INTEGRIDAD DE DATOS

- Consistencia o Corrección de datos en la Base de Datos
- Reglas de Integridad NO son específicas para cada usuario
  (Las Reglas de Seguridad SÍ)

CONSIDERACIONES GENERALES

1. Interesan las **Reglas Integridad** Específicas de una BD (reglas del negocio), además de RI de Entidad, RI Referencial, etc.

2. Veremos las RI definidas sobre **Relaciones Base**, pues están restringidas a contener datos correctos (reflejar la realidad)

La **regla** "Los títulos de las películas son únicos" se aplica a la **tabla base** PELICULA, y a cualquier **VISTA** definida sobre ésta

¿Podemos definir RI sobre Relaciones Derivadas (sobre VISTAS)?

- Sería deseable
- La relación derivada heredaría toda RI de sus Relaciones Base y podría añadir nuevas (clave candidata nueva para la vista, p. ej.)

☆ **Sólo** consideraremos **RI sobre Relaciones Base** (por simplicidad)

INTEGRIDAD: Consideraciones generales

3. Interesa **soporte de Reglas de Integridad Declarativo**
No hablaremos de...
- Procedimientos almacenados (stored procedures), ni de
- Procedimientos disparados (triggered procedures)

4. Una BD en un estado de integridad...
- Es una BD correcta, es decir,
- No viola ninguna Regla de Integridad conocida por el sistema, o sea,
- Satisface AND lógico de TODAS las RI definidas

5. Integridad, interesante en las **etapas de...**
- DISEÑO
  - Estructuras de Datos convenientes
  - Reglas de Integridad adecuadas
- EJECUCIÓN (corrección de la información)

6. RI mantenidas **en** el **Catálogo del Sistema**
SUBSISTEMA DE INTEGRIDAD **del DBMS**:
- **control de operaciones** de **usuario** (INSERT, UPDATE, DELETE...)
- para asegurar que **NO violan las Reglas de Integridad**

REGLAS DE INTEGRIDAD: Componentes

❍ **Nombre** actor_cache_ok
- Regla almacenada en el *Catálogo del Sistema* bajo ese nombre
- Aparecerá en *diagnósticos,* producidos por el sistema como respuesta a intentos de violación de la regla (mensajes de error al usuario)

❍ **Restricción** NOT EXISTS (SELECT * FROM ACTOR WHERE cache≤0)
- Expresión booleana NOTA: Restricción de Integridad ⊆ Regla de Integridad
- La Regla... *se satisface* ⇔ la restricción es TRUE
  es violada ⇔ la restricción es FALSE

❍ **Respuesta a un intento de violación de la regla**
- Indica al sistema qué hacer si un usuario intenta una operación que viola la RI
- Por defecto **REJECT** (rechazar o rehusar) que implica...
  - Deshacer los posibles daños causados por la operación
  - Mostrar información de diagnóstico (mensaje)
- Podría ser un **procedimiento** de complejidad arbitraria: ... **tratarError(...)**

REGLAS DE INTEGRIDAD: Creación, Destrucción y Tipos

• **CREACIÓN** de una Regla de Integridad... (en cualquier momento)
  SGBD comprueba: ¿el **estado actual** de la **BD satisface** la **RI**?
  - **Sí** ⇒ aceptada !
    - el sistema la **almacena en** el **catálogo**
    - la regla es **activada** (entra en vigor)
      DBMS controlará todo INSERT y UPDATE de "cache", en ACTOR
  - **No** ⇒ nueva RI rechazada

• **DESTRUCCIÓN** de Reglas de Integridad

• Las Reglas de Integridad pueden **restringir los valores legales** de...
  - **Dominio**
  - **Atributo**
  - **Relación**
  - **Base de Datos**

INTEGRIDAD en SQL2

TRES CATEGORÍAS de reglas de integridad:

1. **Restricciones de Integridad** (RI) **de Dominio**
   - Asociadas a un dominio de datos específico
   - Aplicadas a cada columna definida sobre el dominio
   - No limitadas a la enumeración de valores posibles del dominio
   - » la RI es una **expresión** cualquiera (complejidad arbitraria) **que define un dominio**

2. **Restricciones de Integridad de Tabla Base**
   - Son RI de complejidad arbitraria incluidas en la definición de una tabla
   - Pueden ser **Restricciones** de **Atributo**
     Restricciones de **Relación** y
     Restricciones de **Base de Datos**
     ≡ abreviaturas de RI de Tabla más comunes
   - No limitadas a hacer referencia sólo a la tabla base en la que se definen

   * Una **relación vacía** cumple cualquier RI de Tabla, aunque esa RI sea "esta tabla no puede estar vacía"

INTEGRIDAD en SQL2

3. **Restricciones de Integridad Generales**
   - Son RI de complejidad arbitraria no incluidas en la definición de ninguna tabla, sino que son otro elemento más de la base de datos (al mismo nivel que una tabla, o vista)

La paga mínima de los actores que actúan en una película es de 15.000$

```
CREATE ASSERTION IR1
CHECK (SELECT MIN(paga) FROM ACTUA_EN) ≥15000 ;
```

Todo actor debe haber participado al menos en una película

```
CREATE ASSERTION IR2
CHECK (EXISTS(SELECT * FROM ACTOR AC, ACTUA_EN AE
        WHERE AC.codA = AE.actor)) ;
```

   - Algunas RI Generales pueden expresarse como RI de tabla base

INTEGRIDAD en SQL2: consideraciones

- Una **RI** es **independiente** de la aplicación informática específica que acceda a la BD
  » no contiene *parámetros* ni *variables host* (referencias a variables del programa de aplicación)

- Es útil ver la **BD sujeta a** una "**RI** *gigante*": **AND** de todas las RI...
  - *Generales*
  - de Tabla
  - de Dominio, aplicadas a cada columna
  - *Significado formal* de la base de datos

· **CARACTERÍSTICAS ADICIONALES** (*pseudo-Restricciones de Integridad*)

1. SQL rechaza todo intento de INSERT o UPDATE que viole una especificación de **tipo de datos** (ej: valor CHAR en columna definida como INTEGER)

nota: Violación de Tipo de Datos es un error sintáctico y se detecta en tiempo de compilación
Violación de RI-dominio se detecta en tiempo de ejecución (antes imposible)

2. SQL rechaza todo intento de INSERT o UPDATE sobre una **vista**, si se viola la **condición de definición de la vista** (si se especificó la opción CHECK)

SQL2: Reglas de Integridad de DOMINIO

· **DEFINICIÓN del conjunto de valores componentes de un dominio**:
- **Enumeración** de **valores** posibles: ('marron','gris','azul','verde','negro','no-se')
- **Expresión** de **definición**: cache > 0 AND cache < 100

· **CREACIÓN DE UN DOMINIO**

CREATE DOMAIN <nomDominio> **[AS]** <tipoDatos>
[DEFAULT <valor> **]**
[[CONSTRAINT <nomRestricción> **] CHECK (**<condición>**)**]+ **;**

donde valor = { literal | funciónSinArgum | NULL }
y funciónSinArgum es

USER ó CURRENT_USER	*authorization-ID* actual
SESSION_USER	*authorization-ID* de la sesión SQL actual
SYSTEM_USER	ID del usuario del SO actual
CURRENT_DATE	fecha del día
CURRENT_TIME	hora actual
CURRENT_TIMESTAMP	marca de tiempo actual

ID=identificador

SQL2: Reglas de Integridad de DOMINIO

- La RI es especificada como **parte** de la **sentencia de definición del dominio:**

  CREATE DOMAIN Color **AS VARCHAR(**10**)**
  DEFAULT 'no-se'
  → **CONSTRAINT** color_valido
  CHECK (**VALUE IN** ('marron','gris','azul','verde','negro','no-se'));

  CREATE DOMAIN SEXO CHAR(1)
  CONSTRAINT valor_sex_ok CHECK (VALUE IN ('M', 'F')) ;

  - CREATE DOMAIN SEXO AS CHAR(1) NOT NULL ---------- ¡¡ INCORRECTO !!
  - CREATE DOMAIN SEXO AS CHAR(1) CHECK (VALUE IS NOT NULL) -- ¡¡ CORRECTO !!

- En la ejecución, los dominios no cambian, sino los atributos (sus valores)
  » una RI-Dominio 'per se' nunca se chequea, sino la RI-Atributo correspondiente

- **ALTERACIÓN DE UN DOMINIO**

  ALTER DOMAIN <nomdominio> ... ;
  Permite Añadir/Eliminar Restricciones de Dominio y Valor por Defecto

SQL2: Reglas de Integridad de DOMINIO

- **ELIMINACIÓN DE UN DOMINIO** existente

  DROP DOMAIN <nomDominio> { **RESTRICT** | **CASCADE** } ;

  RESTRICT
  - La eliminación falla si el **dominio** es **referenciado** en cualquier
    - ✓ definición de **columna** o
    - ✓ definición de **vista** o
    - ✓ **Restricción de Integridad**
  - **En otro caso**, tendrá éxito: el descriptor del **dominio** es **eliminado** del catálogo.

  CASCADE
  - Toda **RI** y **vista** cuya definición referencie al dominio es **eliminada** con él
  - Cada **columna** definida sobre tal dominio **NO es eliminada**, sino **alterada**:
    - **Definida** directamente **sobre** el **Tipo de Datos subyacente** al dominio
    - Si no tiene **DEFAULT** explícito, toma el del dominio (si éste lo tenía)
    - **Hereda** toda **Restricción de Integridad** asociada al **dominio,**
      - **convertida en una Restricción de Tabla Base**
      - **sustituyendo "**VALUE**" por el nombre de la columna**

SQL2: Reglas de Integridad de TABLA BASE

- **RESTRICCIÓN ASOCIADA A** UNA **TABLA BASE específica**
  - Esto no significa que no **pueda afectar a muchas tablas** base, sino que...
    - la RI está definida como parte de la definición de la tabla,
    - la RI no existe si la tabla asociada no existe y
    - eliminar la tabla implica eliminar la RI
- Toda **RI-tabla** puede expresarse como una **RI-general**
  (salvo la parte de una RI de clave externa que indica la acción referencial)
- **RI ESPECIFICADA** DENTRO DE **CREATE TABLE**...

  CREATE TABLE <nombreTabla> (<lista de **elementos** de tabla base>) ;

  donde los elementos pueden ser:
  - Definición de **Columna**
  - Definición de **Restricción Tabla** (precedida o no por CONSTRAINT <nomRestric>)
    - Restricción de **Clave Candidata**
    - Restricción de **Clave Externa**
    - Restricción **CHECK**
- **RI AÑADIDA / ELIMINADA** DENTRO DE **ALTER TABLE** <nombreTabla>...

SQL2: Reglas de Integridad de TABLA BASE

1. Definición de Columna (RI de Atributo)

- **Especificación del dominio** del **atributo** y otras **RI** de atributo
- No necesaria sentencia de creación explícita: forma **parte** de la **definición de atributo dentro de la sentencia de creación de la tabla:**

```
CREATE TABLE ACTOR
( nombre     VARCHAR(30)  NOT NULL,
  cache      INT(9)       NOT NULL  DEFAULT 20000,
  colorOjos  Color        NOT NULL,
  agencia    CHAR(4), ...) ;
```

- Toda RI-Atributo es **comprobada inmediatamente**:
  todo intento de INSERT o UPDATE (sobre el atributo) de valor ∉ dominio, es rechazado al instante
  - » La respuesta a un intento de violación siempre es REJECT (rechazo)
- **La comprobación de la restricción** es **derivada hacia** la **comprobación de la restricción del dominio** del atributo (si existe)
- Una RI-Atributo se destruye al eliminar el atributo de la relación base

SQL2: Reglas de Integridad de TABLA BASE

2. Definición de Clave Candidata (CK)

- Clave Primaria (PK) **PRIMARY KEY (**<listaAtributos>**)** --incluye RI Entidad
- Clave Alternativa (AK) **UNIQUE (**<listaAtributos>**)**

3. Definición de Clave Externa (FK)

- **Interrelación** entre dos **relaciones distintas** de la Base de Datos

FOREIGN KEY (<ListaAtributos>**) REFERENCES** <nomTabla> **(**<listaAtributos>**)**
[**ON DELETE** { **NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL** }]
[**ON UPDATE** { **NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL** }]
↑ acciones referenciales posibles

- **Cualquier columna** (o combinación de columnas) **puede ser una clave externa**
- SQL2 permite que **FK referencie** a **cualquier clave candidata** (no sólo PKs)
- Representa la **referencia** desde fila de T2 a una CK de otra fila en otra tabla T1
  »El problema de asegurar que cada T2.FK tiene un valor de T1.CK existente, es el **problema de la integridad referencial**

SQL2: Reglas de Integridad de TABLA BASE

- Pueden existir **auto-referencias** y **ciclos referenciales**
- SQL2 permite (¡por supuesto!) que una **FK pueda contener el valor NULL**
  – salvo si se indica NOT NULL explícitamente para la FK (en el CREATE TABLE)
- La **FK y** la **CK** a la que referencia, deben...
  – contener el **mismo nº** de **componentes** (columnas o atributos) y
  – éstos estar definidos sobre los **mismos dominios**
- Si **FK no es compuesta** (único atributo), **no** es **necesaria** cláusula **FOREIGN KEY**...

```
CREATE TABLE ACTUA_EN
( actor ... ,
  film ... ,
  papel ... ,
  PRIMARY KEY (actor, film),
  FOREIGN KEY (actor)
          REFERENCES ACTOR(codA) ...,
  FOREIGN KEY (film) REFERENCES PELICULA(codP)
... ) ;
```

```
CREATE TABLE ACTUA_EN
( actor ... REFERENCES ACTOR(codA) ...,
  film ...  REFERENCES PELICULA(codP) ...,
  papel ... ,
  PRIMARY KEY (actor, film)
  ... ) ;
```

✋ NO comprobada inmediatamente: **diferida** hasta fin de transacción (COMMIT)

SQL2: Reglas de Integridad de TABLA BASE

ACCIONES REFERENCIALES (recordatorio)

- Mantenimiento de Integridad Referencial, sin rechazar actualizaciones que la violen
- Ejecución de operaciones adicionales que dejan la BD en estado consistente

ON DELETE...

- Indica la **Regla de Borrado** para filas de **T1** (tabla referenciada) respecto de **T2.FK**

  Qué ocurre si se intenta borrar una fila **r1** de **T1** (fila objetivo o target row) y existe alguna fila **r2** en **T2** que le hace referencia (e.d. contiene un valor **r2.FK = r1.CK**)

- *Acciones referenciales*

a. **NO ACTION** (opción por omisión) **Rechazo** de la operación de borrado sobre **T1**

b. **CASCADE** Elimina, junto con **r1**, toda fila **r2** de **T2** que referencie a **r1** (propagación)

c. **SET DEFAULT**

Cada componente de la FK se pone a su **valor por defecto** en todas las filas **r2**,y elimina **r1**

» Debe existir una fila en **T1** con cada componente de la **T1.CK** a su valor por defecto

d. **SET NULL**

Cada componente de la FK se pone a **valor nulo** en todas las filas **r2**, y elimina **r1**

» Cada componente de la FK debe tener **nulos permitidos**

SQL2: Reglas de Integridad de TABLA BASE

ON UPDATE...

- Indica la **Regla de Actualización** para **T1.CK** respecto de su clave externa **T2.FK**

  Qué ocurre si se intenta modificar la clave candidata dentro de una fila **r1** de T1 y existe alguna fila **r2** en T2 que le hace referencia (e.d. contiene un valor **r2.FK=r1.CK**)

- *Acciones referenciales*

a. **NO ACTION** (opción por omisión) **Rechazo** de la operación de actualización sobre **T1.CK**

b. **CASCADE** Propaga la actualización de la FK a todas las filas **r2** de T2 que referencian **r1**

c. **SET DEFAULT**

Los componentes de la FK que corresponden a componentes modificados de **T1.CK**, toman su **valor por defecto** en todas las filas **r2**

Y se actualiza **r1**

» Debe existir una fila en **T1** con cada componente de l**a T1.CK** a su valor por defecto

d. **SET NULL**

Los componentes de la FK correspondientes a componentes modificados en **T1.CK** se ponen a **nulo** en todas las filas **r2**

Y se actualiza **r1**

» Tales componentes de la FK debe tener **nulos permitidos**

SQL2: Reglas de Integridad de TABLA BASE

COMPORTAMIENTO ANTE OPERACIONES de MANIPULACIÓN de DATOS

Sea **T2.FK** una **clave externa** hacia **T1.CK** ...

* Si la **definición de FK NO incluye cláusula ON DELETE ni ON UPDATE...**

1. **INSERT** en T2 o **UPDATE** en T2.FK, tal que el valor de T2.FK **no existe** en T1.CK
   ⇒ Rechazo (NO ACTION)
2. **DELETE** en T1 o **UPDATE** en T1.CK, si **alguna tupla** de T2 le hace **referencia**
   ⇒ Rechazo (NO ACTION)

* Si la **definición de FK incluye cláusula ON DELETE o bien ON UPDATE...**

1. **INSERT** en T2 o **UPDATE** en T2.FK, tal que el valor de T2.FK **no existe** en T1.CK
   ⇒ Rechazo (NO ACTION)
2. **DELETE** en T1 o **UPDATE** en T1.CK, si **alguna tupla** de T2 le hace **referencia**
   ⇒ Depende de la acción referencial especificada en las cláusulas

SQL2: Reglas de Integridad de TABLA BASE

4. Definición de Restricción CHECK

- **Regla** que se **refiere únicamente** a **una relación**
- Puede especificar restricciones **adicionales** para algún atributo:

  El cache de un actor siempre está entre 50.000 y 200.000 ptas.

```
CREATE TABLE ACTOR ( ...,
  CONSTRAINT actor_cache_ok
        CHECK ( cache > 50000 AND cache < 200000 ),  ... );
```

- Puede especificar restricciones que involucran varios atributos de la relación

  La fecha de estreno de una película debe ser posterior a la fecha de finalización del rodaje de la misma.

```
CREATE TABLE PELICULA ( ...,
  CONSTRAINT pelicula_fechas_ok
        CHECK ( fecha_fin_rodaje < fecha_estreno ),  ... );
```

✋ Toda RI-TablaBase es **comprobada inmediatamente**:

toda operación de modificación sobre la relación incluye (conceptualmente) el chequeo de todas sus RI (como un paso final de dicha operación) **+acción**

SQL2: Reglas de Integridad GENERALES

- **Elemento de la base de datos, independiente de tablas ya existentes**
- Especifica **Restricciones de Integridad** que **NO** son…
  - ✓ de **clave candidata** (primaria o alternativa)
  - ✓ de **integridad referencial** (clave ajena -externa o foránea-)
- **Restricción** de complejidad arbitraria que **incluye** cualquier nº de **columnas** de cualquier nº de **relaciones base**
- Cada aserto tiene un **nombre** y consta de una **condición** (**cláusula CHECK**)
- **CREACIÓN DE UNA RESTRICCIÓN GENERAL (**Aserción o **Aserto)**

```
CREATE ASSERTION <nomRestricción>      -- nombre de restricción obligatorio
  CHECK ( <condición> ) ;
```

  - Si una tupla de la BD hace que condición=FALSE, la **restricción** habrá sido **violada**
  - Un **estado** de la **BD satisface** una **restricción** si **ninguna combinación de tuplas de dicho estado viola la restricción**

SQL2: Reglas de Integridad GENERALES

- Normalmente, condición se expresa "en negativo":

Todo X satisface Y ≡ Ningún X satisface NO(Y)

Todo actor representado por la agencia 1 debe cobrar 50.000 o más

```
CREATE ASSERTION age1_cache_ok
  CHECK (NOT EXISTS (SELECT * FROM ACTOR
                     WHERE agencia=1 AND cache<50000)) ;
```

Ninguna agencia representa a más de 40 actores

```
CREATE ASSERTION num_actores_age_ok
  CHECK (NOT EXISTS (SELECT * FROM AGENCIA
                     WHERE 40 < (SELECT SUM(*) FROM ACTOR
                                 WHERE codAge=agencia ) ) ) ;
```

Los actores no protagonistas de una película no pueden cobrar más que los protagonistas

```
CREATE ASSERTION paga_actores_ok
  CHECK (NOT EXISTS (SELECT * FROM ACTUA_EN ACT, ACTUA_EN PROTA
                     WHERE ACT.film=PROTA.film AND ACT.actor<>PROTA.actor
                     AND PROTA.papel='protagonista' AND ACT.paga>PROTA.paga ));
```

SQL2: Reglas de Integridad GENERALES

Debe de existir al menos una distribuidora de películas

```
CREATE ASSERTION existencia_distribuidora
  CHECK ( 0 < SELECT COUNT (*) FROM DISTRIBUIDORA ) ;
```

NOTA sobre este aserto:
- Debe crearse una vez que ya exista alguna tupla en DISTRIBUIDORA
- Operación DELETE puede o no violarla, pero nunca lo hará un INSERT

El código de cada guión es único (≡ si hay dos guiones con igual código, son el mismo)

```
CREATE ASSERTION guion_codigo_unico
  NOT EXISTS (SELECT * FROM GUION G1, GUION G2
              WHERE G1.codG = G2. codG
              AND NOT (G1.titulo=G2.titulo AND G1.descrip= G2.descrip));
```

NOTA sobre este aserto:
- Equivale a especificar UNIQUE(codG) en el CREATE TABLE

SQL2: Reglas de Integridad GENERALES

Los actores y películas anotados en la relación ACTUA_EN deben existir

```
CREATE ASERTION actor_actuaen_pelicula
  CHECK (NOT EXISTS (SELECT *
                     FROM ACTUA_EN AE, ACTOR A, PELICULA P
                     WHERE NOT (AE.actor = A.codA AND AE.film = P.codP) ) ;
```

NOTA sobre este aserto:
- Equivale a especificar
  FOREIGN KEY (actor) REFERENCES ACTOR(codA)...
  y FOREIGN KEY (film) REFERENCES PELICULA(codP)...
  en el CREATE TABLE

- **ELIMINACIÓN DE UNA RESTRICCIÓN GENERAL**

DROP ASSERTION <nomRestricción> **;**

--ojo: SIN opción RESTRICT o CASCADE

Antiguas versiones de SQL: Disparadores

- En muchos casos conviene especificar la **ACCIÓN** que ejecutar (disparar) **tras** la **violación de** una **restricción**:
  - **Abortar** la **transacción** que provoca la violación, o
  - **Informar** de ello al usuario (mensaje), o
  - **Ejecutar** cierto **procedimiento**, o
  - **Disparar otras actualizaciones**
    - » Esto se consigue mediante los TRIGGERs

TRIGGER (Disparador)

DEFINE TRIGGER <nombre> ON <lista tablas> :
<condición> ACTION_PROCEDURE <llamada procedimiento>

- Especifica una **condición** y una **acción** que realizar si la condición se cumple
- La **ejecución del Trigger** se considera **parte de la operación que intentó violar la RI**

Antiguas versiones de SQL: Disparadores

Disparador que, si algún actor en una película percibe una paga mayor que la de un protagonista, pida confirmación para permitir dicho estado o no

DEFINE TRIGGER Disp_Paga ON ACTUA_EN ACT, ACTUA_EN PROTA**:**
ACT.actor<>PROTA.actor AND ACT.film=PROTA.film
AND PROTA.papel="protagonista" AND ACT.paga > PROTA.paga
ACTION_PROCEDURE SolicitarAprobacion(...) ;

- **Diferencia entre ASSERTION y TRIGGER**
  - ASSERTION prohibe realizar una actualización que viola el aserto (es decir, que hace FALSE la condición)
  - TRIGGER puede permitir la actualización que cumple la condición (es decir, que viola una RI), pero ejecuta la acción (que puede reparar la violación, dejando consistente la BD)
    - » Las **condiciones** especificadas en una y otra son **inversas**
- Disparadores **combinan** los **enfoques declarativo y procedimental**
  - La **condición** del disparador es **declarativa**
  - Su **acción** opera por **procedimientos**

SQL2: *COMPROBACIÓN* DE RESTRICCIONES

- Normalmente, el **sistema comprueba** una **RI inmediatamente**, como último paso de la ejecución de una operación o sentencia SQL
- Si la RI es violada, la sentencia se cancela y su efecto sobre la BD es nulo

- *A veces es* **necesario** *que ciertas* **restricciones no** *sean* **chequeadas hasta pasado un tiempo** (si se comprobaran de inmediato, siempre fallarían)

Ejemplo (ciclo referencial) **T1 ⇄ T2** Inicialmente, **T1** y **T2** están vacías

```
CREATE TABLE T1
( atrib1 ... ,
  FOREIGN KEY (atrib1)
     REFERENCES T2 (...) ... ,
 ... ) ;
```

```
CREATE TABLE T2
( atrib2 ... ,
FOREIGN KEY (atrib2)
  REFERENCES T1 (...) ... ,
 ... ) ;
```

- Con Chequeo Inmediato de las RI Referencial, toda **inserción** de fila en cualquier tabla **fallaría**, pues nunca encontraría la fila *target* en la otra tabla

SQL2: COMPROBACIÓN DE RESTRICCIONES

- En un momento dado, dentro de cierta *transacción* SQL, toda RI debe estar en modo...
  - **INMEDIATE**: debe ser chequeada inmediatamente, **o**
  - **DEFERRED**: será chequeada al final de la transacción

- Toda **definición** de **Restricción de Integridad puede incluir** (al final) la cláusula

  [**INITIALLY** { **DEFERRED** | **IMMEDIATE** }]
  [[**NOT**] **DEFERRABLE**]

INITIALLY DEFERRED o **INITIALLY IMMEDIATE** especifican el **modo inicial** de la RI (después de ser definida la RI y al comienzo de cada transacción SQL)

DEFERRABLE o **NOT DEFERRABLE** indica **si** la RI **puede pasar a** modo **DEFERRED** o no

Nota: En el estándar **SQL2**...
- Restricciones de Dominio: no tiene sentido un chequeo diferido
- Restricciones NOT NULL y de Clave Candidata: deben ser NOT DEFERRABLE

SQL2: COMPROBACIÓN DE RESTRICCIONES

- Si no se indica INITIALLY DEFERRED ni INITIALLY IMMEDIATE, supone **INITIALLY IMMEDIATE**
- Si se indica INITIALLY IMMEDIATE (o se supone)...
  - Si no se indica DEFERRABLE ni NOT DEFERRABLE, se supone **NOT DEFERRABLE**,
  - Si se indica INITIALLY DEFERRED, **no** puede indicarse NOT DEFERRABLE,
  - Puede ponerse DEFERRABLE, aunque se supone

- Sentencia **SET CONSTRAINTS**

  SET CONSTRAINTS { <listaRestricciones> | **ALL** } { **DEFERRED** | **IMMEDIATE** }

  - Establece el **modo** para **restricciones** respecto de la **transacción** SQL **actual**
  - Toda RI mencionada debe ser DEFERRABLE ⇒ **ALL** significa todas las RI deferrables

  · **DEFERRED** hace que **todas** las RI pasen **a modo DEFERRED**

  · **IMMEDIATE** hace que pasen **a modo IMMEDIATE** y **son comprobadas, si el** chequeo de **alguna RI falla**, SET CONSTRAINTS falla y **ninguna** RI **cambia de modo**

- **COMMIT ⇒ SET CONSTRAINTS ALL IMMEDIATE**

  Si una comprobación de **RI falla ⇒ COMMIT falla ⇒ transacción** completa **falla** (rolledback)