Bases de Datos I

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

Restricciones de Integridad
Parte 2

INGENIERÍA DE SISTEMAS

Cursada 2019

Restricciones de Integridad
Declarativas en SQL

- Restricciones específicas sobre los datos → **reglas de negocio**
- La **especificación declarativa** de estas RI sigue la estructura jerárquica del modelo relacional (atributo→ tupla→ tabla→ BD)

  → **RI de dominio (DOMAIN)**
  → **RI de tabla asociada a uno ó + atributos (CHECK de tupla)**
  → **RI de tabla asociada a varias tuplas (CHECK de tabla)**
  → **RI generales de la base de datos (ASSERTION)**

- Se **activan** al realizar alguna operación (alta, baja, modificación) sobre los objetos afectados (atributo/s - tabla/s) y su incumplimiento promueve el rechazo de la operación por parte del SGBD

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 de atributo en la sentencia CREATE TABLE

```
CREATE TABLE NomTabla (...
  NomAtrib  TipoDato [ DEFAULT ValorDefecto ]
  [ [CONSTRAINT NomRestriccion] CHECK (condicion)]...);
```

- o definirlas en un dominio y declarar el atributo perteneciente al dominio

```
CREATE DOMAIN  NomDominio
AS TipoDato  [ DEFAULT ValorDefecto ]
[ [CONSTRAINT NomRestriccion] CHECK (condicion)];

CREATE TABLE NomTabla (
... NomAtrib  NomDominio ... );
```

RI - Condiciones en el CHECK

- Para que una RI se cumpla → la condición en el check **no debe evaluar falsa** (sino verdadera o desconocida, ante presencia de nulos)

- 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 negar la condición

RI de Dominio/Atributo

Ejemplo

El sueldo de un empleado es un valor no nulo, mayor a 0 y con 2 decimales

EMPLEADO (idE, ..., sueldo)

```
CREATE DOMAIN SueldoValido
 AS Numeric (8,2) NOT NULL
     CHECK (value > 0);
```

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

o
```
CREATE TABLE Empleado
( idE integer, …. ,
  sueldo Numeric (8,2) NOT NULL
  CONSTRAINT SueldoValido CHECK (value > 0),
  …. );
```

RI (check) de Tupla

- Es 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 modifica en la tabla)

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

o:

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

RI (check) de Tupla

Ejemplo

La fecha de ascenso de un empleado o es posterior a la fecha de ingreso o es nula

EMPLEADO(idE, ..., FechaAscenso, FechaIngreso)

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

RI (check) de Tabla

- Restricción que afecta diferentes tuplas de una misma tabla
- Ámbito de la restricción: **tabla** (la RI se comprueba cada vez que se actualiza la tabla)
- Casos particulares: PRIMARY KEY, UNIQUE (a nivel tabla)
- Se define de manera similar al check de Tupla

Ejemplo

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

EMPLEADO (idE,…, sueldo, premio)

```
ALTER TABLE Empleado
  ADD CONSTRAINT CHK_premio_max
  CHECK ( premio <= (SELECT avg(sueldo)*0.2 FROM Empleado));
```

Nota: PosgreSQL no permite consultas dentro del check !

RI Generales (ASSERTIONS)

- Son 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
- Sintaxis en SQL estándar:

```
CREATE ASSERTION  NomAssertion CHECK (condición);
```

- Su activación *se produciría* ante actualizaciones sobre cualquiera de las tablas involucradas

→ alto costo para comprobación y mantenimiento

pero… **los SGBD no soportan ASSERTIONS !**
(se deben utilizar otros recursos)

RI Generales (ASSERTIONS)

Ejemplo

El sueldo de cada empleado de un proyecto no puede superar el presupuesto de ese proyecto

EMPLEADO (idE, ..., sueldo, proy) PROYECTO (IdProy, …., presupuesto)

La RI afecta 2 tablas: se debe comparar el sueldo de cada empleado contra el presupuesto del proyecto en que trabaja

¿Cómo chequear que **la RI se satisfaga para todos los casos**?

```
CREATE ASSERTION sueldo_valido
CHECK ( NOT EXISTS ( SELECT 1 FROM Empleado E JOIN Proyecto P
                     ON (E.proy = P.IdProy)
                      WHERE  E.sueldo > P.presupuesto );
```

empleados con sueldo mayor al presupuesto del proyecto (no cumplen la condición)

Si los datos de las tablas satisfacen la RI ☐ la consulta resultaría en una tabla vacía y entonces la condición del CHECK se satisface

SQL no proporciona mecanismos para expresar la condición "para todo X, se cumple C" → se debe utilizar su equivalente "no existe X tal que no cumpla C"

Incorporación de restricciones:

```
CREATE TABLE NombreTabla
    ( { nombre_columna  TipoDato | nomDom [NOT NULL] [DEFAULT valorDef]
                [ [CONSTRAINT nom_restr] CHECK (condicion)]  ... ,}
      [ [CONSTRAINT nom_restr_PK] PRIMARY KEY (lista_columnasPK),]
      { [ [CONSTRAINT nom_restr_U] UNIQUE (lista_columnas),] }
      { [ [CONSTRAINT nom_restr_FK] FOREIGN KEY (lista_col_referencia)
            REFERENCES nombreTablaRef [(lista_col_referenciada)]
            [ MATCH {FULL | PARTIAL | SIMPLE} ]
            [ ON UPDATE AccionRef]  [ ON DELETE AccionRef] ], }
      { [ [CONSTRAINT nom_restr ] CHECK (condicion),] }
    );
```

Nota:
[...] opcional
{...} uno o más
| posibilidades

```
ALTER TABLE NombreTabla
      ADD [ CONSTRAINT nom-restr ] restriccion ;

donde restriccion =  PRIMARY KEY (lista_columnasPK) |
                     UNIQUE (lista_columnas) |
                     FOREIGN KEY (lista_col_referencia) .... |
                     CHECK (condicion)
```

```
CREATE DOMAIN NomDominio AS TipoDato [ DEFAULT ValorDefecto ]
  [ [CONSTRAINT NomRestriccion] CHECK (condicion)];
```

```
CREATE ASSERTION  NomAssertion CHECK (condicion);
```

Eliminación de restricciones:

- Es posible eliminar cualquier restricción previamente declarada… si se conoce su nombre → por esto es recomendable dar explícitamente un nombre a cada restricción que se especifique!
- Si se quiere modificar una restricción conviene eliminarla y volver a definirla

```
ALTER TABLE NombreTabla
      DROP CONSTRAINT NomRestriccion [ RESTRICT|CASCADE];
```

```
ALTER DOMAIN NomDominio
    DROP CONSTRAINT NomRestriccion [RESTRICT|CASCADE];
```

```
DROP ASSERTION  NomAssertion;
```

RESTRICT: se rechaza si hay objetos dependientes (opción por defecto);
CASCADE: procede y genera un efecto en 'cascada" sobre las dependencias (según cual sea el tipo de eliminación)

Consideraciones

- ✔ Ante el planteo de una RI, siempre es recomendable evaluar los siguientes aspectos:
  - Analizar el **ámbito adecuado** de la restricción: ¿afecta a un atributo (RI de atributo), a una fila (RI de tupla), a una tabla o parte de ella (RI de tabla), a varias tablas (RI general)?
  - Entonces **identificar el recurso declarativo** más apropiado para implementar cada restricción
    - según el estándar SQL-1999: DOMAIN, CHECK, ASSERTION
    - considerando las posibilidades del DBMS en particular
- ✔ Los nombres de restricciones deben ser únicos dentro del esquema en que están definidas
- ✔ Al declararse una nueva restricción en la BD, el SGBD verifica si los datos existentes satisfacen la condición (si no, será rechazada)

Consideraciones

Tener presente que:

- Si la condición que define una RI evalúa **falso□ no se cumple**
- Si evalúa en **Verdadero o Desconocido □ se satisface**

Ante una operación, el SGBD verifica todas las RI definidas que son afectadas y si alguna no se satisface:

→ se cancela la ejecución de la sentencia que provocó la activación (rechazo)

→ se revierte la transacción (rollback) sin quedar efectos parciales debido a la ejecución de la sentencia original

→ la BD queda siempre en estado consistente

Consideraciones

Al implementar las restricciones tener la precaución de **involucrar adecuadamente a todos los objetos afectados** en la RI

Ejemplo

Se requiere chequear que los sueldos de los empleados no superen al de Tareas de Supervisores

Si se define la restricción de tabla:

ALTER TABLE **Empleado** ADD CONSTRAINT sueldo_valido
CHECK (sueldo <= SELECT min(salario) FROM **Tarea** where nombre LIKE 'Supervisor%');

¿Cuándo se verifica la RI?

→ al actualizar sobre la tabla a la que la RI está asociada (**Empleado**)
→ pero si se actualiza sobre la tabla **Tarea** la RI no se activará

- Planteo de RI sobre una tabla → problemas con subconsultas que refieren a otra tabla: podrían obtenerse **estados inconsistentes inadvertidos**!
- Se debería plantear una restricción simétrica en la otra tabla (por eso si interviene más de una tabla la solución estándar es plantear una ASSERTION)
- Pero **¿qué hacer si el DBMS no permite implementar ciertas restricciones?**

Una alternativa para especificar RI

Los SGBD no siempre brindan recursos declarativos para:

- implementar restricciones de tabla complejas
- establecer restricciones generales (assertions)
- definir ciertas acciones referenciales o de matching
- implementar acciones específicas de reparación o actualización

y otros requerimientos…

Solución? → programar usando SQL procedural

SQL ofrece lenguaje PL/SQL para escribir *Persistent Stored Modules*:

- **Triggers:** activados automáticamente
- **Stored procedures:** invocados explícitamente por el usuario
- **Functions:** predefinidas o definidas por el usuario, pueden ser invocadas desde un trigger, stored procedure o explícitamente

Disparadores (TRIGGERS)

- Recurso procedural (no declarativo): pieza de código almacenado en la BD que se **dispara automáticamente** ante un evento
- Se considera una regla del tipo ***evento-condición-acción* (ECA)**

*Cuando ocurre el **Evento*** (que dispara su ejecución)

*se evalúa la **Condición** =* expresión 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*** =conjunto de sentencias SQL

- Un trigger se declara sobre una tabla o vista específica
- Se pueden declarar múltiples triggers sobre una misma tabla

Declaración de Triggers

Según estándar SQL:

CREATE [OR REPLACE] TRIGGER nombre_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 de activación**
BEGIN ATOMIC {*sentencia-SQL*} **END** | *sentencia-SQL*; ← **ACCIÓN**

Para eliminar un trigger: **DROP TRIGGER** <nombre trigger>

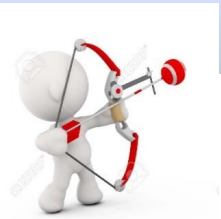
Nota: Cada SGBD fue proponiendo su solución procedural y SQL-99 incorporó algunas características, pero no todos se ajustan al estándar

→ **Chequear sintaxis y características del SGBD usado (ej. PosgreSQL)!**

Triggers - utilidad

Los triggers pueden usarse para:

- Forzar reglas de integridad o del negocio complejas (Ej. cuando no es posible incluirlas declarativamente por limitaciones del SGBD) o con acciones específicas de reparación (diferentes al rechazo y la reparación estándar)
- Mantener datos derivados - Generación automática de datos
- Propagar actualizaciones (sobre la misma tabla u otras tablas)
- Generar logs para soporte de auditoría o estadística de las acciones sobre la base de datos y chequeos de seguridad
- Mantener vistas actualizadas (cuando el SGBD no provee capacidades para hacerlo)



Recordar que un trigger se ejecuta automáticamente "disparado" por el evento sobre el cual está definido

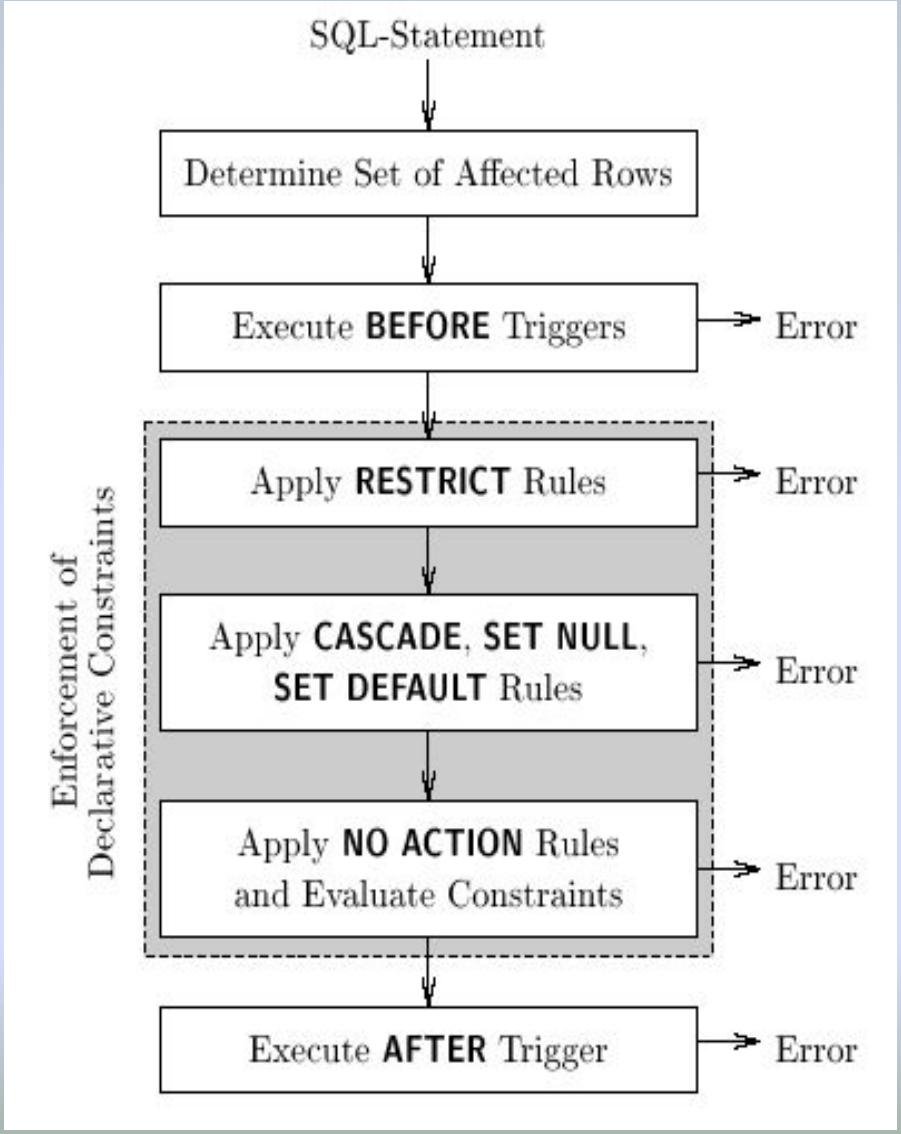
Triggers vs. RI Declarativas

- Triggers → permiten *definir y forzar* reglas de integridad, pero *NO son una restricción de integridad*
- Los triggers deberían usarse sólo cuando NO es posible expresar RI mediante una cláusula declarativa, y entonces se deberá:
  - controlar la restricción con triggers en una o en varias tablas
  - analizar en cada caso los eventos críticos, granularidad, tiempo de activación y sus correspondientes acciones

Tener en cuenta que:

- un trigger definido para forzar una RI NO verifica su cumplimiento para los datos ya almacenados en la base de datos (al introducir una RI declarativa sí se verifica la carga existente en la BD)
- para una RI especificada declarativamente se fuerza su cumplimiento ante cualquier tipo de operación en la BD; en caso de usar triggers, deben declararse e implementarse para todos los eventos críticos
- los triggers permiten implementar controles y otras acciones sobre la BD con mayor flexibilidad y posibilidades que a través de RI

Modelo de Ejecución según SQL-99



Ante una sentencia SQL de modificación de tabla, el SGBD determina las filas afectadas y:

1. Ejecuta los triggers **BEFORE-statement**
2. Por cada fila afectada por la sentencia SQL
   a. Ejecuta los triggers **BEFORE-row**
   b. Ejecuta las RI declarativas
      - *FK modo RESTRICT*
      - *FK modo CASCADE, SET NULL, SET DEFAULT*
      - *RI no nulidad, PK/ UNIQUE , FK modo NO ACTION y demás chequeos*
   c. Ejecuta los triggers **AFTER-row**
3. Ejecuta los triggers **AFTER-statement**

Antes y después de la ejecución de la sentencia, la BD está en **estado consistente**

Nota**:** No todos los SGBD siguen este esquema

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

BIBLIOGRAFÍA

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

Elmasri, R., Navathe, S., "Fundamentals of Database Systems", Addison Wesley, 2011 (y 2016)

Silberschatz, A., Korth, H, Sudarshan, S., "Database System Concepts", McGraw Hill, 2002

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