Tercera Parte

Planificación, Análisis y Diseño de Sistemas

Nota:

Al desarrollar un sistema o planificar una arquitectura de sistemas, las actividades de análisis y diseño se cumplen simultáneamente para procesos y para datos; sin embargo, con fines didácticos, en esta parte del libro, hemos separado el análisis y diseño de procesos del análisis y diseño de datos.

Capítulo XVII

Arquitectura de Datos

Arquitectura de datos
Tabla de contenido

Arquitectura de datos

1.- Las Funciones de manejo de datos

Las funciones para el manejo de datos -data management- en los computadores de segunda generación eran sumamente rudimentarias. Dado que los programas de aplicación tenían que hacer referencia directa al dispositivo donde se alojaban los datos a procesar, prácticamente todo el manejo de archivos debía construirse dentro de los programas. La tercera generación -junto con los nuevos sistemas operativos- permitió funciones mucho más sofisticadas para el manejo de datos y, dentro de ciertos límites, lograron que los programas de aplicación se independizaran de los dispositivos físicos.

Con las nuevas facilidades, a partir de la tercera generación, al hacer operaciones de lectura y escritura (READ, WRITE), los programas se refieren a un archivo por su nombre, en lugar de hacerlo al dispositivo que los almacena, por lo que es posible cambiar el dispositivo que almacena un archivo, sin necesidad de modificar el programa que lo procesa. De igual forma pueden modificarse algunas características físicas de los archivos, como su factor de bloqueo, sin que ello implique tener que modificar los programas de aplicación.

¿Qué importancia tiene esta independencia de los programas? Una muy grande: una instalación puede crecer (cambiar dispositivos) sin necesidad de cambiar drásticamente su cartera de aplicaciones, es decir, minimizando el mantenimiento.

A partir de la tercera generación, muchas de las funciones de manejo de datos pasaron a formar parte del sistema operativo. Para ello los sistemas operativos disponen de métodos de acceso que, antes de iniciarse la ejecución de un programa, permiten establecer la conexión entre el “archivo lógico” al cual el programa hace referencia y el “archivo físico” que se encuentra almacenado (o se desea almacenar) en algún dispositivo. Así pues, se puede hacer un cambio en la estructura física de los archivos sin afectar las aplicaciones.

A pesar de que el sistema operativo independiza los programas de algunas características físicas de los archivos, la independencia no es total. No es posible, por ejemplo, añadir un nuevo campo a un registro sin afectar todos los programas que lo utilizan y mucho menos cambiar la estrategia de acceso -secuencial o al azar- sin impactar el diseño mismo de las aplicaciones.

¿Para qué queremos independizar los programas de la estrategia de acceso a los datos? Para que los sistemas de información puedan crecer, para poder utilizar la información almacenada en sus archivos en diferentes aplicaciones y para que la información pueda ser compartida en forma flexible por diferentes usuarios.

Deseamos destacar que no estamos hablando de un problema de interés académico únicamente o de un deseo fantasioso. Se trata de una necesidad perentoria: no es posible construir sistemas de información integrales sin una arquitectura de datos flexible y no construir sistemas de información integrales significa negarle a la empresa la posibilidad de utilizar su información, no sólo para elaborar reportes, sino también para derivar ventajas para el negocio en el uso de la informática.

Sin una arquitectura de datos flexible, el recurso informático difícilmente puede ser utilizado más allá de mecanizar algunos procesos manuales en forma aislada, lo cual sólo puede representar algunos ahorros. Sin embargo, incorporar la informática en la "vida del negocio" puede representar mucho más que ahorros: puede implicar beneficios adicionales o ventajas sobre la competencia.

Casi simultáneamente con la aparición de la tercera generación, aparece el interés por los sistemas totales, como se les llamó al comienzo, o sistemas de información gerencial, como se les ha llamado después. En la raíz de esos conceptos, su propósito primario siempre fue poner a disposición de todos los niveles de la empresa el uso de la informática, convertir la información en un recurso corporativo. Más de treinta años de sonados fracasos y algunos éxitos nos han enseñado, entre otras cosas, que la columna vertebral de estos sistemas totales o integrales o gerenciales, como se les desee llamar, está en el diseño de la arquitectura de datos y, dentro de ella, el software para manejar las bases de datos constituye la médula.

2.- Objetivos de un sistema de bases de datos

Un sistema de manejo o gestión de bases de datos tiene, pues, dos objetivos fundamentales: por una parte, almacenar datos en forma flexible de tal forma que cualquier usuario o aplicación pueda servirse de

ellos y, por otra parte, lo que se ha denominado “independencia de los datos”, que no es otra cosa que independizar las aplicaciones de la estructura física de los datos, de tal manera que ese gran reservorio pueda crecer sin afectar los programas existentes y viceversa, que la base de datos no requiera modificaciones cada vez que hay modificaciones en los sistemas que se sirven de ella.

Un sistema manejador de bases de datos, de acuerdo con la definición que C. J. Date nos da en su libro “An Introduction to Database Systems”, es un sistema computarizado que mantiene registros, esto es, un sistema cuyo propósito principal es mantener información para ponerla a disposición de todos aquellos que la requieran. El mismo Date apunta que esa información almacenada puede ser cualquier tipo de información relevante para la organización y para los individuos que se sirven de la base de datos.

Dado que estos sistemas de manejo y administración de base de datos, además de tener como objetivo fundamental almacenar y diseminar datos, buscan facilitar y simplificar las tareas de desarrollo y mantenimiento de sistemas de información, puede afirmarse que sus objetivos generales son:

- Disminuir la redundancia de datos

  En los sistemas tradicionales, dado que cada aplicación tiene sus propios archivos privados, es común que un mismo dato (por ejemplo, el sueldo de los empleados) se encuentre repetido en diferentes archivos (archivo de nómina, archivo de personal, etc.). En un ambiente de bases de datos, las redundancias de este tipo pueden ser minimizadas, con las consecuentes ventajas de simplicidad y flexibilidad en el uso y búsqueda de la información.

- Eliminar la inconsistencia de datos

  Si bien lo ideal es que no exista ningún tipo de redundancia, algunas veces no es posible; sin embargo, la tecnología de bases de datos brinda facilidades que pueden ser utilizadas por las aplicaciones con el fin de evitar que los datos redundantes sean inconsistentes de tal forma que si, por ejemplo, el dato SUELDO-EMPLEADO es redundante en dos o más tipos de registro, sus valores coincidan, es decir, que cada empleado aparezca con el mismo sueldo en todos los registros que presenten esa redundancia.

- Compartir datos entre múltiples usuarios

  Un ambiente de bases de datos ofrece facilidades que permiten que la información sea compartida por diversos usuarios y aplicaciones, aún cuando existan requerimientos diferentes o conflictivos.

- Establecer normas de seguridad

  Las facilidades que brinda un manejador de bases de datos permiten aplicar restricciones de seguridad para tener acceso la información, con el fin de proteger la confidencialidad de la misma.

- Proteger la integridad de los datos

  De manera análoga, un manejador de bases de datos ofrece facilidades para proteger la integridad de los datos, evitando que puedan ser destruidos accidental o intencionalmente, y evitando que puedan ser alterados por personas no autorizadas.

- Independizar los programas de las estructuras de datos

  La tecnología de bases de datos busca independizar las aplicaciones de la estrategia de acceso a los datos, de tal forma que pueda modificarse una base de datos (para incluir nuevos datos y registros) sin que ello requiera modificar los programas de aplicación que utilizan esa base datos.

- Establecer estándares y procedimientos

  Un sistema de manejo o gestión de bases de datos permite establecer estándares y normas para el uso de los datos y brinda facilidades para que puedan ponerse en práctica procedimientos generales que regulen el acceso, el uso y la representación de los datos.

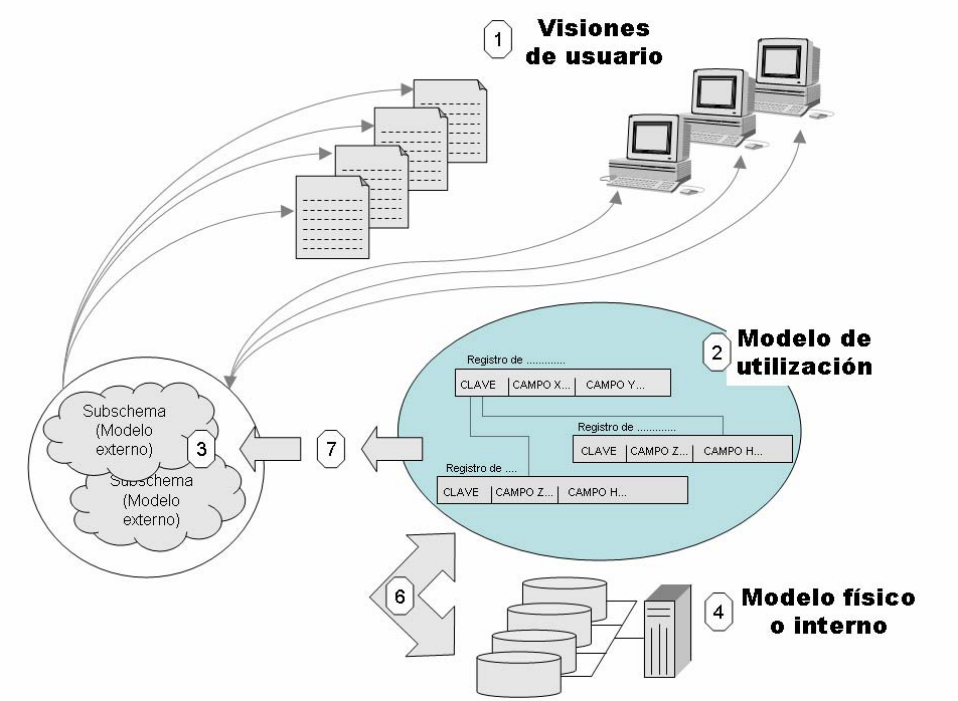
Muchos de esos objetivos no se alcanzan en su totalidad, aún con la tecnología de bases de datos disponible hoy día; sin embargo, la mayoría de los sistemas de manejo de bases de datos existentes en el mercado (DB/2, ORACLE, SQL-Server, etc.) ofrecen facilidades que pueden ser utilizadas por los sistemas de información para alcanzar esos objetivos, no a la perfección, pero sí en un grado razonable.

3.- Estructura de un sistema manejador de bases de datos

Hasta hace poco tiempo, existía una gran variedad de sistemas para el manejo de bases de datos que funcionaban bajo diferentes “modelos”; algunos de ellos organizaban los datos en forma de árbol -modelo

jerárquico-, otros interrelacionaban los registros de datos en forma de red -modelo de redes-. Hoy día, la escena ha sido tomada por el modelo relacional y su lenguaje SQL -structured query language-, el cual se ha convertido en el lenguaje universal para el manejo de las bases de datos, aunque cada fabricante lo implementa con algunas diferencias.

Todos los diferentes sistemas para el manejo de bases de datos que existen en el mercado, independientemente de la forma en que operen, se componen de un conjunto de elementos que a continuación describimos:



1. Las visiones de usuario

En el mundo de los usuarios finales, los datos almacenados en la base de datos son percibidos de manera diferente por cada usuario, de acuerdo con la aplicación o procedimiento en uso. Cada uno de ellos percibe los datos almacenados en la base de datos a través de reportes y formatos de pantalla que han sido creados para satisfacer sus necesidades de información. Los programas de aplicación interactúan con el

manejador de bases de datos y con la comunidad de usuarios, con el fin de satisfacer las necesidades de información particulares de cada uno de los miembros de esa comunidad.

La base de datos es un conjunto, pero cada usuario, al realizar una tarea específica, sólo utiliza o requiere una parte de ese conjunto -para analizar el estado de las cuentas por cobrar, sólo es necesario utilizar la información pertinente de facturas pendientes-. Cada uno de estos subconjuntos de datos se denomina una "visión de usuario", para indicar que se trata de una forma particular de ver o percibir ese todo que es la base de datos.

2. El modelo de utilización

Todas y cada una de las diferentes visiones de usuario se derivan -o son extraídas- de una base de datos común, que el analista denomina base de datos lógica o schema. Realmente ese "gran reservorio de datos" no es un ente real, ya que los verdaderos registros se encuentran almacenados en diferentes archivos físicos, desde los cuales el manejador de bases de datos extrae los datos para dar vida aparente a ese schema.

El schema, a pesar de que sólo existe en los "buffers" o áreas de trabajo del manejador de bases de datos, es algo real para el analista de sistemas y sobre la base de ese modelo, crea los programas de aplicación que se encargarán de satisfacer las necesidades de información de los usuarios del sistema. El schema de los datos es, pues, la visión que el analista de sistemas tiene de la base de datos, es decir, el modelo que define la forma en que los datos serán utilizados, por lo que lo denominamos modelo de utilización de los datos.

3. El modelo externo

Cada aplicación, dado que realiza funciones particulares para un área del negocio, no necesita tener acceso a todos los tipos de registros contenidos en el modelo de utilización de los datos -schema-, sino, únicamente, a un subconjunto de éstos. Estos subconjuntos del modelo de utilización o subschemas, que están destinados a satisfacer las necesidades de un subsistema o aplicación, constituyen una "comunidad de visiones de usuario", conforman lo que se

denomina el modelo externo. Este subnivel es la visión que de la base de datos tienen los programas de aplicación; es decir, es el modelo de utilización de los datos en el nivel del programa.

Para cada aplicación, el manejador de bases de datos se encarga de buscar en los archivos físicos los registros requeridos, con el fin de acomodar las estructuras de datos que conforman el modelo de utilización de los datos y así, poner a disposición de la aplicación las estructuras del subschema requeridas para ensamblar las visiones de usuario correspondientes.

4. El modelo interno

Tal como señalábamos, la base de datos está almacenada en diferentes archivos físicos, sobre los cuales se han creado diferentes apuntadores, índices primarios, índices secundarios, etc., con la finalidad de que el manejador de bases de datos pueda satisfacer los requerimientos del sistema en general y de cada programa en particular. Esta estructura global de la base de datos física se denomina usualmente el modelo físico y pertenece al mundo del administrador de bases de datos.

El administrador de base de datos -DBA- es la persona o equipo de profesionales responsables del control y manejo del sistema de base de datos, generalmente debe tener experiencia en diseño de bases de datos, sistemas operativos, comunicación de datos y programación.

5. El sublenguaje de datos

Cada manejador de bases de datos tiene un grupo de comandos que se añaden al lenguaje de programación, con el fin de que los programas de aplicación puedan comunicarse con él para obtener, insertar, eliminar o reemplazar registros de la base de datos. Estos comandos constituyen el sublenguaje de datos del manejador, llamado así para implicar que forman parte de algún lenguaje que les sirve de huésped (host language). Hoy día, el lenguaje SQL se ha convertido en el estándar como sublenguaje de datos para la mayoría de los sistemas de bases de datos relacionales.

6. Traductor interno/utilización

Antes de que los programas de aplicación puedan obtener acceso a una base de datos, el administrador de bases de datos lleva a cabo un proceso de definición de la base de datos. Para ello, haciendo uso de ciertos comandos, el administrador describe para el manejador las características de la base de datos, sus registros y la forma en que los registros físicos serán presentados al sistema; es decir, le indicará al traductor cómo el modelo físico debe ser convertido a registros del modelo de utilización y viceversa. Hoy día, el lenguaje SQL incorpora todas estas funciones de creación y mantenimiento de bases de datos relacionales.

7. Traductor externo/utilización

Los subconjuntos del modelo de utilización o subschemas, como ya dijéramos, serán presentados por cada programa de aplicación a los diferentes usuarios del sistema. Las funciones de conversión del modelo de utilización a cada subschema y viceversa, son llevadas a cabo también por el manejador de bases de datos, haciendo uso de los comandos que el administrador de bases de datos ha definido para el traductor externo a utilización y utilización a externo.

El lenguaje SQL incluye estos comandos, con los que el administrador de bases de datos establece reglas de uso, pues no sólo especificará los registros y los datos a los que una aplicación puede tener acceso, sino también el nivel de autorización para modificarlos o consultarlos.

4.- Cuatro perspectivas para mirar los mismos datos

Analizando la arquitectura de un sistema manejador de bases de datos, hemos visto que existen cuatro perspectivas para mirar una misma base de datos:

- Las visiones de usuario, que constituyen la forma en que los diversos usuarios miran o perciben la base de datos.
- El nivel externo, compuesto por las estructuras destinadas a satisfacer las visiones de usuario y que constituye la perspectiva desde la cual, el programador ve la base de datos.
- El nivel de utilización de los datos, que es la perspectiva del analista de sistemas.

- El nivel interno o físico, que es la visión del administrador de bases de datos.

Desde el punto de vista del desarrollo de sistemas de información, el corazón de un sistema es el modelo de utilización de los datos, ya que este modelo define la estrategia de acceso a los mismos y, a partir de él, se pueden desarrollar tanto los reportes y pantallas que darán soporte a las funciones del negocio, como las estructuras físicas que el administrador de bases de datos creará para almacenar físicamente los datos. Indudablemente, la definición del modelo de utilización de los datos es el punto crucial en el diseño de un sistema de información.

Este modelo de utilización de los datos debe ser estructurado de tal forma que todas las funciones del negocio a las que el sistema de información debe dar soporte puedan obtener toda la información requerida. Esto sólo se logra cuando el modelo de utilización de los datos ha sido diseñado tomando en cuenta las características del universo de datos manejados por el área del negocio que el sistema debe servir. De allí que sea fundamental elaborar, antes de diseñar el modelo de utilización de los datos, un modelo que represente ese universo de datos manejados por dicha área del negocio, modelo al que denominamos modelo conceptual de datos.

La definición de ese modelo de datos del negocio o modelo conceptual se puede realizar de dos formas diferentes: partiendo de las visiones de usuario -método de abajo hacia arriba- o partiendo de la comprensión del negocio -método de arriba hacia abajo-. Estos métodos serán discutidos en el capítulo dedicado a análisis y diseño de sistemas.

5.- Normalización

Junto con la aparición de los sistemas de bases de datos y la necesidad de resolver problemas de interrelación de datos también apareció un nuevo problema: ¿cómo mantener la integridad “semántica” de las bases de datos? En otras palabras, ¿cómo evitar que, al manejarla, se dañe la información almacenada en una base de datos, no sólo desde el punto de vista físico, sino también de su significado o integridad “semántica”?

Este problema fue ampliamente tratado por E. F. Codd en diferentes trabajos desarrollados durante la década de los 70 y que dieron lugar a la teoría de la normalización, de la cual presentaremos un resumen en las páginas que siguen.

5.1.- Terminología

Antes de entrar de lleno en el tema de normalización es necesario definir algunos de los términos utilizados en este capítulo y en otros relacionados con análisis y diseño de datos.

5.1.1.- Entidad

Una entidad es algo acerca de lo cual guardamos información. Ese algo puede ser un objeto tangible, como es el caso de la entidad EMPLEADO, o puede ser de naturaleza totalmente conceptual, como es el caso de la entidad CARGO (conjunto de responsabilidades o tareas a las que un empleado puede ser asignado).

Una entidad puede estar "poblada" por uno o más objetos y cada uno de ellos se denomina una "ocurrencia". Por ejemplo, en una determinada empresa la entidad EMPLEADO está integrada por todas las personas que trabajan en ella y el empleado Juan Pérez vendría a ser una ocurrencia de esa entidad del negocio.

5.1.2.- Atributo, dato e información

Cada entidad tiene una serie de características o propiedades que denominamos ATRIBUTOS. Por ejemplo, cada ocurrencia de la entidad EMPLEADO tiene nombre, dirección, sexo, edad, fecha de ingreso a la empresa, etc.; todas esas características constituyen los atributos de la entidad empleado. Todos los atributos de una entidad del negocio se aplican o tienen el mismo significado para todas sus ocurrencias.

Los datos son símbolos matemáticos o de lenguaje o de cualquier otra naturaleza que se utilizan para describir los atributos de las entidades. La descripción de una característica particular de una ocurrencia específica dentro de una entidad es el valor del dato. Para cada ocurrencia de una entidad del negocio, cada atributo tiene un valor específico; Juan (ocurrencia de la entidad empleado) tiene una estatura (atributo) de 1,75 metros. Las palabras dato y atributo, a pesar de que tienen un significado diferente, se utilizan, normalmente, como sinónimos, costumbre que seguiremos en nuestros textos.

A los valores de los datos se les da una representación con el fin de poderlos comunicar. Para algunos datos, sus valores pueden ser representados de varias formas; la estatura de un empleado, por ejemplo, puede ser expresada en metros, centímetros, pies y hasta en términos relativos diciendo que es alta, normal o baja.

Los valores que un dato puede tomar varían dentro de un rango que constituye el dominio del dato, que no es sino el conjunto de todos los valores que un dato puede tomar.

La información se produce a partir de datos; un dato se convierte en información cuando toma significado para algún receptor. Utilizando un paralelo industrial, podríamos decir que los datos son la materia prima con la cual se produce información.

5.1.3.- Visiones de usuario

Una visión de usuario es, tal como señalábamos al discutir los conceptos de Base de Datos, la forma particular en que un usuario ve una base de datos. También llamada relación (modelo de bases de datos relacionales), una visión de usuario no es más que un conjunto de datos agrupados en cierta forma, como puede serlo un reporte o un formato de pantalla; es decir, una visión de usuario especifica una relación funcional entre un conjunto de datos.

Las visiones de usuario tienen dos componentes:

- Datos
- Relaciones entre esos datos

Estas visiones de usuario o relaciones son básicamente estructuras de datos como las que encontramos en los flujos y almacenamientos de datos representados en los diagramas de flujo de datos, o las que aparecen en un reporte, como el siguiente:

CODIGO	DESCRIPCION	PRECIO UNITARIO	CANTIDAD
123	Tornillo	0,25	1.756
132	Tuerca	0,35	1.836
143	Destornillador	1,00	2.956

5.2.- Notación

Con el fin de simplificar los ejemplos que se irán discutiendo en este capítulo, se utilizará la siguiente notación para describir estructuras de datos:

=	Está compuesto de
+	Y
< >	Repetición de
[ ]	Opcional
**	Comentario

Veamos algunos ejemplos:

FACTURA =

NO-FACTURA + FECHA + NOMBRE-CLIENTE + DIRECCION-CLIENTE + *1a10*<DETALLE-ITEM> + [DESCUENTO] + TOTAL-FACTURA

DETALLE-ITEM =

DESCRIPCION-ITEM + CODIGO ITEM + CANTIDAD + PRECIO-UNITARIO + MONTO

PAGO =

NO-FACTURA + [NUMERO-CHEQUE]+ MONTO-PAGO

5.3.- Normalización

Una base de datos estructurada a imagen y semejanza de las visiones de usuarios, ofrecería problemas de “redundancia” e “inconsistencia” de los datos almacenados, además de que presentaría lo que E. F. Codd denominó “anomalías de actualización”.

Por ejemplo, si en nuestra base de datos existiera un registro de ORDENES DE COMPRA DE MATERIALES como el siguiente:

ORDEN-DE-COMPRA =

NUMERO-DE-ORDEN +
NUMERO-DE-PROVEEDOR+
NOMBRE-DEL-PROVEEDOR +
DIRECCION-DEL-PROVEEDOR +
FECHA-DE-LA-ORDEN +
CODIGO-DE-MATERIAL +
DESCRIPCION-MATERIAL +
PRECIO-UNITARIO +
CANTIDAD +
MONTO

En este registro, la descripción del material, el nombre del proveedor y su dirección se estarían repitiendo innecesariamente para cada orden. Además de la “redundancia”, también se estaría corriendo el riesgo de perder información. Por ejemplo, si sólo existiera una orden para un determinado material y esa orden fuese cancelada, al eliminar el registro de la orden también estaríamos eliminando la información sobre el material -descripción y precio-; por así decirlo, los datos acerca de ese material “desaparecerían” de nuestra base de datos. Casos como éste son los que E. F. Codd denominó anomalías de actualización, que, en efecto, son anomalías, pues no es razonable que al eliminar la información de “una cosa” -orden de compra- se afecte la información de “otra cosa” -material-.

La idea de “cosa acerca de la cual se maneja información” es una de las ideas centrales del análisis y diseño de datos; de hecho, como puede verse en los capítulos de este libro dedicados al tema, todo el proceso de análisis de datos puede resumirse diciendo que consiste en “identificar las entidades”, entendiéndose por entidades las “cosas” acerca de las cuales una base de datos almacena información. Esas entidades o “cosas” pueden ser objetos, como es el caso de las facturas; eventos, como es el caso de las tareas de un plan de trabajo; o conceptos, como es el caso de las responsabilidades asignadas a un cargo.

La teoría de la normalización establece el marco teórico bajo el cual debe trabajar el analista de sistemas para identificar las entidades que intervienen en un sistema y para diseñar bases de datos sin redundancia de datos y sin anomalías de actualización. Básicamente, la normalización pone de relieve los problemas que pueden presentarse cuando no se estudian todos los hechos y se organizan convenientemente, señalando los tipos de cosas que pueden ir mal si no se toma en cuenta ese marco teórico.

En las páginas que siguen se irán presentando los conceptos básicos y se discutirán los aspectos fundamentales de la teoría de la normalización.

5.4.- Clave de una estructura de datos

Toda visión de usuario o estructura de datos contiene un campo que denominamos clave de la estructura de datos y que distingue cada ocurrencia de la estructura de datos. A veces la clave es un único dato, como, por ejemplo, NUMERO-DE-ORDEN identifica cada ocurrencia de ORDEN-DE-COMPRA; en otros casos, es la combinación de varios datos la que permite identificar unívocamente cada ocurrencia, como sucede con NUMERO-DE-ORDEN + CODIGO-DE-MATERIAL que identifican cada ocurrencia de la estructura de datos LINEA-DE-ORDEN-DE-COMPRA.

En una estructura de datos, un dato debe cumplir dos condiciones para poder ser considerado clave:

- En cada ocurrencia el valor de la clave identifica la ocurrencia.
- Ningún dato, de los que componen la clave, puede ser descartado sin que se pierda la propiedad de identificar unívocamente la estructura de datos.

En una estructura de datos puede haber más de una clave. Todas ellas se denominan claves candidatas y de entre ellas puede seleccionarse una

como clave primaria, la cual será la que normalmente se utilice para identificar cada ocurrencia.

La clave de aquellas estructuras de datos que no pueden ser identificadas con un solo dato y requieren una clave compuesta por varios datos se dice que es una clave concatenada o compuesta. Como ejemplo podemos mostrar las asignaciones que, dentro de diferentes proyectos, un programador puede tener:

NUMERO-DE-PROGRAMADOR+
CODIGO-DE-SISTEMA+
CODIGO-DE-PROGRAMA+
NOMBRE-DEL-PROGRAMA+
FECHA-DE-INICIO+
FECHA-DE-TERMINACION

6.- Las formas normales

Para discutir los conceptos de primera, segunda y tercera forma normal utilizaremos, como ejemplo, una Orden de Compra. Para los efectos de este ejemplo supondremos que esta "visión de usuario" corresponde a una orden de compra de una empresa que dispone de varias bodegas (PUNTO-DE-ENTREGA) especializadas por rubros (CLASE-DE-MATERIAL), debido a las necesidades particulares de almacenamiento de cada uno de ellos (productos que requieren refrigeración, productos que requieren ser manejados con maquinaria especial, productos de pequeño tamaño, etc.). Asimismo, supondremos que los descuentos que se reciben de los proveedores son proporcionales a la cantidad ordenada.

ORDEN-DE-COMPRA =
NUMERO-DE-ORDEN +
NUMERO-DE-PROVEEDOR+
NOMBRE-DEL-PROVEEDOR +
DIRECCION-DEL-PROVEEDOR +
FECHA-DE-LA-ORDEN +
* 1 a 10 *< CODIGO-DE-MATERIAL +
DESCRIPCION-MATERIAL +
PRECIO-UNITARIO +
CANTIDAD +
DESCUENTO +
MONTO +
PUNTO-DE-ENTREGA +

CLASE-DE-MATERIAL> +
TOTAL-ORDEN

6.1.- Primera forma normal

Las estructuras de datos que tienen “campos repetitivos” ofrecen normalmente problemas de inflexibilidad y, desde el punto de vista del programador, resultan más difíciles de manejar. Por ejemplo, el registro de orden de compra arriba mostrado puede almacenar un máximo de 10 renglones. Si alguna orden de compra excediese ese máximo, sería necesario, o bien rehacer el archivo para cambiar su formato, o bien hacer un sinnúmero de artificios de programación.

Para evitar estos problemas de inflexibilidad es necesario simplificar la estructura de datos ORDEN-DE-COMPRA, descomponiéndola o fraccionándola en dos o más estructuras en PRIMERA FORMA NORMAL.

Una estructura en PRIMERA FORMA NORMAL (1NF) no es otra cosa que una estructura plana, es decir, en la que no existen grupos repetitivos y cada dato toma un solo valor.

ENCABEZADO-ORDEN =
NUMERO-DE-ORDEN +
NUMERO-DE-PROVEEDOR +
NOMBRE-DEL-PROVEEDOR +
DIRECCION-DEL-PROVEEDOR +
FECHA-DE-LA-ORDEN +
TOTAL-ORDEN
LINEA-DE-ORDEN-DE-COMPRA =
NUMERO-DE-ORDEN +
CODIGO-DE-MATERIAL+
DESCRIPCION-MATERIAL +
PRECIO-UNITARIO +
CANTIDAD +
DESCUENTO +
MONTO +
CLASE-DE-MATERIAL +
PUNTO-DE-ENTREGA

Al llevar la estructura ORDEN-DE-COMPRA a la primera forma normal (1NF), en lugar de tener un gran registro, tendremos dos registros: uno, para los datos generales de la orden de compra y otro, para cada los renglones de las órdenes de compra.

Las estructuras de datos en primera forma normal no están exentas de problemas. Si nuestra base de datos estuviese conformada por registros en primera forma normal como los que arriba se muestran, el dato DESCRIPCION-MATERIAL, por ejemplo, estaría repetido tantas veces como órdenes soliciten ese material. La primera forma normal es una estructura de datos más simple, aunque ofrece problemas de redundancia en los datos, lo cual puede acarrear problemas de actualización; si por alguna razón fuese necesario cambiar la descripción de algún material, sería necesario cambiar todos los registros de LINEA-DE-ORDEN-DE-COMPRA que lo contuvieran.

Con el fin de evitar este tipo de problemas, las relaciones en primera forma normal se pueden continuar simplificando en relaciones más simples.

6.2.- Dependencia funcional completa

Se dice que un dato B depende funcionalmente de otro dato A (de la misma estructura de datos) si para cada valor de A existe un solo valor de B (y solamente uno). Es decir, conocido el valor de A, el valor de B (dentro de la estructura) queda determinado. Por ejemplo en la estructura de datos LINEA-DE-ORDEN-DE-COMPRA:

EL ATRIBUTO:	DEPENDE FUNCIONALMENTE DE:
PRECIO-UNITARIO	CODIGO-DE-MATERIAL
DESCRIPCION-MATERIAL	CODIGO-DE-MATERIAL
CANTIDAD	NUMERO-DE-ORDEN +
	+ CODIGO-DE-MATERIAL
MONTO	NUMERO-DE-ORDEN +
	+ CODIGO-DE-MATERIAL

En particular, dentro de la estructura de datos LINEA-DE-ORDEN-DE-COMPRA, los atributos CANTIDAD y MONTO dependen de la clave concatenada NUMERO-DE-ORDEN + CODIGO-DE-MATERIAL y puede verse que dependen funcionalmente de toda la clave y no dependen de ninguna de sus partes (la cantidad ordenada de un material puede ser diferente en cada orden, es algo particular de cada combinación orden-material). Se habla en estos casos de una dependencia funcional completa.

6.3.- Segunda forma normal

Una estructura de datos está en SEGUNDA FORMA NORMAL (2NF) si está en primera forma normal y además todos sus atributos que

no son su clave primaria tienen una dependencia funcional completa con dicha clave primaria.

Siguiendo nuestro ejemplo, para que las estructuras del mismo sean de la segunda forma normal, deberemos fraccionar la estructura LINEA-DE-ORDEN-DE-COMPRA en dos o más estructuras que contengan únicamente atributos relacionados funcionalmente en forma completa con su clave primaria.

Así pues, la estructura original

LINEA-DE-ORDEN-DE-COMPRA =
NUMERO-DE-ORDEN +
CODIGO-DE-MATERIAL+
DESCRIPCION-MATERIAL +
PRECIO-UNITARIO +
CANTIDAD +
DESCUENTO +
MONTO +
CLASE-DE-MATERIAL +
PUNTO-DE-ENTREGA

se fraccionará en dos nuevas estructuras 2NF

LINEA-DE-ORDEN-DE-COMPRA =
NUMERO-DE-ORDEN +
CODIGO-DE-MATERIAL+
CANTIDAD +
DESCUENTO +
MONTO
MATERIAL =
CODIGO-DE-MATERIAL +
DESCRIPCION-MATERIAL +
PRECIO-UNITARIO +
CLASE-DE-MATERIAL +
PUNTO-DE-ENTREGA

6.4.- Dependencia transitiva

Consideremos ahora tres atributos A, B y C, de una estructura de datos; si C es funcionalmente dependiente de B y B lo es de A, se dice entonces que C depende transitivamente de A.

En nuestro ejemplo anterior, los datos DESCUENTO y MONTO dependen de la clave concatenada NUMERO-DE-ORDEN + CODIGO-DE-MATERIAL. Sin embargo esa dependencia es transitiva a través de

CANTIDAD, ya que el monto y el descuento dependen de la cantidad ordenada. Análogamente, el dato PUNTO-DE-ENTREGA depende del dato CODIGO-DE-MATERIAL, pero esa dependencia es transitiva, a través de CLASE-DE-MATERIAL, pues el punto de entrega (bodega a la que está destinado el producto) se asigna de acuerdo a la clase de producto.

El ejemplo nos permite hacer una diferencia importante. En el caso de los datos DESCUENTO y MONTO, existe una dependencia transitiva del dato CANTIDAD; sin embargo, es fácil observar que estamos frente a datos derivables (existe una fórmula para calcularlos) y que, por lo tanto, no necesariamente deben guardarse en la base de datos. Por otra parte, PUNTO-DE-ENTREGA no es un dato derivable y depende del dato CLASE-DE-MATERIAL.

Si nuestro almacenamiento estuviese compuesto de registros con este tipo de dependencias transitivas, existiría el riesgo de que al eliminar un registro de material eliminásemos también la información de PUNTO-DE-ENTREGA, pudiendo quedar fuera de nuestros archivos cualquier referencia a una bodega que, aunque vacía, todavía existe físicamente.

6.5.- Tercera forma normal

Para evitar “anomalías” como las señaladas en el párrafo anterior, las relaciones que contengan datos con dependencias transitivas pueden ser descompuestas en relaciones en tercera forma normal, las cuales se definen de la siguiente forma:

Una estructura de datos en TERCERA FORMA NORMAL (3NF) es una estructura que está en segunda forma normal y en la que todos sus atributos que no son su clave primaria dependen en forma no transitiva de esa clave primaria.

Así pues, la estructura MATERIAL puede ser descompuesta en dos o más relaciones en tercera forma normal:

MATERIAL =
CODIGO-DE-MATERIAL +
DESCRIPCION-MATERIAL +
CLASE-DE-MATERIAL +
PRECIO-UNITARIO
CLASES-DE-MATERIAL =
CLASE-DE-MATERIAL +
PUNTO-DE-ENTREGA

6.6.- Forma normal de Boyce y Codd (BCNF)

En su libro *An Introduction to Database Systems*, C. J. Date discute los problemas de actualización y manejo que cada una de las formas normales ofrece. No hemos querido entrar en mayores detalles para no desviar demasiado nuestra atención del objetivo principal como analistas de sistemas, que no es otro que: identificar las "cosas" o entidades acerca de las cuales un sistema maneja información, con el fin de desarrollar los modelos de datos asociados a él y, de esa forma, poder diseñar bases de datos libres de problemas de "integridad semántica".

Si queremos, sin embargo, comentar uno de los problemas que el mencionado autor discute, que es el de las claves compartidas. Siguiendo el proceso de normalización paso a paso, en algunos casos no llegaremos a alcanzar nuestro objetivo ya que existen algunas relaciones que, aún estando en tercera forma normal, no representan una sola entidad. Existen relaciones 3NF que presentan solapamiento de las claves candidatas, es decir, existe un dato compartido por varias claves candidatas. Por ejemplo, la estructura de datos que expresa los productos elaborados en una fábrica podría lucir así:

OPERARIO	MAQUINA	PRODUCTO
1	A	CLAVOS
1	B	MARTILLOS
2	C	CLAVOS
2	D	MARTILLOS

Nótese que esta estructura puede ser vista de dos formas diferentes: una, su clave es OPERARIO + MAQUINA y, otra, su clave es OPERARIO + PRODUCTO. De acuerdo con la definición de 3NF, la estructura está en tercera forma normal; sin embargo, no nos ayuda a describir "el mundo", pues no corresponde a una sola cosa (entidad). Por eso, desde un punto de vista práctico, podríamos haber llegado a una estructura de datos que no nos ayuda a alcanzar nuestro objetivo, que es el de segregar estructuras de datos que nos permitan colocar "cada cosa en su lugar" y tener "un solo lugar para cada cosa".

Con el fin de resolver este tipo de "anomalía", Boyce y Codd establecieron una nueva definición para la tercera forma normal, la cual ha recibido el nombre de de Forma Normal de Boyce y Codd (BCNF: Boyce Codd Normal Form). Para definir esta nueva tercera forma normal, sus autores establecieron primero el concepto de determinante funcional, diciendo que "en una estructura de datos, un atributo puede ser considerado como un determinante funcional si existe algún otro atributo

que dependa funcionalmente de él". De esta forma, si, en una estructura de datos, B depende funcionalmente de A, entonces A es un determinante funcional.

Se dice que una estructura de datos está en la FORMA NORMAL DE BOYCE Y CODD si, y solamente si, todo determinante funcional es una clave candidata.

Analicemos ahora, a la luz de la definición de BCNF, la estructura de nuestro ejemplo. Las máquinas A y C sólo producen clavos, mientras que las otras dos máquinas sólo producen martillos; es decir, PRODUCTO depende funcionalmente de MAQUINA. Esto significa que, dentro de la estructura que nos ocupa, MAQUINA es un determinante funcional, pero no es una clave candidata. Por lo tanto, la estructura de datos no es BCNF.

No deseamos ir más lejos en el análisis de este tipo de problemas. Nos limitaremos a decir que, una vez que nuestro análisis de visiones de usuario nos lleve a estructuras en tercera forma normal, éstas probablemente estarán en la forma BCNF y serán de utilidad para nuestros objetivos. En caso de que alguna de las relaciones no sea BCNF, el resultado deberá ser analizado con cuidado en términos del significado que tienen los datos para el negocio.

Sí deseamos, sin embargo, destacar la importancia del concepto de tercera forma normal BCNF, pues su definición nos permite ver claramente que una estructura de datos BCNF es, en pocas palabras, aquella estructura cuyos datos se refieren o describen a la clave únicamente ("cada dato se relaciona con la clave, sólo la clave y nada más que la clave"). Es decir, una estructura de datos BCNF describe una sola "cosa" o, dicho en otros términos, describe una sola entidad. Ello, a su vez, nos deja ver que, por lo tanto, es posible definir estructuras de datos BCNF sin necesidad de normalizar visiones de usuario, ya que identificando entidades, sus atributos y sus asociaciones, podemos llegar a definir estructuras BCNF.

7.- Ejemplos de normalización

1) **COTIZACION** =
   NUMERO-DE-COTIZACION +
   NUMERO-DE-PROVEEDOR +
   NOMBRE-DEL-PROVEEDOR +
   DIRECCION-DEL-PROVEEDOR +
   FECHA-DE-COTIZACION +
   < CODIGO-DE-PARTE +
   EQUIPO +
   MODELO +
   TIEMPO-DE-ENTREGA +
   PRECIO-COTIZADO >
   TOTAL-COTIZADO
   NOTA:

El Equipo y el Modelo describen la unidad a la cual está destinada la Parte.

PROVEEDOR =
NUMERO-DE-PROVEEDOR +
NOMBRE-DEL-PROVEEDOR +
DIRECCION-DEL-PROVEEDOR
COTIZACION =
NUMERO-DE-COTIZACION +
NUMERO-DE-PROVEEDOR +
FECHA-DE-COTIZACION +
TOTAL-COTIZADO
LINEA-DE-COTIZACION =
NUMERO-DE-COTIZACION +
NUMERO-DE-PROVEEDOR +
CODIGO-DE-PARTE +
TIEMPO-DE-ENTREGA +
PRECIO-COTIZADO
PARTE-EQUIPO =
CODIGO-DE-PARTE +
EQUIPO +
MODELO

2) **PEDIDO-DE-CLIENTE** =
NUMERO-DE-PEDIDO +
FECHA-DE-PEDIDO +
NUMERO-DE-CLIENTE +
NOMBRE-DEL-CLIENTE +
DIRECCION-DEL-CLIENTE +
< CODIGO-DE-ARTICULO +
DESCRIPCION-DEL-ARTICULO +
NUMERO-DE-FABRICANTE +
NOMBRE-DEL-FABRICANTE +
CANTIDAD +
PRECIO-UNITARIO +
PRECIO > +
TOTAL-PEDIDO
PEDIDO =
NUMERO-DE-PEDIDO +
FECHA-DE-PEDIDO +
NUMERO-DE-CLIENTE +
TOTAL-PEDIDO
CLIENTE =
NUMERO-DE-CLIENTE +
NOMBRE-DEL-CLIENTE +
DIRECCION-DEL-CLIENTE +
LINEA-DE-PEDIDO =
NUMERO-DE-PEDIDO +
CODIGO-DE-ARTICULO +
CANTIDAD +
PRECIO
ARTICULO =
CODIGO-DE-ARTICULO +
DESCRIPCION-DEL-ARTICULO +
FABRICANTE =
NUMERO-DE-FABRICANTE +
NOMBRE-DEL-FABRICANTE
FABRICANTE-ARTICULO =
CODIGO-DE-ARTICULO +
NUMERO-DE-FABRICANTE +
PRECIO-UNITARIO

8.- Bases de datos orientadas a objetos

A finales de los años 80 se comenzó a hablar de las bases de datos orientadas a objetos (OODB), con la idea de disponer de bases de datos inteligentes, capaces de soportar el paradigma de orientación a objetos, almacenando no sólo datos, sino también métodos. Sin embargo, como nos señala Robert J. Muller en su libro *Database design for smarties* (1999):

> "El modelo de datos orientado a objetos para los manejadores de bases de datos orientadas a objetos no existe como tal. La estructura de este modelo viene de la programación OO, estructurando los datos con los conceptos de herencia, de encapsulación y abstracción y de polimorfismo."

Hoy día, existe una gran variedad de productos que compiten en el mercado de manejadores de bases de datos orientadas a objetos, entre ellos podemos citar: OpenODB de HP, CACHÉ de InterSystems Corporation, Codebase de Sequiter Inc., ConteXt, db4objects, FastObjects, GemStone de GemStone Systems, Inc., GOODS - Generic Object Oriented Database System, Itasca de IBEX Corporation, Jasmine de Computer Associates International, Inc., MATISSE de ADB S.A., NeoAccess de NeoLogic Systems, O2 de Ardent Software, Objectivity/DB de Objectivity, Inc., ObjectStore de Object Design, Inc., ONTOS DB, POET de POET Software Corp., UniSQL de Cincom Systems, Inc., Versant de Versant Object Technology Corp.

Junto con los esfuerzos que vienen realizando esas empresas, muchos autores han venido justificando la necesidad de disponer de un modelo de datos orientado a objetos ya que no es posible unir "en armonioso matrimonio" la tecnología relacional con la orientación a objetos. En efecto, la experiencia señala que la productividad de los programadores se ve afectada por la dificultad de combinar un lenguaje de programación OO con llamadas al manejador de bases de datos a través del lenguaje SQL, que maneja tablas y no objetos. Dificultad a la que se le ha dado el nombre de "desigualdad en la impedancia" - impedance mismatch[1]-.

Podemos entender esta "desigualdad de impedancia" o impacto en la productividad, si observamos que un programador que utiliza un lenguaje OO y una bases de datos SQL, se ve obligado a manejar en forma

[1] En ingeniería eléctrica, se denomina impedancia a la relación que existe entre la tensión aplicada a un circuito y la intensidad de la corriente producida. La desigualdad de impedancia es un problema de la ingeniería eléctrica, que se presenta cuando se conectan dos circuitos con diferentes impedancias, ya que puede causar atenuación y ruido.

diferente los objetos persistentes -que se guardan en la base de datos-, en lugar de manejarlos en la misma forma como maneja en la memoria los objetos no persistentes -cuya vida expira al terminar el programa-.

Sin embargo, autores como el ya citado Robert J. Muller en su libro *Database design for smarties* (1999): nos señala que:

> "Personalmente, no encuentro que este problema sea tan serio. Escribir un applet de JDBC no es tan difícil y el diseño adicional requerido para desarrollar los métodos para manejar lo relacional no toma mucho esfuerzo serio de diseño o programación. La clave de la productividad en programación es la capacidad que tenga el lenguaje de desarrollo para expresar lo que usted quiere."

Adicionalmente, los defensores del modelo de datos OO señalan la incapacidad de las bases de datos SQL para el manejo de objetos complejos, como: imágenes, mapas de bits, datos que no sean atómicos sino tablas o tuplas, etc.

En una entrevista hecha a Chris J. Date publicada el 29 de julio de 2005 por *O´Reilly Network* en la página http://www.oreillynet.com/pub/a/network/2005/07/29/cjdate.html, el célebre autor expone sus ideas en relación a las polémicas que existen hoy día en relación a los modelos de bases de datos. En pocas palabras, C. J. Date nos indica que:

- Ninguna versión de SQL ha implementado el modelo relacional en su totalidad, en tal sentido señala que debe distinguirse entre las diferentes versiones de este lenguaje y el modelo relacional. En otras palabras, nos hace ver que las críticas o las limitaciones que señalan los defensores de los OODBMS -object oriented database management systems- o de los ORDBMS -object relational database management systems- están dirigidas a las implementaciones del SQL y no tienen sentido como críticas al modelo relacional, ya que este modelo contempla los elementos que sus detractores señalan como carencias del modelo.
- El modelo relacional y toda la teoría sobre la cual descansa es una teoría sólida, capaz de integrarse a la tecnología OO, por lo que, más que buscar nuevos modelos, es necesario implementarlo "como debe ser". A tal efecto, señala que su empresa ha desarrollado el lenguaje experimental *Tutorial D*, que implementa el modelo relacional en su totalidad y resuelve

satisfactoriamente las dificultades que se le atribuyen al modelo relacional, a causa de la "miopía" con la cual se desarrolló SQL.

Muchos autores señalan que la posición asumida por C. J. Date al tratar estos temas es demasiado crítica, severa y arrogante. Sin embargo, podemos entender la severidad de sus críticas, ante la posibilidad de que la industria ponga de lado un logro importante, como es la teoría relacional. No es poco común que los "pseudo científicos", que tanto abundan en la industria de la tecnología de información, propongan metodologías o teorías que no son tales y que no abonan nada en beneficio de la ingeniería de sistemas.

La polémica no es trivial y, con toda seguridad continuará. Para aquellos lectores que puedan estar interesados en el tema recomendamos la revisión de documentos como los siguientes:

- The Object-Oriented Database System Manifesto, publicado por Malcolm Atkinson (University of Glasgow), François Bancilhon, David DeWitt (University of Wisconsin), Klaus Dittrich (University of Zurich), David Maier (Oregon Graduate Center) y Stanley Zdonik (Brown University).
- The Third Manifesto, publicado por C. J. Date y Hugh Darwen.
- El libro *Databases, Types, and the Relational Model: The Third Manifesto* (Addison-Wesley - 2005), en el cual los autores del tercer manifiesto -C. J. Date y Hugh Darwin- critican las implementaciones de SQL y explican en mayor detalle los alcances del manifiesto.