Tema 2

Diseño de bases de datos.
Diseño lógico y físico.
El modelo lógico relacional.
Normalización.

Guion-resumen

1. Diseño de bases de datos

1.1. Diseño conceptual de bases de datos

Es una etapa bastante compleja sobre todo para el diseñador de la base de datos. En esta etapa se tiene que construir un esquema de la información que utiliza la empresa, independientemente de cualquier consideración física (esquema conceptual). El diseñador debe comprender muy bien los datos que utiliza la empresa para de ahí poder obtener las tablas, las relaciones, los tipos de campos, etc.

El objetivo es comprender:

— La naturaleza de los datos.

— La perspectiva que cada usuario tiene de los datos.

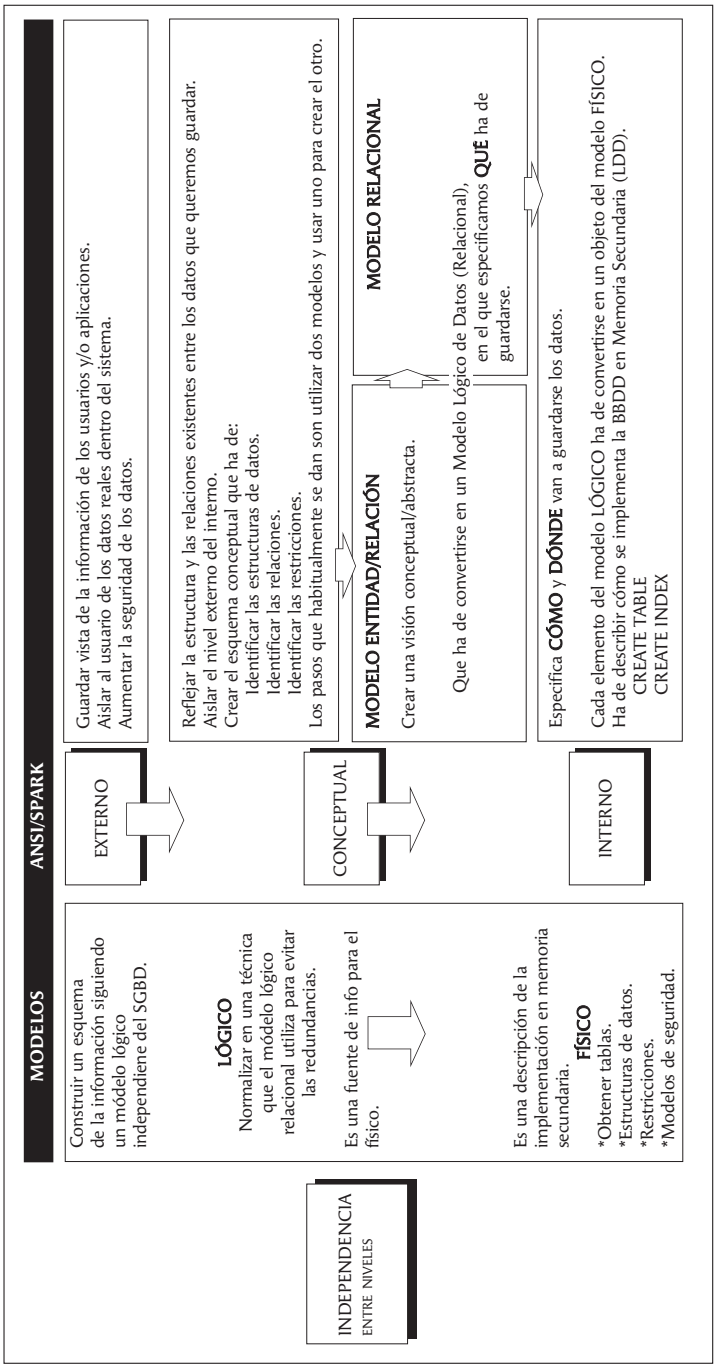
— El uso de los datos a través de las áreas de aplicación.

Este es un período de mucha conversación entre el diseñador y la empresa; el objetivo es que las dos partes entiendan la notación utilizada en el esquema.

La más popular es la notación del modelo entidad/relación.

El esquema conceptual se construye utilizando la información que se encuentra en la especificación de los requisitos de usuario. Este diseño es completamente independiente de los aspectos de implementación, como puede ser el Sistema Gestor de Base de Datos (SGBD) que se vaya a usar, los programas de aplicación, los lenguajes de programación, el hardware disponible o cualquier otra consideración física. Durante todo el proceso de desarrollo de este esquema se deben de realizar pruebas y validarlas con los requisitos de los usuarios.

El esquema conceptual es una fuente de información para el posterior diseño lógico de la base de datos.


MODELOS
ANSI/SPARK
EXTERNO
Guardar vista de la información de los usuarios y/o aplicaciones.
Aislar al usuario de los datos reales dentro del sistema.
Aumentar la seguridad de los datos.
CONCEPTUAL
Reflejar la estructura y las relaciones existentes entre los datos que queremos guardar.
Aislar el nivel externo del interno.
Crear el esquema conceptual que ha de:
Identificar las estructuras de datos.
Identificar las relaciones.
Identificar las restricciones.
Los pasos que habitualmente se dan son utilizar dos modelos y usar uno para crear el otro.
MODELO ENTIDAD/RELACIÓN
Crear una visión conceptual/abstracta.
MODELO RELACIONAL
Que ha de convertirse en un Modelo Lógico de Datos (Relacional), en el que especificamos QUÉ ha de guardarse.
INTERNO
Especifica CÓMO y DÓNDE van a guardarse los datos.
Cada elemento del modelo LÓGICO ha de convertirse en un objeto del modelo FÍSICO.
Ha de describir cómo se implementa la BBDD en Memoria Secundaria (LDD).
CREATE TABLE
CREATE INDEX
INDEPENDENCIA
ENTRE NIVELES
Construir un esquema de la información siguiendo un módelo lógico independiene del SGBD.
LÓGICO
Normalizar en una técnica que el módelo lógico relacional utiliza para evitar las redundancias.
Es una fuente de info para el físico.
Es una descripción de la implementación en memoria secundaria.
FÍSICO
*Obtener tablas.
*Estructuras de datos.
*Restricciones.
*Modelos de seguridad.

2. Diseño lógico y físico

2.1. Diseño lógico

El diseño lógico es el proceso de construir un esquema de la información que utiliza la empresa, basándose en un modelo de base de datos específico, independiente del SGBD concreto que se vaya a utilizar y de cualquier otra consideración física.

En esta etapa, se transforma el esquema conceptual en un esquema lógico que utilizará las estructuras de datos del modelo de base de datos en el que se basa el SGBD que se vaya a utilizar, como puede ser el modelo relacional, el modelo de red, el modelo jerárquico o el modelo orientado a objetos. Conforme se va desarrollando el esquema lógico, este se va probando y validando con los requisitos de usuario.

La **normalización** es una técnica que se utiliza para comprobar la validez de los esquemas lógicos basados en el modelo relacional, ya que asegura que las relaciones (tablas) obtenidas no tienen datos redundantes. Esta técnica se presenta en el capítulo dedicado al diseño lógico de bases de datos.

El esquema lógico es una fuente de información para el diseño físico. Además, juega un papel importante durante la etapa de mantenimiento del sistema, ya que permite que los futuros cambios que se realicen sobre los programas de aplicación o sobre los datos, se representen correctamente en la base de datos.

Tanto el diseño conceptual, como el diseño lógico, son procesos iterativos, tienen un punto de inicio y se van refinando continuamente. Ambos se deben ver como un proceso de aprendizaje en el que el diseñador va comprendiendo el funcionamiento de la empresa y el significado de los datos que maneja. El diseño conceptual y el diseño lógico son etapas clave para conseguir un sistema que funcione correctamente. Si el esquema no es una representación fiel de la empresa, será difícil, sino imposible, definir todas las vistas de usuario (esquemas externos), o mantener la integridad de la base de datos. También puede ser difícil definir la implementación física o mantener unas prestaciones aceptables del sistema. Además, hay que tener en cuenta que la capacidad de ajustarse a futuros cambios es un sello que identifica a los buenos diseños de bases de datos. Por todo esto, es fundamental dedicar el tiempo y las energías necesarias para producir el mejor esquema que sea posible.

2.2. Diseño físico

El diseño físico es el proceso que produce la descripción de la implementación de la base de datos en memoria secundaria: estructura de almacenamiento y métodos de acceso a los datos.

Antes de empezar esta etapa se tiene que tener ya decidido el SGBD a utilizar, pues el esquema físico se adapta a él. En definitiva, el esquema físico es la implementación del esquema lógico. La mayoría de las veces se suele modificar el esquema físico para mejorar la base de datos con lo cual el esquema lógico también suele sufrir modificaciones. Concretamente el diseño físico en el modelo relacional consiste en:

— Obtener un conjunto de relaciones (tablas) y las restricciones que se deben cumplir sobre ellas.

— Determinar las estructuras de almacenamiento y los métodos de acceso que se van a utilizar para conseguir unas prestaciones óptimas.

— Diseñar el modelo de seguridad del sistema.

3. El modelo lógico relacional

A continuación se describen los pasos para llevar a cabo el diseño lógico. Ya que aquí se trata el diseño de bases de datos relacionales, en esta etapa se obtiene un conjunto de relaciones (tablas) que representen los datos de interés. Este conjunto de relaciones se valida mediante la normalización.

3.1. Introducción

El objetivo del diseño lógico es convertir los esquemas conceptuales locales en un esquema lógico global que se ajuste al modelo de SGBD sobre el que se vaya a implementar el sistema. Mientras que el objetivo fundamental del diseño conceptual es la comprensión y expresividad de los esquemas conceptuales locales, el objetivo del diseño lógico es obtener una representación que use, del modo más eficiente posible, los recursos que el modelo de SGBD posee para estructurar los datos y para modelar las restricciones.

Los modelos de bases de datos más extendidos son el modelo relacional, el modelo de red, el modelo jerárquico y el modelo orientado a objetos.

El modelo relacional (y los modelos previos) carecen de ciertos rasgos de abstracción que se usan en los modelos conceptuales. Por lo tanto, un primer paso en la fase del diseño lógico consistirá en la conversión de esos mecanismos de representación de alto nivel en términos de las estructuras de bajo nivel disponibles en el modelo relacional.

3.2. Metodología de diseño lógico en el modelo relacional

La metodología que se va a seguir para el diseño lógico en el modelo relacional consta de dos fases, cada una de ellas compuesta por varios pasos que se detallan a continuación:

1. Construir y validar los esquemas lógicos locales para cada vista de usuario.
    - Convertir los esquemas conceptuales locales en esquemas lógicos locales.
    - Derivar un conjunto de relaciones (tablas) para cada esquema lógico local.
    - Validar cada esquema mediante la normalización.
    - Validar cada esquema frente a las transacciones del usuario.
    - Dibujar el diagrama entidad-relación.

- Definir las restricciones de integridad.
- Revisar cada esquema lógico local con el usuario correspondiente.

2. Construir y validar el esquema lógico global.
   - Mezclar los esquemas lógicos locales en un esquema lógico global.
   - Validar el esquema lógico global.
   - Estudiar el crecimiento futuro.
   - Dibujar el diagrama entidad-relación final.
   - Revisar el esquema lógico global con los usuarios.

En la primera fase, se construyen los esquemas lógicos locales para cada vista de usuario y se validan. En esta fase se refinan los esquemas conceptuales creados durante el diseño conceptual, eliminando las estructuras de datos que no se pueden implementar de manera directa sobre el modelo que soporta el SGBD, en el caso que nos ocupa, el modelo relacional. Una vez hecho esto, se obtiene un primer esquema lógico que se valida mediante la normalización y frente a las transacciones que el sistema debe llevar a cabo, tal y como se refleja en las especificaciones de requisitos de usuario. El esquema lógico ya validado se puede utilizar como base para el desarrollo de prototipos. Una vez finalizada esta fase, se dispone de un esquema lógico para cada vista de usuario que es correcto, comprensible y sin ambigüedad.

3.2.1. Convertir los esquemas conceptuales locales en esquemas lógicos locales

En este paso, se eliminan de cada esquema conceptual las estructuras de datos que los sistemas relacionales no modelan directamente:

a) **Eliminar las relaciones de muchos a muchos**, sustituyendo cada una de ellas por una nueva entidad intermedia y dos relaciones de uno a muchos de esta nueva entidad con las entidades originales. La nueva entidad será débil, ya que sus ocurrencias dependen de la existencia de ocurrencias en las entidades originales.

b) **Eliminar las relaciones entre tres o más entidades**, sustituyendo cada una de ellas por una nueva entidad (débil) intermedia que se relaciona con cada una de las entidades originales. La cardinalidad de estas nuevas relaciones binarias dependerá de su significado.

c) **Eliminar las relaciones recursivas**, sustituyendo cada una de ellas por una nueva entidad (débil) y dos relaciones binarias de esta nueva entidad con la entidad original. La cardinalidad de estas relaciones dependerá de su significado.

d) **Eliminar las relaciones con atributos**, sustituyendo cada una de ellas por una nueva entidad (débil) y las relaciones binarias correspon-

dientes de esta nueva entidad con las entidades originales. La cardinalidad de estas relaciones dependerá del tipo de la relación original y de su significado.

e) **Eliminar los atributos multievaluados**, sustituyendo cada uno de ellos por una nueva entidad (débil) y una relación binaria de uno a muchos con la entidad original.

f) **Revisar las relaciones de uno a uno**, ya que es posible que se hayan identificado dos entidades que representen el mismo objeto (sinónimos). Si así fuera, ambas entidades deben integrarse en una sola.

g) **Eliminar las relaciones redundantes.** Una relación es redundante cuando se puede obtener la misma información que ella aporta mediante otras relaciones. El hecho de que haya dos caminos diferentes entre dos entidades no implica que uno de los caminos corresponda a una relación redundante, eso dependerá del significado de cada relación.

Una vez finalizado este paso, es más correcto referirse a los esquemas conceptuales locales refinados como esquemas lógicos locales, ya que se adaptan al modelo de base de datos que soporta el SGBD escogido.

3.2.2. Derivar un conjunto de relaciones (tablas) para cada esquema lógico local

En este paso, se obtiene un conjunto de relaciones (tablas) para cada uno de los esquemas lógicos locales en donde se representen las entidades y relaciones entre entidades, que se describen en cada una de las vistas que los usuarios tienen de la empresa. Cada relación de la base de datos tendrá un nombre, y el nombre de sus atributos aparecerá, a continuación, entre paréntesis. El atributo o atributos que forman la clave primaria se subrayan. Las claves ajenas, mecanismo que se utiliza para representar las relaciones entre entidades en el modelo relacional, se especifican aparte indicando la relación (tabla) a la que hacen referencia.

A continuación, se describe cómo las relaciones (tablas) del modelo relacional representan las entidades y relaciones que pueden aparecer en los esquemas lógicos.

a) **Entidades fuertes.** Crear una relación para cada entidad fuerte que incluya todos sus atributos simples. De los atributos compuestos incluir solo sus componentes.

   Cada uno de los identificadores de la entidad será una clave candidata. De entre las claves candidatas hay que escoger la clave primaria; el resto serán claves alternativas. Para escoger la clave primaria entre las claves candidatas se pueden seguir estas indicaciones:

   - Escoger la clave candidata que tenga menos atributos.
   - Escoger la clave candidata cuyos valores no tengan probabilidad de cambiar en el futuro.

- Escoger la clave candidata cuyos valores no tengan probabilidad de perder la unicidad en el futuro.
- Escoger la clave candidata con el mínimo número de caracteres (si es de tipo texto).
- Escoger la clave candidata más fácil de utilizar desde el punto de vista de los usuarios.

b) **Entidades débiles.** Crear una relación para cada entidad débil incluyendo todos sus atributos simples. De los atributos compuestos incluir solo sus componentes. Añadir una clave ajena a la entidad de la que depende. Para ello, se incluye la clave primaria de la relación que representa a la entidad padre en la nueva relación creada para la entidad débil. A continuación, determinar la clave primaria de la nueva relación.

c) **Relaciones binarias de uno a uno.** Para cada relación binaria se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. La entidad hijo es la que participa de forma total (obligatoria) en la relación, mientras que la entidad padre es la que participa de forma parcial (opcional). Si las dos entidades participan de forma total o parcial en la relación, la elección de padre e hijo es arbitraria. Además, en caso de que ambas entidades participen de forma total en la relación, se tiene la opción de integrar las dos entidades en una sola relación (tabla). Esto se suele hacer si una de las entidades no participa en ninguna otra relación.

d) **Relaciones binarias de uno a muchos.** Como en las relaciones de uno a uno, se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. Pero ahora, la entidad padre es la de "la parte del muchos" (cada padre tiene muchos hijos), mientras que la entidad hijo es la de "la parte del uno" (cada hijo tiene un solo padre).

Una vez obtenidas las relaciones con sus atributos, claves primarias y claves ajenas, solo queda actualizar el diccionario de datos con los nuevos atributos que se hayan identificado en este paso.

3.2.3. Validar cada esquema mediante la normalización

La normalización se utiliza para mejorar el esquema lógico, de modo que satisfaga ciertas restricciones que eviten la duplicidad de datos. La normalización garantiza que el esquema resultante se encuentra más próximo al modelo de la empresa, que es consistente y que tiene la mínima redundancia y la máxima estabilidad.

La normalización es un proceso que permite decidir a qué entidad pertenece cada atributo. Uno de los conceptos básicos del modelo relacional es que los atributos se agrupan en relaciones (tablas) porque están relacionados a nivel lógico. En la mayoría de las ocasiones, una base de datos normalizada no

proporciona la máxima eficiencia, sin embargo, el objetivo ahora es conseguir una base de datos normalizada por las siguientes razones:

— Un esquema normalizado organiza los datos de acuerdo a sus dependencias funcionales, es decir, de acuerdo a sus relaciones lógicas.

— El esquema lógico no tiene por qué ser el esquema final. Debe representar lo que el diseñador entiende sobre la naturaleza y el significado de los datos de la empresa. Si se establecen unos objetivos en cuanto a prestaciones, el diseño físico cambiará el esquema lógico de modo adecuado. Una posibilidad es que algunas relaciones normalizadas se desnormalicen. Pero la desnormalización no implica que se haya malgastado tiempo normalizando, ya que mediante este proceso el diseñador aprende más sobre el significado de los datos. De hecho, la normalización obliga a entender completamente cada uno de los atributos que se han de representar en la base de datos.

— Un esquema normalizado es robusto y carece de redundancias, por lo que está libre de ciertas anomalías que estas pueden provocar cuando se actualiza la base de datos.

— Los equipos informáticos de hoy en día son mucho más potentes, por lo que en ocasiones es más razonable implementar bases de datos fáciles de manejar (las normalizadas), a costa de un tiempo adicional de proceso.

— La normalización produce bases de datos con esquemas flexibles que pueden extenderse con facilidad.

El objetivo de este paso es obtener un conjunto de relaciones que se encuentren en la forma normal de Boyce-Codd. Para ello, hay que pasar por la primera, segunda y tercera formas normales. El proceso de normalización se describe en el epígrafe 4.

3.2.4. Validar cada esquema frente a las transacciones del usuario

El objetivo de este paso es validar cada esquema lógico local para garantizar que puede soportar las transacciones requeridas por los correspondientes usuarios. Estas transacciones se encontrarán en las especificaciones de requisitos de usuario. Lo que se debe hacer es tratar de realizar las transacciones de forma manual utilizando el diagrama entidad-relación, el diccionario de datos y las conexiones que establecen las claves ajenas de las relaciones (tablas). Si todas las transacciones se pueden realizar, el esquema queda validado. Pero si alguna transacción no se puede realizar, seguramente será porque alguna entidad, relación o atributo no se ha incluido en el esquema.

3.2.5. Dibujar el diagrama entidad-relación

En este momento, se puede dibujar el diagrama entidad-relación final para cada vista de usuario que recoja la representación lógica de los datos desde su punto de vista. Este diagrama habrá sido validado mediante la normalización y frente a las transacciones de los usuarios.

3.2.6. Definir las restricciones de integridad

Las restricciones de integridad son reglas que se quieren imponer para proteger la base de datos, de modo que no pueda llegar a un estado inconsistente. Hay cinco tipos de restricciones de integridad.

a) **Datos requeridos.** Algunos atributos deben contener valores en todo momento, es decir, no admiten nulos.

b) **Restricciones de dominios.** Todos los atributos tienen un dominio asociado, que es el conjunto de los valores que cada atributo puede tomar.

c) **Integridad de entidades.** El identificador de una entidad no puede ser nulo, por lo tanto, las claves primarias de las relaciones (tablas) no admiten nulos.

d) **Integridad referencial.** Una clave ajena enlaza cada tupla de la relación hijo con la tupla de la relación padre que tiene el mismo valor en su clave primaria. La integridad referencial dice que si una clave ajena tiene un valor (si es no nula), ese valor debe ser uno de los valores de la clave primaria a la que referencia. Hay varios aspectos a tener en cuenta sobre las claves ajenas para lograr que se cumpla la integridad referencial.

   1. ¿Admite nulos la clave ajena? Cada clave ajena expresa una relación. Si la participación de la entidad hijo en la relación es total, entonces la clave ajena no admite nulos; si es parcial, la clave ajena debe aceptar nulos.

   2. ¿Qué hacer cuando se quiere borrar una ocurrencia de la entidad padre que tiene algún hijo? O lo que es lo mismo, ¿qué hacer cuando se quiere borrar una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Hay varias respuestas posibles:

      - Restringir: no se pueden borrar tuplas que están siendo referenciadas por otras tuplas.

      - Propagar: se borra la tupla deseada y se propaga el borrado a todas las tuplas que le hacen referencia.

      - Anular: se borra la tupla deseada y todas las referencias que tenía se ponen, automáticamente, a nulo (esta respuesta solo es válida si la clave ajena acepta nulos).

      - Valor por defecto: se borra la tupla deseada y todas las referencias toman, automáticamente, el valor por defecto (esta respuesta solo es válida si se ha especificado un valor por defecto para la clave ajena).

      - No comprobar: se borra la tupla deseada y no se hace nada para garantizar que se sigue cumpliendo la integridad referencial.

3. ¿Qué hacer cuando se quiere modificar la clave primaria de una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Las respuestas posibles son las mismas que en el caso anterior. Cuando se escoge propagar, se actualiza la clave primaria en la tupla deseada y se propaga el cambio a los valores de clave ajena que le hacían referencia.

e) **Reglas de negocio.** Cualquier operación que se realice sobre los datos debe cumplir las restricciones que impone el funcionamiento de la empresa.

Todas las restricciones de integridad establecidas en este paso se deben reflejar en el diccionario de datos para que puedan ser tenidas en cuenta durante la fase del diseño físico.

3.2.7. Revisar cada esquema lógico local con el usuario correspondiente

Para garantizar que cada esquema lógico local es una fiel representación de la vista del usuario lo que se debe hacer es comprobar con él que lo reflejado en el esquema y en la documentación es correcto y está completo.

- **Relación entre el esquema lógico y los diagramas de flujo de datos**

El esquema lógico refleja la estructura de los datos a almacenar que maneja la empresa. Un diagrama de flujo de datos muestra cómo se mueven los datos en la empresa y los almacenes en donde se guardan. Si se han utilizado diagramas de flujo de datos para modelar las especificaciones de requisitos de usuario, se pueden utilizar para comprobar la consistencia y completitud del esquema lógico desarrollado. Para ello:

— Cada almacén de datos debe corresponder con una o varias entidades completas.

— Los atributos en los flujos de datos deben corresponder a alguna entidad.

Los esquemas lógicos locales obtenidos hasta este momento se integrarán en un solo esquema lógico global en la siguiente fase para modelar los datos de toda la empresa.

3.2.8. Mezclar los esquemas lógicos locales en un esquema lógico global

En este paso se deben integrar todos los esquemas locales en un solo esquema global. En un sistema pequeño, con dos o tres vistas de usuario y unas pocas entidades y relaciones, es relativamente sencillo comparar los esquemas locales, mezclarlos y resolver cualquier tipo de diferencia que pueda existir. Pero en los sistemas grandes, se debe seguir un proceso más sistemático para llevar a cabo este paso con éxito:

1. Revisar los nombres de las entidades y sus claves primarias.
2. Revisar los nombres de las relaciones.

3. Mezclar las entidades de las vistas locales.
4. Incluir (sin mezclar) las entidades que pertenecen a una sola vista de usuario.
5. Mezclar las relaciones de las vistas locales.
6. Incluir (sin mezclar) las relaciones que pertenecen a una sola vista de usuario.
7. Comprobar que no se ha omitido ninguna entidad ni relación.
8. Comprobar las claves ajenas.
9. Comprobar las restricciones de integridad.
10. Dibujar el esquema lógico global.
11. Actualizar la documentación.

3.2.9. Validar el esquema lógico global

Este proceso de validación se realiza, de nuevo, mediante la normalización y mediante la prueba frente a las transacciones de los usuarios. Pero ahora solo hay que normalizar las relaciones que hayan cambiado al mezclar los esquemas lógicos locales y solo hay que probar las transacciones que requieran acceso a áreas que hayan sufrido algún cambio.

3.2.10. Estudiar el crecimiento futuro

En este paso, se trata de comprobar que el esquema obtenido puede acomodar los futuros cambios en los requisitos con un impacto mínimo. Si el esquema lógico se puede extender fácilmente, cualquiera de los cambios previstos se podrá incorporar al mismo con un efecto mínimo sobre los usuarios existentes.

3.2.11. Dibujar el diagrama entidad-relación final

Una vez validado el esquema lógico global, ya se puede dibujar el diagrama entidad-relación que representa el modelo de los datos de la empresa que son de interés. La documentación que describe este modelo (incluyendo el esquema relacional y el diccionario de datos) se debe actualizar y completar.

3.2.12. Revisar el esquema lógico global con los usuarios

Una vez más, se debe revisar con los usuarios el esquema global y la documentación obtenida para asegurarse de que son una fiel representación de la empresa.

4. Normalización

Este proceso tiene como objetivo comprobar que las tablas que forman la base de datos cumplen unas determinadas condiciones. La principal condición es evitar la redundancia (se entiende por redundancia la repetición de los datos albergados en la BD) y una cierta coherencia en la configuración mediante un esquema relacional de las entidades y relaciones del modelo conceptual (diagrama E-R).

Gracias a la normalización se pueden evitar errores de diseño y anomalías en la actualización y borrado en las tablas de la base de datos, facilitando la gestión del administrador de la misma y de los desarrolladores de aplicaciones.

Esquema relacional: conjunto de tablas con sus atributos.

Se pretende comprobar si ese esquema es funcionalmente mejorable por medio de las reglas de normalización.

Se dice que una tabla está en una determinada forma normal si satisface un cierto número de restricciones impuestas por esa regla.

El número de estas reglas puede variar hasta seis (dependiendo de autores) pero hay tres de ellas que resultan básicas.

Una tabla normalizada de acuerdo a la primera regla se dice que está en la primera forma normal (1NF). Una tabla normalizada de acuerdo a la segunda regla se dice que está en la segunda forma normal (2NF). Una tabla debe cumplir la primera forma antes que la segunda, y la segunda antes que la tercera pero después que la primera. Las normalizaciones mayores tratan con situaciones específicas y especiales, que los programadores suelen adaptar de forma individual.

Primeramente veamos unos errores muy comunes en el diseño de bases de datos:

Tenemos una base de datos con la cual pretendemos "controlar" a los propietarios de vehículos de una determinada localidad. Necesitamos, en principio, una tabla para almacenar dicha información.

DNI	Nombre	Apellidos	Dirección	Marca	Modelo	Matrícula
5042211	Jorge	Arrainz	C/ Ronda	Audi	A3	abc9090
4012123	Eva	Vall	C/ Tórtola	SEAT	Ibiza	11221bb
5042211	Jorge	Arrainz	C/ Ronda	BMW	850	m-2277-uv
3555655	Ana	Martín	C/ Jadraque	Mercedes	600	434344bb
8989888	Alicia	Márquez	C/ Hita	SEAT	Ibiza	454544as

El error más visible, como se puede apreciar, es la duplicidad de datos. Jorge, que es propietario de dos vehículos ocupa dos registros (1º y 3º), por lo cual se repiten varios campos: DNI, Nombre, Apellidos y Dirección. ¿No sería ocupar espacio de almacenamiento en demasía?

Pensemos en otra posibilidad.

DNI	Nomb	Marca	Model	Matric	Marca2	Modelo2	Matricu2
5042211	Jorge	Audi	A3	abc9090	BMW	850	m-2277uv
4012123	Eva	SEAT	Ibiza	11221bb			
3555655	Ana	Mercedes	600	434344bb			
8989888	Alicia	SEAT	Ibiza	454544as			

Esta tabla resulta aún peor. Al intentar evitar la duplicidad de los datos en la tabla, se está introduciendo duplicidad en la misma estructura de la tabla. Además debemos preguntarnos ¿cuántas personas son **propietarias** de dos o más vehículos? ¿En cuántas filas los campos Marca2, Modelo2 y Matrícula2 quedarían vacíos? En miles, pues la inmensa mayoría de gente no tiene dos o más vehículos por persona.

Primera forma normal

Una tabla no debe contener grupos repetidos. Basándonos en el segundo ejemplo anterior, no podríamos tener los campos matrícula y matrícula2, modelo y modelo2, etc.

Por lo cual un propietario con dos o más vehículos se almacenaría así:

DNI	Nombre	Apellidos	Dirección	Marca	Modelo	Matrícula
5042211	Jorge	Arrainz	C/ Ronda	Audi	A3	abc9090
4012123	Eva	Vall	C/ Tórtola	SEAT	Ibiza	11221bb
5042211	Jorge	Arrainz	C/ Ronda	BMW	850	m-2277-uv
3555655	Ana	Martín	C/ Jadraque	Mercedes	600	434344bb
8989888	Alicia	Márquez	C/ Hita	SEAT	Ibiza	454544as

La primera forma normal no requiere que se divida la tabla en otras. En vez de eso, convierte algunas de las columnas de la tabla en filas adicionales. Ventajas que tiene, siempre dentro de que se puede mejorar, claro está:

— Carece de campos vacíos.

— Carece de limitaciones. Si un propietario tiene, por ejemplo, seis vehículos, tendríamos que haber utilizado seis campos matrícula, seis campos modelo, etc. De esta forma, el propietario aparecería en seis filas.

Una vez que la tabla se encuentra en la primera forma normalizada, se debe pasar a la segunda regla de normalización.

Segunda forma normal

La segunda regla de normalización dice que cualquier campo que no dependa totalmente de la clave principal se debe mover a otra tabla. Aplicándolo a nuestro ejemplo:

Tabla propietarios

DNI	Nombre	Apellidos	Dirección
5042211	Jorge	Arrainz	C/ Ronda
4012123	Eva	Vall	C/ Tórtola
3555655	Ana	Martín	C/ Jadraque
8989888	Alicia	Marquez	C/ Hita

Tabla vehículos

DNI	Marca	Modelo	Matrícula
5042211	Audi	A3	abc9090
4012123	SEAT	Ibiza	11221bb
5042211	BMW	850	m-2277-uv
3555655	Mercedes	600	434344bb
8989888	SEAT	Ibiza	454544as

Ahora las tablas quedan mejor estructuradas. Por medio del campo clave DNI podemos acceder a la información de un propietario y/o de su vehículo/s, y los datos apenas se repiten, salvo el DNI que es clave primaria y que en una relación de uno a varios (un propietario, uno o más vehículos) es imposible de evitar.

Vamos a ampliar la tabla de vehículos con dos nuevos campos: cilindrada e impcircu (impuesto de circulación, según cilindrada).

Tabla vehículos

DNI	Marca	Modelo	Matrícula	Cilindrada	impcircu
5042211	Audi	A3	abc9090	1900	105
4012123	SEAT	Ibiza	11221bb	1600	82
5042211	BMW	850	m-2277-uv	3000	203
3555655	Mercedes	600	434344bb	3000	203
8989888	SEAT	Ibiza	454544as	1600	82

Ahora imaginémonos que cambia el importe del campo de impuestos para algunas cilindradas. La tabla necesita una modificación, pero esa modificación se va a repetir innecesariamente en algún registro (SEAT Ibiza), ya que existen registros iguales.

Tercera forma normal

Esta tercera regla dice que no debe haber dependencias entre campos que no sean clave. Aquí el problema reside en que los relativos a las características del vehículo se repiten en demasía y pecan de tener una dependencia de un campo no clave, impcircu.

Volvamos a normalizar.

Tabla vehículos

Idvehículo	Marca	Modelo	Cilindrada
Au31901	Audi	A3	1900
Seibi1603	SEAT	Ibiza	1600
Bm850i	BMW	850	3000
Me600se	Mercedes	600	3000

Tabla impuestos

Cilindrada	Impuesto_circulación	EmisiónCO2	Características
1100	50	6%	Bajo consumo...
1200	55	8%	Motor ecológico...
1600	82	15%	Versiones con turbo...
1900	105	18%	Consumo alto, turbo...
3000	203	25%	Alto consumo...

Esta tabla, además de no repetir tanta marca y cilindrada, muestra más información a cerca del tipo de motor. El campo cilindrada es la clave principal.

Tabla propietarios

DNI	Nombre	Apellidos	Dirección
5042211	Jorge	Arrainz	C/ Ronda
4012123	Eva	Vall	C/ Tórtola
3555655	Ana	Martín	C/ Jadraque
8989888	Alicia	Marquez	C/ Hita

Esta tabla pasaríamos a normalizarla, para que no aparezcan varias veces los datos personales de un mismo propietario, sino gracias al DNI poder acceder a la tabla de los datos personales.

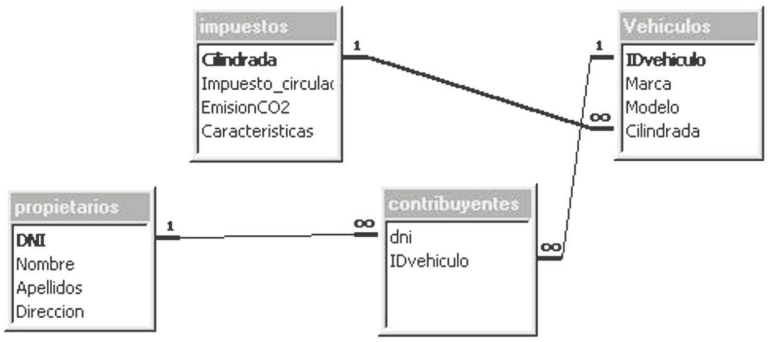
Tabla contribuyentes

DNI	IDvehículo
5042211	Au31901
4012123	Seibi1603
3555655	Me600se
8989888	Seibi1603
5042211	Bm850i

Con esta tabla tenemos acceso a los datos personales y a los datos del vehículo, y como las otras tablas están relacionadas entre sí, podemos acceder a la información de todas las tablas a la vez.

La segunda y tercera forma son prácticamente iguales (de hecho algunos autores las tratan como una sola).

El diagrama de relaciones de una base de datos normalizada quedaría así:



La mejor forma es la intuición y el sentido común para dividir los datos en tablas diferentes. Use una tabla diferente para cada entidad; posteriormente establezca las relaciones entre las tablas, siempre que se pueda.

Si alguna tabla no tiene campo común con otra tabla para poder relacionarlas, deberá crearse una tabla intermedia que contenga dos campos; los

campos clave de ambas tablas. Algo parecido a la tabla contribuyentes de nuestro ejemplo, que permite relacionar la tabla propietarios con las demás.

Forma Normal de Boyce-Codd (BCFN)

Una relación está en la forma normal de Boyce-Codd si, y solo si, todo determinante es una clave candidata.

La 2FN y la 3FN eliminan las dependencias parciales y las dependencias transitivas de la clave primaria. Pero este tipo de dependencias todavía pueden existir sobre otras claves candidatas, si estas existen. La BCFN es más fuerte que la 3FN, por lo tanto, toda relación en BCFN está en 3FN.

La violación de la BCFN es poco frecuente ya que se da bajo ciertas condiciones que raramente se presentan. Se debe comprobar si una relación viola la BCFN, si tiene dos o más claves candidatas compuestas que tienen al menos un atributo en común.

Respecto a la normalización y vínculos

Cada vez que conecta dos tablas con una clave común, SGBD (o DBMS, *Data Base Manager System*) debe llevar a cabo una operación conocida como unión. Esta unión se realiza por medio de las claves principal *(Primary Key)* y externa *(Foreign Key)*. Esta operación de unir "momentáneamente" tablas es muy costosa, debido a su lentitud, por lo cual se aconseja usarla lo menos posible. Es normal que algunas bases de datos utilicen incluso seis uniones para acceder a una información, aunque SQL permite hasta 16 uniones.

En verdad existe un conflicto entre normalización y uniones. Si creamos un gran número de tablas, tendremos que crear un gran número de uniones. Hay muchos administradores y programadores de bases de datos que prefieren repetir un "poco" la información utilizando menos tablas con tal de poder acceder más rápidamente a los datos. Que quede claro que para esto no hay reglas, solo el criterio personal y estudiar muy bien el tipo y número de accesos o peticiones que se van a realizar sobre las tablas.

5. Integridad de la base de datos

Diseñar la base de datos es solo el primer paso, el mayor problema es que la base de datos se mantenga en perfecto estado, operativa. Para el mantenimiento y protección de la base de datos los SGBD se ayudan de unas determinadas reglas de integridad y los administradores y programadores son los encargados de aplicarlas.

5.1. Integridad del dominio

Es una regla de integridad muy simple que indica que cada columna (campo) debe tener un tipo único de datos. Por ejemplo, si la columna suel-

do está definida como campo de tipo numérico, el usuario no podrá introducir fechas en él. El gestor devolverá un error y el usuario deberá actuar en consecuencia.

5.2. Integridad de entidad

Significa que cada entidad (tabla) deberá tener una clave principal válida. Por ejemplo, si se permite valores nulos (null) para esa clave principal, obviamente no se podrán conectar otras tablas a esta fila. Que ocurra esto es muy difícil, pues ningún SGBD permite que si el campo es clave, pueda contener un valor null. A la hora de crear la tabla y a la hora de introducir datos, obligan a cumplimentar dicho campo.

5.3. Integridad referencial

Es una regla de integridad que se encarga de asegurar que las distintas relaciones entre tablas tengan siempre validez. Por ejemplo, tenemos una tabla llamada Vendedores y otra llamada Ventas. En la tabla Vendedores el campo clave es el campo DNI. En la tabla Ventas habrá que tener un campo DNI para conocer qué empleado ha realizado la venta, además de ser el campo que relaciona esta tabla con la de Vendedores.

Tomemos como ejemplo que, a la hora de dar de alta una venta (tabla Ventas), el usuario, sin querer, introduzca un DNI de vendedor que no exista en la tabla de Vendedores. A la hora de pedir información de quién efectuó esa venta, no podríamos conocer los datos del vendedor, más que nada porque dicho vendedor no existe.

Si exigimos integridad referencial, cuando procedemos a introducir un DNI en la tabla de Ventas que no está dado de alta en la tabla de Vendedores, el gestor envía automáticamente un mensaje indicando el error de integridad. Lo mismo pasaría si damos de baja a un vendedor en la tabla de Vendedores.

¿Qué sucedería con sus ventas? Algunos gestores tienen la posibilidad de activar la opción de borrado o actualización en cascada. Si se borra o modifica la clave principal, todos sus registros de la tabla relacionada se borrarán o modificarán de forma automática.

SQL Server no soporta actualizaciones en cascada porque usa un mecanismo aún mejor: **desencadenantes**. Un desencadenante es un procedimiento que se invoca automáticamente, como un evento. Por ejemplo, si un vendedor realiza una nueva venta, se puede usar un desencadenante para que el importe de la misma se acumule en el campo total_venta_acumulada de la tabla Vendedores.

Un desencadenante es un conjunto de sentencias de programación, como son los bucles, las sentencias de control de flujo, las variables, etc., mezclados con SQL.

Como es de imaginar, la integridad referencial se debe indicar antes de introducir la información, en caso contrario, nos podría dar bastantes errores de datos que no coincidan en ambas tablas, errores que el usuario, con paciencia, debe solucionar.

6 El modelo físico relacional

6.1. Introducción

El diseño de una base de datos se descompone en tres etapas: diseño conceptual, lógico y físico. La etapa del diseño lógico es independiente de los detalles de implementación y dependiente del tipo de SGBD que se vaya a utilizar. La salida de esta etapa es el esquema lógico global y la documentación que lo describe. Todo ello es la entrada para la etapa que viene a continuación, el diseño físico.

Mientras que en el diseño lógico se especifica qué se guarda, en el diseño físico se especifica cómo se guarda. Para ello, el diseñador debe conocer muy bien toda la funcionalidad del SGBD concreto que se vaya a utilizar y también el sistema informático sobre el que este va a trabajar. El diseño físico no es una etapa aislada, ya que algunas decisiones que se tomen durante su desarrollo, por ejemplo para mejorar las prestaciones, pueden provocar una reestructuración del esquema lógico.

6.2. Metodología de diseño físico para bases de datos relacionales

El objetivo de esta etapa es producir una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción incluye las estructuras de almacenamiento y los métodos de acceso que se utilizarán para conseguir un acceso eficiente a los datos.

El diseño físico se divide en cuatro fases, cada una de ellas compuesta por una serie de pasos:

1) Traducir el esquema lógico global para el SGBD específico.
2) Diseñar la representación física.
    - Analizar las transacciones.
    - Escoger las organizaciones de ficheros.
    - Escoger los índices secundarios.
    - Considerar la introducción de redundancias controladas.
    - Estimar la necesidad de espacio en disco.
3) Diseñar los mecanismos de seguridad.
    - Diseñar las vistas de los usuarios.
    - Diseñar las reglas de acceso.
4) Monitorizar y afinar el sistema.

6.2.1. Traducir el esquema lógico global

La primera fase del diseño lógico consiste en traducir el esquema lógico global en un esquema que se pueda implementar en el SGBD escogido. Para ello, es necesario conocer toda la funcionalidad que este ofrece. Por ejemplo, el diseñador deberá saber:

— Si el sistema soporta la definición de claves primarias, claves ajenas y claves alternativas.

— Si el sistema soporta la definición de datos requeridos (es decir, si se pueden definir atributos como no nulos).

— Si el sistema soporta la definición de dominios.

— Si el sistema soporta la definición de reglas de negocio.

— Cómo se crean las relaciones base.

• **Diseñar las relaciones base para el SGBD específico**

Las relaciones base se definen mediante el lenguaje de definición de datos del SGBD. Para ello, se utiliza la información producida durante el diseño lógico: el esquema lógico global y el diccionario de datos. El esquema lógico consta de un conjunto de relaciones y, para cada una de ellas, se tiene:

— El nombre de la relación.

— La lista de atributos, entre paréntesis.

— La clave primaria y las claves ajenas, si las tiene.

— Las reglas de integridad de las claves ajenas.

En el diccionario de datos se describen los atributos y, para cada uno de ellos, se tiene:

— Su dominio: tipo de datos, longitud y restricciones de dominio.

— El valor por defecto, que es opcional.

— Si admite nulos.

— Si es derivado y, en caso de serlo, cómo se calcula su valor.

A continuación, se muestra un ejemplo de la definición de la relación INMUEBLE con el estándar SQL.

```
CREATE DOMAIN pnum      AS VARCHAR(5);
CREATE DOMAIN enum      AS VARCHAR(5);
CREATE DOMAIN onum      AS VARCHAR(3);
CREATE DOMAIN inum      AS VARCHAR(5);
CREATE DOMAIN calle     AS VARCHAR(25);
CREATE DOMAIN area      AS VARCHAR(15);
```

```
CREATE DOMAIN poblacion AS VARCHAR(15);
CREATE DOMAIN tipo      AS VARCHAR(1)
       CHECK(VALUE IN ('A',`C',`D',`P',`V'));
CREATE DOMAIN hab       AS SMALLINT
       CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN alquiler  AS DECIMAL(6,2)
       CHECK(VALUE BETWEEN 0 AND 9999);

CREATE TABLE inmueble (
   inum        INUM          NOT NULL,
   calle       CALLE         NOT NULL,
   area        AREA,
   poblacion   POBLACION     NOT NULL,
   tipo        TIPO          NOT NULL DEFAULT `P',
   hab         HAB           NOT NULL DEFAULT 4,
   alquiler    ALQUILER      NOT NULL DEFAULT 350,
   pnum        PNUM          NOT NULL,
   enum        ENUM,
   onum        ONUM          NOT NULL,
   PRIMARY KEY (inum),
   FOREIGN KEY (pnum) REFERENCES propietario
               ON DELETE no action ON UPDATE cascade,
   FOREIGN KEY (enum) REFERENCES plantilla
               ON DELETE set null  ON UPDATE cascade,
   FOREIGN KEY (onum) REFERENCES oficina
               ON DELETE no action ON UPDATE cascade
   );
```

- **Diseñar las reglas de negocio para el SGBD específico**

Las actualizaciones que se realizan sobre las relaciones de la base de datos deben observar ciertas restricciones que imponen las reglas de negocio de la empresa. Algunos SGBD proporcionan mecanismos que permiten definir estas restricciones y vigilan que no se violen.

Por ejemplo, si no se quiere que un empleado tenga más de diez inmuebles asignados, se puede definir una restricción en la sentencia CREATE TABLE de la relación INMUEBLE:

```
CONSTRAINT inmuebles_por_empleado
        CHECK (NOT EXISTS (SELECT enum
                           FROM   inmueble
                           GROUP BY enum
                           HAVING COUNT(*)>10))
```

Otro modo de definir esta restricción es mediante un disparador (trigger):

```
CREATE TRIGGER inmuebles_por_empleado
     ON inmueble
     FOR INSERT,UPDATE
     AS IF ((SELECT COUNT(*)
               FROM inmueble i
               WHERE i.inum=INSERTED.inum)>10)
            BEGIN
               PRINT "Este empleado ya tiene 10 inmuebles asignados"
               ROLLBACK TRANSACTION
            END
```

Hay algunas restricciones que no las pueden manejar los SGBD, como por ejemplo "a las 20:30 del último día laborable de cada año archivar los inmuebles vendidos y borrarlos". Para estas restricciones habrá que escribir programas de aplicación específicos. Por otro lado, hay SGBD que no permiten la definición de restricciones, por lo que estas deberán incluirse en los programas de aplicación.

Todas las restricciones que se definan deben estar documentadas. Si hay varias opciones posibles para implementarlas, hay que explicar por qué se ha escogido la opción implementada.

6.2.2. Diseñar la representación física

Uno de los objetivos principales del diseño físico es almacenar los datos de modo eficiente. Para medir la eficiencia hay varios factores que se deben tener en cuenta:

— **Productividad de transacciones.** Es el número de transacciones que se quiere procesar en un intervalo de tiempo.

— **Tiempo de respuesta.** Es el tiempo que tarda en ejecutarse una transacción. Desde el punto de vista del usuario, este tiempo debería ser el mínimo posible.

— **Espacio en disco.** Es la cantidad de espacio en disco que hace falta para los ficheros de la base de datos. Normalmente, el diseñador querrá minimizar este espacio.

Lo que suele suceder es que todos estos factores no se pueden satisfacer a la vez. Por ejemplo, para conseguir un tiempo de respuesta mínimo, puede ser necesario aumentar la cantidad de datos almacenados, ocupando más espacio en disco. Por lo tanto, el diseñador deberá ir ajustando estos factores para conseguir un equilibrio razonable. El diseño físico inicial no será el definitivo, sino que habrá que ir monitorizándolo para observar sus prestaciones e ir ajustándolo como sea oportuno. Muchos SGBD proporcionan herramientas para monitorizar y afinar el sistema.

Hay algunas estructuras de almacenamiento que son muy eficientes para cargar grandes cantidades de datos en la base de datos, pero no son eficientes para el resto de operaciones, por lo que se puede escoger dicha estructura de almacenamiento para inicializar la base de datos y cambiarla, a continuación,

para su posterior operación. Los tipos de organizaciones de ficheros disponibles varían en cada SGBD. Algunos sistemas proporcionan más estructuras de almacenamiento que otros. Es muy importante que el diseñador del esquema físico sepa qué estructuras de almacenamiento le proporciona el SGBD y cómo las utiliza.

Para mejorar las prestaciones, el diseñador del esquema físico debe saber cómo interactúan los dispositivos involucrados y cómo esto afecta a las prestaciones:

— **Memoria principal.** Los accesos a memoria principal son mucho más rápidos que los accesos a memoria secundaria (decenas o centenas de miles de veces más rápidos). Generalmente, cuanta más memoria principal se tenga, más rápidas serán las aplicaciones. Sin embargo, es aconsejable tener al menos un 5% de la memoria disponible, pero no más de un 10%. Si no hay bastante memoria disponible para todos los procesos, el sistema operativo debe transferir páginas a disco para liberar memoria (paging). Cuando estas páginas se vuelven a necesitar, hay que volver a traerlas desde el disco (faltas de página). A veces, es necesario llevar procesos enteros a disco (swapping) para liberar memoria. El hacer estas transferencias con demasiada frecuencia empeora las prestaciones.

— **CPU.** La CPU controla los recursos del sistema y ejecuta los procesos de usuario. El principal objetivo con este dispositivo es lograr que no haya bloqueos de procesos para conseguirla. Si el sistema operativo, o los procesos de los usuarios, hacen muchas demandas de CPU, esta se convierte en un cuello de botella. Esto suele ocurrir cuando hay muchas faltas de página o se realiza mucho swapping.

— **Entrada/salida a disco.** Los discos tienen una velocidad de entrada/salida. Cuando se requieren datos a una velocidad mayor que esta, el disco se convierte en un cuello de botella. Dependiendo de cómo se organicen los datos en el disco, se conseguirá reducir la probabilidad de empeorar las prestaciones. Los principios básicos que se deberían seguir para repartir los datos en los discos son los siguientes:

  • Los ficheros del sistema operativo deben estar separados de los ficheros de la base de datos.

  • Los ficheros de datos deben estar separados de los ficheros de índices.

  • Los ficheros con los diarios de operaciones deben estar separados del resto de los ficheros de la base de datos.

— **Red.** La red se convierte en un cuello de botella cuando tiene mucho tráfico y cuando hay muchas colisiones.

Cada uno de estos recursos afecta a los demás, de modo que una mejora en alguno de ellos puede provocar mejoras en otros.

• **Analizar las transacciones**

Para realizar un buen diseño físico es necesario conocer las consultas y las transacciones que se van a ejecutar sobre la base de datos. Esto incluye tanto información cualitativa, como cuantitativa. Para cada transacción, hay que especificar:

— La frecuencia con que se va a ejecutar.

— Las relaciones y los atributos a los que accede la transacción y el tipo de acceso: consulta, inserción, modificación o eliminación. Los atributos que se modifican no son buenos candidatos para construir estructuras de acceso.

— Los atributos que se utilizan en los predicados del WHERE de las sentencias SQL. Estos atributos pueden ser candidatos para construir estructuras de acceso dependiendo del tipo de predicado que se utilice.

— Si es una consulta, los atributos involucrados en el JOIN de dos o más relaciones. Estos atributos pueden ser candidatos para construir estructuras de acceso.

— Las restricciones temporales impuestas sobre la transacción. Los atributos utilizados en los predicados de la transacción pueden ser candidatos para construir estructuras de acceso.

• **Escoger las organizaciones de ficheros**

El objetivo de este paso es escoger la organización de ficheros óptima para cada relación. Por ejemplo, un fichero desordenado es una buena estructura cuando se va a cargar gran cantidad de datos en una relación al inicializarla, cuando la relación tiene pocas tuplas; también cuando en cada acceso se deben obtener todas las tuplas de la relación, o cuando la relación tiene una estructura de acceso adicional, como puede ser un índice. Por otra parte, los ficheros dispersos (hashing) son apropiados cuando se accede a las tuplas a través de los valores exactos de alguno de sus campos (condición de igualdad en el WHERE). Si la condición de búsqueda es distinta de la igualdad (búsqueda por rango, por patrón, etc.), la dispersión no es una buena opción. Hay otras organizaciones, como la ISAM, InnoDB para MySQL o los árboles B+ para Oracle.

Las organizaciones de ficheros elegidas deben documentarse, justificando en cada caso la opción escogida.

• **Escoger los índices secundarios**

Los índices secundarios permiten especificar caminos de acceso adicionales para las relaciones base. Por ejemplo, la relación INMUEBLE se puede haber almacenado en un fichero disperso a través del atributo inum. Si se accede a menudo a esta relación a través del atributo "alquiler", se puede plantear la creación de un índice sobre dicho atributo para favorecer estos accesos. Pero hay que tener en cuenta que estos índices conllevan un coste de mantenimiento que hay que sopesar frente a la ganancia en prestaciones. A la hora de seleccionar los índices, se pueden seguir las siguientes indicaciones:

— Construir un índice sobre la clave primaria de cada relación base.

— No crear índices sobre relaciones pequeñas.

— Añadir un índice sobre los atributos que se utilizan para acceder con mucha frecuencia.

— Añadir un índice sobre las claves ajenas que se utilicen con frecuencia para hacer joins.

— Evitar los índices sobre atributos que se modifican a menudo.

— Evitar los índices sobre atributos poco selectivos (aquellos en los que la consulta selecciona una porción significativa de la relación).

— Evitar los índices sobre atributos formados por tiras de caracteres largas.

Los índices creados se deben documentar, explicando las razones de su elección.

• **Considerar la introducción de redundancias controladas**

En ocasiones puede ser conveniente relajar las reglas de normalización introduciendo redundancias de forma controlada, con objeto de mejorar las prestaciones del sistema. En la etapa del diseño lógico se recomienda llegar, al menos, hasta la tercera forma normal para obtener un esquema con una estructura consistente y sin redundancias. Pero, a menudo, sucede que las bases de datos así normalizadas no proporcionan la máxima eficiencia, con lo que es necesario volver atrás y desnormalizar algunas relaciones, sacrificando los beneficios de la normalización para mejorar las prestaciones. Es importante hacer notar que la desnormalización solo debe realizarse cuando se estime que el sistema no puede alcanzar las prestaciones deseadas. Y, desde luego, la necesidad de desnormalizar en ocasiones no implica eliminar la normalización del diseño lógico: la normalización obliga al diseñador a entender completamente cada uno de los atributos que se han de representar en la base de datos. Por lo tanto, hay que tener en cuenta los siguientes factores:

— La desnormalización hace que la implementación sea más compleja.

— La desnormalización hace que se sacrifique la flexibilidad.

— La desnormalización puede hacer que los accesos a datos sean más rápidos, pero ralentiza las actualizaciones.

Por regla general, la desnormalización de una relación puede ser una opción viable cuando las prestaciones que se obtienen no son las deseadas y la relación se actualiza con poca frecuencia, pero se consulta muy a menudo. Las redundancias que se pueden incluir al desnormalizar son de varios tipos: se pueden introducir datos derivados (calculados a partir de otros datos), se pueden duplicar atributos o se pueden hacer joins de relaciones.

El incluir un atributo derivado dependerá del coste adicional de almacenarlo y mantenerlo consistente con los datos de los que se deriva, frente al coste de calcularlo cada vez que se necesita.

No se pueden establecer una serie de reglas que determinen cuándo desnormalizar relaciones, pero hay algunas situaciones muy comunes en donde puede considerarse esta posibilidad:

— **Combinar relaciones de uno a uno.** Cuando hay relaciones (tablas) involucradas en relaciones de uno a uno, se accede a ellas de manera conjunta con frecuencia y casi no se les accede separadamente, se pueden combinar en una sola relación (tabla).

— **Duplicar atributos no clave en relaciones de uno a muchos para reducir los joins.** Para evitar operaciones de join, se pueden incluir atributos de la relación (tabla) padre en la relación (tabla) hijo de las relaciones de uno a muchos.

— **Tablas de referencia.** Las tablas de referencia (lookup) son listas de valores, cada uno de los cuales tiene un código. Por ejemplo, puede haber una tabla de referencia para los tipos de inmueble, con las descripciones de estos tipos y un código asociado. Este tipo de tablas son un caso de relación de uno a muchos. En la relación INMUEBLE habrá una clave ajena a esta tabla para indicar el tipo de inmueble. De este modo, es muy fácil validar los datos, además de que se ahorra espacio escribiendo solo el código y no la descripción para cada inmueble, además de ahorrar tiempo cuando se actualizan las descripciones. Si las tablas de referencia se utilizan a menudo en consultas críticas, se puede considerar la introducción de la descripción junto con el código en la relación (tabla) hijo, manteniendo la tabla de referencia para validación de datos.

— **Duplicar claves ajenas en relaciones de uno a muchos para reducir los joins.** Para evitar operaciones de join, se pueden incluir claves ajenas de una relación (tabla) en otra relación (tabla) con la que se relaciona (habrá que tener en cuenta ciertas restricciones).

— **Duplicar atributos en relaciones de muchos a muchos para reducir los joins.** Durante el diseño lógico se eliminan las relaciones de muchos a muchos introduciendo dos relaciones de uno a muchos. Esto hace que aparezca una nueva relación (tabla) intermedia, de modo que si se quiere obtener la información de la relación de muchos a muchos, se tiene que realizar el join de tres relaciones (tablas). Para evitar algunos de estos joins se pueden incluir algunos de los atributos de las relaciones (tablas) originales en la relación (tabla) intermedia.

— **Introducir grupos repetitivos.** Los grupos repetitivos se eliminan en el primer paso de la normalización para conseguir la primera forma normal. Estos grupos se eliminan introduciendo una nueva relación (tabla), generando una relación de uno a muchos. A veces, puede ser conveniente reintroducir los grupos repetitivos para mejorar las prestaciones.

— **Creación de Vistas Materializadas.** Son vistas sobre tablas muy grandes en las que hay que definir y/o programar cuándo han de refrescarse para que sean consistentes los datos redundantes con los originales. Su razón de ser es la de otorgar velocidad de acceso ante una tabla con muchas filas (millones de filas) y sobre la que hay que realizar una consulta muy costosa en cuanto a ordenaciones o condiciones a cumplir cuya ejecución en tiempo real lo hace inviable. De este modo se accede a una tabla que es una redundancia de una tabla y que cumple unas ciertas condiciones.

Todas las redundancias que se introduzcan en este paso se deben documentar y razonar. El esquema lógico se debe actualizar para reflejar los cambios introducidos.

- **Estimar la necesidad de espacio en disco**

En caso de que se tenga que adquirir nuevo equipamiento informático, el diseñador debe estimar el espacio necesario en disco para la base de datos. Esta estimación depende del SGBD que se vaya a utilizar y del hardware. En general, se debe estimar el número de tuplas de cada relación y su tamaño. También se debe estimar el factor de crecimiento de cada relación.

6.2.3. Diseñar los mecanismos de seguridad

Los datos constituyen un recurso esencial para la empresa, por lo tanto su seguridad es de vital importancia. Durante el diseño lógico se habrán especificado los requerimientos en cuanto a seguridad que en esta fase se deben implementar. Para llevar a cabo esta implementación, el diseñador debe conocer las posibilidades que ofrece el SGBD que se vaya a utilizar.

- **Diseñar las vistas de los usuarios**

El objetivo de este paso es diseñar las vistas de los usuarios correspondientes a los esquemas lógicos locales. Las vistas, además de preservar la seguridad, mejoran la independencia de datos, reducen la complejidad y permiten que los usuarios vean los datos en el formato deseado.

- **Diseñar las reglas de acceso**

El administrador de la base de datos asigna a cada usuario un identificador que tendrá una palabra secreta asociada por motivos de seguridad. Para cada usuario o grupo de usuarios se otorgarán permisos para realizar determinadas acciones sobre determinados objetos de la base de datos. Por ejemplo, los usuarios de un determinado grupo pueden tener permiso para consultar los datos de una relación base concreta y no tener permiso para actualizarlos.

6.2.4. Monitorizar y afinar el sistema

Una vez implementado el esquema físico de la base de datos, se debe poner en marcha para observar sus prestaciones. Si estas no son las deseadas, el esquema deberá cambiar para intentar satisfacerlas. Una vez afinado el esquema, no permanecerá estático, ya que tendrá que ir cambiando conforme lo requieran los nuevos requisitos de los usuarios. Los SGBD proporcionan herramientas para monitorizar el sistema mientras está en funcionamiento.