Introducción a las Bases de Datos y Bases de Datos

RESTRICCIONES DE INTEGRIDAD

TEORÍA 6

2017

Tecnicaturas TUPAR y TUDAI

RESTRICCIONES DE INTEGRIDAD (RI) - CONCEPTO

"Un SGBD debe ayudar a prevenir el ingreso incorrecto de información"

RI

- condiciones que restringen los valores en la BD
- descripción de estados correctos en tiempo de diseño
- previenen inconsistencias

forzar las RI → garantizar instancias legales de la BD

Ejemplos

- Los nombres y apellidos de los voluntarios no pueden ser nulos
- Un voluntario no puede aportar más de 10 horas semanales.
- Un voluntario puede cambiar de tarea o de institución solamente una dos veces al año.

CÓMO MANTENER LA INTEGRIDAD EN UNA BD

DBA: especifica las RI sobre los datos

- → código en las aplicaciones que acceden a los datos
- → restricciones (reglas o chequeos) en la BD que interpreta el SGBD

SGBD: evita manipulaciones de datos que no cumplan las RI

- → rechazando de la operación (alta, baja, modif.)
- → realizando acciones adicionales reparadoras

ambas respuestas deben dejar la BD en un estado consistente

CLASIFICACIÓN DE RI

Según su naturaleza:

- **Inherente -** se asumen por definición del modelo de datos y no se requiere especificaciones adicionales.
- **Implícitas -** provienen del modelo de datos (representada en el esquema) y se especifican durante la creación del esquema
- **Explícita -** establecen restricciones adicionales y se pueden incorporar a la BD:
  - → *De manera Declarativa (establece qué, no cómo)*
  - → *En forma Procedural (indica qué y cómo)*

CLASIFICACIÓN DE RI

Por los estados involucrados:

- **RI de estado -** restringe los valores que pueden tomar los datos en estado no transitorio

  ej: un sueldo no puede ser negativo, el DNI no puede repetirse, restricciones sobre los vínculos entre entidades y otras…

- **RI de transición -** restringe los posibles cambios de valores entre estados sucesivos de los datos

  ej. el sueldo no puede decrecer → implica conocer el estado "viejo" y el "nuevo" para compararlos y chequear

CLASIFICACIÓN DE RI

De Estado:

- **Unicidad**: no puede haber claves repetidas
- **No Nulidad**: el valor de un atributo no puede ser nulo
- **Dominio**: los valores de un atributo deben pertenecer a un conjunto (dominio) definido
- **Cardinalidad de una relación**: el número de veces que una entidad participa de una relación
  Ej. Los empleados sólo pueden participar en un máximo de 5 proyectos
- **Participación en una relación**: participación obligatoria u opcional en una relación
  Ej. Un empleado debe (o puede) estar vinculado a un área

RESTRICCIONES DE NO-NULIDAD Y DE UNICIDAD

- La clave de una relación R es un conjunto no vacío de atributos que identifican unívocamente cada tupla de R

- En toda relación siempre hay, al menos, una clave candidata. Puede haber más de una, en ese caso:

→ **Clave primaria**: es la clave candidata elegida para identificar unívocamente las tuplas de la relación (PRIMARY KEY en SQL)

→ **Clave/s alternativa/s**: otra/s clave/s candidata/s (UNIQUE en

Los atributos que integran la clave primaria no pueden ser ni total ni parcialmente nulos

Importante: los valores nulos representan información desconocida o inaplicable
Si algún atributo clave fuera nulo → las tuplas no podrían ser identificadas

RESTRICCIONES DE NO-NULIDAD Y DE UNICIDAD

REPASO:

- La restricción NOT NULL sobre un atributo (clave o no) hace que el DBMS rechace cualquier intento de insertar un nulo en esa columna
- Las claves primarias deben especificarse como NOT NULL
- Se puede especificar un valor por defecto en una columna mediante la cláusula DEFAULT
- Las claves primarias (PK) y alternativas (UNIQUE) se especifican en la definición de la tabla (o en la definición del atributo si es único)
- Se puede asociar varias restricciones UNIQUE a una tabla, pero sólo una restricción PRIMARY KEY

RESTRICCIONES DE NO-NULIDAD Y DE UNICIDAD

En SQL: **CREATE TABLE** NombreTabla
({ nom_col TipoDato [NOT NULL] [DEFAULT valorDefecto], … }
[[CONSTRAINT PK_nom] **PRIMARY KEY** (lista_col_PK),]
{ [[CONSTRAINT nom_restr] **UNIQUE** (lista_col),] }
…);

{}: repetición
[]: opcional

Recomendable!

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

ídem UNIQUE

Ejemplos

Empleado (TipoE, NroE, Nombre, …) Proyecto (IdProy, NomProy, …)

CREATE TABLE Empleado
(TipoE char(20) NOT NULL,
NroE integer NOT NULL,
nombre char(50) NOT NULL, ….);

ALTER TABLE Empleado
ADD CONSTRAINT PK_Emp
PRIMARY KEY (TipoE, NroE));

CREATE TABLE Proyecto
(IdProy integer NOT NULL
NomProy char(100) NOT NULL,
…..
PRIMARY KEY (IdProy),
UNIQUE (NomProy),
……);

RESTRICCIONES DE INTEGRIDAD REFERENCIAL (RIRS)

- Una clave extranjera (FOREIGN KEY en SQL) de una tabla *A* (***referenciante***) es un conjunto no vacío de atributos cuyos valores coinciden con los valores de otro conjunto de atributos, que son clave de otra tabla B (***referenciada***)
  Nota: *R* y *R´* podrían coincidir

- Las claves extranjeras (FK) especifican relaciones entre entidades y las RIRs permiten mantener la consistencia entre tuplas de esas entidades

El conjunto de valores de la clave extranjera de una tabla A debe coincidir con otro conjunto de atributos en *B* al que hace referencia, o bien ser nulo

si es posible (por la definición de los datos)

RESTRICCIONES DE INTEGRIDAD REFERENCIAL (RIRS)

Formalmente: *A* [*lista_col_referenciante*] << *B* [*lista_col_referenciada*]: (b,m)

donde: b= acción referencial ante baja
y m=acción referencial ante modificación (a derecha)

- *lista_col_referenciante* es una lista de atributos que se corresponden con una clave (primaria o alternativa) de la tabla referenciada B
- *lista_col_referenciante* y *lista_col_referenciada* deben tener igual número de columnas y tipos de datos compatibles (no requiere igualdad de nombres)
- *lista_col_referenciada* puede omitirse si es la PK de *B*
- Si la FK es de único atributo, puede especificarse en su definición

RESTRICCIONES DE INTEGRIDAD REFERENCIAL (RIRS)

En SQL: **CREATE TABLE** NombreTabla
({ nombre_columna TipoDato [NOT NULL] … }
[[CONSTRAINT PK_nom] PRIMARY KEY (lista_columnasPK),]
{ [[CONSTRAINT U_nom] UNIQUE (lista_columnas),] }
{ [[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

RIRS – ACCIONES REFERENCIALES

R [*lista_col_referenciante*] << *R'* [*lista_col_referenciada*]: (**b**, m)

Qué hacer al intentar borrar una tupla referenciada en *R* por la FK en *R´*?

→ **Rechazar la operación**

- ✓ **NO ACTION:** no permite borrar la tupla referenciada en *R´* si hay referencias en *R (es la opción por defecto)*
- ✓ **RESTRICT :** misma semántica pero se chequea antes de otras RI

→ **Aceptar la operación y realizar acciones reparadoras adicionales**

- ✓ **CASCADE:** borra la tupla referenciada en *R´* y se propaga el borrado a las tuplas que la referencian mediante la FK en *R*
- ✓ **SET NULL:** borra la tupla referenciada en *R´* y las tuplas que la referenciaban ponen a nulo la FK (sólo si admite nulos)
- ✓ **SET DEFAULT:** borra la tupla referenciada en *R´* y las tuplas que la referenciaban ponen en la FK el valor por defecto definido para la misma

RIRS – ACCIONES REFERENCIALES

R [*lista_col_referencia*] << *R´* [*lista_col_referenciada*]: (b, **m**)

Qué hacer si se intenta modificar el valor de la clave de la tupla referenciada en *R´* por la FK en *R*? (modificación a derecha)

→ **Rechazar la operación**

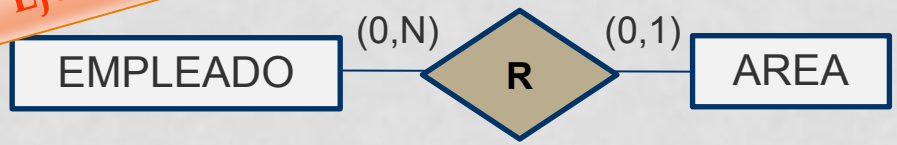
- ✓ **NO ACTION:** no permite modificar el valor de la clve en la tupla referenciada en *R´* si hay referencias en *R* *(es la opción por defecto)*
- ✓ **RESTRICT :** misma semántica pero se chequea antes de otras RI

→ **Aceptar la operación y realizar acciones reparadoras adicionales**

- ✓ **CASCADE:** modifica el valor de la clave de la tupla referenciada en *R´* y propaga la modificación a las tuplas que la referencian en *R*
- ✓ **SET NULL:** modifica la tupla referenciada en *R´* y las tuplas que la referenciaban ponen a nulo la FK (sólo si admite nulos)
- ✓ **SET DEFAULT:** modifica la tupla referenciada en *R´* y las tuplas que la referenciaban ponen en la FK el valor por defecto definido

RIRS – ACCIONES REFERENCIALES

Ejemplo



EMPLEADO(idE,nombre,..,AreaT) **AREA(idArea, …)**

```
CREATE TABLE Empleado (…);
CREATE TABLE Area (…);
ALTER TABLE Empleado
  ADD CONSTRAINT FK_R
  FOREIGN KEY (AreaT) REFERENCES Area
      ON UPDATE ….
      ON DELETE ….;
```

Cómo proceden las sig. operaciones considerando las diferentes acciones referenc.? (considerar la instancia dada para las tablas y result. individuales, no acumulativos)

- **DELETE FROM Area WHERE IdArea= 101;**
- **DELETE FROM Area WHERE IdArea= 102;**
- **DELETE FROM Area;**
- **UPDATE Area set IdArea = 201 where IdArea= 101;**
- **UPDATE Area set IdArea = 202 where IdArea= 102;**

EMPLEADO			
IdE	Nombre	…	AreaT
1	E1	…	101
2	E2	…	101

AREA	
IdArea	…
101	…
102	…

Desarrollo en clase

RIRS – ACCIONES REFERENCIALES

Ejemplo

Empleado (TipoE, NroE, Nombre, Cargo)
Proyecto (IdProy, NomProy, AñoComienzo, AñoFinal)
TrabajaEn (TipoE, NroE, IdProy, Mes, Año, Cant_horas, Tarea)
Auspicio (IdProy, NombreAuspiciante, TipoEContacto, NroEContacto)

RIRs

R1: TrabajaEn (TipoE, NroE) << Empleado (TipoE, NroE) : [C, R] *[baja, mod.der.]*
R2: TrabajaEn (IdProy) << Proyecto (IdProy) : [R, R]
R3: Auspicio (IdProy) << Proyecto(IdProy): [R, R]
R4: Auspicio (TipoEContacto,NroEContacto) << Empleado (TipoE,NroE): [N, R]

Desarrollo en clase

→ *activación de RIRs (suponer existencia de tupla/s asociada/s) :*

delete from Proyecto **where** IdProy = 13;

update Proyecto **set** IdProy = 78 **where** IdProy = 13;

RIRS – TIPOS DE MATCHING

R [*lista_col_referencia*] << *R´* [*lista_col_referenciada*]

- 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

RIRS – TIPOS DE MATCHING

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

Ninguna de las columnas de la FK es NULL y existe una tupla en la tabla referenciada cuyos valores de clave coinciden con los de tales columnas, o

- Al menos una de las columnas en la FK es NULL (MATCH SIMPLE)
- Los valores de los atributos no nulos de la FK se corresponden con los correspondientes valores de la clave, al menos en una tupla de la tabla referenciada (MATCH PARTIAL)
- Todas las columnas de la FK son NULL (MATCH FULL)

RIRS – ACCIONES REFERENCIALES

Ejemplo

Empleado (TipoE, NroE, Nombre, Cargo)
Proyecto (IdProy, NomProy, AñoComienzo, AñoFinal)
TrabajaEn (TipoE, NroE, IdProy, Mes, Año, Cant_horas, Tarea)
Auspicio (IdProy, NombreAuspiciante, TipoEContacto, NroEContacto)

HACER PARA LOS MATCHING!!

R1: TrabajaEn (TipoE, NroE) << Empleado (TipoE, NroE) : [C, R] *[baja, mod.der.]*

RIRs

R2: TrabajaEn (IdProy) << Proyecto (IdProy) : [R, R]

R3: Auspicio (IdProy) << Proyecto(IdProy): [R, R]

R4: Auspicio (TipoEContacto,NroEContacto) << Empleado (TipoE,NroE): [N, R]

Especificación de RIR:

```
ALTER TABLE Auspicio
ADD CONSTRAINT FK_R4
FOREIGN KEY (TipoEContacto,NroEContacto)
   REFERENCES Empleado
   ON UPDATE RESTRICT
   ON DELETE SET NULL;
```

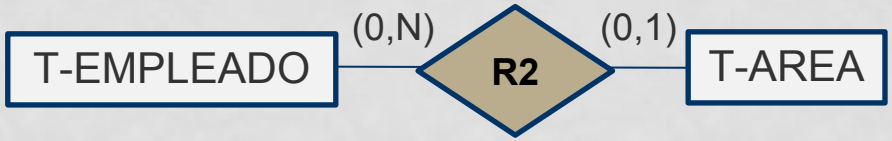
Operaciones → activación de RIRs:
(suponer existencia de tupla/s asociada/s)

```
delete from Proyecto where IdProy = 13;
update TrabajaEn set IdProy = 78 where
   IdProy = 13;
```

Desarrollo en clase

RIRS – TIPOS DE MATCHING

Ejemplo



EMPLEADO(idE,..,CodA,TipoA) **AREA(idArea, TipoA, …)**

Analizar la posibilidad de alta de las sig. tuplas en T-EMPLEADO según los distintos tipos de matching (suponiendo que la FK admita nulos)

T-AREA

IdArea	TipoA	…
a1	b1	…
a1	b2	…
a2	b1	…

T-EMPLEADO

IdE	…	CodA	TipoA
1	…	a1	b1
2	…	a2	b2
3	…	null	null
4	…	null	b1
5	…	a3	null

MATCHING		
Simple	**Parcial**	**Full**
ok	ok	ok
X	X	X
ok	ok	ok
ok	ok	X
ok	X	X

Desarrollo en clase

OTRAS RESTRICCIONES DE INTEGRIDAD EN SQL

- Además de las anteriores, se puede requerir otras RI específicas sobre los datos según la estrategia de funcionamiento de la organización
- La especificación declarativa de RI sigue la estructura jerárquica del modelo relacional (atributo→tupla→tabla→BD):

  → **RI Dominio** (DOMAIN)

  → **RI de tabla asociada a uno ó más atributos** (CHECK de tupla)

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

  → **RI generales de la base de datos** (ASSERTION)

- Se **activan** siempre que se realice alguna operación sobre los datos afectados por la restricción
- Su incumplimiento promueve el **rechazo** de la operación

OTRAS RESTRICCIONES DE INTEGRIDAD EN SQL

- Otra alternativa para especificar RI → SQL Procedural:

  DISPARADORES (**TRIGGERS**)

  → Es una pieza de código almacenada en la BD que "se dispara" automáticamente ante la ocurrencia de algún evento

  PROCEDIMIENTOS

  FUNCIONES

- Recurso útil ante la imposibilidad de definir en los DBMS:
  - ✓ restricciones complejas en forma declarativa
  - ✓ ciertas acciones referenciales
  - ✓ acciones específicas de reparación

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);
```

La condición debe evaluar como VERDADERA o DESCONOCIDA

RI DE DOMINIO / ATRIBUTO

Pueden plantearse distinto tipo de condiciones:

- **Comparación simple:** operadores (=,<,>,<=,>=,<>) Ej: Sueldo>0
- **Rango:** [NOT] BETWEEN (incluye extremos) Ej: nota BETWEEN 0 AND 10
- **Pertenencia:** [NOT] IN Ej: Area IN ('Académica', Posgrado', 'Extensión')
- **Semejanza de Patrones:** [NOT] LIKE

  % (para 0 o más caracteres) Ej: LIKE 's%' - (para un carácter simple) Ej: LIKE 's_'
- **Test de Nulidad** → IS [NOT] NULL Ej: FechaIngreso IS NOT NULL

→ **AND, OR** se utilizan para concatenar distintas condiciones

→ Se antepone **NOT** para negarlas

RI DE DOMINIO / ATRIBUTO

El sueldo de un empleado es un valor no nulo, mayor a 0 e inferior a 50000,
2 decimales

Ejemplo

EMPLEADO(idE,.., sueldo)

```
CREATE DOMAIN SueldoValido
 AS Numeric (7,2) NOT NULL
 CHECK (value BETWEEN 0 AND 50000);
```

o
```
CREATE TABLE Empleado
( …. ,
  sueldo Numeric (7,2) NOT NULL
   CHECK (sueldo BETWEEN 0 AND 50000),
```

o
```
sueldo SueldoValido,,

    …. );
```

```
CREATE TABLE Empleado
( …. ,
  sueldo SueldoValido,  ...  );
```

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)

En SQL: **CREATE TABLE** NombreTabla
(.....
{ [[CONSTRAINT nom_restr] CHECK (condición)] });

La condición debe evaluar como VERDADERA o DESCONOCIDA

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

RI (CHECK) DE TUPLA

Ejemplo

Un empleado o bien no ha ascendido o, si ha ascendido, la fecha de ascenso no puede ser anterior a la fecha de ingreso

EMPLEADO(idE,.., FechaAscenso, FechaIngreso)

```
ALTER TABLE Empleado
 ADD CONSTRAINT Ascenso
 CHECK ( (FechaAscenso IS NULL)
          OR (FechaIngreso < FechaAscenso ));
```

RI (CHECK) DE TABLA

- Representa una restricción que afecta diferentes tuplas de una misma tabla
- Ámbito de la restricción: tabla
- Casos particulares: PRIMARY KEY, UNIQUE (a nivel tabla)

Ejemplo

El premio asignado a un empleado no debe superar el 20% del promedio de todos los sueldos

EMPLEADO(idE,.., premio)

```
ALTER TABLE Empleado
 ADD CONSTRAINT premio_max
  CHECK ( premio < (SELECT AVG(sueldo)*0.2 FROM Empleado));
```

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);

La condición debe evaluar como VERDADERA o DESCONOCIDA

- Su activación *se daría* ante actualizaciones sobre las tablas involucradas
- *Requerirían* alto costo para comprobación y mantenimie

→ los DBMS comerciales no soportan ASSERTIONS !

RI GLOBALES (ASSERTIONS)

Ejemplo

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 ) );
```

SQL no proporciona un mecanismo para expresar la condición «para todo X, P(X)»

(P=predicado) → se debe utilizar su equivalente «no existe X tal que no P(X)»

UNA ALTERNATIVA PARA ESPECIFICAR RESTRICCIONES

✓ imposibilidad de utilizar assertions en DBMS

✓ carencia de implementación de ciertas acciones referenciales

✓ no disponibilidad de acciones específicas diferentes al rechazo y la reparación estándar

necesidad de una herramienta útil para escribir restricciones complejas, acciones específicas de etc.

→ Triggers (disparadores)

RI MEDIANTE TRIGGERS

- **Trigger**: pieza de código (no declarativo) almacenada que "se dispara" automáticamente ante la ocurrencia de un evento sobre la base de datos
- Puede considerarse una regla *evento-condición-acción* (ECA)
- Es persistente y accesible para todas las operaciones de la BD (según se haya definido)

cuando ocurre el ***Evento*** (sentencia o situación que dispara su ejecución)

… se evalúa la ***Condición*** (expresión booleana que debe evaluar en VERDADERO para que el trigger se active. Si evalúa en FALSO o DESCONOCIDO no se ejecuta. *Solo para Triggers a nivel fila*)

… y si se satisface se ejecuta la ***Acción*** (procedimiento que

TRIGGERS – SINTAXIS SQL-99

CREATE [OR REPLACE] TRIGGER <nombre del trigger>
BEFORE | **AFTER** | **INSTEAD OF** ← tiempo de activación
INSERT| **DELETE**| **UPDATE** [**OF** <columna/s >] ← **EVENTO**
ON <nombre tabla/vista>
[**REFERENCING OLD** | **NEW** [**ROW**|**TABLE]** [**AS**] alias]
[**FOR EACH ROW** | **FOR EACH STATEMENT**] ← granularidad
[**WHEN** (condición)] ← **CONDICIÓN**
BEGIN ATOMIC <2 o más sentencias> **END** | sentencia-SQL; ← **ACCIÓN**

- El uso de **OR REPLACE** permite sobreescribir un trigger existente. Si se omite, y el trigger existe, se producirá, un error.
- Para eliminación de un trigger existente: **DROP TRIGGER** <nombre trigger>

→ *Cada DBMS fue proponiendo su propia solución procedural y SQL-1999 incorporó algunas de esas características… pero no todo lo definido antes se ajustó al estándar*

TRIGGERS – ACTIVACIÓN Y EVENTOS

EVENTO: puede ser una operación de actualización sobre la tabla o vista a la que está asociado el trigger:

- Inserción (INSERT)
- Actualización (UPDATE): se puede especificar columna/s
- Eliminación (DELETE)

TIEMPO DE ACTIVACIÓN: en relación a la sentencia que lo activa, el trigger puede ejecutarse:

- antes de la sentencia disparadora (BEFORE)
- después de la sentencia disparadora (AFTER)
- en lugar de la sentencia disparadora (INSTEAD OF)
  - si la relación es una vista, se puede usar para ejecutar sus actualizaciones y propagar las modificaciones a las tablas base

TRIGGERS – GRANULARIDAD

Los triggers pueden ser:

- FOR EACH ROW: se ejecuta una vez por cada fila afectada
- FOR EACH STATEMENT: se ejecuta una vez para la sentencia SQL disparadora, independientemente de la cantidad de filas que afecte

  (Por defecto → FOR EACH STATEMENT)

TRIGGERS – REFERENCIAS

La sentencia INSERT manipula una *nueva* fila (si el trigger es FOR EACH ROW) o un *nuevo conjunto de filas* (si es FOR EACH STATEMENT)

DELETE manipula una fila *vieja* (para triggers a nivel fila) o un conjunto de filas o tabla *vieja* (para triggers de sentencia)

UPDATE manipula estados *viejos* y *nuevos*, tanto de filas como de conjuntos de filas, según corresponda

Corresponde referirse a estos elementos como :new y :old
o establecer alias mediante:

REFERENCING [NEW | OLD] [ROW | TABLE] AS <alias>

TRIGGERS – ACCIÓN

- La acción consiste en una sentencia SQL aislada o un conjunto de sentencias, delimitadas en un bloque BEGIN . . . END
- Puede referirse a valores anteriores y nuevos que se modifican, nuevos que se insertan, o anteriores que se eliminaron, según el evento que desencadenó la acción
- Pueden incluir sentencias de control (IF … ELSE, FOR, WHILE, …)
- No pueden incluir sentencias del DDL (*CREATE, ALTER, DROP)*
- Un trigger BEFORE no debería contener sentencias SQL que alteren datos (*INSERT, UPDATE, DELETE*): esto puede disparar otros triggers BEFORE (sus acciones van quedando pendientes)
- La acción del trigger es un procedimiento atómico → Si cualquier sentencia del cuerpo del trigger falla, la acción completa del trigger se deshace, incluyendo las correspondientes a la sentencia que lo disparó

TRIGGERS – COMPORTAMIENTO

- La acción del trigger es un procedimiento atómico ….
- Ante un cierto evento sobre una tabla → pueden activarse varios triggers!
- Se puede producir una activación de triggers en cascada → Si la activación de un trigger *T1* dispara otro trigger *T2*: se suspende la ejecución de *T1*, se ejecuta el trigger anidado *T2* y luego se retoma la ejecución de *T1*

  → esto podría dar lugar a una cadena "infinita" de activaciones!

SQL statement
UPDATE T1 SET …;

UPDATE_T1 Trigger
BEFORE UPDATE ON T1
FOR EACH ROW
INSERT INTO T2 VALUES (...);

INSERT_T2 Trigger
BEFORE INSERT ON T2
FOR EACH ROW
INSERT INTO ... VALUES (...);

Los DBMS suelen limitar la longitud de las cadenas de disparadores

TRIGGERS – UTILIDAD

Los *triggers* se pueden usar para:

- Mantener datos derivados - Generación automática de datos
- Forzado de reglas de integridad o del negocio complejas *(Ej. cuando no es posible incluirlas declarativamente)* o con acciones específicas de reparación (*diferentes al rechazo y la reparación estándar*)
- Propagación de actualizaciones
- Generación de logs para soporte de auditoría de las acciones de la base de datos y chequeos de seguridad
- Mantener vistas actualizadas (*cuando el DBMS no provee capacidades para hacerlo*)

TRIGGERS – EJEMPLOS

✓ **Forzado de reglas de integridad**

create trigger <*nombre*>
before <*operación crítica sobre la BD*>
when <*condición por la que una RI es incumplida*>

< acción(es) del trigger > →rechazo (acción pasiva) /reparación (acción activa)

Ejemplo

Verificar que el sueldo de un empleado no se reduzca

```
CREATE TRIGGER sueldo_no_se_reduce
BEFORE UPDATE OF sueldo ON Empleado
FOR EACH ROW
WHEN (:old.sueldo > :new.sueldo)
BEGIN ATOMIC
    SIGNAL ('No se puede reducir el salario de un empleado');
END;
```

TRIGGERS – EJEMPLOS

✓ Actualización (automática) de datos derivados

Ejemplo

Mantener *automáticamente* la cantidad total de empleados del Area (ante altas, bajas o modificaciones en Empleado)

EMPLEADO(idE,nombre,..,AreaT) AREA(idArea, … CantEmp)

```
CREATE TRIGGER Incrementar_EmpArea
AFTER INSERT ON Empleado
FOR EACH ROW
BEGIN ATOMIC
  UPDATE Area SET CantEmp = CantEmp +1
     WHERE Area.IdArea = :new.AreaT;
END;
```

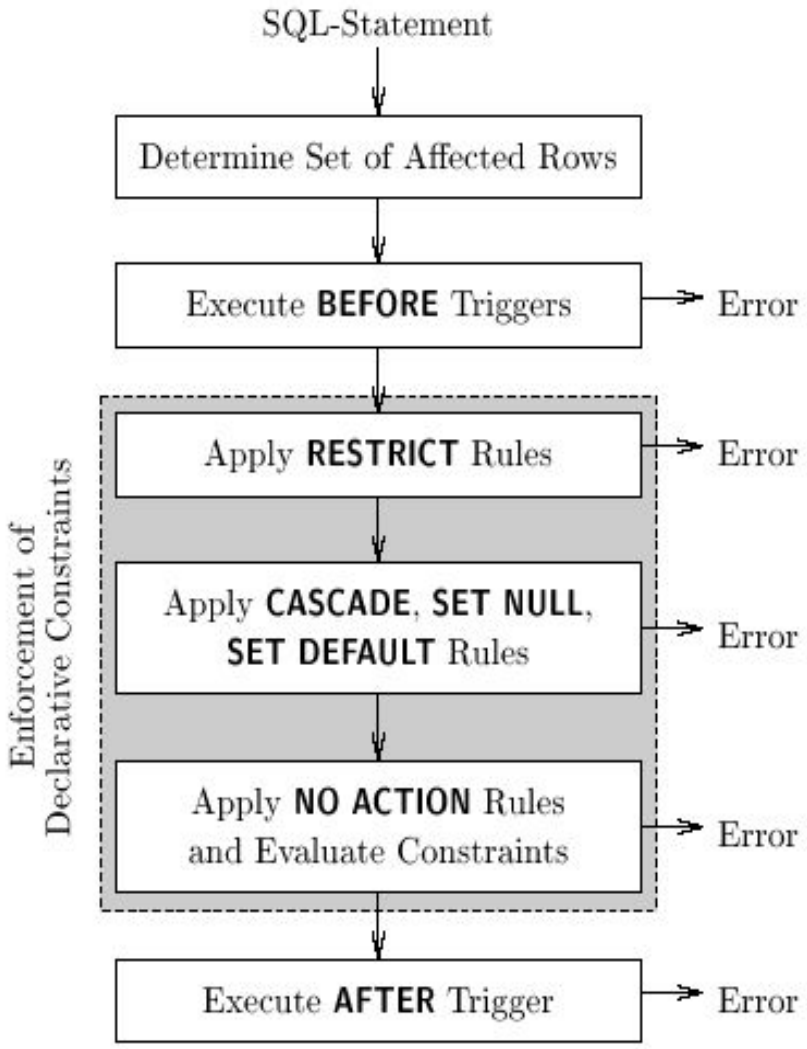
```
CREATE TRIGGER Decrementar_EmpArea
AFTER DELETE ON Empleado
FOR EACH ROW
BEGIN
  UPDATE Area SET CantEmp = CantEmp -1
     WHERE Area.IdArea = :old.AreaT;
END;
```

```
CREATE TRIGGER Modificar_EmpArea
AFTER UPDATE OF AreaT ON Empleado
FOR EACH ROW
……..
```

TRIGGERS vs. RI DECLARATIVAS

- Triggers → permiten *definir y forzar* reglas de integridad, pero *NO son una restricción de integridad*
- Un trigger definido para forzar una RI no verifica su cumplimiento para los datos ya almacenados en la BD (una RI declarativa verifica la *carga existente* en la BD)
- Los triggers deberían usarse sólo cuando una RI no puede ser expresada mediante una cláusula declarativa

MODELO DE EJECUCIÓN SQL-99 (+ TRIGGERS)



De: *"Semantic Integrity Support in SQL-99 and Commercial (Object-) Relational Database Management Systems"* (Türker y Gertz)

1. Ejecuta todos los triggers **BEFORE-statement**
2. Realiza un ciclo por todas las filas afectadas por la sentencia SQL
   a. Ejecuta todos los triggers **BEFORE-row**
   b. Bloquea y actualiza cada fila y ejecuta los chequeos de integridad declarat. *(El bloqueo no se levanta hasta el final de la transacción)*
   c. Ejecuta todos los triggers **AFTER-row**
3. Completa las acciones correspondientes a la verificación diferida de integridad expresada declarativamente
4. Ejecuta todos los triggers **AFTER-statement**

BIBLIOGRAFÍA

Date, C., "An Introduction to Database Systems". 7º ed., Addison Wesley, 2000

Elmasri, R., Navathe, S., "Fundamentals of Database Systems", Addison Wesley, 2011

Silberschatz, A., Korth, H, Sudarshan, S., "Database System Concepts", McGraw Hill, 2001

Sumathi S., Esakkirajan S., Fundamentals of Relational Database Management Systems, 2007