BD

Fecha	@4 de abril de 2024 14:06
Asignatura	Base de Datos
Materiales	https://drive.google.com/file/d/1gFLZKyx0NcginAFLsqaWHmWcoEPycz6j/view https://www.mockaroo.co
Revisado	☐

Binarias 1:N :

Los atributos identificadores de la entidad (clave de la relación) del 'lado 1', se agregan como atributos en la tabla correspondiente a la entidad del 'lado N'
→constituyen una clave extranjera

Binarias 1:1 :

El identificador de la derecha se pasa a la izquierda como FK

Unaria 1:N :

Se une con la misma tabla y
debe renobrarse el PK

Binarias N:N :

Se crean tablas para las entidades y una para la relacion, la misma, esta compuesta por los atributos PK de cada entidad
Alumno→Cursa←— Clases tabla para alumno para cursa y para clases

Unarias N:N :

En ves de unirse con si misma se une dos veces con otra tabla, la primera ves para unir el PK y la segunda para crear uno en la tabla. Alumn←—→ EsTutor

La tabla seria
Alumno id PK

Es tutor id PK FK
jefe PK FK

```
CREACION DE UNA TABLA

CREATE TABLE ALUMNO(
LU integer NOT NULL,
Apellido varchar(30) NOT NULL,
Nombre varchar(30) NOT NULL,
FechaNac date,
CONSTRAINT PK_ALUMNO PRIMARY KEY (LU));


CREATE TABLE CARRERA(
IdCarrera integer NOT NULL,
NombreCarrera varchar(40) NOT NULL,
```

```
PlanEstudio varchar(15) NOT NULL,
CONSTRAINT PK_CARRERA PRIMARY KEY (IdCarrera));

CREATE TABLE CURSA(
LU integer NOT NULL,
IdCarrera integer NOT NULL,
CONSTRAINT PK_ CURSA PRIMARY KEY (LU, IdCarrera));


ALTER TABLE CURSA ADD CONSTRAINT FK_CURSA_CARRERA
FOREIGN KEY (IdCarrera)
REFERENCES CARRERA(IdCarrera);


ALTER TABLE CURSA ADD CONSTRAINT FK_CURSA_ALUMNO
FOREIGN KEY (LU)
REFERENCES ALUMNO(LU);
```

Por ejemplo algunas sentencias SQL para la definición de datos (DDL-Data Definition Languaje):

CREATE TABLE <nom_tabla> (….); → creación de una tabla

ALTER TABLE <nom_tabla> … ; → modificación de una tabla

DROP TABLE <nom_tabla> ; → eliminación de una tabla

INSERT INTO <nom_tabla> (…) VALUES (….); → inserción de registros

UPDATE <nom_tabla> SET (…) WHERE (….); → modificación de registros

DELETE FROM <nom_tabla> WHERE (….); → borrado de registros

SELECT <lista_atrib> FROM <lista_tablas> WHERE (….); → consulta de datos

---

```
CREATE TABLE [ IF NOT EXISTS ] nombre_tabla (

{ nombre_columna tipo_dato [NOT NULL | NULL | DEFAULT default_expr ]

[ restricción_de_columna [ CONSTRAINT constraint_name ]

{ UNIQUE index_parameters |

PRIMARY KEY index_parameters | ]}

{ restricción_de_tabla [ CONSTRAINT constraint_name ]

{ UNIQUE ( column_name [, ... ] ) |
```

```
PRIMARY KEY ( column_name [, ... ] ) |

FOREIGN KEY ( column_name [, ... ] ) ]

});}
CREATE TABLE CARRERA(
IdCarrera integer NOT NULL,
NombreCarrera varchar(40) NOT NULL,
PlanEstudio varchar(15) NOT NULL,
CONSTRAINT PK_CARRERA PRIMARY KEY (IdCarrera));

_____________________________________________________________________

Una vez creada una tabla es posible realizar algunas
modificaciones en su definición (ej, agregar o quitar
columnas, especificar valores por defecto, incorporar
restricciones o quitarlas, etc. La sintaxis de PostgreSQL

ALTER TABLE tabla ADD COLUMN columna tipo; Agrega una columna

ALTER TABLE tabla DROP COLUMN columna; Borra una columna

ALTER TABLE tabla RENAME COLUMN Renombra una columna

Columna_vieja TO columna_nueva;

ALTER TABLE tabla columna TYPE nuevo_tipo; Cambia el tipo de tato

ALTER TABLE tabla ALTER COLUMN columna Asigna o elimina el

[SET DEFAULT value | DROP DEFAULT] valor por defecto

ALTER TABLE tabla ALTER COLUMN columna Asigna o elimina la

[SET NOT NULL | DROP NOT NULL] restricción de nulidad

ALTER TABLE tabla Agrega una restricción

ADD CONSTRAINT nombre definición_de_constraint
```

GENERA DATOS DE MANERA ALEATORIA!

https://www.mockaroo.com/

Entidad Debil

Una entidad débil puede ser unívocamente identificada sólo en el contexto de otra entidad fuerte o propietaria

Entidades fuerte y débil están vinculadas por una relación binaria (1,1):(*,N). Siempre la cardinalidad del lado 1 es 1

```
SELECT * | { [DISTINCT] columna | expresión [alias],...}

FROM <lista tablas>
```

Por defecto, ante una consulta, se recuperan
todas las filas, incluidas las filas duplicadas.

Para eliminar los valores repetidos se debe
usar la cláusula DISTINCT

SELECT apellido || ‘,’ || nombre AS “Apellido,nombre”
FROM voluntario;
El alias AS renombra un encabezamiento de columna o
tabla.
Operador de concatenación || para cadenas de caracteres

La cláusula WHERE se usa para realizar las restricciones.
Una cláusula WHERE contiene una condición lógica, la cual
usa:
operadores de comparación (<, >, =, <=, >=, <> o !=)
operadores lógicos (AND, OR, NOT).

Las filas recuperadas son aquellas cuyos datos que
satisfacen la/s condición/es lógicas

```
SELECT *
FROM voluntario
WHERE id_tarea != 'ST_MAN';
```

BETWEEN trata a los valores de los extremos incluidos
dentro del rango.
BETWEEN x AND y es equivalente a a >= x AND a <= y
NOT BETWEEN x AND y es equivalente a a < x OR a > y

```
SELECT * FROM voluntario
WHERE nro_voluntario BETWEEN 100 AND 120;
```

No siempre se conoce el valor exacto a buscar.

Se puede buscar coincidencias con un patrón de caracteres
mediante el operador LIKE. También se emplea en la forma
negativa NOT LIKE.

```
SELECT * FROM voluntario WHERE nombre LIKE ‘_a%n’;
```

%: cualquier secuencia de cero o más caracteres
_ : denota un solo carácter

Si se comparan valores nulos usando los otros
operadores (=, >, etc.)el resultado será siempre
DESCONOCIDO porque un valor nulo no puede ser
igual, mayor, menor o distinto a otro valor.

```
Si se desea incluir en el resultado los datos de aquellas
columnas que tengan nulos hay que hacerlo
explícitamente.

SELECT * FROM voluntario
WHERE porcentaje <= 0.1
OR porcentaje IS NULL;

_____________________________________________________________________

El operador AND retorna
VERDADERO si ambas
condiciones evaluadas
son VERDADERAS

El operador OR retorna
VERDADERO si alguna
de las condiciones es
VERDADERA.

El operador NOT invierte
el resultado de la
expresión.

SELECT nro_voluntario,apellido,id_institucion,id_coordinador
FROM voluntario
WHERE (id_coordinador=100
OR id_coordinador=124)
AND id_institucion=50;

Ejemplo: Seleccionar los voluntarios que son coordinados por los
voluntarios nro 100 y 124 que están trabajando para la institución
cuyo código es 50.
_____________________________________________________________________
SELECT apellido, nombre
FROM voluntario
WHERE id_coordinador=124
ORDER BY apellido DESC, nombre;

_____________________________________________________________________
SELECT lista de atributos
FROM tabla/s
[ORDER BY ... ]
[LIMIT { número | ALL}]
[OFFSET número];

Debería ser usado siempre con la cláusula ORDER BY.

La cláusula LIMIT limita la cantidad de filas a retornar.

La cláusula OFFSET determina a partir de qué fila del
resultado se retorna.
_____________________________________________________________________

SUM( ) → sumatoria de la columna especificada

AVG( ) → promedio de la columna especificada
```

```
STDDEV() → desvío estándar de la columna especificada

MAX( ) → valor máximo de la columna especificada

MIN ( ) → valor mínimo de la columna especificada

COUNT ( ) → cantidad de tuplas

SELECT SUM(horas_aportadas),
AVG(horas_aportadas),
MAX(horas_aportadas),
MIN(horas_aportadas)
FROM voluntario;

SELECT MAX(fecha_nacimiento) AS voluntario_mas_joven,
MIN(fecha_nacimiento) AS voluntario_mas_viejo
FROM voluntario;

SELECT COUNT(*)
SELECT COUNT(ciudad) AS cantidad__de_ciudades
FROM voluntario;
_____________________________________________________________________________

Si se usa la cláusula GROUP BY en una sentencia
SELECT, se dividen las filas de la tabla consultada en
grupos

Se aplican las funciones en la lista SELECT a cada
grupo de filas y retorna una única fila por cada grupo.

SELECT lista columnas, función de grupo (columna)
FROM <lista tablas>
[ WHERE condición ]
[GROUP BY expresión de grupo | lista columnas
[ ORDER BY <lista atributos orden> ];

No se puede utilizar la cláusula WHERE para restringir
grupos

Se debe utilizar la cláusula HAVING para restringir grupos.
No se pueden utilizar funciones de grupo en la cláusula
WHERE

SELECT id_coordinador, COUNT(*) AS
cantidad_de_voluntarios
FROM voluntario
GROUP BY id_coordinador
HAVING COUNT(*) > 7;
```

Para Recordar!!!!

La sentencia SQL empleada para la recuperación de los datos a partir de las tablas cargadas en la base de datos es el SELECT

SELECT identifica las columnas a recuperar – EL QUE

FROM identifica la/s tabla/s - DE DONDE obtener los datos

WHERE se usa para realizar las restricciones sobre los datos (filtrar)

Para eliminar los valores repetidos se debe usar la cláusula DISTINCT

Los operadores de comparación se utilizan en la cláusula WHERE para comparar expresiones. Usar paréntesis y sangrías para mejorar la legibilidad.

ORDER BY puede usarse para ordenar las filas, y se debe colocar como última cláusula de la sentencia SELECT

Las funciones de agregación operan sobre conjuntos de filas para proporcionar un resultado por grupo.

GROUP BY especifica como se deben agrupar las filas seleccionadas. Todas las columnas de la lista SELECT, excepto las funciones de grupo, deben estar en la clausula GROUP BY. No se pueden utilizar funciones de grupo en la cláusula WHERE.

```
El operador JOIN combina dos tablas según una condición
para obtener registros compuestos por atributos de las dos
tablas combinadas.

SELECT v.apellido, v.nombre
FROM voluntario v INNER JOIN tarea t ON
v.id_tarea = t.id_tarea
INNER JOIN institucion i ON
v.id_institucion = i.id_institucion;

utilizamos el JOIN
múltiple para recuperar sólo aquellos voluntarios
a los que se les asignó un tarea y una
institución. De los 11 registros de la tabla
voluntario, sólo 9 se retornaron; ni Nancy
Greenberg ni John Russell están en el resultado
de la consulta.



SELECT nombre_tarea, apellido, nombre
FROM tarea t LEFT JOIN voluntario v
ON (t.id_tarea = v.id_tarea)
JOIN institucion i ON (i.id_institucion = v.id_institucion);
_____________________________________________________________________________


Subconsultas de una Sola Fila

SELECT nombre, id_tarea, horas_aportadas
FROM voluntario
WHERE id_tarea = (SELECT id_tarea
FROM voluntario
```

```
WHERE id_voluntario = 141)
AND horas_aportadas > (SELECT horas_aportadas
FROM voluntario
WHERE id_voluntario = 143);

Se desea seleccionar los voluntarios que
realizan la misma tarea que el voluntario 141 y que
aportan más horas que el voluntario 143
```

```
Cláusula HAVING en subconsultas

Se ejecuta en primer lugar la subconsulta. Devuelve resultados
a la cláusula HAVING (correspondiente a la consulta principal)
que luego se usan para chequear la condición de grupo.

SELECT id_institucion, MIN(horas_aportadas)
FROM voluntario
GROUP BY id_institucion
HAVING MIN(horas_aportadas) > (SELECT MIN(horas_aportadas)
FROM voluntario WHERE id_institucion = 40);
```

```
El operador IN permite especificar múltiples valores en una
cláusula WHERE. Podría decirse que es una forma
abreviada de varias condiciones OR. La forma es:

SELECT nro_voluntario, apellido, nombre
FROM voluntario
WHERE id_tarea IN (‘ST_CLERK’, ‘SA_MAN’,
‘SA_REP’, ‘IT_PROG’);

SELECT e.nombre, e.apellido
FROM empleado e
WHERE (e.id_departamento, e.id_distribuidor) IN (

SELECT d.id_departamento,d.id_distribuidor
FROM departamento d

WHERE d.id_ciudad IN (
SELECT c.id_ciudad
FROM ciudad c

WHERE c.id_pais IN (
SELECT p.id_pais
FROM pais p
WHERE nombre_pais = 'ARGENTINA')));

DE ABJO HACIA ARRIBA MIRALO!!!
```

```
El operador EXISTS comprueba la existencia de filas en
el conjunto de filas del resultado de la consulta.

Si se encuentra un valor de fila de la subconsulta:

La búsqueda no continúa en la consulta interna.
```

```
Se señaliza a la condición como TRUE.

Si no se encuentra un valor de fila de la subconsulta:

Se señaliza a la condición como FALSE.

La búsqueda continúa en la consulta interna.

SELECT e.nombre, e.apellido
FROM empleado e
WHERE EXISTS (

SELECT 'X'
FROM departamento d
WHERE e.id_departamento=d.id_departamento
AND e.id_distribuidor=d.id_distribuidor
AND EXISTS (

SELECT 'Y'
FROM ciudad c
WHERE d.id_ciudad = c.id_ciudad

AND EXISTS (
SELECT 'Z'
FROM pais p
WHERE p.id_pais=c.id_pais
AND nombre_pais = 'ARGENTINA')));


Como la subconsulta 4 encontró una tupla que
cumple la restricción, devuelve ‘Z’ a la
subconsulta 3. Entonces, la subconsulta 3
encontró un “Z” y devuelve ‘Y’ a la subconsulta
2. La subconsulta 2 encontró un “Y”, entonces
devuelve ‘X’ a la subconsulta 1.

Finalmente la consulta 1 agrega la tupla a la
tabla de resultados y continua el proceso con
la siguiente tupla de la tabla Empleado.
```

Restricciones paaaa
Esto es facil nene:

condiciones que restringen los valores en la BD

descripción de estados correctos en tiempo de diseño

previenen inconsistencias

RI: Restricciones

DBA: Aplicar restricciones sobre los datos
SGBD: Evitar la maniulacion de datos que no cumplan las RI

```
CREATE TABLE AREA (
tipo_area char(2) NOT NULL,
id_area int NOT NULL,
descripcion varchar(50) NOT NULL,
CONSTRAINT PK_AREA PRIMARY KEY (tipo_area,id_area));


CREATE TABLE EMPLEADO (
id_empleado int NOT NULL,
nombre varchar(50) NOT NULL,
apellido varchar(50) NOT NULL,
fecha_nac date NOT NULL,
tipo_area char(2) NULL,
id_area int NULL,
CONSTRAINT PK_EMPLEADO PRIMARY KEY (id_empleado));




ALTER TABLE EMPLEADO ADD CONSTRAINT FK_EMPLEADO_AREA
FOREIGN KEY (tipo_area, id_area)
REFERENCES AREA (tipo_area, id_area) ;


CREATE TABLE NombreTabla (
{ nom_col TipoDato [NOT NULL] [DEFAULT valorDefecto], … }
[ [CONSTRAINT PK_nom] PRIMARY KEY (lista_col_PK),]
{ [ [CONSTRAINT UK_nom] UNIQUE (lista_col),] }
... );

o: ALTER TABLE NombreTabla
ADD [CONSTRAINT PK_nom] PRIMARY KEY (lista_col_PK);
ALTER TABLE NombreTabla
ADD [CONSTRAINT UQ_nom] UNIQUE (lista_col_UQ);

_____________________________________________________________________________
¿Qué sucede si se intenta borrar (delete) un registro en la tabla AREA que está
siendo referenciado en la tabla EMPLEADO por la FK?

¿Qué sucede si se intenta modificar (update) la clave primaria de un registro en
la tabla AREA que está siendo referenciada en tabla EMPLEADO por la FK?


Opciones 1 - Rechazo de la operación

NO ACTION: no permite borrar un registro cuya clave
primaria está siendo referenciada por un registro en la Tabla
EMPLEADO (es la opción por defecto)

RESTRICT : misma semántica que NO ACTION, pero se
chequea antes de las otras RI
_____________________________________________________________________________

Opciones 2 - Acepta la operación y realiza acciones reparadoras
adicionales: borra el registro en la tabla AREA y si es

CASCADE: se propaga el borrado a todos los registros que
```

```
referencian a dicha clave primaria mediante la FK en la tabla
EMPLEADO

SET NULL: les coloca nulos en la FK de los registros que
referencian a dicha clave primaria en la tabla EMPLEADO
(sólo si admite nulos)

SET DEFAULT: les coloca el valor por defecto en la FK de
los registros que referencian a dicha clave primaria en la
tabla EMPLEADO (si es posible)


CREATE TABLE NombreTabla ( …...

{ [ [CONSTRAINT FK_nom] FOREIGN KEY (lista_columnasFK)
REFERENCES nombreTablaRef [(lista_columnasRef)]

[ MATCH {FULL | PARTIAL | SIMPLE}]
[ON UPDATE AccionRef]
[ON DELETE AccionRef] ] } ….. );


o: ALTER TABLE NombreTabla
ADD CONSTRAINT FK_nom FOREIGN KEY (lista_columnasFK) …;

AccionRef= NO ACTION | CASCADE | SET NULL | SET DEFAULT | RESTRICT


_____________________________________________________________________________
```

Tipos de Matching

Los tipos de matching afectan cuando las FK se definen sobre varios atributos, y pueden contener valores nulos

Indican los requisitos que deben cumplir los conjuntos de valores de atributos de la FK en R, respecto de los correspondientes en la clave referenciada en R´

MATCH SIMPLE (Opción por defecto para SQL estandar y PostgreSQL)

MATCH PARTIAL

MATCH FULL

La integridad referencial se satisface si para cada registro en la tabla referenciante se verifica lo siguiente:

1- Ninguno de los valores de las columnas de la FK es NULL y existe un registro en la tabla referenciada cuyos valores de clave coinciden con los de tales columnas,

Al menos un valor en una de las columnas de la FK es NULL y puede o NO el resto de los valores de las columnas hacer referencia a la PK (

MATCH SIMPLE)

Los valores de las columnas no nulos de la FK se corresponden con los correspondientes valores de clave en al menos en un registro de la tabla referenciada (
MATCH PARTIALL)

Todas las columnas de la FK son NULL (**MATCH FULL**) o hacen referencia a la PK completa

La especificación declarativa de RI sigue la estructura jerárquica del modelo relacional (atributo→tupla→tabla→BD):

RI Dominio o de atributo (DOMAIN o CHECK de atributo)

RI de tabla asociada a uno ó más atributos (CHECK de registro)

RI de tabla asociada a varias tuplas (CHECK de tabla)

RI generales de la base de datos (ASSERTION)

RI de Dominio/Atributo

Permiten definir el conjunto de los valores válidos de un atributo

Casos particulares: NOT NULL, DEFAULT, PRIMARY KEY, UNIQUE

Ámbito de la restricción: atributo

Se pueden especificar las RI del atributo en la sentencia CREATE TABLE o definirlas en un dominio y declarar el atributo perteneciente al dominio

CREATE DOMAIN NomDominio

AS TipoDato [DEFAULT ValorDefecto]

[[CONSTRAINT NomRestriccion] CHECK (condición);

```
Las especialidades de los ingenieros pueden ser "INTELIGENCIA
EMPRESARIAL" , "TECNOLOGÍAS MOVILES" , "GESTIÓN IT" o
"DESARROLLO"

ALTER TABLE ej_ingeniero
ADD CONSTRAINT ck_ej_ingeniero_especialidad
CHECK (especialidad IN (
'INTELIGENCIA EMPRESARIAL' ,
'TECNOLOGÍAS MOVILES' ,
'GESTIÓN IT', 'DESARROLLO'));

Otro ejemplo: la remuneración de un ingeniero debe ser mayor o igual a
25000$ y menor o igual a 250.000$

ALTER TABLE ej_ingeniero
ADD CONSTRAINT ck_ej_ingeniero_remuneracion
```

```
CHECK (remuneracion BETWEEN 25000 AND 250000);


_____________________________________________________________________________
RI (CHECK) de Tupla

Representa una restricción específica sobre los valores que
puede tomar una combinación de atributos en una tupla

Ámbito de la restricción: tupla (la RI se comprueba para cada
fila que se inserta o actualiza en la tabla)

CREATE TABLE NombreTabla
( …..
{ [[CONSTRAINT nom_restr] CHECK (condición) ] } );

o: ALTER TABLE NombreTabla
ADD [CONSTRAINT nom_restr] CHECK (condición) ;

La condición debe evaluar como
VERDADERA o DESCONOCIDA

Los tipos de areas BC sólo pueden tener id_areas que van del 3 al 7 para el resto no
habría controles

ALTER TABLE AREA
ADD CONSTRAINT ck_control_area
CHECK ( ( (tipo_area = 'BC') AND (id_area BETWEEN (3 AND 7) ) OR
tipo_area <> 'BC');

ALTER TABLE ej_proyecto
ADD CONSTRAINT ck_ej_proyecto_presupuesto
CHECK (fecha_fin IS NULL AND presupuesto <= 1000000);
OR (fecha_fin IS NOT NULL) );

ALTER TABLE EJ_TRABAJA
ADD CONSTRAINT ck_cant_ingenieros
CHECK ( NOT EXIST ( SELECT 1
FROM EJ_TRABAJA
GROUP BY id_sector, nro_proyecto
HAVING COUNT(*) > 10));
```

RI Globales (ASSERTIONS)

Permiten definir restricciones sobre un número arbitrario de atributos de un número arbitrario de tablas

Ámbito de la restricción: base de datos

No están asociadas a un elemento (tabla o dominio) en particular

CREATE ASSERTION NomAssertion CHECK (condición);

Su activación se daría ante actualizaciones sobre las tablas involucradas

Requerirían alto costo para comprobación y mantenimiento

→ los DBMS comerciales no soportan ASSERTIONS !

```
El sueldo de los empleados de un área no puede ser mayor al sueldo del gerente
de esa área

EMPLEADO (idE,.., sueldo, AreaT) AREA (IdArea, …., gerente)
CREATE ASSERTION salario_valido
CHECK ( NOT EXISTS ( SELECT 1 FROM Empleado E, Empleado G, Area A
WHERE E.sueldo > G.sueldo
AND E.AreaT = A.IdArea
AND G.IdE = A.gerente ) );
```

Restricciones de Integridad Referencial (RI):

1. **RI de Dominio o Atributo**:
   - **Explicación**: Estas restricciones definen el conjunto de valores válidos para un atributo específico en una tabla. Se aplican al definir el dominio o durante la creación de la tabla.
   - **Ejemplo**: Limitar las especialidades de los ingenieros a un conjunto específico.

     ```
     sqlCopy code
     CREATE DOMAIN especialidad AS varchar(20)
     CHECK( VALUE IN ('inteligencia empresarial', 'tecnologías móviles', 'gestión de TI',
     'desarrollo'));
     ```

2. **RI de Tupla**:
   - **Explicación**: Se aplican a combinaciones específicas de atributos en una tupla (fila). Se definen durante la creación de la tabla o mediante la modificación de la misma.
   - **Ejemplo**: Limitar el presupuesto de proyectos sin fecha de finalización asignada a $100,000.

     ```
     sqlCopy code
     ALTER TABLE EJ_PROYECTO
     ADD CONSTRAINT ck_proyectosenfecha
     CHECK ((fecha_fin IS NULL AND presupuesto < 100000) OR (fecha_fin IS NOT NULL));
     ```

3. **RI de Tabla**:
   - **Explicación**: Se aplican a toda la tabla y se comprueban para cada fila individualmente. Se definen durante la creación de la tabla o mediante la modificación de la misma.
   - **Ejemplo**: Limitar el número máximo de ingenieros que pueden trabajar en un proyecto a 10.

     ```
     sqlCopy code
     ALTER TABLE EJ_TRABAJA
     ADD CONSTRAINT ck_cant_ingenieros
     CHECK (NOT EXISTS (SELECT 1 FROM EJ_TRABAJA GROUP BY id_sector, nro_proyecto HAVING
     COUNT(*) > 10));
     ```

4. **RI Globales (ASSERTIONS)**:
   - **Explicación**: Se aplican a múltiples tablas y pueden involucrar lógica más compleja que no se puede expresar fácilmente en una sola tabla. Se definen por separado como ASSERTIONS.
   - **Ejemplo**: Asegurar que cada director asignado a un proyecto haya trabajado en al menos 5 proyectos.

     ```
     sqlCopy code
     CREATE ASSERTION CK_PROY_DIRE
     CHECK (NOT EXISTS (
     ```

```
    SELECT P.director, COUNT(*) AS "cantidad proyectos"
    FROM EJ_PROYECTO P
    JOIN EJ_TRABAJA T ON (P.director = T.id_ingeniero)
    JOIN EJ_PROYECTO PP ON (PP.id_sector = T.id_sector AND PP.nro_proyecto = T.nro_p
royecto)
    WHERE PP.fecha_fin IS NOT NULL
    GROUP BY P.director
    HAVING COUNT(*) < 5
));
```

Restricciones Adicionales:

1. **Triggers**:
   - **Explicación**: Son piezas de código almacenadas en la base de datos que se ejecutan automáticamente en respuesta a ciertos eventos. Se definen por separado en la base de datos.
   - **Ejemplo**: Un trigger que se dispara cuando se inserta un nuevo empleado y actualiza automáticamente la tabla de salarios.

     ```
     sqlCopy code
     CREATE TRIGGER actualiza_salario
     AFTER INSERT ON Empleado
     FOR EACH ROW
     BEGIN
         UPDATE Salarios SET salario = salario + NEW.aumento WHERE id_empleado = NEW.id_e
     mpleado;
     END;
     ```

2. **Tipos de Matching**:
   - **Explicación**: Afectan cómo se verifican las restricciones de integridad referencial y pueden ser SIMPLE, PARTIAL o FULL. Se definen al establecer la clave foránea.
   - **Ejemplo**: Definir que una clave foránea debe coincidir completamente con la clave primaria de la tabla referenciada.

     ```
     sqlCopy code
     CREATE TABLE Ejemplo (
         id INT PRIMARY KEY,
         id_referencia INT,
         FOREIGN KEY (id_referencia) REFERENCES OtraTabla(id) MATCH FULL
     );
     ```

3. **Especificaciones Declarativas**:
   - **Explicación**: Son restricciones adicionales que se pueden incorporar a la base de datos, ya sea como restricciones de dominio, de tupla, de tabla o a través de assertions.
   - **Ejemplo**: Especificar que el sueldo de un empleado no puede ser menor que el salario mínimo legal.

     ```
     sqlCopy code
     ALTER TABLE Empleado
     ADD CONSTRAINT ck_sueldo_minimo
     CHECK (sueldo >= 1000);
     ```

Estos elementos son fundamentales para mantener la integridad y la consistencia de los datos en una base de datos, y cada uno tiene su papel específico en el aseguramiento de la calidad de los datos.

Restricciones Declarativas (SQL2):

1. **RI de Dominio o Atributo**: Esta restricción se implementa de manera declarativa al definir un dominio con un conjunto específico de valores válidos para un atributo.
2. **RI de Tupla**: La restricción de tupla se implementa de manera declarativa mediante la cláusula `CHECK` en la creación o modificación de la tabla.
3. **RI de Tabla**: Al igual que la restricción de tupla, la restricción de tabla se define de manera declarativa mediante la cláusula `CHECK` en la creación o modificación de la tabla.
4. **RI Globales (ASSERTIONS)**: Las assertions también se definen de manera declarativa utilizando la cláusula `CREATE ASSERTION`.
5. **Especificaciones Declarativas**: Las especificaciones adicionales, como las restricciones de dominio, también se implementan de manera declarativa utilizando las cláusulas `CREATE DOMAIN` o `ALTER TABLE ... ADD CONSTRAINT`.

Restricciones Procedurales:

1. **Triggers**: Los triggers son procedimientos almacenados que se ejecutan en respuesta a ciertos eventos. Aunque los triggers pueden utilizarse para aplicar restricciones de integridad referencial, su implementación implica lógica procedural y no son puramente declarativos.

Triggers

Definición y Utilidad:

- **Mantener Datos Derivados:** Los triggers se usan para generar datos automáticamente cuando se realizan ciertas operaciones en la base de datos.
- **Forzar Reglas de Integridad o Negocio:** Permiten implementar reglas complejas que no se pueden declarar directamente en la base de datos, y realizar acciones específicas de corrección.
- **Propagación de Actualizaciones:** Ayudan a propagar cambios entre tablas relacionadas.
- **Generación de Logs:** Se utilizan para registrar acciones en la base de datos para auditoría y chequeos de seguridad.
- **Mantener Vistas Actualizadas:** Actualizan automáticamente vistas cuando el DBMS no tiene esta capacidad nativa.

Procedimientos Almacenados

Definición y Ventajas:

- **STORED PROCEDURE:** Es un programa almacenado físicamente en la base de datos, y su implementación varía entre distintos DBMS.
- **Acceso Directo a Datos:** Permiten acceder directamente a los datos desde dentro del DBMS.
- **Aislamiento y Eficiencia:** Aíslan procesos comunes de las aplicaciones y mejoran la eficiencia al enviar solo resultados, reduciendo el tráfico de datos.

Desventajas:

- **Variedad de Lenguajes:** Cada proveedor de bases de datos usa su propio lenguaje procedural.
- **Estándares:** Aunque SQL3 incorporó características de procedimientos almacenados, la implementación aún no es estándar entre los distintos DBMS.

PL/pgSQL

Definición y Funcionalidad:

- **Lenguaje Procedural:** PL/pgSQL es un lenguaje imperativo utilizado en PostgreSQL que permite ejecutar sentencias SQL combinadas con estructuras de control como bucles y condiciones.
- **Funciones y Triggers:** Se pueden crear funciones que se invocan en sentencias SQL normales o mediante triggers.
- **Definición de Estructuras:** Permite definir variables, tipos y estructuras de datos complejas.
- **Argumentos y Retornos:** Las funciones pueden aceptar argumentos y devolver varios tipos de valores, incluidos registros y tablas.

Uso en Triggers:

- **Automatización:** Utiliza PL/pgSQL para especificar acciones automáticas que se ejecutan cuando ocurren ciertos eventos en la base de datos.
- **Modelo ECA (Evento-Condición-Acción):**
  - **Evento:** Operaciones en la base de datos que activan la regla.
  - **Condición:** Determina si la acción debe ejecutarse (opcional).
  - **Acción:** Puede ser una secuencia de sentencias SQL, una transacción de base de datos o un programa externo que se ejecuta automáticamente.

En resumen, tanto los triggers como los procedimientos almacenados (especialmente utilizando PL/pgSQL en PostgreSQL) son herramientas poderosas para automatizar y optimizar tareas dentro de una base de datos, asegurar la integridad de los datos, y mejorar la eficiencia y seguridad de las operaciones

Implementación de Triggers en PostgreSQL

Los triggers en PostgreSQL son mecanismos que permiten ejecutar automáticamente una función definida por el usuario en respuesta a ciertos eventos (como `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`) en una tabla o vista. Aquí se presenta un resumen detallado de los aspectos esenciales para la implementación de un trigger.

1. Definición del Trigger

La estructura básica para crear un trigger es la siguiente:

```
sqlCopiar código
CREATE TRIGGER nombre_trigger
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT | UPDATE [ OF columna1 [, columna2 ... ] ] | DELETE | TRUNCATE }
ON nombre_tabla
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condición ) ]
EXECUTE PROCEDURE nombre_funcion();
```

2. Elementos de un Trigger

- **Nombre del Trigger:** Identificador único para el trigger.
- **Activación:**
  - `BEFORE`: Se ejecuta antes de la operación que lo dispara.
  - `AFTER`: Se ejecuta después de la operación que lo dispara.
  - `INSTEAD OF`: Sustituye la operación que lo dispara (usualmente para vistas).
- **Evento:** Operación que dispara el trigger:
  - `INSERT`
  - `UPDATE` (opcionalmente especificando columnas)
  - `DELETE`
  - `TRUNCATE` (solo para tablas)

- **Tabla/Vista:** La entidad sobre la cual se define el trigger.
- **Granularidad:**
  - `FOR EACH ROW`: Se ejecuta una vez por cada fila afectada.
  - `FOR EACH STATEMENT`: Se ejecuta una vez por la sentencia SQL disparadora.
- **Condición (opcional):** Una condición SQL que debe cumplirse para que el trigger se ejecute.
- **Acción:** La función (procedimiento almacenado) que se ejecuta.

3. Creación de Funciones para Triggers

Las funciones utilizadas por los triggers deben devolver el tipo `trigger`. La estructura básica es:

```
sqlCopiar código
CREATE FUNCTION nombre_funcion()
RETURNS trigger AS $$
DECLARE
  -- Declaración de variables (opcional)
BEGIN
  -- Lógica PL/pgSQL
  RETURN NEW;  -- o RETURN OLD; dependiendo del caso
END;
$$ LANGUAGE plpgsql;
```

4. Variables Disponibles en la Función del Trigger

Dentro del cuerpo de la función, se tienen a disposición las siguientes variables:

- `NEW`: Registro que almacena la nueva fila (para `INSERT` / `UPDATE`).
- `OLD`: Registro que almacena la antigua fila (para `UPDATE` / `DELETE`).
- `TG_NAME`: Nombre del trigger.
- `TG_WHEN`: `BEFORE` o `AFTER`.
- `TG_LEVEL`: `ROW` o `STATEMENT`.
- `TG_OP`: `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`.
- `TG_TABLE_NAME`: Nombre de la tabla.

5. Ejemplo Completo

Función del Trigger:

```
sqlCopiar código
CREATE OR REPLACE FUNCTION actualizar_fecha_modificacion()
RETURNS trigger AS $$
BEGIN
  NEW.fecha_modificacion := CURRENT_TIMESTAMP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger que Usa la Función:

```
sqlCopiar código
CREATE TRIGGER trigger_actualizar_fecha
BEFORE UPDATE
ON productos
FOR EACH ROW
```

```
WHEN (OLD.precio IS DISTINCT FROM NEW.precio)
EXECUTE PROCEDURE actualizar_fecha_modificacion();
```

En este ejemplo:

- Se crea una función `actualizar_fecha_modificacion` que actualiza el campo `fecha_modificacion` con la fecha y hora actuales.
- Se crea un trigger `trigger_actualizar_fecha` que se ejecuta **antes** de una operación `UPDATE` en la tabla `productos`, pero solo si el precio cambia (`WHEN (OLD.precio IS DISTINCT FROM NEW.precio)`).

6. Comportamiento y Consideraciones

- **Atomicidad:** La acción del trigger es atómica; si alguna sentencia falla, se deshace toda la acción del trigger, incluyendo la sentencia que lo disparó.
- **Activación en Cascada:** Un trigger puede activar otro trigger, creando una cadena de activaciones. Esto debe manejarse cuidadosamente para evitar bucles infinitos.
- **Limitaciones de Triggers `BEFORE`:** No deberían contener sentencias SQL que alteren datos (`INSERT`, `UPDATE`, `DELETE`), ya que esto podría activar otros triggers `BEFORE`, llevando a una pila de acciones pendientes.

Conclusión

Los triggers son herramientas poderosas para la automatización y mantenimiento de la integridad de datos en PostgreSQL. Su implementación requiere una comprensión clara de su estructura, las variables disponibles, y el flujo de ejecución para evitar errores y bucles infinitos.

Implementación de Restricción de Integridad (RI) con Triggers

El siguiente ejemplo muestra cómo implementar una restricción de integridad para asegurar que no se puedan asignar más de 15 palabras clave (`id_articulo`) a un artículo en la tabla `CONTIENE`. Esto se logra mediante la creación de un trigger y una función asociada en PostgreSQL.

Paso 1: Definición de la Restricción con `ALTER TABLE`

Primero, añadimos una restricción de verificación a la tabla `CONTIENE` para asegurarnos de que el número de palabras clave por artículo no exceda 15. Sin embargo, esta restricción por sí sola no impide la inserción de nuevas filas que puedan violar la restricción, ya que las restricciones `CHECK` no pueden verificar condiciones en múltiples filas. Por eso, utilizamos un trigger.

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

Paso 2: Creación de la Función para el Trigger

Creamos una función que será ejecutada por el trigger. Esta función verifica el número de palabras clave asociadas a un artículo y levanta una excepción si se supera el límite de 15.

```
sqlCopiar código
CREATE OR REPLACE FUNCTION FN_MAXIMO_PL_CLAVES()
RETURNS trigger AS $$
BEGIN
```

```
  -- Contar el número de palabras clave asociadas al artículo
  IF (SELECT COUNT(*) FROM CONTIENE WHERE id_articulo = NEW.id_articulo) > 14 THEN
    -- Levantar una excepción si se supera el límite
    RAISE EXCEPTION 'Superó la cantidad de palabras claves en el artículo %', NEW.id_artic
ulo;
  END IF;

  -- Retornar la nueva fila
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Paso 3: Creación del Trigger

Finalmente, creamos el trigger que invoca la función `FN_MAXIMO_PL_CLAVES` antes de cada inserción o actualización en la columna `id_articulo` de la tabla `CONTIENE`.

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

Resumen del Proceso

1. **Restricción `CHECK`:** Aunque se define una restricción `CHECK`, PostgreSQL no la puede aplicar directamente en múltiples filas de la manera que necesitamos, por lo que esta parte es más bien ilustrativa.
2. **Función de Trigger (`FN_MAXIMO_PL_CLAVES`):** Verifica la cantidad de palabras clave para cada `id_articulo` y lanza una excepción si se supera el límite.
3. **Trigger (`TR_MAXIMO_PL_CLAVES`):** Se activa antes de cada inserción o actualización para asegurar que no se violen las reglas de integridad.

Consideraciones Adicionales

- **Atomicidad y Rendimiento:** Asegúrate de que la lógica dentro de la función del trigger sea eficiente, ya que cada inserción o actualización que afecta a `id_articulo` desencadenará esta verificación.
- **Manejo de Excepciones:** La excepción levantada en la función del trigger interrumpirá la operación, lo cual es útil para mantener la integridad de los datos pero puede necesitar manejo adicional en la lógica de la aplicación que interactúa con la base de datos.

Con esta configuración, se garantiza que la tabla `CONTIENE` no contendrá más de 15 palabras clave por artículo, asegurando así la integridad de los datos en la base de datos.