Práctica 23

Procedimientos Almacenados en PostgreSQL

Objetivo

En esta práctica se introduce la forma de elaborar programas ejecutables, llamados procedimientos, la forma de elaboración y ejecución, a través de diversos ejemplos.

Introducción

Un procedimiento almacenado es un conjunto de comandos SQL que pueden guardarse en el servidor para que los usuarios puedan utilizarlo, refiriéndose al procedimiento almacenado por su nombre.

Los procedimientos almacenados pueden ser particularmente útiles cuando la seguridad es muy importante. Los bancos, por ejemplo, usan procedimientos almacenados para todas las operaciones comunes. Esto proporciona un entorno seguro y consistente, y los procedimientos pueden asegurar que cada operación se ejecuta apropiadamente, ya que los usuarios no tendrían acceso directo a las tablas de la base de datos.

Los procedimientos almacenados pueden mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. Esto le permite al administrador tener bibliotecas de procedimientos y de funciones en el servidor de base de datos. Los procedimientos almacenados se crean con comandos CREATE PROCEDURE. Un procedimiento se invoca usando un comando CALL, y sólo puede pasar valores usando variables de salida.

Para crear un procedimiento, es necesario tener el permiso CREATE ROUTINE, y los permisos ALTER ROUTINE y EXECUTE se asignan automáticamente a su creador.

Los procedimientos almacenados se asocian con una base de datos. Los procedimientos almacenados pueden llamar otros procedimientos almacenados.

Cuando se elimina una base de datos, todos los procedimientos almacenados asociados con ella también se eliminan.

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. En esta práctica se van a definir los procedimientos necesarios para registrar movimientos en las cuentas bancarias por parte de los clientes:

```
postgres=# \c banco
```

4. Ahora proceda a transcribir el siguiente procedimiento, el cual incluye una sentencia SQL para registrar a un nuevo cliente:

```
banco=# CREATE PROCEDURE ctenuevo(a integer, b varchar(15),
banco(# c varchar(15), d varchar(12))
banco-# LANGUAGE SQL
banco-# AS $$
banco$# INSERT INTO cliente VALUES (a, b, c, d);
banco$# $$;
```

5. Obtenga una imagen completa de la pantalla, mostrando la creación del procedimiento y consérvela como evidencia.
6. Ahora pruebe el procedimiento ***ctenuevo*** para insertar dos clientes nuevos mediante las siguientes llamadas:

```
banco=# SELECT * FROM cliente;
Banco=# CALL ctenuevo(31,'Laura','Ramos','443 345 9823');
Banco=# CALL ctenuevo(52,'Miguel', 'Flores','443 331 1189');
```

7. Vuelva a ejecutar la consulta para confirmar las inserciones de clientes mediante el procedimiento:

```
banco=# SELECT * FROM cliente;
```

8. Obtenga una imagen completa de la pantalla, mostrando la ejecución del procedimiento y la consulta a la tabla de clientes, y consérvela como evidencia.
9. A continuación, se deberá crear un procedimiento para dar de alta una cuenta:

```
banco=# CREATE OR REPLACE PROCEDURE Creacuenta(num INTEGER,
banco(# cte INTEGER, suc INTEGER)
banco-# LANGUAGE plpgsql
banco-# AS $$
banco$# BEGIN
banco$#    INSERT INTO cuentas VALUES (num, cte, suc, 0);
banco$#    EXCEPTION WHEN OTHERS THEN
```

```
banco$#    RAISE NOTICE 'Algún dato es erróneo';
banco$# END;
banco$# $$;
```

10. Obtenga una imagen completa de la pantalla, mostrando la creación del procedimiento y consérvela como evidencia.
11. Pruebe el procedimiento ejecutando las siguientes sentencias:

```
banco=# SELECT * FROM cuentas;
banco=# CALL Creacuenta(6127,52,183);
banco=# CALL Creacuenta(3491,31,341);
```

12. Este procedimiento no permite la inserción de una cuenta duplicada y provoca que se mande una notificación.

```
NOTICE:  Algún dato es erróneo
```

13. Vuelva a ejecutar la consulta para confirmar que sólo se hizo la inserción de la cuenta válida:

```
banco=# SELECT * FROM cuentas;
```

14. Obtenga una imagen completa de la pantalla, mostrando la ejecución del procedimiento y la consulta a la tabla de cuentas, y consérvela como evidencia.
15. A continuación, el alumno deberá crear una tabla llamada ***empleado*** con los atributos siguientes: ***Clave*** (entero, llave principal), ***Nombre*** (varchar de 25), ***Apellido_Pat*** (varchar de 25), ***Apellido_Mat*** (varchar de 25), ***Teléfono*** (varchar de 12), ***Sueldo*** (Money).
16. Verifique que se creó correctamente la tabla:

```
banco=# \d empleado
```

17. Obtenga una imagen completa de la pantalla, mostrando la estructura de la tabla empleado y consérvela como evidencia.
18. Ahora proceda a transcribir el siguiente procedimiento, el cual incluye una sentencia SQL para registrar a un nuevo cliente:

```
banco=# CREATE PROCEDURE emp_alta(Cve integer, Nom varchar(25),
banco(# ApP varchar(25), ApM varchar(25), Tel varchar(12),
banco(# Sue integer)
banco-# LANGUAGE SQL
banco-# AS $$
banco$#    INSERT INTO empleado VALUES (Cve, Nom, ApP, ApM,
Tel, Sue);
banco$# $$;
```

19. Obtenga una imagen completa de la pantalla, mostrando la creación del procedimiento y consérvela como evidencia.
20. Ahora, inserte los datos de los ***empleados*** que se muestran en la siguiente tabla, usando el procedimiento anterior:

Clave	Nombre	Apellido_Pat	Apellido_Mat	Teléfono	Sueldo
443	Lucas	Montes	Torres	443 312 1754	5500
512	Martha	Mares	Rosas	443 324 1689	6700
517	Karla	Cuevas	Mina	443 212 4317	8200
597	Alberto	Vidal	Luna	443 331 6536	7700

21. Revise lo capturado con:

```
banco=# SELECT * FROM empleado;
```

22. Obtenga una imagen completa de la pantalla, mostrando la ejecución del procedimiento y la consulta de la tabla, consérvela como evidencia.

23. Ahora, el alumno deberá crear una tabla llamada ***movimientos***, que contenga estos campos: ***Folio*** (serial, llave principal), ***Fecha*** (date), ***Cuenta*** (int), ***Empleado*** (int), ***Importe*** (numeric), ***Tipo*** (char(1)).

24. A continuación, deberá crear una restricción de llave ajena llamada ***fk_cuenta*** que vincule el atributo ***Cuenta*** con el ***Número*** de la tabla ***cuentas***, permitiendo actualización en cascada.

25. Y ahora, debe crear otra restricción de llave ajena llamada ***fk_empleado*** que vincule el atributo ***Empleado*** con la ***Clave*** de la tabla ***empleado***, permitiendo actualización en cascada.

26. Revise su resultado con:

```
banco=# \d movimientos
```

27. Obtenga una imagen completa de la pantalla, mostrando la estructura de la tabla movimientos y consérvela como evidencia.

28. Ahora debe escribir el siguiente procedimiento, para registrar un depósito a una determinada cuenta:

```
banco=# CREATE OR REPLACE PROCEDURE depósito(Cta int, Emp int,
Imp numeric)
banco-# LANGUAGE plpgsql
banco-# AS $$
banco$# BEGIN
banco$# UPDATE cuentas SET Saldo=Saldo+Imp WHERE Número = Cta;
banco$# INSERT INTO movimientos(Fecha, Cuenta, Empleado,
banco$# Importe, Tipo) VALUES (current_date,Cta,Emp,Imp,'D');
banco$# END;
banco$# $$;
```

29. Obtenga una imagen completa de la pantalla, mostrando la creación del procedimiento y consérvela como evidencia.

30. Para probar el procedimiento anterior, ejecute dos depósitos a cuenta:

```
banco=# CALL depósito(6127, 443, 1500);
banco=# CALL depósito(6993, 512, 2450);
```

31. Para confirmar que se hicieron los cambios, consulte las tablas

correspondientes:

```
banco=# SELECT * FROM movimientos;
banco=# SELECT * FROM cuentas;
```

32. Obtenga una imagen completa de la pantalla, mostrando la ejecución del procedimiento y la consulta de las tablas, consérvela como evidencia.

33. Como ejercicio, el alumno debe copiar el siguiente procedimiento ***retiro***, para registrar los retiros de una cuenta, la diferencia fundamental es que se verifica que el saldo disponible en la cuenta sea suficiente, antes de proceder a la transacción y el tipo de movimiento es ***R***.

```
banco=# CREATE OR REPLACE PROCEDURE retiro (Cta int, Emp int,
Imp numeric)
banco-# LANGUAGE plpgsql
banco-# AS $$
banco$# DECLARE
banco$# s NUMERIC;
banco$# BEGIN
banco$#    SELECT Saldo INTO s FROM cuentas WHERE Número = Cta;
banco$# IF s < Imp THEN
banco$# RAISE EXCEPTION 'Datos proporcionados son erróneos';
banco$# END IF;
banco$# UPDATE cuentas SET Saldo=Saldo-Imp WHERE Número = Cta;
banco$# INSERT INTO movimientos (Fecha, Cuenta, Empleado,
banco$# Importe, Tipo) VALUES (current_date,Cta,Emp,Imp,'R');
banco$# END;
banco$# $$;
```

34. Obtenga una imagen completa de la pantalla, mostrando la creación del procedimiento y consérvela como evidencia.

35. Para probar el procedimiento anterior, ejecute estos retiros a las mismas cuentas:

```
banco=# CALL retiro(6127, 517, 1000);
banco=# CALL retiro(6993, 597, 2000);
banco=# CALL retiro(6127, 517, 1000);
ERROR:  Datos proporcionados son erróneos
```

36. Como podrá verse, este último intento de retiro no procede por falta de fondos:

```
banco=# SELECT * FROM movimientos;
banco=# SELECT * FROM cuentas;
```

37. Obtenga una imagen completa de la pantalla, mostrando la ejecución del procedimiento y la consulta de las tablas, consérvela como evidencia.

38. Cierre la ventana de ***psql***.

39. Fin 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 procedimientos almacenados mediante ***psql***, creando una base de datos, creando varias tablas, agregando diversos registros y alterando los registros mediante un procedimiento que contiene una transacción.

Bibliografía

- https://www.geeksforgeeks.org/postgresql-create-procedure/
- https://www.postgresql.org/docs/current/sql-createprocedure.html