Modelo Relacional

Guido Urdaneta (Revisado por E. DeBourg, H. Corzo y A. Romero)

26 de abril de 2004

El modelo relacional establece una forma de representar los datos mediante tablas y de manipularlos.

El modelo relacional se divide en tres partes:

1. Estructura de los datos
2. Integridad de los datos
3. Manipulación de los datos

1. Estructura[1]

1.1. Conceptos Básicos

- *Relación*: corresponde en términos generales a una Tabla.
- *Tupla*: una fila de esa Tabla.
- *Cardinalidad*: número de Tuplas en una Tabla.
- *Atributo*: una columna de la Tabla.
- *Grado*: número de atributos de una Tabla.
- *Clave primaria*: una columna o combinación de columnas tal que no existen dos filas con los mismos valores en dichas columnas.
- *Dominio*: colección de valores de los cuales los atributos obtienen sus valores reales.

Equivalencias informales:

Relación → Tabla

Tupla → Fila

Cardinalidad → Número de Filas

Atributo → Columna o campo

Grado → Número de Columnas

Clave Primaria → Identificador único

Dominio → Valores legales

1.2. Dominios

El punto de partida para el estudio del modelo relacional es la *menor unidad semántica de información*, la cual suponemos es el valor de un dato individual. Llamaremos a estos valores escalares. Estos valores suponemos que son atómicos en el sentido de que son indivisibles sin perder su significado.

Un dominio es un conjunto de valores escalares, todos del mismo tipo. Un ejemplo de dominio de valores es el conjunto de todos los enteros positivos.

La importancia de los dominios en el modelo relacional radica en el hecho de que éstos restringen las comparaciones. Por ejemplo, no tiene mucho sentido comparar un peso con una cadena de caracteres o un peso con un monto de dinero, ya que tienen dominios distintos.

Un aspecto importante de los dominios es su naturaleza conceptual. Es posible que estén almacenados explícitamente en la base de datos, pero en la mayoría de los casos no es así. Sin embargo deberán especificarse en la definición de la base de datos (e.g. CREATE DOMAIN NOMBRE CHAR(10))

Puede decirse que un nombre más preciso para los dominios de valores escalares es el de dominios simples, para distinguirlos de los dominios compuestos. Un dominio compuesto es una combinación de dominios simples. Por ejemplo, podría definirse un dominio FECHA_GREGORIANA como la combinación de tres enteros DÍA, MES y AÑO o un dominio COMPLEJO con dos números reales REAL e IMAGINARIO.

1.3. Relaciones

Una relación sobre un conjunto de dominios $D_1, D_2, ..., D_n$ (no necesariamente todos distintos) es un par ordenado (*cabecera*, *cuerpo*) donde: .

- La cabecera está formada por un conjunto fijo de atributos, o en términos más precisos, de pares atributo-dominio $\{(A_1, D_1), (A_2, D_2), ..., (A_n, D_n)\}$ tales que cada atributo A_j corresponde a uno y sólo uno de los dominios subyacentes D_j ($j = 1, 2, \cdots, n$).
- El cuerpo está formado por un conjunto de tuplas, el cual varía con el tiempo. Cada tupla está formada por un conjunto de pares atributo-valor $\{(A_1, v_{i1}), (A_2, v_{i2}), ..., (A_n, v_{in})\}$ ($i = 1, 2, ..., m$), donde *m* es el número de tuplas del conjunto). En cada tupla hay un par atributo-valor para cada atributo A_j de la cabecera. Para cada par atributo-valor(A_j, v_{ij}), v_{ij} es un valor del dominio único D_j asociado al atributo A_j.

Los valores *m* y *n* reciben el nombre de cardinalidad y grado.

Las relaciones tienen las siguientes propiedades:

1. No hay tuplas repetidas.
2. Las tuplas no están ordenadas.
3. Los atributos no están ordenados.
4. Todos los valores de los atributos son atómicos (Esto significa que todas los relaciones en el modelo relacional están normalizadas, o más precisamente en primera forma normal o 1NF).

1.3.1. Tipos de Relaciones

En un sistema relacional pueden existir los siguientes tipos de relaciones:

1. *Relaciones base*: Una relación base es una relación autónoma (no se deriva de otras relaciones), con nombre y forma parte de la base de datos de manera directa.
2. *Vistas (relaciones virtuales)*: Son relaciones derivadas con nombre, representadas dentro del sistema únicamente mediante su definición en términos de otras relaciones con nombre
3. *Instantáneas (snapshots)*: Una relación instantánea es también una relación derivada, pero a diferencia de las vistas, las instantáneas son reales en el sentido de que están definidas no sólo por su definición en términos de otras relaciones sino también por sus propios datos almacenados. La creación de un snapshot es similar a la ejecución de una consulta, pero son de sólo lectura y el resultado es almacenado de forma periódica (es decir, cada cierto tiempo se ejecuta la consulta y el resultado se convierte en el nuevo valor del snapshot).
4. *Resultados de consultas*: Son relaciones resultantes de la ejecución de una consulta. Pueden o no tener nombre (normalmente no lo tienen) y no tienen existencia persistente en la base de datos.
5. *Resultados intermedios*: Un resultado intermedio es una relación resultante de una expresión relacional anidada dentro de otra expresión relacional más grande.
6. *Relaciones temporales*: Es una relación con nombre, similar a una relación base pero que es destruida automáticamente en un momento determinado (e.g. al terminar una transacción).

1.4. Bases de datos relacionales

Una base de datos relacional es una base de datos percibida por el usuario como una colección de relaciones normalizadas de diversos grados que varía con el tiempo.

Es importante la frase "percibida por el usuario" ya que las ideas del modelo relacional se aplican a un nivel externo o conceptual en un sistema de bases de datos, pero no a un nivel interno. Este modelo representa un nivel de abstracción independiente de los detalles de la máquina y el sistema de almacenamiento subyacente.

2. Integridad[1]

Toda base de datos debe incluir un conjunto de reglas de integridad que permitan garantizar que los valores de los datos almacenados sólo puedan reflejar la realidad. Por ejemplo, no tiene sentido que un valor de edad sea un número negativo, todas las partes de determinado color deben despacharse a determinado destino, etc.

La mayoría de las reglas de integridad son específicas de la base de datos correspondiente. Los ejemplos indicados previamente son específicos en ese sentido. El modelo relacional, sin embargo, incluye dos reglas generales de integridad (generales en el sentido en que aplican a todas las bases de datos relacionales). Estas reglas se refieren a los conceptos de *claves primarias* y *claves foráneas* o *ajenas*.

Las reglas de integridad (tanto específicas como generales) se aplican a las relaciones base ya que son éstas las que almacenan la información que se supone debe reflejar la realidad y por lo tanto deben contener sólo configuraciones correctas (o al menos factibles) de valores.

2.1. Claves Primarias

El conjunto de atributos K de la relación R es una *clave candidata* si y sólo si satisface las siguientes dos propiedades:

1. *Unicidad*: Nunca existen dos tuplas en R con el mismo valor de K.
2. *Minimalidad*: Si K está compuesto por más de un atributo, no será posible eliminar ningún elemento de K sin destruir la propiedad de unicidad.

Del conjunto de claves candidatas de una relación dada, se elige una y sólo una como *clave primaria* de esa relación; las demás, si existen, se llamarán *claves alternativas*.

Se denomina *superclave* a un conjunto de atributos S que satisface la propiedad de unicidad, sin requerir que satisfaga la propiedad de minimalidad.

Surgen los siguientes puntos:

1. Toda relación tiene por lo menos una clave candidata (que sería la clave primaria) sin excepción.
2. El razonamiento para elegir la clave primaria está fuera del modelo relacional.
3. La clave primaria es la que tiene verdadera importancia; los conceptos de clave candidata y alternativa surgen como parte del proceso de definir el concepto más importante de clave primaria.
4. Aunque es verdad que toda relación posee una clave primaria, el concepto se aplica de manera más directa a las relaciones base.

2.2. Regla de Integridad de las Entidades

La primera de las dos reglas generales de integridad relacional se conoce como regla de integridad de las entidades.

- *Ningún componente de la clave primaria de una relación base puede aceptar nulos*.

Con "nulo" se refiere a información faltante por alguna razón (no aplica, no se conoce, etc.).

La justificación de esta regla es la siguiente:

1. Las tuplas de una relación base corresponden a entidades del mundo real.
2. Las entidades son distinguibles (identificables).
3. Por lo tanto, los representantes de entidades en la base de datos deben ser distinguibles.
4. Admitir nulos en la clave primaria significaría que es posible una entidad sin identificación.
5. Si nulo significa "No Aplicable" entonces la tupla carece de sentido.
6. Si nulo se desconoce surgen todo tipo de problemas (e.g. podría representar una entidad conocida, no se podría saber cuantas entidades hay, etc.).
7. Una entidad sin identidad es una contradicción de términos.
8. Los mismos argumentos aplican para valores de clave primaria parcialmente nulos.

En una base de datos relacional nunca se registra información de algo que no se pueda identificar.

Es importante recalcar lo siguiente:

1. En el caso de claves primarias compuestas la regla de integridad de entidades dice que cada valor individual debe ser no nulo en su totalidad
2. La regla de integridad de entidades sólo aplica a relaciones base. Es posible que, por ejemplo, el resultado de una consulta produzca una relación con nulos en la clave primaria.
3. La regla de integridad de entidades sólo aplica a las claves primarias. Es admisible permitir nulos en las claves alternativas (e.g. un número de cédula no es necesariamente obligatorio, pero si se tiene debe ser único).

2.3. Claves Foráneas

Una clave foránea es un conjunto de atributos de una relación R_2 cuyos valores deben concordar con los de la clave primaria de una relación R_1 (R_1 y R_2 no son necesariamente distintos) (no se requiere lo inverso, es decir, la clave primaria de R_1 puede contener valores que no estén referenciados por tuplas en la relación R_2).

Un conjunto de atributos ϕ de una relación base R_2 es una clave foránea si y sólo si:

1. Cada valor de ϕ es totalmente nulo o totalmente no nulo.
2. Existe una relación base R_1 con clave primaria K tal que cada valor no nulo de ϕ es idéntico al valor de K en alguna tupla de R_1.

Consideraciones:

1. Una clave foránea y la clave primaria asociada deben definirse sobre el mismo dominio.
2. La clave foránea no necesita ser un componente de la clave primaria de la relación que la contiene. En general, cualquier atributo puede ser una clave foránea.
3. Una relación R_1 dada puede ser tanto una relación referida como una relación referencial (e.g. $R_3 \Rightarrow R_2 \Rightarrow R_1$).
4. Puede darse el caso de un ciclo referencial: $R_n \Rightarrow R_{n-1} \Rightarrow ... \Rightarrow R_2 \Rightarrow R_1 \Rightarrow R_n$. De hecho, las relaciones autoreferenciales ("interrelaciones unarias") son el caso particular $R_1 \Rightarrow R_1$.
5. Las claves foráneas pueden, en ocasiones, aceptar nulos.
6. Se utilizan las concordancias entre claves foráneas y primarias para expresar interrelaciones entre tuplas.

2.4. Regla de Integridad Referencial

La segunda regla general de integridad es la regla de integridad referencial:

- *La base de datos no debe contener valores de clave foránea sin concordancia*

La justificación de esta regla es obvia: así como los valores de clave primaria representan identificadores de entidades, los valores de clave foránea representan referencias a entidades (a menos que sean nulos, desde luego). La regla de integridad referencial dice tan solo que si B referencia a A, entonces A debe existir.

Consideraciones:

1. La regla de integridad referencial exige concordancia de las claves foráneas con las claves primarias, no con las claves alternativas.
2. Los conceptos de integridad referencial y claves foráneas se definen uno en términos del otro, así los términos "manejo de claves foráneas" y "manejo de integridad referencial" significan exactamente lo mismo.

2.5. Reglas para Claves Foráneas

Para cada clave foránea es necesario responder tres preguntas:

1. ¿Puede aceptar nulos la clave foránea?.
2. ¿Qué pasa si se intenta eliminar el objetivo de una clave foránea?. En este caso existen tres posibilidades:
   - Restricción: Impedir la eliminación del objetivo
   - Propagación (CASCADE): La operación de eliminación se propaga (en cascada) eliminando las tuplas con la clave foránea (e.g. eliminar un cliente causa la eliminación de los envíos asociados).
   - Nulificación (SET NULL): Se asignan nulos a la clave foránea y se elimina el objetivo.
3. ¿Que pasa si se intenta modificar el valor de la clave primaria?. En este caso existen tres posibilidades:
   - Restricción: Impedir la modificación del objetivo
   - Propagación (CASCADE): La operación de modificación se propaga (en cascada) modificando las tuplas con la clave foránea.
   - Nulificación (SET NULL): Se asignan nulos a la clave foránea y se modifica el objetivo.

3. Manipulación[4, 2]

La manipulación de datos en el modelo relacional se refiere a las operaciones de inserción, modificación, eliminación y consulta de datos. Generalmente, la principal operación realizada por los usuarios es la consulta. El álgebra relacional ofrece la posibilidad de estudiar la operación de consulta de datos a una base de datos relacional desde un punto de vista abstracto.

El álgebra relacional es un álgebra especial que permite construir nuevas relaciones a partir de relaciones existentes. Las expresiones del álgebra relacional utilizan relaciones como operandos y se denominan consultas (*queries*). Puede decirse que el álgebra relacional es un lenguaje de consulta.

Las operaciones del álgebra relacional pueden dividirse en cuatro categorías:

1. Las operaciones usuales de la teoría de conjuntos: *unión*, *intersección* y *diferencia*.

2. Operaciones que remueven partes de una relación: *proyección*, que permite eliminar atributos de una relación; y *selección*, que permite eliminar tuplas.

3. Operaciones que combinan las tuplas de dos relaciones: *producto cartesiano*, *producto* (*join*) *natural* y *producto* (*join*) *teta*.

4. Una operación llamada "renombrar", que no afecta las tuplas de una relación, pero cambia la cabecera de la relación, es decir, los nombres de los atributos y/o el nombre de la relación.

3.1. Operaciones de teoría de conjuntos

Las operaciones más comunes de la teoría de conjuntos son la unión, intersección y diferencia. En el álgebra relacional estas operaciones actúan sobre los cuerpos de las relaciones (e.g. $R \cap S$ produce una relación cuyo cuerpo contiene la intersección de los conjuntos de tuplas de las relaciones *R* y *S).* Estas operaciones requieren que las relaciones utilizadas como operandos tengan cabeceras idénticas.

id	nombre	sexo
1111	Lucía Bustamante	F
2222	Andreína Romero	F

Relación X

id	nombre	sexo
1111	Lucía Bustamante	F
3333	Elizabeth De Bourg	F

Relación Y

Figura 1: Dos relaciones

Ejemplo: Suponga que se tienen las dos relaciones X y Y indicadas en la Figura 1, entonces la unión $X \cup Y$ es

id	nombre	sexo
1111	Lucía Bustamante	F
2222	Andreína Romero	F
3333	Elizabeth De Bourg	F

Nótese que la tupla de Lucía Bustamante aparece sólo una vez a pesar de estar en ambas relaciones.

La intersección $X \cap Y$ es

id	nombre	sexo
1111	Lucía Bustamante	F

Nótese que Lucía Bustamante es la única que aparece porque es la única que está en ambas relaciones.

La diferencia $X - Y$ es

id	nombre	sexo
2222	Andreína Romero	F

En este caso las tuplas de Lucía y Andreína aparecen en X y por lo tanto son candidatas para $X - Y$, sin embargo, la tupla de Lucía también aparece en Y y por lo tanto no está en $X - Y$.

3.2. Proyección

El operador de proyección se utiliza para producir a partir de una relación *R* una nueva relación que tiene sólo algunas de las columnas de *R*. El valor de la expresión $\Pi_{A_1,A_2,...,A_n}(R)$ es una relación cuya cabecera incluye sólo los atributos $A_1, A_2, ..., A_n$de *R*.

Ejemplo: Considérese la relación *X* de la Figura 1. La relación resultante de la expresión $\Pi_{id,nombre}(X)$ es

id	nombre
1111	Lucía Bustamante
2222	Andreína Romero

La relación resultante de la expresión $\Pi_{sexo}(X)$ es

sexo
F

Nótese que sólo hay una tupla en la relación resultante, pues ambas tuplas de X tienen el mismo valor para el atributo sexo.

3.3. Selección

El operador de selección, aplicado a una relación *R*, produce una nueva relación cuyo cuerpo contiene un subconjunto de las tuplas de *R*. Las tuplas de la relación resultante son aquellas que satisfacen una condición *C* que involucra los atributos de *R*. Esta operación se denota $\sigma_C(R)$. La cabecera de la relación resultante es la misma de *R* y por conveniencia se muestran los atributos en el mismo orden en que se muestran para *R*[1].

C es una expresión lógica (tiene un valor verdadero o falso) cuyos operandos son constantes o atributos de *R*. Se aplica *C* a cada tupla *t* de *R* sustituyendo para cada atributo *A* que aparezca en la condición *C*, el componente de *t* para el atributo *A*. Si *C* es verdadera después de sustituir todos los atributos que apliquen, entonces *t* es una de las tuplas que aparecen en el resultado de $\sigma_C(R)$, de otro modo *t* no está en el resultado.

Ejemplo: Considérese la relación *X* de la Figura 1. La relación resultante de la expresión $\sigma_{id>2000}(X)$ es

id	nombre	sexo
2222	Andreína Romero	F

La tupla de Lucía no aparece en la relación resultante pues no satisface la condición $id > 2000$ pues cuando sustituimos el valor 1111 por *id* la condición se vuelve $1111 > 2000$, lo cual es falso. La tupla de Andreína sí aparece en el resultado pues al sustituir el valor 2222 en la condición queda la expresión $2222 > 2000$, la cual es verdadera.

[1]Recuérdese que en el modelo relacional los atributos de una relación no tienen orden. Se muestran siguiendo un orden determinado sólo para mejorar la inteligibilidad

A	B
1	2
3	4

Relación R_1

B	C	D
2	5	6
4	7	8
9	10	11

Relación R_2

A	$R_1.B$	$R_2.B$	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Producto Cartesiano $R_1 \times R_2$

Figura 2: Dos Relaciones

3.4. Producto Cartesiano

En álgebra relacional, el producto cartesiano de dos relaciones *R* con cabecera $\{A_1, A_2, ..., A_n\}$ y *S* con cabecera $\{B_1 B_2, ..., B_m\}$es la relación $R \times S$ cuya cabecera es $\{A_1 A_2, ..., A_n, B_1 B_2, ..., B_m\}$ y cuyo cuerpo incluye todas las tuplas $\{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n), (B_1, b_1), (B_2, b_2), ..., (B_m, b_m)\}$ tales que $\{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n)\}$ es una tupla de *R* y $\{(B_1, b_1), (B_2, b_2), ..., (B_m, b_m)\}$ es una tupla de *S*.

Como puede verse, la cabecera de $R \times S$ es la unión de las cabeceras de *R* y *S*. Sin embargo, es posible que *R* y *S* tengan algunos atributos con el mismo nombre; en este caso es necesario inventar un nuevo nombre para al menos uno de cada par de atributos con el mismo nombre. Para eliminar la ambigüedad de un atributo *A* que aparezca en *R* y *S* se utilizará el nombre *R.A* para el atributo *A* de *R* y el nombre *S.A* para el de *S*. En general, para cualquier atributo *A* de una relación *R* se podrá utilizar la notación *R.A* para referirse a dicho atributo, incluso si no hay ambigüedad en el nombre.

Ejemplo: La Figura 2 muestra el producto cartesiano de dos relaciones.

Nótese que las tuplas del producto cartesiano $R_1 \times R_2$ resultan de todas las posibles combinaciones de las tuplas de R_1 y R_2 y que el atributo *B* de cada relación aparece precedido del nombre de la relación correspondiente para evitar ambigüedades.

3.5. Producto Natural (*Join*)

El producto natural (*join*) de dos relaciones *R* con cabecera $\{A_1, A_2, ..., A_n, C_1, C_2, ..., C_k\}$ y *S* con cabecera $\{B_1, B_2, ..., B_m, C_1, C_2, ..., C_k\}$ es la relación $R \bowtie S$ con la cabecera

$\{A_1, A_2, ..., A_n, B_1, B_2, ..., B_m, C_1, C_2, ..., C_k\}$ y con el cuerpo conformado por las tuplas $\{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n), (B_1, b_1), (B_2, b_2), ..., (B_m, b_m), (C_1, c_1), (C_2, c_2), ..., (C_k, c_k)\}$ que satis-

facen la condición de que existe en *R* la tupla
$\{(A_1, a_1), (A_2, a_2), ..., (A_n, a_n), (C_1, c_1), (C_2, c_2), ..., (C_k, c_k)\}$ y en *S* la tupla
$\{(B_1, b_1), (B_2, b_2), ..., (B_m, b_m), (C_1, c_1), (C_2, c_2), ..., (C_k, c_k)\}$.

Ejemplo: El producto natural $R_1 \bowtie R_2$ de las relaciones de la Figura 2 es

A	*B*	*C*	*D*
1	2	5	6
3	4	7	8

En este ejemplo el único atributo común de las relaciones R_1 y R_2 es el atributo *B*. De esta manera, sólo aparecen en la relación resultante aquellas tuplas que tienen en común el atributo *B.* En este ejemplo, el atributo *B* de la primera tupla de R_1 coincide con el de la primera tupla de R_2, lo cual da el resultado $(1, 2, 5, 6)$. La segunda tupla de R_1 coincide con la segunda tupla de R_2, lo cual da el resultado $(3, 4, 7, 8)$. La tercera tupla de R_2 no coincide con ninguna de R_1 y por lo tanto no tiene ningún efecto en el resultado.

La Figura 3 muestra otro ejemplo del producto natural de dos relaciones. En este caso, las cabeceras de las relaciones *U* y *V* comparten dos atributos, *B* y *C*. La primera tupla de *U* coincide (en los atributos comunes) con las dos primeras tuplas de *V* y la segunda y tercera tuplas de *U* coinciden con la tercera tupla de *V*.

A	*B*	*C*
1	2	3
6	7	8
9	7	8

Relación *U*

B	C	D
2	3	4
2	3	5
7	8	10

Relación *V*

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Relación $R \bowtie S$

Figura 3: Producto Natural de dos Relaciones

3.6. Producto Exterior (*Outer Join*)

El producto exterior (*outer join*) es una operación que produce una relación similar a la producida por el producto natural, pero con la única diferencia de que las tuplas de una relación que no tienen coincidencia en los atributos comunes aparecen también en el resultado con valores nulos para los atributos que corresponden a la otra relación. Por ejemplo, el producto exterior $R_1 \bowtie_O R_2$ de las relaciones de la Figura 2 es

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_6	c_7

Relación P

A	B	D
a_1	b_1	d_1
a_3	b_3	d_3
a_5	b_5	d_5

Relación Q

A	B	C	D
a_1	b_1	c_1	d_1
a_3	b_3	c_3	d_3
a_2	b_2	c_2	null
a_4	b_4	c_4	null
a_5	b_6	c_7	null
a_5	b_5	null	d_5

Relación $P \bowtie_O Q$

Figura 4: Dos relaciones y su producto exterior

A	B	C	D
1	2	5	6
3	4	7	8
null	9	10	11

En este caso, la tercera tupla de R_2 no coincide con ninguna de R_1 (en el atributo común B) y por lo tanto en el resultado los atributos que corresponden a R_1 aparecen nulos.

La Figura 4 muestra el producto exterior de dos relaciones con tuplas sin coincidencia tanto en el operando izquierdo como en el derecho.

3.7. Producto Exterior Izquierdo (*Left Outer Join*)

El producto exterior izquierdo (*left outer join*) es similar al producto natural, pero con la diferencia de que las tuplas del operando izquierdo que no tienen coincidencia en los atributos comunes aparecen también en el resultado con valores nulos para los atributos que corresponden al operando derecho. Por ejemplo, el producto exterior izquierdo $P \bowtie_{LO} Q$ de las relaciones P y Q que aparecen en la Figura 4 es

A	B	C	D
a_1	b_1	c_1	d_1
a_3	b_3	c_3	d_3
a_2	b_2	c_2	null
a_4	b_4	c_4	null
a_5	b_6	c_7	null

3.8. Producto Exterior Derecho (*Right Outer Join*)

El producto exterior derecho (*right outer join*) es similar al producto natural, pero con la diferencia de que las tuplas del operando derecho que no tienen coincidencia en los atributos comunes aparecen también en el resultado con valores nulos para los atributos que corresponden al operando izquierdo. Por ejemplo, el producto exterior derecho $P \bowtie_{RO} Q$ de las relaciones P y Q que aparecen en la Figura 4 es

A	B	C	D
a_1	b_1	c_1	d_1
a_3	b_3	c_3	d_3
a_5	b_5	null	d_5

3.9. Producto Teta

El producto teta es una operación que permite combinar la selección y el producto cartesiano en una sola operación. El producto teta de dos relaciones R y S queda definido como:

$$R \bowtie_{\Theta} S = \sigma_{\Theta}(R \times S)$$

Ejemplo: Considérense las relaciones U y V de la Figura 3, entonces el producto teta de U y V para la condición $A < D$ es

A	U.B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

3.10. Renombramiento

El operador de renombramiento permite asignar o cambiar el nombre de una relación y cambiar el nombre de sus atributos. La operación $\rho_{S(A_1,A_2,...,A_n)}(R)$ tiene exactamente las mismas tuplas que R, pero ahora el nombre de la relación es S y los de sus atributos son $A_1, A_2, ..., A_n$.

Ejemplo: Supóngase que en el ejemplo de la Sección 3.4 no se deseara identificar a las dos versiones del atributo B como $R_1.B$ y $R_2.B$, sino dejar el nombre B para el atributo de R_1 y utilizar el nombre X para el atributo de R_2. Entonces pueden renombrarse los atributos de R_2 de manera tal que el primero reciba el nombre X con la operación $\rho_{R_2(X,C,D)}(R_2)$. De esta manera, cuando se realiza el producto cartesiano $R_1 \times \rho_{R_2(X,C,D)}(R_2)$ el resultado es la relación $R_1 \times R_2$ de la Figura 2, con la excepción de que los atributos reciben los nombres A, B, X, C, y D. Esta relación se muestra en la Figura 5.

A	B
1	2
3	4

Relación R_1

B	C	D
2	5	6
4	7	8
9	10	11

Relación R_2

A	B	X	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Producto Cartesiano $R_1 \times \rho_{(X,C,D)}(R_2)$

Figura 5: Dos relaciones y su producto cartesiano

Alternativamente se pudo haber tomado el producto sin renombramiento, tal y como se hizo en la Sección 2 y luego renombrar el resultado. La expresión $\rho_{R_3(A,B,X,C,D)}(R_1 \times R_2)$ produce la misma relación de la Figura 5, aunque ahora la relación tiene el nombre R_3 mientras que la relación de la Figura 5 no tiene nombre.

4. Dependencias Funcionales[3, 4, 5]

Sea $\{A_1, A_2, ..., A_n\}$ un conjunto de atributos de una relación *R* tales que si los valores de dos tuplas de *R* coinciden en los atributos $A_1, A_2, ..., A_n$ entonces también coinciden en otro conjunto de atributos $\{B_1, B_2, ..., B_m\}$. Esto se escribe con la notación

$$A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$$

Como puede verse, el concepto de dependencia funcional puede verse como una generalización de los conceptos de clave y superclave. Sea κ un conjunto de atributos que constituye una superclave de una relación *R* y sea $\eta = cab(R)$ su cabecera; es decir, η es el conjunto de todos los atributos de *R*. Entonces se cumple que

$$\kappa \rightarrow \eta$$

En general, si se tiene un conjunto de dependencias funcionales, éstas pueden implicar que existen otras. Por ejemplo, si se nos dice que una relación *R* con los atributos *A*, *B* y *C*, satisfacen las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$, entonces podemos deducir que *R* también satisface la dependencia $A \rightarrow C$. Para demostrar esto consideremos las dos tuplas (a, b_1, c_1) y (a, b_2, c_2) de *R* que coinciden en el atributo *A*. Puesto que *R* satisface $A \rightarrow B$, b_1 tiene que ser igual a b_2 y podemos

decir que las tuplas son realmente (a, b, c_1) y (a, b, c_2). Similarmente, puesto que *R* satisface $B \rightarrow C$, y las tuplas coinciden en *B*, también coinciden en *C*. Con esto demostramos que cualesquiera dos tuplas de *R* que coincidan en *A* también coinciden en *B* y por lo tanto se satisface la dependencia funcional $A \rightarrow C$.

Las dependencias funcionales frecuentemente pueden ser presentadas de diferentes maneras sin cambiar el conjunto de instancias legales de la relación; en este caso decimos que los conjuntos de dependencias son equivalentes.

4.1. Cerradura de un Conjunto de Dependencias Funcionales

Sea *F* un conjunto de dependencias funcionales. Se denomina *cerradura* de *F* al conjunto de todas las dependencias funcionales implicadas por *F*. La cerradura de un conjunto de dependencias funcionales *F* se denota como F^+.

Se dice que un conjunto de dependencias funcionales *G* cubre o genera otro conjunto de dependencias funcionales *F* si *F* puede ser derivado a partir de *G*. En otras palabras, *G* es un generador o una cobertura de *F* si $F^+ \subseteq G^+$. Cuando un generador *G* no tiene subconjuntos propios que también sean generadores de *F*, se dice que *G* es una base de *F*. Un conjunto de dependencias funcionales puede tener múltiples generadores y bases.

Se dice que dos conjuntos de dependencias *F* y *G* son equivalentes si *F* genera a *G* y *G* genera a *F*.

4.2. Dependencias Triviales

La dependencia funcional $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ es:

- *Trivial*: si el conjunto de todos los atributos B_i son un subconjunto de los atributos A_j.
- *No trivial*: si al menos uno de los B_i no está incluido en el conjunto de los A_j.
- *Completamente no trivial*: si ninguno de los B_i está incluido en el conjunto de los A_j.

Siempre se pueden remover del lado derecho de una dependencia funcional aquellos atributos que aparecen a la izquierda. En otras palabras:

- La dependencia funcional $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ es equivalente a $A_1, A_2, ..., A_n \rightarrow C_1, C_2, ..., C_p$ donde los C_i son todos los B_j que no están incluidos en el conjunto de los A_k.

4.3. Axiomas de Armstrong

Existe un conjunto de reglas de inferencia, también llamadas axiomas de Armstrong, que permiten derivar cualquier dependencia funcional implicada por un conjunto de dependencias dado (es decir, permiten encontrar F^+ para un conjunto *F* de dependencias dado). Estos axiomas son:

- *Reflexividad*: Si $B_1, B_2, ..., B_m \subseteq A_1, A_2, ..., A_n$, entonces $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ (esto es el concepto de dependencias triviales).
- *Amplificación*: Si $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$, entonces $A_1, A_2, ..., A_n, C_1, C_2, ..., C_k \rightarrow B_1, B_2, ..., B_m, C_1, C_2, ..., C_k$ para cualquier conjunto de atributos $C_{1,}, C_2, ..., C_k$.
- *Transitividad*: Si $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ y $B_1, B_2, ..., B_m \rightarrow C_1, C_2, ..., C_k$, entonces $A_1, A_2, ..., A_n \rightarrow C_1, C_2, ..., C_k$.

En ocasiones, no es fácil utilizar los axiomas para encontrar F^+ , así que se utilizan otras reglas derivadas de los mismos:

- *Unión*: Si $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ y $A_1, A_2, ..., A_n \rightarrow C_1, C_2, ..., C_k$, entonces $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m, C_1, C_2, ..., C_k$.
- *División*: Si $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m, C_1, C_2, ..., C_k$, entonces $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ y $A_1, A_2, ..., A_n \rightarrow C_1, C_2, ..., C_k$.
- *Pseudotransitividad*: Si $A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m$ y $B_1, B_2, ..., B_m, C_1, C_2, ..., C_k \rightarrow D_1, D_2, ..., D_p$, entonces $A_1, A_2, ..., A_n, C_1, C_2, ..., C_k \rightarrow D_1, D_2, ..., D_p$.

4.4. Cerradura de un Conjuntos de Atributos

Sea $\alpha = \{A_1, A_2, ..., A_n\}$ un conjunto de atributos y *S* un conjunto de dependencias funcionales. La cerradura de α bajo las dependencias en *S* es el conjunto de todos los atributos determinados funcionalmente por α y se denota $\alpha^+ = \{A_1, A_2, ..., A_n\}^+$.

A continuación se presenta un algoritmo que permite calcular la cerradura de un conjunto de atributos α:

1. Sea β un conjunto de atributos que eventualmente se convertirá en la cerradura. Primero, inicializamos β para que sea α.
2. Ahora, repetidamente buscamos alguna dependencia funcional $B_1, B_2, ..., B_m \rightarrow C$ tal que todos los atributos $B_1, B_2, ..., B_m$ estén incluidos en β y C no. Entonces añadimos C a β.
3. Repetimos el paso 2 tantas veces como sea necesario hasta que no se puedan añadir más atributos a β.
4. El conjunto β después de que no se pueda añadir ningún otro atributo es α^+.

Ejemplo: Considérese una relación cuya cabecera es $\{A, B, C, D, E, F\}$. Suponga que esta relación tiene las dependencias funcionales $AB \rightarrow C$, $BC \rightarrow AD$, $D \rightarrow E$, $CF \rightarrow B$. ¿Cuál es la cerradura de $\{A, B\}$?

- Comenzamos con $\beta = \{A, B\}$
- Observamos que los atributos a la izquierda de $AB \rightarrow C$ están en β, así que añadimos el atributo C y β queda como $\{A, B, C\}$.
- Ahora, el lado izquierdo de $BC \rightarrow AD$ está en β, así que añadimos a β sólo el atributo D, pues A ya está en β. En este momento $\beta = \{A, B, C, D\}$.
- El lado izquierdo de $D \rightarrow E$ está en β, por lo que añadimos el atributo E y nos queda $\beta = \{A, B, C, D, E\}$.
- No es posible hacer más cambios a β. En particular, la dependencia $CF \rightarrow B$ no puede ser usada, porque su lado izquierdo nunca llega a estar contenido en β. Así, $\{A, B\}^+ = \{A, B, C, D, E\}$.

4.5. Cobertura Canónica de un Conjunto de Dependencias Funcionales

Una *cobertura canónica* de un conjunto de dependencias funcionales F es un conjunto equivalente de dependencias funcionales F_c que es una base de F. Tiene las siguientes propiedades:

1. Ninguna dependencia funcional $\alpha \rightarrow \beta$ en F_c contiene atributos superfluos en α. Un atributo superfluo en α es aquel que puede ser removido de α sin cambiar F_c^+. En otras palabras, A es un atributo superfluo en α si $A \in \alpha$ y

   $$(F_c - \{\alpha \rightarrow \beta\}) \cup \{\alpha - A \rightarrow \beta\}$$

   equivale a F_c.

2. Ninguna dependencia funcional $\alpha \rightarrow \beta$ en F_c contiene atributos superfluos en β. Un atributo superfluo en β es aquel que puede ser removido de β sin cambiar F_c^+. En otras palabras, A es un atributo superfluo en β si $A \in \beta$ y

   $$(F_c - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow \beta - A\}$$

   equivale a F_c.

3. El lado izquierdo de cada dependencia funcional en F_c es único. Es decir, no existen dos dependencias funcionales $\alpha_1 \rightarrow \beta_1$ y $\alpha_2 \rightarrow \beta_2$ tales que $\alpha_1 = \alpha_2$.

Para calcular una cobertura canónica F_c para F, se puede utilizar el siguiente procedimiento:

1. Aplicar la regla de unión para reemplazar dependencias de la forma $\alpha_1 \rightarrow \beta_1$ y $\alpha_1 \rightarrow \beta_2$ por $\alpha_1 \rightarrow \beta_1\beta_2$.
2. Evaluar cada una de las dependencias $\alpha \rightarrow \beta$ para determinar si hay atributos superfluos en α.
3. Evaluar cada una de las dependencias $\alpha \rightarrow \beta$ para determinar si hay atributos superfluos en β.
4. Repetir los pasos 1, 2 y 3 hasta que no se puedan realizar más cambios.

Ejemplo. Para la relación R tal que $cab(R) = \{A, B, C\}$ con el conjunto F de dependencias funcionales

$$A \rightarrow BC$$

$$B \rightarrow C$$

$$A \rightarrow B$$

$$AB \rightarrow C$$

el cálculo de F_c es:

- Existen dos dependencias con el mismo lado izquierdo: $A \rightarrow BC$ y $A \rightarrow B$. Aplicamos unión y las reemplazamos por $A \rightarrow BC$.

- Hay un atributo superfluo en $AB \rightarrow C$ ya que $B \rightarrow C$ implica $AB \rightarrow C$.
- Entonces después de la primera pasada nos queda el siguiente conjunto:

$$A \rightarrow BC$$

$$B \rightarrow C$$

- Existe todavía un atributo superfluo en $A \rightarrow BC$ ya que $A \rightarrow B$ y $B \rightarrow C$ implican $A \rightarrow BC$.
- Entonces definitivamente F_c queda como:

$$A \rightarrow B$$

$$B \rightarrow C$$

4.5.1. Determinación de atributos superfluos

Sea F un conjunto de dependencias funcionales tales que $(\alpha \rightarrow \beta) \in F$, sea A un atributo tal que $A \in \alpha$ y sea B un atributo tal que $B \in \beta$.

Para determinar si A es superfluo, calcular $(\alpha - \{A\})^+$. Si $(\alpha - \{A\})^+$ incluye a β, entonces A es superfluo.

Para determinar si B es superfluo, calcular $\alpha^+_{F'}$; esto es, la cerradura de α bajo el conjunto de dependencias F' (no el conjunto original F), siendo $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$. Si $\alpha^+_{F'}$ contiene a B, entonces B es superfluo.

5. Normalización[3, 4, 5]

5.1. Anomalías

Cuando se intenta meter muchos datos en una relación ocurren problemas como la redundancia de la información. Estos problemas se denominan *anomalías*. Las principales anomalías que se presentan en el diseño de una base de datos relacional son:

1. *Redundancia*. La información puede estar repetida innecesariamente en varias tuplas. Un ejemplo de esto son la duración y el color en la relación *Películas* de la Figura 6.
2. *Anomalías de actualización*. Es posible que se cambie información en una tupla pero que se deje la misma información sin cambios en otra. Por ejemplo, si descubrimos que *Star Wars* tenía una duración de 125 minutos, es posible que cambiemos la duración en la primera tupla de la Figura 6 pero no en la segunda o tercera tupla. Por supuesto, se puede alegar que no se debe ser tan descuidado, pero también es posible rediseñar la relación de manera tal que el riesgo de este error no exista.
3. *Anomalías de eliminación*. Si eliminamos un conjunto de valores, podemos perder otra información como un efecto secundario. Por ejemplo, si eliminamos a *Tom Hanks* del conjunto de protagonistas de *Forrest Gump*, entonces esa película quedará sin protagonistas, pero también desaparecerá toda la información relativa a la película.

nombre	*año*	*duración*	*color*	*estudio*	*protagonista*
Star Wars	1977	124	verdadero	Fox	Carrie Fisher
Star Wars	1977	124	verdadero	Fox	Mark Hamill
Star Wars	1977	124	verdadero	Fox	Harrison Ford
Forrest Gump	1994	141	verdadero	Paramount	Tom Hanks
The Godfather	1972	171	verdadero	Paramount	Al Pacino
The Godfather	1972	171	verdadero	Paramount	Marlon Brando

Figura 6: Relación Películas con anomalías

5.2. Descomposición de relaciones

La manera aceptada de eliminar estas anomalías es *descomponer* relaciones. La descomposición de una relación *R* implica dividir sus atributos para construir cabeceras de dos nuevas relaciones. La manera de introducir las tuplas de *R* en las relaciones resultantes de la descomposición es utilizar la operación de proyección del álgebra relacional.

Una descomposición de una relación *R* es un conjunto de relaciones $\{R_1, R_2, ..., R_k\}$ tales que:

1. $cab(R) = \cup_{i=1}^{k} cab(R_i)$
2. Las tuplas en la relación R_i son las proyecciones de todas las tuplas de *R* en los atributos de R_i. Esto es, $R_i = \Pi_{cab(R_i)}(R)$

5.3. Descomposición sin pérdida

Dadas una relación R y una descomposición $\{R_1, R_2, ..., R_k\}$ de R. Siempre se cumple que:

$$R \subseteq \bowtie_{i=1}^{k} R_i$$

Se dice que una descomposición $\{R_1, R_2, ..., R_k\}$ de una relación R es sin pérdida si se cumple que:

$$R = \bowtie_{i=1}^{k} R_i$$

Es esencial que una descomposición sea sin pérdida, ya que de otro modo es imposible garantizar que se puede reconstruir la información de la relación original a partir de las relaciones resultantes de la descomposición. En tal sentido es necesario tener un criterio para determinar cuándo una descomposición produce pérdida.

Sea R una relación y F un conjunto de dependencias funcionales en R. R_1 y R_2 formarán una descomposición de R. Esta descomposición es una descomposición sin pérdida si al menos una de las siguientes dependencias funcionales está en F^+ :

1. $cab(R_1) \cap cab(R_2) \rightarrow cab(R_1)$
2. $cab(R_1) \cap cab(R_2) \rightarrow cab(R_2)$

5.4. Conservación de dependencias

Al diseñar una base de datos relacional se debe considerar un objetivo adicional: la conservación de dependencias. El sistema debe garantizar que la actualización de la base de datos no producirá una relación ilegal; es decir, una que no satisfaga las dependencias funcionales establecidas.

Sea F un conjunto de dependencias funcionales sobre una relación R. Sea $\{R_1, R_2, ..., R_k\}$ una descomposición de R. Sea F_i el conjunto de todas las dependencias funcionales en F^+ que incluyen únicamente atributos de R_i. Sea F' la unión de todas las restricciones, $\cup_{i=1}^{k} F_i$. F' es un conjunto de dependencias funcionales en R, aunque en general $F' \neq F$. Sin embargo, puede ser que $F'^+ = F^+$. Si esto es cierto, entonces F' implica lógicamente todas las dependencias de F y si se comprueba que se cumple F', se habrá verificado que se cumple F. Se dice que una descomposición que tiene esta propiedad *conserva las dependencias*.

A continuación se presenta un algoritmo que permite determinar si una descomposición $\{R_1, R_2, ..., R_k\}$ conserva las dependencias:

1. Calcular F^+.
2. Asignar a F_i el conjunto de dependencias funcionales en F^+ que contienen únicamente atributos que aparecen en R_i. $i = 1, 2, ..., k$.
3. $F' = \bigcup_{i=1}^{k} F_i$
4. Calcular F'^+
5. Si $F'^+ = F^+$ entonces la descomposición conserva las dependencias. De otro modo la descomposición no conserva las dependencias.

En la práctica, es más fácil utilizar las reglas de la Sección 4.3 para determinar si F' genera a F, ya que el cálculo de F^+ y F'^+es una operación de orden exponencial.

5.5. Forma normal Boyce-Codd

El proceso de descomposición de una relación permite eliminar redundancia. Pueden definirse varias formas normales utilizando las dependencias funcionales. La distintas formas normales representan los grados de eliminación de redundancia que pueden lograrse.

Una de las formas normales que más conviene obtener es la forma normal Boyce-Codd (BCNF). Una relación R está en BCNF si y sólo si para cualquier dependencia funcional no trivial $\alpha \rightarrow \beta$, α es una superclave de R.

5.5.1. Estrategia de descomposición a BCNF

Cualquier relación R se puede descomponer en una colección de relaciones en BCNF. La estrategia de descomposición a utilizar consiste en buscar una dependencia funcional completamente no trivial $\alpha \rightarrow \beta$ que viole la condición de BCNF, es decir, α, no es una superclave y dividir la relación en dos relaciones tales que:

- La cabecera de una relación debe estar formada por todos los atributos involucrados en la dependencia funcional que viola BCNF.
- La cabecera de la otra relación debe estar formada por todos los atributos de la relación que no estén involucrados en la dependencia y los atributos del lado izquierdo de la dependencia (es decir, todos los atributos de la relación excepto los incluidos en β).

Al aplicar esta estrategia se recomienda que para todas las dependencias $\alpha \rightarrow \beta$ que violen la condición BCNF, β sea $\alpha^+ - \alpha$, es decir, una forma maximal totalmente no trivial, de manera tal que se reduzca el número de descomposiciones a realizar.

Nótese que esta estrategia siempre produce una descomposición sin pérdida, ya que al sustituir R por las relaciones con cabeceras $cab(R) - \beta$ y $\alpha \cup \beta$ se cumple la condición de la Sección 5.3:

$$(cab(R) - \beta) \cap (\alpha \cup \beta) \rightarrow \alpha \cup \beta$$

ya que

$$(cab(R) - \beta) \cap (\alpha \cup \beta) = \alpha$$

y se sabe de antemano que $\alpha \rightarrow \beta$.

Al aplicar esta estrategia hay que verificar que las dependencias funcionales se conserven, ya que esto no está garantizado.

Por ejemplo, supóngase una relación con los atributos `nombre_película`, `cine`, `ciudad` y las siguientes dependencias funcionales

`cine`→`ciudad`

`nombre_película ciudad` →`cine`

Evidentemente `cine`→`ciudad` viola BCNF pues cine no es una superclave[2].

La segunda dependencia no viola la condición de BCNF ya que el conjunto de atributos {`nombre_película`, `ciudad`} determina todos los atributos de la relación y por lo tanto constituye una superclave. Esta dependencia funcional podría surgir de una política de una compañía administradora de cines que consista en proyectar una película determinada en sólo un cine de la ciudad.

La descomposición que indica la estrategia BCNF son las relaciones con cabeceras {`cine`, `ciudad`} y {`cine`, `nombre_película`}, pero claramente aquí se deja de cumplir la dependencia `nombre_película ciudad`→`cine`.

5.6. Tercera forma normal

En aquellos casos en que la normalización BCNF no permite la conservación de las dependencias funcionales, se abandona la BCNF y se acepta una forma normal más débil llamada *tercera forma normal* (3NF). Siempre es posible encontrar una descomposición sin pérdida que conserve las dependencias funcionales y que esté en 3NF. Una relación R está en 3NF si para cada dependencia totalmente no trivial $\alpha \rightarrow \beta$ se cumple al menos una de las siguientes condiciones:

1. α es una superclave.
2. β es parte de una clave.

La definición de 3NF permite ciertas dependencias que no se permiten en BCNF. Una dependencia $\alpha \rightarrow \beta$ que satisfaga únicamente la segunda condición de la definición de 3NF no se permite en BCNF. Estas dependencias se llaman *dependencias transitivas*.

Nótese que toda relación en BCNF está también en 3NF.

Para obtener una descomposición 3NF de una relación R puede aplicarse el siguiente algoritmo:

[2]Se supone que en un cine se pueden proyectar varias películas (e.g. un cine tipo multiplex)

```
Calcular cobertura canónica F_c del conjunto de dependencias funcionales en R
i=0
Para cada dependencia funcional α → β en F_c
{
    i=i+1
    cab(R_i) = α ∪ β
}
Si (no hay claves de R en ninguna R_j, 1 ≤ j ≤ i)
{
    i++
    cab(R_i) =una clave de R
}
```

La desventaja de 3NF sobre BCNF es que al permitirse dependencias transitivas en las relaciones ocurre algo de redundancia, inexistente en BCNF. En el ejemplo de la Sección 5.5.1 la relación no estaba en BCNF, pero sí en 3NF, ya que `ciudad` forma parte de una clave ({`ciudad`, `nombre_película`} es una clave). En este caso puede notarse que cada tupla va a almacenar redundantemente la ciudad asociada a un cine.

Si desea eliminar la redundancia permitida por 3NF y utilizar BCNF entonces hay dos posibilidades:

1. Se arriesga la integridad de la información en la base de datos.
2. Se introduce una lógica en el DBMS más allá de las restricciones de clave foránea que permita asegurar que las dependencias se satisfagan, en este caso, asegurarse que para una ciudad en la relación de cabecera {`cine`, `ciudad`} haya sólo una película en la relación {`cine`, `nombre_película`}, lo cual podría reducir drásticamente el rendimiento del sistema.

La primera opción es inaceptable y la segunda es poco atractiva. Cuando se presenta un caso como este, el grado limitado de redundancia que producen las dependencias transitivas es el menor de los males. Por lo tanto generalmente se escoge la 3NF.

Como resumen, puede decirse que el objetivo de un diseño de base de datos es:

- BCNF.
- Join sin pérdida.
- Conservación de dependencias.

Si no se puede lograr esto, se acepta:

- 3NF
- Join sin pérdida.
- Conservación de dependencias.

6. Doce Reglas de Codd[2]

1. *Regla de la Información*: Toda la información de una base de datos relacional se representa explícitamente a nivel lógico de una única manera: mediante valores en relaciones.

2. *Regla de acceso garantizado*: Todos y cada uno de los datos (valores atómicos) de una base de datos relacional deben ser accesibles lógicamente mediante una combinación de nombre de relación, valor de clave primaria y nombre de atributo.

3. *Tratamiento sistemático de valores nulo*s: Un DBMS relacional debe soportar valores nulos para representar información desconocida o inaplicable de una manera sistemática, independientemente del tipo de dato.

4. *Catálogo dinámico en línea basado en el modelo relacional*: La descripción de la base de datos está representada a nivel lógico de la misma manera en la que se representan los datos ordinarios, de manera tal que los usuarios autorizados puedan consultar el catálogo utilizando el mismo lenguaje relacional utilizado para consultar datos ordinarios.

5. *Regla de sublenguaje de datos*: Un sistema relacional debe proveer al menos un lenguaje cuyas sentencias puedan expresarse, mediante una sintaxis bien definida, como cadenas de caracteres y que soporte las siguientes características:

   a) Definición de datos
   b) Definición de vistas
   c) Manipulación de datos (interactiva y programática)
   d) Restricciones de integridad
   e) Autorización
   f) Manejo de transacciones (begin, commit, rollback)

6. *Regla de actualización de vistas*: todas las vistas que son teóricamente actualizables (modificables) deben ser actualizables por el sistema.

7. *Inserción, eliminación y modificación de alto nivel*: La capacidad de manipular una relación como un operando simple aplica no sólo a las consultas, sino también a la inserción , modificación y eliminación de datos. Esto significa que debe ser posible insertar, modificar y eliminar conjuntos de tuplas en una sola operación.

8. *Independencia física de los datos*: Los programas de aplicación y las actividades de terminal deben permanecer lógicamente inalteradas cuando se realizan cambios en la representación de almacenamiento de los datos. Básicamente significa que los usuarios y los programas de aplicación no dependen de la estructura física de la base de datos.

9. *Independencia lógica de los datos*: Los programas de aplicación y las actividades de terminal deben permanecer lógicamente inalteradas cuando se realizan cambios en la estructura de las relaciones base que permiten la no alteración. Por ejemplo, los programas de aplicación no deben verse afectados si se agregan columnas a una tabla.

10. *Independencia de integridad*: Las restricciones de integridad específicas para una base de datos relacional particular deben poderse definir en un sublenguaje de datos relacionales y deben almacenarse en el catálogo, no en los programas de aplicación.

11. *Independencia de distribución*: Un DBMS relacional tiene independencia de distribución. Esto significa que los datos de una base de datos relacional pueden estar distribuidos en varias computadoras y esto debe ser transparente para el usuario.

12. *Regla de no subversión*: Si un lenguaje relacional tiene un lenguaje de bajo nivel (a nivel de registros o tuplas), ese lenguaje no puede utilizarse para subvertir o saltar las restricciones de integridad o seguridad definidas en el lenguaje relacional de alto nivel (a nivel de relaciones).

Referencias

[1] Date, C.J. Introducción a los Sistemas de Bases de Datos. Volumen 1. Quinta Edición. Addison-Wesley Iberoamericana, 1993.

[2] Bush, B. Short Course on Relational Databases. Los Alamos National Laboratory, 1994.

[3] Korth, H.F. Fundamentos de Bases de Datos. McGraw-Hill, 1988.

[4] Ullman, J. y Widom, J. A First Course in Database Systems. Prentice Hall, 1997.

[5] Osmar Rachid Zaïane. Database Systems and Structures. 1998.Simon Fraser University . Canadá.