

# Restricciones dinámicas en Bases de Datos

Mauricio Zambrano

Departamento de Ingeniería Informática y Ciencias de la Computación

Universidad de Concepción

`mauricio@inf.udec.cl`

**\* Resumen:** se presenta un método de creación y chequeo de restricciones dinámicas usando extensiones del MER. El método considera la especificación de restricciones de transición a nivel conceptual y la implementación de mecanismos de chequeo en SGBDR comerciales.

**Palabras claves:** integridad de bases de datos, restricciones dinámicas, restricciones de integridad, triggers.

## 1 Introducción

Las restricciones de integridad nacen con la definición de las bases de datos, al cumplirse estas reglas se obtiene un nivel de confianza en la integridad de los datos.

Se pueden clasificar en dos grupos:

1. las restricciones estáticas, que expresan propiedad independientes del estado de la base de datos, sólo dependen de su estado actual, por ejemplo, "el sueldo de un empleado es menor que el de un gerente".
2. las restricciones dinámicas que permiten expresar condiciones, generalmente relacionadas con el tiempo, sobre una secuencia de dos o más estados de una base de datos, por ejemplo "el estado civil de una persona sólo puede pasar de soltero a casado".

Un tipo particular de restricciones dinámicas consiste en las llamadas restricciones de transiciones que imponen restricciones en pares de estados, antes y después de una transacción.

En este artículo repasaremos los modelos ya existentes, luego veremos una propuesta para su modelamiento y finalmente se abordarán aspectos de la implementación de esta alternativa que no necesita ningún elemento adicional para implementar restricciones dinámicas.

## 2 Marco conceptual

Las restricciones simples como las de integridad estática han sido ampliamente estudiadas y han sido efectivamente implementadas en SGBDR. Restricciones más generales como las dinámicas no

son incluidas en estos sistemas por su complejidad.

Las restricciones dinámicas han sido abordados por varios autores[5] que han abordado distintos enfoques, proponiendo soluciones específicas entre las que destacan:

- Restricciones por tipo, ya sea dinámicas generales o transiciones.
- Tipo de base de datos. Se considera como punto de partida del método esta característica y se estudia una solución específica dependiendo de si es relacional, deductiva, orientada al objeto, activa o temporal.
- Tipo de refuerzo de restricción: si es de mantención o de chequeo de la integridad de la base de datos.
- Temporalidad del refuerzo de la restricción.
- Lenguaje de restricción usado. En el caso de las transiciones, basta con un lenguaje de lógica de primer orden como el de las restricciones estáticas extendido con operadores como antes y después pero para las restricciones dinámicas generales es necesario tener extensiones con cuantificadores temporales lógicos que pueden tener una representación implícita o explícita del tiempo.
- Direcciones temporales de la fórmula: cuando se usa un lenguaje lógico temporal implícito, los cuantificadores temporales pueden estar relacionados hacia el pasado (con operadores como *previamente*, *en el pasado*) o a futuro (*en el futuro*, *próximamente*) de las bases de datos. Si el lenguaje de restricción es un tipo de lógica temporal con tiempo explícita, se llama bidirijida, dependiendo del tiempo en que se evalúa la restricción y la relación entre las variables temporales de la fórmula, puede llamarse dirigida al pasado o presente.
- Almacenamiento de los datos históricos: nos indica si el método necesita o no almacenar el historial completo de las transacciones en la base de datos para reforzar las restricciones. Generalmente estas son activadas en la cercanía de un cambio en la base de datos. Los métodos que necesitan almacenar esta información son llamados *basados en historia*, los que no, *sin-historia*.

## 2.1 Aproximaciones en el contexto de las bases de datos relacionales

### 2.1.1 La aproximación de Chomicki[2]

En Chomicki (1992) se presenta un método para la verificación de restricciones dinámicas que asume una formulación de las restricciones en lógica temporal pasada sin presencia explícita del tiempo. Este método es sin historial ya que todo estado de la base de datos es extendido de relaciones auxiliares que contienen la información histórica necesaria para verificar las restricciones. Las relaciones auxiliares pueden ser implementadas como vistas y son derivadas de la definición de las restricción. De esta forma, el chequeo de la restricción se realiza con los datos existentes en el momento.

Esta aproximación hace que estas restricciones dinámicas sean parecidas a las estáticas ya que ambas con definidas en lógica de primer orden.

### 2.1.2 La aproximación de Bidoit y De Amo [1]

Introduce un clase de restricciones dinámicas en el marco del modelo relacional: las dependencias dinámicas algebraicas (DAD). Estos DAD permiten expresar que 'si una propiedad es válida ahora, entonces en el pasado tenía alguna otra propiedad'. Esta aproximación es también sin historial.

En esta investigación revisaremos un método sin historial, que usa elementos del MER para modelar transformaciones dinámicas y restricciones dinámicas más generales. Al especificar claramente las restricciones dinámicas, el diseñador puede concentrarse en las políticas (lo que debe ser reforzado) en vez de preocuparse por el mecanismo, se simplifica las aplicaciones que se ejecuten sobre la base de datos, posibilita el cambio de las restricciones sin cambiar el esquema estático de la base de datos ni las aplicaciones. Esto permite revisar claramente la consistencia de la restricción[4]. Este método incluye en su diseño, elementos temporales añadidos al MER que permiten modelar cualquier transición.

Otros modelos que en la actualidad manejan a alto nivel restricciones de este tipo son el CCER[6] y el UML mediante su extensión OCL[7] aunque no son todavía soportados por algún SGBD.

## 3 Propuesta

Se pueden definir al momento del diseño conceptual las restricciones dinámicas sin afectar el modelo estático. Esto se consigue añadiendo un documento que define explícitamente todas las transiciones dinámicas de los atributos que puedan adaptarse a este tipo de restricción. Se encontró una forma genérica de modelar las restricciones dinámicas generales y las transiciones, con lo que podemos asegurar el chequeo e integridad de la base de datos.

Pasos para crear una base de datos con restricciones dinámicas de integridad:

1. Crear el modelo estático.
2. Identificar todos los atributos con restricciones dinámicas.
3. Identificar todos los estados válidos.
4. Identificar todos los estados iniciales.
5. Identificar todas las transiciones entre estados de un mismo atributo.
6. Ingresar estos datos a la base de datos.
7. activar los triggers correspondientes a cada entidad.

Los pasos 1 a 5 corresponden al nivel conceptual y los restantes a la implementación. Básicamente sólo hay que preocuparse de diseñar y de identificar los componentes de la restricción, la implementación es la misma para todas las transiciones.

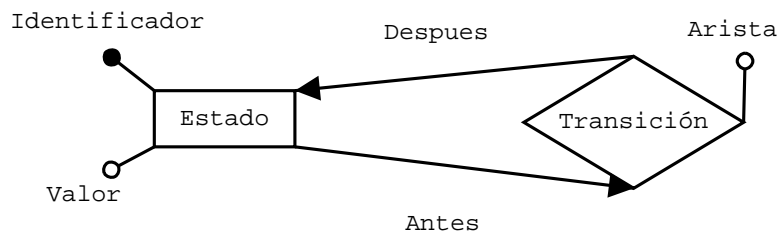


Figura 1: Modelo genérico para definir una restricción dinámica

### 3.1 Restricciones dinámicas genéricas

Para las transiciones se propone la siguiente solución:

Cada restricción dinámica es definida mediante un grafo (ver figura 2), como título del diagrama debe ir el nombre de la entidad y el atributo que sufre esta restricción. Se deben especificar todos los estados permitidos y todas las transiciones asociadas a la restricción. Además, deben señalarse todos los estados iniciales permitidos. El nombre de la entidad (futura tabla) es importante para el elemento encargado del chequeo(trigger).

Esta solución es solamente viable si el conjunto de estados es conocido y es un conjunto numerable. Sólo en algunos casos de conjuntos no finitos se podrá implementar con una ligera variación si se conoce el orden dentro del conjunto de los estados. En este último caso, bastaría con modificar los elementos de chequeo dinámico de la restricción ya que sería imposible ingresar todos los estados. Un ejemplo sería la restricción dinámica que el sueldo de un empleado no disminuye. Claramente se conocen todas las relaciones de orden en los reales y es imposible tener todos los reales almacenados, peor aún con las relaciones. Pero el computador conoce de antemano las relaciones de orden en este dominio por lo que no es necesario ingresar ni los elementos, ni las transiciones permitidas.

Estos datos se almacenan como lo muestra la figura 1 en 2 relaciones, la de estados, la de transiciones y por motivos de implementación, se añade una tercera que almacena todos los estados iniciales permitidos, indicados con  $\Lambda$  ya que la restricción está compuesta de dos partes, una primera en la que se chequea si el estado inicial ingresado es permitido y una segunda en la que se revisa si la transición es válida. Por lo tanto se almacena la restricción junto a los datos.

La particularidad de la relación es que sus brazos implican una relación temporal ya que por un lado une al elemento anterior (por el lado de antes) y por el otro une el elemento futuro o anterior.

Los elementos dinámicos que vigilan la integridad son dos triggers, uno de inserción y el otro de actualización de datos.

### 3.2 Fuentes de inconsistencias

Como podemos sospechar, los cambios de entidades pueden ser fuente de inconsistencia en los datos o de pérdida de información si no conservan los atributos y relaciones de la entidad pasada. Para cada caso es necesario evaluar esta necesidad. En este artículo no nos ocuparemos de tales casos.

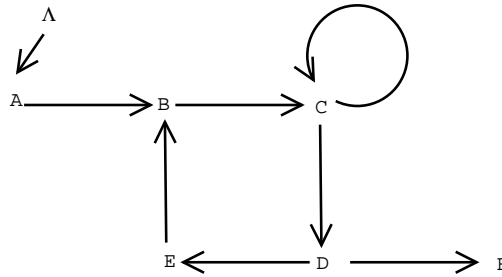


Figura 2: Grafo de la restricción implementada

Dentro de lo posible se tratará que las restricciones generadas no creen inconsistencias a ningún nivel. Como cada restricción puede ser definida como una lógica proposicional de primer orden, se puede verificar con relativa facilidad si produce el efecto deseado.

La primera restricción básica e importante se refiere a la definición explícita de los dominios dinámicos, es decir se deben conocer todos sus elementos y todas las transiciones entre estos elementos.

### 3.3 Ingreso a la base de datos de estas restricciones

El primer paso consiste en detectar todos los estados posibles para el atributo. Luego el modelador debe ingresar todas las transiciones permitidas. Como no se puede suponer ninguna propiedad sobre este conjunto, sólo se supondrá que hay relaciones entre elementos anteriores y posteriores relacionados, es decir no supondremos propiedades como transitividad o reflexividad, sólo relaciones.

Claramente, los únicos elementos que se activan al producirse un cambio en una base de datos son los triggers. Debido a la naturaleza de este modelamiento de restricciones dinámicas, podemos suponer que son fácilmente mapeables a sql. Los triggers que refuerzan esta restricción son genéricos, por lo que sin mayores modificaciones pueden ser utilizados para modelar cualquier restricción dinámica.

Se han creado exitosamente una aplicación que almacena estados, transiciones y estados iniciales con el que efectivamente se pueden crear restricciones dinámicas[8].

## 4 Implementación

Se realizó en Oracle8i para linux, en otros trabajos[4] ya se había realizado en bases de datos activas como Ingres y Stardust, pero con procedimientos mucho más complicados y compiladores a nivel de la restricción. La tecnología disponible en la actualidad nos permite con pocos elementos la creación de estas restricciones.

La estructura de este trigger no permite transitividad pero se podría implementar. Uno de los

principales problemas de esta restricción es el alto costo de procesamiento que requiere ya que para chequear una actualización, hay que hacer una consulta de tres tablas. Habría que valerse de entidades adicionales para mejorar la eficiencia de esta restricción que almacenarían por ejemplo todos los futuros estados a partir de un valor actual por ejemplo. Las bases teóricas de una estructura similar fueron descritas[3] y siguen en este caso la estructura:

```
SI actualiza
CUENTE elementos RELACIONADOS
SINO levanta ERROR
FIN
```

En caso de encontrar elementos, la transición o inserción es aceptada. Respecto a una solución que admita transitividad vemos que la complejidad de tal algoritmo aumenta exponencialmente por lo que puede constituirse en un gran inconveniente pero podremos, con el avance de la velocidad de procesamiento y optimizaciones en las bases de datos ir reduciendo año a año estos problemas.

## 4.1 Triggers

Chequeo de inserciones:

```
CREATE OR REPLACE TRIGGER ins_check0 BEFORE INSERT on tabla
for each row
declare
    numero integer;
    no_puede EXCEPTION;
begin
    SELECT COUNT(*) into numero
    FROM inicial, estado
    where inicial.identificador = estado.identificador
and estado.valor=:NEW.estado;
    if ( numero < 1 )
        THEN RAISE no_puede;
    end if;
EXCEPTION
    WHEN no_puede
    THEN raise_application_error(-20240,'NO existe ese estado inicial');

end;
```

Se revisa si la inserción es permitida mediante el query que cuenta la cantidad de elementos que tienen la característica dada.

Chequeo de transiciones:

```

CREATE OR REPLACE TRIGGER ins_check1 BEFORE UPDATE on tabla
for each row
declare
    numero integer;
    no_despues EXCEPTION;
begin
    SELECT count(*) into numero
    FROM transicion
    WHERE transicion.antes =
(SELECT estado.identificador FROM estado WHERE estado.valor=:OLD.estado )
and transicion.despues = (SELECT estado.identificador
    FROM estado WHERE estado.valor=:NEW.estado);
    if ( numero < 1 )
        THEN RAISE no_despues;
    end if;
EXCEPTION
    WHEN no_despues
    THEN raise_application_error(-20241,'No existe esa transicion');
end;

```

La verificación se genera mediante un query compuesto de dos otro subqueries y funciona contando la cantidad de estados siguientes dado un valor inicial. Si son menos de uno, no existe esa transición.

## 4.2 Tablas

```

CREATE TABLE estado(
    identificador NUMBER,
    valor VARCHAR2(100) NOT NULL,
    CONSTRAINT u_valor UNIQUE (valor),
    CONSTRAINT pk_estado PRIMARY KEY (identificador)
);

CREATE TABLE transicion(
    arista NUMBER,
    antes,
    despues,
    CONSTRAINT pk_transicion PRIMARY KEY (antes, despues),
    CONSTRAINT fk_antes FOREIGN KEY (antes)
        REFERENCES estado(identificador)
    ON DELETE CASCADE,
    CONSTRAINT fk_despues FOREIGN KEY (despues)
        REFERENCES estado(identificador)
    ON DELETE CASCADE
);

```

```
CREATE TABLE inicial(
    identificador NUMBER,
    CONSTRAINT pk_inicial PRIMARY KEY (identificador),
    CONSTRAINT fk_inicial FOREIGN KEY (identificador)
    REFERENCES estado(identificador)
    ON DELETE CASCADE);
```

tabla es el nombre de la tabla de prueba.

## 5 Conclusiones

Con los elementos disponibles en la actualidad en los SGBD, se pueden implementar sin dificultades procedimientos de restricciones dinámicas. Esta restricción se apoya en el MER y extensiones lógicas temporales. Se genera una base de datos con restricciones dinámicas sin historial. Cada restricción dinámica es definida al momento del diseño conceptual mediante un grafo que representa todos los estados y transiciones permitidos. El tamaño de estas restricciones está en directa relación con la cantidad de elementos que la componen.

Se probó un prototipo web[8] con la restricción definida por la figura 2 que ya ha procesado exitosamente más de 100 transacciones.

Otro inconveniente que se descubre es la falta de un modelo para especificar restricciones sobre conjuntos de datos. Se llega a la contradicción de que existe un modelo para definir los datos (MER) pero no existe uno para modelar su comportamiento.

En este artículo se presentaron maneras de implementar restricciones sobre conjuntos no finitos (restricciones más generales) queda por especificar más claramente las formas de solucionar dichos problemas genéricos. Se pretende mejorar su diseño y profundizar más las implicaciones de tal restricción y la finalización de una herramienta para su modelamiento.

## Referencias

- [1] N. Bidoit and S. De Amo. Contraintes dynamiques d'inclusion et schémas transactionnels, 1994.
- [2] J. Chomicki. History-less checking of dynamic integrity constraints. In *Proc. IEEE CS Intl. Conf. No. 8 on Data Engineering, Tempe, AZ*, 1992.
- [3] Jan Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Transactions on Database Systems*, 20(2):149–186, 1995.
- [4] Jan Chomicki and David Toman. Implementing temporal integrity constraints using an active DBMS. *Knowledge and Data Engineering*, 7(4):566–582, 1995.
- [5] Pacheco Silva. Dynamic integrity constraints definition and enforcement in databases: a classification framework.



- [6] M. Varas. Ccer: Constraint centered entity relationship model, 2000.
- [7] Varios. Object constraint language specification, version 1.1, <http://www-4.ibm.com/software/-ad/library/standards/ocl.html>, 1997.
- [8] Mauricio Zambrano. Restricciones dinámicas, <http://arcadia.inf.udec.cl/mauro/prototipo>, 2001.