

BDR - Devoir surveillé

2 avril 2022 - durée 2h

Documents autorisés. Appareils électroniques de communication interdits.

Exercice 1 : *Des indexes.* (5 points)

Soit T une table qui contient 1000000 lignes. Une ligne de la table fait 81 octets. Un bloc fait 8192 octets.

Question 1.1 : Sachant qu’une ligne est stockée entièrement dans le même bloc, et qu’un bloc contient un entête de 20 octets, combien de lignes peut-on mettre au maximum par bloc ?

Question 1.2 : En déduire le nombre de blocs utilisés pour stocker le million de lignes de la table T .

Question 1.3 : On définit un index (un B+ arbre) Idx sur la clé primaire id de T , id étant un entier entre 1 et 1000000. La clé fait 32 octets et une adresse fait 8 octets. Dans un bloc "feuille" de l’index, on stocke des couples (clé, adresse). On supposera qu’il n’y a pas d’entête dans ces blocs. Combien a-t-on de blocs feuilles si Idx est un index dense (justifier) ?

- environ 200
- environ 1000
- environ 5000
- environ 1000000

Question 1.4 : L’index Idx possède 3 niveaux : la racine, un niveau interne et le niveau des feuilles. Pour chaque requête, est-ce que le moteur SQL a intérêt à utiliser l’index (justifier) :

1. `select * from T where id=100;`
2. `select * from T where id >= 50;`
3. `select * from T where id between 101 and 200;`

Question 1.5 : Soit T une table comprenant un attribut A , et un index sur A . Pourquoi n’est-il pas raisonnable d’attendre d’un SGBD qu’il soit toujours capable d’utiliser l’index pour n’importe quelle requête de la forme `SELECT * FROM T WHERE fonction(T.A) = 0`, où `fonction` est une fonction arbitraire ?

Exercice 2 : *Stockage et recherche.* (3 points)

Soit T une table sur laquelle nous avons besoin de répondre fréquemment à des requêtes de la forme `select * from T where T.A = x`, où A est un attribut de la table et x est une valeur valide pour cet attribut. Pour améliorer le temps d’exécution de ce type de requêtes, nous envisageons deux solutions possibles :

1. La table est stockée triée par valeurs croissantes de A , et les blocs successifs sont adjacents sur le disque ;

2. La table est stockée triée par valeurs croissantes de A , mais les blocs consécutifs ne sont plus adjacents sur le disque mais sont *chainés* (c'est-à-dire que chaque bloc contient l'adresse de son successeur).

Question 2.1 : Est-il possible de faire une recherche par dichotomie avec la solution 1 ? Avec la solution 2 ? Justifiez.

Question 2.2 : Quels sont les avantages et inconvénients des solutions 1 et 2 ? Dans quels cas est-il intéressant de choisir l'une plutôt que l'autre ?

Exercice 3 : *Fonctions stockées.* (3 points)

En lisant le schéma d'une base de données, vous trouvez la fonction stockée suivante :

```
CREATE OR replace FUNCTION nbClientsMoyen()
  RETURNS FLOAT
AS $$
  DECLARE
    e record;
    nb INTEGER;
    nb_comptes INTEGER;
    total INTEGER;
  BEGIN
    total := 0;
    nb_comptes := 0;
    FOR e IN SELECT ncompte FROM compte_client
      LOOP
        nb_comptes := nb_comptes + 1;
        SELECT COUNT(ncli) INTO nb FROM compte_client WHERE ncompte =e.ncompte;
        total := total+ nb;
      END LOOP;
    RETURN CAST(total AS FLOAT)/CAST(nb_comptes AS FLOAT);
  END;
$$
LANGUAGE plpgsql;
```

La table `compte_client` indique à quels clients (représentés par leurs numéros de client `ncli`) appartiennent quels comptes (représentés par leurs numéros de compte `ncompte`) et est décrite par le schéma suivant :

```
create table banque.compte_client(
  ncompte integer references compte,
  ncli integer references client,
  constraint compte_client_pkey primary key(ncompte, ncli)
);
```

Question 3.1 : Que fait cette fonction ? Pouvez-vous écrire une requête qui renvoie le même résultat ?

Question 3.2 : Vous semble-t-il préférable d'utiliser cette fonction ou une requête ? Pourquoi ?

Exercice 4 : *Des flux.* (5 points)

Rappelons-nous que, lors du premier TP, nous avons modélisé en Python les résultats des opérateurs relationnels comme des flux de dictionnaires, chaque dictionnaire représentant un tuple du résultat de la manière suivante : les clés sont les divers attributs de la table, et la valeur associée à une clé est la valeur de cet attribut pour le tuple.

Question 4.1 : Quel est l'intérêt d'implémenter les opérateurs relationnels comme des flux ?

Dans ce contexte, nous avons modélisé un index sur un attribut A d'une table comme étant un dictionnaire qui, à chaque valeur possible de A , associe la liste des adresses des tuples de la table ayant cette valeur pour A . Nous utilisons alors un tel index via la fonction `trouver_sur_disque(fichier, adresses)`, qui renvoie le flux des tuples dans `fichier` ayant des adresses physiques dans `adresses`.

Question 4.2 : Complétez la fonction suivante.

```
def sum_group_by(fichier, idx, B):
    """~idx~ est un index sur l'attribut A d'une table T stockée dans le
        fichier ~fichier~.
        Construit le flux des tuples (dictionnaires) correspondant au résultat
        de la requête SELECT A, SUM(B) from T GROUP BY A. La colonne pour la
        somme sera nommée "sum_B"""
```

Question 4.3 : Si `idx[v]` retourne effectivement une *liste* d'adresses, quel est l'empreinte mémoire pire-cas de votre implémentation pour la fonction `sum_group_by` ?

Question 4.4 : Faisons maintenant l'hypothèse que `idx[v]` retourne un *flux* d'adresses dont l'empreinte mémoire est faible. Quelle est l'empreinte mémoire pire-cas de votre implémentation pour la fonction `sum_group_by` ? Si `idx` est implémenté comme un arbre-B, pensez-vous que cette hypothèse est raisonnable ? (justifiez)

Exercice 5 : *Dénormalisation.* (4 points)

On considère qu'on stocke des produits dans différents entrepôts. On manipule donc des **produits**, des **depots** et des **stocks**. La table `produit` contient outre un numéro qui sert de clé, un nom d'article, une taille. La table `depot` contient les informations relatives à chaque dépôt, en particulier son nom et adresse. La table `stock` permet de savoir quelle quantité d'articles de chaque produit se trouve dans chacun des dépôts.

```
create table produit(
    nprod integer constraint produit_pkey primary key,
    nomprod varchar(70) not null,
    tailleprod varchar(5) not null
);
create table depot(
    ndepot integer constraint depot_pkey primary key,
    nomdepot varchar(30) not null,
    adrdepot varchar(230) not null,
```

```

villedepot varchar(30)
);
create table stock(
  nprod integer not null references produit,
  ndepot integer not null references depot,
  qteDepot int not null
  constraint stock_pkey primary key (nprod, ndepot)
);

```

Question 5.1 : Ces produits sont en vente sur Internet. On veut donc pouvoir à tout instant préciser aux clients si le produit est encore disponible ou pas. Par ailleurs, pour les produits dont la quantité totale (et non par entrepôt) est inférieure à 10, on veut pouvoir afficher le nombre d’articles du produit concerné, s’il reste. La requête est faite très souvent sur un volume de données important. Répondez aux questions suivantes en justifiant votre réponse (on ne demande pas d’écrire du code SQL, mais d’expliquer et de justifier ce que vous feriez) :

1. Que proposez vous pour optimiser la requête ?
2. Que proposez vous pour maintenir à jour ces informations lorsqu’on modifie la table `stock` ?

Pour organiser les ventes, la base de données contient des tables `commande` et `ligne_de_commande` décrites comme suit :

```

CREATE TABLE commande(
  ncommande INTEGER CONSTRAINT commande_pkey PRIMARY KEY,
  adrcommande VARCHAR(230) NOT NULL -- adresse de livraison
);

CREATE TABLE ligne_de_commande(
  nligne INTEGER CONSTRAINT ligne_pkey PRIMARY KEY,
  ncommande INTEGER NOT NULL REFERENCES commande,
  nprod INTEGER NOT NULL REFERENCES produit
);

```

On suppose par ailleurs que nous disposons d’une fonction stockée `distance` qui étant donnée deux adresses calcule la distance entre les deux adresses.

Question 5.2 : Écrire une requête qui étant donnée une adresse `adr` et un numéro de produit `np` renvoie le numéro de dépôt le plus proche de `adr` disposant de ce produit.

Question 5.3 : Pour les commandes, on souhaite que chaque produit de la commande soit envoyé depuis le dépôt le plus proche disposant de ce produit. Ainsi, lorsqu’une ligne de commande est ajoutée, supprimée ou encore modifiée, il faut maintenir la cohérence de la base de données. Comment vous y prendriez-vous (on ne demande pas d’écrire du code SQL, mais d’expliquer et de justifier ce que vous feriez) ?