P5 - Vistas y Disparadores

Fecha: 26 de octubre de 2025
Autor: alu0101474311@ull.edu.es (Tomás Pino Pérez)

Comandos útiles

```
# Restaurar backup en tar
sudo -u postgres createuser cliente
sudo -u postgres pg_restore --create -d postgres -F t
AlquilerPractica.tar

# Exportar base de datos a tar
sudo -u postgres pg_dump -F t alquilerdvd > alquilerdvd.tar
```

Vistas de las distintas consultas

```
-- view_4a_ventas_por_categoria
CREATE OR REPLACE VIEW view_4a_ventas_por_categoria AS
SELECT c.name AS categoria,
sum(p.amount) AS total_ventas
FROM (((((payment p
JOIN rental r ON ((p.rental_id = r.rental_id)))
JOIN inventory i ON ((r.inventory_id = i.inventory_id)))
JOIN film f ON ((i.film_id = f.film_id)))
JOIN film_category fc ON ((f.film_id = fc.film_id)))
JOIN category c ON ((fc.category_id = c.category_id)))
GROUP BY c.name
ORDER BY (sum(p.amount)) DESC;

-- view_4b_ventas_totales_ciudad_pais_encargado
CREATE OR REPLACE VIEW view_4b_ventas_totales_ciudad_pais_encargado
AS
SELECT s.store_id,
(((city.city)::text || ', '::text) || (country.country)::text) AS
ciudad_pais,
(((mgr.first_name)::text || ' '::text) || (mgr.last_name)::text) AS
encargado,
sum(p.amount) AS total_ventas
FROM (((((((payment p
```

```
JOIN rental r ON ((p.rental_id = r.rental_id)))
JOIN inventory i ON ((r.inventory_id = i.inventory_id)))
JOIN store s ON ((i.store_id = s.store_id)))
JOIN staff mgr ON ((s.manager_staff_id = mgr.staff_id)))
JOIN address a ON ((s.address_id = a.address_id)))
JOIN city ON ((a.city_id = city.city_id)))
JOIN country ON ((city.country_id = country.country_id)))
GROUP BY s.store_id, city.city, country.country, mgr.first_name,
mgr.last_name
ORDER BY (sum(p.amount)) DESC;

-- view_4c_peliculas_categorias_actores
CREATE OR REPLACE VIEW view_4c_peliculas_categorias_actores AS
SELECT f.film_id,
f.title,
f.description,
string_agg(DISTINCT (c.name)::text, ', '::text) AS categorias,
f.rental_rate AS precio,
f.length AS duracion,
f.rating AS clasificacion,
string_agg(DISTINCT (((a.first_name)::text || ' '::text) ||
(a.last_name)::text), ', '::text) AS actores
FROM ((((film f
LEFT JOIN film_category fc ON ((f.film_id = fc.film_id)))
LEFT JOIN category c ON ((fc.category_id = c.category_id)))
LEFT JOIN film_actor fa ON ((f.film_id = fa.film_id)))
LEFT JOIN actor a ON ((fa.actor_id = a.actor_id)))
GROUP BY f.film_id, f.title, f.description, f.rental_rate, f.length,
f.rating;

-- view_4d_actores_categorias_peliculas_concatenadas
CREATE OR REPLACE VIEW
view_4d_actores_categorias_peliculas_concatenadas AS
SELECT a.actor_id,
a.first_name,
a.last_name,
string_agg(DISTINCT (c.name)::text, ':'::text ORDER BY
(c.name)::text) AS categorias,
string_agg(DISTINCT (f.title)::text, ':'::text ORDER BY
(f.title)::text) AS peliculas
FROM ((((actor a
JOIN film_actor fa ON ((a.actor_id = fa.actor_id)))
JOIN film f ON ((fa.film_id = f.film_id)))
LEFT JOIN film_category fc ON ((f.film_id = fc.film_id)))
LEFT JOIN category c ON ((fc.category_id = c.category_id)))
GROUP BY a.actor_id, a.first_name, a.last_name;
```

Restricciones `CHECK`

```
-- En tabla payment: el importe no puede ser negativo
ALTER TABLE payment
ADD CONSTRAINT chk_payment_amount_positive CHECK (amount ⩾ 0);

-- En tabla rental: la fecha de devolución no puede ser anterior a la
fecha de alquiler
ALTER TABLE rental
ADD CONSTRAINT chk_return_date_valid CHECK (return_date IS NULL OR
return_date ⩾ rental_date);

-- En tabla film: la duración del alquiler debe estar entre 1 y 30
días
ALTER TABLE film
ADD CONSTRAINT chk_rental_duration CHECK (rental_duration BETWEEN 1
AND 30);

-- En tabla film: el coste de reemplazo debe ser mayor que cero
ALTER TABLE film
ADD CONSTRAINT chk_replacement_cost_positive CHECK (replacement_cost
> 0);

-- En tabla film: la tarifa de alquiler debe ser mayor que cero
ALTER TABLE film
ADD CONSTRAINT chk_rental_rate_positive CHECK (rental_rate > 0);

-- En tabla customer: el campo active solo puede ser 0 o 1
ALTER TABLE customer
ADD CONSTRAINT chk_customer_active CHECK (active IN (0, 1));

-- En tabla staff: el email debe tener un formato válido
ALTER TABLE staff
ADD CONSTRAINT chk_staff_email_format CHECK (email ~* '^[A-Za-z0-
9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

-- En tabla customer: el email debe tener un formato válido si no es
nulo
ALTER TABLE customer
ADD CONSTRAINT chk_customer_email_format CHECK (email IS NULL OR
email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
```

Explicación Trigger en la tabla `costumer`

```
-- Trigger a analizar
last_updated BEFORE UPDATE ON customer
FOR EACH ROW EXECUTE PROCEDURE last_updated();
```

Explicación:

- `BEFORE UPDATE`: se ejecuta antes de cualquier actualización de un registro.
- `FOR EACH ROW`: se aplica a cada fila afectada, no a toda la tabla.
- `EXECUTE PROCEDURE last_updated()`: llama a la función `last_updated()`, que típicamente actualiza la columna `last_update` con la fecha y hora actuales.
- Propósito: mantener un registro automático de cuándo se modificó cada fila.

Tablas con solución similar:
Generalmente todas las tablas que tienen la columna `last_update` usan este mismo trigger. En tu base de datos `alquilerdvd` se utiliza en: `actor`, `address`, `category`, `city`, `country`, `film`, `film_actor`, `film_category`, `inventory`, `language`, `rental`, `staff`, `store`.

```
-- Consulta para verificar lo anterior
-- Todas las tablas listadas mostrarán un trigger parecido a
`last_updated`
SELECT event_object_table AS tabla, trigger_name AS trigger
FROM information_schema.triggers
WHERE trigger_name LIKE '%last_updated%';
```

Disparador cuando se insertó registro en tabla `film`

```
-- Crear tabla para logs de inserts en film
CREATE TABLE film_insert_log (
film_id INT,
insert_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Función para trigger de insert
CREATE OR REPLACE FUNCTION log_film_insert()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO film_insert_log(film_id, insert_date)
VALUES (NEW.film_id, CURRENT_TIMESTAMP);
RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

-- Trigger para insert
CREATE TRIGGER trigger_film_insert
AFTER INSERT ON film
FOR EACH ROW
EXECUTE FUNCTION log_film_insert();
```

Disparador cuando se eliminó registro en tabla `film`

```
-- Crear tabla para logs de deletes en film
CREATE TABLE film_delete_log (
film_id INT,
delete_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Función para trigger de delete
CREATE OR REPLACE FUNCTION log_film_delete()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO film_delete_log(film_id, delete_date)
VALUES (OLD.film_id, CURRENT_TIMESTAMP);
RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Trigger para delete
CREATE TRIGGER trigger_film_delete
AFTER DELETE ON film
FOR EACH ROW
EXECUTE FUNCTION log_film_delete();
```

Significado y relevancia de las secuencias

- Las secuencias (`sequence`) generan valores únicos incrementales, comúnmente usados como `serial`/`identity` para claves primarias.
- Funciones clave: `nextval('seq')` obtiene y avanza valor; `currval('seq')` obtiene el último valor en la sesión; `setval('seq', n)` fija el valor.

- Relevancia: garantizan unicidad sin bloqueos extensos y persisten su estado entre transacciones.
- Riesgo: saltos de números si la transacción falla o si se hace `nextval` varias veces. No garantizan que no falten números, solo unicidad y orden creciente.