN 0 THEN CASE MOD((num_mvt+1),3)
                   WHEN 0 THEN 'A' WHEN 1 THEN 'C' ELSE 'B' END
                   WHEN 1 THEN CASE MOD((num_mvt+1),3)
                               WHEN 0 THEN 'a' WHEN 1 THEN 'c' ELSE 'b' END
                   WHEN 2 THEN CASE MOD((num_mvt+1),3)
                               WHEN 0 THEN 'A' WHEN 1 THEN 'C' ELSE 'B' END
                   WHEN 3 THEN CASE ASCII(SUBSTR(cumul_mouvement,
                                                 CAST((num_mvt+1-3)/4 AS INTEGER)+1,1))
                               BETWEEN 65 AND 67
                   -- Dépend du mouvement précédent : Code ascii de A, B et C
                                WHEN TRUE THEN CASE MOD((num_mvt+1),3)
                                               WHEN 0 THEN 'A' WHEN 1 THEN 'C' ELSE 'B' END
                                               ELSE CASE MOD((num_mvt+1),3)
                                               WHEN 0 THEN 'a' WHEN 1 THEN 'c' ELSE 'b' END
       END END
FROM   hanoi WHERE num_mvt+1<2^nb_disque-1)
SELECT num_mvt,param_mvt.code_mouvement,designation_mouvement
FROM   hanoi JOIN   param_mvt
       ON SUBSTR(cumul_mouvement,num_mvt+1,1)=param_mvt.code_mouvement;
```



Clauses WITH & Fonctions «Window»

Introduction

Cas pratiques de clauses WITH sans récursivité

Cas pratiques de clauses WITH avec récursivité

Clauses WITH et fonctions « Window » combinées

Evolutions prévues

CLAUSE WITH & « WINDOWS »

- Traitement de journaux serveurs
- Affichage du dictionnaire de données PostgreSQL
- Calcul du nième nombre premier

Traitement de journaux serveurs

CLAUSE WITH & « WINDOWS »

On souhaite des journaux serveurs qui indiquent l'heure du relevé et si le système est en marche ou arrêté. Il s'agit ensuite de traiter ces informations pour indiquer quelles sont les périodes de fonctionnement et les périodes d'arrêt. Si un relevé est en état de marche et que le suivant est en état d'arrêt, l'heure estimée de début d'arrêt correspond à la moitié de l'intervalle entre les deux relevés. La borne inférieure initiale est arrondie au début de l'heure. La borne supérieure finale est arrondie à la fin de l'heure.

Heure d'audit du journal	Marche ?
2010-01-01 23:10:00.000	Oui
2010-01-01 23:20:00.000	Oui
2010-01-01 23:30:00.000	Non
2010-01-01 23:40:00.000	Oui
2010-01-01 23:50:00.000	Non
2010-01-02 00:00:00.000	Non
2010-01-02 00:10:00.000	Oui

→ *La fonction « Window » **LAG** est utilisée pour comparer la valeur courante de marche du serveur par rapport à la valeur précédente*

Rupture
Non
Non
Oui
Oui
Oui
Non
Oui

→ *La récursivité est utilisée pour parcourir en récursivité une par une chaque ligne et faire +1 sur la colonne plage s'il y a rupture et garder la valeur sinon.*

Plage
1
1
2
3
4
4
5

Le résultat attendu :

Début plage estimée timestamp without time zone	Fin plage estimée timestamp without time zone	Statut serveur text
2010-01-01 23:00:00	2010-01-01 23:25:00	MARCHE
2010-01-01 23:25:00	2010-01-01 23:35:00	ARRET
2010-01-01 23:35:00	2010-01-01 23:45:00	MARCHE
2010-01-01 23:45:00	2010-01-02 00:05:00	ARRET
2010-01-02 00:05:00	2010-01-02 01:00:00	MARCHE

Traitement de journaux serveurs

La requête récursive affiche les résultats attendus en utilisant les fonctions « Window » LAG, LEAD (comparaison de la valeur courante par rapport à la valeur suivante : inverse de LAG) et RANK.

CLAUSE WITH & « WINDOWS »

```
WITH   RECURSIVE journal (heure_audit,marche) AS (
VALUES (CAST('2010-01-01 23:10:00.000' AS TIMESTAMP),TRUE),(CAST('2010-01-01 23:20:00.000' AS TIMESTAMP),TRUE),
       (CAST('2010-01-01 23:30:00.000' AS TIMESTAMP),FALSE),(CAST('2010-01-01 23:40:00.000' AS TIMESTAMP),TRUE),
       (CAST('2010-01-01 23:50:00.000' AS TIMESTAMP),FALSE),(CAST('2010-01-02 00:00:00.000' AS TIMESTAMP),FALSE),
       (CAST('2010-01-02 00:10:00.000' AS TIMESTAMP),TRUE)),
       rupture_journal (rupture,ordre,heure_audit,marche) AS (
SELECT COALESCE((LAG(marche) OVER w_tri_heure)!=marche,TRUE) AS rupture,
       RANK() OVER w_tri_heure AS ordre,heure_audit,marche
FROM   journal
WINDOW w_tri_heure AS (ORDER BY heure_audit)),
       recursif_plage_rupture(rupture,ordre,plage,heure_audit,marche) AS (
SELECT rupture,ordre,1,heure_audit,marche FROM rupture_journal WHERE ordre=1
UNION  ALL
SELECT rupture_journal.rupture,rupture_journal.ordre,
       CASE rupture_journal.rupture WHEN TRUE THEN plage+1 ELSE plage END AS plage,
       rupture_journal.heure_audit,rupture_journal.marche
FROM   rupture_journal JOIN recursif_plage_rupture
       ON rupture_journal.ordre=recursif_plage_rupture.ordre+1),
       plage_reelle (plage,marche,heure_min,heure_max) AS (
SELECT plage,marche,MIN(heure_audit),MAX(heure_audit) FROM recursif_plage_rupture
GROUP  BY plage,marche ORDER  BY plage)
SELECT COALESCE(DATE_TRUNC('second',((heure_min-LAG(heure_max) OVER w_tri_plage)/2)+
       LAG(heure_max) OVER w_tri_plage),CAST(SUBSTR(TO_CHAR(heure_min,'YYYY-MM-DD hh24:mm:ss'),
       1,14)||'00:00' AS TIMESTAMP)) AS "Début plage estimée",
       COALESCE(DATE_TRUNC('second',(LEAD(heure_min) OVER w_tri_plage -heure_max)/2+heure_max),
       CASE SUBSTR(TO_CHAR(heure_max,'YYYY-MM-DD hh24:mm:ss'),15,5)
            WHEN '00:00' THEN heure_max
            ELSE CAST(SUBSTR(TO_CHAR(heure_max,'YYYY-MM-DD hh24:mm:ss'),1,14)||
                      '00:00' AS TIMESTAMP)+interval '1 hour' END) AS "Fin plage estimée",
       CASE marche WHEN TRUE THEN 'MARCHE' ELSE 'ARRET' END AS "Statut serveur"
FROM   plage_reelle WINDOW w_tri_plage AS (ORDER BY plage);
```

Affichage du dictionnaire PostgreSQL

La requête affiche les informations sur les données définies dans les tables en parcourant les tableaux d'index et de contraintes de manière récursive pour mettre en évidence les index et les contraintes. Pour se faire, les fonctions « Window » MAX et DENSE_RANK sont utilisées. La requête tient sur 3 pages.

CLAUSE WITH & « WINDOWS »

Récupération des contraintes → Récupération des index → Eclatement des tableaux de ces 2 types d'objets

```
WITH   RECURSIVE index_contrainte
       (id_objet,tableau_col,id_table,type_obj,indice_debut,id_ftable,code_obj) AS (
SELECT oid,conkey,conrelid,contype,1,confrelid,consrc FROM pg_constraint
UNION

SELECT indexrelid,indkey,indrelid,'i',0,NULL,CASE WHEN indisunique THEN '(Un.)' ELSE ''
END||CASE WHEN indisclustered THEN '(Table org. index)' ELSE '' END
FROM   pg_index WHERE  indisvalid),
       eclate_colonne_obj (id_objet,tableau_col,num_col_obj,limite_nb_col_obj,
                  position_col_obj,id_table,type_obj,indice_debut,complement_objet) AS (
SELECT id_objet,tableau_col,tableau_col[indice_debut],
       MAX(array_length(tableau_col,1)) OVER (),1,id_table,
       CASE type_obj WHEN 'u' THEN 'UN' WHEN 'p' THEN 'PK'
                     WHEN 'f' THEN 'FK' WHEN 'i' THEN 'IX' ELSE 'CK' END,indice_debut,
       CASE type_obj WHEN 'f' THEN '('||(SELECT relname FROM pg_class
                                         WHERE  pg_class.oid=id_ftable)||')'
                     WHEN 'c' THEN code_obj WHEN 'i' THEN code_obj ELSE '' END
FROM   index_contrainte
UNION  ALL
SELECT id_objet,tableau_col,tableau_col[position_col_obj+indice_debut],limite_nb_col_obj,
       position_col_obj+1,id_table,type_obj,indice_debut,complement_objet
FROM   eclate_colonne_obj WHERE  position_col_obj+1 <=limite_nb_col_obj),
```

Affichage du dictionnaire PostgreSQL

Suite de la requête d'interrogation du dictionnaire…

CLAUSE WITH & « WINDOWS »

Récupération des rangs des colonnes par type d'objet

Concaténation par colonne de ses informations

```
        rang_col_obj (id_objet,id_table,num_col_obj,resultat_affiche,
                      rang_col_obj,type_objet) AS (
SELECT  id_objet,id_table,num_col_obj,type_obj||DENSE_RANK() OVER w_table_typobj||
        complement_objet||'.pos'||position_col_obj AS resultat_affiche,
        DENSE_RANK() OVER w_table_numcol,type_obj
FROM    eclate_colonne_obj WHERE  num_col_obj IS NOT NULL AND num_col_obj>=0
WINDOW  w_table_typobj AS (PARTITION BY id_table,type_obj ORDER BY id_table,type_obj,id_objet),
        w_table_numcol AS (PARTITION BY id_table,num_col_obj
                           ORDER BY id_table,num_col_obj,id_objet)),
        max_rang_col_obj (max_rang) AS (SELECT MAX(rang_col_obj) FROM rang_col_obj),
        concatene_result_col(id_table,num_col_obj,resultat_affiche,num_cour) AS (
SELECT  id_table,num_col_obj,resultat_affiche,1 FROM rang_col_obj WHERE  rang_col_obj=1
UNION   ALL
SELECT  concatene_result_col.id_table,concatene_result_col.num_col_obj,
        concatene_result_col.resultat_affiche||' '||rang_col_obj.resultat_affiche,
        concatene_result_col.num_cour+1
FROM    concatene_result_col CROSS JOIN max_rang_col_obj
JOIN    rang_col_obj ON concatene_result_col.id_table=rang_col_obj.id_table
AND     concatene_result_col.num_col_obj=rang_col_obj.num_col_obj
AND     concatene_result_col.num_cour+1=rang_col_obj.rang_col_obj
WHERE   num_cour<=max_rang),
        affichage_final_col_obj(id_table,num_col_obj,resultat_affiche) AS (
SELECT  id_table,num_col_obj,MAX(resultat_affiche)
FROM    concatene_result_col GROUP BY id_table,num_col_obj)
```

Affichage du dictionnaire PostgreSQL

Fin de la requête… : l'affichage final

Affichage final en essayant de récupérer les commentaires sur les tables et les colonnes (jointures externes)

```
SELECT nspname AS "Schéma",relname AS "Table",desc_tab.description AS "Desc Table",
       CASE WHEN relhasindex THEN 'Oui' ELSE NULL END AS "Idx Tab?",
       reltuples AS "Nb lignes",relnatts AS "Nb col.",attname AS "Colonne",
       attnum AS "No col",desc_col.description AS "Desc Col",pg_type.typname AS "Type col",
       CASE attlen WHEN -1 THEN CASE atttypmod WHEN -1 THEN NULL ELSE atttypmod END
                   ELSE attlen END AS "Lg phys. col.",
       CASE  WHEN attnotnull THEN 'Oui' ELSE NULL END AS "Oblig.?",
       resultat_affiche AS "Compléments (index & contraintes)"
FROM   pg_attribute JOIN pg_class ON attrelid=pg_class.oid
                    JOIN pg_namespace ON pg_class.relnamespace = pg_namespace.oid
                    JOIN pg_type ON pg_attribute.atttypid=pg_type.oid
         LEFT OUTER JOIN pg_description desc_tab
                      ON desc_tab.objoid=attrelid AND desc_tab.objsubid=0
         LEFT OUTER JOIN pg_description desc_col
                      ON desc_col.objoid=attrelid AND desc_col.objsubid=attnum
         LEFT OUTER JOIN affichage_final_col_obj ON attrelid=id_table AND attnum=num_col_obj
WHERE  nspname =('public') AND attnum>=1 AND relam=0 AND NOT relistemp AND relkind='r'
ORDER  BY nspname,relname,attnum;
```

Schéma name	Table name	Desc Table text	Idx Tab? text	Nb lignes real	Nb col. smallint	Colonne name	No col smallint	Desc Col text	Type col name	Lg phys. col. integer	Oblig.? text	Compléments (index & contraintes) text
public	acteur		Oui	88	2	id acteur	1		int4	4	Oui	IX1(Un.).pos1 PK1.pos1
public	acteur		Oui	88	2	nom acteur	2		text			
public	casting		Oui	159	2	id film	1		int4	4	Oui	IX1(Un.).pos1 PK1.pos1 FK2(film).pos1
public	casting		Oui	159	2	id acteur	2		int4	4	Oui	IX1(Un.).pos2 PK1.pos2 FK1(acteur).pos1
public	film		Oui	78	3	id film	1		int4	4	Oui	IX1(Un.).pos1 PK1.pos1
public	film		Oui	78	3	titre	2		text			
public	film		Oui	78	3	annee	3		int4	4		

C & « WINDOWS WITH CLAUSE »

Calcul du nième nombre premier

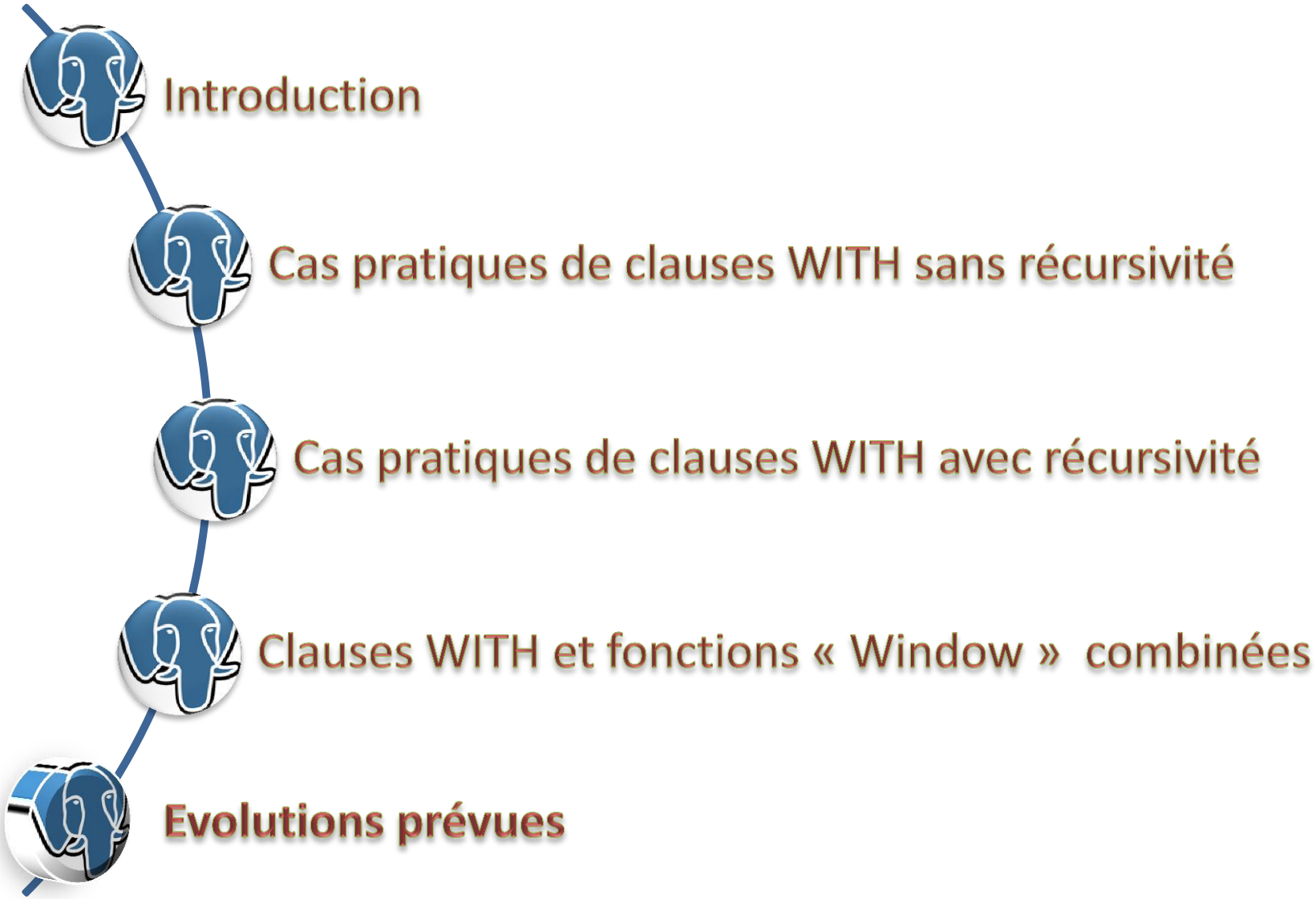
La requête utilise les fonctionnalités de la clause WITH et des fonctions « WINDOW » COUNT et RANK.

Génération et criblage de nombres entiers → On élimine ensuite les entiers non premiers → On calcule le rang de chaque nombre premier

C L A U S E W I T H & « W I N D O W S »

```
WITH    parametre (limite,nieme) AS (VALUES (CAST(10000 AS BIGINT),25)),
        genere_nombre (nombre_genere) AS
(SELECT GENERATE_SERIES FROM GENERATE_SERIES(1,(SELECT limite FROM parametre))),
        premiers_nombres_premiers (premier) AS
(VALUES (CAST(2 AS BIGINT)),(3),(5),(7),(11),(13),(17),(19),(23),(29),
             (31),(37),(41),(43),(47),(53),(59),(61),(67),(71)),
        divise (diviseur) AS
        -- Utilise le principe du crible d'Eratostène pour maximiser les recherches
        -- en supprimant les multiples des 20 premiers nombres premiers
(SELECT premier FROM premiers_nombres_premiers
 UNION
(SELECT nombre_genere FROM genere_nombre
 EXCEPT
 SELECT nombre_genere*premier
 FROM   genere_nombre CROSS JOIN premiers_nombres_premiers
 WHERE  nombre_genere<=(SELECT limite FROM parametre)/premier) ORDER  BY 1),
        genere_nombre_premier(nombre,premier) AS
(SELECT scan_nombre.nombre_genere AS "Nombre premier",
        COUNT(*) OVER (PARTITION BY scan_nombre.nombre_genere
                       ORDER BY scan_nombre.nombre_genere)=1
 FROM   divise JOIN genere_nombre scan_nombre ON MOD(scan_nombre.nombre_genere, diviseur)=0
               AND diviseur <= CAST(SQRT(scan_nombre.nombre_genere) AS INTEGER)
 WHERE  scan_nombre.nombre_genere<=(SELECT limite FROM parametre)
    AND scan_nombre.nombre_genere>=2),
        ordre_nombre_premier (rang,nombre) AS
(SELECT RANK() OVER (ORDER BY nombre),nombre FROM genere_nombre_premier WHERE premier)
 SELECT nombre FROM ordre_nombre_premier WHERE rang=(SELECT nieme FROM parametre);
```

Evolutions prévues



EVOLUTIONS PREVUES

Evolutions sur les CTE en PostgreSQL 9.1

Evolutions sur les CTE en 9.1

EVOLUTIONS PREVUES

WITH [RECURSIVE]
<Nom sous requête1> [(*<colonne>* [*<type>*] [,…])] AS (

[SELECT | VALUES | **INSERT | UPDATE | DELETE [RETURNING]** …]
[UNION [ALL]

SELECT | **INSERT | UPDATE | DELETE [RETURNING] …)**

[, *<sous_requête_2>*

[, *<sous_requête_n >***]]**

SELECT | **INSERT | UPDATE | DELETE …**
;

Pour échanger & s'entrainer

La liste de diffusion PostgreSQL française	*pgsql-fr-generale@postgresql.org*
Le forum de PostgreSQL.fr	*forums.postgresql.fr*
Le forum SQL de Developpez.com	*sgbd.developpez.com/*
Des problèmes mathématiques	*projecteuler.net/*
Des challenges en SQL Server (TSQL), mais aisément adaptable à PostgreSQL	*beyondrelational.com/*
Echanger sur des problématiques SQL directement avec moi	*jm.souchard@gmail.com*

Merci !

Des questions ?