TIn311 - Bases de données - IPSA Toulouse

Alexandre Condette – alexandre.condette@spacebel.fr

2021-2022

5 TP n° 3 : Vues, indexes et `GROUP BY`

Ce TP se réalise toujours avec la base de données `arcep.db` du TP précédent.

5.1 Vues

Une vue est une table virtuelle basée sur une requête `SELECT`. Une vue peut représenter un sous-ensemble des données d'une base et permet de masquer la complexité de la requête sous-jacente.

Exemples :

```
-- création de vues, notez le mot clef AS qui sert à nouveau
CREATE VIEW Urls_IPSA AS
SELECT url FROM urls WHERE url LIKE "%ipsa%";

CREATE VIEW Top10 AS
SELECT url, visit_count, datetime(last_visit_date/1000000, "unixepoch") AS last_visit
FROM moz_places
WHERE (last_visit_date/1000000) > CAST(strftime("%s", "now", "-1 month") AS INT)
ORDER BY visit_count DESC
LIMIT 10;

-- utilisation
SELECT * FROM Top10;

-- suppression d'une vue
DROP VIEW Top10;
```

5.1.1 Opérateurs présents dans les départements

Dans un exercice précédent, on a codé la requête suivante pour lister tous les départements dans lequel l'opérateur Orange est présent.

```
SELECT DISTINCT d.nom AS departements_ou_y_a_orange
FROM Departement AS d
INNER JOIN Point AS p ON d.code_insee = p.code_insee_dep
INNER JOIN Mesure AS m ON p.id_point = m.id_point
INNER JOIN Operateur AS op ON m.id_operateur = op.id_operateur
WHERE op.nom = "Orange"
ORDER BY d.nom;
```

⇒ Créez une vue pour lister tous les opérateurs et les départements dans lesquels ils sont présents.

⇒ Utilisez cette vue pour obtenir le même résultat pour Orange.

Correction

```
CREATE VIEW DepOps AS
SELECT DISTINCT d.nom AS dep, op.nom AS op
FROM Departement AS d
```

```
INNER JOIN Point AS p ON d.code_insee = p.code_insee_dep
INNER JOIN Mesure AS m ON p.id_point = m.id_point
INNER JOIN Operateur AS op ON m.id_operateur = op.id_operateur
ORDER BY d.nom;

SELECT * FROM DepOps WHERE dep = "Orange";
```

5.2 Indexes

Étant donné la table et la requête suivantes :

```
CREATE TABLE Joueur (
    nom TEXT NOT NULL,
    faction TEXT NOT NULL,
    race TEXT NOT NULL,
    niveau INT NOT NULL,
    pièces_d_or INT NOT NULL,

    PRIMARY KEY (nom)
);

--
SELECT * FROM Joueur WHERE niveau = 120;
```

Cette requête implique que le SGBD doit parcourir séquentiellement toutes les lignes de la table pour trouver celle qui satisfont à la condition du `WHERE`.

Quand la table contient *beaucoup* de lignes, cette opération devient longue. Si on a besoin de souvent réaliser ce type de recherches, cela devient un problème de performances.

Un index sur une ou plusieurs colonnes d'une table est une structure de données qui permet d'accélérer ce type d'opérations. Cette structure de données permet au SGBD de localiser efficacement les lignes correspondant au(x) critère(s) de recherche.

Un index est automatiquement créé pour chaque clef primaire. Le SGBD utilise ensuite automatiquement les indexes s'ils existent pour accélérer les requêtes (pas de changement de syntaxe). Créer un index est une modification permanente de la structure de la base (on peut le supprimer bien évidemment).

Les indexes sont mis à jour automatiquement par le SGBD : chaque insertion, mise-à-jour ou suppression de lignes lui nécessite un travail supplémentaire. Il s'agit donc d'un compromis, un index occupant d'ailleurs de l'espace disque supplémentaire.

Avoir un index sur une colonne très fréquemment mise à jour mais peu utilisée comme critère de recherche peut donc avoir un impact négatif sur les performances.

Exemple :

```
CREATE INDEX Joueur_niveau_i ON Joueur(niveau); -- le nom de l'index est arbitraire

DROP INDEX Joueur_niveau_i;
```

Par ailleurs, on peut également créer un index unique, c'est-à-dire qu'il garantit l'unicité des valeurs dans les colonnes qu'il indexe. On rajoute ainsi une contrainte.

Exemple :

```
CREATE UNIQUE INDEX etudiant_email_u ON Etudiant(email);
```

5.2.1 Accélérer les requêtes du TP précédent

Reprenez les requêtes du TP précédents §4.4.5 à §4.4.9. Certaines sont relativement lentes (> 1 seconde de temps d'exécution).

⇒ Analysez les colonnes qui sont utilisées dans ces requêtes et comment, et voyez si en créant quelques indexes bien placés vous pouvez accélérer ces requêtes

Correction

```
CREATE UNIQUE INDEX Departement_nom_idx ON Departement(nom);
CREATE INDEX Point_code_insee_dep_idx ON Point(code_insee_dep);
CREATE INDEX Point_longitude_idx ON Point(longitude);
```

5.3 GROUP BY

Une requête SELECT renvoie un ensemble de lignes qui satisfont aux critères de la requête. On a cependant déjà rencontré des fonctions d'agrégations, c'est-à-dire des requêtes qui nous renvoient un résultat portant sur un ensemble de lignes : AVG, SUM, MIN, MAX et bien sûr COUNT.

Dans une requête SELECT, la clause GROUP BY nous permet de regrouper plusieurs lignes selon des critères et d'appliquer une de ces fonctions à chacun des "groupes".

Par exemple, étant donné la table Etudiant ci-dessous :

INE	Nom	Prenom	Age	Promo
123	Dupont	Jean	21	AERO3
125	Dupond	Luc	22	AERO3
133	Bob	Alice	20	AERO2
156	Granger	Hermione	18	AERO1
167	Potter	Harry	18	AERO1
189	Ovid	Frias	19	ING1

la requête suivante donne la liste des promotions, le nombre d'étudiants et la moyenne d'âge de *chaque promotion*, triée par moyenne d'âge croissante, pour toutes les promotions AERO.

```
SELECT promo, COUNT(INE) AS nb_etudiants, AVG(Age) AS age_moyen
FROM Etudiant
WHERE promo LIKE "AERO%"
GROUP BY promo
ORDER BY AVG(AGE);
```

Résultat :

promo	nb_etudiants	age_moyen
AERO1	2	18
AERO2	1	20
AERO3	2	21.5

On peut en plus filtrer les groupes renvoyés par un GROUP BY en utilisant la clause HAVING.

Par exemple, la requête ci-dessous donne la même liste de promotions que précédemment mais uniquement pour les promotions avec au moins 2 étudiants.

Attention, la clause HAVING filtre au niveau de chaque "groupe" alors que WHERE s'applique à chaque ligne individuellement. La clause WHERE est appliquée en premier aux lignes, qui ensuite sont regroupées par GROUP BY et enfin HAVING est appliqué.

```
SELECT promo, COUNT(INE) AS nb_etudiants, AVG(Age) AS age_moyen
FROM Etudiant
WHERE promo LIKE "AERO%"
GROUP BY promo
HAVING COUNT(INE) >= 2
ORDER BY AVG(AGE);
```

Résultat :

promo	nb_etudiants	age_moyen
AERO1	2	18
AERO3	2	21.5

5.3.1 Syntaxe générale, ordre des clauses

```
SELECT ...
FROM ...
... JOIN ... ON ...
-- et d'autres JOIN
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
; -- on n'oublie pas le ; /!\
```

5.3.2 Donnez la requête pour obtenir le nombre de points par département.

Correction

```
SELECT d.nom AS nom_departement, COUNT(p.id_point) AS nb_points
FROM Departement AS d
LEFT JOIN Point AS p ON d.code_insee = p.code_insee_dep
GROUP BY d.code_insee
ORDER BY COUNT(p.id_point) DESC;
```

5.3.3 Donnez la requête pour obtenir le débit moyen mesuré par département.

Correction

```
SELECT d.nom AS nom_departement, AVG(m.qualite) AS debit_moyen
FROM Departement AS d
LEFT JOIN Point AS p ON d.code_insee = p.code_insee_dep
LEFT JOIN Mesure AS m ON m.id_point = p.id_point
GROUP BY d.code_insee
ORDER BY AVG(m.qualite) DESC;
```

5.3.4 Donnez la requête pour lister les départements dont le débit moyen est inférieur à la moyenne.

Correction

```
SELECT d.nom AS nom_departement, AVG(m.qualite) AS debit_moyen
FROM Departement AS d
LEFT JOIN Point AS p ON d.code_insee = p.code_insee_dep
LEFT JOIN Mesure AS m ON m.id_point = p.id_point
GROUP BY d.code_insee
HAVING AVG(m.qualite) < (SELECT AVG(qualite) AS qualite_moyenne FROM Mesure)
    OR AVG(m.qualite) IS NULL
ORDER BY AVG(m.qualite);
```

5.3.5 Donnez la requête pour compter les départements dans lesquels l'opérateur de votre choix est présent avec une qualité moyenne supérieure à 10 Mbit/s.

Correction

Avec nos connaissances, on peut résoudre ce problème en s'aidant d'une vue.

```
CREATE VIEW Debits_Moyens AS
SELECT d.nom AS nom_departement, ope.nom AS operateur, AVG(m.qualite) AS debit_moyen
FROM Departement AS d
```

```
INNER JOIN Point AS p ON d.code_insee = p.code_insee_dep
INNER JOIN Mesure AS m ON m.id_point = p.id_point
INNER JOIN Operateur AS ope ON m.id_operateur = ope.id_operateur
GROUP BY d.code_insee, ope.id_operateur;

SELECT COUNT(*)
FROM Debits_Moyens
WHERE operateur = "SFR" AND debit_moyen > 10
ORDER BY nom_departement;
```

5.3.6 Donnez la requête pour compter le nombre de points sur lesquels moins de 3 opérateurs ont été mesurés.

Correction

```
CREATE VIEW nb_ops_par_point AS
SELECT p.*, COUNT(m.id_operateur) AS nb_operateurs
FROM Point AS p
LEFT JOIN Mesure AS m ON m.id_point = p.id_point
GROUP BY p.id_point;

SELECT COUNT(*) FROM nb_ops_par_point WHERE nb_operateurs < 3;
```

5.3.7 Donnez la liste des opérateurs dont la qualité moyenne est inférieure à 30Mbit/s

Correction

```
SELECT ope.nom AS operateur, AVG(m.qualite) AS debit_moyen
FROM Operateur AS ope
LEFT JOIN Mesure AS m ON  m.id_operateur = ope.id_operateur
LEFT JOIN Point AS p ON m.id_point = p.id_point
GROUP by ope.id_operateur
HAVING debit_moyen IS NULL OR debit_moyen < 30;
```

5.3.8 Donnez la requête pour trouver l'opérateur le plus rapide en moyenne.

Correction

```
-- 1ère solution en utilisant une vue
CREATE VIEW Debits_Operateurs AS
SELECT op.*, AVG(m.qualite) AS debit_moyen
FROM Operateur AS op
INNER JOIN Mesure AS m ON op.id_operateur = m.id_operateur
GROUP BY op.id_operateur;

SELECT *, MAX(debit_moyen) FROM Debits_Operateurs;

-- 2ème solution avec une requête imbriquée
SELECT *, MAX(debit_moyen)
FROM    (SELECT op.*, AVG(m.qualite) AS debit_moyen
         FROM Operateur AS op
         INNER JOIN Mesure AS m ON op.id_operateur = m.id_operateur
         GROUP BY op.id_operateur);
```