



Optimisation de SQL

Modèle relationnel élégant... et efficace? – 15 ans de R&D!

Fabien Coelho – MINES ParisTech

Composé avec L^AT_EX, révision 4038

1

2

Optimisation : un problème global

interface client : GUI, web, mobile, ...

application charge, requêtes, latence vs débit

cache applicatifs, connexions...

base de données

!

réseau reliant les éléments

système d'exploitation, de fichier

matériel disques, bus, processeurs, mémoire...

Différents types de requêtes

transactionnel accès à quelques tuples seulement

- accès rapide : utilisation d'index, cache mémoire
- débit (tps) vs latence (ms/t) VISA 2000 tps, CB 400 tps

décisionnel calcul systématique de statistiques

- toutes les données parcourues!
- recherche du meilleur plan de calcul...
- **pas/peu** d'utilisation d'index
- méthodes en mémoire vs accès disque vs mixte...

Deux conseils préalables

3

4

1. Configurations logicielles et matérielles

Postgres configuration `postgresql.conf`

- mémoire : `shared_buffers` `effective_cache_size`...
- WAL, checkpoint, vacuum...
- paramètres de coûts : HDD vs SSD?

Système et matériel

- machine : virtuelle/réelle? partagée/dédiée? plusieurs? CPU?
- mémoire (cache OS) : disponible? extension?
- stockage HDD/SSD? lents/rapides? nombre? W cache? RAID?
- choix et configurations : OS, FS, montage...

5

6

2. Connaître ses données et sa charge

taille gros ou petit *vs mémoire disponible*

usage principal : transactionnel vs décisionnel

accès lectures (SELECT) vs écritures (INSERT, UPDATE, DELETE)

modèle grosses tables et jointures? *vs petites tables périphériques*

besoin latence (interactif) vs débit (batch)?

requêtes fréquences, variabilité, criticité? *requêtes lentes?*

charge dans le temps : min (0?), max, périodes critiques?

limite de performance : I/O, CPU, connexions...

stats cache hits? qu'espère-t-on? est-ce réaliste?

Comment ça marche ?

Planification d'une requête

plans d'évaluation d'une requête

requêtes algébriquement équivalentes

différentes opérations élémentaires (selon les index)

coût évaluation d'un plan

`EXPLAIN...`

statistiques sur les valeurs et tailles des données

`ANALYZE...`

comparaison avec exécution effective

`EXPLAIN ANALYZE...`

choix problème d'optimisation difficile

combinatoire ordre des jointures? placement des filtres?

approximation coûts non connus... éviter le pire

heuristiques, recuit simulé...

7

8

Opérations élémentaires d'une requête

- parcours+filtrage** d'une relation WHERE...
recherche des tuples satisfaisant une condition
- tri** d'une relation ORDER BY...
fixe l'ordre des tuples
- jointure+condition** de deux relations JOIN ON...
mise en correspondance de tuples selon une condition
- agrégation** d'une relation GROUP BY, DISTINCT, UNION...
regroupement des éléments selon un critère
- etc.

9

Coût des opérations élémentaires

- taille** des données : nombre de tuples, d'attributs, leurs tailles
- valeur** des données ! sélectivité d'une conditions (proba. d'être vrai)
implique souvent la taille des résultats !
- ```
WHERE id = 110; -- clef primaire (0-1)
WHERE id > 0; -- 100%
WHERE promo = 110; -- fraction des élèves...
WHERE age <> 110; -- 100%
WHERE année = 110; -- 0%
```
- ressources** disponibles : mémoire, charge machine...
- statistiques** min/max, moyenne, distribution avec ANALYZE

10

## Stockage des données : la hiérarchie mémoire

un processeur passe son temps à attendre les données !

| Type       | Taille       | Latence      | Débit max R   | €/TB   |
|------------|--------------|--------------|---------------|--------|
| registre   | $n$ Ko       | 1 ns         | 50 GB/s       | -      |
| SRAM cache | 256 Ko-4 Mo  | 10 ns        | 16 GB/s       | 100000 |
| DRAM       | 512 Mo-16 Go | 100 ns       | 6 GB/s        | 6250   |
| 3D Xpoint  | 100-500 Go   | 5-10 $\mu$ s | 1-9 GB/s      | 10000  |
| SSD 1      | 128 Go-2 To  | 100 $\mu$ s  | 50-150 MB/s   | 100    |
| HDD 1      | 500 Go-16 To | 10 ms        | 50-200 MB/s   | 30     |
| **D RAID 5 | -            | -            | 100-500 MB/s  | -      |
| réseau     | -            | 150 $\mu$ s  | 100-1000 MB/s | -      |

**Latence** : HDD/RAM =  $\times 100,000$  SSD/RAM =  $\times 1,000$

11

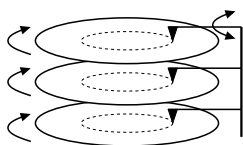


- IBM 350 Disk File**
- 1956
  - HDD du 305 RAMAC  
ram account. syst.
  - 24 disques
  - capacité 4.4 MB!  
5 M 7bit chars
  - location  
\$35,000 par an

12

## Technologie des disques magnétiques

- plateaux, faces, cylindres, secteurs, blocs...



- pérenne par rapport à la mémoire vive (panne courant)  
**attention** aux options FS, HDD et montage  
BBU vs *write cache* (RAID et \*\*D)
- entrées-sorties limitent les performances *I/O bound*  
si la base est petite, elle tient en mémoire !
- cache disque DB et système

13

## Mesure de coûts : I/O !

- unité d'accès** page PostgreSQL : 8 KB (Linux *ext4* : 4 KB)
- accès séquentiel HDD/SSD** typiquement X00 MB/s  
variation : bus IDE/SCSI, vitesse rotation, RAID, charge...
- accès aléatoire HDD** typiquement 1 à 10 MB/s  
latence disque : déplacement du bras (3-15 ms) et rotations  
un ordre de grandeur du séquentiel
- accès aléatoire SSD** 10-300 MB/s, X0 K IOPS  
typiquement 60-70% performance accès séquentiels
- calculs et accès mémoire** simplement négligés !  
très rapide par rapport aux accès disques

14

## Évolutions des *Solid-State Drive* : SSD

- mémoire** flash **permanente** (appareils photos, MP3)  
cache de disques magnétiques ? disques complets ?
- performances** des SSD : R/W, débit/latence, aléa/séq
- lecture  $\approx$  écriture (100-200 MB/s)  $\approx$  HDD
  - aléatoire  $\approx$  séquentiel  $\neq$  HDD
  - **usure** : si plein écriture aléatoire lente, nombre limité !  
cycle W page (4 KB) vs Erase block (512 KB)
- impact** coûts  $\times 8 - 15$ , capacité  $\times \frac{1}{4} - \frac{1}{8}$
- performances & optimisations ?
- vitesse par *tablespace*...



15

## Paramètres de modélisation I/O HDD

- $B$  taille d'un bloc de la base en octet (vs bloc FS)
- $k$  lecture séquentielle d'un bloc en seconde
- $K \gg k$  lecture aléatoire d'un bloc en seconde
- $n_R$  nombre de tuples d'une relation R
- $t_R$  taille en octets d'un tuple de R
- $S_R = n_R t_R$  taille de la relation
- $s_a$  taille en octet d'un attribut particulier
- $g$  probabilité de regroupement (coefficient d'agrégation)
- $j$  probabilité de jointure selon une condition...
- $M$  mémoire disponible (cache, données temporaires)

16

**Parcours+Filtrage : parcours séquentiel**

**sequential scan**

- lecture d'une relation sur disque
- filtrage des tuples au cours du parcours

```
SELECT * FROM oeuvres
WHERE titre LIKE 'A%';
```

```
seq scan on oeuvres
filter: titre LIKE 'A%'
```

$$k \frac{tn}{B}$$



**Parcours+Filtrage : parcours indexé**

**index scan**

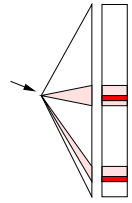
$$K + K \ln(ns/B)$$

accès *direct* à quelques tuples... mais coût de l'utilisation de l'index!

```
SELECT * FROM oeuvres
WHERE id BETWEEN 12 AND 14
AND titre LIKE 'A%';
```

```
index scan using oeuvres_pkey on oeuvres
index cond: id>=12 AND id<=14
filter: titre LIKE 'A%'
```

```
index scan using oeuvres_titre_idx on oeuvres
index cond: titre LIKE 'A%'
filter: id>=12 AND id<=14
```



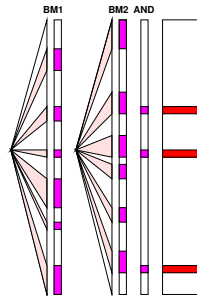
**Parcours+Filtrage(S) : index bitmap**

**bitmap index scan** création de bitmaps combinables à partir d'index

1 page = 1 bit, puis accès aux pages

```
SELECT * FROM oeuvres
WHERE oid < 100 AND cid < 10;
```

```
bitmap heap scan on oeuvres
bitmap and
 bitmap index scan on oeuvres_pk
 index cond: oid < 100
 bitmap index scan on oeuvre_cid
 index cond: cid < 10
```



**Indexation des attributs d'une relation principes simples, détails compliqués et subtils...**

retrouver des tuples rapidement selon le critère indexé

**Hash** sur disque

- permet uniquement des recherches = et <>
- 10-20% plus rapide de B-tree ?

**B-tree** (Balanced tree)

relation d'ordre!

- variantes : denses ? partiels ? unique ? multi-niveau ? équilibre ?
- égalités = <>, intervalles < <= > >=, extrêmes **MIN MAX**

**GIN** Generalized INverted index – tableau, store...

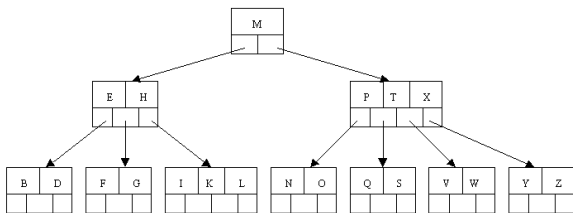
**GiST** Generalized Search Tree – full texte, géométrie

**Bloom** agrégation de valeurs de colonnes = **AND**

... SP-GiST, BRIN...

**B-tree**

- arbre *n*-aire, séparation sur valeurs limites (<)
- feuilles référencent les tuples ou les pages contenant les tuples



Dessin : Br. David Carlson and Br. Isidore Miner, Saint Vincent College

P?K?R?

**Création d'un index**

- unique (pour une clef) ou non
- précise la méthode d'indexation
- attributs/expressions indexés **ensembles** (ordre lexicographique)
- options diverses (NULLS, ...)
- création automatique : **UNIQUE**, e.g. clefs primaires

```
CREATE [UNIQUE] INDEX idxname
ON tablename USING [hash | btree | rtree | gist]
({ column | (expr) }[, ...]) [WHERE predicat];
```

**Importance des index**

- utilisation **éventuelle** par les requêtes! **SELECT**
- vérification des contraintes **INSERT UPDATE DELETE**  
existence de clefs étrangères, unicité d'une clef...
- forcer l'ordre de stockage **CLUSTER...**  
parcours séquentiel dans l'ordre d'un index particulier

**Coût des index**

- espace disque utilisé
- présence en mémoire
- maintenance à chaque opération **INSERT UPDATE DELETE**

**Index sur un attribut**

```
SELECT * FROM oeuvre
WHERE titre = 'Triple Jeu';
```

```
CREATE INDEX oeuvre_titre
ON oeuvre(titre);
```

## Index partiel, avec prédicats

- indexation partielle : taille et coût réduits
- restreint à une recherche avec ce prédicat
- utile pour contraintes partielles...  
*si telle condition, tel attribut est unique...*

```
SELECT * FROM answer
WHERE idUsr=12 AND isLast;
```

```
CREATE INDEX active_user_answer
ON answer (idUsr)
WHERE isLast;
```

25

## Index fonctionnels

- indexation de calculs sur les attributs
- restreint à une recherche avec cette fonction *immutable*

```
SELECT titre FROM oeuvre
WHERE UPPER(titre)='SLALOMS';
```

```
CREATE INDEX titre_upper
ON oeuvre
USING BTREE ((UPPER(titre)));
```

26

## Texte : ordre alphabétique selon la langue...

| posix |
|-------|
| Akin  |
| Akin  |
| côtée |
| côté  |
| grot  |
| groß  |
| ninon |
| niño  |

| français |
|----------|
| Akin     |
| Akin     |
| côté     |
| côtée    |
| groß     |
| grot     |
| niño     |
| ninon    |

27

## Index texte

- indexation de champs texte par défaut sur octets (plus rapide) mais ordre incompatible avec les caractères
- préciser une option, éventuellement la langue **COLLATE** indexation basée sur les caractères

```
SELECT * FROM auteur WHERE nom LIKE 'Début%';
```

```
CREATE INDEX auteur_nom_fr
ON auteur (nom text_pattern_ops);
```

```
CREATE INDEX auteur_nom_fr
ON auteur (nom COLLATE "fr_FR" text_pattern_ops);
```

28

## Index Bloom

<http://blog.coelho.net/database/2016/12/11/postgresql-bloom-index.html>

## Index R-Tree

## Index pour Full Text Search (FTS)

## Index sur trigrammes pgtrgm

29

## Tri d'une relation

- tri fusion *merge sort* de bloc triés
- espace mémoire et disque utilisable ?  
meilleures techniques avec parcours séquentiels...
- nombreuses variantes...  $kXn \ln(n)$
- demandée par l'utilisateur, ou techniques de calcul

**ORDER BY** évidemment !

**GROUP BY** et regroupement des données contiguës

**DISTINCT** similaire au précédent

**JOIN ON** jointure par fusion

30

## Jointures de relations

- relation (= ou autre...) entre attribut(s), clefs ou non...
- tables jointes de cardinalité  $n_R$  et  $n_P$ , résultat  $j n_R n_P$
- évaluation de  $j$  selon les données ?

| vnom          | dpt |
|---------------|-----|
| Avon          | 77  |
| Briare        | 45  |
| Cahors        | 46  |
| Figeac        | 46  |
| Fontainebleau | 77  |
| Gien          | 45  |
| Héricy        | 77  |
| Marseille     | 13  |

| dpt | dnom             |
|-----|------------------|
| 13  | Bouches du Rhône |
| 36  | Indre            |
| 45  | Loiret           |
| 46  | Lot              |
| 75  | Paris            |
| 73  | Savoie           |
| 77  | Seine et Marne   |

31

## Jointure par fusion triée

- tri des relations selon le critère de jointure !  
puis parcours parallèle et fusion au vol des relations  
 $\frac{k}{B}(t_R(X n_R \ln(n_R) + n_R) + t_P(Y n_P \ln(n_P) + n_P) + t_{RP} j n_R n_P)$
- écriture ou exploitation au vol du résultat ?

| vnom      | dpt |
|-----------|-----|
| Marseille | 13  |
| Briare    | 45  |
| Orléans   | 45  |
| Gien      | 45  |
| Cahors    | 46  |
| Figeac    | 46  |
| Paris     | 75  |
| Vulaines  | 77  |

| dpt | dnom             |
|-----|------------------|
| 13  | Bouches du Rhône |
| 36  | Indre            |
| 45  | Loiret           |
| 46  | Lot              |
| 73  | Savoie           |
| 75  | Paris            |
| 77  | Seine et Marne   |

32

## Jointure par hash

- relation mise en **mémoire**, indexée par critère de jointure puis parcours séquentiel de l'autre relation

$$\frac{k}{B}(t_{RN} + t_{PN} + t_{RP}j_{NR}n_P)$$

- la mémoire doit être suffisante... risque de *swap* ?

## Autres techniques de calculs de jointure

**boucles** différentes variantes, avec index...

**filtre** produit cartésien, puis filtrage...

33

## Agrégations

- regroupements si attributs égaux : **GROUP BY, DISTINCT**
- évaluation du taux de regroupement *g* ?
- parallélisation depuis postgres 9.6...

| Département      | Ville         |
|------------------|---------------|
| Seine et Marne   | Avon          |
| Loiret           | Briare        |
| Lot              | Cahors        |
| Lot              | Figeac        |
| Seine et Marne   | Fontainebleau |
| Loiret           | Gien          |
| Seine et Marne   | Héricy        |
| Bouches du Rhône | Marseille     |

34

## Agrégation par fusion triée

- tri de la relation selon le critère d'agrégation
- groupement ou agrégation des éléments contigus

$$\frac{k}{B}(Xn \ln(n) + tn + gtn)$$

| Département      | Ville     |
|------------------|-----------|
| Bouches du Rhône | Marseille |
| Loiret           | Briare    |
| Loiret           | Orléans   |
| Loiret           | Gien      |
| Lot              | Cahors    |
| Lot              | Figeac    |
| Paris            | Paris     |
| Seine et Marne   | Vulaines  |

| Département      | Nb villes |
|------------------|-----------|
| Bouches du Rhône | 1         |
| Loiret           | 3         |
| Lot              | 2         |
| Paris            | 1         |
| Seine et Marne   | 7         |

35

## Agrégation par hash

- allocation en mémoire du résultat *gtn*!
- remplissage lors du parcours, puis écriture ou exploitation

$$\frac{k}{B}(tn + gtn)$$

36

## Exemples de EXPLAIN et EXPLAIN ANALYZE

### EXPLAIN plan de la requête

- méthode de calcul
- temps (sans unité) premier résultat et fin
- nombreux paramètres ajustables
- volumes nb tuples et tailles

### EXPLAIN ANALYZE plan et exécution

- ajoute le réalisé
- détection des erreurs d'évaluation

interprétation <http://explain.depesz.com/> ou <https://explain.dalibo.com/>

37

### Select simple

```
SELECT titre FROM films WHERE fid=3;
```

explain

```
Seq Scan on films (cost=0.00..1.14 rows=1 width=12)
 Filter: (fid = 3)
```

explain analyze

```
Seq Scan on films (cost=0.00..1.14 rows=1 width=12)
 (actual time=0.013..0.015 rows=1 loops=1)
 Filter: (fid = 3)
 Rows Removed by Filter: 10
 Total runtime: 0.044 ms
```

38

### Select jointure

```
SELECT titre
FROM films JOIN personnes USING (pid)
WHERE nom='Chaplin';
```

explain

```
Hash Join (cost=1.09..2.26 rows=2 width=12)
 Hash Cond: (films.pid = personnes.pid)
-> Seq Scan on films (cost=0.00..1.11 rows=11 width=16)
-> Hash (cost=1.07..1.07 rows=1 width=4)
 -> Seq Scan on personnes (cost=0.00..1.07 rows=1 width=4)
 Filter: (nom = 'Chaplin'::text)
```

explain analyze

```
Hash Join (cost=1.09..2.26 rows=2 width=12)
 (actual time=0.051..0.124 rows=6 loops=1)
 Hash Cond: (films.pid = personnes.pid)
-> Seq Scan on films (cost=0.00..1.11 rows=11 width=16)
 (actual time=0.008..0.075 rows=11 loops=1)
-> Hash (cost=1.07..1.07 rows=1 width=4)
 (actual time=0.021..0.021 rows=1 loops=1)
 Buckets: 1024 Batches: 1 Memory Usage: 1kB
 -> Seq Scan on personnes (cost=0.00..1.07 rows=1 width=4)
 (actual time=0.010..0.016 rows=1 loops=1)
 Filter: (nom = 'Chaplin'::text)
 Rows Removed by Filter: 5
 Total runtime: 12.654 ms
```

39

40

**Analyse d'une requête pour proposer un INDEX**

1. à partir de EXPLAIN (ANALYSE)

index pour parcours séquentiels filtrés (coûteux) *Seq Scan Filter*  
 mais dépend du plan particulier choisit...

2. à partir de la requête

- partir des informations les plus sélectives WHERE en particulier sur les grosses tables
- vérifier que les index utiles existent
- propager le long des jointures
- vérifier que les index utiles existent

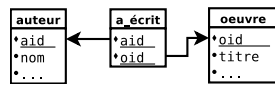
**Exemples de création d'index**

41

42

**Exemple de proposition d'index (1)**

```
SELECT titre
FROM auteur
JOIN a_écrit USING (aid)
JOIN oeuvre USING (oid)
WHERE nom = 'Goscinnny';
```

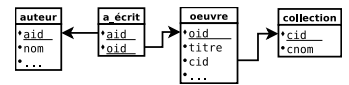


- nom → aid sur auteur **index auteur (nom)**
- aid → oid sur a\_écrit **index a\_écrit (aid)**  
*existe déjà, préfix clef composite*
- oid → titre sur oeuvre : **index oeuvre (oid)**  
*existe déjà, oid est clef primaire*

43

**Exemple de proposition d'index (2)**

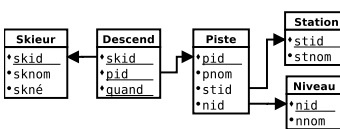
```
SELECT DISTINCT pseudo
FROM auteur
JOIN a_écrit USING (aid)
JOIN oeuvre USING (oid)
JOIN collection USING (cid)
WHERE cnom = 'Lapinot';
```



- cnom → cid sur collection **index collection (cnom)**  
*existe déjà, contrainte d'unicité*
- cid → oid sur oeuvre **index oeuvre (cid)**
- oid → aid sur a\_écrit **index a\_écrit (oid)**
- aid → pseudo sur auteur **index auteur (aid)**  
*existe déjà, clef primaire*

44

**TD : Trouvez les index !**



**Skieur** PK(skid) U(sknom,skid)  
**Descend** PK(skid,pid,quand)  
**Piste** PK(pid) U(pnom,stid)  
**Station** PK(stid) U(stnom)  
**Niveau** PK(nid) U(nnom)

- Les skieurs nés en avril 2005 ?
- Les plus jeunes skieurs ?
- Les skieurs ayant descendu des pistes en 2020 ?
- Les pistes de niveau extrême ?
- Les skieurs ayant descendu des pistes à Courchevel ?
- Les stations fermées en 2016 ?

45

**Collecte d'informations**

- niveaux à considérer : client, **base de données**, OS
- très nombreuses informations disponibles
- mais non historisées

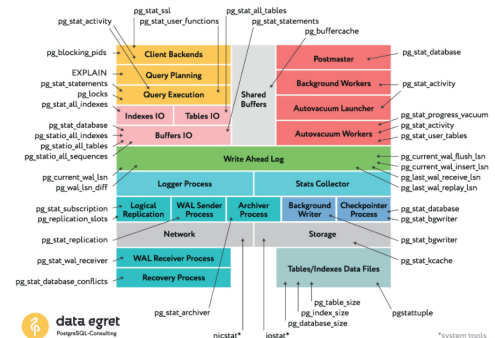
46

**Informations**

- volumes de données **pg\_\*\_size()**
- statistiques sur données pour optimisation **analyze**
- historique requêtes et performances **log**
- état courant (connexions, verrous, requêtes) **système**
- opérations tables/indexes **stat rel (row)**
- comptages accès disque/cache **stat io (block)**
- informations sur les requêtes **pg\_stat\_statements**

47

**Postgres Performance Observability – Alexey Lesovsky 2017**



<http://blog.postgresql-consulting.com/2017/02/deep-dive-into-postgres-stats.html>

48

## Informations sur les volumes

**pg\_column\_size** volume d'une donnée

**pg\_database\_size** volume d'un catalogue

**pg\_relation\_size** volume d'une table, d'un index

**pg\_total\_relation\_size** complet avec index et *toast*

**pg\_tablespace\_size** volume d'un espace (partition ?)

**pg\_size\_pretty** interprétation avec unités (kB, M...)

```
SELECT pg_size_pretty(pg_database_size('test'));
```

| pg_size_pretty |
|----------------|
| 12 MB          |

49

## Table pg\_statistic

— statistiques sur les valeurs de chaque colonne de chaque table

— mise à jour par la commande **ANALYZE** (sinon, vague défaut)

— utilisé par l'optimiseur pour évaluer les tailles de relations

**filtrage** nombreux opérateurs WHERE ...

**regroupement** GROUP BY ...

**jointure** JOIN ... ON ...

— selon le type analysé : min, max, moyenne, distribution...

— problème de droits d'accès : vue pg\_stats

50

## Collecte d'informations : requêtes, temps (attention au coût)

**log\_connection** connexions à la base

**log\_statement** requête exécutée

**log\_duration** durée d'exécution

```
[15611] LOG: query:
SELECT DISTINCT domaine, libMin
FROM thes, statel
WHERE statel.code_dipl=thes.domaine
AND domaine BETWEEN 50000 AND 59999
ORDER BY libMin
[15611] LOG: duration: 7.228555 sec
```

51

## Configuration de la collection de statistiques

**stats\_start\_collector** collecteur démarré

processus séparé de collecte, pas temps-réel

**stats\_command\_string** requête en cours

**stats\_row\_level** statistiques sur les tuples

*insert, update, fetch via indexes ou scans*

**stats\_block\_level** accès blocs cache ou disque

52

## Consultation des informations collectées

— nombreuses tables systèmes spéciales `pg_stat*`

**niveau** database, table et index, table, index...

**opération** scan avec ou sans index, ins/upd/del...

**cache** *read* (disque) vs *hit* (mémoire)

**obsolète** *fetch* vs *read*

— informations différées... pas trop de charge de collecte

— détection des indexes inutilisés, des tables scannées...

53

**pg\_stat\_activity** connexions et requêtes en cours (si longue)

| datid | datname   | procpid | username  | current_query                | query_start   |
|-------|-----------|---------|-----------|------------------------------|---------------|
| 19007 | corrector | 563     | corrector | SELECT COUNT(*) FROM answer; | 2017-03-20... |
| 19007 | corrector | 549     | postgres  | <IDLE>                       | 2017-03-20... |
| 19209 | comics    | 676     | coelho    | <IDLE>                       | 2017-03-20... |

54

**pg\_stat\_database** par base de données

— # connexions en cours

— # transactions confirmées (*commit*) vs annulées (*rollback*)

— blocks lus sur disque, lus dans le cache

on espère `read`  $\ll$  `hit`

| datid | datname   | numbackends | xact_commit | xact_rollback | blks_read | blks_hit |
|-------|-----------|-------------|-------------|---------------|-----------|----------|
| 17232 | coelho    | 0           | 112         | 7             | 149       | 9077     |
| 19007 | corrector | 2           | 87633       | 7             | 1085      | 9681100  |
| 19209 | comics    | 0           | 7785        | 2593          | 716       | 1047755  |
| 19499 | comtest   | 0           | 326         | 171           | 347       | 82055    |
| ...   | ...       | ..          | ...         | ...           | ...       | ...      |

55

**pg\_stat\_user\_tables** utilisation des tables all user sys

— nb parcours séquentiels et nb tuples lus (selon taille table)

— nb parcours indexés et nb tuples lus (comparables ?)

— nb d'insertions, mises à jour et effacement

| relname    | sq_scan | sq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
|------------|---------|-------------|----------|---------------|-----------|-----------|-----------|
| a_ecrit    | 75      | 31935       | 0        | 0             | 492       | 0         | 0         |
| collection | 81      | 3535        | 0        | 0             | 87        | 0         | 0         |
| langue     | 44      | 48          | 0        | 0             | 2         | 0         | 0         |
| exemplaire | 82      | 20766       | 0        | 0             | 279       | 0         | 0         |
| editeur    | 106     | 2586        | 0        | 0             | 67        | 0         | 0         |
| auteur     | 204     | 15064       | 0        | 0             | 151       | 0         | 16        |
| oeuvre     | 91      | 23387       | 119      | 116           | 298       | 13        | 0         |

56

**pg\_stat\_io\_user\_tables** disque/cache all user sys

- nombre de blocks table lus sur disque/cache
  - index de blocks des indexes de la table lus sur disque/cache
  - idem pour données *toastées* (attributs volumineux)...
- on espère read  $\ll$  hit

| relname    | heap_read | heap_hit | idx_read | idx_hit |
|------------|-----------|----------|----------|---------|
| oeuvre     | 3         | 1048     | 6        | 505     |
| exemplaire | 4         | 506      | 2        | 50      |
| editeur    | 2         | 271      | 4        | 168     |
| a_écrit    | 3         | 894      | 3        | 198     |
| auteur     | 2         | 605      | 2        | 154     |
| langue     | 2         | 85       | 0        | 0       |
| collection | 2         | 287      | 4        | 172     |

57

**pg\_stat\_user\_indexes** analyse par index all user sys

- nb parcours avec l'index
  - nb tuples lus par l'index
  - nb tuples récupérés par l'index
- différent nb tuples lus pour tuples expirés*

| relname    | indexrelname        | idx_scan | idx_tup_read | idx_tup_fetch |
|------------|---------------------|----------|--------------|---------------|
| auteur     | auteurs_pkey        | 36       | 2414         | 2414          |
| collection | collections_nom_key | 2        | 86           | 86            |
| collection | collections_pkey    | 1        | 43           | 43            |
| exemplaire | exemplaires_pkey    | 0        | 0            | 0             |
| oeuvre     | oeuvres_pkey        | 1106     | 9935         | 9935          |
| oeuvre     | oeuvres_titre_key   | 15       | 3069         | 3069          |
| a_écrit    | a_écrit_pkey        | 127      | 53975        | 53975         |
| ...        | ...                 | ...      | ...          | ...           |

58

**pg\_stat\_io\_user\_indexes** disque/cache all user sys

| relname    | indexrelname        | idx_blks_read | idx_blks_hit |
|------------|---------------------|---------------|--------------|
| auteur     | auteurs_pkey        | 2             | 73           |
| collection | collections_nom_key | 2             | 5            |
| collection | collections_pkey    | 2             | 3            |
| editeur    | editeurs_nom_key    | 2             | 1            |
| editeur    | editeurs_pkey       | 2             | 1            |
| exemplaire | exemplaires_pkey    | 2             | 1            |
| langue     | langues_nom_key     | 2             | 1            |
| langue     | langues_pkey        | 2             | 1            |
| oeuvre     | oeuvres_pkey        | 2             | 2213         |
| oeuvre     | oeuvres_titre_key   | 4             | 60           |
| a_écrit    | a_écrit_pkey        | 4             | 511          |

59

**pg\_stat\_io\_user\_sequences** disque/cache all user sys

| relid | schemaname | relname                  | blks_read | blks_hit |
|-------|------------|--------------------------|-----------|----------|
| 19008 | public     | connection_idcon_seq     | 1         | 3        |
| 19019 | public     | users_idusr_seq          | 1         | 19       |
| 19034 | public     | class_idcls_seq          | 1         | 2        |
| 19121 | public     | correctiontype_idcor_seq | 1         | 13       |
| 19133 | public     | question_idque_seq       | 1         | 75       |
| 19062 | public     | exercice_idexe_seq       | 1         | 1        |
| 19163 | public     | answer_idans_seq         | 1         | 659      |
| 19078 | public     | session_idses_seq        | 1         | 3        |

60

**Extension pg\_stat\_statements**

- extension standard à charger
 

```
CREATE EXTENSION pg_stat_statements;
```
- `postgresql.conf`:
 

```
-- shared_preload_library = 'pg_stat_statements'
```
- statistiques par requêtes *similaires*, constantes ignorées
 

```
SELECT * FROM films WHERE titre LIKE 'C%';
SELECT * FROM films WHERE titre LIKE 'A%';
SELECT * FROM films WHERE titre LIKE 'L%';
```
- regroupés sous :
 

```
SELECT * FROM films WHERE titre LIKE ?;
```

61

**Exemple d'extraction des requêtes lentes**

```
-- 10 requêtes les plus longues
SELECT query, total_time/calls AS avg
FROM pg_stat_statements
ORDER BY avg DESC LIMIT 10;

-- requêtes avec plus d'accès disques
SELECT query
FROM pg_stat_statements
ORDER BY shared_blks_read DESC LIMIT 10;
```

62

**Monitoring système et base**

- ps top htop** processus en cours...
- iostat iotop df du** charge et volume des partitions
- nagios, cactus** surveillance et alertes
- pg watch** interface web (PHP) de monitoring PostgreSQL
- pg\_activity pgtop** requêtes en cours...

<https://wiki.postgresql.org/wiki/Monitoring>

[https://wiki.postgresql.org/wiki/Performance\\_Analysis\\_Tools](https://wiki.postgresql.org/wiki/Performance_Analysis_Tools)

63

**Conclusion sur les performances**

- optimisation** difficile : selon valeurs, statistiques nécessaires
- identification** extension `pg_stat_statements`
- indexation** requête transactionnelles, coût de maintenance
  - choix des index et types pour les requêtes fréquentes
  - collecte statistiques, utilisation des logs...
- contrôle** explicite des optimisations ?
  - forcer les jointures : `join_collapse_limit=1`
  - limites : `WITH, LIMIT 0`
- caches** données souvent sollicitées gardées en mémoire...
  - en particulier *petites* tables périphériques

64



**influence** du matériel à l'application

- matériel disponible, dédié ou partagé à l'application, duplication ruptures technologiques : SSD vs HDD (vs Memristors ?)
- séparation des flux (log, WAL, bgwriter)
- configuration de l'OS, de la base de donnée. . .
- maintenance DB `VACUUM FULL ANALYZE...`
- application : requêtes inutiles, relations mal conçues. . .
- dénormalisation pour réduire les jointures ?

65

**Autres solutions**

- réplication *hot standby* et *pooling* ?
- distributions (applicative ?) sur des bases indépendantes ? *sharding* e.g. *déclarations d'impôts par régions*. . .
- caches mémoire distribué ? solutions NoSQL ?
- **uniquement** si nécessaire : modèle dénormalisé/dynamique utilisation plus lourde, programmation vs algèbre pas nécessairement plus rapide, selon usage

66

**List of Slides**

- 1 Optimisation de SQL
- 2 Optimisation : un problème global
- 3 Différents types de requêtes
- 4 Deux conseils préalables
- 5 1. Configurations logicielles et matérielles
- 6 2. Connaître ses données et sa charge
- 7 Comment ça marche ?
- 8 Planification d'une requête
- 9 Opérations élémentaires d'une requête
- 10 Coût des opérations élémentaires
- 11 Stockage des données : la hiérarchie mémoire
- 12 IBM 350 Disk File
- 13 Technologie des disques magnétiques
- 14 Mesure de coûts : I/O!
- 15 Évolutions des *Solid-State Drive* : SSD
- 16 Paramètres de modélisation I/O HDD
- 17 Parcours+Filtrage : parcours séquentiel
- 18 Parcours+Filtrage : parcours indexé
- 19 Parcours+Filtrage(S) : index bitmap
- 20 Indexation des attributs d'une relation
- 21 B-tree
- 22 Création d'un index
- 23 Importance des index
- 23 Coût des index
- 24 Index sur un attribut
- 25 Index partiel, avec prédicats
- 26 Index fonctionnels
- 27 Texte : ordre alphabétique selon la langue. . .
- 28 Index texte
- 29 Index *Bloom*
- 29 Index *R-Tree*
- 29 Index pour *Full Text Search* (FTS)
- 29 Index sur trigrammes `pgtrgm`
- 30 Tri d'une relation
- 31 Jointures de relations
- 32 Jointure par fusion triée
- 33 Jointure par hash
- 33 Autres techniques de calculs de jointure
- 34 Agrégations
- 35 Agrégation par fusion triée
- 36 Agrégation par hash
- 37 Exemples de `EXPLAIN` et `EXPLAIN ANALYZE`
- 38 Select simple
- 39 Select jointure
- 41 Exemples de création d'index
- 42 Analyse d'une requête pour proposer un `INDEX`
- 43 Exemple de proposition d'index (1)
- 44 Exemple de proposition d'index (2)
- 45 TD : Trouvez les index !
- 46 Collecte d'informations
- 47 Informations
- 48 Postgres Performance Observability – Alexey Lesovsky 2017
- 49 Informations sur les volumes
- 50 Table `pg_statistic`
- 51 Collecte d'informations : requêtes, temps (attention au coût)
- 52 Configuration de la collection de statistiques
- 53 Consultation des informations collectées
- 61 Extension `pg_stat_statements`
- 62 Exemple d'extraction des requêtes lentes
- 63 Monitoring système et base
- 64 Conclusion sur les performances
- 66 Autres solutions