Bases de Datos I

Trigger, SP, Funciones PlpgSQL

Parte 1

Facultad de Ciencias Exactas
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

INGENIERÍA DE SISTEMAS

Cursada 2019

Repaso

Quiz

Marque la opción correcta

1- ¿Qué es una restricción de integridad declarativa (RID)?

- A. Es una regla sobre una columna, tabla o conjunto de cualquiera de éstas, que el DBMS automáticamente hace que se cumpla.
- B. Es una regla sobre una columna, tabla o conjunto de cualquiera de éstas, que el usuario de la BD por cada operación hace que se cumpla.
- C. Es un procedimiento almacenado, creado por el DBA, que controla que se cumpla una regla sobre los datos.

Repaso
Quiz

Marque la/s opción/es correcta/s

2- Al declarar una nueva Restricción Declarativa (RD) …

A. …. el DBMS no debe realizar ningún control.

- B. …. el DBMS primero debe asegurarse que los datos de la BD la satisfagan
- C. …. el DBMS debe reparar automáticamente los datos de la BD y luego agregar la restricción.
- D. ….el usuario primero debe modificar los datos para que cumplan la RD y luego declararla.

Repaso

Quiz

Marque sólo una opción correcta

1- ¿Cómo se implementa una restricción de integridad de manera procedural?

- A. Con un procedimiento almacenado, que en cada operación el usuario debe encargarse de despertarlo
- B. Con un procedimiento almacenado, creado por el DBA, que controla que se cumpla una regla sobre los datos que se actualizan
- C. Con un procedimiento almacenado que controla que se cumpla una regla sobre los datos cuando éstos se actualizan

Repaso

Quiz

Marque sólo una opción correcta

Al declarar un trigger, que implementa una RI, el DBMS primero verifica que los datos de la BD la satisfagan.

- A. VEDADERO
- B. FALSO

Repaso
Quiz

Marque sólo una opción correcta

1- ¿Cuándo es deseable que una RI Declarativa sea implementada de manera procedural?

- A. Siempre
- B. Nunca
- C. Sólo cuando el DBMS no me provea los mecanismos para hacerlo de manera declarativa

Repaso

Quiz

Marque las frases verdaderas

- A. Las assertion no modifican los datos, solo verifican ciertas condiciones
- B. Los triggers son más potentes porque pueden verificar las condiciones y también modificar los datos
- C. Las assertion están vinculadas a tablas específicas en la base de datos y a eventos específicos
- D. Los triggers no están vinculados a tablas específicas y eventos específicos

Disparadores (TRIGGERS)

Pueden utilizarse para especificar acciones automáticas que debe realizar el DBMS cuando ocurran ciertos eventos y condiciones.

Funcionalidad de las Bases de Datos Activas, que siguen un modelo de **reglas ECA** (**Evento-Condición-Acción**)

1. **Eventos** que activan la regla (generalmente operaciones de actualización de la BD)
2. **Condición** que determina si la acción de la regla debe ejecutarse (Opcional). Si se especifica, entonces primero se evalúa, y si evalúa a verdadera se ejecuta la acción de la regla.
3. **Acción**, suele ser una secuencia de sentencias SQL, pero también podría ser una transacción de base de datos o un programa externo que se ejecutará automáticamente.

Triggers

Según PostgreSQL

CREATE [CONSTRAINT] TRIGGER *nombre*

{ BEFORE | AFTER | INSTEAD OF } **ACTIVACIÓN**

{ *evento* [OR ...] } ***Evento que lo dispara***

INSERT, UPDATE, DELETE, or TRUNCATE (para UPDATE OF columna1 [, columna2 ...]

ON *tabla*

[FOR [EACH] { ROW | STATEMENT }] **GRANULARIDAD**

[WHEN (*condicion*)] ***Condición***

EXECUTE { FUNCTION | PROCEDURE }

función_específica (arg) ***Acción***

Funciones

En PostgreSQL

CREATE FUNCTION nombre ( ) RETURNS trigger AS $$
[DECLARE] [declaraciones de variables]
BEGIN
Código PlpgSQL
END;
$$ LANGUAGE **'plpgsql'**;

Dentro del cuerpo de la función tengo a disposición diferentes variables:

NEW Tipo de dato RECORD; variable que almacena la nueva fila para las operaciones INSERT/UPDATE en Triggers a nivel ROW, ***en los Triggers a nivel STATEMENT es NULL***

OLD Tipo de datos RECORD; variable que almacena la antigua fila para operaciones UPDATE/DELETE en Triggers de nivel ROW, ***en Triggers de nivel STATEMENT es NULL***

Funciones

En PostgreSQL

Algunas otras (hay muchas más) variables especiales son:

TG_NAME Tipo de dato text; variable que contiene el nombre del trigger actualmente disparado.

TG_WHEN Tipo de dato text; una cadena conteniendo el string BEFORE o AFTER dependiendo de la definición del trigger.

TG_LEVEL Tipo de dato text; una cadena conteniendo el string ROW o STATEMENT dependiendo de la definición del trigger.

TG_OP Tipo de dato text; una cadena conteniendo el string INSERT, UPDATE o DELETE indicando por cuál operación se disparó el trigger.

TG_TABLE_NAME Tipo de dato text; variable que contiene el nombre de la tabla que disparó el trigger

Triggers - utilidad

Forzar las reglas de integridad que no pudieron ser especificadas de forma declarativa dentro del DBMS
Cómo comienzo a transformar las RI declarativas en procedurales?

Paso 1 - Determinando los eventos críticos, usar la matriz de ayuda

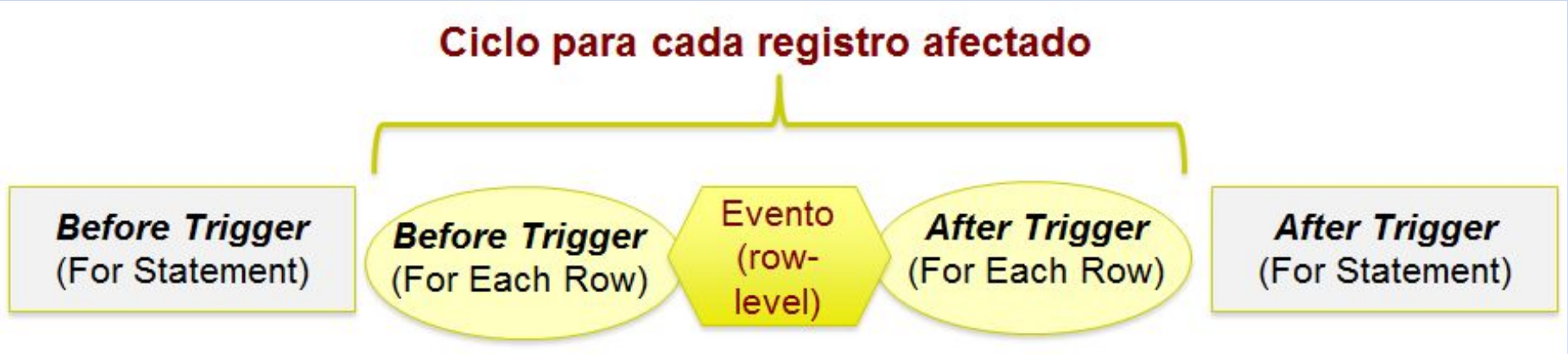
Tabla / Evento	INSERT	UPDATE	DELETE
nombre_tabla		especificar atributos	

Qué tipo de granularidad debe tener un trigger que controla una RI?

Qué tipo de tiempo de activación debe tener un trigger que controla una RI?

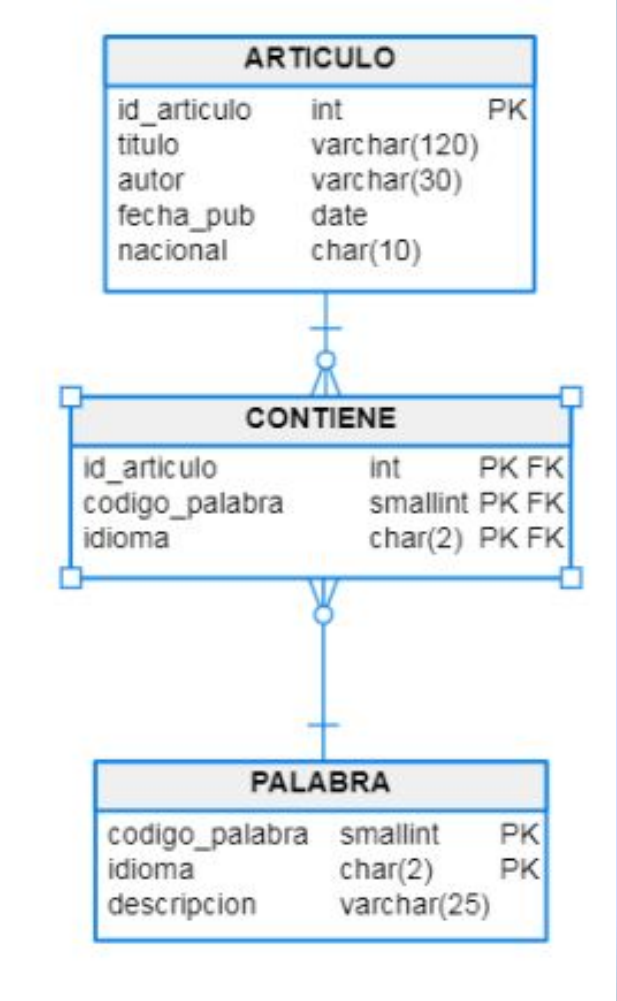
Miremos un poco el el orden de disparo de los trigger…..

Ejecución de los Trigger



RI Declarativa

Ejercicio 3 - TP 3 - Los artículos pueden tener como máximo 15 palabras claves



Implementación de RI con Trigger

Ejercicio 3 - TP 3 - Los artículos pueden tener como máximo 15 palabras claves

(RI Declarativa)

```
ALTER TABLE CONTIENE
ADD CONSTRAINT
CK_MAXIMO_PL_CLAVES
CHECK NOT EXISTS (
        SELECT 1
        FROM CONTIENE
        GROUP BY id_articulo
        HAVING COUNT(*) > 15);
```

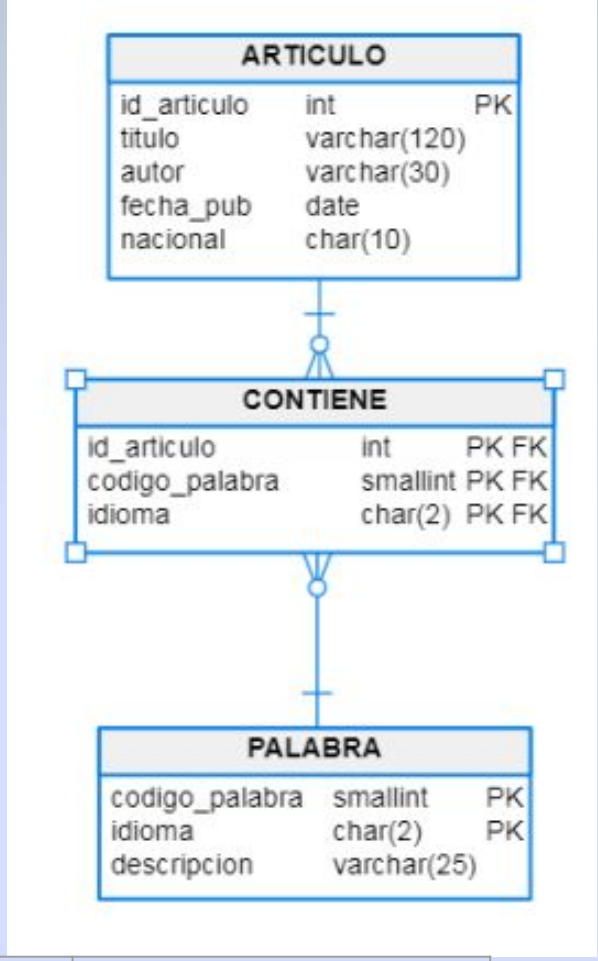


Tabla / Evento	INSERT	UPDATE	DELETE

Implementación de RI con Trigger

Ejercicio 3 - TP 3 - Los artículos pueden tener como máximo 15 palabras claves

(RI Declarativa)

```
ALTER TABLE CONTIENE
ADD CONSTRAINT
CK_MAXIMO_PL_CLAVES
CHECK NOT EXISTS (
        SELECT 1
        FROM CONTIENE
        GROUP BY id_articulo
        HAVING COUNT(*) > 15);
```

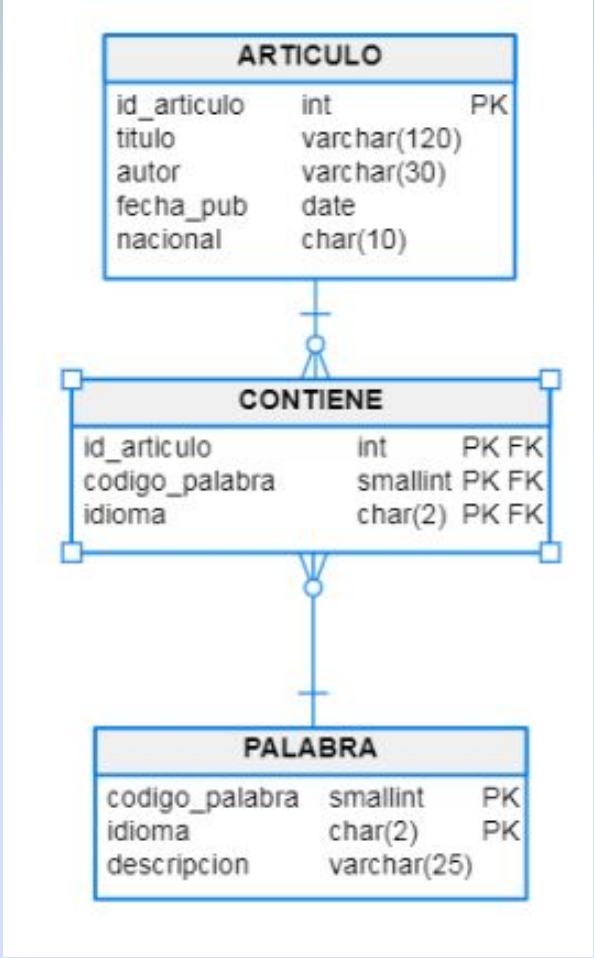


Tabla / Evento	INSERT	UPDATE	DELETE
CONTIENE	SI	SI - id_articulo	NO

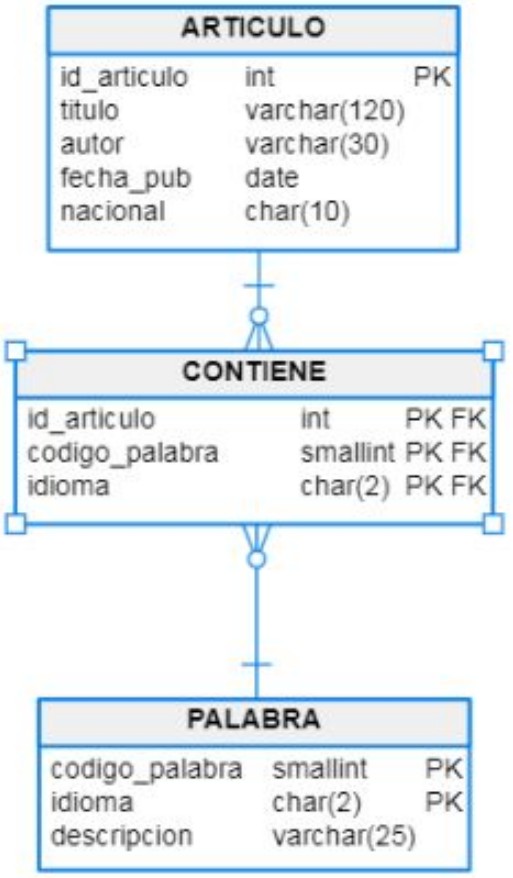
Implementación de RI con Trigger

```
CREATE OR REPLACE FUNCTION FN_MAXIMO_PL_CLAVES() RETURNS trigger AS
$$
DECLARE
   cant       integer;
BEGIN
  SELECT count(*) INTO cant
  FROM CONTIENE
  WHERE id_articulo = NEW.id_articulo;
  IF (cant > 2 ) THEN
    RAISE EXCEPTION 'Superó la cantidad de palabras claves: %', cant
  END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';

CREATE TRIGGER TR_MAXIMO_PL_CLAVES
BEFORE INSERT OR UPDATE OF id_articulo
ON CONTIENE
   FOR EACH ROW EXECUTE PROCEDURE FN_MAXIMO_PL_CLAVES();
```

RI Declarativa

Ejercicio 4 - TP 3 - Sólo se pueden publicar artículos argentinos que contengan hasta 10 palabras claves.



Implementación de RI con Trigger

Ejercicio 4 - TP 3 - Sólo se pueden publicar artículos argentinos que contengan hasta 10 palabras claves.

(RI Declarativa)

```
CREATE ASSERTION
ASS_ART_ARG_MAXIMO_PL_CLAVES
CHECK NOT EXISTS (
        SELECT 1
        FROM ARTICULO A JOIN CONTIENE C
        ON (a.id_articulo = c.id_articulo)
        WHERE nacionalidad = 'Argentino'
        GROUP BY a.id_articulo
        HAVING COUNT(*) > 10);
```

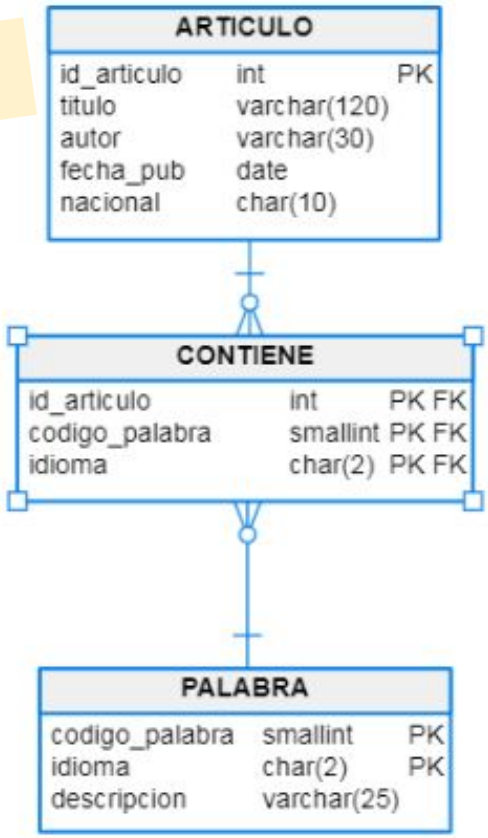


Tabla / Evento	INSERT	UPDATE	DELETE

Implementación de RI con Trigger

Ejercicio 4 - TP 3 - Sólo se pueden publicar artículos argentinos que contengan hasta 10 palabras claves.

(RI Declarativa)

```
CREATE ASSERTION
ASS_ART_ARG_MAXIMO_PL_CLAVES
CHECK NOT EXISTS (
        SELECT 1
        FROM ARTICULO A JOIN CONTIENE C
        ON (a.id_articulo = c.id_articulo)
        WHERE nacionalidad = 'Argentino'
        GROUP BY a.id_articulo
        HAVING COUNT(*) > 15);
```

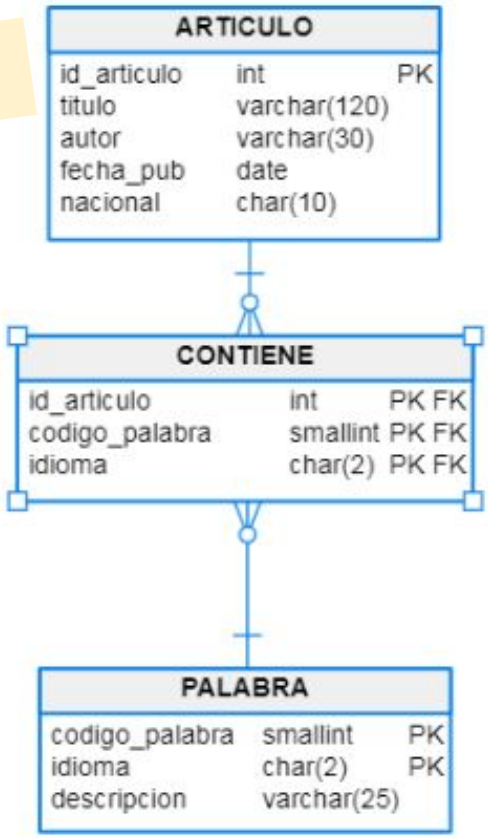


Tabla / Evento	INSERT	UPDATE	DELETE
ARTICULO	NO	SI nacionalidad	NO
CONTIENE	SI	SI id_articulo	NO

Implementación de RI con Trigger

```
CREATE OR REPLACE FUNCTION FN_ART_ARG_MAXIMO_PL_CLAVES() RETURNS
trigger AS $$
DECLARE
   cant       integer;
      v_nacional articulo.nacional%type;
BEGIN
--cuando el trigger se despierta por un update en articulo
  IF (TG_TABLE_NAME = 'articulo') THEN
     SELECT count(*) INTO cant
     FROM CONTIENE
     WHERE articulo_id_articulo = NEW.id_articulo;
     IF (cant > 2 ) THEN
       RAISE EXCEPTION 'Superó la cantidad % palabras claves', cant;
     END IF;
  ELSE
```

Implementación de RI con Trigger

```
--cuando el trigger se despierta por un insert or update en contiene
            SELECT nacional into v_nacional
            FROM ARTICULO A
            WHERE id_articulo = NEW.id_articulo;
            IF (v_nacional = 'Argentina') THEN
                  SELECT count(*) INTO cant
                  FROM CONTIENE
                  WHERE articulo_id_articulo = NEW.articulo_id_articulo;
                  IF (cant > 2 ) THEN
                        RAISE EXCEPTION 'Superó la cantidad % palabras claves', cant;
                  END IF;
            END IF;
   END IF;
RETURN NEW;
END $$
LANGUAGE 'plpgsql';
```

Implementación de RI con Trigger

```
CREATE TRIGGER TR_ART_ARG_MAXIMO_PL_CLAVES_ART
     BEFORE UPDATE OF nacionalidad
     ON ARTICULO
    FOR EACH ROW
     WHEN (NEW.nacional = 'Argentina')
     EXECUTE PROCEDURE FN_ART_ARG_MAXIMO_PL_CLAVES();

CREATE TRIGGER TR_ART_ARG_MAXIMO_PL_CLAVES_CONT
     BEFORE INSERT OR UPDATE OF id_articulo
     ON CONTIENE
    FOR EACH ROW
     EXECUTE PROCEDURE FN_ART_ARG_MAXIMO_PL_CLAVES();
```