FACULTAD DE CIENCIAS EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas (TUDAI)

Base de Datos

Tema 5: Restricciones de Integridad - Parte 1

2025

Restricciones de Integridad (RI)
Concepto

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

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 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 actualizaciones en los datos que no cumplan las RI

→ rechazando la operación (insert, delete, update)

→ realizando acciones reparadoras extras

ambas respuestas deben dejar la BD en un estado consistente

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 UK_nom] **UNIQUE** (lista_col),] }
…);

Recomendable!

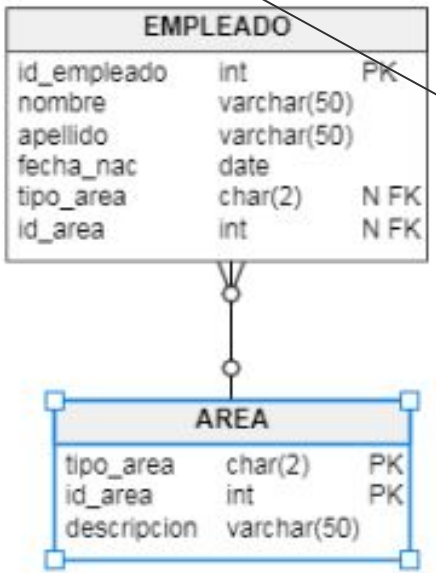
o: **ALTER TABLE** NombreTabla
ADD [CONSTRAINT PK_nom] **PRIMARY KEY** (lista_col_PK);

ALTER TABLE NombreTabla
ADD [CONSTRAINT UQ_nom] **UNIQUE** (lista_col_UQ);

Restricciones de Integridad Referencial (RIR)

Una clave extranjera (FOREIGN KEY en SQL) de una tabla *A* (***referenciante***) es un conjunto de columnas cuyos valores coinciden con los valores de otro conjunto de columnas, que son clave de otra tabla B (***referenciada***)

Nota: *A* y *B* podrían ser la misma tabla.




```
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) ;
```

Restricciones de Integridad Referencial (RIR)

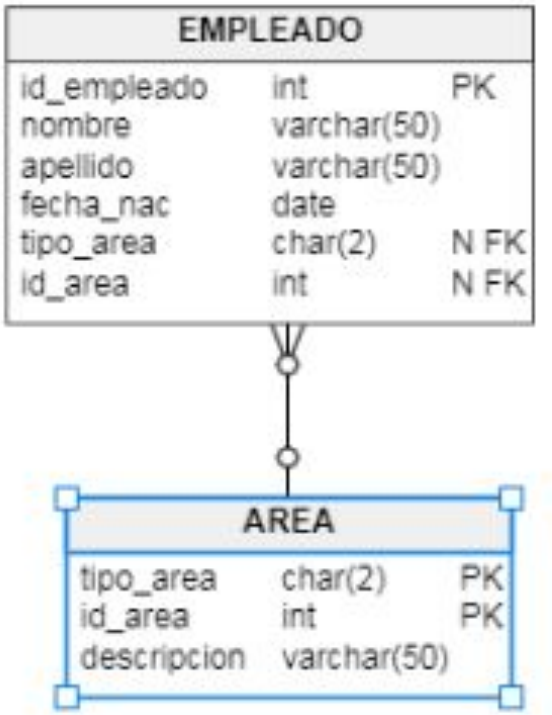
Las claves extranjeras (FOREIGN KEY) especifican relaciones entre tablas y permiten mantener la consistencia entre registros de esas tablas

El conjunto de valores de la clave extranjera de una tabla A debe coincidir al menos con un valor de la clave primaria de la tabla *B*, a la que hace referencia, o bien ser nulo

RIR: Acciones Referenciales

¿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?




```
ALTER TABLE EMPLEADO ADD CONSTRAINT
FK_EMPLEADO_AREA
    FOREIGN KEY (tipo_area, id_area)
    REFERENCES AREA (tipo_area, id_area)
    ON DELETE acción referencial para borrado
    ON UPDATE acción referencial para modificación;
```

RIR: Acciones Referenciales

Qué sucede si se intenta borrar (delete) un registro en la tabla AREA que está siendo referenciado en la 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

RIR: Acciones Referenciales

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

→ **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)*

RIR: Acciones Referenciales

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 modificar 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

RIR: Acciones Referenciales

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 2 - Acepta la operación y realiza acciones reparadoras adicionales:** modifica la clave primaria del registro en la tabla AREA y

- ✓ **CASCADE:** propaga la modificación a todos los registros que referencian a dicha clave primaria mediante la FK en *la tabla EMPLEADO*
- ✓ **SET NULL:** coloca nulos en la FK de los registros que referencian a dicha clave primaria en *la tabla EMPLEADO* (sólo si admite nulos)
- ✓ **SET DEFAULT:** coloca el valor por defecto en la FK de los registros que referencian a dicha clave primaria en *la tabla EMPLEADO (si es posible)*

(RIR) Definición

En SQL: **CREATE TABLE** NombreTabla (…...

{ [[CONSTRAINT FK_nom] **FOREIGN KEY** (lista_columnasFK)

REFERENCES nombreTablaRef [(lista_columnasRef)]

[**MATCH** {FULL | PARTIAL | SIMPLE}]

[**ON UPDATE** AccionRef]

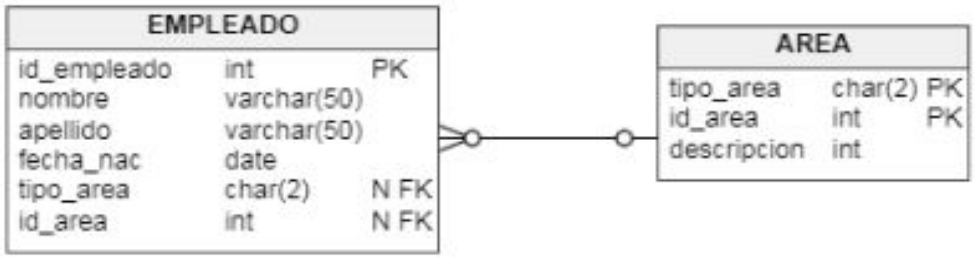
[**ON DELETE** AccionRef]] } …..);

o: **ALTER TABLE** NombreTabla

ADD CONSTRAINT FK_nom **FOREIGN KEY** (lista_columnasFK) …;

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

RIR: Acciones Referenciales



Ejemplo

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 tipo_area= 'A' AND id_area= 1;**
- **DELETE FROM Area WHERE tipo_area= 'B' AND id_area = 2;**
- **DELETE FROM Area;**
- **UPDATE Area set id_area= 3 where tipo_area= 'B' AND id_area= 1;**
- **UPDATE Area set id_area= 3 where tipo_area= 'B' AND id_area= 2;**

id_empleado	nombre	apellido	fecha_nac	tipo_area	id_area
2	José	Mares	1990-03-06	A	1
3	Ana	Castro	1980-08-01	B	1
4	Ximena	Lopez	1985-08-07	A	2
6	Iris	Dominc	1978-05-07	B	1

tipo_area	id_area	descripcion
A	1	AREA A1
A	2	AREA A2
B	1	AREA B1
B	2	AREA B2

RIR – 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**

RIR – Tipos de Matching

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

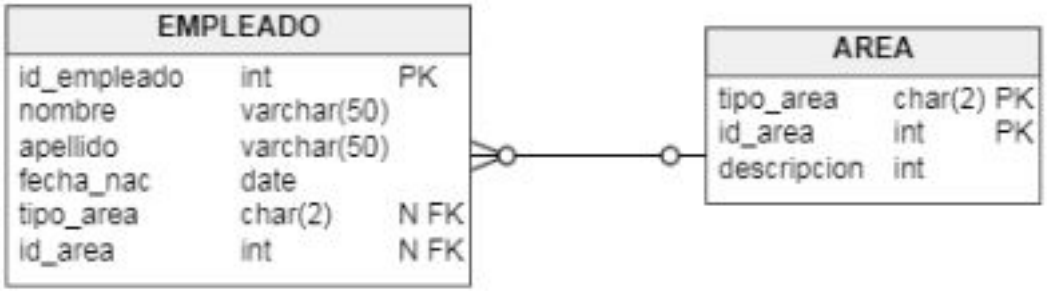
1- Ninguno de los valores de las columnas de la FK es NULL y existe un registro en la tabla referenciada cuyos valores de clave coinciden con los de tales columnas,

2- o

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

RIR – Tipos de Matching

Ejemplo



Analizar la posibilidad de alta de las sig. tuplas en EMPLEADO según los distintos tipos de matching

tipo_area	id_area	descripcion
A	1	AREA A1
A	2	AREA A
B	1	AREA B
B	2	AREA B

id_empleado	nombre	apellido	fecha_nac	tipo_area	id_area
2	José	Mares	1990-03-06	A	1
1	Juan	Sans	2000-08-08	C	*NULL*
3	Ana	Castro	1980-08-01	B	*NULL*
4	Ximena	Lopez	1985-08-07	*NULL*	*NULL*
6	Iris	Dominc	1978-05-07	*NULL*	1

MATCHING

Simple	Parcial	Full
ok	ok	ok
ok	X	X
ok	ok	X
ok	ok	ok
ok	ok	X

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 o de atributo** (DOMAIN o CHECK de atributo)
- **RI de tabla asociada a uno o más atributos** (CHECK de registro)
- **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

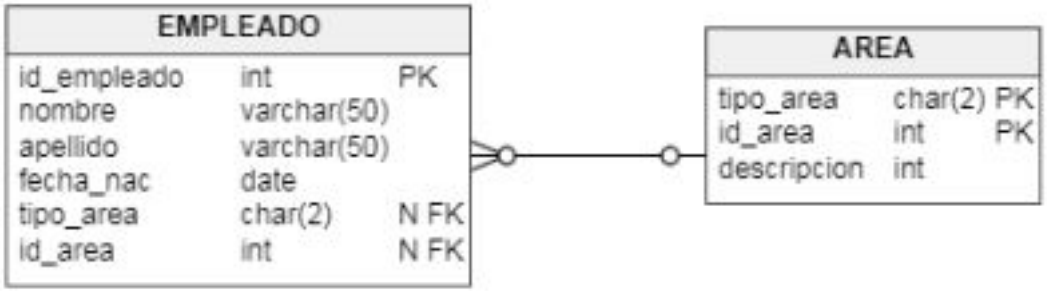
RI de Dominio/Atributo

- 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

La condición debe evaluar como VERDADERA o DESCONOCIDA

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

RI de Dominio/Atributo



Ejemplo

Los tipos de áreas son cadenas de 2 caracteres que pueden tomar los valores AA, AB, BB, BC

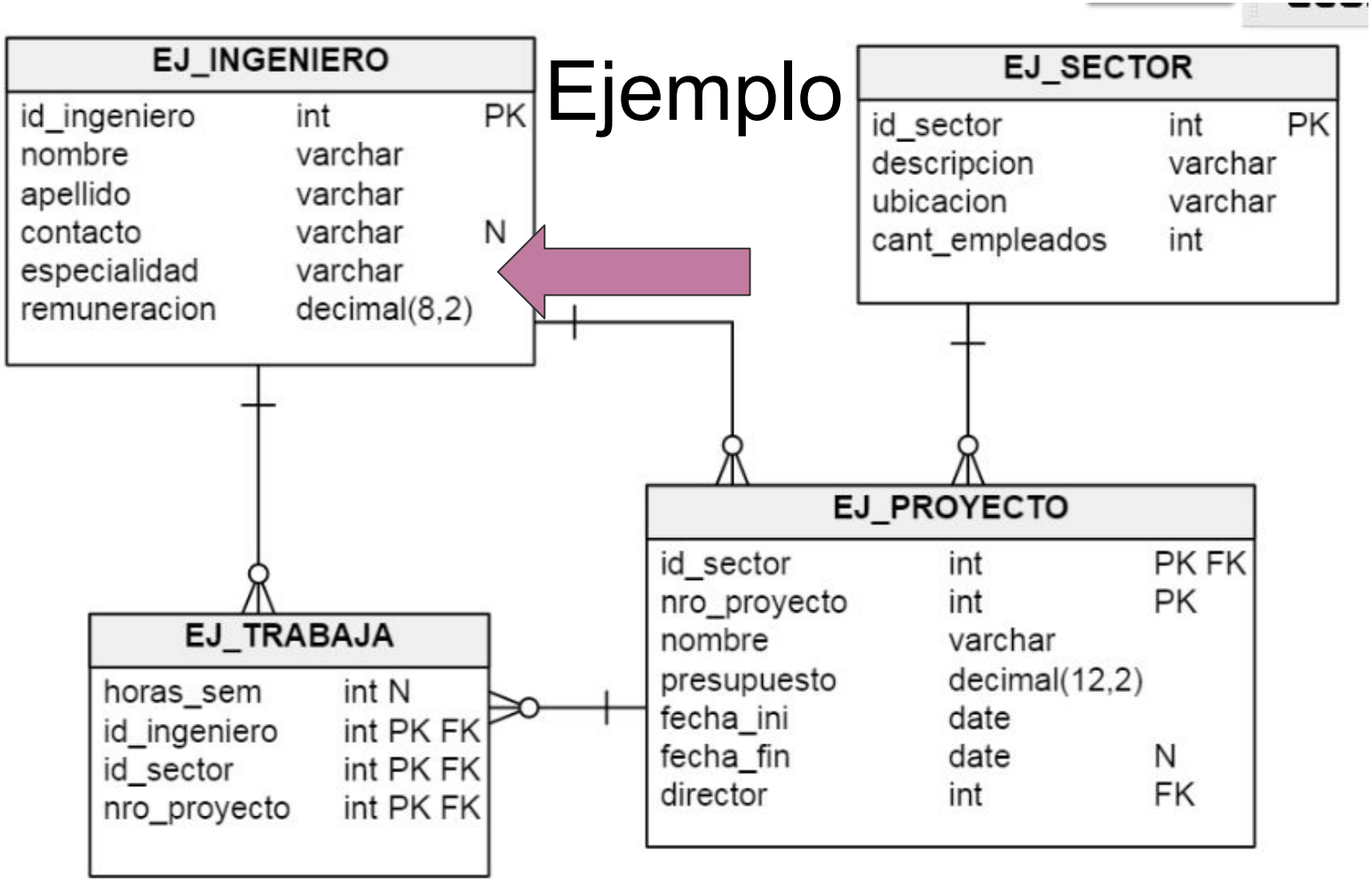
```
CREATE DOMAIN area_valida
  AS CHAR(2) NOT NULL
  CHECK (value IN( 'AA','AB','BB','BC'));

CREATE TABLE Area
    ( …. ,
      tipo_area  area_valida
     );

o   CREATE TABLE Area
    ( …. ,
      tipo_area CHAR(2) NOT NULL
       CHECK (tipo_area  IN( 'AA','AB','BB','BC')) );
```

```
ALTER TABLE Area
ADD CONSTRAINT ck_empleado_tipo_area
CHECK (tipo_area IN ( 'AA','AB','BB','BC'));
```

Ejemplo



El siguiente script crea las tablas del diagrama BD_05_RestInteg_P1.sql y debemos controlar que:

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

Ejemplo

```
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 DE DOMINIO /ATRIBUTO

- **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

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_'

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

(…..

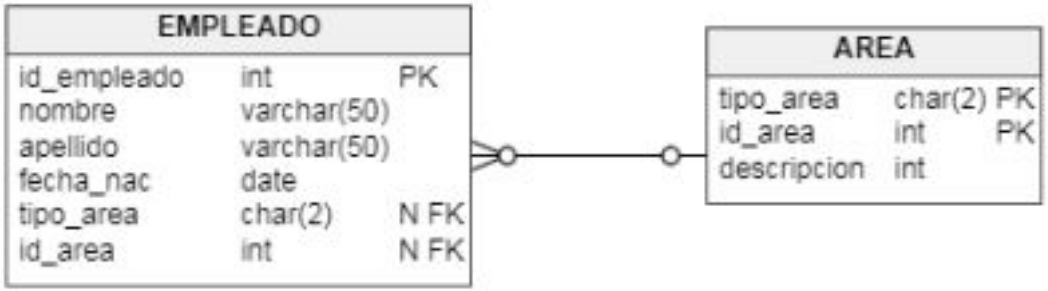
{ [[CONSTRAINT nom_restr] CHECK (condición)] });

La condición debe evaluar como VERDADERA o DESCONOCIDA

o: **ALTER TABLE** NombreTabla

ADD [CONSTRAINT nom_restr] CHECK (condición) ;

RI (CHECK) de Tupla

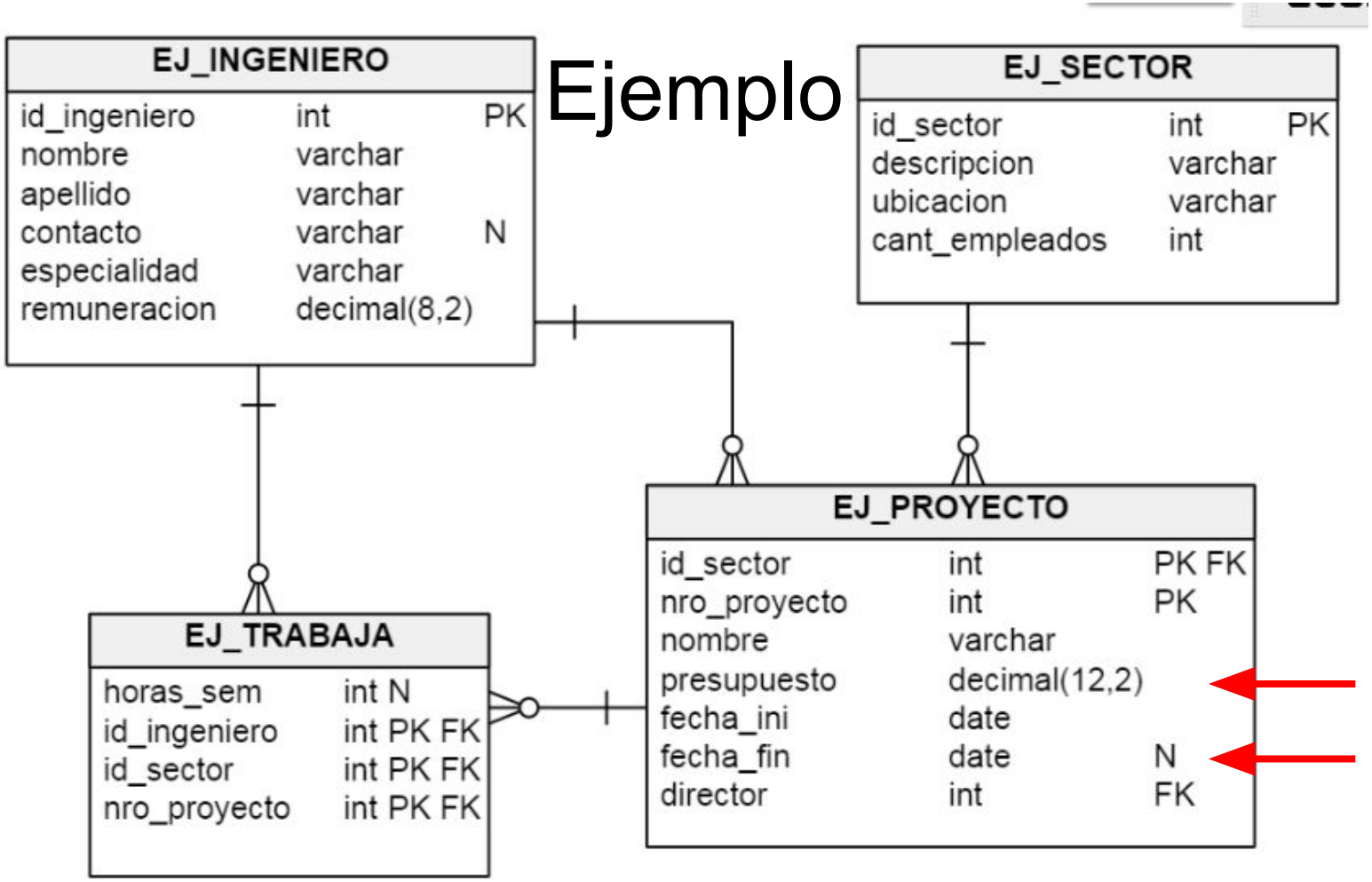


Ejemplo

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');
```

Ejemplo



Ahora debemos controlar que:

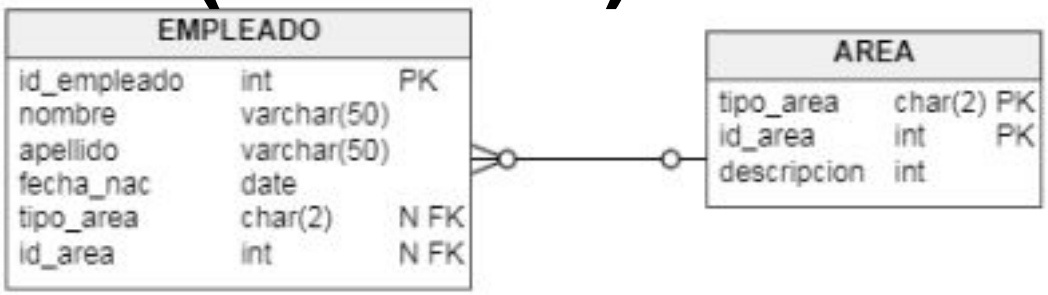
Los proyectos sin fecha de finalización asignada no deben superar $100.000 de presupuesto

Ejemplo

EJ_PROYECTO				
id_sector	nro_proyecto	presupuesto	fecha_fin	.....
1	1	100.000 $	N	
1	2	110.000 $	N	
1	3	1.500.000 $	1/1/2021	
1	4	900.000 $	2/1/2021	

```
ALTER TABLE ej_proyecto
ADD CONSTRAINT ck_ej_proyecto_presupuesto
   CHECK ((fecha_fin IS NULL AND presupuesto <= 100000000)
         OR   (fecha_fin IS NOT NULL) );
```

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)

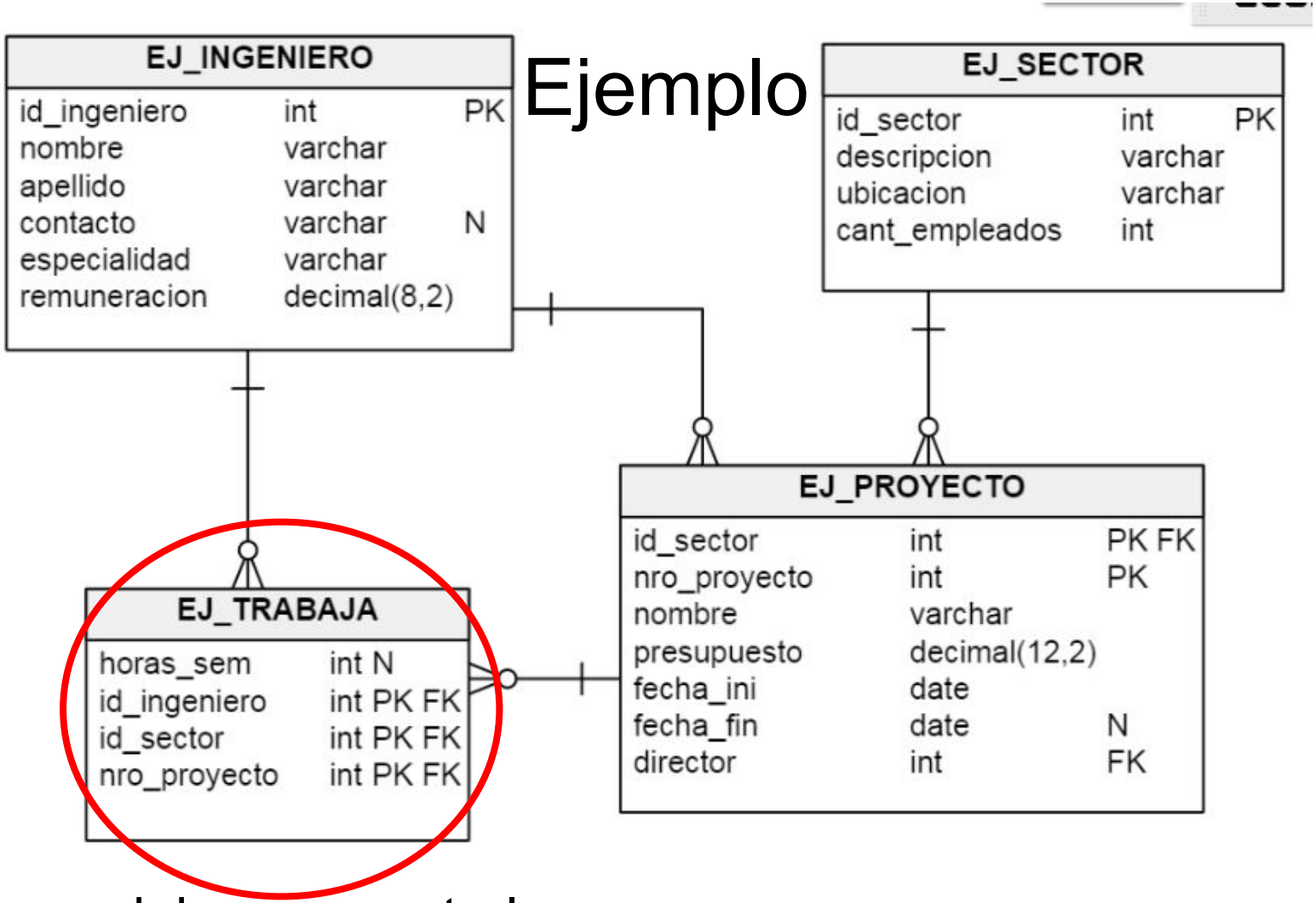
Ejemplo

No puede haber más de 30 empleados por área

```
ALTER TABLE Empleado
 ADD CONSTRAINT ck_area_max
  CHECK ( NOT EXISTS (SELECT 1
                      FROM Empleado
                      GROUP BY  tipo_area,  id_area
                      HAVING count(*) > 30));
```

La gran mayoría de los DBMS comerciales no las soportan

Ejemplo



Ahora debemos controlar que:

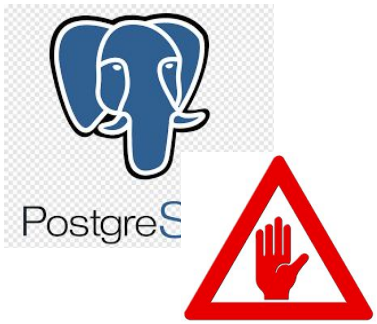
En cada proyecto pueden trabajar 10 ingenieros como máximo

Ejemplo

La pregunta que me tengo que hacer…. Cómo seleccionar los identificadores de los proyectos en los que trabajan más de 10 ingenieros?

```
SELECT id_sector, nro_proyecto
FROM EJ_TRABAJA
GROUP BY id_sector, nro_proyecto
HAVING COUNT(*) > 10));
```


```
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));
```



Error de SQL:

```
ERROR:  cannot use subquery in check constraint
```

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 mantenimiento

→ los DBMS comerciales no soportan ASSERTIONS !

RI Globales (ASSERTIONS)

Ejemplo

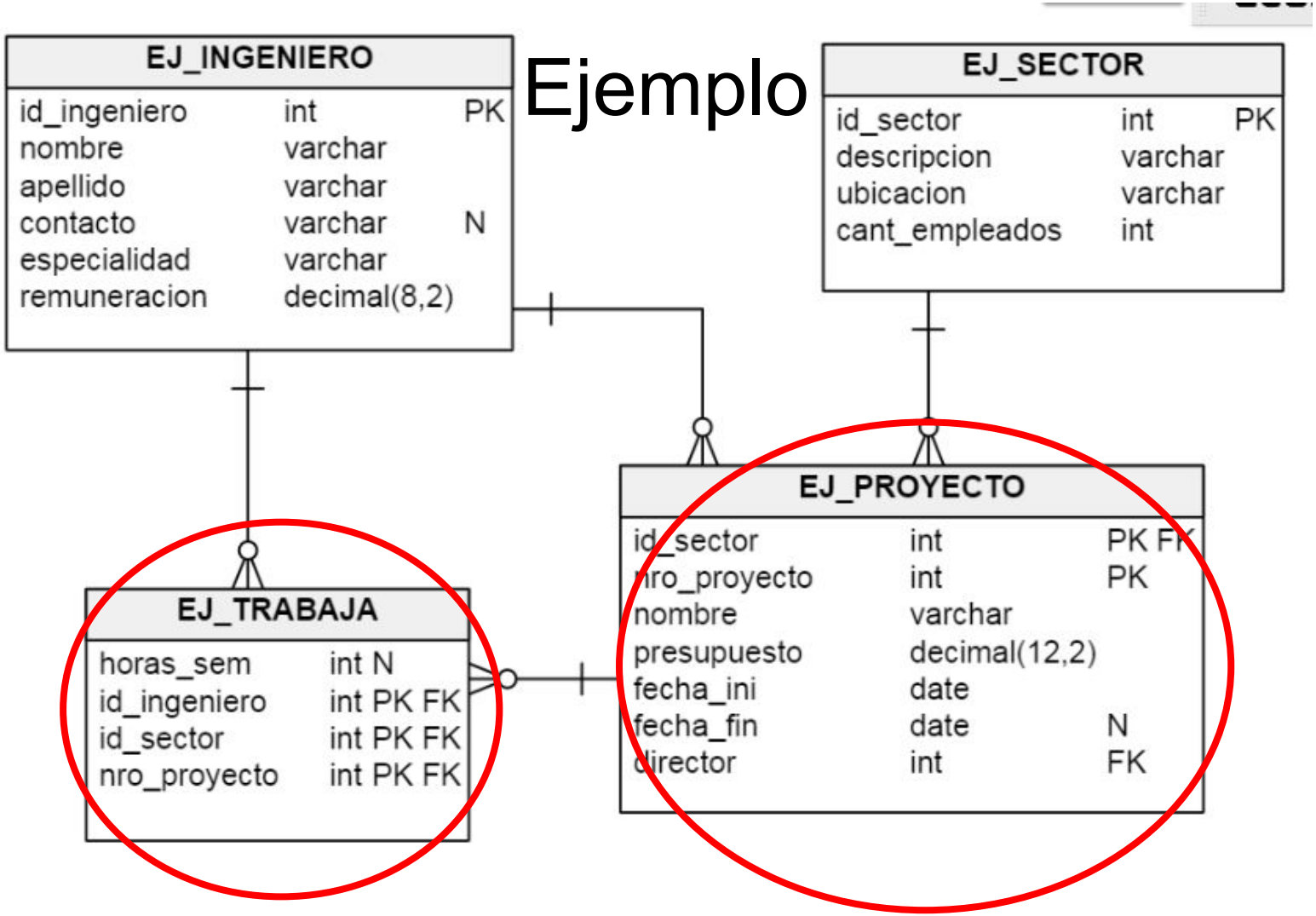
El sueldo de los empleados de un área no puede ser mayor al sueldo del gerente de esa área

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)»

Ejemplo



Ahora debemos controlar que:

El director asignado a un proyecto debe haber trabajado al menos en 5 proyectos ya finalizados

Ejemplo

Nuevamente la pregunta que me tengo que hacer es…. Cómo busco lo que está mal.. cómo selecciono los proyectos que tienen un director que trabajan en menos de 5 proyectos?

```
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_proyecto)
WHERE PP.fecha_fin IS NOT NULL
GROUP BY P.director
HAVING COUNT(*) < 5;
```

Ejemplo

Nuevamente la pregunta que me tengo que hacer es…. Cómo busco lo que está mal.. cómo selecciono los proyectos que tienen un director que trabajan en menos de 5 proyectos?

```
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_proyecto)
            WHERE PP.fecha_fin IS NOT NULL
            GROUP BY P.director
            HAVING COUNT(*) < 5));
```

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. *Declarativa o Procedural*

Por los estados involucrados:

- **RI de estado -** restringe los valores que pueden tomar los datos en un momento
- **RI de transición de estados-** restringe los posibles cambios de valores entre estados sucesivos de los datos

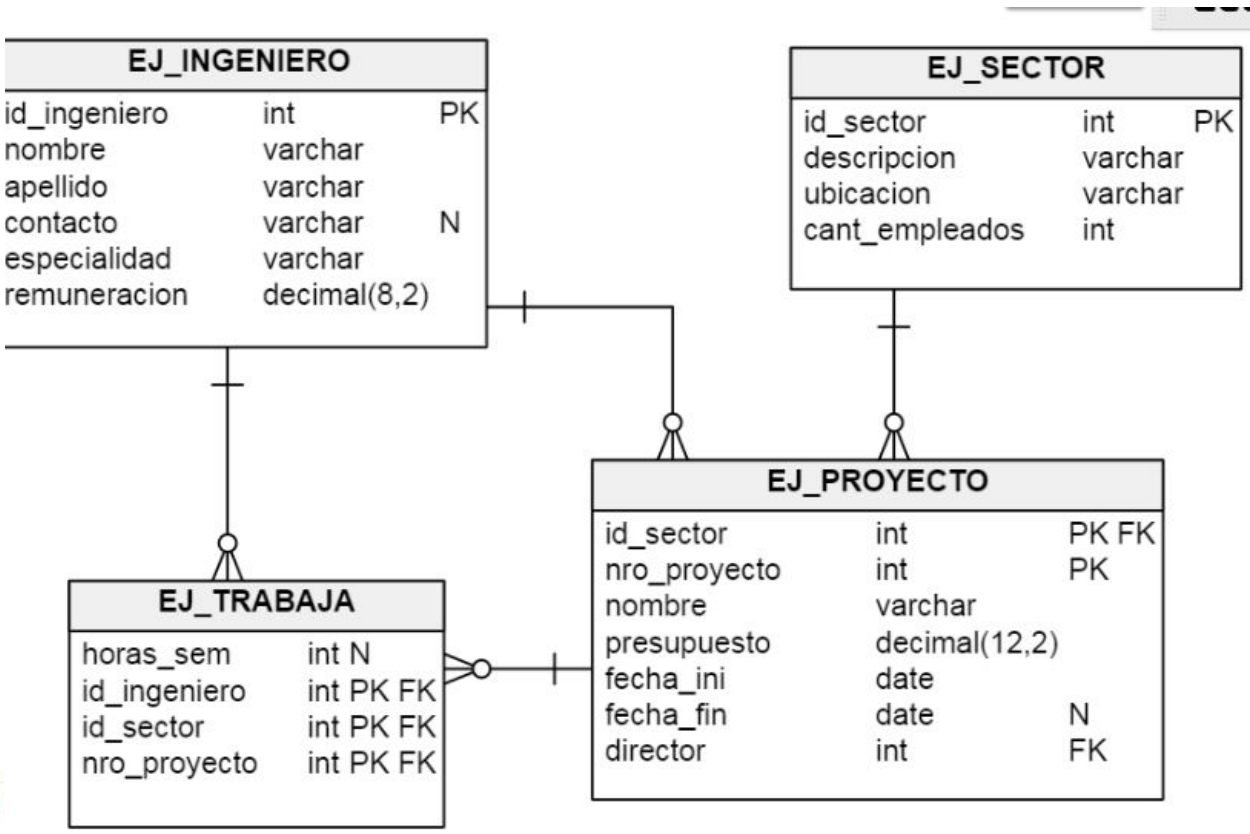
Resumen del Ejemplo

1- Las especialidades de los ingenieros pueden ser "inteligencia empresarial" , "tecnologías móviles" , "gestión de TI" o "desarrollo"

2- Los proyectos sin fecha de finalización asignada no deben superar $100000 de presupuesto

3- En cada proyecto pueden trabajar 10 ingenieros como máximo

4- El director asignado a un proyecto debe haber trabajado al menos en 5 proyectos ya finalizados en el mismo sector



Resumen del Ejemplo

1- Las especialidades de los ingenieros pueden ser "inteligencia empresarial" , "tecnologías móviles" , "gestión de TI" o "desarrollo"

Ámbito: Atributo (columna)

Tipo RI: de dominio o atributo

CREATE DOMAIN especialidad AS varchar(20)

CHECK(VALUE IN (‘inteligencia empresarial’ , ‘tecnologías móviles’ , ‘gestión de TI’, ‘desarrollo’));

Resumen del Ejemplo

2- Los proyectos sin fecha de finalización asignada no deben superar $100000 de presupuesto

Ámbito: tupla

Tipo RI: de Tupla

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

Resumen del Ejemplo

3- En cada proyecto pueden trabajar 10 ingenieros como máximo

Ámbito: Tabla

Tipo RI: de Tabla

```
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));
```

Resumen del Ejemplo

4- El director asignado a un proyecto debe haber trabajado al menos en 5 proyectos.

Ámbito: más de una tabla

Tipo RI: Global

```
CREATE ASSERTION CK_PROY_DIRE
CHECK ( NOT EXIST
            (   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_proyecto)
                WHERE PP.fecha_fin IS NOT NULL
                GROUP BY P.director
                HAVING COUNT(*) < 5));
```

BIBLIOGRAFÍA

Capitulo 36 del Manual de PostgreSQL . www.postgresql.org

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