Bases de données avancées
TP n° 9 : Les index et plans d'exécution

I) Les index sous postgres

On crée un index sur un attribut en utilisant la syntaxe suivante (version simplifiée) :

```
CREATE INDEX nom_index
ON nom_table (nom_attribut);
```

On peut préciser le type d'index, HASH (table de hachage) ou BTREE (arbre de recherche). (Il existe aussi les index GIN et GIST pour les types complexes.) Par défaut les index sont des BTREE. Ceux-ci permettent la comparaison et le parcours séquentiel (en ordre trié). Lorsque ces fonctionnalités ne sont pas utiles, les index HASH permettent un accès plus rapide sur critère d'égalité.

```
CREATE INDEX nom_index
ON nom_table  USING HASH (nom_attribut);
```

On peut également créer un index sur une liste d'attributs (et dans ce cas, l'ordre des attributs est important) ou sur une expression.

II) Les plans d'exécution

Afin de comprendre comment a été optimisée une requête, on peut consulter son plan d'exécution en tapant `EXPLAIN` devant la requête. L'affichage représente un arbre qui correspond à l'ordre dans lequel les jointures et autres opérations sont effectuées. Pour aussi exécuter la requête et obtenir des détails sur son temps d'exécution, préfixez la requête de `EXPLAIN ANALYZE` ; la requête est calculée mais son résultat n'est pas retournée.
ATTENTION : comme `EXPLAIN ANALYZE`, exécute réellement la requête, il ne faut pas l'appeler sur des requêtes trop coûteuses.

III) Préparation

La présence ou non des index a un impact sur les plans d'exécution et l'efficacité des requêtes SQL. Comme les tables sont volumineuses, nous avons préparé plusieurs variantes des tables de données de la ratp. Vous ne créerez pas les index sur une copie personnelle des tables, car ceci serait trop coûteux en espace disque.
Pour chacune des tables suivantes, consultez la structure de la table à l'aide de la commande `\d`, et répondez aux questions suivantes :

1. Quels sont les index présents sur la table ?
2. Sur quel attribut, liste d'attributs, ou expression porte chaque index ?
3. De quel type d'index s'agit-il ? (BTREE ou HASH ?)

Les tables sont dans le schéma public :

A. `trips, trips_1, trips_2`

B. `calendar_dates, calendar_dates_2`

C. `routes, routes_2`

D. `calendar`

IV) Un premier plan d'exécution : rôles des index

Considérons la requête suivante qui effectue une sélection sur une jointure des tables `routes, trips` et `calendar_dates`.

1. Pour chaque table, notez sa taille (nombre de lignes).
2. Tapez la requête suivante.

   ```
   EXPLAIN SELECT  route_id, route_short_name, date
   FROM routes NATURAL JOIN trips NATURAL JOIN calendar_dates
   WHERE route_id = 990429 ;
   ```

   Regardez le plan d'exécution retourné par `EXPLAIN`. Dans quel ordre les exécutions se font-elles ? Notez le coût estimé. (Il est calculé en équivalent "récupération d'une page de disque".)
3. Que se passe-t-il si on fait la même requête en utilisant la table `trips_1` au lieu de `trips` ?
4. Et si on reprend la première en prenant `routes_2` au lieu de `routes` ?
5. Laquelle de ces trois variantes est la plus efficace ? Quel est l'index qui est le plus déterminant ? Pourquoi ?

V) Arbre de jointure

On reprend la requête précédente mais sans la sélection `route_id = 990429`.

1. Comment le plan d'exécution a-t-il changé ?
2. Les index servent-ils encore ?
3. Refaites la même requête avec les mêmes combinaisons que plus haut. Y a-t-il encore des index qui servent ?
4. Essayez maintenant de faire la requête en utilisant `calendar_dates_2` au lieu de `calendar_dates`. Que se passe-t-il ?

VI) Deux requêtes qui font presque la même chose

Il faut parfois réécrire une requête pour qu'elle soit efficace. Les requêtes NOT IN et les requêtes contenant des sous-requêtes sont souvent difficilement optimisables par l'optimiseur, ceci d'autant plus que les sous-requêtes sont corrélées.
Voici un exemple de requête de ce type. On veut savoir quelles sont les trajets (routes) qui sont concernés par tous les services. On peut faire la requête de la façon ci-dessous. **Ne la lancez pas, elle met énormément de temps**. Mais analysez-la avec `EXPLAIN`.

```
explain select route_id
from routes R
where not exists (
    select *
    from calendar
    where service_id not in (
         select service_id
         from trips T
         where R.route_id = T.route_id
    )
);
```

Avec quel(s) index peut-on espérer réduire le temps d'exécution ? Essayez en choisissant la ou les bonnes versions de tables.
On peut faire une requête presque équivalente.

```
select route_id
from trips
group by route_id
having count (distinct service_id) = (select count(*)
                                      from calendar);
```

On remarquera que si jamais la table `calendar` était vide les deux requêtes ne donneraient pas le même résultat. C'est la seule différence.
Vous pouvez faire `EXPLAIN ANALYZE` pour la seconde requête. Que remarque-t-on ?