Ejercicio 1: Política de mantenimiento de llaves foráneas 3.

¿Qué es?

Es una forma de manejar situaciones en las que se deben modificar, actualizar o eliminar los datos que guardan las llaves foráneas que están referenciadas a otras llaves foráneas o primarias de otras tablas, debido a que los valores deben corresponder a los que están en otras tablas o bien se deben tomar medidas para que en caso de ser eliminadas, no sigan existiendo en algúna tabla y eso ocasione un mal funcionamiento en la base de datos.

En ese sentido, PostgreSQL es muy accesible para lograr esto a través de la fácil implementacion de llaves foráneas que tiene y sus múltiples maneras de modificarlas a través de *constraints*. Esto previene que exista pérdida de datos e incluso que al usar el *Query Tool* se eficienticen las acciones pedidas a tráves de las llaves foráneas. En general, PostgreSQL es ideal para bases de datos que necesitan alta consistencia y robustez en la integridad referencial.

Tipos de Políticas:

- **NO ACTION:** Es importante notar que cuando no se establecen políticas de mantenimiento explícitamente en el script de la base de datos a través de *constraints* u otros medios, PostgreSQL define por default una política de **NO ACTION.**

    - **Forma de implementarla en SQL:** Utilizando *constraints* en PostgreSQL.

      ```
      CONSTRAINT fk_tabla
          FOREIGN KEY(atributo_de_la_tabla) REFERENCES
          tabla2(atributo_de_la_tabla)
      ```

    - **Objetivo:** Esta busca conservar la integridad referencial dentro de la base de datos a pesar de que no se contemplen políticas aún, es decir que en la construcción de ésta aún no se implemente otro tipo de política.

    - **Funcionamiento**: Consiste en arrojar un error en cuanto la integridad de la base de datos este en riesgo, por ejemplo, al intentar insertar un dato no válido en alguna llave o actualizar incorrectamente el tipo de dato de una llave foránea.

- **Ventajas:** Dado que no es necesario declarala explícitamente (no más que un *constraint*), es una manera muy elemental que tiene PostgresSQL de conservar la integridad referencial de los datos sin que se le pida.

- **Desventajas:** Su principal desventaja es que es muy general, si se levanta el error será solamente cuando el *constraint* de la tabla específica que la contiene sea checado, y no inmediatamente después de realizar alguna acción de actualizacion sobre la tabla. En algunas ocasiones es más recomendable utilizar RESTRICT.

En caso de querer implementar otro tipo de política, existen los siguientes tipos, a continuación se explican:

- **CASCADE:**

  - **Forma de implementarla en SQL:** Aqui vale la pena hacer mención de las dos formas más generales de implementarla. Una a través de **ON DELETE CASCADE** que consistirá en eliminar todas las filas en la tabla dependiente que hacen referencia a una fila. A continuación se muestra una implementación:

```
CREATE TABLE nombre_tabla2(
        atributo1 TIPOVALOR,
        atributo2 TIPOVALOR,
  PRIMARY KEY(atributo1),
  CONSTRAINT fk_atributo2
        FOREIGN KEY(atributo_2)
        REFERENCES nombre_tabla1(atributo2)   - - Asumimos que tenemos otra tabla definida a la que esta referenciada la llave foránea.
        ON DELETE CASCADE
);
```

  El otro caso es **ON UPDATE CASCADE** que consiste en que si la clave primaria de la tabla referenciada cambia, las llaves foráneas en la tabla 'hijo' se actualizarán automáticamente para reflejar este cambio. A continuación se muestra su implementación:

```
CREATE TABLE tabla1 (
   id INT PRIMARY KEY
 );
– -Notemos que se puede declarar sin un constraint y junto con ON
– - DELETE
CREATE TABLE tabla2 (
```

```
    id INT PRIMARY KEY,
      id_a INT REFERENCES tabla1(id) ON UPDATE CASCADE ON
DELETE CASCADE
  );
```

- **Objetivo:** Mantener la integridad referencial de los datos en tablas distintas asegurandose que se elimine o actualice cualquier tipo de referencias que podrían ponerla en riesgo.

- **Funcionamiento:** Consiste en eliminar o actualizar las filas que están relacionadas con la llave foránea en la tabla 'hijo' cuando la tabla 'padre' se elimina o actualiza. Normalmente el ON DELETE se implementa junto con ON UPDATE para manejar los casos donde la llave en la tabla 'padre' se actualice.

- **Ventajas**: Es muy sencillo de implementar y uno de los más comúnes por lo que se puede usar en muchas situaciones.

- **Desventajas:** Dado que elimina datos, no será posible recuperarlos. Además, si existen otras acciones como SET NULL o SET DEFAULT, CASCADE siempre tiene precedencia por lo que debe usarse con cuidado.

- **SET NULL:**

  - **Forma de implementarla en SQL:** Notemos que al igual que CASCADE, SET NULL se implementa en dos casos, con ON DELETE o con ON UPDATE. A continuación se muestra como implementarlo sin constraint y abajo con constraint:

```
CREATE TABLE tabla1 (
        id INT PRIMARY KEY
);

CREATE TABLE tabla2 (
        id INT PRIMARY KEY,
        tabla2_id INT REFERENCES tabla2(id) ON UPDATE SET NULL
        ON DETELE SET NULL
 );

– - con constraint

CREATE TABLE tabla1 (
```

```
        tabla1_id SERIAL PRIMARY KEY,
);

CREATE TABLE tabla2(
        tabla2_id SERIAL PRIMARY KEY,
        tabla1_id INT,
CONSTRAINT fk_tabla2 FOREIGN KEY (tabla1_id)
REFERENCES tabla1(tabla1_id) ON DELETE SET NULL
);
```

- **Objetivo:** Mantener la integridad referencial entre tablas de una manera flexible ya que normalmente se usa cuando la relación entre dos tablas no es estrictamente necesaria, pero resulta útil en caso de existir.

- **Funcionamiento:** Consiste en que cuando se elimina una fila en la tabla o se actualiza la llave a la que hace referencia, las llaves foráneas en la tabla dependiente se configuran a NULL en lugar de eliminarse o actualizarse.

- **Ventajas:** Es ideal cuando necesitas que un elemento de la tabla siga existiendo sin la referencia ‘original’. Además, es útil si queremos ‘rastrear’ las referencias de la tabla ‘hijo’ incluso después de eliminar la tabla ‘padre’.

- **Desventajas:** La principal desventaja es que dado que al definir una tabla se define el tipo de valor que acepta en cada columna, ésta deberá aceptar valores nulos porque sino para el caso de ON UPDATE, ocurirrá un error.

- **SET DEFAULT:**

  - **Forma de implementarla en SQL:**

```
CREATE TABLE tabla1 (
        id INT PRIMARY KEY DEFAULT 'algo'
 );

CREATE TABLE tabla2 (
        id INT PRIMARY KEY,
        tabla1_id INT DEFAULT 9999 REFERENCES tabla1(id) ON
        UPDATE SET DEFAULT ON DELETE SET DEFAULT
 );
```

– - otra manera es modificando la tabla y agregando un *constraint*
ALTER TABLE tabla2
ADD CONSTRAINT nombre_constraint FOREIGN KEY
(tabla1_id) REFERENCES tabla1(id);

- **Objetivo:** Similar al de SET NULL, solo que se utiliza cuando se requiere que las referencias apunten a un valor distinto de NULL.

- **Funcionamiento:** Consiste en que al eliminar o actualizar una fila en la tabla referenciada, los valores de las llaves foráneas en la tabla dependiente se establecerán en su valor predeterminado.

- **Ventajas:** La principal ventaja es que peromite simplificar la entrada de datos y reduce el riesgo de errores causados por NULL o valores faltantes.

- **Desventajas:** Al igual que SET NULL si el valor por default de una columna es NULL entonces ocurrirá un error similar a SET NULL cuando no se soporta ese tipo de valor.

- **RESTRICT:**

  - **Forma de implementarla en SQL:**

    CREATE TABLE tabla2 (
    id SERIAL PRIMARY KEY,
    tabla1_id int REFERENCES tabla1(tabla1_id) ON DELETE
    RESTRICT ON UPDATE RESTRICT
    );

  - **Objetivo:** Es importante notar que se usa principalmente para evitar que se realicen cambios o eliminaciones cuando aún existen dependencias activas. Sin embargo, es equivalente al NO ACTION y es útil para script complejos pero en general se utiliza mejor el NO ACTION.

  - **Funcionamiento:** A grandes rasgos consiste en impedir o bloquear eliminaciones o actualizaciones si aún existen referencias en la tabla dependiente.

- **Ventajas:** Su principal ventaja es que mantiene la integridad referencial evitando pérdida de datos accidental bloqueando operaciones que la podrían poner en riesgo.

- **Desventajas:** Hay ocasiones donde las restricciones limitan eliminaciones o actualizaciones complejas que deseas hacer.

4.

La política de mantenimiento de llaves foráneas que elegiremos para el proyecto final será SET NULL con **ON DELETE SET NULL** y **ON UPDATE SET NULL** ya que consideramos necesaria la flexibilidad en las eliminaciones de números de lineas, hangares o estaciones del metro ya que el valor nulo 'conservará' la referencia sin eliminarla lo que permitirá 'mantener' las conexiónes con alguna otra tabla aunque ya no existan. Esto nos lleva a la preservación de un 'historial' que extrapolando el problema a algún proyecto de expansión futuro o algo similar, resultará útil como parte de la información disponible para la toma de desiciones.

Asimismo, consideramos que esta política reducirá la pérdida de información relevante ya que será más útil al agregar y actualizar números de lineas, hangares o estaciones del metro porque las conexiones entre ellas se mantendrán y solo se actualizaran a NULL que posteriormente podrán modificarse a algún valor deseado. En conclusión, implementar **SET NULL** permitira mantener la integridad referencial y adaptarse a posibles cambios sin comprometer la estructura ni la información en la base de datos del metro.

Referencias:

Cockroach Labs. (n.d.). *Foreign Key Constraint*. CockroachDB Documentation. Recuperado 6 de noviembre del 2024 de, https://www.cockroachlabs.com/docs/stable/foreign-key

GeeksforGeeks. (2024). *PostgreSQL - Foreign Key*. GeeksforGeeks. Recuperado 6 de noviembre del 2024 de, https://www.geeksforgeeks.org/postgresql-foreign-key/

Neon. (2024). *PostgreSQL Foreign Key*. Neon. Recuperado 6 de noviembre del 2024 de, https://neon.tech/postgresql/postgresql-tutorial/postgresql-foreign-key

PostgreSQL Global Development Group. (n.d.). *5.5. Constraints.* PostgreSQL Documentation. Recuperado 6 de noviembre del 2024 de, https://www.postgresql.org/docs/current/ddl-constraints.html#DDL-CONSTRAINTS-FK

Stack Overflow. (2013). *Difference between RESTRICT and NO ACTION* [Discussion forum]. Recuperado 6 de noviembre del 2024 de, https://stackoverflow.com/questions/14921668/difference-between-restrict-and-no-action

Timescale. (n.d.). *Understanding Foreign Keys in PostgreSQL*. Timescale. Recuperado 6 de noviembre del 2024 de, https://www.timescale.com/learn/understanding-foreign-keys-in-postgresql