Projet de système de base de données pour le site de préférences vidéo

Aperçu du projet

Le but de ce projet est de concevoir et d'implémenter un système de base de données robuste pour un site de replay vidéo qui gère les inscriptions des utilisateurs, les détails des vidéos, l'historique de visionnage et les suggestions personnalisées de vidéos. La plateforme permettra aux utilisateurs d'explorer des vidéos par catégories, de suivre leurs favoris et de recevoir des recommandations personnalisées basées sur la popularité des vidéos. La base de données gérera également le cycle de vie des vidéos, y compris l'archivage du contenu plus ancien.

Objectifs

1. **Développer une base de données relationnelle pour gérer :**
   (a) **Informations des utilisateurs** : inscription, identifiants de connexion, détails du profil et préférences.
   (b) **Détails des vidéos** : métadonnées, disponibilité, catégorisation et cycle de vie.
   (c) **Activité des utilisateurs** : historique de visionnage, favoris et abonnements.
   (d) **Suggestions** : recommandations personnalisées basées sur la popularité récente des vidéos.
2. **Assurer que la base de données peut supporter des fonctionnalités telles que :**
   (a) Suivi de la disponibilité des vidéos et automatisation de l'archivage après l'expiration de la période de visionnage.
   (b) Génération de suggestions de visionnage dynamiques pour les utilisateurs inscrits.
   (c) Enregistrement des interactions des utilisateurs pour des expériences personnalisées et des analyses historiques.

Portée

Entités de la base de données

— **Utilisateurs :**

- **Attributs** : Identifiant (login), mot de passe, nom de famille, prénom, date de naissance, email, préférences (catégories d'intérêt) et statut d'abonnement à la newsletter.
- **Fonctionnalités** :
  - Suivre les vidéos favorites.
  - S'abonner à des émissions pour des suggestions automatiques.
  - Accéder à l'historique de visionnage.

— **Vidéos :**
- **Attributs** : Nom, description, durée, date de sortie, pays d'origine, support multilingue, format d'image et statut de disponibilité (actif/archivé).
- **Fonctionnalités** :
  - Catégorisées en genres (ex. : culture, cinéma).
  - Support des émissions à plusieurs épisodes.
  - Gestion du cycle de vie (disponibilité minimale de 7 jours, règles d'archivage).

— **Historique de visionnage :**
- **Attributs** : ID utilisateur, ID vidéo, date de visionnage.
- **Fonctionnalités** :
  - Maintenir un enregistrement des vidéos visionnées pour chaque utilisateur.

— **Suggestions :**
- **Générées en fonction de** :
  - Popularité au sein des catégories (nombre de vues au cours des deux dernières semaines).
  - Abonnements de l'utilisateur aux émissions.

Principales fonctionnalités

1. **Gestion des utilisateurs :**
   - Permettre aux utilisateurs de créer des comptes, de se connecter et de gérer leurs préférences.
   - Fournir l'option de s'abonner aux newsletters et à des catégories de programmes spécifiques.
2. **Cycle de vie des vidéos :**
   - Archiver automatiquement les vidéos après une durée spécifique tout en assurant au moins 7 jours de disponibilité.
3. **Expérience de visionnage :**
   - Permettre aux utilisateurs de marquer des vidéos comme favorites et de les afficher sur une page personnalisée.
   - Mettre en évidence les vidéos proches de la fin de leur disponibilité.
4. **Système de recommandation :**
   - Générer des suggestions basées sur la popularité des catégories au cours des deux dernières semaines.
5. **Analyse des données :**
   - Suivre les vues des vidéos et les tendances de popularité.

Entités

Voici un aperçu détaillé des entités et de leurs attributs associés :

1. Utilisateur

- **Attributs** :
  - Login (Identifiant unique)
  - Mot de passe
  - Prénom
  - Nom de famille
  - Date de naissance
  - Adresse email
  - Catégories d'intérêt
  - Abonnement à la newsletter hebdomadaire (Booléen : Oui/Non)
- **Relations** :
  - S'abonne à des émissions (Plusieurs-à-plusieurs avec Programme)
  - Marque des vidéos comme favorites (Plusieurs-à-plusieurs avec Vidéo)
  - A un historique de visionnage (Un-à-plusieurs avec Visionnage Vidéo)

2. Vidéo

- **Attributs** :
  - ID Vidéo (Identifiant unique)
  - Nom
  - Description
  - Durée
  - Date de sortie
  - Pays d'origine
  - Disponibilité multilingue (Booléen : Oui/Non)
  - Format d'image
  - Statut (Disponible/Archivé)
- **Relations** :
  - Appartient à un programme (Plusieurs-à-un avec Programme)
  - Apparaît dans les favoris de l'utilisateur (Plusieurs-à-plusieurs avec Utilisateur)
  - Vue dans l'historique de l'utilisateur (Un-à-plusieurs avec Visionnage Vidéo)

3. Programme

- **Attributs** :
  - ID Programme (Identifiant unique)
  - Nom
  - Catégorie (ex. : Culture, Cinéma)
- **Relations** :
  - Inclut des épisodes/vidéos (Un-à-plusieurs avec Vidéo)

4. Visionnage Vidéo

- **Attributs** :

— ID Visionnage (Identifiant unique)
— ID Utilisateur (Clé étrangère vers Utilisateur)
— ID Vidéo (Clé étrangère vers Vidéo)
— Horodatage du visionnage

— **Relations** :
— Suit l'historique de visionnage des vidéos de l'utilisateur (Plusieurs-à-un avec Utilisateur)
— Suit quelle vidéo a été visionnée (Plusieurs-à-un avec Vidéo)

5. Catégorie

— **Attributs** :
— ID Catégorie (Identifiant unique)
— Nom

— **Relations** :
— Associée à des programmes (Un-à-plusieurs avec Programme)

6. Suggestions de Visionnage

— **Attributs** :
— ID Suggestion (Identifiant unique)
— Horodatage de génération
— Données de popularité de catégorie (Nombre de vues au cours des deux dernières semaines)

— **Relations** :
— Vidéos suggérées (Un-à-plusieurs avec Vidéo)

Création de la Base de Données

J'ai utilisé **PostgreSQL 17** pour la gestion de la base de données.

Schéma SQL

```sql
-- 1. Table Utilisateurs
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    login VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    date_of_birth DATE,
    email VARCHAR(100) UNIQUE,
    interested_categories TEXT[], -- Tableau de cat'egories
    newsletter_subscription BOOLEAN DEFAULT FALSE
);

-- 2. Table Cat'egories
CREATE TABLE categories (
    category_id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
```

```
);

-- 3. Table Programmes
CREATE TABLE programs (
    program_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category_id INT REFERENCES categories(category_id) ON DELETE SET
        NULL
);

-- 4. Table Vid'eos
CREATE TABLE videos (
    video_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    duration INTERVAL NOT NULL,
    release_date DATE NOT NULL,
    country_of_origin VARCHAR(50),
    multi_language_available BOOLEAN DEFAULT FALSE,
    image_format VARCHAR(50),
    status VARCHAR(10) DEFAULT 'Available', -- 'Available' ou
        'Archived'
    program_id INT REFERENCES programs(program_id) ON DELETE CASCADE
);

-- 5. Table Vid'eos Favorites (Relation Plusieurs-'a-plusieurs entre
    Utilisateurs et Vid'eos)
CREATE TABLE favorite_videos (
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    video_id INT REFERENCES videos(video_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, video_id)
);

-- 6. Table Visionnage Vid'eo
CREATE TABLE video_viewings (
    viewing_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    video_id INT REFERENCES videos(video_id) ON DELETE CASCADE,
    viewing_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 7. Table Suggestions de Visionnage
CREATE TABLE viewing_suggestions (
    suggestion_id SERIAL PRIMARY KEY,
    generated_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    category_id INT REFERENCES categories(category_id) ON DELETE
        CASCADE,
    video_id INT REFERENCES videos(video_id) ON DELETE CASCADE
);

-- 8. Table Programmes Abonn'es (Relation Plusieurs-'a-plusieurs entre
    Utilisateurs et Programmes)
CREATE TABLE subscribed_programs (
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    program_id INT REFERENCES programs(program_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, program_id)
);
```

Après les avoir exécutées, pour voir la liste des tables, j'ai écrit la requête ci-dessous :

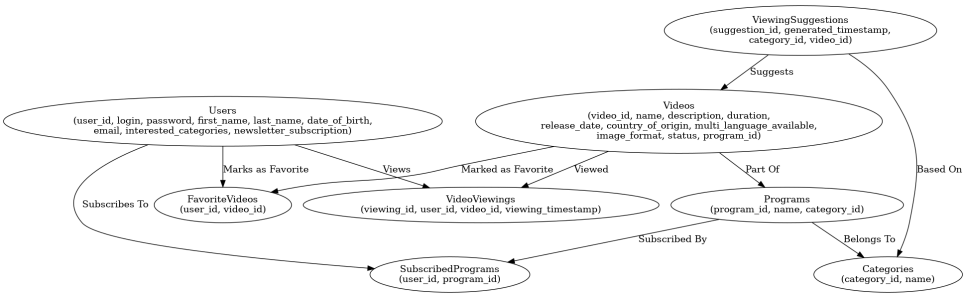
```sql
SELECT * FROM pg_catalog.pg_tables WHERE schemaname='public';
```

Illustration de la liste des tables :

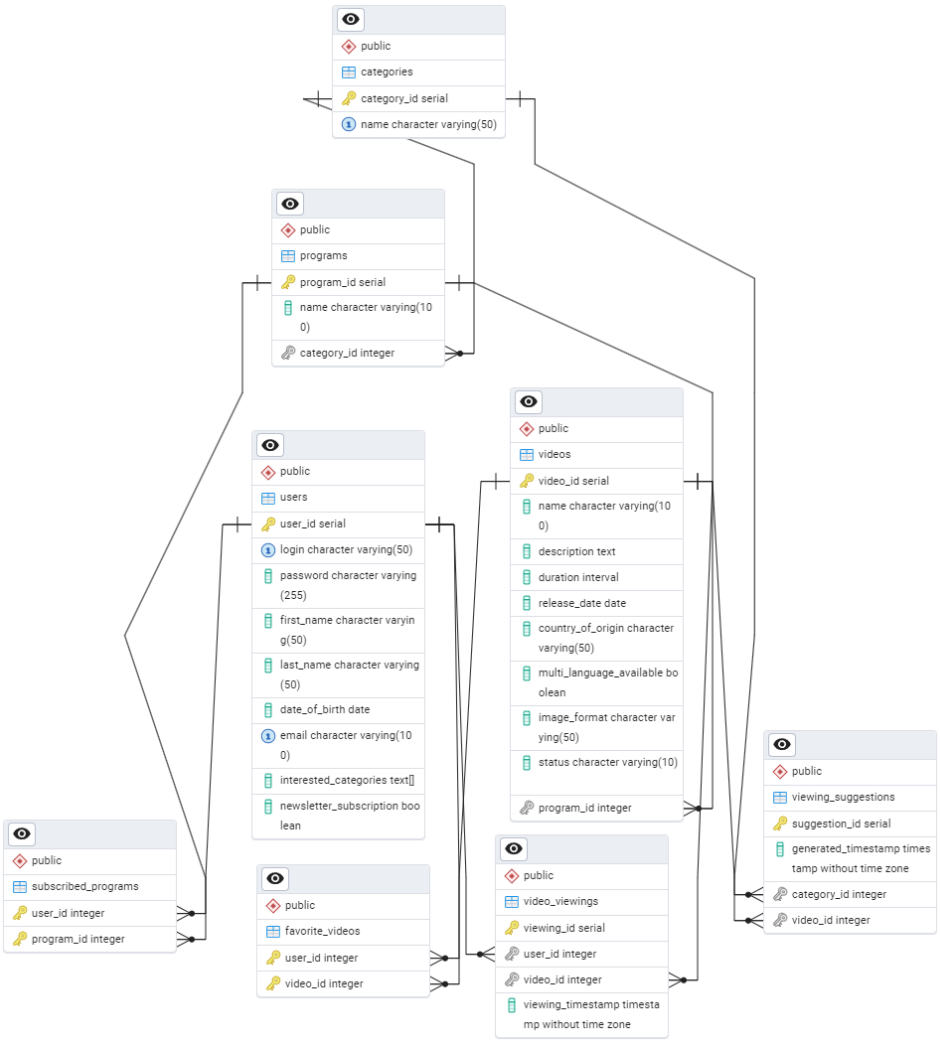
	schemaname name	tablename name	tableowner name	tablespace name	hasindexes boolean	hasrules boolean	hastriggers boolean	rowsecurity boolean
1	public	categories	postgres	[null]	true	false	true	false
2	public	programs	postgres	[null]	true	false	true	false
3	public	videos	postgres	[null]	true	false	true	false
4	public	users	postgres	[null]	true	false	true	false
5	public	favorite_videos	postgres	[null]	true	false	true	false
6	public	video_viewings	postgres	[null]	true	false	true	false
7	public	viewing_suggestions	postgres	[null]	true	false	true	false
8	public	subscribed_programs	postgres	[null]	true	false	true	false

Modèle Entité/Association

Diagramme E/A :



Le modèle ERD est :



Requêtes SQL

1. Nombre de vues des vidéos par catégorie, pour les vues datant de moins de deux semaines

```sql
SELECT
    c.name AS category_name,
    COUNT(vv.viewing_id) AS total_views
FROM
    video_viewings vv
JOIN
    videos v ON vv.video_id = v.video_id
JOIN
    programs p ON v.program_id = p.program_id
JOIN
    categories c ON p.category_id = c.category_id
WHERE
    vv.viewing_timestamp >= NOW() - INTERVAL '14 days'
GROUP BY
    c.name
ORDER BY
```

```
    total_views DESC;
```

2. Par utilisateur, le nombre d'abonnements, de favoris et de vidéos visionnées

```
SELECT
    u.user_id,
    u.first_name || ' ' || u.last_name AS user_name,
    COALESCE(subscriptions.count, 0) AS total_subscriptions,
    COALESCE(favorites.count, 0) AS total_favorites,
    COALESCE(viewings.count, 0) AS total_videos_viewed
FROM
    users u
LEFT JOIN
    (SELECT user_id, COUNT(*) AS count
     FROM subscribed_programs
     GROUP BY user_id) subscriptions
ON u.user_id = subscriptions.user_id
LEFT JOIN
    (SELECT user_id, COUNT(*) AS count
     FROM favorite_videos
     GROUP BY user_id) favorites
ON u.user_id = favorites.user_id
LEFT JOIN
    (SELECT user_id, COUNT(*) AS count
     FROM video_viewings
     GROUP BY user_id) viewings
ON u.user_id = viewings.user_id
ORDER BY
    u.user_id;
```

Ajout de la colonne pays à la table utilisateurs

```
ALTER TABLE users ADD COLUMN country VARCHAR(50);
```

3. Comparaison des vues des vidéos entre la France et l'Allemagne

```
SELECT
    v.video_id,
    v.name AS video_name,
    COALESCE(french_views.count, 0) AS french_views,
    COALESCE(german_views.count, 0) AS german_views,
    ABS(COALESCE(french_views.count, 0) - COALESCE(german_views.count,
        0)) AS view_difference
FROM
    videos v
LEFT JOIN
    (SELECT vv.video_id, COUNT(*) AS count
     FROM video_viewings vv
     JOIN users u ON vv.user_id = u.user_id
     WHERE u.country = 'France'
     GROUP BY vv.video_id) french_views
```

```
ON v.video_id = french_views.video_id
LEFT JOIN
    (SELECT vv.video_id, COUNT(*) AS count
     FROM video_viewings vv
     JOIN users u ON vv.user_id = u.user_id
     WHERE u.country = 'Germany'
     GROUP BY vv.video_id) german_views
ON v.video_id = german_views.video_id
ORDER BY
    ABS(COALESCE(french_views.count, 0) - COALESCE(german_views.count,
       0)) DESC;
```

4. Épisodes de programmes ayant au moins deux fois plus de spectateurs que la moyenne des autres épisodes du programme

```
SELECT
    v.video_id,
    v.name AS episode_name,
    v.program_id,
    p.name AS program_name,
    COUNT(vv.viewing_id) AS episode_viewers
FROM
    videos v
JOIN
    programs p ON v.program_id = p.program_id
JOIN
    video_viewings vv ON v.video_id = vv.video_id
GROUP BY
    v.video_id, v.name, v.program_id, p.name
HAVING
    COUNT(vv.viewing_id) >= 2 * (
        SELECT AVG(episode_views)
        FROM (
            SELECT
                COUNT(vv_inner.viewing_id) AS episode_views
            FROM
                videos v_inner
            JOIN
                video_viewings vv_inner ON v_inner.video_id =
                   vv_inner.video_id
            WHERE
                v_inner.program_id = v.program_id
            GROUP BY
                v_inner.video_id
        ) AS program_averages
    )
ORDER BY
    episode_viewers DESC;
```

5. Les 10 paires de vidéos apparaissant le plus souvent simultanément dans l'historique de visionnage d'un utilisateur

```
SELECT
```

```
    vh1.video_id AS video_1,
    vh2.video_id AS video_2,
    COUNT(*) AS pair_count
FROM
    video_viewings vh1
JOIN
    video_viewings vh2
ON
    vh1.user_id = vh2.user_id AND vh1.video_id < vh2.video_id
GROUP BY
    vh1.video_id, vh2.video_id
ORDER BY
    pair_count DESC
LIMIT 10;
```

Procédures et fonctions PL/SQL

1. Fonction pour convertir les informations d'une vidéo en format JSON

```
CREATE OR REPLACE FUNCTION get_video_json(video_id_input INT)
RETURNS JSON AS $$
DECLARE
    video_info JSON;
BEGIN
    SELECT
        JSON_BUILD_OBJECT(
            'video_id', v.video_id,
            'name', v.name,
            'description', v.description,
            'duration', v.duration,
            'release_date', v.release_date,
            'country_of_origin', v.country_of_origin,
            'multi_language_available', v.multi_language_available,
            'image_format', v.image_format,
            'status', v.status,
            'program', JSON_BUILD_OBJECT(
                'program_id', p.program_id,
                'program_name', p.name
            )
        )
    INTO video_info
    FROM videos v
    LEFT JOIN programs p ON v.program_id = p.program_id
    WHERE v.video_id = video_id_input;

    RETURN video_info;
END;
$$ LANGUAGE plpgsql;
```

Utilisation :

```
SELECT get_video_json(1);
```

Résultat :

```
{
    "video_id": 1,
    "name": "Vidéo Exemple",
    "description": "Ceci est une vidéo exemple.",
    "duration": "00:30:00",
    "release_date": "2024-01-01",
    "country_of_origin": "USA",
    "multi_language_available": true,
    "image_format": "HD",
    "status": "Available",
    "program": {
        "program_id": 10,
        "program_name": "Programme Exemple"
    }
}
```


2. Procédure pour générer un texte initial de newsletter avec les sorties de la semaine

```
CREATE OR REPLACE PROCEDURE generate_weekly_newsletter()
LANGUAGE plpgsql
AS $$
DECLARE
    current_week_start DATE := DATE_TRUNC('week', CURRENT_DATE); -- D'ebut de la semaine en cours
    current_week_end DATE := current_week_start + INTERVAL '6 days'; -- Fin de la semaine en cours
    weekly_releases TEXT := ''; -- Accumulateur pour les sorties de la semaine
    video_record RECORD;
BEGIN
    -- Parcourir toutes les sorties vid'eo de la semaine
    FOR video_record IN
        SELECT
            name,
            release_date,
            description
        FROM
            videos
        WHERE
            release_date BETWEEN current_week_start AND current_week_end
    LOOP
        -- Ajouter chaque vid'eo au texte de la newsletter
        weekly_releases := weekly_releases ||
        '$\bullet$ ' || video\_record.name || ' (Sortie le : ' || video\_record.release\_date || ') - ' ||
            video_record.description || E'\n';
    END LOOP;

    -- Afficher le texte final de la newsletter
    RAISE NOTICE 'Newsletter Hebdomadaire : %',
    E'Voici les nouvelles sorties de cette semaine :\n\n' || weekly_releases ||
```

```
    E'\nRestez \'a l'\,\'ecoute pour plus de mises \'a jour !';

END;
$$;
```

Utilisation :

```
CALL generate_weekly_newsletter();
```

Résultat :

Newsletter Hebdomadaire :
Voici les nouvelles sorties de cette semaine :

- Vidéo Exemple 1 (Sortie le : 2024-11-27) - Une aventure palpitante.
- Vidéo Exemple 2 (Sortie le : 2024-11-28) - Un drame réconfortant.

Restez à l'écoute pour plus de mises à jour !

3. Fonction pour recommander des vidéos populaires à un utilisateur

```
CREATE OR REPLACE FUNCTION recommend_videos(user_id_input INT)
RETURNS TABLE (
    video_id INT,
    video_name TEXT,
    description TEXT,
    release_date DATE,
    popularity_score INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        v.video_id,
        v.name AS video_name,
        v.description,
        v.release_date,
        COUNT(vv.viewing_id) AS popularity_score
    FROM
        videos v
    JOIN
        programs p ON v.program_id = p.program_id
    JOIN
        categories c ON p.category_id = c.category_id
    JOIN
        users u ON u.user_id = user_id_input
    LEFT JOIN
        video_viewings vv ON v.video_id = vv.video_id
    WHERE
        c.category_id = ANY(u.interested_categories) -- Correspond aux
            cat'egories suivies par l'utilisateur
        AND v.status = 'Available'
    GROUP BY
        v.video_id, v.name, v.description, v.release_date
    ORDER BY
        popularity_score DESC, v.release_date DESC
```

```
    LIMIT 10;
END;
$$ LANGUAGE plpgsql;
```

Utilisation :

```
SELECT * FROM recommend_videos(1);
```

Triggers

1. Limiter les favoris d'un utilisateur à 300 vidéos

Étape 1 : Fonction de trigger

```
CREATE OR REPLACE FUNCTION enforce_bookmark_limit()
RETURNS TRIGGER AS $$
BEGIN
    -- V'erifier le nombre actuel de favoris pour l'utilisateur
    IF (
        SELECT COUNT(*)
        FROM favorite_videos
        WHERE user_id = NEW.user_id
    ) >= 300 THEN
        -- Lever une erreur si la limite est d'epass'ee
        RAISE EXCEPTION 'L''utilisateur % ne peut pas marquer plus de
           300 vid'eos comme favorites.', NEW.user_id;
    END IF;

    -- Autoriser l'insertion si la limite n'est pas d'epass'ee
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Étape 2 : Création du trigger

```
CREATE TRIGGER check_bookmark_limit
BEFORE INSERT ON favorite_videos
FOR EACH ROW
EXECUTE FUNCTION enforce_bookmark_limit();
```

Test :

```
INSERT INTO favorite_videos (user_id, video_id)
VALUES (1, 101);
```

Si l'utilisateur a déjà 300 favoris, cette requête échouera avec :

```
ERREUR : L'utilisateur 1 ne peut pas marquer plus de 300 vidéos comme favorites.
```

2. Archiver une vidéo supprimée

Étape 1 : Création de la table d'archive

```
CREATE TABLE archived_videos (
    video_id INT PRIMARY KEY,
    name VARCHAR(100),
    description TEXT,
```

```
    duration INTERVAL,
    release_date DATE,
    country_of_origin VARCHAR(50),
    multi_language_available BOOLEAN,
    image_format VARCHAR(50),
    archived_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Étape 2 : Fonction de trigger

```
CREATE OR REPLACE FUNCTION archive_video_before_delete()
RETURNS TRIGGER AS $$
BEGIN
    -- Ins'erer les donn'ees de la vid'eo dans la table d'archive
    INSERT INTO archived_videos (
        video_id,
        name,
        description,
        duration,
        release_date,
        country_of_origin,
        multi_language_available,
        image_format
    )
    VALUES (
        OLD.video_id,
        OLD.name,
        OLD.description,
        OLD.duration,
        OLD.release_date,
        OLD.country_of_origin,
        OLD.multi_language_available,
        OLD.image_format
    );

    -- Autoriser la suppression
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

Étape 3 : Création du trigger

```
CREATE TRIGGER archive_video_on_delete
BEFORE DELETE ON videos
FOR EACH ROW
EXECUTE FUNCTION archive_video_before_delete();
```

Test :

```
-- Supprimer une vid'eo
DELETE FROM videos WHERE video_id = 1;

-- V'erifier si la vid'eo est archiv'ee
SELECT * FROM archived_videos WHERE video_id = 1;
```

3. Limiter les visionnages d'un utilisateur à 3 par minute

Étape 1 : Fonction de trigger

```sql
CREATE OR REPLACE FUNCTION enforce_viewing_limit()
RETURNS TRIGGER AS $$
DECLARE
    recent_view_count INT;
BEGIN
    -- Compter le nombre de visionnages par le m\^eme utilisateur dans
       la derni'ere minute
    SELECT COUNT(*) INTO recent_view_count
    FROM video_viewings
    WHERE user_id = NEW.user_id
      AND viewing_timestamp >= NOW() - INTERVAL '1 minute';

    -- Lever une exception si le nombre d'epasse 3
    IF recent_view_count >= 3 THEN
        RAISE EXCEPTION 'L''utilisateur % ne peut pas lancer plus de 3
           visionnages par minute.', NEW.user_id;
    END IF;

    -- Autoriser le nouveau visionnage si la limite n'est pas
       d'epass'ee
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Étape 2 : Création du trigger

```sql
CREATE TRIGGER check_viewing_limit
BEFORE INSERT ON video_viewings
FOR EACH ROW
EXECUTE FUNCTION enforce_viewing_limit();
```

Test :

```sql
-- Simuler trois visionnages en une minute pour un utilisateur
INSERT INTO video_viewings (user_id, video_id) VALUES (1, 101);
INSERT INTO video_viewings (user_id, video_id) VALUES (1, 102);
INSERT INTO video_viewings (user_id, video_id) VALUES (1, 103);

-- Tenter un quatri'eme visionnage dans la m\^eme minute
INSERT INTO video_viewings (user_id, video_id) VALUES (1, 104);
```

Résultat :

```
ERREUR : L'utilisateur 1 ne peut pas lancer plus de 3 visionnages par minute.
```

Indexes suggérés

1. Table Utilisateurs (users)

```sql
CREATE UNIQUE INDEX idx_users_login ON users(login);
CREATE UNIQUE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_interested_categories ON users USING
   gin(interested_categories);
```

2. Table Catégories (categories)

```sql
CREATE UNIQUE INDEX idx_categories_name ON categories(name);
```

3. Table Programmes (programs)

```sql
CREATE INDEX idx_programs_category_id ON programs(category_id);
```

4. Table Vidéos (videos)

```sql
CREATE INDEX idx_videos_program_id ON videos(program_id);
CREATE INDEX idx_videos_status ON videos(status);
CREATE INDEX idx_videos_release_date ON videos(release_date);
```

5. Table Vid'eos Favorites (favorite_videos)

```sql
CREATE UNIQUE INDEX idx_favorite_videos_user_video ON
    favorite_videos(user_id, video_id);
CREATE INDEX idx_favorite_videos_video_id ON favorite_videos(video_id);
```

6. Table Visionnage Vid'eo (video_viewings)

```sql
CREATE INDEX idx_video_viewings_user_timestamp ON
    video_viewings(user_id, viewing_timestamp);
CREATE INDEX idx_video_viewings_video_id ON video_viewings(video_id);
```

7. Table Suggestions de Visionnage (viewing_suggestions)

```sql
CREATE INDEX idx_viewing_suggestions_category_video ON
    viewing_suggestions(category_id, video_id);
```

8. Table Programmes Abonn'es (subscribed_programs)

```sql
CREATE UNIQUE INDEX idx_subscribed_programs_user_program ON
    subscribed_programs(user_id, program_id);
CREATE INDEX idx_subscribed_programs_program_id ON
    subscribed_programs(program_id);
```

Définitions des contraintes d'intégrité

1. Table Utilisateurs

```
CONSTRAINT unique_login UNIQUE (login),
CONSTRAINT unique_email UNIQUE (email),
CONSTRAINT not_null_login CHECK (login IS NOT NULL),
CONSTRAINT not_null_password CHECK (password IS NOT NULL),
CONSTRAINT valid_date_of_birth CHECK (date_of_birth <= CURRENT_DATE)
```

2. Table Catégories

```
CONSTRAINT unique_category_name UNIQUE (name),
CONSTRAINT not_null_category_name CHECK (name IS NOT NULL)
```

3. Table Programmes

```
CONSTRAINT not_null_program_name CHECK (name IS NOT NULL),
CONSTRAINT fk_program_category FOREIGN KEY (category_id) REFERENCES
    categories(category_id) ON DELETE SET NULL
```

4. Table Vidéos

```
CONSTRAINT not_null_video_name CHECK (name IS NOT NULL),
CONSTRAINT not_null_video_duration CHECK (duration IS NOT NULL),
CONSTRAINT not_null_video_release_date CHECK (release_date IS NOT
    NULL),
CONSTRAINT valid_status CHECK (status IN ('Available', 'Archived')),
CONSTRAINT valid_release_date CHECK (release_date <= CURRENT_DATE),
CONSTRAINT fk_video_program FOREIGN KEY (program_id) REFERENCES
    programs(program_id) ON DELETE CASCADE
```

5. Table Vidéos Favorites

```
CONSTRAINT pk_favorite_videos PRIMARY KEY (user_id, video_id),
CONSTRAINT fk_favorite_video_user FOREIGN KEY (user_id) REFERENCES
    users(user_id) ON DELETE CASCADE,
CONSTRAINT fk_favorite_video_video FOREIGN KEY (video_id) REFERENCES
    videos(video_id) ON DELETE CASCADE
```

6. Table Visionnage Vidéo

```
CONSTRAINT pk_video_viewings PRIMARY KEY (viewing_id),
CONSTRAINT fk_viewing_user FOREIGN KEY (user_id) REFERENCES
    users(user_id) ON DELETE CASCADE,
CONSTRAINT fk_viewing_video FOREIGN KEY (video_id) REFERENCES
    videos(video_id) ON DELETE CASCADE
```

7. Table Suggestions de Visionnage

```
CONSTRAINT pk_viewing_suggestions PRIMARY KEY (suggestion_id),
CONSTRAINT fk_suggestion_category FOREIGN KEY (category_id) REFERENCES
    categories(category_id) ON DELETE CASCADE,
CONSTRAINT fk_suggestion_video FOREIGN KEY (video_id) REFERENCES
    videos(video_id) ON DELETE CASCADE
```

8. Table Programmes Abonnés

```
CONSTRAINT pk_subscribed_programs PRIMARY KEY (user_id, program_id),
CONSTRAINT fk_subscribed_user FOREIGN KEY (user_id) REFERENCES
    users(user_id) ON DELETE CASCADE,
CONSTRAINT fk_subscribed_program FOREIGN KEY (program_id) REFERENCES
    programs(program_id) ON DELETE CASCADE
```