Práctica 29

Herencia, reglas y particiones

Objetivo

*En esta práctica se definen reglas y se muestra el empleo de ellas definiendo particiones mediante herencia, empleando el software **psql**.*

Introducción

Una forma sencilla de proteger la información contenida en una tabla, ante la posibilidad de que alguien pueda insertar, actualizar o eliminar datos de manera indebida es mediante la definición de reglas.

Una regla es una definición de una acción alternativa que debe realizarse al intentase una modificación sobre una tabla o vista, reemplazando el comando original por otro.

El sistema de reglas se sitúa entre el traductor de la consulta y el optimizador. Toma la salida del traductor, un árbol de la consulta, y las reglas de reescritura contenidas en el catálogo ***pg_rewrite***, y crea cero o muchos árboles de consulta como resultado.

Las particiones de tablas se pueden implementar empleando la herencia de tablas, lo que ofrece varias ventajas que no existen en el particionamiento declarativo:

- En primer lugar, con la herencia de tablas, las tablas hijas pueden tener columnas adicionales a las que existen en la tabla padre.
- La herencia de tablas permite la herencia múltiple.
- La herencia de tablas permite dividir los datos de una manera definida por el usuario.

Equipo necesario

Una computadora con sistema operativo ***Windows*** que cuente con ***psql***.

Metodología

1. Ejecute la herramienta ***SQL Shell (psql)***.

2. Deje el usuario ***postgres*** y proporcione la contraseña definida en la primera práctica.
3. Cambie a la base de datos Biblioteca:

```
postgres=# \c Biblioteca
```

4. Como primer ejemplo, se van a crear tres reglas que impiden cualquier modificación a la tabla ***Libros***:

```
Biblioteca=# CREATE RULE r1 AS ON INSERT TO Libros DO INSTEAD
NOTHING;
Biblioteca=# CREATE RULE r2 AS ON UPDATE TO Libros DO INSTEAD
NOTHING;
Biblioteca=# CREATE RULE r3 AS ON DELETE TO Libros DO INSTEAD
NOTHING;
```

5. Una vez creadas estas reglas, ejecute cada una de las siguientes sentencias:

```
Biblioteca=# INSERT INTO Libros VALUES ('978-84-9825-335-1',
'La Odisea', 'Homero Simpson', 'Anagrama', 2009, 250,
'Español');
Biblioteca=# UPDATE Libros SET Precio=100 WHERE
Precio='$10.00';
Biblioteca=# DELETE FROM Libros WHERE "ISBN" LIKE '9%';
```

6. Obtenga una imagen completa de la pantalla, mostrando la creación de las tres reglas y la ejecución de las tres sentencias, consérvela como evidencia.
7. Se podrá ver que en los tres casos se ejecutaron las sentencias sin error, pero indicando que hubo cero cambios,
8. Consulte la tabla ***Libros***, para confirmarlo:

```
Biblioteca=# SELECT * FROM Libros;
```

9. Una forma más interesante para utilizar el sistema de reglas consiste en crear una regla que registre los datos del usuario y el momento en que se realizó el cambio de precio de un libro, para ello se utilizará una tabla de ***registro***, de manera similar a un ***trigger***.

```
Biblioteca=# CREATE TABLE registro (isbn CHAR(17), anterior
text, nuevo text, persona text, momento timestamp);
Biblioteca=# CREATE RULE cambio AS ON UPDATE TO Libros WHERE
NEW.Precio!= OLD.Precio DO INSERT INTO registro VALUES
(NEW."ISBN", OLD.Precio, NEW.Precio,
getpgusername(),current_timestamp);
```

10. Obtenga una imagen completa de la pantalla, mostrando la creación de la tabla y la nueva regla, consérvela como evidencia.
11. Para verificar el funcionamiento de la regla ***cambio***, se deberá eliminar la regla ***r2***:

```
Biblioteca=# DROP RULE r2 on Libros;
```

12. Ahora repita la sentencia de actualización anterior:

```
Biblioteca=# UPDATE Libros SET Precio=100 WHERE
Precio='$10.00';
```

13. En este caso si se realizó el cambio de precios, como se puede verificar:

```
Biblioteca=# SELECT * FROM Libros;
```

14. Y también se generaron los respectivos registros:

```
Biblioteca=# SELECT * FROM registro;
```

15. Obtenga una imagen completa de la pantalla, mostrando el resultado de ambas consultas, consérvela como evidencia.

16. Ahora, se pide crear la base de datos ***poblacion***:

```
postgres=# CREATE DATABASE poblacion;
```

17. Se selecciona la base de datos ***poblacion***:

```
postgres=# \c poblacion
```

18. Dentro de ella cree la siguiente tabla llamada ***ciudades***:

```
poblacion=# CREATE TABLE ciudades (Nombre VARCHAR(25),
población INT, altitud INT);
```

19. Ahora cree la tabla llamada ***capitales***, que hereda los atributos de **ciudades**:

```
poblacion=# CREATE TABLE capitales (Estado VARCHAR(25))
INHERITS (ciudades);
```

20. Ingrese algunos valores a la tabla ***ciudades***:

```
poblacion=# INSERT INTO ciudades VALUES ('Zacapu', 55287,
2002);
poblacion=# INSERT INTO ciudades VALUES ('Uruapan', 299523,
1620);
poblacion=# INSERT INTO ciudades VALUES ('Morelia', 743275,
1920);
poblacion=# INSERT INTO ciudades VALUES ('Mérida', 921771, 14);
poblacion=# INSERT INTO ciudades VALUES ('Acapulco', 658609,
18);
```

21. Ahora ingrese algunos valores a la tabla ***capitales***, que también se registrarán como ciudades.

```
poblacion=# INSERT INTO capitales VALUES ('Guadalajara',
1385621, 1566, 'Jalisco');
poblacion=# INSERT INTO capitales VALUES ('Zacatecas', 1622138,
2061,'Zacatecas');
```

22. Consulte las tablas de ciudades y de capitales:

```
poblacion=# SELECT * FROM ciudades;
```

```
poblacion=# SELECT * FROM capitales;
```

23. Obtenga una imagen completa de la pantalla, mostrando el resultado de ambas consultas, consérvela como evidencia.

24. Ni Morelia ni Mérida se registraron como capitales de sus respectivos estados, para corregir este error, deberá eliminar de ***ciudades*** a ambos registros y agregarlos a la tabla de ***capitales***:

```
poblacion=# DELETE FROM ciudades WHERE nombre='Morelia' OR
nombre='Mérida';
poblacion=# INSERT INTO capitales VALUES ('Morelia', 743275,
1920, 'Michoacán'), ('Mérida', 921771, 14, 'Yucatán');
```

25. Si en la consulta a la tabla ciudades se añade la declaración ONLY, mostrará a las ciudades que no son capitales:

```
poblacion=# SELECT * FROM ONLY ciudades;
```

26. En la siguiente consulta, se agrega una columna que indica la procedencia del registro mostrado:

```
poblacion=# SELECT c.nombre, c.población, c.altitud, p.relname
FROM ciudades c INNER JOIN pg_class p ON c.tableoid = p.oid;
```

27. Obtenga una imagen completa de la pantalla, mostrando el resultado de ambas consultas, consérvela como evidencia.

28. Ahora, debe crear la tabla ***Maestra*** llamada ***alcalde***, esta tabla no deberá contener datos, así como tampoco ningún tipo de restricción:

```
poblacion=# CREATE TABLE alcalde(ciudad VARCHAR(25), apellido
VARCHAR(25), partido VARCHAR(6), edad INTEGER, sexo CHAR(1),
fecha_inicio DATE);
```

29. Y ahora proceda a crear las tablas hijas, que para este ejemplo serán cuatro, correspondientes a los cuatro partidos políticos a los que pertenecen los alcaldes, para ello se heredan los atributos de la tabla alcalde, pero además se agrega una condición CHECK.

```
poblacion=# CREATE TABLE alcalde_ppr (CHECK (partido='PPR'))
INHERITS(alcalde);
poblacion=# CREATE TABLE alcalde_pdd (CHECK (partido='PDD'))
INHERITS (alcalde);
poblacion=# CREATE TABLE alcalde_pon (CHECK (partido='PON'))
INHERITS (alcalde);
poblacion=# CREATE TABLE alcalde_pin (CHECK (partido='PIN'))
INHERITS (alcalde);
```

30. El siguiente paso es crear las llaves primarias para cada una de las tablas hijas:

```
poblacion=# ALTER TABLE alcalde_ppr ADD PRIMARY KEY (ciudad);
poblacion=# ALTER TABLE alcalde_pdd ADD PRIMARY KEY (ciudad);
poblacion=# ALTER TABLE alcalde_pon ADD PRIMARY KEY (ciudad);
```

```
poblacion=# ALTER TABLE alcalde_pin ADD PRIMARY KEY (ciudad);
```

31. Obtenga una imagen completa de la pantalla, mostrando todas estas sentencias, consérvela como evidencia.

32. A continuación, se deben crear las reglas que garanticen el llenado en cascada de los datos, de tal forma que cuando se ingrese un registro en la tabla Maestra, se redirijan a la tabla hija que le corresponda según el partido político.

```
poblacion=# CREATE RULE inserta_ppr AS ON INSERT TO alcalde
WHERE (partido='PPR') DO INSTEAD INSERT INTO alcalde_ppr VALUES
(NEW.ciudad, NEW.apellido, NEW.partido, NEW.edad, NEW.sexo,
NEW.fecha_inicio);
poblacion=# CREATE RULE inserta_pdd AS ON INSERT TO alcalde
WHERE (partido='PDD') DO INSTEAD INSERT INTO alcalde_pdd VALUES
(NEW.ciudad, NEW.apellido, NEW.partido, NEW.edad, NEW.sexo,
NEW.fecha_inicio);
poblacion=# CREATE RULE inserta_pon AS ON INSERT TO alcalde
WHERE (partido='PON') DO INSTEAD INSERT INTO alcalde_pon VALUES
(NEW.ciudad, NEW.apellido, NEW.partido, NEW.edad, NEW.sexo,
NEW.fecha_inicio);
poblacion=# CREATE RULE inserta_pin AS ON INSERT TO alcalde
WHERE (partido='PIN') DO INSTEAD INSERT INTO alcalde_pin VALUES
(NEW.ciudad, NEW.apellido, NEW.partido, NEW.edad, NEW.sexo,
NEW.fecha_inicio);
```

33. Obtenga una imagen completa de la pantalla, mostrando la creación de estas reglas, consérvela como evidencia.

34. Ahora proceda a insertar algunos registros en la tabla producción:

```
poblacion=# INSERT INTO alcalde VALUES ('Morelia', 'Martínez',
'PIN', 34, 'M', '01/11/2021');
poblacion=# INSERT INTO alcalde VALUES ('Acapulco', 'Pérez',
'PDD', 44, 'F', '01/10/2022');
poblacion=# INSERT INTO alcalde VALUES ('Mérida', 'García',
'PON', 59, 'M', '15/09/2021');
poblacion=# INSERT INTO alcalde VALUES ('Zacapu', 'Delgado',
'PIN', 29,'F','15/11/2021');
poblacion=# INSERT INTO alcalde VALUES ('Zacatecas', 'Morales',
'PPR', 39, 'M', '01/09/2022');
```

35. Obtenga una imagen completa de la pantalla, mostrando la ejecución de estas sentencias de inserción, consérvela como evidencia.

36. Y ahora ejecute las siguientes consultas:

```
poblacion=# SELECT * FROM alcalde;
poblacion=# SELECT * FROM alcalde_ppr;
poblacion=# SELECT * FROM alcalde_pdd;
```

```
poblacion=# SELECT * FROM alcalde_pon;
poblacion=# SELECT * FROM alcalde_pin;
```

37. Obtenga una imagen completa de la pantalla, mostrando la ejecución de estas consultas, consérvela como evidencia.
38. Ahora descarte la herencia de la tabla ***alcalde_ppr***.

```
poblacion=# ALTER TABLE alcalde_ppr NO INHERIT alcalde;
```

39. Ahora inserte un registro de un alcalde del partido ***PPR***:

```
poblacion=# INSERT INTO alcalde VALUES ('Guadalajara',
'Moreno', 'PPR', 31, 'F','01/09/2023');
```

40. Repita las siguientes consultas para ver el resultado de eliminar la herencia:

```
poblacion=# SELECT * FROM alcalde;
poblacion=# SELECT * FROM alcalde_ppr;
```

41. Ahora elimine a la tabla hija ***alcalde_pdd***.

```
poblacion=# DROP TABLE alcalde_pdd CASCADE;
```

42. Ahora inserte un registro de un alcalde del partido ***PDD***:

```
poblacion=# INSERT INTO alcalde VALUES ('Uruapan', 'Suárez',
'PDD', 41, 'M', '01/10/2023');
```

43. Finalmente ejecute la siguiente consulta:

```
poblacion=# SELECT * FROM alcalde;
```

44. Obtenga una imagen completa de la pantalla, mostrando la ejecución de estas consultas, consérvela como evidencia.
45. Cierre la consola del ***psql***.
46. Por último, en la carpeta DATA genere un archivo llamado ***poblacion.sql*** con la base de datos ***poblacion*** completa, desde la ventana de comandos, mediante el comando ***pg_dump***, de esta manera:

```
C:\Program Files\PostgreSQL\16\bin>pg_dump -U postgres -f
"C:\DATA\poblacion.sql" poblacion
```

47. Proporcione su contraseña y oprima **enter**.
48. Cierre la ventana de comandos.
49. Anexe a su reporte el archivo ***poblacion.sql*** como evidencia.
50. Fin de la Práctica

Evidencias

El alumno deberá enviar al instructor las evidencias requeridas durante la realización de la práctica.

Sugerencias didácticas

El instructor deberá atender a los alumnos que tengan dificultades en la interpretación y la realización de las instrucciones de la práctica.

Resultados

Se aprendió a crear funciones y definir reglas utilizando la herramienta ***psql***.

Bibliografía

- http://es.tldp.org/Postgresql-es/web/navegable/user/sql-createrule.html