Cinéphoria

Transaction SQL

```sql
-- • La base de données à utiliser pour la transaction est définie.
USE cinephoriasqldb;

-- • La procédure est supprimée si elle existe déjà, afin de permettre une création propre.
DROP PROCEDURE IF EXISTS transaction_example_add_new_film;

-- • Un délimiteur est utilisé pour encadrer la définition de la procédure stockée.
DELIMITER //

-- • Une procédure stockée est créée pour ajouter un nouveau film.
CREATE PROCEDURE transaction_example_add_new_film(
    IN p_film_title VARCHAR(100),
    IN p_film_description VARCHAR(255),
    IN p_film_img VARCHAR(255),
    IN p_film_duration SMALLINT,
    IN p_film_favorite BOOLEAN,
    IN p_film_minimum_age TINYINT,
    IN p_film_active_date DATE
)

BEGIN
proc_transaction_example_add_new_film: BEGIN
    DECLARE film_count INT;
    DECLARE genre_count INT;
    DECLARE screening_count INT;
    DECLARE new_film_id INT;
    -- • Des variables sont déclarées pour compter les films, genres et séances, et pour stocker l'identifiant du nouveau film.
    START TRANSACTION;

    -- • Avant l'insertion, la procédure vérifie si un film portant le même titre existe. Si c'est le cas,
    --   la transaction est annulée et la procédure est quittée avec ROLLBACK.
    IF EXISTS (SELECT 1 FROM Film WHERE film_title = p_film_title) THEN
        ROLLBACK;
        SELECT 'Transaction rollback - Film already exists' AS result;
        LEAVE proc_transaction_example_add_new_film;
    END IF;

    -- • Les informations du film sont ensuite définies à partir des paramètres fournis.
    SET @film_title = p_film_title;
    SET @film_description = p_film_description;
    SET @film_img = p_film_img;
    SET @film_duration = p_film_duration;
    SET @film_favorite = p_film_favorite;
    SET @film_minimum_age = p_film_minimum_age;
    SET @film_active_date = p_film_active_date;

    -- • Le nouveau film est inséré avec le statut actif par défaut et sans notes initiales.
    INSERT INTO Film (film_title, film_description, film_img, film_duration, film_favorite, film_minimum_age, film_active_date, film_publishing_state, film_average_rating)
    VALUES (@film_title, @film_description, @film_img, @film_duration, @film_favorite, @film_minimum_age, @film_active_date, 'active', 0);
    -- • L'identifiant du dernier film inséré est récupéré pour les relations suivantes.
    SET @new_film_id = LAST_INSERT_ID();

    -- • Les genres sont associés au film en fonction des catégories existantes.
    INSERT INTO Genre_Film (genre_id, film_id)
    SELECT genre_id, @new_film_id
    FROM Genre
    WHERE genre_type IN ('Fantastique', 'Aventure'); -- • Supposer que ces genres existent (modifier en conséquence)

    -- • Les séances sont programmées pour le nouveau film dans les salles disponibles.
    INSERT INTO Screening (screening_date, screening_status, cinema_id, film_id, room_id)
    SELECT
        DATE_ADD(NOW(), INTERVAL 1 DAY) + INTERVAL (ROW_NUMBER() OVER (PARTITION BY Cinema.cinema_id ORDER BY Room.room_id) - 1) DAY AS screening_date,
        'active' AS screening_status,
        Cinema.cinema_id,
        @new_film_id,
        Room.room_id
    FROM Cinema
    JOIN Room ON Cinema.cinema_id = Room.cinema_id
    LIMIT 10; -- • Limiter à 10 projections pour la démonstration

    -- • Une vue est créée pour les détails du nouveau film.
    CREATE OR REPLACE VIEW v_new_film_details AS
    SELECT f.film_id, f.film_title, GROUP_CONCAT(g.genre_type) AS genres
    FROM Film f
    LEFT JOIN Genre_Film gf ON f.film_id = gF.film_id
    LEFT JOIN Genre g ON gf.genre_id = g.genre_id
    GROUP BY f.film_id;
    -- • Le résultat de cette vue peut être interrogé avec SELECT * FROM v_new_film_details;

    -- • Une vue est créée pour les projections du nouveau film.
    CREATE OR REPLACE VIEW v_new_film_screenings AS
    SELECT f.film_id, f.film_title, s.screening_id, s.screening_date, c.cinema_name, r.room_number
    FROM Film f
    LEFT JOIN Screening s ON f.film_id = s.film_id
    LEFT JOIN Cinema c ON s.cinema_id = c.cinema_id
    LEFT JOIN Room r ON s.room_id = r.room_id;
    -- • Le résultat de cette vue peut être interrogé avec SELECT * FROM v_new_film_screenings;

    -- • Les vues peuvent être consultées via des requêtes simples ou pour supprimer, décommenter les 2 lignes suivantes
    -- DROP VIEW IF EXISTS v_new_film_details;
    -- DROP VIEW IF EXISTS v_new_film_screenings;

    -- • Des compteurs sont utilisés pour valider que toutes les étapes de la transaction ont réussi.
    SET @film_count = (SELECT COUNT(*) FROM Film WHERE film_id = @new_film_id);
    SET @genre_count = (SELECT COUNT(*) FROM Genre_Film WHERE film_id = @new_film_id);
    SET @screening_count = (SELECT COUNT(*) FROM Screening WHERE film_id = @new_film_id);

    -- • La transaction est validée si toutes les opérations sont réussies, sinon elle est annulée avec ROLLBACK.
    IF @film_count = 1 AND @genre_count > 0 AND @screening_count > 0 THEN
        COMMIT;
        SELECT 'Transaction completed - New film and screenings successfully added' AS result;
    ELSE
        ROLLBACK;
        SELECT 'Transaction failed - The new film and screenings were not added' AS result;
    END IF;

    END proc_transaction_example_add_new_film;
    -- • Les variables temporaires sont nettoyées à la fin de la procédure.
    SET @film_title = NULL;
    SET @film_description = NULL;
    SET @film_img = NULL;
    SET @film_duration = NULL;
    SET @film_favorite = NULL;
    SET @film_minimum_age = NULL;
    SET @film_active_date = NULL;
END //

DELIMITER ;

-- • La procédure et transaction peut être appelée indépendamment avec différents paramètres pour ajouter d'autres films.
-- • Exemple d'appel de la procédure avec des détails de film spécifiques.
CALL transaction_example_add_new_film(
    'La Légende de la mers',
    'Une aventure épique au fond des océans pour toute la famille.',
    'assets/img/la_legende_de_la_mer.webp',
    95,
    FALSE,
    0,
    '2025-08-27'
);
```