TD sur les Index

PARTIE 1 : index et performance de requête

Objectifs : compréhension des **mécanismes d'indexation** pour l'optimisation des requêtes

1. mettre en place les tables suivantes
   ```
   CREATE TABLE T1 (id INT, A INT);
   CREATE TABLE T2 (id INT, B INT);
   INSERT INTO T1 SELECT generate_series(1, 20000) AS id, (1000 * random())::INT AS A;
   INSERT INTO T2 SELECT generate_series(1,  1000) AS id, (1000 * random())::INT AS B;
   ```
2. Vérifier le contenu de T1 en affichant que les 10 premières lignes.a
3. Exécuter la requête `SELECT * FROM T1, T2 WHERE T1.A = T2.B;`
4. Combien de temps semble prendre cette requête (utiliser éventuellement `\timing`)

Plan d'exécution (*i.e.* comment un SGBD prévoit de calculer une requête) : pour cela, nous allons utiliser **pgAdmin**

1. Lancer l'analyseur de requête *Explain query* sur la requête précédente. Le plan d'exécution s'affiche sous une forme graphique.
2. On utilisera maintenant Explain analyze (Maj+F7) pour avoir à la fois le plan d'exécution et le temps d'exécution réel (en bas à droite). Vérifier que le temps d'exécution est cohérent avec la mesure faite précédemment en 4).
3. Pour mieux mettre en évidence le rôle des index, on va empêcher PostgreSQL d'optimiser les requêtes. Pour cela, taper :
   ```
   SET enable_mergejoin TO off;
   SET enable_hashjoin TO off;
   SET enable_material TO off;
   ```
4. Mettre en place des index sur certaines colonnes (a priori celles qui vont être utilisées lors des jointures ?). Utiliser le type d'index par défaut.
5. Analyser le plan d'exécution produit pour la requête SELECT précédemment utilisée.
6. Mesurer l'impact sur le temps d'exécution. Quel index est utilisé et pourquoi ?
7. Enlever un des 2 index et vérifier si l'autre index est utilisé. Quelles sont les nouvelles performances ?
8. Supprimer cet index.
9. Mettre en place un seul index qui soit du type le plus adapté (cf. Type d'index PostgreSQL).

PARTIE 2 : Index et recherche textuelle

Objectifs : compréhension des **mécanismes d'indexation** notamment pour la recherche « plein texte »

ATTENTION => beaucoup de fonctions sont uniquement utilisable sur un SGBD PostgreSQL !!

Préambule :

- vérifier la bonne installation de certaines extension (http://www.postgresql.org/docs/9.1/static/catalog-pg-extension.html) :
- Si vous étiez super-utilisateur, vous devriez écrire :

```
CREATE EXTENSION unaccent ; – pour accent français qui pourrait poser problème
CREATE EXTENSION tablefunc;
CREATE EXTENSION dict_xsyn;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION pg_trgm; – similarité de texte par correspondance trigramme
CREATE EXTENSION cube;
```

Vérifier par **\dx** que ces extensions sont bien installées pour votre BD.

Exercice 1 : Recherche plein texte : dictionnaire filtrant

1) Copier la configuration française par défaut

```
create text search configuration fr (COPY = french);
```

2) Tester cette configuration

```
select to_tsvector('fr', 'une fois dans le répertoire contrib/unaccent du répertoire des sources') @@ to_tsquery('fr'', 'répertoire') ;
select to_tsvector('fr', 'une fois dans le répertoire contrib/unaccent du répertoire des sources') @@ to_tsquery('fr'', 'repertoire') ;
```

Que remarquez-vous ?

3) Mettre en place le dictionnaire filtrant **unaccent**

```
alter text search configuration fr alter mapping for hword, hword_part, word with unaccent, french_stem ;
```

4) Refaire la question 2)

5) Donc on va appliquer la vectorisation (découpage mot à mot) sur la colonne *titre* de **bookbay.livres**

```
alter table bookbay.livres add column titre_vectorise tsvector ;
update bookbay.livres set titre_vectorise = to_tsvector(titre) ;
```

6) vérifier le contenu des colonnes *titre* et *titre_vectorise* de **bookbay.livres**

7) analyser la requête précédente avec EXPLAIN; regarder le coût de la requête.

8) créer un index **idxLivres_titreVectorise** sur *titre_vectorise* avec USING **GIN**(titre_vectorise) puisque c'est par elle que l'on va chercher des mots. Pour bien faire, il faudrait aussi mettre en place un trigger pour qu'à chaque nouvelle entrée ou modification de *titre*, on insère dans *titre_vectorise* son découpage en mot.

9) analyser la requête précédente avec EXPLAIN ; regarder le coût de la requête.

10) classement des livres selon le poids associés pour le mot 'amour' dans le titre

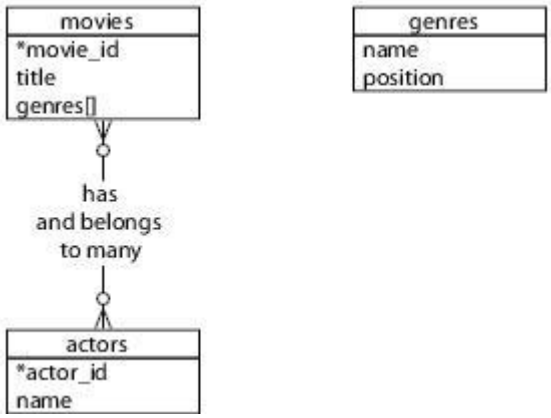
```
select titre, ts_rank(titre_vectorise, to_tsquery('amour')) from bookbay.livres where titre_vectorise @@ to_tsquery('amour') ;
```

N.B. : l'utilisation dans les dictionnaires des synonymes de préfixes va permettre d'affecter des correspondances . Par exemple, le mot « indices » est souvent utilisé comme pluriel de « index » donc dans le dictionnaire, on peut définir que « indices » a comme synonyme « index* » ce qui permettra une correspondance avec tout mot commençant par index.

Exercice 2 : Recherche et recommandation de films

- recherche: textuelle, fuzzy matching, phonetique
- moteur de recommandation basique

Le schéma utilisé est :



1) Vérifier le script contenant les tables et l'index ***movies_genre_cube*** de type **GIST** sur la colonne **genre** de la table **movies**
2) Lire la documentation **GIST à http://docs.postgresql.fr/9.6/gist.html**
3) Lire la documentation **GIN à http://docs.postgresql.fr/9.6/gin.html**
4) Types de recherche :

a) Recherche exacte / pattern matching : like ou regEX

Quels sont les films qui ont le mot *stardust* dans leur nom ?

Compter tous les films dont le titre ne commence pas par « the ».

Quels sont les films qui ont le mot *war* dans leur nom mais pas en dernière position ?

Afficher le plan d'exécution de la dernière requête (EXPLAIN).

Comment accélérer cette requête ?

```
CREATE INDEX movies_title_pattern ON movies (lower(title) text_pattern_ops);
```

Afficher le plan d'exécution de la dernière requête (EXPLAIN).

Est-ce que cela a changé ?

b) Distance de Levenshtein => typos simple

Utiliser les fonctions de l'extension **fuzzystrgmatch** (http://docs.postgresql.fr/9.1/fuzzystrmatch.html) pour trouver :

- La distance de **Levenstein** entre *intention* et *execution*
- Quels sont les films qui sont à une distance de Levenstein inférieure à 10 de « *a hard day nght* »

Principe :

I	N	T	E	*	N	T	I	O	N
|	|	|	|	|	|	|	|	|	|
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

Tableau 1 : distance Levenstein = nb minimal d'opérations d, s, i (delete, substitute, insert)

Voir un autre exemple en annexe.

c) N-Gram / Recherche similarité (%) => trouver les erreurs modérées

Principe : voir annexe

Documentation pour **pgtrim** : http://docs.postgresql.fr/9.1/pgtrgm.html

Écrire des requêtes pour :

- trouver tous les trigrammes du mot *Avatar*.
- la similarité entre *VOTKA* et *VODKA*
- Tous les films dont le titre est similaire à Avatre

Afficher le plan d'exécution de la dernière requête.

Rajouter un index pour l'accélérer

```
CREATE INDEX movies_title_trigram ON movies USING gist (title gist_trgm_ops);
```

d) Full text match @@ => similarité grammaticale

Principe : ignorer les mots de liaisons, pluriels, etc. contenu dans le texte…

ALGO :

- extraire les racines des mots (lexèmes) => **ATTENTION**, spécifique au langage donc dicco adéquat !
- Comparer les vecteurs des lexèmes.

  select **to_tsvector**('A hard day'' night'), **to_tsquery**('english', 'night & day'),
  to_tsvector | to_tsquery

'day':3 'hard':2 'night':5 | 'night' & 'day'

où ts-vector = lexeme :position
ts_query = lexèmes séparée par &

Documentation :
Recherche plein texte : http://docs.postgresql.fr/9.1/textsearch.html
Indexation GIST, GIN : http://docs.postgresql.fr/9.1/textsearch-indexes.html

Trouver les films qui contiennent les formes grammaticales des mots « night » et day »

```
SELECT title FROM movies
WHERE to_tsvector(title) @@ to_tsquery('english', 'night & day');

SELECT title FROM movies WHERE title @@ 'night & day' ;
```

A Hard Day's Night
Six Days Seven Nights
Long Day's Journey Into Night

Afficher le plan d'exécution de la dernière requête.

Rajouter un index pour l'accélérer

```
CREATE INDEX movies_title_trigram ON movies USING gist (title gist_trgm_ops);
```

e) Métaphone => similarité phonétique

Plusieurs fonctions pour la codification phonétique des mots
Documentation pour **fuzzystrmatch**: http://docs.postgresql.fr/9.1/fuzzystrmatch.html
Exemple :

```
SELECT name, dmetaphone(name), dmetaphone_alt(name), metaphone(name, 8), soundex(name)
FROM actors;
```

```
name     | dmetaphone | dmetaphone_alt | metaphone | soundex
---------------+------------+----------------+-----------+---------
50 Cent      | SNT      | SNT          | SNT     | C530
Aaron Eckhart  | ARNK     | ARNK         | ARNKHRT   | A652
Agatha Hurle   | AK0R     | AKTR         | AK0HRL    | A236
```

Trouver les films qui ont des acteurs dont les noms se prononcent pareils.

Trouver les acteurs qui ont un nom similaire à *Robin Williams,* triés par similarité (combiner %, metaphone et Levenstein)

```
actor_id | name
----------+-----------------
   4093 | Robin Williams
   2442 | John Williams
   4479 | Steven Williams
   4090 | Robin Shou
```

RECHERCHE MULTI-DIMENSIONNELLE (cube olap…)
Documentation : http://docs.postgresql.fr/9.1/cube.html
On utilise :

- le type **cube** pour mapper sur un vecteur **n-dimensionnel** de valeurs (score du film)
  Voir dans le script la création de la table **movies**.
  Pour faire une insertion dans ce cube :

```
INSERT INTO movies (movie_id,title,genre) VALUES
(1,'Star Wars','(0,7,0,0,0,0,0,0,0,7,0,0,0,0,10,0,0,0)') ;
```

- Les noms pour les **dimensions** sont définis dans la table **Genres** (vérifier unicité de la colonne *name*).

```
INSERT INTO genres (name,position) VALUES ('Action',1), ('Adventure',2),
('Animation',3),…, ('Sport',16), ('Thriller',17), ('Western',18);
```

Utiliser le module **cube p**our recommander des films similaires (du même genre) :

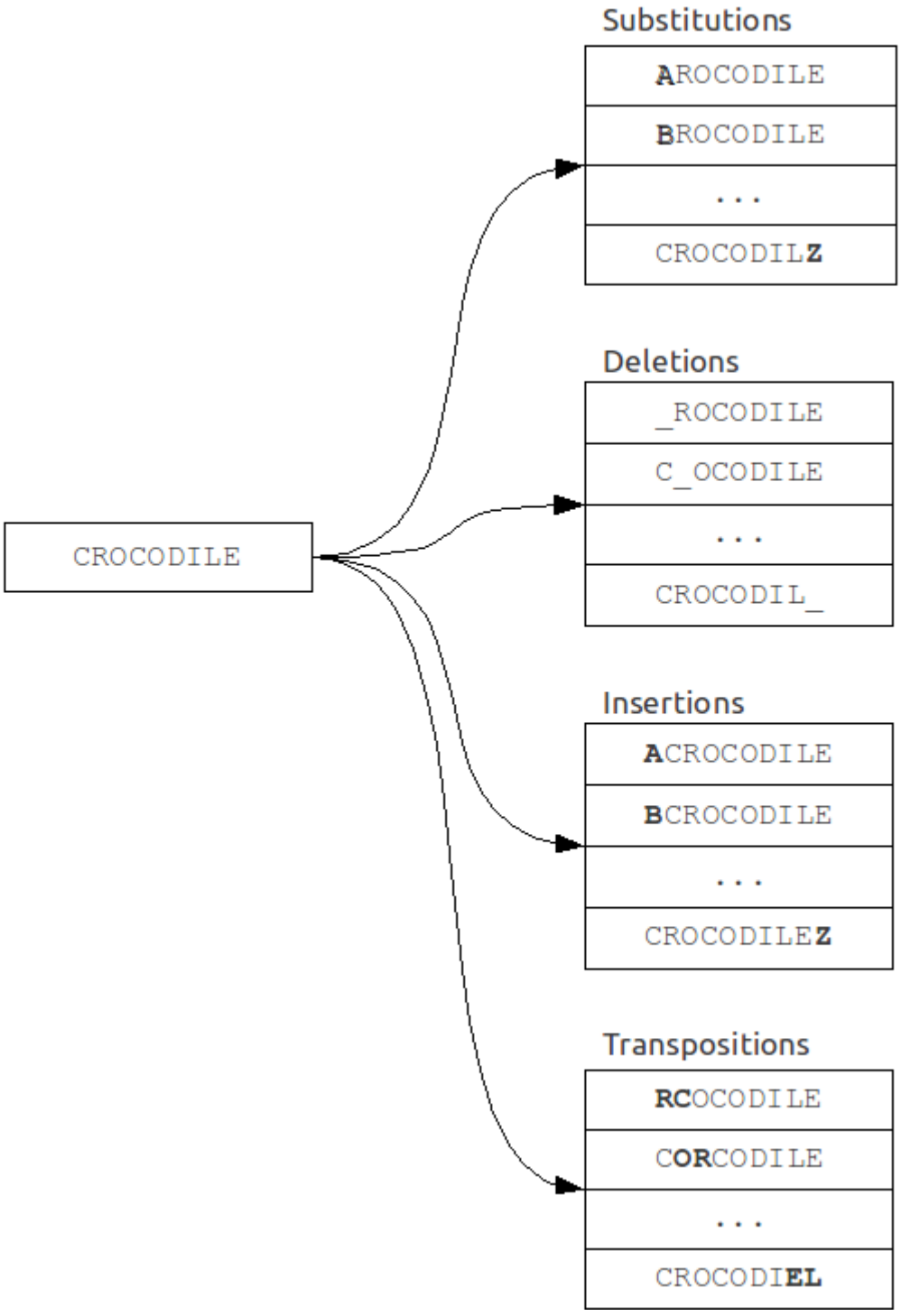
- afficher le vecteur des notes du film ***Star Wars***
- quel est la note du film ***Star Wars*** dans la catégorie '*Animation*' ?
- afficher les films avec les meilleurs notes dans la categorie *SciFi*.
- afficher les filmes similaire (**cube_distance**) a ***Star Wars*** (vecteur = (0, 7, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 10, 0, 0, 0)) du plus similaire au moins similaire

```
title                                |      dist
-------------------------------------------------+------------------
Star Wars                            |           0
Star Wars: Episode V - The Empire Strikes Back |           2
Avatar                               |           5
Explorers                            | 5.74456264653803
Krull                                | 6.48074069840786
E.T. The Extra-Terrestrial           | 7.61577310586391
```

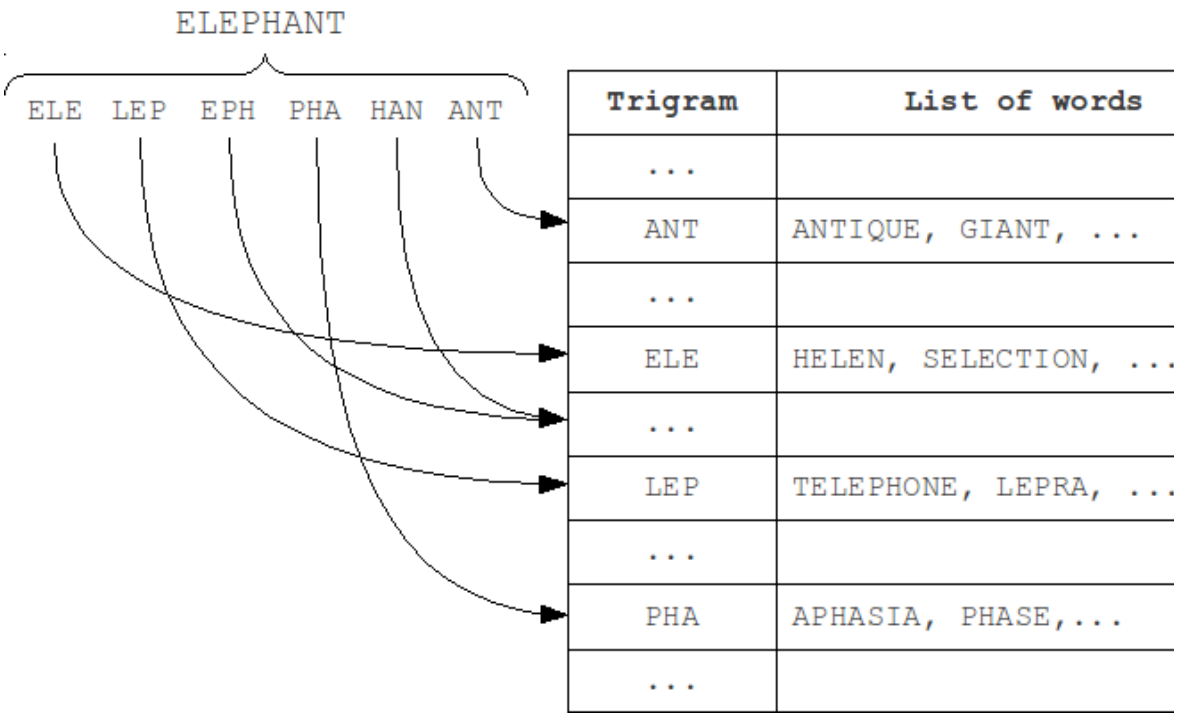
- écrire une requête pour trouver les films qui sont a moins de 5 points de différence sur chaque dimension (utiliser *cube_enlarge* et *@>*).

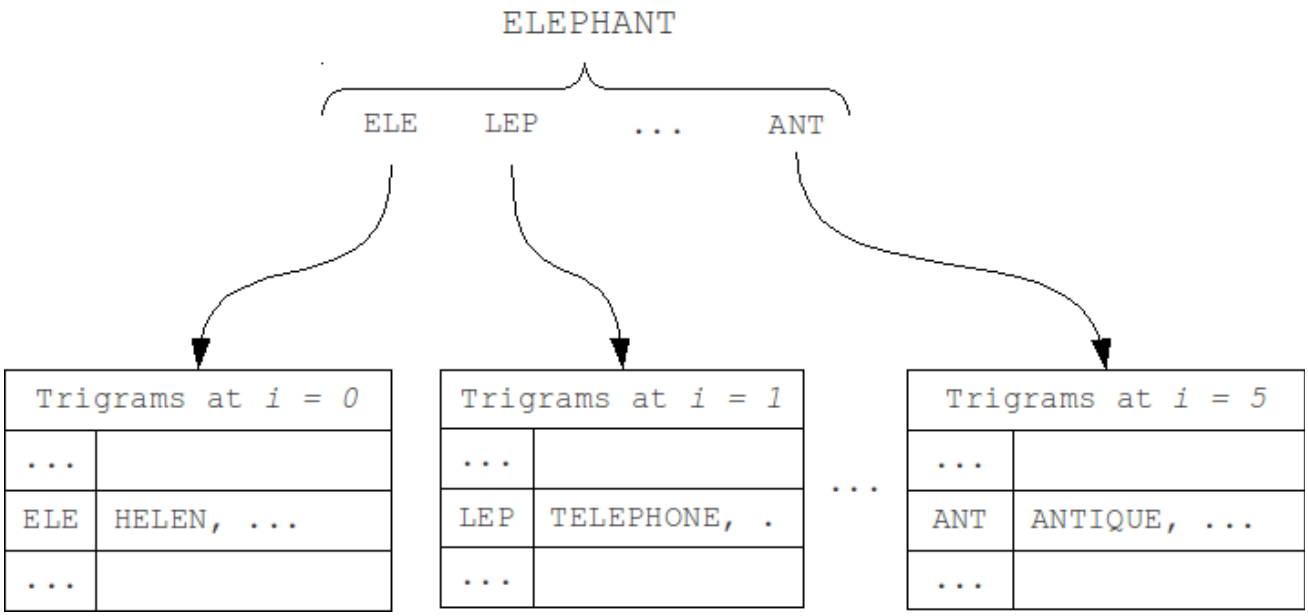
ANNEXES

Exemple Levenstein



Exemple N-gram


ELEPHANT
ELE LEP EPH PHA HAN ANT
Trigram
List of words
...
ANT
ANTIQUE, GIANT, ...
...
ELE
HELEN, SELECTION, ...
...
LEP
TELEPHONE, LEPRA, ...
...
PHA
APHASIA, PHASE,...
...


ELEPHANT
ELE LEP ... ANT
Trigrams at i = 0
... |
ELE | HELEN, ...
... |
Trigrams at i = 1
... |
LEP | TELEPHONE, .
... |
...
Trigrams at i = 5
... |
ANT | ANTIQUE, ...
... |