**DISEÑO DE BBDD RELACIONALES**

# Conceptos

**Relaciones**: conjunto de tuplas

**Tupla**: conjunto de atributos o valores agrupados

**Atributo**:

**Llave:** todo determinante de una relación es una llave

**Super llave**: corresponde a aquella llave cuyo cierre es igual a la relación. Es decir, aquella llave que puede determinar todos los atributos de una relación.

**Llave primaria**: Es aquella llave candidata escogida para fungir como llave en la relación. Toda llave primaria es a la vez una superllave: todos sus atributos son únicos y no pueden tener atributos nulos.

**Llave alterna:** aquellas llaves candidatas que no fueron elegidas para ser llaves primarias.

**Llave foránea**:

**Llave candidata**: son aquellas superllaves que pueden fungir como llave primaria en la relación y que además no posean subconjuntos propios dentro de ella que también sean una superllave. Si tengo dos super llaves a y b, donde a es subconjunto propio b, eso implica que, aunque ambas sean superllaves sólo “a” es llave candidata.

**Subconjunto propio:** conjunto donde la totalidad de sus elementos forma parte de otro conjunto más grande.

**Atributo primordial**: En un esquema de relación, es un atributo que no tiene flechas entrantes, es un que no puede ser determinado mediante otro atributo. Por lo tanto, un atributo primordial siempre ha de formar parte de una superllave.

**Cierre de una llave**: El cierre de una llave es igual al conjunto de sus atributos determinados, es decir, todos los atributos que dependen de esa llave.

Dadas:

a = un conjunto cualquiera de atributos de R

F = conjunto de dependencias funcionales de R

¿Cuál es el mayor conjunto de atributos de R que podemos encontrar usando F?

Ese conjunto se denomina “cierre de a bajo F” o “cierre de a” y se denota “a+“

**Alfa**: atributo o conjunto de atributos determinantes

**Beta**: Atributo o conjunto de atributos dependientes.

**R**: relación

**Esquema relacional**: R (A, B, C, D):

**Dependencia funcional:** Existe una relación funcional si dado un atributo o conjunto de ellos puedo encontrar otro atributo o conjunto de ellos.

**Dependencia funcional trivial**: Donde los atributos dependientes son un subconjunto de los determinantes

**Dependencia funcional no trivial**: Donde los atributos dependientes no son un subconjunto de los determinantes, aunque tengan elementos en común.

**Dependencia funcional no trivial completa**: Donde la intersección de ambos conjuntos es el conjunto vacío, es decir, no hay elementos en común entre alfa y beta

**Dependencia funcional aumentada**:

a,b, y son permutaciones de R

Si

a -> b

Entonces

ay -> by

**Dependencia funcional transitiva**:

Si a -> b y b -> c

entonces

a -> c

**Unión natural**

**Vista teóricamente actualizable:** aquella vista que puede actualizar los datos subyacentes desde la misma vista, ej: cuando se pueden modificar datos sobre la vista de la query

**Vista teóricamente no actualizable:** aquella vista que no puede actualizar datos subyacentes, por ejemplo, cuando tenemos una query con datos agregados, no se pueden actualizar los datos subyacentes porque el RDBMS no sabría especificamente qué datos actualizar.

**Entidad asociativa:** entidad que sirve para asociar otras entidades, generalmente tienen más de una llave foránea, al menos una por cada entidad asociada. Se refiere a aquellas tablas que sirven para formar relaciones de muchos a muchos.

**Entidad débil:** entidad que debe su existencia a otra entidad, por ejemplo, una entidad “detalle de facturas” debe su existencia a la entidad “facturas”.

**Entidad supertype/subtype:** Son entidades que comparten los mismos atributos sin embargo las entidades subtype representan un subconjunto de las supertype. Por ejemplo, una entidad EMPLEADOS puede tener entidades subtype EMPLEADOS IT, EMPLEADOS HR y a su vez compartir los mismos atributos con la entidad padre.

**Grupo repetitivo**:

Atributos que tienen el mismo dominio presente en una misma relación. Desde un punto de vista tabular, son campos de un mismo grupo equivalentes entre sí, sin ninguna distinción entre ellos. Ej: Dado un grupo de campos DIRECCION1, DIRECCION2, DIRECCION3. Representan un grupo repetitivo, porque los campos están destinados a almacenar información equivalente y a priori no se puede saber cuántos campos son necesarios para satisfacer los requerimientos de almacenar información relativa a las Direcciones*.* Causan varios problemas:

* Imponen un límite a la cantidad de atributos que pueden registrarse, si hay un grupo repetitivo que consta de 3 campos, para una misma tupla pueden registrarse sólo esos tres campos como máximo.
* Complica la búsqueda de información: si quisiéramos buscar información sobre el dominio de un grupo repetitivo tuviéramos que hacer consultas sobre todos los campos que componen dicho grupo, por ejemplo, si tuviéramos el grupo repetitivo curso1, curso2, y curso3 y quisiéramos buscar información, tendríamos que aplicar consultas sobre esos tres campos.
* Desperdicio de espacio reservado: Si hay tuplas que no alcanzan a llenar la información de todos los atributos del grupo repetitivo se desperdicia el espacio de aquellos atributos con valor nulo.

**Anomalías de diseño**

* Anomalía de inserción: en una tupla con atributos no relacionados correctamente, si se desea registrar un atributo, no se puede hacer sin insertar también los otros atributos no relacionados.
* Anomalía de actualización: en una relación con redundancia de datos, si se actualiza un valor, se debe actualizar varias veces en todas las ocurrencias, si no se hace correctamente, los datos se vuelven inconsistentes.
* Anomalía de eliminación: en una tupla con atributos no relacionados correctamente, si se desea eliminar un registro, se debe eliminar la tupla completa con la consecuente eliminación no intencionada de datos.

**Notación de chen**

Rectángulo: entidad.

Rectángulo con otro adentro: entidad débil (tienen una sóla llave foránea).

Rectángulo con rombo dentro: entidad asociativa (tienen dos o más llaves foráneas, para representar relaciones de muchos a muchos).

Ovalo con nombre dentro: Atributo sencillo, se subraya el atributo llave con línea completa y la llave parcial con linea punteada

Ovalo con otro dentro: atributo multivaluado

Óvalo delineado con línea punteada: Atributo derivado.

Ovalo con otros óvalos vinculados: atributo compuesto

Rombo: relación fuerte: existencia de la entidad no depende que otra entidad también exista

Rombo con otro dentro: relación débil, la existencia de la entidad depende de la existencia de la entidad padre.

Línea de relación completa: relación obligatoria.

Línea de relación punteada: relación opcional.

Línea doble entre entidad y rombo de relación: relación recursiva, donde la entidad se relaciona consigo misma.

Cardinalidad: 1, N, M.

Línea doble de relación muy junta: participación total: Chen la define cuando todas las tuplas de una entidad deben estar relacionadas con tuplas de otra entidad.

Línea de relación normal: participación parcial

## Cierre canónico

También llamado cierre irreductible o base canónica, es un conjunto formado por la menor cantidad de atributos que sigue manteniendo la misma funcionalidad que las mismas dependencias funcionales originales y ese cierre es el que vamos a usar como dependencias funcionales de nuestra relación

Proceso mediante el cual, tomando una relación y su conjunto de dependencias funcionales, se elimina la redundancia entre dependencias. Este proceso se aplica cuando se tienen relaciones muy grandes con muchas dependencias funcionales las cuales pueden ser redundantes entre sí.

Una dependencia con un determinante compuesto no puede dividirse en relaciones más pequeñas, sin embargo si se puede dividir los términos dependientes de una dependencia funcional.

ab -> c esta dependencia funcional no se puede dividir en las siguientes porque el determinante compuesto no puede dividirse.

~~a -> c~~

~~b > c~~

a -> by esta relación si puede dividirse porque se pueden dividir los téminos dependientes.

a -> b

a -> y

Ejemplo de un cierre canónico:

Redundancia en los atributos dependientes

Tenemos la siguiente relación y sus dependencias funcionales

R(A,B,C,D)

B -> A

AD -> BC

C -> ABD

Dividimos las dependencias en dependencias menores:

B -> A

AD -> B

AD -> C

C -> A

C -> B

C -> D

Calculamos el cierre del determinante de una dependencia funcional cualquiera, esto significa calcular el cierre tomando en cuenta no sólo la dependencia funcional con la que estamos trabajando sino también tomar en cuenta TODAS las dependencias funcionales que tenemos listadas:

B+ = BA

Luego calculamos el cierre nuevamente tomando en cuenta todas las dependencias funcionales, pero sin tomar en cuenta la existencia de la dependencia funcional con la que estamos trabajando, anulamos B -> A

B+ = B

Podemos observar que hubo una pérdida de información por lo tanto la dependencia funcional B -> A es una dependencia funcional esencial y no debe ser eliminada.

Repetimos el proceso con la siguiente dependencia funcional:

AD -> B

AD+ = ADBC

Calculamos el cierre nuevamente sin tomar en cuenta la dependencia funcional AD -> B

AD+ = ADCB

Como ambos cierres son iguales, es decir contienen los mismos atributos, decimos que la dependencia funcional AD -> es no esencial y puede ser eliminada.

Este proceso se repite con todas las dependencias funcionales hasta que obtenemos sólo aquellas dependencias que son esenciales para poder derivar información sin pérdida de la misma.

Una vez finalizado el proceso tenemos una lista depurada con aquellas dependencias funcionales esenciales:

B -> A

AD -> C

C -> B

C -> D

Volvemos a unir aquellas dependencias con determinantes iguales:

B -> A

AD -> C

C -> BD

Este nuevo conjunto de dependencias mantiene la misma funcionalidad que el original, sin embargo, no es único, puede haber otros productos de aplicar el mismo proceso, pero variar el orden en que procesamos las dependencias funcionales, sin embargo, todos los conjuntos producto de este proceso mantendrán la misma funcionalidad.

Redundancia en los atributos determinantes

Posteriormente debemos examinar si existe redundancia en los determinantes, ya que ya eliminamos la redundancia en los atributos dependientes.

Para ello tomamos las dependencias que tengan atributo determinante compuesto

AD -> C

Calculamos el cierre de cada uno de sus determinantes por separado, quitando uno de ellos en cada iteración, además al realizar esta operación no tomamos en cuenta la dependencia funcional AD -> C para obtener el cierre

(AD-D) + = A+ = A

(AD-A) = D+ = D

Vemos que el atributo dependiente C no constituye un subconjunto del cierre de A+ o de D+ por lo tanto no hay redundancia en los determinantes de la dependencia funcional AD -> C y debe permanecer así.

Cuando se realiza esta operación: descomponer la llave compuesta y calcular el cierre de los determinantes restando uno de ellos en cada iteración. Se hace para responder la pregunta ¿puedo obtener los mismos atributos si quito uno de los determinantes?

Para calcular este cierre, como se mencionó anteriormente, no se toma en cuenta la DF original y se calcula el cierre tomando en cuenta el resto de DF y la DF temporal que hemos creado para tal fin. Acá una [pregunta y respuesta](https://www.udemy.com/course/diseno-de-bases-de-datos-relacionales/learn/lecture/8255720#questions/16905648) que aclara este concepto.

# Normalización

Proceso secuencial, cíclico y repetitivo cuya finalidad es proteger la integridad de los datos, eliminar la redundancia y eliminar las anomalías de diseño. En este proceso de aplican las **formas normales:** normas bien definidas diseñadas para ejecutar el proceso de normalización.

## Características del modelo relacional

Hay algunas características del modelo relacional que algunos proveedores de información las incluyen como parte de la primera forma normal, sin embargo, son características impuestas por el modelo relacional y no son elementos de la primera forma normal. Estas carácterísticas deben presentarse antes de iniciar el proceso de normalización:

1. Repetición de tuplas: No deben existir tuplas repetidas
2. El orden de las tuplas no debe importar
3. Debe existir una llave primaria
4. Los atributos deben ser atómicos: la atomicidad se determina basado en las reglas del negocio y la precisión requerida. Un atributo no-atómico / compuesto es un campo donde o bien pueden registrarse varios valores del mismo dominio: por ejemplo, en un campo TELÉFONOS podrían registrarse más de un teléfono en cada celda. o bien puede ser un atributo que puede dividirse en atributos más pequeños que lo componen, por ejemplo, el atributo DIRECCIÓN puede descomponerse en CALLE, NUMERACIÓN, DISTRITO. ¿Qué es atómico y qué no lo es?: una apreciación que ayuda a definir cuando se debe atomizar:

*In the context of the problem I'm trying to solve, does it ever make a sense to access any part of the value, or*  *I always access the whole value?*

## Primera forma normal

Forma de los grupos repetitivos

En una relación pueden existir [grupos repetitivos](https://docs.google.com/document/d/1Mzgc5Rc4tlyALZBLZduc9QX3xKmGx2MfWD8ts6S54zc/edit#bookmark=kix.aqy9jq34az2l), eliminando estos grupos se consigue aplicar la primera forma normal

¿Cómo se detectan?

Revisa el modelo lógico y observa cada una de las entidades: observa cada uno de los atributos y revisa que no haya ningún atributo con el mismo dominio dentro de la entidad.

¿Cómo se soluciona?

Se detecta la relación que está en violación de la forma normal y se mueve a una nueva relación.

## Segunda forma normal

Forma de la dependencia funcional completa. Se cumple sí y solo sí la relación:

1. Está en primera forma normal
2. No existen dependencias funcionales parciales, es decir, todas las dependencias funcionales son completas.

En relaciones donde existe una llave primaria compuesta por 2 o más atributos puede existir la posibilidad que existan dependencias parciales: algunos atributos dependen únicamente de una parte de la llave primaria, de alguno de sus atributos y no de todos. Eso significa que, si una relación posee un único atributo como llave primaria, y todo el resto de atributos dependen de éste, ya la relación se encuentra en segunda forma normal.

¿Cómo se detectan?

Revisa cada una de las entidades y su llave primaria. Sólo para aquellos casos donde exista una llave primaria compuesta se constata que no existan dependencias parciales.

¿Cómo se soluciona?

Se detecta la relación que está en violación de la forma normal y se mueve a una nueva relación.

## Tercera forma normal

Forma de la dependencia funcional transitiva. Ésta dependencia funcional se cumple si y solo sí la relación:

1. Está en segunda forma normal
2. Ningún atributo no primo depende funcionalmente de otro atributo no primo.

El problema de la existencia de las dependencias transitivas en una relación radica en que, a diferencia de la llave primaria, los atributos no primos pueden repetirse y también pueden ser nulos, por lo que la dependencia por transitividad no garantiza que siempre se pueda encontrar la información.

¿Cómo se detectan?

Revisa cada uno de los atributos de las entidades y analiza si dependen única y exclusivamente de la llave primaria y no de otro atributo no primo.

¿Cómo se soluciona?

Se detecta la relación que está en violación de la forma normal y se mueve a una nueva relación. además se elimina el atributo no primo de la relación original

Hay veces que en una relación con pocas dependencias puede observarse claramente una transitividad entre dependencias, sin embargo antes de eliminar una dependencia es conveniente calcular el [cierre canónico](https://docs.google.com/document/d/1Mzgc5Rc4tlyALZBLZduc9QX3xKmGx2MfWD8ts6S54zc/edit#bookmark=kix.ywd38v9nkph4) del conjunto de dependencias para saber cuales son esenciales y cuáles no, de esa manera al eliminar las dependencias redundantes podremos observar con mayor claridad la transitividad.

## Normalización superior

En la mayoría de los casos es suficiente con aplicar las tres primeras formas normales, sin embargo hay casos inusuales donde se aplica la llamada “normalización superior”, la cual se detalla a continuación

## Forma normal boyce-codd (NFCB)

En casos muy inusuales donde hay más de una llave candidata en una relación y éstas llaves candidatas tienen atributos en común es donde puede aplicarse esta forma normal. Esta forma normal se cumple si la relación:

1. Está en tercera forma normal
2. Para cada [dependencia funcional no trivia](https://docs.google.com/document/d/1Mzgc5Rc4tlyALZBLZduc9QX3xKmGx2MfWD8ts6S54zc/edit#bookmark=kix.kadjz9yu6el)l que exista, el determinante es una superllave: esto implica que si existe alguna dependencia funcional que no sea una super llave dentro de la relación, está dependencia violaría la NFCB.

Las dependencias funcionales aumentadas, sirven para constituir una superllave cuando no existe una, sin embargo ésta no elimina las dependencias funcionales subyacentes. Por lo que a éstas dependencias funcionales subyacentes se les evalúa por separado verificando que se cumpla la NFCB

Si en la relación existe una dependencia funcional no trivial que no es una superllave, eso implica que esa relación viola la forma normal boyce-codd. Para que se cumplan todas las dependencias funcionales de la relación deben tener como determinante una superllave.

¿Cómo se soluciona?

Se detecta la relación que está en violación de la forma normal y se mueve a una nueva relación, al mismo tiempo se deja en la relación original el determinante que haya quedado y el resto de los atributos.

## Cuarta forma normal

Forma normal de las dependencias multivaluadas, también llamadas dependencias funcionales multiplicadoras de tuplas.

1. Está en tercera forma normal o el Forma normal Boyce y Codd
2. Para cada dependencia multivaluada no trivial que exista, el determinante es una superllave

¿Si una Relación no tiene Dependencia Multivaluadas y ya está en Tercera Forma Normal, es correcto decir que también está en Cuarta Forma Normal?

Sí, por el simple hecho que no hay dependencias multivaluadas.

No me quedó muy claro el concepto de dependencia multivaluada, ver [video](https://www.udemy.com/course/diseno-de-bases-de-datos-relacionales/learn/lecture/8255730#questions/5235602) para profundizar y aclarar el tema. Más o menos lo que me quedó claro del concepto de dependencia multivaluada es que es aquella donde el determinante puede “Multideterminar” un dependiente, que puede asignar valores diferentes dependiendo de la tupla, creando de esa forma multiples tuplas.

Éste comportamiento también se puede visualizar al introducir un nuevo valor que éste hecho crea múltiples tuplas.

Siempre que exista una relación de multidependencia, ésta debe ser dividida en relaciones más pequeñas y el determinante claramente debe ser una superllave.

¿Cómo detectarlo?

Partiendo de la tercera forma normal sólo debe haber dependencias entre los no primos y la llave, no hay dependencias entre no primos, por lo que se analiza la dependencia entre los no primos y la llave y se observa si puede existir más de un valor para la misma llave. Si sólo puede existir un valor para la misma llave la relación está en cuarta forma normal.

Examina atributo por atributo y pregunta ¿puede haber más de una ocurrencia para la misma llave?, puede darse el caso que sí, pero si el propósito del sistema es almacenar una sola ocurrencia, no hay problema.

## Quinta forma normal

Forma normal de la proyección-unión

1. Está en cuarta forma normal
2. No puede ser descompuesta no aditivamente en relaciones más pequeñas

La Quinta Forma Normal dice que una Relación debe dividirse en las proyecciones más pequeñas posibles siempre y cuando puedan volver a unirse mediante: la unión natural no aditiva de estas proyecciones. Una unión natural no aditiva voy

No me quedó muy claro, ver el video para profundizar.

## Desnormalización

Permitir la redundancia con la finalidad de mejorar el desempeño de la base de datos o para mantener un registro estático de detalles que no deben variar con las actualizaciones que vayan ocurriendo a los registros.

Por ejemplo, un detalle de facturas debe permanecer estático para siempre y si se deriva mediante relaciones entre tablas este detalle, si ocurre alguna modificación en las tablas que originan el detalle, el detalle será modificado, lo cual no puede suceder.

Existen RDBMS mecanismos que permiten la desnormalización:

Vistas indexadas o materializadas

Información pre-procesada: consiste en tomar ciertos datos que se encuentran en relaciones totalmente normalizadas y pegarlos en relaciones menos normalizadas como por ejemplo los cubos OLAP, tablas de data warehouse, tablas de data mining, esquemas de copo de nieve o estrella.

Para proteger la integridad de los datos en estos entornos deben existir reglas estrictas y procesos especiales de inserción, modificación y eliminación para proteger la integridad de los datos.

Generalmente las bases de datos transaccionales (OLTP) están altamente normalizadas y mediante procesos posteriores se desnormalizan los datos para

mejorar el proceso de lectura en nuevas bases de datos de consulta (OLAP/BI).

# Modelado

**Herramientas para diagramar**: Microsoft Visio, Lucid Chart, ERDPlus, AquaData.

ERD PLUS

En el modelado lógico (Relational Schema) en ERD PLUS. La nomenclatura indica lo siguiente:

Cabeza de flecha = 1

Pie de flecha = muchos

PhpmyAdmin

En el modelo de esta aplicación:

Punto chico = 1

Punto grande = muchos

## Modelo conceptual y lógico

**Modelo conceptual:** Es un modelo con entidades y relaciones, representado por un diagrama entidad relación, utilizando la notación de backers preferiblemente.

Proceso:

* Leemos el universo de discurso y enlistamos entidades.
* Creamos un modelo con entidades y relaciones, sin atributos, para mantenerlo simple.
* Pasamos al modelo lógico

**Nombres en inglés**: Nombres cortos ayudan a escribir código. Puede entenderse por cualquier persona en el mundo.

**Prefijos**: Es mejor no incluir prefijos en los nombres de las columnas porque de igual forma es buena práctica incluir el nombre de la tabla en la construcción de queries, en caso que la sea engorroso incluir largos nombres de tablas en cada campo que queramos referenciar, se puede asignar alias a los nombres de las tablas para que sea más fácil referenciarlos en la query.

HYPHENS: Nunca usar Hyphens (-) porque obligaría al uso de backticks en todos los nombres.

CASE: Algunos motores son case-sensitive y algunos no, MySQL respecta el case en los output pero otros no lo hacen, por lo tanto si nombras con CamelCase y migras a otra base de datos que outputea todo en lower-case vas a tener nombres como EsteEsUnNombre convertido en esteesunnombre.

Lo mejor es stick to lowercase para los nombres

ABREVIACIONES: Evita abreviaciones y nombra lo más descriptivo posible para que no haya que estar revisando documentación o el contenido de una columna para entender que contiene.

CONCLUSIÓN: lowercase y underscores: table\_name, field\_name, sin hyphens, sin abreviaciones, sin prefijos y mantener la consistencia across the database.

**Modelo lógico:** Es la consecución del modelo conceptual, adicionando los atributos a cada una de las entidades y las relaciones debidamente identificadas a través de los atributos que mantienen la relación.

* Determinamos los atributos de cada relación
* Determinamos las llaves primarias, deben ser pequeñas en términos de longitud
* Normalizamos.
* Creamos las relaciones observando el modelo entidad relación, si hay relaciones muchos a muchos se crean las tablas de relación.

Número de identificación nacional como clave primaria:

No debe usarse nunca por las siguientes razones:

1. Human supplied data: Es un dato suministrado por un humano, lo cual puede inducir a errores, puede estar incorrecto intencionalmente o no intencionalmente, también puede darse la situación donde el usuario no desee suministrarlo.
2. Sensible data: Es información sensible por lo que debería estar protegido, las leyes pueden influir mucho en cómo debe almacenarse esta información y si es una primary key de una tabla puede complicar mucho las cosas.
3. External entity: Es un valor creado por una entidad externa (gobierno), es improbable que haya errores en su creación pero no es un valor que pueda controlarse internamente.

**Escogencia de la llave primaria y su función como cluster index**

No necesariamente el cluster index de una tabla corresponde con la llave primaria, aunque casi siempre es así, de hecho, también puede existir la posibilidad que no exista índice primario en una tabla. Asumiendo que la llave primaria será a su vez cluster index debemos procurar que:

1. Debe constar de valores secuencialmente crecientes
2. Deben ser lo más pequeños posible

**Modelo entidad relación:** es una representación gráfica de la estructura de la base de datos, puede ser conceptual (más abstracta) o lógica (más detallada). Para ello se utilizan distintas notaciones las cuales se pueden mezclar, sin embargo, existe un aparente consenso en el uso de las notaciones.

Es un proceso cíclico, aquí se expresa una secuencia, sin embargo, algunos pasos pueden volver a realizar se de forma cíclica o iterativa, durante este proceso se realizan nuevas indagatorias, aparecen entidades, atributos o relaciones y vuelve a repetirse el ciclo desde el principio.

1. Leer el universo de discurso y resaltar las entidades y atributos con un color, luego pasarlos a un listado. Primero lo que está explícitamente escrito en el universo del discurso y luego lo que está implícito. Seleccionamos los sustantivos del enunciado para identificar entidades
2. Identificamos las relaciones y su cardinalidad.
   1. Nota: En las relaciones de 1 a 1 la llave foránea puede ir en una entidad u otra, en las relaciones 1 a N la llave foránea se coloca en la entidad hija. Mientras que en las relaciones N a M se crea una entidad asociativa con dos llaves foráneas. Se crean dos relaciones de 1 a N apuntando hacia la entidad nueva.
3. Precisamos el modelo conceptual
4. Realizamos el modelo lógico: para incluir los atributos se debe hacer un análisis muy profundo del sistema actual: qué datos y atributos se llevan actualmente y qué datos se desean llevar con el nuevo sistema.
5. Identificamos atributos y llaves y relaciones
6. Revisamos que el modelo cumpla con las [características del modelo relacional](https://docs.google.com/document/d/1Mzgc5Rc4tlyALZBLZduc9QX3xKmGx2MfWD8ts6S54zc/edit#bookmark=kix.plezgpg1jv8e)
   1. Revisamos la atomicidad de los datos: una vez atomizados vemos la probabilidad de repetición de un atributo y los ordenamos de mayor orden de repetición a menor orden de repetición. Se toman los atributos que más se repitan y se colocan en nuevas tablas con la finalidad de eliminar la redundancia. En los sistemas donde sea muy importante el correcto almacenamiento de un atributo, se suele crear una tabla para cada uno de esos atributos importantes. Atendiendo el caso de la entidad dirección (país, estado, ciudad, cod postal, calle), con sacar el atributo pais y estado a dos tablas diferentes es más que suficiente. En un sistema donde la dirección sea muy importante se crearían nuevas tablas para todos estos atributos.
   2. Se aconseja que las llaves primarias sean pequeñas y evitar las llaves primarias extensas especialmente si se van a utilizar como llave foránea en otras tablas, si se usa como llave primaria un número de 11 dígitos, por ejemplo, puede que sea comprometedor para el rendimiento de la base de datos. Para que la llave primaria sea pequeña se puede crear una llave sustituta.
   3. No se deben crear atributos porque sí y tampoco debe faltar ningún atributo en el diseño
7. Revisar la normalización y revisar todo el modelo hasta que se tiene la seguridad que representa lógica y fidedignamente el sistema que se desea representar.

La diagramación del diseño debe realizarse FUERA del cliente RDBMS. En ocasiones hay confusión ya que los clientes RDBMS poseen herramientas para representar gráficamente las tablas y relaciones, sin embargo esta representación corresponde a la BBDD real y cualquier modificación afecta a la BBDD, por lo tanto se diseña o rediseña fuera del cliente y luego se crea esa representación propiamente en el RDBMS.

## Modelo físico

Corresponde a la implementación formal en un RDBMS.

### Conceptos

**Cambio de terminología:**

Entidades = Tablas

Tuplas = Filas o Registros

Atributos = Columnas

Relaciones = Tablas de relación (relaciones muchos a muchos) o Atributos con igual dominio / llaves foráneas

**Problema singular vs plural:** Problema que se manifiesta al elaborar modelos conceptuales y denominar entidades, por costumbre en el modelo físico queda eliminado, las tablas preferiblemente se nombran en plural a menos que sepamos que esa entidad siempre va a tener un único registro.

**Nomenclatura:** Depende de las reglas de RDBMS y de las políticas de la empresa. Las más usadas son Pascal-Case o Camel-Case. Lo importante es la consistencia, se debe mantener la misma nomenclatura a través de todo el sistema.

**Índices:** Estructura de árbol balanceado (B+ Tree) que utiliza la DB para acceder a la información mediante un algoritmo va recorriendo los nodos del árbol de arriba hacia abajo hasta dar con la llave primaria que necesita encontrar. Si un nodo tiene registros grandes, caben menos datos en cada nodo y por ende se deben hacer más lecturas físicas para encontrar los datos.

**Índice primario (Cluster):** Es el índice principal de la DB

**Índice secundario (Non Cluster):** Son índices que asocian atributos con valores de la llave del índice primario. Ayudan a encontrar atributos que no están cubiertos por el índice primario, sin embargo, tienen el contra que cada vez que se realiza escritura el RDBMS debe hacer mantenimiento a este índice. No se debe abusar de estos índices o comparar rendimiento de lectura y escritura para ver si conviene o no mantener un índice.

La elección de claves primarias pequeñas es importantísima para mejorar el rendimiento de lectura de una DB, si se usa una llave con 11 caracteres de largo vs una de 4 bytes, las diferencias de rendimiento son del 75%.

**Tipos de datos:** Restricción general del dominio.

La modificación de tipos de datos cuando una DB está en producción puede ser, al igual que cualquier corrección en el diseño, algo extremadamente difícil, debe ajustarse al mínimo almacenamiento para el máximo valor posible.

Números

Exactos

Enteros (1,2,4,8 bytes)

Decimales (5, 9, 13, 17 bytes)

Dinero (4,8 bytes)

Lógicos (1 bit)

Aproximados (punto flotante)

Reales (4 bytes)

Flotantes (n). n = bits de almacenamiento para la mantisa (4, 8 bytes)

Fechas

Fecha (3 bytes)

Fecha-hora (8 bytes)

Hora (5 bytes)

Cadenas de caracteres:

Fijos

char(n). n bytes

nchar(n). n \* 2 bytes (unicode)

Variables: se debe seleccionar lo que parezca “razonable”, el mínimo almacenamiento posible para el máximo valor

varchar(n). n + 2 bytes

nvarchar(n). n \*2 + 2 bytes (unicode)

Binarios

Espaciales

XML

Identificadores (UID)

Text: mejor usar VARCHAR(MAX) o NVARCHAR(MAX) si se necesitan más de 4000 caracteres.

Longitud de campos

Reasonable length of fields:

Email = VARCHAR(254)

FirstName = VARCHAR(50)

FirstName = VARCHAR(50)

AddressName = VARCHAR(200)

AddressComuna = VARCHAR(50)

AddressNum = VARCHAR(10)

Llave primaria tabla transaccional**:** El tamaño debería ser lo más grande posible: un entero de 8 bytes por ejemplo que permite 9,223,372,036,854,775,808 transacciones para evitar tener que redimensionar ese campo.

A pesar de esta apreciación, en stackoverflow indican los siguiente:

*W3C's recommendation:*

*If designing a form or database that will accept names from people with a variety of backgrounds, you should ask yourself whether you really need to have separate fields for given name and family name. Bear in mind that names in some cultures can be quite a lot longer than your own. … Avoid limiting the field size for names in your database. In particular, do not assume that a four-character Japanese name in UTF-8 will fit in four bytes – you are likely to actually need 12.*

[*https://www.w3.org/International/questions/qa-personal-names*](https://www.w3.org/International/questions/qa-personal-names) *For database fields, VARCHAR(255) is a safe default choice, unless you can actually come up with a good reason to use something else. For typical web applications, performance won't be a problem. Don't prematurely optimize.*

**Restricciones**

**Restricciones de dominio**: El tipo de dato es un tipo de restricción de dominio que delimita el extremo superior de un dominio, cuán grande puede ser un dato, sin embargo, las restricciones de dominio permiten delimitar aún más el dominio, como:

* Valor mínimo
* Longitud mínima
* Rango de datos válido
* Restricción de un campo basado en otro campo.
* Aplicación de reglas, funciones. Un ejemplo son los dígitos verificadores, si conociéramos el algoritmo que los generales podríamos incluir como una regla de restricción de dominio para que sólo se incluyan en la DB valores válidos con el dígito verificador correspondiente, por ejemplo, un RUT válido y no uno falso.
* Unicidad: no repetición
* Obligatoriedad: no anulación
* Valores por defecto: no es una restricción, pero es una herramienta que permite indicar un valor por defecto para que un valor nunca sea

**Restricciones de relaciones:** ayudan a evitar filas huérfanas: en tablas relacionadas, filas que no están relacionadas con su respectiva tabla padre.

Actualización en cascada: modificaciones en registros padres afectan registros hijos pueden hacerse

Eliminación en cascada: si se elimina un registro padre elimina los registros hijos

Modificación en cascada: idem

Anulación de la llave foránea: Si se elimina un registro padre que no elimine los registros hijos pero que modifique la llave foránea en la tabla hija aplicando un valor NULL o un valor por defecto en los registros hijos.

### Errores típicos:

Hay una serie de errores conocidos y bien documentados que deben evitarse a toda costa para evitar rediseños de la BBDD en producción.

**Registros muy grandes:** si se tiene una tabla con muchos campos, los registros serán muy grandes y cabrán menos registros en cada bloque lo cual involucra mayor cantidad de lecturas físicas, conviene tener registros pequeños. Un approach es tomar una relación y dividirla en dos más pequeñas: una tabla de transacciones con la información que se consulta frecuentemente y otra con información adicional que no se consulta tan a menudo.

**Menos tablas no es una mejor base de datos**

**Pobre indexación:** ante problemas de rendimiento casi siempre la culpa es del índice.

* Faltan índices necesarios para mejorar el rendimiento de lectura
* Sobran índices: empeoran el tiempo de actualización.
* El índice está mal hecho.

**Tamaño de las tablas:** depende de los datos históricos, estos tienen una baja tasa de cambio y una baja tasa de consulta. El tamaño de las tablas depende del tamaño del disco, sin embargo, los manejadores de BBDD les cuesta más lidiar con tablas grandes.

Se deben gestionar los datos históricos: dejar los históricos en una tabla y los recientes en otra tabla.

* Particionamiento vertical: Dejar datos de períodos antiguos en una o más tablas independientes de la tabla principal.
* Particionamiento horizontal: Utilizar una clave para diferenciar distintos tipos de transacciones y almacenarlas en tablas diferentes.

**Documentación y autodocumentación**

En la medida de lo posible lo mejor es autodocumentar: incluir comentarios en los campos de las tablas para especificar detalles y lo que no se pueda autodocumentar, se documenta normalmente. Ejemplo de comentarios:

1. Unidad de medida
2. Descripción de estatus: para estatus abstractos se describe qué significa cada valor.

**No realizar pruebas:**

Pruebas de funcionalidad

Pruebas de esfuerzo: carga y lectura

Pruebas de concurrencia

**No incluir las reglas del negocio directamente en la BBDD siempre que se pueda**

Procedimientos almacenados (Stored Procedures)

Funciones (cálculos)

Disparadores (Triggers)

Vistas

**No considerar dispositivos físicos**

Siempre que se pueda y esté en tus posibilidades debes priorizar los mejores dispositivos disponibles, procesadores y memoria, política de respaldo.

### Implementación modelo físico

Generalmente se hace una traducción del modelo lógico a lenguaje SQL, de ésta traducción se puede encargar el modelador, copiamos la traducción a un archivo de texto y revisamos

* 1. Tipo de campos y longitud de campos (usa buscador)
     1. Claves primarias sustitutas BIGINT UNSIGNED (tabla transaccional y otras con muchos registros), INT UNSIGNED (resto de tablas). TINYINT o SMALLINT aquellos que lo ameriten.
  2. AUTO\_INCREMENT sobres las llaves primarias que así lo requieran, a las llaves foráneas no se les declara esta restricción.
  3. Revisa el diagrama y arréglalo
  4. Restricciones de dominio: más allá de los tipos de datos y su valor máximo,

¿Cómo proceder?

Usamos como herramienta de control el diagrama, si es un diagrama muy complejo y largo enlistamos todos los campos de la BBDD en un Excel y vamos gestionando uno a uno. Como medida rápida se puede ver el diagrama y anotar en una nota, bloc o papel aquellos campos que queremos restringir y que tipo de restricción queremos aplicar, luego la vamos implementando una a una en el script. El uso de GUI no ha sido útil porque cada GUI es diferente, compleja y navegar en ella es difícil. Será mejor crear un buen script SQL con todo lo necesario para implementar, además que se puede replicar la implementación si se usa un script, en cambio mediante GUI se pierde el SQL.

* + - Valor mínimo: aplica a los campos de tipo número. Generalmente se aplica sobre los campos de tipo precio y cantidad para establecer precios y cantidades mínimas.
    - Longitud mínima: aplica al campo de tipo número y string, pensamos la longitud mínima que puede existir. LEN(NombreCampo) > LongitudMínima
    - Longitud fija: LEN(NombreCampo) = LongitudFija
    - Introducción numérica a tipo de dato string
    - Rango de datos válido por valores en caso de datos de tipo numérico o por longitud en caso de valores de tipo string.
  1. Antes de proceder a crear las tablas en el modelo físico, configurar el default charset de la base de datos para que las tablas se creen con ese charset.
  2. Es preferible definir las claves primarias y foráneas, así como otros elementos de las tablas dentro del propio statement CREATE TABLE para intentar reducir la cantidad de excepciones que se presentan al modificar posteriormente claves foráneas, tipos de datos, etc, si hay un error de Dropea la tabla entera y se vuelve a crear, eso sí, habrá que consultar las claves foráneas y dropearlas primero.

**Tablas de relación**

**Cascada múltiple de eventos**

No está permitida la transitividad entre modificaciones en cascada en MSQLS. Cuando se crea una tabla de relación y no le puede asignar la restricción de modificación en cascada debido a que generaría una cascada múltiple de eventos. El comportamiento deseado de modificar las filas en la tabla de relación ante una modificación o eliminación en las tablas referenciadas debe programarse con otro mecanismo como un Trigger

**Llaves foráneas**

NombreCampo TipoDato FOREIGN KEY REFERENCES NombreTabla (NombreCampo)

ON UPDATE

CASCADE

SET NULL

NO ACTION

ON DELETE

CASCADE

SET NULL

NO ACTION

Y esto se debe a que las filas de una tabla de relación representan relaciones, y al eliminar registros en las tablas referenciadas éstas relaciones deben ser eliminadas.

#### Triggers

objeto de la base de datos que se ejecuta cuando ejecutamos una acción

**DML Trigger:** cuando insertamos, modificamos o eliminamos datos de una tabla o vista.

**DDL Trigger:**

Si tienes dos triggers que se ejecutan ante la misma acción y en la misma tabla, se ejecutan en el orden en que fueron creados, sin embargo este comportamiento se puede editar para especificar cual se ejecuta primero.

CREATE TRIGGER NombreTrigger

ON {Tabla, Vista) -Objeto referenciado

{FOR, AFTER, INSTEAD OF} – Momento de ejecución

{[INSERT] [,] [UPDATE] [,] [DELETE]} – Acción disparadora, debe ejecutarse ante una acción de este tipo en el objeto referenciado

AS

BEGIN

–Instrucciones a ejecutar (bloque de ejecución)

– Cuando se ejecuta un trigger dentro de este bloque de ejecución se crean dos tablas temporales:

–Inserted: Tabla temporal que contiene los registros recién insertados. Esta tabla va a estar vacía en caso que se haga – una eliminación

–Deleted: Tabla temporal que contiene los registros recién eliminados. Esta tabla va a estar vacía en caso que se haga una inserción

END;

**Aplicaciones recomendadas:**

1. Extender el control del dominio de los datos y lograr restricciones complejas que no pueden ser logradas utilizando Constraints
2. Lograr acciones en cascada cuando no pueda aplicarse una restricción equivalente debido a cascadas múltiple de eventos.

**Ventajas**

Además de la ejecución automática, sirven para

Aplicar restricciones complejas: restricciones basadas en datos de otras tablas

Seguridad: Controlar qué actualizaciones están permitidas y manejar otras reglas de negocio

Auditoría: Ante cada operación efectuada en la DB, crear un registro en una tabla de auditoría para poder revisar todo lo que se ha hecho en las DB.

Manejo de errores: Permite generar mensajes de error personalizados

Reutilización y centralización de código: Al almacenar un trigger a nivel de DB, en lugar de incluir esa lógica en todas las distintas aplicaciones que modifican las facturas, con un sólo trigger almacenado se puede ejecutar la lógica.

Encapsulamiento: esconde de los programadores y de los usuarios la lógica del negocio.

**Pseudo desventajas**

1. Los trigger forman parte de la metadata de la tabla: esto es falso pero es mejor creerlo como si fuera una verdad. Al modificar un trigger, el manejador de BBDD bloquea la tabla para hacer chequeos. Por ende es difícil modificar un trigger en una tabla que tiene muchos accesos

1. *Vas a olvidar que ese código está ahí.Crees que no pero lo harás la próxima persona ni va a pensar en echar un vistazo a los triggers el resultado neto es que hay cosas pasando en la base de datos y nadie sabe nada al respecto.¿Y si hay un error en el trigger? va a tomar todo el tiempo del mundo encontrarlo porque los desarrolladores se olvidan de los triggers, este es un gran problema. Tan solo por esta razón tan seria debe abstenerse de poner mucho código en los triggers.* Este problema es principalmente de los programadores, no de los triggers, pero hay que tenerlo en consideración.
2. Los programadores no tienen control total sobre la BBDD. Esto puede ser una ventaja/desventaja.
3. Es fácil olvidarse de los trigger si no hay documentación y los programadores pueden no saber lo que sucede: para ello debe existir la documentación siempre y en caso que no la haya los programadores deben ser inducidos en todo lo relacionado con la BBDD, revisar todos los triggers que existen y todos los objetos que hay.

**Desventajas**

1. Es fácil ver las relaciones entre las tablas, las relaciones y los índices, los triggers son más difíciles de ver. Para ello ¿habría que descargar la metadata de las tablas y observar de forma global todos los triggers existentes?
2. No es fácil seguir su lógica, es mucho más sencillo ver la lógica de un programa, porque están distribuidos a lo largo de la base de datos y es difícil ver la secuencialidad que hay entre ellos.
3. Los triggers bloquean la tabla que afectan, por ende es dificil mantener y modificarlos en tablas de alta utilización, accesos concurrentes:  Su mejor función es la de extender el control del dominio de los datos y lograr restricciones complejas que no pueden ser logradas utilizando Constrains. En cuanto a cuándo se necesita modificar un trigger, pues típicamente es cuando se cometió un error al crearlo o cuando cambian las reglas del negocio, al igual que con cualquier código.

#### Funciones

**Funciones del sistema**

**Funciones Escalares**: funciones que reciben 1 valor, ejecutan operaciones y devuelven 1 solo valor.Se pueden usar en cualquier lugar donde se pueda una expresión, por ejemplo en una cláusula WHERE, en un campo calculado, asignar su resultado a una variable o en condicionales, la gran mayoría son deterministas.

**Funciones deterministas**: siempre devuelven el mismo resultado ante los mismos inputs.

**Conversión:** CAST, CONVERT, PARSE.

**Matemáticas:** ABS, SQRT, LOG, RADIANS, EXP, SIN, COS, FLOOR.

**Cadenas de caracteres:** ASCII, CONCAT, FORMAT, LEFT, RIGHT, TRIM, STUFF, SUBSTRING, CHAR

**Fechas:** GETDATE, DATEPART, DAY, MONTH, YEAR, DATEADD, DATEDIFF, EOMONTH

**Funciones no deterministas**: para el mismo conjunto de datos de entrada, pueden devolver resultados diferentes.

Todas las funciones que consultan parámetros del sistema o valores que pueden cambiar como la metadata son funciones no deterministas.

La función RAND() es no determinista cuando no se le pasa ningún valor y si se le pasa un valor por ejemplo Rand(8) es determinista.

Sistema: @@ERROR, @@ROWCOUNT, @@IDENTITY, CHECKSUM

**Funciones de acumulación**: reciben un conjunto de valores y  devuelven 1 solo valor. Todas las funciones de acumulación son deterministas.

Ejemplo de una función de acumulación

SELECT \*

FROM ENVÍOS

WHERE FECHA = ‘16-12-2022’

GROUP BY TIENDA

HAVING VALOR = 500

La  cláusula WHERE es una condición sobre todos los registros de la relación y la cláusula Having es una condición sobre los resultados después de aplicar un agrupamiento y luego de aplicar las funciones de acumulación.

**Función definida por el usuario**

**Funciones Escalares**

SELECT Alumnos\_Curso(156) – aquí se usa como parte de una consulta select

DECLARE @Alumnos int = 0;

SET @Alumnos = Alumnos\_Curso(384); – aquí se usa para declarar una variable con el resultado que arroja la función

SELECT Cod\_Curso, Nombre, Peso

FROM Curso

WHERE Alumno\_Curso(Cod\_Curso)>20; – aquí se usa como parte de una condición WHERE.

Pueden usarse como parámetros de otra función

Funciones de tipo tabla

Función de una sóla instrucción (In-Line): aquella cuyo resultado se puede obtener de una sola instrucción SQL. Al retornar una tabla, podemos usarla como parte de cualquier consulta SELECT en el FROM de la consulta.

Función de múltiples instrucciones (Multi-Statement)

**Restricciones**

No pueden afectar el estado de la base de datos, por lo tanto se pueden usar para realizar consultas, crear vistas, pero no para insertar, modificar o borrar datos, así como tampoco modificar ningún otro objeto de la BBDD ni cambiar la configuración del sistema. Algunas funciones no deterministas como por ejemplo NEWID o RAND no pueden usarse dentro de las funciones UID porque modifican el estado del sistema cada vez que son utlizadas.

**Aplicaciones recomendadas**

Para condicionar una inserción de datos, si se crea una función que retorne un BIT puede crearse una restricción en un campo de una tabla que no permita la inserción si el resultado de la función es 0 o 1.

#### Vistas

Usar vistas como tabla base para otra vista **no es buena práctica**

**Vistas indexadas**

Las vistas no contienen datos, són instrucciones almacenadas como metadata, hasta que se crea un índice en la vista creando una vista indexada, de esta forma el RDBMS crea una tabla materializada con los datos de la vista. Las operaciones sobre ésta vista materializada son mucho más rápidas.

Se debe ser muy cuidadoso al crear vistas indexadas porque ralentiza mucho los tiempos de modificación, cualquier modificación sobre las tablas base deben realizarse también sobre los datos e índices de las vistas materializadas.

**Restricciones**

* Primero se debe crear un índice primario cluster y luego se pueden crear los non-cluster
* Sólo se pueden usar funciones deterministas en la vista.
* Otras condiciones establecidas por el RDBMS.
* Realizar SELECT sobre una vista formada por varias tablas obliga al RDBMS a buscar por la condición WHERE en todas las tablas que conforman la vista, sin embargo si en la definición de la propia vista ya se incluyen condiciones WHERE evidentes que definan los datos de cada una de las tablas que conforman la vista, se le ahorra al RDBMS la tarea de buscar sobre las tablas que no cumplen con la condición inicial.

**Vistas actualizables**

**Restricciones**

* Tablas base y registro unequívocos: Para modificaciones, sólo puede referenciar columnas de una misma tabla base, no se pueden modificar datos para distintas tablas base, por otro lado los registros deben ser identificables dentro de la vista, para ello debe incluirse la llave primaria de la tabla en la vista.
* Columnas derivadas: No se puede modificar derivadas: no pueden provenir de funciones de agregación, columnas agrupadas, columnas calculadas ni provenir de operaciones de conjuntos.
* Insertar valores nulos: Al insertar valores en una vista donde no estén todas las columnas de la tabla base, éstas columnas no referenciadas deben permitir valores nulos porque estos valores no serán incluidos en la inserción.
* Violar la condición WHERE  de la vista: Hay algunos RDBMS que no permiten actualizar vistas de tal forma que no se pueda modificar los valores violando la condición que hizo que aparecieran en la vista en primer lugar. Si se pudiera hacer estos valores serían excluidos de la vista la próxima vez que ésta se genere, ésto se evita con una instrucción WITH CHECK OPTION en la definición de la vista.
* Modificar vistas no actualizables: es posible actualizarlas usando un trigger INSTEAD OF.
* Prohibir las inserciones, eliminaciones y modificaciones sobre la vista, INSTEAD OF INSERT, DELETE, UPDATE.
* Preservar la vista ante cambios de esquema o eliminación de la tabla: con la instrucción WITH SCHEMABINDING en la definición de la vista se evitan modificaciones en la BBDD que puedan afectar la estructura o existencia de la vista.
* Nombre del esquema: todas las tablas bases en la definición de una vista deben incluir el nombre del esquema.

**Beneficios**

* **Cuando** se tiene la BBDD particionada, las vistas pueden servir para mostrar los datos particionados como si fueran una sóla tabla.
* Las vistas son equivalentes a funciones de tipo tabla sin parámetros, con la diferencia que algunas vistas son actualizables mientras las funciones no lo son.

**Seguridad**

Ocultar información y darle a los usuarios una vista con un condicional WHERE para restringir la visualización de datos privados.

Excluir o formatear columnas de la tabla, como por ejemplo mostrar sólo los últimos 4 números del seguro social y no el número completo

**Esquemas**

Los esquemas son agrupaciones de objetos dentro de una base de datos. Al crear una BBDD se suelen crear esquemas por defecto sin embargo esto puede ser modificado. En MSSQLS todos los objetos que se crean sin especificar el esquema, se crean dentro del esquema dbo

Si cambias de BBDD aunque especifiques el esquema no vas a poder consultar objetos de otras BBDD. La única forma de hacerlo es usando identificadores de tres partes: NombreBBDD.NombreEsquema.NombreObjeto

**Identificadores**

Servidor.[BBDD.[Esquema]].Objeto

#### Campos calculados

Al definir un campo calculado el manejador va a almacenar la metadata, la definición de cómo el campo debe ser calculado, se va a ejecutar el cálculo únicamente cuando sea necesario, por ejemplo al ejecutar una instrucción SELECT.

Si necesitamos que se almacene el resultado del cálculo en la tabla se agrega la cláusula PERSISTED. Cada vez que se inserte un registro o cada vez que se modifique un registro, el campo será calculado. Esto también sería necesario si quisiéramos crear un índice basado en éste campo.

NombreCampo AS ExpresiónDelCampo PERSISTED

A éste campo no se le asigna ningún tipo de dato, será asignado automáticamente por el manejador.

#### Fase de pruebas

**Inserción de datos**

**Violaciones de dominio:** Tipos de datos, reglas de chequeos, anulación y unicidad.

**Llaves primarias:** unicidad y anulación

**Transacciones**

**Violar llaves foráneas**

**Violaciones sobre relaciones**

**Eventos en cascada**

**Campos autoincrementales**

INSERT nombre\_tabla (col1, col2, …, colN)

VALUES (val1, val2, …, valN)

– no pueden haber columnas autogeneradas y columnas calculadas porque esas se generan automáticamente, deben incluirse las Not Null y las PK que no sean autogeneradas.

– En la inserción de valores, no es necesario que tengan el mismo orden que la tabla, sólo deben coincidir posicionalmente con las columnas en la cláusula INSERT INTO

INSERT INTO nombre\_tabla – en este caso sí deben coincidir con el orden de las columnas de la tabla, **no es buena práctica**

SELECT col1, col2, …, col1+col2

FROM tabla