ITuniversity

Historique

On va mettre en place un système pour tracer l'historique de changement de champ.

ITuniversity

Création de la table Historique

```
CREATE TABLE film_rental_rate_history (
    history_id      SERIAL PRIMARY KEY,
    film_id         INTEGER NOT NULL,
    old_rate        NUMERIC(4,2),
    new_rate        NUMERIC(4,2),
    changed_at      TIMESTAMP DEFAULT NOW(),
    changed_by      TEXT DEFAULT CURRENT_USER
);
```

ITuniversity

Fonction Trigger : détecter les changements.

La fonction doit :

1. Vérifier si rental_rate a changé
2. Enregistrer l'ancienne et la nouvelle valeur
3. L'insérer dans la table historique
4. Retourner NEW

ITuniversity

Fonction trigger : code

```
CREATE OR REPLACE FUNCTION log_rental_rate_changes()
RETURNS trigger AS $$
BEGIN
    -- Vérifier si rental_rate a changé
    IF NEW.rental_rate IS DISTINCT FROM OLD.rental_rate THEN
        INSERT INTO film_rental_rate_history(
            film_id, old_rate, new_rate, changed_at, changed_by
        )
        VALUES (
            OLD.film_id,
            OLD.rental_rate,
            NEW.rental_rate,
            NOW(),
            CURRENT_USER
        );
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

ITuniversity

Fonction trigger : code

On applique ce trigger à chaque UPDATE de la table Film

```
CREATE TRIGGER trg_log_rental_rate
AFTER UPDATE ON film
FOR EACH ROW
EXECUTE FUNCTION log_rental_rate_changes();
```

Pourquoi AFTER UPDATE ?

- la row NEW est validée
- insertion dans l'historique ne bloque pas la modification
- logique métier cohérente

ITuniversity

Fonction trigger : exemple d'utilisation

On modifie le prix d'un film :

```
UPDATE film
SET rental_rate = 4.99
WHERE film_id = 12;
```

On regarde dans la table historique

```
SELECT * FROM film_rental_rate_history
WHERE film_id = 12
ORDER BY changed_at DESC;
```

ITuniversity

TP

Rajouter un autre trigger pour un autre champ