SQL avancé

Ludovic Deneuville

Données utilisées

```sql
CREATE TABLE echecs.joueuse (
    id_joueuse     SERIAL         PRIMARY KEY,
    nom            VARCHAR(50)    NOT NULL,
    prenom         VARCHAR(50)    NOT NULL,
    date_naissance TIMESTAMP,
    elo            INT            CHECK (elo > 0),
    mail           VARCHAR(100)   UNIQUE,
    est_arbitre    BOOLEAN        DEFAULT false,
    code_pays      VARCHAR(2),
    taux_victoires FLOAT          CHECK (taux_victoires BETWEEN 0 AND 1)
);
```

Sous-requête

Dans un filtre

Vous souhaitez lister les joueuses avec un elo supérieur à la moyenne.

```sql
SELECT nom,
       prenom,
       elo
FROM echecs.joueuse
WHERE elo >= (SELECT AVG(elo)
                FROM echecs.joueuse);
```

Imbriquer les SELECT

```sql
SELECT j.nom,
       j.prenom,
       j.elo,
       mec.max_elo_club
  FROM (SELECT id_club,
               MAX(elo) AS max_elo_club
          FROM joueuse
         GROUP BY id_club) mec
  JOIN joueuse j USING(id_club);
```

EXISTS

```
SELECT *
  FROM echecs.club c
 WHERE EXISTS (SELECT 1
                 FROM echecs.joueuse j
                WHERE c.id_club = j.id_club
                  AND j.elo > 2400);
```

La syntaxe ressemble à celle du `IN` mais le `EXISTS` est plus performant.

```
SELECT *
  FROM echecs.club c
 WHERE c.id_club IN (SELECT j.id_club
                       FROM echecs.joueuse j
                      WHERE j.elo > 2400);
```

Les vues

Joueuses Françaises

> **i** Question
>
> Imaginons que nous avons de nombreuses requêtes à écrire sur les joueuses françaises ayant plus de 2000 elo.
>
> Et nous n'avons pas envie de nous embêter à écrire à chaque fois toutes les restrictions.

Nouvelle table

Solution 1 : créons une nouvelle table

```
CREATE TABLE echecs.joueuse_fr_2000
AS
SELECT *
  FROM echecs.joueuse
 WHERE code_pays = 'FR'
   AND elo >= 2000
```

> **🔥** Problèmes
>
> - Duplication des données
> - Mise à jour

Common Table Expression

Pour une utilisation ponctuelle, utilisez une CTE 😎

- Commencez par définir votre sous-requête *joueuses_fr_elo2000* avec `WITH`
- Appelez *joueuses_fr_elo2000* dans la requête principale

```sql
WITH joueuses_fr_elo2000
AS (SELECT *
      FROM echecs.joueuse
     WHERE code_pays = 'FR'
       AND elo > 2000
)
SELECT nom, prenom, elo
  FROM joueuses_fr_elo2000
 WHERE prenom LIKE 'A%';
```

Table temporaire

N'existe que le temps de la session.

```sql
CREATE TEMPORARY TABLE joueuses_fr_elo2000
AS
SELECT *
  FROM echecs.joueuse
 WHERE code_pays = 'FR'
   AND elo > 2000;
```

> Remarque
>
> Une table temporaire n'est pas définie dans un schéma.

Vue

> Définition
>
> Une vue représente une table virtuelle, dont la structure et les données proviennent du résultat d'une requête SQL prédéfinie.
>
> Elle ne stocke pas directement les données, mais la requête elle-même, et s'exécute à chaque utilisation.

```sql
CREATE VIEW echecs.joueuses_fr_elo2000
AS
SELECT *
  FROM echecs.joueuse
 WHERE code_pays = 'FR'
   AND elo > 2000;
```

Avantages et inconvénients

- ✓ Simplicité d'accès aux données
- ✓ Meilleure organisation des données
- ✓ Contrôle des accès aux données
- ❌ Performance

Vues matérialisées

Photo à un instant t (snapshot) 📷

```
CREATE MATERIALIZED VIEW joueuses_fr_elo2000
AS
SELECT *
  FROM echecs.joueuse
 WHERE code_pays = 'FR'
   AND elo > 2000;
```

Gérer les droits

L'utilisateur qui crée une table peut distribuer des droits aux autres.

Voir section *priviléges* de la doc PostgreSQL

Donner des droits sur une table

```
-- Lecture pour tous
GRANT SELECT ON echecs.joueuse TO PUBLIC;

-- Tous les droits
GRANT ALL PRIVILEGES ON echecs.joueuse TO chantal WITH GRANT OPTION;
```

- `PUBLIC` : tous les utilisateurs
- Suffixe `WITH GRANT OPTION` pour donner le droit de transmettre ce droit

Retirer des droits

```
-- Lecture
REVOKE SELECT ON echecs.joueuse TO chantal;

-- Tous les droits
REVOKE ALL PRIVILEGES ON echecs.joueuse TO chantal;
```

Suffixe `CASCADE` pour retirer les droits à ceux qui ont reçu les droits de la part de…

Rôle

Pour une gestion simplifiée des droits

```
-- créer un rôle
CREATE ROLE eleve;

-- ajouter des droits au rôle
GRANT SELECT, INSERT ON echecs.joueuse TO eleve;

-- donner le rôle à Chantal
GRANT eleve TO chantal;
```

Droits sur un schéma

```
-- droit d'accès au schéma i.e. de consulter les objets du schéma
GRANT USAGE ON SCHEMA echecs TO PUBLIC;

-- retire à Chantal le droit de créer de nouveaux objets (tables, vues, etc.) dans le schéma
REVOKE CREATE ON SCHEMA echecs FROM chantal;

-- Autorise à effectuer des requêtes SELECT sur toutes les tables du schéma
GRANT SELECT ON ALL TABLES IN SCHEMA echecs TO PUBLIC;
```

Transactions

Exemple classique

> ! Mission
>
> Vous devez transférer discrétement et sans vous tromper 10 millions entre les comptes de Shakira et Cristiano.
>
> Malheureusement l'opération est assez longue et vous devez vous prémunir contre le risque d'une interruption.
>
> Comment faire ? 😕

Transfert de fonds

```
BEGIN; -- Début de la transaction

UPDATE compte SET solde = solde - 10000000 WHERE id = 1;
UPDATE compte SET solde = solde + 10000000 WHERE id = 2;

COMMIT; -- Valider si ok
```

```
ROLLBACK; -- en cas d'erreur
```

Propriétés ACID

Pour définir la fiabilité des transactions :

- **Atomicité** : Une transaction est "tout ou rien"
- **Cohérence** : La base reste dans un état valide avant et après la transaction
- **Isolation** : Les transactions concurrentes n'interfèrent pas entre elles
- **Durabilité** : Une transaction validée est enregistrée de manière permanente, même en cas de panne

Index

Optimisation

Possible d'optimiser à plusieurs niveaux :

- Logique : Normalisation / Dénormalisation
- Physique : Indexation, Partitionnement
- Requêtes SQL

Créer un index

> ℹ Définition
>
> Un index est une structure de données qui améliore la vitesse des opérations de lecture sur une table en facilitant la recherche des données.
>
> Il permet une lecture plus rapide sur une colonne utilisée fréquemment dans *WHERE*, *JOIN*, *GROUP BY*, *ORDER BY*

```
CREATE INDEX echecs.idx_prenom_joueuse ON echecs.joueuse (prenom);
```

Application

```
SELECT *
  FROM echecs.joueuse
 WHERE prenom = 'Amandine';
```

- Sans index : parcourt intégral de la table
- Avec index :
  - recherche sous forme d'arbre, récupère les id
  - lit directement les lignes concernées

Avantages et inconvénients

- ✓ Haute cardinalité
- ✓ Peu de valeurs nulles
- ❌ Créations, mises à jour
- ❌ Prend de la place

Fonction

- Bloc de code nommé qui retourne une valeur
- Encapsuler une logique métier simple.

```
SELECT nom,
       get_categorie(date_naissance)
  FROM echecs.joueuse;
```

Exemple

```
CREATE OR REPLACE FUNCTION echecs.get_categorie(p_date_naissance DATE)
RETURNS VARCHAR(20) AS $$
DECLARE
    age INT;
BEGIN
```

```
    age := DATE_PART('year', AGE(CURRENT_DATE, p_date_naissance));

    IF age < 20 THEN
        RETURN 'Juniore';
    ELSIF age < 60 THEN
        RETURN 'Seniore';
    ELSE
        RETURN 'Vétérante';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Procédure

- Bloc de code SQL exécuté manuellement avec CALL
- Peut modifier la base de données
- Ne retourne pas de valeur
- Automatiser une série d'actions

```
CALL augmenter_elo(3, 50);
```

Trigger

- Se déclenche quand un événement survient sur une table
- Servent à maintenir la cohérence ou enregistrer des historiques

Exemples :

- Tenir un journal d'audit, historiser les changements
- Appliquer des règles automatiques

Exemple

```
CREATE OR REPLACE FUNCTION echecs.format_nom_prenom()
RETURNS TRIGGER AS $$
BEGIN
    NEW.nom := INITCAP(NEW.nom);
    NEW.prenom := INITCAP(NEW.prenom);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_format_nom_prenom
BEFORE INSERT OR UPDATE ON echecs.joueuse
FOR EACH ROW
EXECUTE FUNCTION echecs.format_nom_prenom();
```

Exemple 2

```
CREATE OR REPLACE FUNCTION echecs.block_update_taux()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.taux_victoires <> OLD.taux_victoires THEN
        RAISE EXCEPTION 'Le taux de victoires ne peut pas être modifié manuellement.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```