

SQL Avancé

Fabien Coelho

Mines Paris – PSL

Janvier 2023

Composé avec L^AT_EX, révision 4420

1 / 45

SQL Avancé

Fabien

VALUES

Séries

LATERAL

UPSERT

Groups

WINDOW

WITH

JSON

EXTENSION

Conclusion



<https://datacharmer.org/>

2 / 45

Valeurs directes

Construction d'une table au vol

- construire table constante au vol
- `SELECT ... UNION SELECT ... UNION ...`
- `VALUES (...), (...), ...`
utilisé pour `INSERT`, expressions `ANY/ALL...`

justifie le S !

```
SELECT 1 AS jour, mois.nom AS mois
FROM (VALUES
  ('juillet'), ('août'), ('septembre'),
  ('octobre'), ('novembre'), ('décembre'),
  ('...'))
  AS mois(nom);
```

jour	mois
1	juillet
1	août
1	septembre
1	octobre
1	novembre
1	décembre
1	...

3 / 45

Séries

Séquence énumérée

- combien de vendredi 13 au 20ème siècle ?
- quelles valeurs de clés primaires sont inutilisées ?

Génération d'une relation

`generate_series`

- génération d'une relation, début, fin et saut
- entiers ou `TIMESTAMPTZ` (bornes) et `INTERVAL`

```
SELECT COUNT(*) AS "Vendredi 13"
FROM generate_series(1901,2000) AS year
CROSS JOIN generate_series(1,12) AS month
WHERE EXTRACT(DOW FROM DATE (year || '-' || month || '-13')) = 5;
```

Vendredi 13
171

4 / 45

Requête à côté

LATERAL

UPSERT

INSERT ou UPDATE ou DELETE

Dans une jointure

- utiliser les valeurs des tuples à sa *gauche*
- nouvelle relation *fonction* de ce qui précède
- utile ? expressions, jointures, agrégations, fenêtres. . .

```
-- version avec LATERAL
SELECT i, j
FROM generate_series(0, 3) AS i
CROSS JOIN
  LATERAL generate_series(0, i) AS j;
```

i	j
0	0
1	0
1	1
2	0
2	1
2	2
3	0
3	1
3	2
3	3

Fusion de données

- insertion ou mise à jour si existe déjà
économise un test et sa latence
- différentes syntaxes selon les bases de données. . .
`INSERT ... ON CONFLICT ..., MERGE ..., ...`
- `INSERT ... ON CONFLICT DO NOTHING / UPDATE ...`
avec clause `WHERE`, données initiales `EXCLUDED`
- `MERGE INTO ... USING .. ON WHEN ...`
permet toutes les opérations

Exemple INSERT ... ON CONFLICT ...

Exemple MERGE

Données initiales

```
INSERT INTO Heroes(name) VALUES ('Calvin')
  ON CONFLICT (name) DO NOTHING;
INSERT INTO Heroes VALUES (2, 'Hobbes')
  ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name;
INSERT INTO Heroes VALUES (3, 'Susie')
  ON CONFLICT (name) DO NOTHING;
```

Données finales

id	name
1	Calvin
2	hbs

id	name
1	Calvin
2	Hobbes
3	Susie

Stock initial

nom	qt
Champagne	10
Coteau du Layon	12
Riesling	3

Variations

nom	qt
Champagne	-3
Riesling	-3
Sancerre	6

Stock final

nom	qt
Champagne	7
Coteau du Layon	12
Sancerre	6

```
MERGE INTO Stock AS s
USING LivraisonVente AS lv ON s.nom = lv.nom
WHEN NOT MATCHED AND lv.qt > 0 THEN
  INSERT VALUES(lv.nom, lv.qt)
WHEN MATCHED AND s.qt + lv.qt > 0 THEN
  UPDATE SET qt = s.qt + lv.qt
WHEN MATCHED THEN
  DELETE;
```

GROUPING SETS, CUBE, ROLLUP

Exemple GROUPING SETS

Aggrégations combinées

- aggrégation sur des sous-ensembles de GROUP BY GROUPING SETS liste de groupes de colonnes ROLLUP tous les préfixes de colonnes, y compris vide CUBE tous les sous-ensembles de colonnes
- équivalent à UNION, valeurs non gardées remplacées par NULL

```
CREATE TABLE PaysLanguePopulation(  
  id SERIAL PRIMARY KEY,  
  pays TEXT NOT NULL,  
  langue TEXT NOT NULL,  
  pop NUMERIC NOT NULL,  
  UNIQUE (pays, langue)  
);
```

pays	langue	pop
Allemagne	Allemand	81.5
Autriche	Allemand	8.7
Belgique	Allemand	0.1
Belgique	Français	4.1
Belgique	Néerlandais	7.0
France	Français	66.1
Italie	Italien	60.8
Pays-Bas	Néerlandais	16.9
Suisse	Allemand	5.2
Suisse	Français	1.7
Suisse	Italien	0.5

Cumuls par pays et par langues

Ou avec UNION

moins performant

```
SELECT pays, langue, SUM(pop) AS pop  
FROM PaysLanguePopulation  
GROUP BY GROUPING SETS  
  ((pays), (langue), ())  
ORDER BY pays, langue;
```

pays	langue	pop
Allemagne		81.5
Autriche		8.7
Belgique		11.2
France		66.1
Italie		60.8
Pays-Bas		16.9
Suisse		7.4
	Allemand	95.5
	Français	71.9
	Italien	61.3
	Néerlandais	23.9
		252.6

```
SELECT pays AS pays, NULL AS langue,  
  SUM(pop) AS pop  
FROM PaysLanguePopulation  
GROUP BY pays -- premier groupe  
UNION  
SELECT NULL, langue, SUM(pop)  
FROM PaysLanguePopulation  
GROUP BY langue -- second groupe  
UNION  
SELECT NULL, NULL, SUM(pop)  
FROM PaysLanguePopulation  
-- dernier groupe  
ORDER BY pays, langue;
```

pays	langue	pop
Allemagne		81.5
Autriche		8.7
Belgique		11.2
France		66.1
Italie		60.8
Pays-Bas		16.9
Suisse		7.4
	Allemand	95.5
	Français	71.9
	Italien	61.3
	Néerlandais	23.9
		252.6

Exemple ROLLUP

```
SELECT pays, langue,
       SUM(pop) AS pop
FROM PaysLanguePopulation
GROUP BY ROLLUP(pays, langue)
-- équivalent à GROUPING SETS
-- ((pays, langue), (pays), ())
ORDER BY langue, pays
LIMIT 12;
```

pays	langue	pop
Allemagne	Allemand	81.5
Autriche	Allemand	8.7
Belgique	Allemand	0.1
Suisse	Allemand	5.2
Belgique	Français	4.1
France	Français	66.1
Suisse	Français	1.7
Italie	Italien	60.8
Suisse	Italien	0.5
Belgique	Néerlandais	7.0
Pays-Bas	Néerlandais	16.9
Allemagne		81.5

Exemple CUBE

```
SELECT pays, langue,
       SUM(pop) AS pop
FROM PaysLanguePopulation
GROUP BY CUBE(pays, langue)
-- équivalent à GROUPING SETS
-- ((pays, langue),
--    (pays), (langue), ())
ORDER BY pays, langue
LIMIT 13;
```

pays	langue	pop
Allemagne	Allemand	81.5
Allemagne		81.5
Autriche	Allemand	8.7
Autriche		8.7
Belgique	Allemand	0.1
Belgique	Français	4.1
Belgique	Néerlandais	7.0
Belgique		11.2
France	Français	66.1
France		66.1
Italie	Italien	60.8
Italie		60.8
Pays-Bas	Néerlandais	16.9

13 / 45

14 / 45

Fonctions de fenêtrage

window functions

Principe

- accès aux tuples voisins dans **SELECT**
un peu comme **GROUP BY**, mais *sans* le regroupement
- numérotation selon un tri, une partition, les deux...

Syntaxe

fonctions et clauses

aggrégation	usuelles	COUNT SUM...
fonctions	spécifiques	RANK LAG...
filtrage	clause	FILTER (WHERE ...)
fenêtre	clause	OVER (...)
nommage	pour réutilisation, clause	WINDOW ... AS (...)

Aggrégations partielles

FILTER

Syntaxe

- **AGG(...)** **FILTER** (WHERE ...)
- applique une aggrégation sur certains tuples seulement

```
-- nombre de films avant et après 1950
SELECT
  COUNT(*) FILTER(WHERE année < 1950) AS "avant",
  COUNT(*) FILTER(WHERE année >= 1950) AS "après",
  COUNT(*) AS "total"
FROM films;
```

avant	après	total
5	7	12

15 / 45

16 / 45

Numérotation selon un tri

Syntaxe

■ `RANK()` `OVER (ORDER BY ...)`

```
SELECT *,  
  RANK() OVER (ORDER BY val DESC)  
FROM Notes  
ORDER BY id;
```

id	val	rank
1	10	5
2	15	2
3	17	1
4	13	4
5	15	2

```
SELECT *,  
  RANK() OVER (ORDER BY val ASC) AS r1,  
  RANK() OVER (ORDER BY val ASC, id ASC) AS r2,  
  RANK() OVER (ORDER BY val DESC, id DESC)  
    AS r3  
FROM Notes  
ORDER BY id;
```

id	val	r1	r2	r3
1	10	1	1	5
2	15	3	3	3
3	17	5	5	1
4	13	2	2	4
5	15	3	4	2

17 / 45

Regroupement selon une partition

Syntaxe

- `AVG(...)` `OVER (PARTITION BY ...)`
- à comprendre comme un `GROUP BY`

```
SELECT eleve, cours, note,  
  -- moyenne par cours  
  AVG(note) OVER (PARTITION BY cours)  
    AS avc,  
  -- moyenne par élève  
  AVG(note) OVER (PARTITION BY eleve)  
    AS ave  
FROM Notations  
ORDER BY ave DESC, note DESC;
```

eleve	cours	note	avc	ave
Susie	Maths	19	12.25	18.5
Susie	Physics	18	13.25	18.5
Hobbes	Physics	19	13.25	17.5
Hobbes	Maths	16	12.25	17.5
Calvin	Physics	11	13.25	10.5
Calvin	Maths	10	12.25	10.5
Moe	Physics	5	13.25	4.5
Moe	Maths	4	12.25	4.5

18 / 45

Calculer la valeur médiane (le tuple médian)

délais d'attente

nom	délais
calvin	8
mum	13
dad	7
suzy	10
hobbes	123450

attente moyenne

round
24697.6

tuple médian

nom	délais
suzy	10

19 / 45

Aggrégations sur groupes ordonnés

Syntaxe

- `AGG(...)` `WITHIN GROUP (ORDER BY ...)`
- Fonctions `mode` `percentile_cont` `percentile_disc`

```
SELECT *  
FROM AttentePatients  
WHERE delais = (  
  -- calcule l'attente médiane  
  SELECT percentile_disc(0.5) WITHIN GROUP (ORDER BY delais)  
    FROM AttentePatients  
);
```

20 / 45

Générer une somme partielle

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

id	quand	montant	description	sum
1	2005-02-01 00:00:00	1000.00	versement initial	1000.00
2	2005-05-09 00:00:00	-100.00	tirage	900.00
3	2005-05-11 00:00:00	-450.00	sous-tirage	450.00
4	2006-01-01 00:00:00	200.00	ouf	650.00
5	2006-12-23 00:00:00	-700.00	joyeux noel	-50.00
6	2007-02-28 00:00:00	123.45	des sous !	73.45

21 / 45

Jointure sur opérateur $\leq$

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

Principe

- associe *tous* les précédents à chaque opération
- complexité en n^2 ... plus ou moins inutilisable

```
-- jointure speciale...  
SELECT cc.id, cc.quand, cc.montant, cc.description,  
       SUM(run.montant)  
FROM CompteCheque AS cc  
JOIN CompteCheque AS run ON run.id<=cc.id  
GROUP BY cc.id, cc.quand, cc.montant, cc.description  
ORDER BY cc.id ASC;
```

22 / 45

Avec une fenêtre

window

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

Syntaxe

- `AGG(...) OVER (ORDER BY ...)`

```
SELECT id, quand, montant, description,  
       SUM(montant) OVER (ORDER BY id ASC)  
FROM CompteCheque;
```

id	quand	montant	description	sum
1	2005-02-01 00:00:00	1000.00	versement initial	1000.00
2	2005-05-09 00:00:00	-100.00	tirage	900.00
3	2005-05-11 00:00:00	-450.00	sous-tirage	450.00
4	2006-01-01 00:00:00	200.00	ouf	650.00
5	2006-12-23 00:00:00	-700.00	joyeux noel	-50.00
6	2007-02-28 00:00:00	123.45	des sous !	73.45

23 / 45

Générer une moyenne mobile

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

Syntaxe

- `AGG(...) OVER (ORDER BY ... ROWS ...)`
- description de la fenêtre des valeurs à prendre
- ajustement automatique au début et à la fin

```
SELECT *,  
       AVG(qt) OVER (ORDER BY yr  
                    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)  
       AS ma  
FROM oilprod  
ORDER BY yr ASC;
```

yr	qt	ma
1980	3.1	3.6
1981	4.2	4.2
1982	5.3	5.2
1983	6.2	6.2
1984	7.1	7.1
1985	7.9	7.7
1986	8.1	8.1
1987	8.3	8.1
1988	7.8	7.9
1989	7.6	7.7

24 / 45

Requête *réursive* et fermeture transitive

Nommage d'une requête temporaire

WITH

- sorte de vue locale à une requête, de sous-requête
- calcul indépendant, stockage dans une table temporaire
- utilisation à la suite immédiate
- peut inclure **INSERT**, **UPDATE**, **DELETE**
- attention, barrière d'optimisation : implémentation matérialisée

Calcul itératif

WITH RECURSIVE

- calcul fermeture transitive : arrêt quand ajout vide

25 / 45

Fermeture transitive

WITH RECURSIVE

- table EnfantDe des ascendants directs
- calcul des ascendants et leur degré

- stockage table Ascendant

```
CREATE TABLE Ascendant(  
  enfant TEXT NOT NULL,  
  parent TEXT NOT NULL,  
  degre INTEGER NOT NULL,  
  PRIMARY KEY(enfant,parent)  
);
```

EnfantDe

enfant	parent
Calvin	Dad
Calvin	Mum
Granny	Great Granny
Mum	Grand Pa
Mum	Granny

27 / 45

Exemple moyenne inférieure à la moyenne

WITH

- réutilisation locale d'une sous-requête nommée...

```
WITH auteur_moy(auteur,moy) AS (  
  SELECT nom, AVG(durée)  
  FROM Films JOIN Personnes USING (pid)  
  GROUP BY nom)  
SELECT auteur, moy  
FROM auteur_moy  
WHERE moy < (SELECT AVG(moy) FROM auteur_moy)  
ORDER BY auteur;
```

auteur	moy
Allen	01:42:00
Ozu	01:37:00

26 / 45

Exemple

WITH RECURSIVE

```
WITH RECURSIVE Ascend(enfant,parent,degre)  
AS (  
  -- initialisation  
  SELECT enfant, parent, 1  
  FROM EnfantDe  
  UNION  
  -- itérations  
  SELECT ed.enfant, a.parent, 1+a.degre  
  FROM Ascend AS a  
  JOIN EnfantDe AS ed ON (ed.parent=a.enfant)  
)  
-- stocke le résultat dans Ascendant  
INSERT INTO Ascendant(enfant, parent, degre)  
SELECT enfant, parent, degre  
FROM Ascend;
```

enfant	parent	d
Calvin	Dad	1
Calvin	Mum	1
Granny	Great Granny	1
Mum	Grand Pa	1
Mum	Granny	1
Calvin	Grand Pa	2
Calvin	Granny	2
Mum	Great Granny	2
Calvin	Great Granny	3

28 / 45

Contraintes

WITH RECURSIVE

WITH avec INSERT UPDATE DELETE

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

- **WITH RECURSIVE** démarrage
- **SELECT** simple, contenu initial
- **UNION** avec l'incrément d'itération
- **SELECT** sur la table elle-même qui ne doit apparaître qu'une fois !
- calcul fermeture transitive : implémentation par une boucle sur une table temporaire

29 / 45

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

Retours

- retour des tuples ajoutés, modifiés ou effacés **RETURNING ...**

```
WITH archived AS (  
    DELETE FROM Invoice  
    WHERE paid AND sent < CURRENT_DATE - INTERVAL '1 month'  
    RETURNING *  
)  
INSERT INTO Archive  
SELECT * FROM archived;  
  
WITH changed AS (  
    UPDATE stuff  
    SET something = 'changed'  
    WHERE condition  
    RETURNING *  
)  
SELECT * FROM changed;
```

30 / 45

Plus longues séquences de températures croissantes

...

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

<http://tapoueh.org/>

- variations journalières
LAG(..., 1) **OVER (ORDER BY ...)**
- longueur des séquences croissantes
WITH RECURSIVE
- sélection des plus longues
MAX

date	temp
2021-01-01	-0.5
2021-01-02	0.9
2021-01-03	2.0
2021-01-04	4.4
2021-01-05	-0.5
2021-01-06	-1.0
2021-01-07	-1.1
2021-01-08	0.5
2021-01-09	-0.8
2021-01-10	1.8

31 / 45

SQL Avancé
Fabien
VALUES
Séries
LATERAL
UPSERT
Groups
WINDOW
WITH
JSON
EXTENSION
Conclusion

```
WITH RECURSIVE  
Delta AS ( -- day-on-day temperature delta  
    SELECT LAG(date, 1) OVER (ORDER BY date) AS dstart,  
           date AS dend,  
           temp - LAG(temp, 1) OVER (ORDER BY date) AS inc  
    FROM Temperature),  
RisingTemp AS ( -- increasing temperature sequences  
    SELECT dstart, dend, 1 AS cnt  
    FROM Delta  
    WHERE inc >= 0  
    UNION  
    SELECT p.dstart, d.dend, p.cnt + 1  
    FROM RisingTemp AS p  
    JOIN Delta AS d ON (p.dend = d.dstart)  
    WHERE d.inc >= 0)  
-- keep longest sequences found  
SELECT dstart, MAX(cnt) AS cnt  
FROM RisingTemp  
GROUP BY dstart  
ORDER BY cnt DESC, dstart ASC LIMIT 5;
```

32 / 45

Conclusion WITH et RECURSIVE

Options associées

- NOT MATERIALIZED vs MATERIALIZED précalcul ou non
- SEARCH BREADTH vs DEPTH ... ordre d'énumération
- CYCLE ... détection de cycles, construction de chemin

SQL est Turing complet

- <http://blog.coelho.net/tags.html#Turing-ref>
- WITH RECURSIVE + MAX(...) OVER + CROSS JOIN

33 / 45

Support JSON

JSON dans une base de donnée relationnelle

- représentation hiérarchique d'objects, listes, valeurs...
 - issue (natif) de JavaScript, RFC 8259
 - modèle *objet* des applications, très flexible, non relationnel
 - traités comme une valeur unique, indexable, manipulable
- compromis pour limiter la complexité du modèle relationnel
- intégré au standard SQL:2016 (ISO/IEC 9075:2016)
 - impl. partielles : IBM DB2, Oracle DB, Microsoft SQL Server, Postgres
- base de données *uniquement* JSON : MongoDB, Couchbase, CouchDB

```
{
  "personnage": "Calvin",
  "age": 6,
  "auteur": "Bill Watterson"
}
```

34 / 45

Support JSON

Implémentation Postgres

types JSON (texte), JSONB (binaire, pré-parsé)

opérateurs nombreux, quelques exemples

- J @> D : J contient D
- J -> 2, J -> 'a' accès indexé liste ou objets vers JSON
- J ? t : clé dans l'objet ?

fonctions nombreuses JSON_* JSONB_*, quelques exemples

- TO_JSON conversion vers JSON
- JSON_INSERT insertion dans une structure
- JSON_PRETTY prettyprinter

aggrégations JSON_AGG JSONB_AGG JSON_OBJECT_AGG JSONB_OBJECT_AGG

relations JSON_ARRAY_ELEMENTS JSON_EACH ...

JSON path langage d'extraction, filtrage pour JSON

35 / 45

Exemples JSON

tableau

```
SELECT i,
       JSON '["dim","lun","mar","mer","jeu","ven","sam"]' -> i AS jour
FROM generate_series(-1, 7) as i;
```

i	jour
-1	"sam"
0	"dim"
1	"lun"
2	"mar"
3	"mer"
4	"jeu"
5	"ven"
6	"sam"
7	

```
SELECT v FROM json_array_elements('["one", 2, true]') as v;
```

v
"one"
2
true

36 / 45

Exemples JSON

aggrégation

SQL Avancé

Fabien

Jour

jid	jnom
1	lundi
2	mardi
3	mercredi
4	jeudi
5	vendredi
6	samedi
7	dimanche

```
SELECT JSON_AGG(jnom) AS jours FROM Jour;
```

jours
["lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"]

37 / 45

Exemples JSON

objet

SQL Avancé

Fabien

```
CREATE TABLE IF NOT EXISTS Book(  
  title TEXT NOT NULL,  
  toc JSONB NOT NULL  
);  
  
INSERT INTO Book(title, toc) VALUES  
('The Christmas Murder Game',  
 '{ "title": "The Christmas Murder Game",  
   "contents": [  
     {"n": 1, "title": "Chapter One"},  
     {"n": 2, "title": "Chapter Two"},  
     {"n": 3, "title": "Chapter Three"},  
     {"n": 43, "title": "Chapter Forty-Three"}  
   ]  
 }')  
;
```

38 / 45

Exemples JSON

objet

SQL Avancé

Fabien

```
SELECT  
  title AS "titre",  
  -- titres des première et dernière parties du contenu  
  toc -> 'contents' -> 0 -> 'title' AS "début",  
  toc -> 'contents' -> -1 -> 'title' AS "fin"  
FROM Book;
```

titre	début	fin
Tintin et les Picaros	"Preface"	"Bibliography"
PostgreSQL Documentation	"Introduction"	"Révisions"
Systèmes d'information	"Chapter One"	"Chapter Forty-Three"

39 / 45

Extensions Postgres

SQL Avancé

Fabien

Extensions

- regroupement cohérent d'objets liés à une extension
fonctions, opérateurs, aggrégations, casts, tables...
- installation/désinstallation automatisée, gestion des versions...

```
CREATE EXTENSION intagg;
```

- **pgxn** : PostgreSQL eXtension Network
répertoire d'extensions, description standardisée, tags...

<https://pgxn.org>

40 / 45

Exemples d'extensions disponibles

pgmp entiers et rationnels
units gestion d'unités SI
semver gestion de versions
pgsphere coordonnées sphériques
safeupdate requiert une clause **WHERE** sur **DELETE UPDATE**
UPDATE Mesure **SET** valeur = 10.0; -- *Error!*
cron tâches périodiques dans Postgres
dirtyread accès aux tuples cachés par le MVCC
orafce compatibilité avec Oracle
tuplock verrouillage de tuples

POW(MPZ '2', 1024)
UNIT '3.6 km/hour'
SEMVER '1.2.3-beta'
SPOINT '(0d, 90d)'

Exemples d'utilisations d'extensions

Extension pgmp

- Nombre de Mersenne si $2^n - 1$ est premier

```
SELECT
  n AS "M",
  probab_prime((MPZ '1' << n) - 1, 10) AS prime
FROM generate_series(606, 609) AS n;
```

M	prime
606	0
607	1
608	0
609	0

Exemples d'utilisations d'extensions

Extension pg_sphere

- distance entre deux coordonnées sphériques <->

nom	coord (radians)
New-York	(4.99164166070378, 0.712094334813686)
Paris	(0.0401425727958696, 0.853466004225227)
Sydney	(2.63893782901543, -0.575958653158129)
Tokyo	(2.43997029428807, 0.623082542961976)

```
SELECT v1.nom, v2.nom,
  ROUND((v1.coord <-> v2.coord) * 6371) AS "km"
FROM Ville AS v1 CROSS JOIN Ville AS v2
WHERE v1.nom < v2.nom
ORDER BY 3 DESC, 1, 2;
```

nom	nom	km
Paris	Sydney	16896
New-York	Sydney	15950
New-York	Tokyo	10835
Paris	Tokyo	9713
Sydney	Tokyo	7731
New-York	Paris	5826

Conclusion

Autres sujets

- **PREPARE EXECUTE** préparation des requêtes répétées
- **INHERITS** héritage entre tables, relationnel-objet
- **FOREIGN DATA WRAPPER** accès à une autre DB
Oracle, MySQL, MS SQL Server... Amazon S3, LDAP, Twitter...

Bibliographie

- Documentation Postgres <https://www.postgresql.org/docs/current/>
- Planet PostgreSQL <https://planet.postgresql.org/>
- SQL Hacks, Tips & Tools for Digging into Your Data
Andrew Cumming & Gordon Russel, O'Reilly Hacks Series

SQL Avancé

Fabien Coelho

Mines Paris – PSL

Janvier 2023

Composé avec $\text{\LaTeX}$, révision 4420