Explication Technique de la Transaction SQL

But de la Requête

L'objectif de cette transaction SQL est de gérer la réservation de sièges pour une séance de cinéma de manière sécurisée et atomique. Cette transaction s'assure que les sièges demandés ne sont pas déjà réservés avant d'insérer une nouvelle réservation. Elle utilise des mécanismes de verrouillage et de contrôle pour garantir que toutes les opérations se déroulent sans conflit et qu'aucune donnée incohérente n'est introduite dans la base de données.

Détails de la Transaction

1. Début de la Transaction :

La commande BEGIN démarre une nouvelle transaction. Toutes les opérations suivantes seront traitées comme une seule unité de travail. Si une partie de la transaction échoue, toutes les modifications seront annulées.

```
1   BEGIN;
```

2. Verrouillage de la Transaction pour Éviter les Conflits :

```
3   -- Verrouillage de la transaction pour éviter les conflits
4   SELECT pg_advisory_xact_lock(1);
```

La fonction pg_advisory_xact_lock(1) obtient un verrou transactionnel. Cela garantit qu'aucune autre transaction concurrente ne peut accéder aux mêmes ressources, évitant ainsi les conflits de réservation.

3. Vérification de la Disponibilité des Sièges Réservés :

```
-- Vérification de la disponibilité des sièges réservés
DO $$
DECLARE
  seats_reserved_json jsonb := '[{"seat_id": 1}, {"seat_id": 2}]';
  showtimes_id integer := 123;
BEGIN
  IF EXISTS (
    SELECT 1
    FROM reservations
    WHERE showtimes_id = showtimes_id AND seats_reserved @>
    seats_reserved_json
  ) THEN
    RAISE EXCEPTION 'One or more seats are already reserved!';
  END IF;
END $$;
```

Ce bloc PL/pgSQL vérifie si les sièges demandés sont déjà réservés pour la même séance (showtimes_id).

- seats_reserved_json contient les sièges réservés sous forme de JSON.
- showtimes_id identifie la séance de cinéma.
- La requête SELECT vérifie l'existence de toute réservation qui inclut les sièges demandés.
- Si de telles réservations existent, une exception est levée avec le message 'One or more seats are already reserved!'.

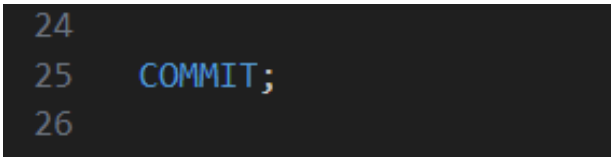
4. Insertion d'une Nouvelle Réservation :

```
-- Insertion d'une nouvelle réservation
INSERT INTO reservations (user_id, cinema_id, showtimes_id, seats_reserved,
status, reserved_at, token)
VALUES (1, 1, 123, '[{"seat_id": 1}, {"seat_id": 2}]'::jsonb, false, NOW(),
'random_token_string');
```

Si les sièges ne sont pas déjà réservés, cette commande insère une nouvelle ligne dans la table reservations avec les détails suivants :

- user_id : Identifiant de l'utilisateur effectuant la réservation.
- cinema_id : Identifiant du cinéma.
- showtimes_id : Identifiant de la séance de cinéma.
- seats_reserved : Les sièges réservés, stockés sous forme de JSON.
- status : Statut de la réservation (initialement false).
- reserved_at : Date et heure de la réservation.
- token : Un jeton unique pour la réservation.

5. Validation de la Transaction :



La commande COMMIT termine la transaction et applique toutes les modifications apportées à la base de données. Si une partie de la transaction échoue avant cette commande, toutes les modifications seront annulées, garantissant que l'opération est atomique (tout ou rien).

Cette transaction SQL assure que les sièges demandés pour une séance de cinéma ne sont réservés que si aucun conflit n'est détecté, garantissant ainsi l'intégrité des données. Le processus est atomique, ce qui signifie que toutes les opérations doivent réussir pour que les modifications soient appliquées, sinon aucune modification n'est apportée. Ce mécanisme de transaction est essentiel pour maintenir la consistance et l'intégrité des données dans des systèmes concurrents comme la réservation de sièges de cinéma.