Optimisation de SQL

Fabien Coelho, Claire Medrala

Mines Paris – PSL, MESR

Janvier 2025

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...

1. Configurations logicielles et matérielles

Matériel utilisé

- virtuelle/réelle ? partagée/dédiée ?
- nombre de cpu
- mémoire (cache OS) disponible
- type stockage HDD/SSD ? RAID ?

Système d'exploitation

- configurations : File system, options de montage...

Postgres configuration `postgresql.conf`

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

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 ?

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...`
  statisques 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...

Opérations élémentaires d'une requête

- **filtre** parcours avec filtrage d'une relation — `FROM WHERE...`
  - recherche des tuples
  - selon une condition
- **tri** d'une relation — `ORDER BY...`
  - ordre des tuples affichés
- **joint** jointure de deux relations — `JOIN ON...`
  - mise en correspondance de tuples
  - selon une condition
- **groupe** aggrégation d'une relation — `GROUP BY`, `DISTINCT`, `UNION...`
  - regroupement des éléments
  - selon un critère
  - peut utiliser un tri !
- etc.

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...
- **statisques** min/max, moyenne, distribution avec `ANALYZE`

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 μs	1-9 GB/s	10000
SSD 1	128 Go-2 To	100 μs	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 μs	100-1000 MB/s	-

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

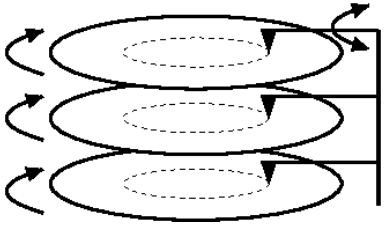
Disque dur

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

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

Mesure de coûts : I/O !

unité page PostgreSQL 8 KB
- peut être changée, page FS ?

sequentiel HDD/SSD X00 MB/s
- bus IDE/SCSI/..., charge
- vitesse rotation, RAID...

aléatoire HDD 1-10 MB/s
- latence : bras (3-15 ms) et rotations (0.5 tr)
- un ordre de grandeur du séquentiel

aléatoire SSD 10-300 MB/s, X0 K IOPS
- 60-70% performance séquentiels
- lecture vs écriture

calculs mémoire simplement négligés ! 0 !
- très rapide par rapport aux accès disques

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 ≈ écriture (100-200 MB/s) ≈ **HDD**
- aléatoire ≈ séquentiel ≠ **HDD**
- **usure** : si plein écriture aléatoire lente, nombre limité ! cycle W page (4 KB) vs Erase block (51

impact coûts $\times 8 - 15$, capacité $\times \frac{1}{4} - \frac{1}{8}$
performances & optimisations ?
vitesse par *tablespace*...

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'aggrégation)
j	probabilité de jointure selon une condition...
M	mémoire disponible (cache, données temporaires)

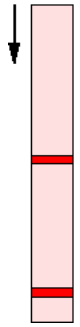
Filtrage : parcours séquentiel — seq scan

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

- 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%'
```



Filtrage : parcours indexé — index scan

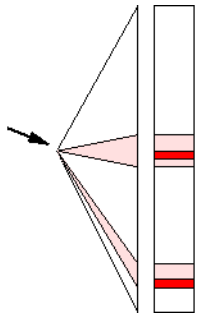
index scan $K + K \ln(ns/B)$

- 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
```



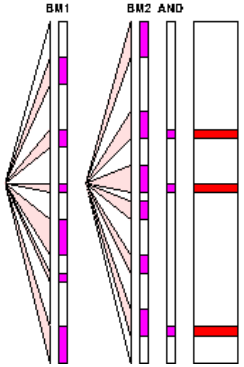
Filtrages combinés — bitmap index scan

bitmap index scan

- création de bitmaps 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. . .

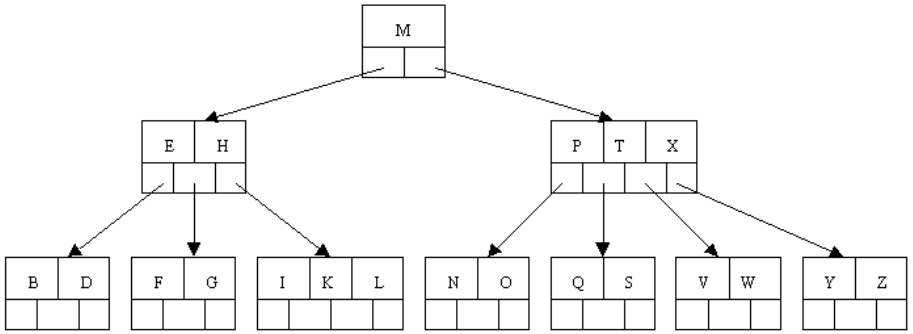
GiST *Generalized Search Tree* – full texte, géométrie

Bloom aggrégation 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 Minerd, Saint Vincent College

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`, `PRIMARY KEY`

```
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`
  existance 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;
```


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)));
```

Texte : ordre alphabétique selon la langue. . .

posix
Akin
Akın
cotée
côté
grot
groß
ninon
niño

français
Akin
Akın
côté
cotée
groß
grot
niño
ninon

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);
```


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`

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

Jointures de relations

- relation ($=$ ou autre...) entre attribut(s), clefs ou non...
- tables jointes de cardinalité n_R et n_P, résultat $jn_\mathrm{R}n_\mathrm{P}$
- évaluation de j selon les données ?

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

```
SELECT vnom, dpt FROM exp.Ville ORDER
BY vnom LIMIT 8;
```


Jointure par fusion triée

- tri des relations selon le critère de jointure !
  puis parcours parallèle et fusion au vol des relations
  $$\frac{k}{B}(t_\mathrm{R}(Xn_\mathrm{R}\ln(n_\mathrm{R})+n_\mathrm{R})+t_\mathrm{P}(Yn_\mathrm{P}\ln(n_\mathrm{P})+n_\mathrm{P})+t_\mathrm{RP}jn_\mathrm{R}n_\mathrm{P})$$
- 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

Jointure par hash

- relation mise en **mémoire**, indexée par critère de jointure
  puis parcours séquentiel de l'autre relation
  $$\frac{k}{B}(t_\mathrm{R}n_\mathrm{R}+t_\mathrm{P}n_\mathrm{P}+t_\mathrm{RP}jn_\mathrm{R}n_\mathrm{P})$$
- la mémoire doit être suffisante... risque de *swap* ?

Autres techniques de calculs de jointure

boucles différentes variantes, avec index...

filtre produit cartésien, puis filtrage...

Agrégations

- regroupements si attributs égaux : `GROUP BY`, `DISTINCT`
- évaluation du taux de regroupement g ?
- parallélisation !

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

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

Agrégation par hash

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

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

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 pas évidente
- `http://explain.depesz.com/`
- `https://explain.dalibo.com/`

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
```

Select jointure 1

```
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)
```


Select jointure 1 (détails)

explain analyze

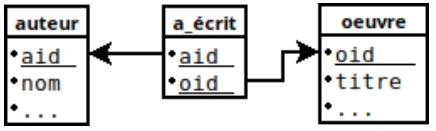
```
Hash Join  (cost=1.09..2.26 rows=2 width=12)
           (actual time=0.051..0.124 rows=6 loops=1)
  Hash Cond: (films.pid = personnes.pid)
  -> Seq Scan on films  (cost=0.00..1.11 rows=11 width=16)
                        (actual time=0.008..0.075 rows=11 loops=1)
  -> Hash  (cost=1.07..1.07 rows=1 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
```


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
     - attention à l'ordre des UNIQUE

Exemple de proposition d'index (1)

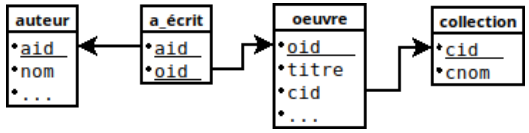
```
SELECT titre
FROM auteur
JOIN a_ecrit USING (aid)
JOIN oeuvre USING (oid)
WHERE nom = 'Goscinny';
```



- `nom` → `aid` sur `auteur` — index `auteur(nom)`
- `aid` → `oid` sur `a_ecrit` — index `a_ecrit(aid)`
  existe déjà, préfix clef composite
- `oid` → `titre` sur `oeuvre` : — index `oeuvre(oid)`
  existe déjà, `oid` est clef primaire

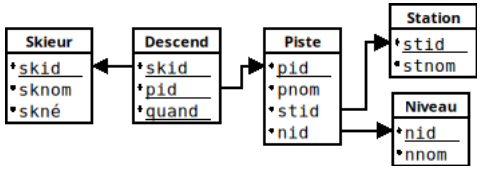
Exemple de proposition d'index (2)

```
SELECT DISTINCT pseudo
FROM auteur
JOIN a_ecrit 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_ecrit` — index `a_ecrit(oid)`
- `aid` → `pseudo` sur `auteur` — index `auteur(aid)`
  existe déjà, clef primaire

TD : Trouvez les index !



Skieur *PK(skid), U(sknom, skné)*
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 *janvier 2020* ?
- Les pistes de niveau *extrême* ?
- Les skieurs ayant descendu des pistes à *Courchevel* ?
- Les stations fermées en *mars 2020* ?

Collecte d'informations

- niveaux à considérer : client, **base de données**, OS
- très nombreuses informations disponibles
- mais non historisées
- extension `pg_stat_statements` indispensable
  statistiques par requêtes

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`

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

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

Conclusion sur les performances... suite

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 ?