UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CENTRO DE EDUCACIÓN A DISTANCIA Y GESTIÓN DEL CONOCIMIENTO

APUNTE ELECTRÓNICO

Bases de Datos

Licenciatura en Informática

UNAM FCA CED:GEC Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

COLABORADORES

DIR DIRECTOR DE LA FCA
Mtro. Tomás Humberto Rubio Pérez

SECRETARIO GENERAL
Dr. Armando Tomé González

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefa del Centro de Educación a Distancia y Gestión del Conocimiento

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

COORDINACIÓN DE MULTIMEDIOS
L.A. Heber Javier Mendez Grajeda
FCA-UNAM

AUTOR
Mtro. Carlos Francisco Méndez Cruz

REVISIÓN PEDAGÓGICA
Lic. Mayra Lilia Velasco Chacón

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero

DISEÑO EDITORIAL
Mtra. Marlene Olga Ramírez Chavero

Dr. Enrique Luis Graue Wiechers
Rector

Dr. Leonardo Lomelí Vanegas
Secretario General

Mtro. Tomás Humberto Rubio Pérez
Director

Dr. Armando Tomé González
Secretario General

Mtra. Gabriela Montero Montiel
Jefa del Centro de Educación a Distancia
y Gestión del Conocimiento / FCA

Bases de Datos
Apunte electrónico

Edición: agosto de 2017

Facultad de Contaduría y Administración
Circuito Exterior s/n, Ciudad Universitaria
Delegación Coyoacán, C.P. 04510, México, Distrito Federal.

ISBN: En trámite
Plan de estudios 2012, actualizado 2016.

Hecho en México

OBJETIVO GENERAL

El alumno obtendrá los conocimientos necesarios sobre los diferentes modelos de bases de datos, así como la metodología para construir la base de datos de un sistema informático.

TEMARIO OFICIAL

(64 horas)

	Horas
1. Plataforma teórico - conceptual	4
2. Modelo relacional	10
3. Modelo orientado a objetivos	10
4. Diseño	12
5. Construcción	10
6. Administración	12
7. Nuevas tecnologías	6
Total	**64**

INTRODUCCIÓN A LA ASIGNATURA

Una de las principales actividades profesionales a las que podrás dedicarte como Licenciado en Informática es al desarrollo de sistemas de información para las organizaciones. Esta labor es compleja, pero apasionante y de gran creatividad. Uno de los aspectos de mayor reto en el desarrollo de sistemas de información es el que tiene que ver con la tecnología de bases de datos organizacionales.

Por tal razón, en esta materia revisaremos los temas más importantes relacionados con la teoría, diseño, construcción y administración de una base de datos. Conocerás en qué consiste esta tecnología, que hoy en día es fundamental y necesaria para las empresas, desde una perspectiva teórica y práctica.

Así, en la primera Unidad, **plataforma teórico conceptual**, trataremos los fundamentos de las bases de datos mediante una revisión de los conceptos básicos para entender esta tecnología. Entre los principales están el concepto de base de datos y el concepto de un sistema manejador de bases de datos.

La segunda Unidad te proporciona las bases teóricas del modelo de bases de datos más utilizado actualmente en las organizaciones: el **modelo relacional**. Echarás un vistazo a la propuesta de Edgar Codd, fundador de este modelo de base de datos.

Una vez revisado el modelo relacional, atenderemos a un novedoso modelo que está ganando terreno en la industria. Nos referimos al **modelo orientado a objetos**, que si bien no tiene tanto rigor teórico como el relacional, parece una excelente opción para mejorar el desarrollo de sistemas de información.

En la Unidad de **diseño de bases de datos** discutirás la propuesta de modelado de Peter Chen y manejarás los conceptos para el desarrollo de un modelo Entidad-

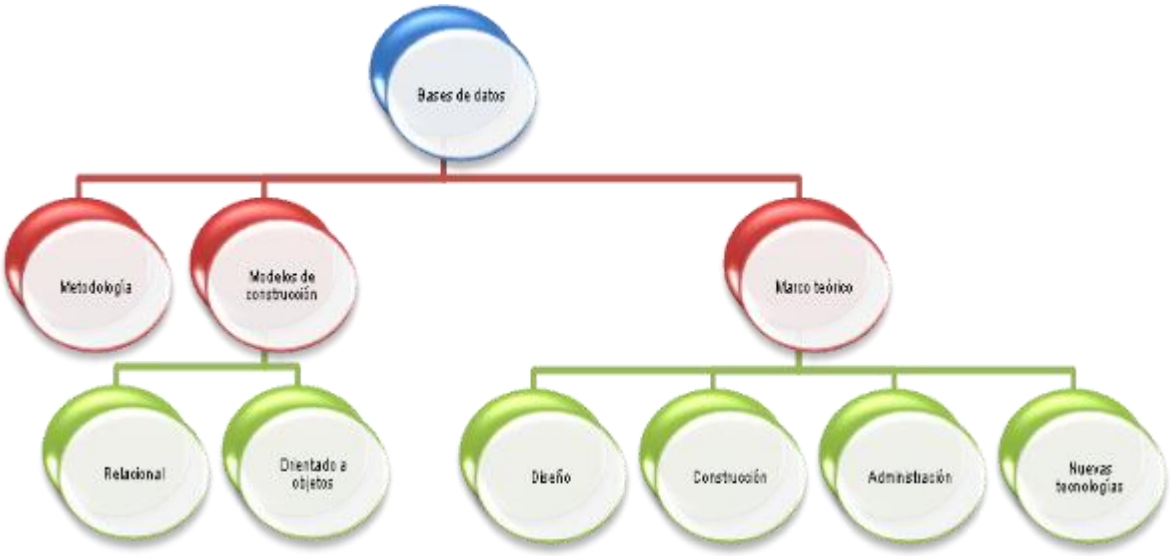
Relación. Además, esta Unidad incluye el uso de una representación gráfica llamada Diagrama Entidad-Relación.

Cuando en un proyecto de desarrollo de sistemas hemos terminado con el diseño de la base de datos, éste se implementa en un sistema manejador de bases de datos mediante programación. Esta etapa es cubierta en la Unidad de **construcción de la base de datos** en la cual enumeraremos los objetos programables de un sistema de base de datos relacional. Además, revisaremos cómo se almacenan datos en tablas y cómo son recuperados con consultas. Toda la Unidad está basada en el lenguaje de bases de datos relacionales SQL.

Pero la labor de un experto en bases de datos no queda sólo en su construcción, también es necesario el resguardo, mantenimiento y monitoreo del funcionamiento de la misma. Todas estas actividades que realiza el experto de bases de datos serán abordadas en la unidad dedicada a la **administración de la base de datos**.

Finalmente, dado el enorme crecimiento de la cantidad de información que guardan las bases de datos en las empresas, repasarás algunos aspectos introductorios a dos **nuevas tecnologías** del manejo de grandes bases de datos: el *Data Warehousing* y la Minería de Datos.

ESTRUCTURA CONCEPTUAL



Jerarquías	**Tablas**
Relaciones	**Bases de Datos**
Archivos	**Modelo**

Licenciatura: Informática

UNIDAD 1

PLATAFORMA

TEÓRICO-CONCEPTUAL

UNAM FCA
CEDUGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno conocerá el contexto histórico del surgimiento de los manejadores de bases de datos.

TEMARIO DETALLADO

(4 horas)

1. Plataforma Teórico - Conceptual

- 1.1. Historia
  - 1.1.1 Manejadores de archivos (campo y registro).
- 1.2. Definición e bases de datos
- 1.3. Definición de sistema administrador de bases de datos
  - 1.3.1. Elementos
  - 1.3.2. Modelo
  - 1.3.3. Objetivos

INTRODUCCIÓN

Con el fin de conocer el concepto y la importancia de las bases de datos, en esta Unidad estudiaremos su antecedente histórico: los manejadores de archivos. Esta tecnología de almacenamiento de información tuvo un uso muy extendido entre las empresas. Consistía básicamente en archivos de datos y lenguajes de programación que accedían a ellos. A pesar de que dichos lenguajes de programación se volvieron mejores en su labor, con el tiempo la tecnología de bases de datos vino a resolver fácilmente problemas que los manejadores de archivos resolvían de forma más compleja. Es importante mencionar que hoy en día la utilización de archivos de datos no ha quedado en desuso.

Después, destacaremos las definiciones de base de datos y de sistema manejador de bases de datos. Éstas son fundamentales para la formación de un informático y son retomadas en muchas de las materias de la carrera.[1]

Profundizaremos en los elementos de un sistema administrador de bases de datos, el modelo que sirve de base para su constitución y los objetivos que persigue. Esto nos dará una base conceptual para entender la importancia y repercusión de las bases de datos en la vida diaria de las empresas.

[1] En primer lugar estaría la materia Desarrollo de Aplicaciones con Manejadores de Bases de Datos Relacionales, además, las referentes al proceso de desarrollo de sistemas, ingeniería de software y construcción de aplicaciones.

1.1. Historia

La administración se ha ido modificando a través del tiempo con aportaciones y técnicas que han denotado el desenvolvimiento de nuevas teorías que proporcionan oportunidades para aplicar las nuevas tendencias de la administración.

Al inicio del siglo XVIII, hay que resaltar los sucesos que se dieron y tuvieron una fuerte influencia en relación con las prácticas administrativas, por citar algunas, el crecimiento de las grandes ciudades, la especialización, el invento de la imprenta y el crecimiento a gran escala en revolución industrial.

De este último punto podemos citar que la revolución inglesa se gestó durante los años 1700 a 1785 en donde los administradores se emplean por sus propios conceptos, aplicación de técnicas y principios. Cabe destacar que en este periodo se pasa vertiginosamente de una sociedad rural o agraria a una sociedad mercantil plena.

En los inicios del siglo XVIII es notorio que las labores del sistema doméstico consistían en producir para cubrir sus necesidades básicas en lugar de dedicarse a la caza o a la recolección, pero cuando se logra la especialización en donde una persona produce para satisfacer no solamente sus propias necesidades sino lo hace para ofrecer estos bienes a otros a través de venta o de trueque, se repunta el sistema doméstico. En este sistema no hubo oportunidad para establecer o aplicar técnicas administrativas, las funciones administrativas se establecieron de manera informal ya que incluía a la familia.

La siguiente etapa fue una evolución o desarrollo del sistema doméstico, conocido como sistema de trabajo a domicilio, que básicamente consistía en adquirir las producciones de las familias, con ello se comprometían a otorgar las materias

primas necesarias y pagar por el producto a una tasa por pieza. La venta de grandes lotes de los artículos que se requerían, hizo necesario tener el control de sus fuentes de oferta para evitar quedar sin la producción requerida.

El sistema fabril se caracterizó por un estricto control. Es a partir de este momento que los dueños fueron clasificados como comerciantes manufactureros y el interés fue mayor por la venta de su producción que por aspectos relativos a la administración. El sistema fabril fue establecido a partir de una costosa adquisición de maquinaria que trabajaba por medio de energía y considerando los aspectos administrativos el querer manejar y controlar a los hombres, a las máquinas y todo lo relacionado a la producción, la problemática de falta de control y coordinación: permite que sean necesarias las funciones del administrador, así como sus prácticas.

Es bien sabido que desde la antigüedad el hombre ha tenido la necesidad de guardar información sobre su acontecer. Por ello, en un pasado remoto, los sucesos importantes eran preservados en pinturas, grabados, papiros y después en papel. Con el paso del tiempo, la sociedad se volvió más compleja y la manera de guardar la información que ésta producía también cambió.

El surgimiento de organizaciones bien establecidas con distintos fines, económicos o sociales, trajo consigo la utilización de libros de registros. El crecimiento de estas empresas produjo que dichos registros se volvieran difíciles de manejar. Afortunadamente, la llegada de las computadoras proporcionó medios de registro y procesamiento más simples y ágiles, naciendo una nueva tecnología de almacenamiento de datos. En seguida revisaremos la primera solución tecnológica al almacenamiento de datos.

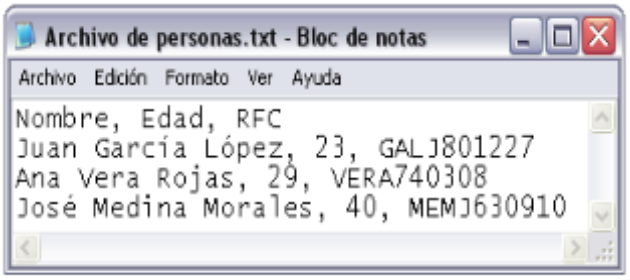
1.1.1. Manejadores de archivos (campo y registro)

El surgimiento de las computadoras brindó la posibilidad del procesamiento de grandes cantidades de datos. Esta situación requirió de la invención de una manera de almacenar el conjunto de datos que serían posteriormente procesados. La primera solución y que resolvió los problemas tecnológicos de las empresas durante mucho tiempo fueron los archivos de datos.

Con estos archivos de datos surgió la primera tecnología de almacenamiento. En ella, los datos del mundo real se representaban como un conjunto de caracteres. Cuando un conjunto de caracteres se refería a un dato particular, por ejemplo, el nombre de una persona, podíamos hablar de un campo. El conjunto de campos relacionados entre sí, de acuerdo con una asociación del mundo real, formaba un registro, por ejemplo, el nombre, edad y dirección de una persona. Finalmente, el grupo de registros asociados a un concepto determinado, digamos una nómina o el catálogo de una biblioteca, formaba un archivo.

Hoy en día, podemos hacer un archivo de datos tan sólo con abrir un editor de textos y formar campos y registros. Por ejemplo, en la figura puedes ver el fragmento de un archivo de personas. Cada campo: nombre, edad y RFC, está separado por una coma (,) y en él encontramos tres registros, uno por cada línea.[2]

[2] Este tipo de archivo es conocido como archivo separado por comas o archivo de valores separados por comas, calco del inglés *Comma Separated Values* (CSV). Este no es el único formato de archivos que ha sido utilizado en tecnologías de almacenamiento. Podemos encontrar también archivos separados por tabuladores o cualquier otro carácter. Algunas veces se prefieren archivos de ancho fijo, es decir, donde cada campo es del mismo tamaño.



Ejemplo de archivo de datos

Al principio, estos archivos eran procesados por lenguajes de programación de aplicación general, como Pascal o C. Después fueron manejados con lenguajes específicos para procesar archivos de datos, como Cobol o Clipper. Finalmente, surgieron sistemas manejadores de archivos especializados como DBase, Informix y FoxPro, en sus primeras versiones. Estos últimos comenzaron a utilizar archivos en formato binario y no sólo formato de texto o ASCII.

Estos manejadores de archivos fueron utilizados mucho tiempo para dar respuesta a las necesidades de información de las empresas. Esta situación permitió encontrar los límites y debilidades de esta tecnología.

Los principales problemas eran:

a) Ya que los grandes sistemas requerían de muchos archivos, mantener relacionada la información entre unos y otros a veces resultaba en programas muy complejos. Relacionado con esto, la cantidad de archivos que el sistema operativo podía mantener abiertos era otro problema.
b) Por ser simples archivo de texto o binarios, era posible utilizar distintos lenguajes o programas para modificarlos, brincando las rutinas que aseguraban la relación entre archivos o las rutinas de seguridad de los mismos.

c) Era común que interrupciones de energía o problemas de memoria del sistema operativo dañara los archivos cuando estaban abiertos, provocando registros perdidos.
d) La complejidad de los programas para procesar los archivos de datos hizo que las personas que los desarrollaban se volvieran indispensables. De igual manera, muchos de los lenguajes quedaron en desuso o las escuelas ya no los enseñaron.

Por estos y otros problemas, la tecnología de almacenamiento y procesamiento de grandes cantidades de datos evolucionó en lo que hoy conocemos como bases de datos (Véase, Silberschatz, Korth & Sudarshan, 2006, pp. 22-24).

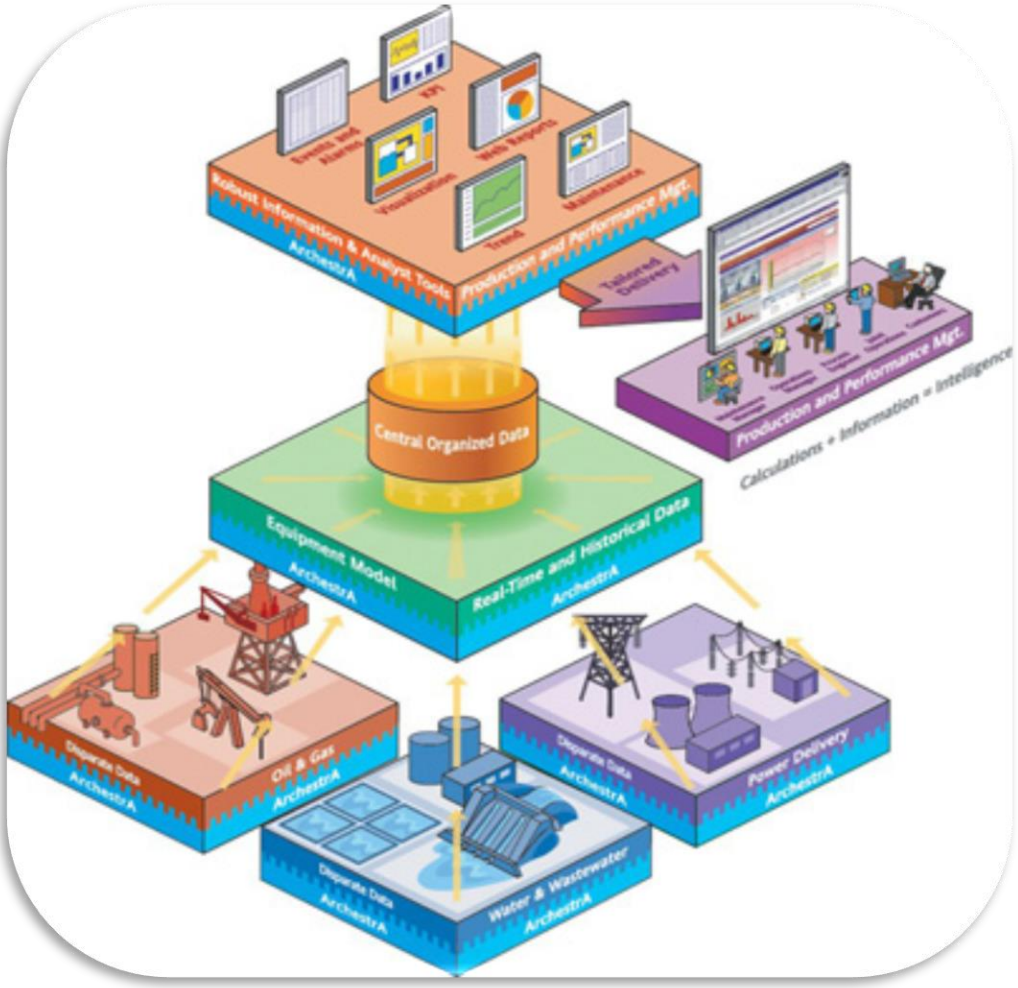
1.2. Definición de bases de datos

Para establecer una definición del concepto de base de datos vamos a separar los datos en sí mismos, de los programas de aplicación que los procesan y controlan. En este sentido, podemos definir una base de datos como una colección de datos relacionados, organizados, estructurados y almacenados de manera persistente. La persistencia es la característica de los datos que nos permite recuperarlos en el futuro, es decir que un dato es persistente si los podemos almacenar a través del tiempo.

También, la colección de datos debe estar organizada de acuerdo con un modelo que dictará la forma de las estructuras que almacenarán los datos. Estos modelos serán abordados en los temas siguientes, en los que hablaremos preferentemente del modelo relacional, ya que es el más utilizado en las empresas.

Una base de datos es finalmente un reflejo de la realidad. Esto quiere decir que a partir de observar un hecho del mundo, podemos modelarlo en términos de datos y crear una estructura que los almacene. En este sentido, y siendo estrictos, una base de datos no necesariamente debe estar computarizada, pero hoy en día no es fácil concebirlo así. Las organizaciones privadas y públicas de nuestra actualidad ya no pueden existir sin una base de datos computarizada que les brinde información veraz y oportuna para su toma de decisiones.

Para terminar este tema, debemos puntualizar que una base de datos requiere de programas que procesen, recuperen, compartan, aseguren y controlen sus datos. El conjunto de programas que hacen esto conforman lo que llamaremos Sistema Administrador de Bases de Datos, y que estudiaremos en la siguiente sección.



Ejemplo de base de datos computarizada (Recuperada)

1.3. Definición de sistema administrador de bases de datos

Una vez que contamos con una colección de datos, surge la necesidad de programas de aplicación que nos permitan almacenar, procesar, recuperar, compartir y asegurar esos datos, a este conjunto de programas lo llamaremos *Sistema Administrador de Bases de Datos*. Estos sistemas son conocidos también como Sistemas gestores de bases de datos, Sistemas manejadores de bases de datos, Sistemas de bases de datos o DBMSs, por las siglas del inglés *Database Management Systems*.

Los sistemas de base de datos permiten manejar grandes volúmenes de información. Son ellos los que brindan posibilidades de modificar y recuperar datos de forma ágil. Además, un sistema de base de datos debe tener mecanismos de seguridad que garanticen la integridad de la información y que impidan intentos de accesos no autorizados. Esta seguridad se vuelve aún más importante porque los datos están compartidos para muchos usuarios al mismo tiempo en una red de cómputo.

Con el fin de reafirmar el concepto de base de datos y de sistema administrador de base de datos, vamos a exponer algunas definiciones provistas por varios autores:

AUTOR	DEFINICIÓN
C. J. Date 	Una base de datos es un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada. (2001, p. 10)
James L. Johnson	Una base de datos es un conjunto de elementos de datos que se describe a sí mismo, con relaciones entre esos elementos, que presenta una interfaz uniforme de servicio. Un sistema de administración de bases de datos (DBMS) es un producto de software que presta soporte al almacenamiento confiable de la base de datos, pone en marcha las estructuras para mantener relaciones y restricciones, y ofrece servicios de almacenamiento y recuperación a usuarios; más funciones se ocupan de otras tareas, como son el acceso simultáneo, seguridad, respaldo y recuperar (lectura) de datos. (1997, p. 8) Un sistema de administración de bases de datos (DBMS) proporciona el método de organización necesario para el almacenamiento y recuperación flexibles de grandes cantidades de datos. (1997, p. 3)
Abraham Silberschatz 	Un sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada base de datos, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de

	datos de manera que sea tanto práctica como eficiente. (Silberschatz, Korth & Sudarshan, 2006, p. 1)

Una de las principales ventajas que ofrece el uso de un sistema de administración de bases de datos es la división de niveles de abstracción de datos. En el cuadro de abajo se presenta en dos columnas los tres niveles y su descripción.[3]

NIVEL	DESCRIPCIÓN
Nivel físico o interno	En este nivel se describe cómo están almacenados físicamente los datos.
Nivel conceptual o lógico	Describe la base de datos en término de estructuras de almacenamiento. Este conjunto de estructuras es también llamado esquema. Las estructuras están basadas en el modelo de datos que seleccionemos.
Nivel externo o de vistas	Es un conjunto de vistas a los datos que ocultan la base completa y están orientados a usuarios específicos.

Cuadro de niveles de abstracción

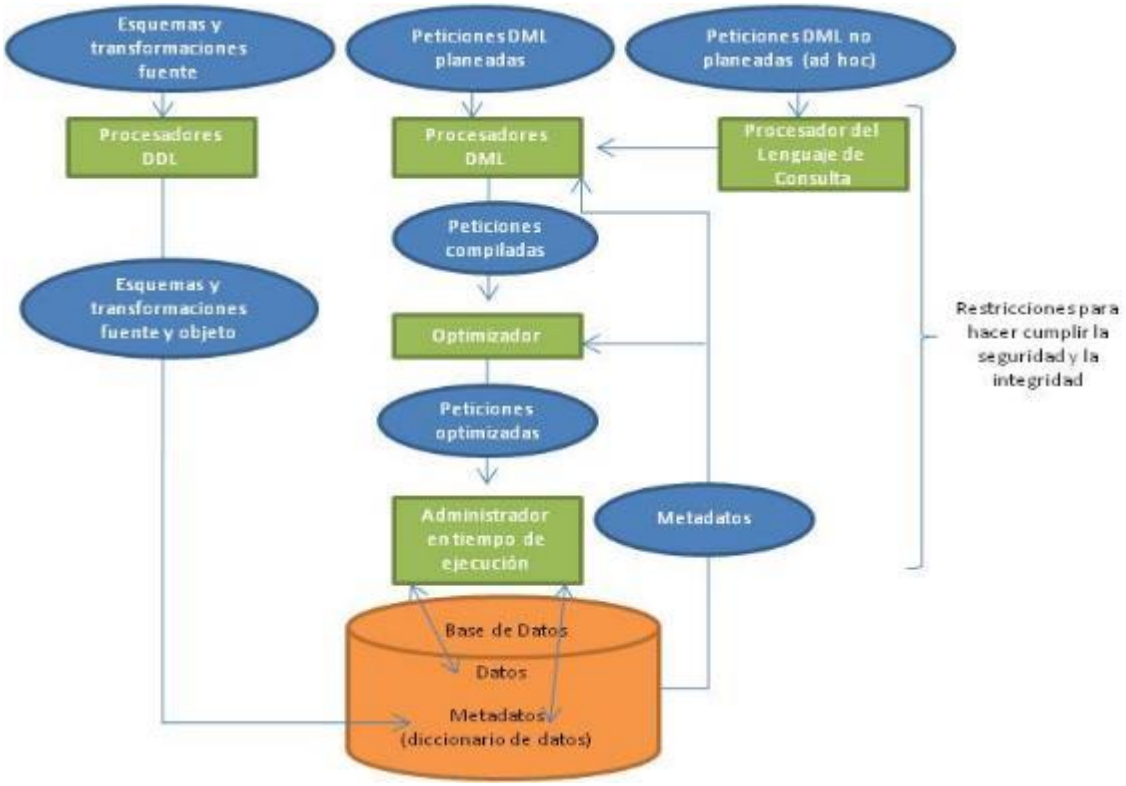
Un sistema administrador de bases de datos debe incluir un conjunto de lenguajes que le permitan definir estructuras de almacenamiento, manipular y consultar datos y controlar su acceso. En la práctica, estos lenguajes se encuentran unidos en uno solo, como el lenguaje SQL que revisaremos en la unidad II.

[3] Si quieres profundizar en los niveles de abstracción de un DBMS, revisa el texto de Date (2001, pp. 33-40) pues allí extiende la explicación de estos niveles en el capítulo 2.

Lenguaje

La división de lenguajes no es consistente entre los distintos autores del cuadro anterior, algunos consideran que son sólo dos: DML (Lenguaje de Manipulación de Datos) y DDL (Lenguaje de Definición de Datos). Es común que se diga que el DML incluye al DQL (Información de Query Language) y el DDL al DCL (Lenguaje de Control de Datos); así lo hace, por ejemplo, Silberschatz y otros (2006, p. 6).

Para terminar esta sección, creemos pertinente mencionar que un DBMS cuenta con una arquitectura. Ésta muestra la interacción de los distintos programas involucrados en la operación del sistema, es decir, cómo son procesadas las peticiones del usuario y cómo son manipulados los datos. Presentamos a continuación la arquitectura propuesta por Date (2001, p. 45) a manera de ejemplo. Así que confronta esta arquitectura con la de Johnson (1997, p. 17) y la de Silberschatz y otros (2006, p. 20).



Arquitectura de un DBMS

1.3.1. Elementos

Para Date (2001, p. 5) un sistema de administración de base de datos comprende cuatro elementos: datos, hardware, software y usuarios.

Los datos deben estar disponibles para varios usuarios al mismo tiempo, esto significa que el DBMS proporciona concurrencia de datos. Además, deben estar protegidos contra caídas del sistema e intentos de modificación por personas ajenas a la organización.

El software de un sistema administrador de bases de datos debe ser instalado en computadoras con características de hardware suficientes para brindar buen desempeño. Hoy en día, existen fabricantes especializados en sistemas de cómputo idóneos para bases de datos corporativas. Por lo general, basta con ponerse en contacto con ellos y exponerles las necesidades de información y las proyecciones de tamaño de nuestra base de datos.

Un DBMS comprende también un software encargado de hacer las gestiones con el sistema operativo y de dar los servicios de cómputo de la base de datos. Cuando este software está en funcionamiento, es frecuente llamarle servidor de base de datos. Este software incluye programas especializados para actualizar, recuperar, asegurar y compartir los datos de la base. Es habitual referirse al sistema administrador de bases de datos como un producto de software ofrecido por alguna compañía tecnológica. En el siguiente cuadro se listan algunos de los manejadores comerciales y de software libre más conocidos:

COMPAÑÍA	SOFTWARE	TIPO
Oracle	Oracle http://www.oracle.com	Comercial
Microsoft	SQL Server http://www.microsoft.com	Comercial
PostgreSQL Developer Group	PostgreSQL http://www.postgresql.org	Libre
MySQL	MySQL http://www.mysql.com	Libre
IBM	DB2 Universal Database http://www-01.ibm.com/software/data/db2/	Comercial

Manejadores de bases de datos comerciales y libres

Los usuarios que entran en juego en un sistema de bases de datos son principalmente los programadores de aplicaciones, programadores de bases de datos, los usuarios finales y el administrador de bases de datos. Los primeros se encargan de programar las interfaces gráficas que usarán los usuarios finales para almacenar y recuperar datos de la base. Esta actividad la realizan con distintos entornos de desarrollo mediante varios lenguajes de programación (java, php, c++). Los segundos crean las estructuras de almacenamiento y los objetos de base de datos necesarios para procesar los datos. Estos objetos serán revisados en la unidad V del temario.

Por otro lado, los usuarios finales son muy importantes ya que determinan las necesidades de información que deberá cubrir el sistema administrador de base de datos y finalmente serán los que alimentarán la base de datos. El administrador de la base de datos, llamado DBA por el inglés *Database Administrator*, es el encargado de llevar a cabo las tareas necesarias para un funcionamiento óptimo del DBMS, es común también que diseñe la base de datos y establezca las configuraciones necesarias al nivel de software y de seguridad. Las actividades del DBA se verán con mayor amplitud en la unidad VI.

1.3.2. Modelo

Un modelo de datos es una “colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica y las restricciones de consistencia” (Silberschatz y otros, 2006, p. 6). Existen dos modelos principales: el relacional y el orientado a objetos. Adoptamos un determinado modelo para crear la base de datos, de esta manera las estructuras de almacenamiento y sus relaciones estarían basadas en principios preestablecidos por el modelo. Por ejemplo, si nos decidimos por el modelo orientado a objetos tendremos a nuestra disposición para construir la base de datos los conceptos de herencia, polimorfismo y encapsulación. Repasaremos este modelo en la Unidad III.

Hoy en día, el modelo más extendido y utilizado es el relacional, que surgió a raíz de la propuesta de Edgar Codd en los años 70; sobre éste profundizaremos en la Unidad II.

1.3.3. Objetivos

Los objetivos principales de un sistema de base de datos son disminuir los siguientes aspectos:

1. Redundancia e inconsistencia en los datos

Es necesario evitar, en la medida de lo posible, la información repetida ya que aumenta el costo de almacenamiento y puede provocar problemas en el acceso a los datos. La inconsistencia en los datos se da cuando se pierde la relación lógica entre la información, por ejemplo, permitir que en la base de datos se registre un cargo sin su correspondiente abono.

2. Dificultad para tener acceso a los datos

Un DBMS debe cubrir las necesidades de información del usuario mediante un lenguaje de consultas sólido, esto implica prevenir cualquier petición o situación posible de ser solicitada.

3. Aislamiento de los datos

Antes del surgimiento de los sistemas administradores de bases de datos se utilizaban grupos de archivos por cada departamento de la empresa, los cuales muchas veces eran de distintos tipos, textuales o binarios, y eran tratados mediante diversos lenguajes de programación. Dicha situación causaba problemas para tener información centralizada. Los sistemas de bases de datos deben permitir la centralización de datos reduciendo su aislamiento.

4. Anomalías de acceso concurrente

Evitar inconsistencias por actualizaciones de usuarios que acceden al mismo tiempo a la base de datos. Era común que los administradores de archivos tuvieran problemas con la concurrencia.

5. Problemas de seguridad

La información que se guarda en una base de datos no debe ser vista con la misma profundidad por todos los usuarios de la misma. Por esta razón, el DBMS debe admitir niveles de usuarios y restricciones para consultar la información. También se requieren niveles de seguridad en contra de *hacking* o *cracking*.

6. Problemas de integridad

Los datos que ingresan a una base deben estar bien filtrados de manera que no se almacene información errónea o sin el formato adecuado. Para esto será necesario que el DBMS tenga mecanismos para implementar restricciones de integridad basadas en reglas de negocio.

Hemos expuesto arriba una cantidad considerable de conceptos asociados a la tecnología de bases de datos. Dos de ellos son los fundamentales: base de datos y sistema manejador de base de datos. Hoy en día, es prácticamente imposible imaginar una organización que no utilice bases de datos como parte de su labor cotidiana. Por ello es importante que seas capaz de reconocer los fundamentos expuestos en este tema.

Las bases de datos vinieron a mejorar la tecnología de almacenamiento de datos y se han vuelto indispensables gracias a los beneficios que ofrecen los DBMSs actuales. Conocer esta tecnología requiere de estudiar a los sistemas de bases de datos, sus elementos y modelos asociados. Por esto, en el siguiente tema abordaremos las especificaciones del modelo de datos más utilizado en la actualidad: el modelo relacional.

RESUMEN

El surgimiento de organizaciones bien establecidas con distintos fines, económicos o sociales, trajo consigo la utilización de libros de registros. El crecimiento de estas empresas produjo que dichos registros se volvieran difíciles de manejar. Afortunadamente, la llegada de las computadoras proporcionó medios de registro y procesamiento más simples y ágiles, naciendo una nueva tecnología de almacenamiento de datos. La primera solución que resolvió los problemas tecnológicos de las empresas durante mucho tiempo fueron los archivos de datos. Con estos archivos de datos surgió la primera tecnología de almacenamiento.

Los archivos se componen de forma jerárquica de la siguiente forma:

Caracteres

- Conjunto de datos con los cuales se representa el mundo real.

Campo

- Nombre que se le da al conjunto de caracteres que contiene datos particulares, por ejemplo el nombre de una persona.

Registro

- Conjunto de campos.

Entre más información manejaban, el Sistema Operativo tuvo la necesidad de ayuda, por lo que se desarrollaron lenguajes manejadores de archivos. Estos manejadores de archivos fueron utilizados mucho tiempo para dar respuesta a las necesidades de información de las empresas. Esta situación permitió encontrar los límites y debilidades de esta tecnología.

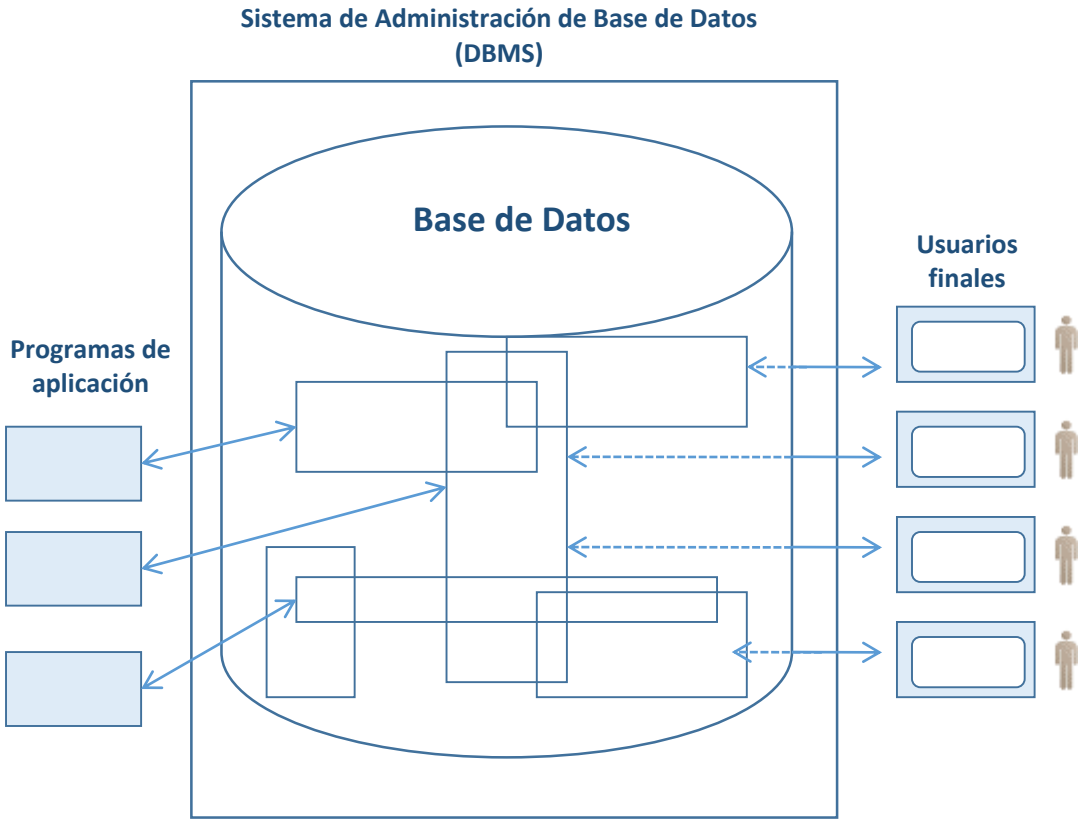
Los principales problemas eran:

a) Ya que los grandes sistemas requerían de muchos archivos, mantener relacionada la información entre unos y otros a veces resultaba en programas muy complejos. Relacionado con esto, la cantidad de archivos que el sistema operativo podía mantener abiertos era otro problema.

b) Por ser simples archivos de texto o binarios, era posible utilizar distintos lenguajes o programas para modificarlos, brincando las rutinas que aseguraban la relación entre archivos o las rutinas de seguridad de los mismos.

c) Era común que interrupciones de energía o problemas de memoria del sistema operativo dañara los archivos cuando estaban abiertos.

Estos lenguajes evolucionaron en *Sistema Administrador de Bases de Datos*, los cuales se refieren al conjunto de programas que procesan, recuperan, comparten, aseguran y controlan a sus datos dentro de la base de datos y así evitan la redundancia, inconsistencias y conservan la consistencia y persistencia de los datos y su disponibilidad para los Usuarios.



Este Modelado ayudó a darle ventajas y facilidades al Diseño de la Base de Datos.

BIBLIOGRAFÍA

SUGERIDA

Autor	Páginas
Date, C.J. (2001).	5-54
Elmasri, R. y Navathe. (2002)	1-37
Johnson, James L. (1997).	Completo
Silberschatz, A; Korth, HF. & Sudarshan, S. (2006).	1-18

Licenciatura: Informática

UNIDAD 2

MODELO RELACIONAL

UNAM FCA
CED:GEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno conocerá y entenderá los conceptos y elementos del modelo relacional de base de datos para su correcta aplicación en sistemas informáticos.

TEMARIO DETALLADO

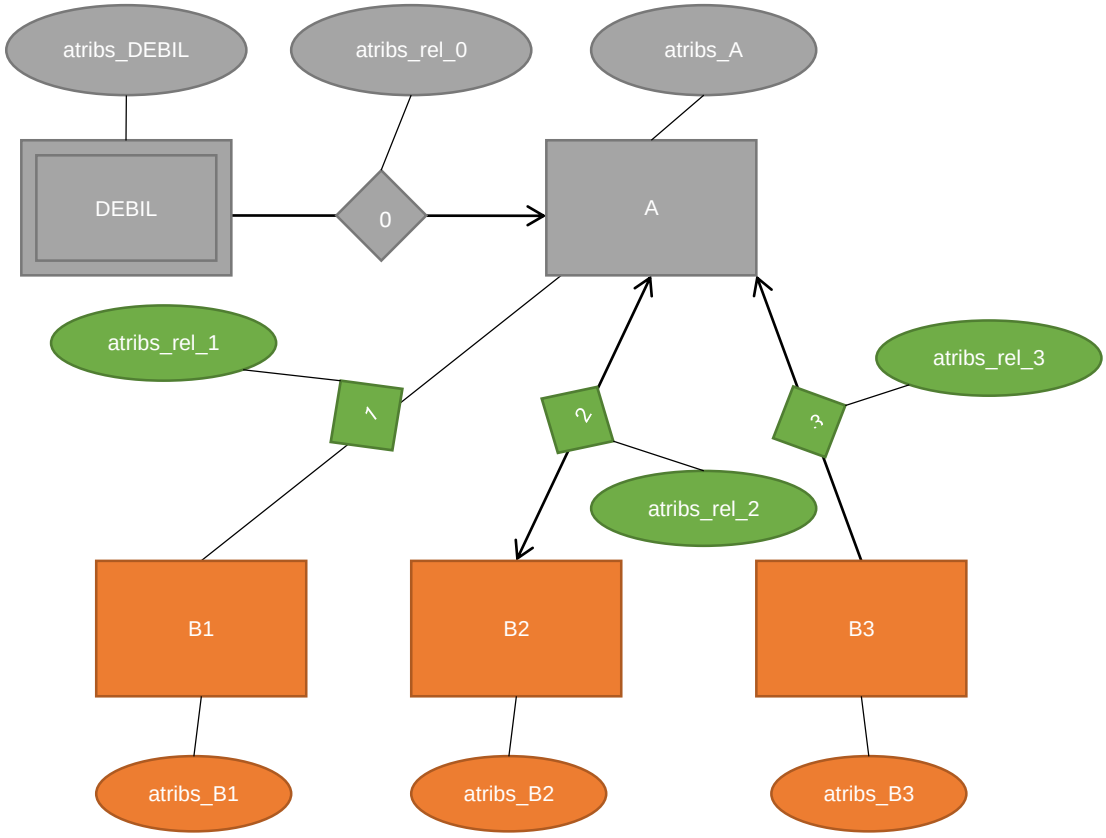
(10 horas)

2. Modelo Relacional

- 2.1.Introducción
  - 2.1.1. Modelos pre-relacionales
  - 2.1.2. Modelos post-relacionales
- 2.2. Definición de relación
  - 2.2.1. Partes
- 2.3. Propiedades de una relación
- 2.4. Dominio y tipos de datos
- 2.5. Álgebra relacional y cálculo relacional
- 2.6. Normalización
  - 2.6.1. Formas normales
  - 2.6.2. Proceso de descomposición sin perdida
- 2.7. Reglas de CODD
- 2.8. Estándares SQL

INTRODUCCIÓN

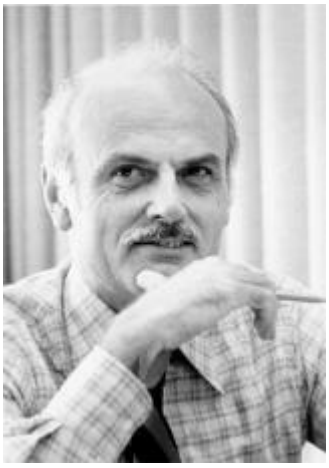
El modelo relacional de base de datos surge a finales de los 60, no obstante, es hoy en día el modelo más utilizado en sistemas empresariales. Los principales manejadores de bases de datos comerciales o de software libre están basados en este modelo y brindan soluciones tecnológicas robustas para todo tipo de empresas. Es por estas razones que el Licenciado en Informática debe conocer el modelo relacional de base de datos.



En esta unidad, revisaremos algunas características de los modelos pre-relacionales y pos-relacionales para brindar parámetros de diferenciación con el modelo relacional. También estudiaremos los fundamentos teóricos del modelo a partir de los conceptos de relación, dominio, álgebra y cálculo relacional.

En el proceso de desarrollo de una base de datos relacional, resulta importante evitar problemas de redundancia y de actualización de datos, por esto repasaremos el procedimiento conocido como normalización, el cual se basa en la descomposición sin pérdida de distintas relaciones para ajustarlas a formas normales.

Fue Edgar F. Codd quien puso las bases de este modelo y formuló lo que hoy se conoce como las “12 reglas de Codd”. En esta unidad dedicaremos un apartado a repasar estas reglas. Finalmente, discutidos los fundamentos del modelo, brindaremos aspectos generales del lenguaje estándar de desarrollo de bases de datos relacionales llamado SQL.



2.1. Introducción

A finales de 1968, Edgar F. Codd, matemático investigador de IBM, propuso el uso de las matemáticas para dar cierto rigor al campo de las bases de datos. Codd publicó sus ideas en un artículo que hoy es clásico: "*A Relational Model of Data for Large Shared Data Banks*" (1970). En ese artículo y subsiguientes, Codd presenta los conceptos fundamentales del modelo y sus beneficios frente a la tecnología de ese tiempo.

Tal como lo mencionamos en la unidad 1, la manera de manejar datos antes del advenimiento de las bases de datos era mediante archivos de datos. El uso de estos sistemas de archivos causaba distintos problemas que provocaban pérdidas de datos, datos duplicados innecesariamente, datos incompletos y erróneos.

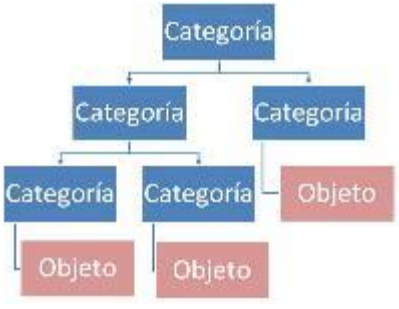
Los investigadores en computación de aquellos tiempos comenzaron a proponer nuevas opciones de almacenamiento hasta que surgieron los primeros modelos de bases de datos. A continuación, hablaremos de algunos de ellos.

Son básicamente dos modelos los antecesores del modelo relacional de base de datos. Ambos están basados en una estructura de nodos interconectados que almacenan la información.

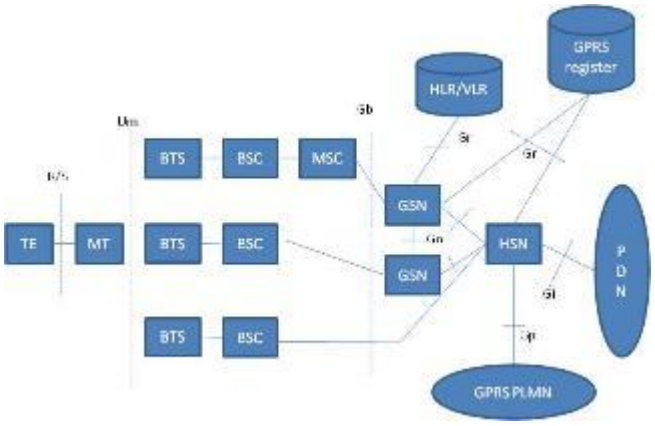
2.1.1. Modelos pre-relacionales

Dentro de estos modelos, podemos observar en un primero plano el **modelo jerárquico**, que interconecta nodos en una jerarquía estricta de padre e hijos, donde no podía haber relación entre nodos de distintos niveles o entre los del mismo nivel. Este modelo fue útil hasta que se empleó para resolver problemas de almacenamiento más complejos. La interconexión de nodos y, por consiguiente, el uso de apuntadores, comenzó a ser un inconveniente difícil de manejar por los sistemas de aplicación. El otro problema principal de este modelo es que no puede implementar relaciones de M:M (muchos a muchos) entre instancias de entidades del mundo real.

El segundo **modelo** es el **de red**. Se trata de una interconexión de nodos mediante apuntadores, sin la restricción del jerárquico, es decir, pueden salir de cada nodo varios arcos apuntando a otros nodos. A diferencia del modelo jerárquico, en este modelo se permiten las conexiones entre nodos de cualquier tipo, por lo que resulta bueno para relaciones M:M. Su principal desventaja radica en los problemas de implementación para lograr un rendimiento óptimo.



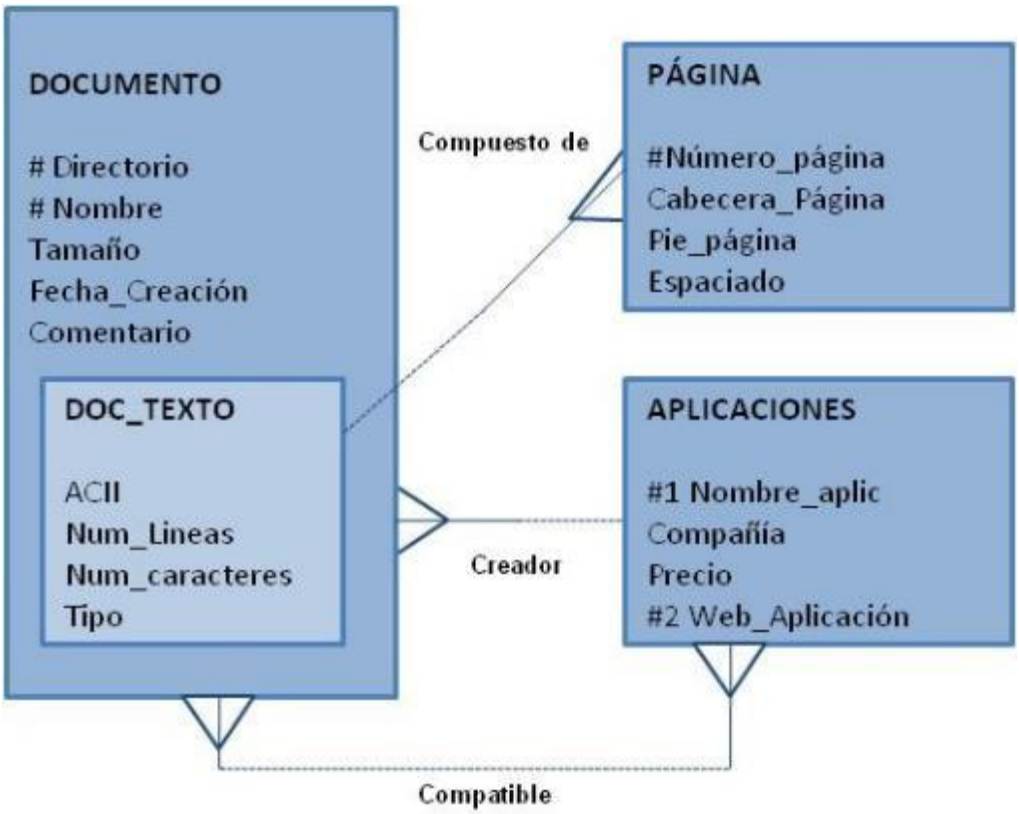
Modelo jerárquico simple



Modelo de red

2.1.2. Modelos post-relacionales

Los modelos pos-relacionales más conocidos son el modelo orientado a objetos, el distribuido, el deductivo, el multidimensional y el modelo semiestructurado.



Tendremos la oportunidad de revisar el modelo orientado a objetos en la unidad 3, así que no diremos nada sobre él en este apartado. El **modelo distribuido** está basado en el modelo relacional o en el orientado a objetos y su principal característica es que la base de datos está fragmentada en una red. Así, partes de las tablas de usuarios se encuentran en distintos lugares ya sea a nivel horizontal (renglones) o a nivel vertical (columnas). Uno de los retos principales de estos manejadores es que deben presentar la información al usuario como si las tablas estuvieran almacenadas localmente. Tanto la consulta como la actualización de datos deben estar garantizadas desde cualquier punto de la red.

El **modelo deductivo** se apoya en un conjunto de datos y reglas de inferencia. Es conocido también como modelo inferencial o modelo basado en la lógica. Las consultas de datos en este modelo son demostraciones de axiomas deductivos. El objetivo primordial de este tipo de bases de datos es inferir ciertos datos a partir de las reglas.

Las **bases de datos multidimensionales** son llamadas así porque los datos están almacenados en arreglos multidimensionales. Estos arreglos pueden ser interpretados en términos de variables dependientes e independientes. Las últimas determinan las dimensiones sobre las que se guardan valores (variable dependiente). Este modelo forma parte de los sistemas conocidos como MOLAP (Procesamiento Analítico en Línea Multidimensional) que tiene por objeto la generación de informes analíticos de datos.

Por último, el **modelo semiestructurado** surge ante la necesidad de que cada elemento almacenado cuente con un conjunto de atributos propio, sin necesidad de que este conjunto sea obligatorio como sucede en el modelo relacional. Este tipo de bases de datos se crea con el lenguaje de marcado extensible XML.

2.2. Definición de relación

El modelo relacional de bases de datos es llamado así porque se basa en estructuras de almacenamiento de datos llamadas relaciones. Las dos características fundamentales del modelo relacional de bases de datos son:

1. Los datos son percibidos por el usuario como relaciones y nada más que relaciones.
2. Para consultar los datos, el usuario cuenta con operadores que generan nuevas relaciones a partir de otras (álgebra relacional).

2.2.1. Partes

El modelo relacional se fundamenta en una teoría abstracta de datos que toma ciertos aspectos de las matemáticas (teoría de conjuntos y lógica de predicados). De esta manera, podemos definir una relación como un conjunto de tuplas y atributos, y que se compone de dos partes: **encabezado** *y* **cuerpo***.*

El **encabezado** de una relación denota un cierto predicado o función valuada como verdadera. Por su parte, cada fila en el **cuerpo** denota una cierta proposición verdadera obtenida del predicado por medio de la sustitución de ciertos valores, del tipo apropiado, en los indicadores de posición o parámetros de ese predicado. En otras palabras, podemos decir que el cuerpo es un conjunto de instancias o ejemplos del predicado. Veámoslo en un ejemplo:

Predicado

El empleado NUMEMP se llama NOMBRE_EMP, trabaja en el departamento NUMDEPTO y gana un salario SALARIO_EMP

Proposiciones verdaderas obtenidas del predicado:

- El empleado E1 se llama Azucena López, trabaja en el departamento D1 y gana el salario 20,000.
- El empleado E2 se llama Alberto Juárez, trabaja en el departamento D1 y gana el salario 8,000.

Si representamos gráficamente lo anterior tendríamos una relación como la siguiente:

NUMEMP	NOMBRE_EMP	NUMDEPTO	SALARIO_EMP
E1	**Azucena López**	**D1**	**20,000**
E2	**Alberto Juárez**	**D1**	**8,000**

Ahora bien, desde la perspectiva de la teoría de conjuntos, el encabezado es un conjunto de *n* atributos necesariamente distintos de la forma NombredeAtributo:NombredeTipo. Mientras que el cuerpo 1, el número de atributos de una relación se conoce como grado y el número de tuplas como cardinalidad.

2.3. Propiedades de una relación

Una relación debe contar con cuatro propiedades que son:

1. No existen tuplas duplicadas

La teoría de conjuntos descarta los elementos duplicados, además, no podemos afirmar dos veces el mismo hecho verdadero. La siguiente no es una relación:

NÚMERO	NOMBRES	APELLIDOS
1	Arturo	Jiménez
1	Arturo	Jiménez

2. Las tuplas están en desorden

La teoría de conjuntos nos dice que sus elementos están en desorden. Esto implica que no existen los conceptos de “tupla 1”, “siguiente tupla”, “cuarta tupla”; entonces nos debemos referir a una tupla por su proposición verdadera.

3. Los atributos están en desorden

Esto es porque el encabezado es un conjunto de atributos y los conjuntos no permiten elementos duplicados. Por tanto, tampoco hay conceptos de “siguiente atributo”, “primer atributo”, siempre se hace referencia a ellos por nombre, nunca por su posición

4. Los atributos deben contener valores atómicos

Esto significa que cada valor para cada tupla en un determinado atributo no debe ser semánticamente divisible y por tanto tiene que representar un valor único. La siguiente no es una relación, debido a que el atributo DEPARTAMENTO no guarda valores atómicos:

NÚMERO	NOMBRES	APELLIDOS	DEPARTAMENTO
1	Arturo	Jiménez	01 Finanzas
2	Arturo	Jiménez	02 Contabilidad

En una relación es posible encontrar tres tipos de claves:

Clave candidata	Clave principal	Clave foránea
Es un atributo que identifica de manera única a cada tupla de una relación. Una relación puede tener varias claves candidatas y estas pueden ser simples, formadas por un atributo, o compuestas cuando se forman por varios atributos. A las últimas se les llama superclaves.	Una clave principal es una clave candidata seleccionada para ser la clave principal de la relación. En este caso, a diferencia de las claves candidatas, sólo existe una por relación. También pueden ser superclaves.	Una clave foránea es un atributo que hace referencia a una clave principal en otra relación. Esta referencia implica que no pueda existir un valor en la clave foránea que no exista primero en la clave principal.

2.4. Dominio y tipos de datos

Un dominio es el conjunto de valores válidos para un atributo. De esta manera, podríamos decir que cada atributo tendrá asociado siempre un dominio. Es común relacionar este concepto con el de tipo de dato, pero no son lo mismo. Un dominio es más restrictivo en cuanto a la semántica de lo que guarda, por ejemplo, el dominio CIUDAD es el conjunto de todas las ciudades posibles, misma que se expresan en caracteres alfanuméricos mediante un tipo de dato carácter (char, varchar: son datos compuestos y de valores grandes como puede ser 200^-34) que podría aceptar cualquier combinación de estos.

Además, es posible utilizar ciertas restricciones para que los dominios acepten sólo determinados valores, pero siempre sobre la base de un tipo de dato simple como integer o char. Los tipos de datos están predefinidos en nuestro manejador de base de datos y contamos con la posibilidad de hacer conversiones entre distintos tipos.

Tipos de datos de valores grandes

En versiones anteriores de SQL Server, el trabajo con tipos de datos de valores grandes requería un tratamiento especial. Los tipos de datos de valores grandes son aquellos que superan el tamaño máximo de fila, 8 KB.SQL Server 2005 introduce un especificador máximo para los tipos de datos varchar, nvarchar y varbinary para permitir el almacenamiento de valores de hasta 2^31 bytes. Las columnas de tablas y las variables Transact-SQL pueden especificar los tipos de datos varchar(max), nvarchar(max) o varbinary(max).

Los escenarios principales en los que se trabaja con tipos de valores grandes implican su recuperación de una base de datos o agregarlos a una base de datos. Lo siguiente describe diferentes enfoques para realizar estas tareas.

Recuperación de tipos de valores grandes de una base de datos

Al recuperar un tipo de datos de valores grandes, que no sean binarios, de una base de datos, como el tipo de datos varchar(max), un planteamiento es leer esos datos como una secuencia de caracteres. En el siguiente ejemplo, se usa el método executeQuery de la clase SQLServerStatement para recuperar datos de la base de datos y devolverlos como un conjunto de resultados. A continuación, se usa el método getCharacterStream de la clase SQLServerResultSet para leer los datos de valores grandes en el conjunto de resultados.

(Véase) *http://msdn.microsoft.com/es-es/library/ms378813(SQL.90).aspx*

2.5. Álgebra relacional y cálculo relacional

Álgebra relacional

El álgebra relacional es el conjunto de operadores que permiten manipular las relaciones (consultar datos). Estos operadores siempre generan nuevas relaciones a partir de otras. Las operaciones de álgebra relacional son:

1. Restricción

Regresa una relación que contiene todas las tuplas de una relación especificada, que satisfacen una condición. Un ejemplo sería:

R(ALUMNO)

IDALUMNO	NOMBRE
1	Jaime
2	Raúl
3	Aurora

Restricción

IDALUMNO = 2

R(RESULTADO)

IDALUMNO	NOMBRE
2	Raúl

2. Proyección

Regresa una relación que contiene un subconjunto de atributos de una relación especificada. Por ejemplo:

R(ALUMNO)

IDALUMNO	NOMBRE
1	Jaime
2	Raúl
3	Aurora

Proyección

NOMBRE

R(RESULTADO**)**

NOMBRE
Jaime
Raúl
Aurora

3. Producto

Regresa una relación que contiene todas la tuplas posibles obtenidas por la combinación de todas las tuplas de la primera relación con todas las tuplas de la segunda relación. El siguiente es un ejemplo:

R(A)

IDA
1
2

PRODUCTO

R(B)

IDB
10
20

R(RESULTADO)

IDA	IDB
1	10
1	20
2	10
2	20

4. Unión

Regresa una relación que contiene todas las tuplas que aparecen en las dos relaciones, omitiendo las tuplas duplicadas. Mira el siguiente ejemplo:

R(A)

IDA	CVE
1	A
2	B

UNIÓN

R(B)

IDB	CVE
2	B
3	C
4	D

R(RESULTADO)

IDAB	CVE
1	A
2	B
3	C
4	D

5. Intersección

Regresa una relación que contiene todas las tuplas que aparecen en las dos relaciones especificadas. El siguiente es un ejemplo:

R(A)

IDA	CVE
1	A
2	B

INTERSECCIÓN

R(B)

IDB	CVE
2	B
3	C
4	D

R(RESULTADO**)**

IDAB	CVE
2	B

6. Diferencia

Regresa una relación que contiene todas las tuplas que aparecen en la primera pero no en la segunda de las dos relaciones específicas. Por ejemplo:

R(A)

IDA	CVE
1	A
2	B

DIFERENCIA

R(B)

IDB	CVE
2	B
3	C
4	D

R(RESULTADO)

IDAB	CVE
1	A

7. Junta (join)

Regresa una relación que contiene tuplas de dos relaciones a partir de la combinación de valores de uno o más atributos en común. De no existir una columna en común, no sería posible la junta. Observa el siguiente ejemplo:

R(PROD)

IDP	NOMBRE
1	Aretes
2	Pulseras
3	Collares

JUNTA

IDP = IDP

R(VTA)

IDP	CANT
2	100
1	30
3	250

R(RESULTADO)

NOMBRE	CANT
Pulseras	100
Aretes	30
Collares	250

8. División

A partir de dos relaciones unarias r(A) y r(B) y una relación binaria r(C), regresa una relación que contiene todas las tuplas de r(A) siempre y cuando su combinación con todas las tuplas de r(B) aparezca en r(C). Observa bien el siguiente ejemplo y verifica por qué esas tuplas de r(A) aparecen en la relación RESULTADO.

R(A)

IDA
1
2

R(B)

IDB
10
20

R(C)

IDA	IDB
1	10
2	10
2	20

DIVISIÓN

R(RESULTADO)

IDA
2

El resultado es 2 ya que es la única tupla de R(A) que al combinarse con todas las de R(B) (2-10, 2-20) aparece en R(C).

Cálculo relacional

Es posible identificar dos tipos de cálculo relacional: el cálculo de tuplas y el cálculo de dominios. En ambos casos no se especifica cómo realizar la consulta, esto significa que se indica únicamente una proposición verdadera que deben cumplir las tuplas o los dominios; en el álgebra relacional se indica el procedimiento para resolver la consulta, en el cálculo no. El principal operador de este tipo de consultas en el operador “existe” (*exists* en SQL).

2.6. Normalización

La normalización es el proceso de ajuste de relaciones a ciertas reglas llamadas formas normales. Tiene por objetivo reducir los problemas de redundancia y actualización en las relaciones; su ventaja es que lo hace mediante un procedimiento bien formalizado.

A continuación, te presentamos la definición de las formas normales. E. F. Codd propuso las tres primeras y con la aportación de posteriores investigadores en computación se establecieron las demás.

2.6.1. Formas normales

Como habíamos mencionado, las formas normales son reglas estrictas que deben cumplir las relaciones para disminuir problemas de redundancia y actualización. Varias de ellas están basadas en el concepto de Dependencia Funcional que explicaremos brevemente.

Dependencias funcionales
Una dependencia funcional (DF) es una relación muchos a uno que va de un conjunto de atributos a otro dentro de una determinada relación. En otras palabras: sea R una relación y sean X y Y subconjuntos cualesquiera del conjunto de atributos de R, entonces podemos decir que Y es dependiente funcionalmente de X, en símbolos X → Y[4], si y sólo si en todo valor válido posible de R, cada valor X está asociado precisamente con un valor de Y; siempre que dos tuplas coincidan en su valor X, también coincidirán en su valor Y. La parte izquierda de una DF se denomina determinante y la derecha dependiente.

[4] También se puede decir que “X determina funcionalmente a Y.

Dependencias triviales y no triviales

Una dependencia trivial se da si y solo si la parte derecha es un subconjunto de la parte izquierda. Las no triviales son las que no son triviales. Estas son las que realmente importan en el proceso de normalización mientras que las otras no son tomadas en cuenta.

Dependencias transitivas

Supongamos que tenemos una relación R con tres atributos: A, B y C, tales que las DFs A → B y B → C son válidas para R. Entonces es fácil ver que la DF A → C también es válida en dicha relación. Aquí la DF A → C es un ejemplo de DF transitiva y decimos que C depende de A transitivamente a través de B.

Cierre de un conjunto de dependencias

Al conjunto de todas las DFs implicadas en una relación se le llama cierre de dependencias y se escribe S+. Para obtener este cierre se utilizan reglas de inferencia para obtener nuevas DFs a partir de las ya dadas. Los axiomas de Armstrong nos permiten realizar esta labor y son los siguientes:

Sean A, B y C subconjuntos cualesquiera del conjunto de atributos de la relación dada R, entonces pueden ser aplicados los siguientes axiomas para producir nuevas DFs:

Reflexividad:	si B es subconjunto de A, entonces A → B.
Aumento:	si A → B, entonces AC → BC.
Transitividad:	si A → B y B → C, entonces A → C.
Autodeterminación:	A → A.
Descomposición:	si A (BC, entonces A (B y A (C.
Unión:	si A (B y A (C, entonces A (BC.
Composición:	si A (B y C (D, entonces AC (BD.

A continuación un ejemplo del uso de estos axiomas. Dadas las siguientes DF:

1) A (BC
2) B (E
3) CD (EF

Se pueden producir nuevas DFs a partir de la aplicación de los siguientes axiomas:

4) A (C por descomposición de 1.
5) AD (CD por aumento en 4.
7) AD (EF por transitividad entre 5 y 3.
8) AD (F por descomposición en 7.

Dependencias irreducibles

Una dependencia funcional se considera irreducible si la parte derecha involucra sólo un atributo y no es posible descartar ningún atributo del determinante sin cambiar el cierre. En otras palabras, en el caso de que la relación presente una clave principal compuesta, todo atributo a la izquierda de la DF debe depender de toda la clave y no sólo de una parte.

Primera forma normal (1FN)

Una relación está en 1FN si y sólo si toda tupla contiene exactamente un valor para cada atributo. Este concepto es el que ya habíamos explicado como la propiedad de una relación, por lo que muchos autores proponen que toda relación está en 1FN por definición.

Segunda forma normal (2FN)

Una relación está en 2FN si y sólo si está en 1FN y todo atributo que no sea clave es dependiente irreduciblemente de la clave primaria. Podemos decir, en otras palabras que la 2FN exige que todas la DFs de una relación sean irreducibles.

Tercera forma normal (3FN)

Una relación está en 3FN si y sólo si está en 2FN y todos los atributos que no son clave son dependientes en forma no transitiva de la clave primaria. La 3FN elimina la posibilidad de DFs transitivas.

Forma normal de Boyce/Codd (FNBC)

Una relación está en FNBC si y sólo si toda DF no trivial, irreducible a la izquierda, tiene una clave candidata como su determinante.

2.6.2. Proceso de descomposición sin pérdida

Pero qué sucede si una relación no cumple plenamente con una determinada forma normal: las formas normales también son acumulativas, es decir, no es posible estar en una forma normal más alta sin cumplir las formas normales inferiores.

Bueno, si una relación no cumple una forma normal, debemos descomponerla mediante un proceso conocido como *descomposición sin pérdida*. Éste consiste en una proyección de la relación para obtener nuevas relaciones que cumplan la forma normal exigida, y decimos que es sin pérdida si al juntar de nuevo las proyecciones regresamos a la relación original, preservando tanto el grado como la cardinalidad.

Revisemos ahora las llamadas **formas normales superiores**: 4FN y 5FN, para lo cual será necesario estudiar dos tipos más de dependencias.

Dependencias multivaluadas
Las dependencias multivaluadas se dan entre dos atributos. Uno de ellos caracteriza o determina a un conjunto bien definido de valores del otro atributo. Esta caracterización es independiente de otros atributos. Las dependencias multivaluadas se dan entre atributos multivaluados independientes entre sí donde se establece una combinatoria de todos contra todos. Veamos un ejemplo ilustrativo de De Miguel y Marcos (2000, p. 178).

R (ASIGNATURAS)

NOM_ASIGNATURA	PROFESOR	TEXTO
Ficheros y BD	Sr. Sánchez Sra. Hidalgo	Concepción y diseño de BD Fundamentos de BD
BD avanzadas	Sra. Hidalgo Sr. Martín	Diseño de BD avanzadas

Podemos hacer las siguientes observaciones sobre el ejemplo:
“Un profesor debe utilizar todos los textos. Por esto, un profesor no puede determinar un texto.”

La clave primaria sería una superclave con los tres atributos. Si normalizamos el modelo inicial obtendríamos una relación en FNBC, pero que tendría mucha redundancia.

R(ASIGNATURAS)

NOM_ASIGNATURA	PROFESOR	TEXTO
Ficheros y BD	Sr. Sánchez	Concepción y diseño de BD
Ficheros y BD	Sr. Sánchez	Fundamentos de BD
Ficheros y BD	Sra. Hidalgo	Concepción y diseño de BD
Ficheros y BD	Sra. Hidalgo	Fundamentos de BD
BD avanzadas	Sra. Hidalgo	Diseño de BD avanzadas
BD avanzadas	Sr. Martín	Diseño de BD avanzadas

Una dependencia multivaluada X→→Y se da cuando para cada valor de X hay un conjunto de cero o más valores de Y, independientes de los valores de los otros atributos de la relación. La siguiente relación, tomada de De Miguel y Marcos (2000, p. 180), no tiene una dependencia multivaluada ya que el texto “Modelo Relacional” no aparece en inglés.

R(CURSOS)

COD_CURSO	TEXTO	IDIOMA
A2783	Introducción a las BD	Español
A2783	Introducción a las BD	Inglés
A2783	Modelo relacional	Español
B2341	Concepción y diseño de BD	Francés
B2341	Modelo relacional	Español

Cuarta forma normal (4FN)

Una relación está en cuarta forma normal si y sólo si las dependencias multivaluadas tienen como determinante una clave. En este sentido, la relación ASIGNATURA no está en 4FN, por lo que es necesario descomponerla en dos relaciones:

ASIGNATURA_PROFESOR (Nom_asignatura, Profesor)

ASIGNATURA_TEXTO (Nom_asignatura, Texto)

Dependencias de combinación

Este tipo de dependencia se da cuando existe interdependencia entre los atributos de una relación y la descomposición en dos relaciones causa pérdida de información. Por tanto, la descomposición tiene que ser en varias proyecciones. Esta dependencia es también llamada de junta y se expresa así:

DJ * (R1, …, Rj)

Y significa que R = R1 join R2 join.. join Rj. Veamos un ejemplo de De Miguel y Marcos (2000, p. 190).

R(EDITA)

EDITORIAL	IDIOMA	TEMA
RA-MA	Inglés	BD
RA-MA	Español	CASE
Addison Wesley	Español	BD
RA-MA	Español	BD

Esta relación implica que si se publica un tema de bases de datos (BD) en un determinado idioma, por ejemplo, español, entonces todas las editoriales deben publicar ese tema y en ese idioma. Por esto, si Addison Wesley publicara BD en francés, sería necesario que RA-MA también lo hiciera, provocando insertar una tupla adicional.

La descomposición de la relación en dos relaciones provocaría tuplas espurias. Por ejemplo:

R(EDITA1)

Editorial	Idioma
RA-MA	Inglés
RA-MA	Español
Addison Wesley	Español

R(EDITA2)

Idioma	Tema
Inglés	BD
Español	CASE
Español	BD

R(EDITA1) join R(EDITA2)

Editorial	Idioma	Tema	
RA-MA	Inglés	BD	
RA-MA	Español	CASE	
RA-MA	Español	BD	
Addison Wesley	Español	CASE	←Tupla espuria
Addison Wesley	Español	BD	

Para que sea una descomposición sin pérdida sería necesario descomponer en tres relaciones, agregando la siguiente relación:

R(EDITA3)

EDITORIAL	TEMA
RA-MA	BD
RA-MA	CASE
Addison Wesley	BD

Quinta forma normal (5FN)

Una relación está en 5FN si y sólo si no existen dependencias de combinación. Si dicha relación no está en 5FN se deben hacer tantas proyecciones como descriptores involucrados en la dependencia. En el ejemplo eran tres atributos involucrados en la dependencia de junta y por eso se hicieron tres proyecciones.

2.7. Reglas de CODD

E.F. Codd propuso doce reglas que definen los requisitos de un manejador de base de datos relacionales. No obstante la mayoría de los manejadores comerciales no cumplen al 100 por ciento todas las reglas, buena parte de ellas han sido contempladas en los software de base de datos. Las reglas son:

1. Regla de la información

Toda la información de una base de datos relacional está representada explícitamente a nivel lógico mediante valores en tablas.

2. Regla de acceso garantizado

Todo dato (valor atómico) en una base de datos relacional es accesible de manera garantizada mediante la combinación del nombre de tabla, llave primaria y nombre de columna.

3. Regla del tratamiento sistemático de valores nulos

Los valores nulos (Null), que son diferentes a la cadena vacía, el carácter de espacio en blanco y al cero, son manejados por un sistema de bases de datos relacionales de manera sistemática con el objeto de representar la información desconocida o faltante; debe hacerlo de forma independiente del tipo de dato.

4. Regla del catálogo basado en el modelo relacional

La descripción de los datos dentro de una base de datos, es decir, el catálogo (nombres de tablas, nombres de columnas, tipos de datos de cada columna, nombres de restricciones, etc.) debe estar representada a nivel lógico de la misma manera que los datos normales de usuario, es decir, a través de tablas. Esto permitirá utilizar el mismo lenguaje relacional para recuperar datos del catálogo y datos normales de usuario.

5. Regla del sub-lenguaje de datos entendible

Un sistema relacional debe soportar varios tipos de lenguajes y varios modos de uso por parte del usuario. Sin embargo, debe existir al menos un sub-lenguaje relacional que permita expresar sentencias, mediante una sintaxis bien definida, con cadenas de caracteres. Este sub-lenguaje debe ser capaz de soportar las siguientes operaciones:

- Definición de datos.
- Consulta de datos.
- Manipulación de datos.
- Restricciones de integridad.
- Manejo de autorizaciones para los datos.

6. Regla de la actualización de vistas

Todas las vistas que sean teóricamente actualizables deben ser actualizables por el sistema de bases de datos.

7. Regla de inserciones, actualizaciones y eliminaciones de alto nivel

La posibilidad de manejar una relación como un único operador aplica no sólo a la recuperación de datos sino también a la inserción, actualización y eliminación de datos. Debe ser posible realizar estas operaciones para un conjunto de renglones.

8. Independencia física de los datos

Los programas de aplicación no sufren modificaciones a pesar de los cambios en el nivel de físico de almacenamiento o en los métodos de acceso.

9. Independencia lógica de los datos

Los programas de aplicación no sufren modificaciones a pesar de los cambios hechos a las tablas.

10. Regla de independencia de integridad

Las restricciones de integridad especificadas para una relación deben definirse con el sub-lenguaje de datos relacional, y almacenarse en el catálogo y no en los programas de aplicación.

11. Independencia de distribución

Un sistema relacional puede estar distribuido en distintos equipos o sitios de una red y las tablas deben verse como si estuvieran localmente.

12. Regla de la no subversión

Ningún lenguaje de bajo nivel puede ser usado para violar las restricciones de integridad expresadas en el lenguaje relacional de alto nivel.

2.8. Estándar SQL

En la sección anterior hemos visto que Codd propuso la utilización de un sub-lenguaje relacional que permitiera operaciones de definición, consulta, manipulación, restricción y control de acceso a datos. Hoy en día contamos con un lenguaje de programación para bases de datos relacionales que cumple en buena medida los requerimientos de Codd. Este lenguaje es conocido como SQL (*Structured Query Language*).

1. SQL89

La primera versión reconocida por la ANSI como estándar del SQL fue la de 1989, aunque el trabajo para desarrollar un estándar para bases relacionales había comenzado tiempo atrás en los laboratorios de investigación de IBM. Una versión inicial de la ANSI es de 1986, pero sufrió mejoras hasta llegar a la versión que hoy conocemos como SQL89.

Las principales características es que contaba con instrucciones DDL (create y drop), DML (select, insert, delete, update) y DCL (grant y revoke). Se había establecido que toda sentencia SQL debería comenzar con un verbo y después una serie de cláusulas que comienzan por palabras claves. Otra característica importante fue que SQL podía ser integrado a otros lenguajes de aplicación o podría ser ejecutado por una terminal interactiva directamente sobre la base de datos relacional.

2. SQL2 (SQL92)

En 1992 fue revisado el estándar por la ANSI y resueltos varios problemas de la primera versión. Entre las mejoras estuvieron:

* Soporte a caracteres de longitud variable.
* Expresiones para cambios de tipo de datos.
* Operador *join*.
* Manejo de transacciones.
* Uso de subconsultas.
* Uso de operadores de unión, intersección y diferencia.

3. SQL3 (SQL99)

Este nuevo estándar introdujo conceptos de la orientación a objetos y el uso de tipos de datos complejos. Hoy en día los principales sistemas manejadores de bases de datos han adoptado este estándar y se han convertido en lo que podemos llamar *manejadores objeto-relacionales de bases de datos.*

No es claro si las empresas cambiarán su tecnología de bases de datos en un futuro cercano. Además, es innegable que el modelo relacional sigue siendo el más utilizado y el que ha logrado resolver los requerimientos de información más significativos de distintas organizaciones. Por esto es muy importante que como informático conozcas bien este modelo y seas capaz de utilizarlo para resolver problemas de información.

RESUMEN

En esta unidad conocimos el modelo propuesto por Codd, matemático investigador de IBM, el cual ofreció a las bases de datos los fundamentos esenciales para que hoy en día sea el modelo sobre el cual se basa el diseño de los demás. Asimismo conocimos sus antecesores y sus predecesores, así como sus características principales (relaciones) y las reglas propuestas por este investigador que definen los requisitos de un manejador de base de datos relacional.

De igual manera, abordamos los dominios, los tipos de datos y la manera que se relaciona con los atributos; el álgebra relacional en función con la operación de las relaciones (normalización).

Por supuesto, se abordó el lenguaje de programación que es un claro ejemplo del modelo de Codd, el lenguaje conocido como SQL (*Structured Query Language*).

Finalmente se abordaron las etapas de la Normalización aplicadas al diseño de las Bases de Datos.

BIBLIOGRAFÍA

SUGERIDA

Autor	Páginas
Codd. EF. (1970).	58-82
Elmasri, R. y Navathe. (2002)	1139-262
Johnson, James L. (1997).	Completo
Silberschatz, A; Korth, HF. & Sudarshan, S. (2006).	87-247

UNIDAD 3

MODELO ORIENTADO A OBJETOS

UNAM FCA
CEDIGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno conocerá y entenderá los conceptos y elementos del modelo orientado a objetos de base de datos para su correcta aplicación en sistemas informáticos.

TEMARIO DETALLADO

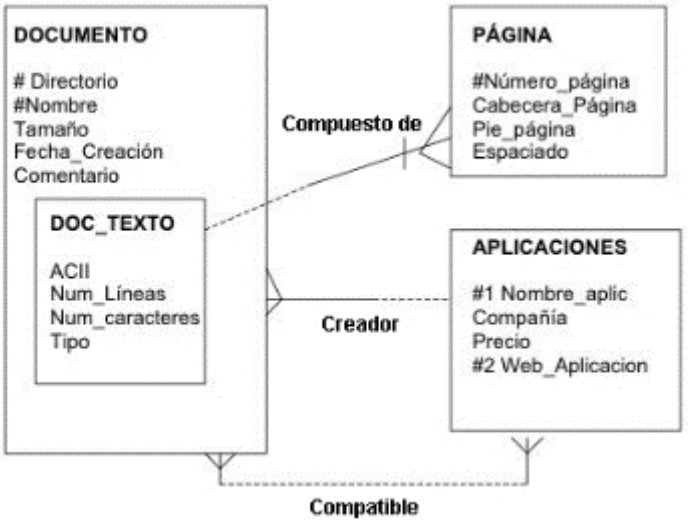
(10 horas)

3. Modelo orientado a objetivos

- 3.1. Introducción
  - 3.1.1. Retos actuales de los sistemas manejadores de bases de datos
  - 3.1.2. Tendencias actuales en la tecnología de bases de datos
  - 3.1.3. Orientación a objetos
  - 3.1.4. Persistencia
- 3.2. Sistemas de administración de bases de datos orientadas a objetos
  - 3.2.1. Antecedentes
  - 3.2.2. Primera generación
  - 3.2.3. Segunda generación
  - 3.2.4. Tercera generación
  - 3.2.5. Definición
  - 3.2.6. Características
- 3.3. Estándar ODMG

INTRODUCCIÓN

La tecnología de bases de datos vive un momento de lenta transición del modelo relacional a otros novedosos modelos con aplicaciones interesantes. Entre estos modelos están el multidimensional para sistemas OLAP, el semiestructurado para bases de datos XML de intercambio electrónico de información y el modelo dimensional para creación de Data Warehouse. Un modelo que llama mucho la atención por sus ventajas tecnológicas es el orientado a objetos.



La orientación a objetos llegó para quedarse y desde los años 80 se convierte día a día en la tecnología de desarrollo de aplicaciones por excelencia. Este hecho bastaría para entender el surgimiento de los sistemas manejadores de bases de datos orientados a objetos (OODBMS), pero más adelante revisaremos algunas motivaciones adicionales para el origen de esta tecnología.

Algo importante que debemos resaltar es que a pesar de las ventajas para el desarrollo de aplicaciones empresariales que tienen los OODBMS, hoy en día las empresas siguen utilizando los manejadores de bases de datos relacionales y no se sabe aún si serán suplantadas por completo, ni cuándo.

En los apartados siguientes revisaremos las situaciones tecnológicas que dieron origen a los manejadores de bases de datos orientados a objetos. Repasaremos los fundamentos de la orientación a objetos. Detallaremos la evolución de este tipo de sistemas de bases de datos. Finalmente se propone una definición y sus principales características.

3.1. Introducción

Los DBMS (sistemas manejadores de bases de datos) surgieron para responder a las necesidades de manejo de información de las organizaciones en grandes volúmenes. Se trata de un conjunto de datos persistentes y de programas para acceder a ellos y actualizarlos.

La tecnología de bases de datos tiene más de treinta años. Primero surgieron sistemas administradores de archivos de tipo ISAM (*Indexed Secuencial Access Mode*), que trabajaban con archivos separados. Después vinieron sistemas que centralizaban los archivos en una colección llamada base de datos, quienes utilizaron el modelo jerárquico (sistemas IMS y 2000). Posteriormente surgieron los desarrollados por la CODASYL (*Conference on Data Systems Languages*), como IDS, TOTAL, ADABAS e IDMC. La siguiente generación fue la de bases de datos relacionales. Éstas utilizaron lenguajes más accesibles y poderosos en la manipulación de datos como el SQL, QUEL y QBE.

Los DBMS cuentan con un modelo de datos, es decir, estructuras lógicas para describir los datos y operaciones para manipularlos (recuperación y actualización). Las operaciones sobre los datos se hacen por medio de tres lenguajes: un **DDL** (Lenguaje de definición de datos, *Data Definition Language* por sus siglas en inglés) para definir el esquema y la integridad, un **DML** (Lenguaje de Manipulación de Datos, *Data Manipulation Language*) para la actualización de los datos y un **DCL** (Lenguaje de Control de Datos, DCL por sus siglas en inglés: *Data Control Language*) para el manejo de las autorizaciones en la base de datos.
Adicionalmente un DBMS incluye mecanismos de seguridad, acceso a los datos, recuperación, control de concurrencia y optimización de consultas.

Los DBMS evolucionan con el afán de satisfacer nuevos requerimientos tecnológicos y de información. En seguida describiremos lo que ha motivado el surgimiento de sistemas orientados a objetos.

3.1.1. Retos actuales de los sistemas manejadores de bases de datos

Aunque los DBMS relacionales (RDBMS) son actualmente líderes del mercado y brindan las soluciones necesarias a las empresas comerciales, existen aplicaciones que necesitan funciones con las que no cuentan. Ejemplos de ellas son las CAD/CAM y CASE. Adicionalmente, los sistemas multimedia, como los geográficos y de medio ambiente, sistemas de gestión de imágenes y documentos, y los sistemas de apoyo a las decisiones necesitan de modelos de datos complejos difíciles de representar como tuplas de una tabla.

En general, estas aplicaciones necesitan manipular objetos y los modelos de datos deben permitirles expresar su comportamiento y las relaciones entre ellos. Los nuevos DBMS deben tomar en cuenta las siguientes operaciones:

- Ser capaces de definir sus propios tipos de datos.
- Manejar versiones de objetos y estados de evolución.
- El tamaño de los datos puede ser muy grande.
- La duración de las transacciones puede ser muy larga.
- Recuperar rápidamente objetos complejos.
- Ofrecer comunicación efectiva a los clientes del sistema, principalmente en desarrollos grupales.
- Permitir cambios en el esquema de la base.
- Manejar objetos completos y sus componentes.
- Lenguajes de consulta de objetos y lenguajes computacionalmente complejos.

- ❖ Mecanismos de seguridad basados en la noción de objeto.
- ❖ Funciones para definir reglas deductivas y de integridad.
- ❖ Tener la capacidad para comunicarse con las aplicaciones ya existentes y manipular sus datos.

3.1.2. Tendencias actuales en la tecnología de bases de datos

Con miras a superar los retos antes mencionados, las bases de datos están tomando varias tendencias. En general se están auxiliando de los lenguajes de programación orientados a objetos, los lenguajes lógicos y la inteligencia artificial. En este sentido, podemos determinar cuatro tendencias actuales:

1	Sistemas relacionales extendidos. Incluyen manejo de objetos y triggers.	**2**	Sistemas de bases de datos orientadas a objetos. Integran el paradigma de la orientación a objetos a la tecnología de bases de datos.
3	Sistemas de bases de datos deductivas. Unen las bases de datos con la programación lógica. Cuentan con mecanismos de inferencia, basados en reglas, para generar información adicional a partir de los datos almacenados en la base.	**4**	Sistemas de bases de datos inteligentes. Incorporan técnicas desarrolladas en el campo de la inteligencia artificial.

3.1.3. Orientación a objetos

La orientación a objetos representa el mundo real y resuelve problemas a través de objetos, ya sean tangibles o digitales. Este paradigma tecnológico considera un sistema como una entidad dinámica formada de componentes. Un sistema sólo se define por sus componentes y la manera en que éstos interactúan. Las principales características de la orientación a objetos son:

- Es una tecnología para producir modelos que reflejen un dominio de negocio y utiliza la terminología propia de tal dominio.
- Cuenta con cinco conceptos subyacentes: objeto, mensajes, clases, herencia y polimorfismo.
- Un objeto tiene un estado, un comportamiento y una identidad.
- Los mensajes brindan comunicación entre los objetos.
- Las clases son un tipo de plantilla usada para definir objetos, los cuales son instancias del mundo real.
- Cada objeto tiene un nombre, atributos y operaciones.

3.1.4. Persistencia

La persistencia es una característica necesaria de los datos en un sistema de bases de datos. Recordemos que consiste en la posibilidad de recuperar datos en el futuro. Esto implica que los datos se almacenan a pesar del término del programa de aplicación. En resumen, todo manejador de base de datos brinda persistencia a sus datos.

En el caso de los OODBMS, la persistencia implica almacenar los valores de atributos de un objeto con la transparencia necesaria para que el desarrollador de aplicaciones no tenga que implementar ningún mecanismo distinto al mismo lenguaje de programación orientado a objetos.

Lo anterior traería como ventaja que no sería necesario el uso de dos lenguajes de programación para construir una aplicación. Es decir, actualmente, el desarrollo de aplicaciones se hace con lenguajes de programación orientada a objetos almacenando datos en bases relacionales, por lo que el desarrollador debe utilizar un lenguaje para la aplicación (Java, PHP, C++) y otro para la base de datos (SQL).

El objetivo básico de un OODBMS es, entonces, dar persistencia a los objetos. Por lo anterior algunos autores ven estos sistemas sólo como lenguajes de orientación a objetos con persistencia y no como manejadores completos. Para ver una discusión acerca de si los OODBMS son en realidad DBMS, véase Date (2001, pp. 845-847).

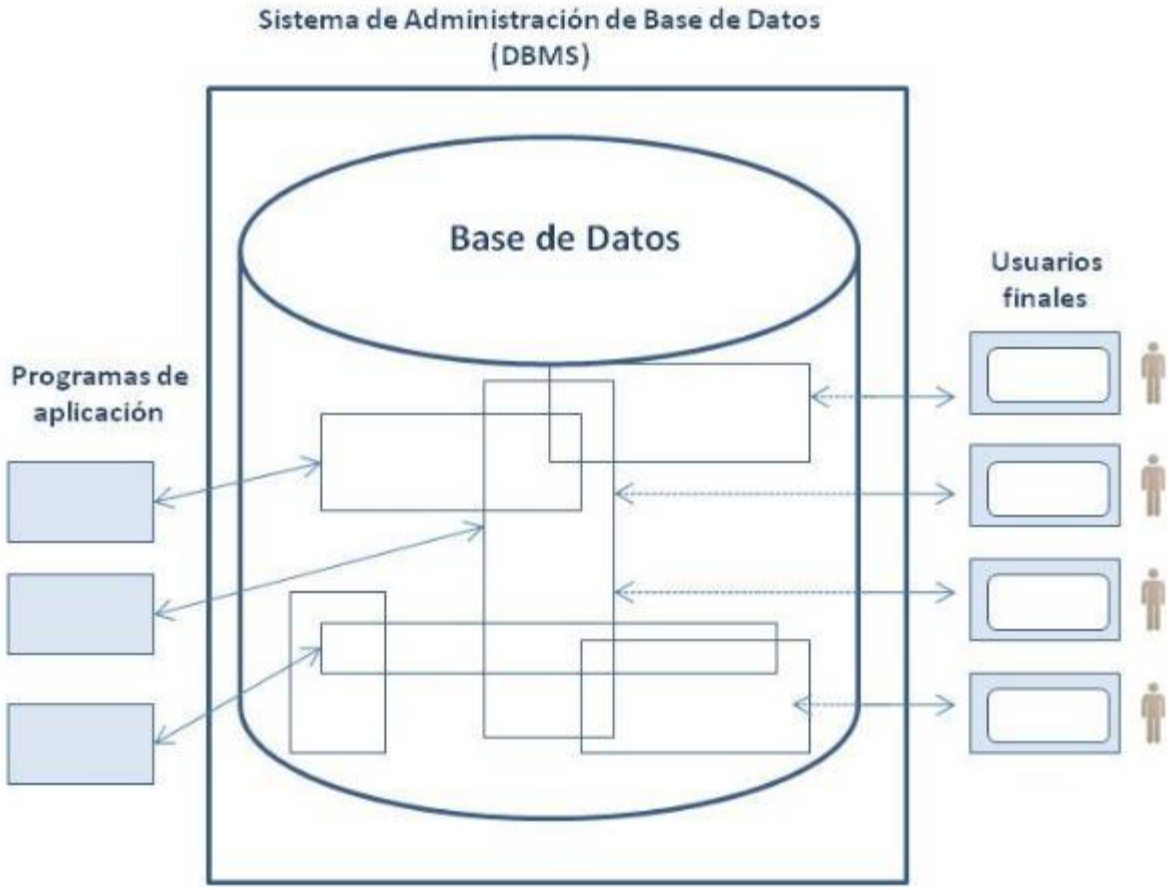
3.2. Sistemas de administración de bases de datos orientadas a objetos

Los sistemas de bases de datos orientados a objetos parecen ser la tecnología más prometedora para los próximos años, aunque carecen de un modelo de datos común y de fundamentos formales, además de que su comportamiento en seguridad y manejo de transacciones no están a la altura de los programas actuales de manejadores de Bases de Datos.

Hay organismos en pro de la estandarización de este tipo de sistemas manejadores de bases de datos, como el OMG (*Object Management Group*), la *CAD Framework Initiative* y el grupo de trabajo de ANSI (*American National Standards Institute*).

Algo que apoya esta tendencia es que a pesar de que la ingeniería de software orientada a objetos requiere mucho tiempo de análisis, la mayoría de los proyectos de desarrollo son más cortos y requieren menos personas, además de que la cantidad de código es menor.

A pesar de lo dicho anteriormente, sería difícil para las empresas dejar ‘de un día para otro’ los sistemas actuales, debido principalmente, a la falta de personal calificado, al efecto sobre la continuidad de sus operaciones y a la ausencia de garantías en la reutilización de los datos.



3.2.1. Antecedentes

La evolución de los sistemas de bases de datos orientados a objetos está muy ligada al mercado de manejadores y a las compañías que han apostado por este tipo de bases de datos. Como mencionamos, este tipo de DBMS no tiene estrictos fundamentos teóricos como el modelo relacional y por tanto no se puede establecer una historia de su concepción.

A continuación, se describe brevemente las tres generaciones de OODBMS según Bertino y Martino (1995).

3.2.2. Primera generación

Comienza en 1986 cuando el sistema G-Base fue lanzado por la compañía francesa Grápale. En 1987 Servio Corp introduce GemStone y en 1988 Ontologic promueve su Vbase, seguido de Statics por la empresa Simbolics. Estos sistemas estaban basados en lenguajes propios y plataformas independientes del mercado. Estos sistemas fueron considerados lenguajes orientados a objetos con persistencia.

3.2.3. Segunda generación

Se da con la salida al mercado de Ontos en 1989. Siguieron los productos Object Design, Objectivity y Versant Object Technology. Todos utilizaron una arquitectura cliente/servidor y una plataforma en C++, X Windows y UNIX.

3.2.4. Tercera generación

La generación comienza con Itasca, lanzado en agosto de 1990 por Microelectronics and Computer Corporation. Le siguieron O2 producido por la compañía francesa Altair, y después Zeitgeist por Texas Instruments. Estos ya son sistemas administradores de bases de datos con características avanzadas, un DDL y DML orientados a objetos.

3.2.5. Definición

No obstante, en la actualidad hay mucha atención hacia los OODBS, tanto en el terreno de desarrollo como en el teórico, no hay una definición estándar de lo que estos sistemas significan.

Existen tres problemas principales que impiden una definición generalizada:

1. **La falta de un modelo de datos común entre los diferentes sistemas**. Los sistemas de bases de datos relacionales cuentan con especificaciones claras dadas por Codd, pero los orientados a objetos no tienen algo así. Se pueden encontrar muchos textos que describen diferentes modelos, pero no hay uno como estándar.

2. **La carencia de fundamentos formales**. El fundamento teórico de la programación orientada a objetos es escaso en comparación con otras áreas como la programación lógica. Además se carece de definiciones de diversos conceptos.

3. **Una actividad experimental muy fuerte**. Existe mucho trabajo experimental, la mayoría de los desarrollos son sistemas prototipo no comerciales, no hay trabajo de conceptualización y definición de estándares. El diseño de estos sistemas está orientado por las aplicaciones que los requieren y no por un modelo común

El problema de estos sistemas es similar al de las bases de datos relacionales a mitad de los setenta. La gente se dedicaba a desarrollar implementaciones en lugar de definir las especificaciones para luego hacer la tecnología que permitiera implementarlas.

Se espera que de los prototipos y desarrollos actuales de los OODBS surja un modelo. Aunque también se corre el riesgo de que alguno de estos se convierta en el estándar por su demanda en el mercado.

A manera de definición podemos decir que un OODBS debe satisfacer dos criterios:

Debe ser un DBMS
El primer criterio incluye características de cualquier DBMS, que podemos listar como: persistencia, administración de almacenamiento secundario, concurrencia, recuperación y facilidad de consultas personalizadas.
Debe ser un sistema orientado a objetos (consistente con los lenguajes de programación orientada a objetos)
El segundo criterio corresponde a características que se comparten con la programación orientada a objetos: objetos complejos, identidad de objetos, encapsulación, herencia, sobre-escritura y sobrecarga, y completa capacidad computacional (*computational completeness*).

3.2.6. Características

1. Objetos complejos

Los objetos complejos son creados a partir de objetos (tipos de datos) simples. Estos objetos simples son: enteros, caracteres, cadenas de bytes, booleans y números de punto flotante. Los objetos complejos pueden ser, por ejemplo: tuplas, conjuntos (sets), listas y arreglos. Un OODBMS debe tener como mínimo conjuntos (set), listas y tuplas.

Los conjuntos (sets) son la manera natural de representar colecciones del mundo real. Las tuplas permiten representar de manera natural las propiedades de una entidad y son importantes por la aceptación ganada con el modelo relacional. Finalmente, las listas o arreglos resultan importantes porque capturan orden, cosa que ocurre en el mundo real; además de que ayudan a representar matrices y series de datos en el tiempo.

2. Identidad de objetos

La identidad de objetos ha existido desde hace mucho tiempo en los lenguajes de programación, pero en las bases de datos es más reciente. El objetivo es contar con objetos que tengan una existencia independiente de sus valores. Así, dos objetos pueden ser idénticos si son el mismo objeto o pueden ser iguales si tienen los mismos valores.

La identidad de objetos cobra relevancia cuando un objeto se comparte con otros y cuando se actualiza. En un modelo basado en identidad, dos objetos pueden compartir un objeto hijo. Por ejemplo, pensemos en dos personas, Arturo y Valeria, y cada uno tiene un hijo llamado Jaime. En la vida real hay dos posibles situaciones:

a) Arturo y Valeria son padres de Jaime, por lo que existe identidad del objeto Jaime, es decir, se trata en realidad del mismo objeto.
b) Hay dos niños del mismo nombre. Esto implica igualdad de dos objetos Jaime, porque tienen el mismo nombre, pero no son el mismo objeto.

Asumiendo que Arturo y Valeria son en realidad padres de un mismo niño Jaime, todas las actualizaciones al hijo de Arturo, también son aplicadas al hijo de Valeria, ya que se trata del mismo objeto, Jaime. Si el sistema no tuviera identidad, sino que se basara en igualdad de objetos, serían necesarias dos actualizaciones a los dos objetos Jaime.

Soportar identidad de objetos implica que el OODBMS ofrece operaciones como asignación de objetos, copiado de objetos y comprobación de la identidad o igualdad de objetos.

La principal manera de implementar la identidad de objetos es mediante un OID (*Object Identifier*) independiente de los valores de los atributos del objeto. Estos son implementados por el sistema y muchas veces a bajo nivel, lo que mejora el rendimiento.

Según Date (2001, p. 847), los OID son innecesarios e indeseables en el nivel del modelo. En comparación con las claves primarias, los OID están ocultos al usuario mientras que las claves no. El uso de estos *object identifiers* no elimina el uso de claves primarias ya que son necesarias para unir al sistema con la realidad en la que se inserta; pensemos, por ejemplo, en los folios de facturación.

3. Encapsulación

La idea de encapsulación es tomada de los lenguajes de programación en los que para todo objeto existe una parte visible que permite especificar el conjunto de operaciones que pueden ser realizadas sobre el objeto. La otra parte no es visible y contiene los datos que almacena ese objeto.

Traducido a bases de datos, un objeto encapsula programas y datos. Por ejemplo, en un sistema relacional, un empleado es representado por una tupla. Para ser consultado desde una aplicación es necesario usar un lenguaje de programación para programar procedimientos que serán almacenados fuera de la base datos; es más, ese lenguaje puede ser en realidad la combinación de un lenguaje de alto nivel con el lenguaje estándar relacional SQL.

En un sistema de bases de datos orientado a objetos, definimos al empleado como un objeto que tiene una parte de datos (probablemente muy similar al registro que definiríamos en el sistema relacional) y una parte de operaciones, la cual consiste en operaciones de aumentodeSalario() y bajaDefinitiva(), por ejemplo, que accederían a los datos del empleado. Cuando se almacene un nuevo empleado, datos y operaciones serían almacenados en la base de datos y no fuera de ella.

4. Herencia

La herencia tiene dos ventajas: es una herramienta poderosa de modelado, ya que brinda una descripción precisa del mundo y ayuda a simplificar la implementación de las aplicaciones. Para entender el manejo de la herencia en los sistemas de bases de datos orientados a objetos vamos a establecer un ejemplo:

Asumamos que tenemos empleados y estudiantes. Cada empleado tiene un nombre, edad y salario, además podemos aumentar su sueldo. Por su parte, cada estudiante tiene edad, nombre y un conjunto de calificaciones con la cuales podemos obtener su promedio.

En un sistema relacional, el diseñador de bases de datos definiría una relación empleado y una estudiante, también escribiría el código para la operación de aumentar sueldo. Para la relación empleado tendría que escribir el código para la operación de obtener promedio.

En un sistema orientado a objetos, usando adecuadamente la herencia, nos daríamos cuenta de que *empleado* y *estudiante* son personas y comparten los atributos: *nombre* y *edad.* Entonces, declararíamos una clase empleado como un tipo especial de la clase persona, el cual incluiría una operación especial de aumentar Sueldo() y un atributo de salario. De forma similar se declararía el estudiante como un tipo especial de la clase persona con el atributo adicional de conjunto de Grados y la operación especial obtener Promedio().

El modelo es más cercano a la realidad y nos permite ahorrar código de programación. Por esto se dice que la herencia ayuda a reutilizar código ya que cada programa está disponible para ser compartido.

5. Sobreescritura y sobrecarga

En la programación orientada a objetos tenemos la ventaja de poder reescribir métodos con el mismo nombre. Esto significa contar con varios métodos que se llamen igual, pero que realicen distintas operaciones. Para poder programar estos métodos llamados sobrecargados, es necesario que cambie algo en sus parámetros, como el número, orden o tipo de dato. Esto mismo es posible de una base de datos orientada a objetos.

6. Completa capacidad computacional (*Computational completeness*)

Los manejadores de bases de datos relacionales cuentan con un lenguaje para realizar procesos computacionales sobre los datos: el SQL. Además, adicionan un lenguaje procedimental (de *procedural*) que permite la definición de variables, manejo de excepciones, ciclos y estructuras condicionales. Estos lenguajes son pl/sql para Oracle, pl/pgsql para PostgreSQL y Transact-SQL para SQL Server de Microsoft, por dar unos ejemplos.

Los manejadores de bases de datos orientadas a objetos, también deben contar con un lenguaje que puede realizar cualquier procesamiento. En este sentido, lo más común es que los OODBMS integren lenguajes computacionalmente completos dentro de la base de datos. Estos pueden ser los que ya existen en el mercado y que se usan como lenguajes aplicación general (Java, C++, etc.).

3.3. Estándar ODMG

El *Object Database Management Group* (ODMG) surge en 1991 formado por un grupo de proveedores de OODBMS. Generaron un primer estándar en 1993 en donde exponían las características que consideraban necesarias en un sistema de bases de datos de este tipo. En 2001 aparece la nueva versión del estándar que se acomoda a las especificaciones de la tecnología de Java.

Los principales componentes de un OODBMS según el ODMG son:

a) Lenguaje de definición de objetos (ODL).
b) Lenguaje de consulta de objetos (OQL).
c) Conexión con los lenguajes C++, Smalltalk y Java.

En 1989, el artículo "*The Object-Oriented Database System Manifesto*", aunque con un enfoque demasiado limitado en temas de administración, Stonebraker et al. (1990), se propone una definición compuesta de tres tipos de reglas que deben respetar un OODBMS:

I. **Reglas Obligatorias**: Las cuales, el sistema debe imperativamente seguir para merecer la calidad de OODBMS.
II. **Reglas Facultativas**: Lineamientos suplementarios del sistema.
III. **Reglas Abiertas**: Propiedades alternativas del sistema que puede ejercer.

El estándar ODMG-93: Un estándar para bases de datos puramente orientadas a objetos.

Es el resultado de trabajos que duraron 18 meses por los 5 principales distribuidores de OODBMS. Su objetivo fue asegurar la portabilidad de las aplicaciones de un sistema a otro. En este objetivo son definidas tres interfaces:

a) ODL (Lenguaje de Definición de Objetos) El lenguaje de definición del objeto permite definir el modelo de datos. Es compatible con IDL, el lenguaje del OMG (Grupo de Administración de Objetos). Permite la definición de objetos complejos, de relación entre esos objetos y de métodos asociados a dichos objetos.

b) OQL (Lenguaje de Consulta al Objeto) El lenguaje de requerimientos permite consultar los objetos de estructuras complejas, de enviar mensajes a objetos, efectuar join y otras operaciones de tipo asociativo. Su sintaxis es del tipo SQL.

c) Conexión vía C++ y Smalltalk. Esta interfaz ("bindings", enlazamientos), especifica cómo se debe hacer la programación en C++ o Smalltalk de una aplicación sobre una base de datos que ha sido declarada en ODL. La conexión se basa sobre la noción de "puntero inteligente" que permite manejar los objetos persistentes como objetos ordinarios vía punteros persistentes.”

Hay una cosa más que podemos hacer con los campos, podemos asignar las limitaciones, por ejemplo: Si se asigna ÚNICO, obstáculo para cualquier columna, entonces no se podría insertar en ella cualquier valor que ya está en su misma columna. Ayudar a las limitaciones en el mantenimiento de la integridad de los datos del sistema.

SQL no es sino un conjunto de comandos / declaraciones y se clasifican en cinco grupos, a saber., Dql: Información de Query Language, LMD: Lenguaje de Manipulación de Datos, DDL: Data Definition Language, TCL: Lenguaje de control de las transacciones, DCL: Lenguaje de control de datos.

DQL: SELECT Dql: SELECT

DML: DELETE, INSERT, UPDATE LMD: SUPR, INSERT, UPDATE

DDL: CREATE, DROP, TRUNCATE, ALTER DDL: CREATE, DROP, TRUNCATE, ALTER

TCL: COMMIT, ROLLBACK, SAVEPOINT TCL: COMMIT, ROLLBACK, SAVEPOINT

DCL: GRANT, REVOKE DCL: GRANT, y REVOKE

RESUMEN

Los DBMS surgieron para responder a las necesidades de información de las organizaciones. Se trata de un conjunto de datos persistentes y de programas para acceder a ellos y actualizarlos.

Surgieron diversos sistemas administradores de archivos hasta llegar a la generación de las bases de datos relacionales que utilizaban lenguajes más accesibles y poderosos en la manipulación de datos como el SQL, QUEL y QBE. Estos deben contar con un modelo de datos, es decir, estructuras lógicas para describir los datos, y operaciones para manipularlos (recuperación y actualización).

Los nuevos DBMS deben tomar en cuenta varias operaciones para llevar con eficiencia la Administración, Validación, Creación y Eliminación de las Bases de Datos y poseer características de Orientación a Objetos y Persistencia.

Existen organismos en pro de la estandarización de este tipo de sistemas manejadores de bases de datos como el OMG (*Object Management Group*), la CAD Framework Initiative y el grupo de trabajo de ANSI. Las siguientes características de los OODBMS son las que se estudiaron: objetos complejos, identidad de objetos, encapsulamiento, herencia, sobreescritura, sobrecarga y completa capacidad computacional.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo
Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. y Zdonik, S. (1992).	223-240
Bertino, E. y Martino, L. (1995)	Completo
Date, C.J. (2001).	616-650
Hughes, J.G. (1991).	5-120
Johnson, J.L. (1997).	Completo
Silberschatz, A; Korth, HF. & Sudarshan, S. (2006)	Completo

Licenciatura: Informática

UNIDAD 4

DISEÑO

UNAM FCA CEDIGEC Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno diseñará modelos E/R y modelos de clases con la notación adecuada que permitan la posterior construcción de la base de datos en un manejador de bases de datos.

TEMARIO DETALLADO

(12 horas)

4. Diseño

- 4.1. Introducción al diseño
- 4.2. Modelo semántico
- 4.3. Modelo lógico
  - 4.3.1. E/R
  - 4.3.2. E/R extendido
- 4.4. Modelo físico
  - 4.4.1. Implementación de un E/R al modelo relacional
- 4.5. Modelo de clases (UML)

INTRODUCCIÓN

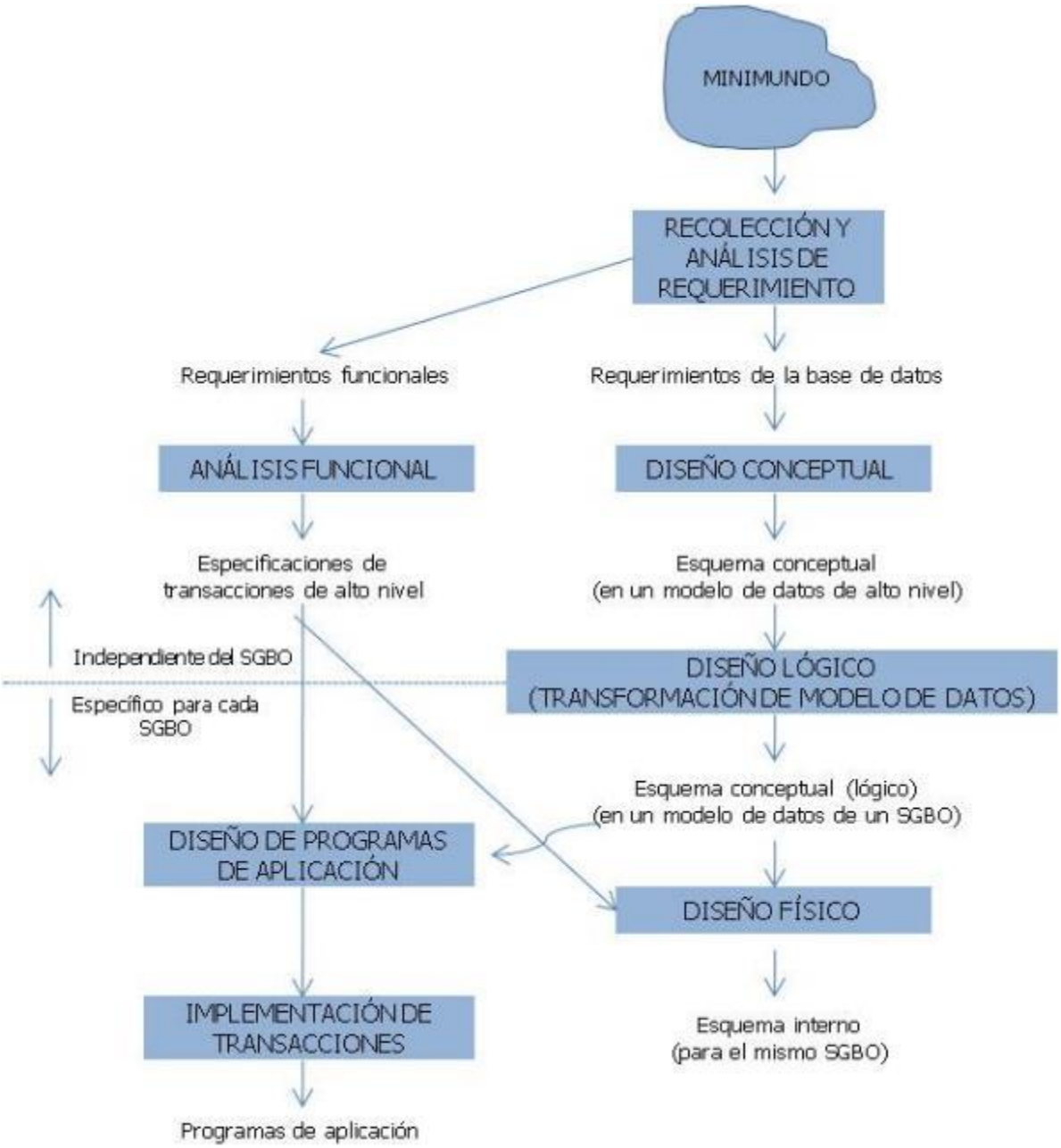
El tema que presentamos a continuación tiene gran relevancia para tu formación como informático, principalmente en el aspecto del desarrollo de sistemas de información. Así, una fase del proceso de desarrollo de sistemas es el diseño de la base de datos.

Con la información obtenida en la etapa de análisis se desarrolla una solución de almacenamiento de datos mediante un modelo (relacional, orientado a objetos, etc.). Por lo tanto, revisaremos en esta unidad la herramienta de modelado o diseño de datos más utilizada en la vida real: el modelo entidad-relación.

Aquí estudiarás los fundamentos teóricos de este modelo y los procedimientos para llevarlo a la práctica. Su principal ventaja es que resulta independiente del modelo de base de datos en el que será implementado.

El diseño es fundamental para la buena construcción de nuestra base de datos. Este proceso nos permitirá definir las restricciones de integridad del mundo real y llevarlas a la base de datos, además nos brindará la posibilidad de determinar cómo serán las estructuras de almacenamiento de datos y sus relaciones.

El éxito de la base de datos y del sistema en general depende (entre otras cosas) de un buen diseño. Planearlo bien nos garantiza eliminar problemas de redundancia y actualización de datos, además de ahorrar recursos de cómputo y facilidad de consultas de datos.

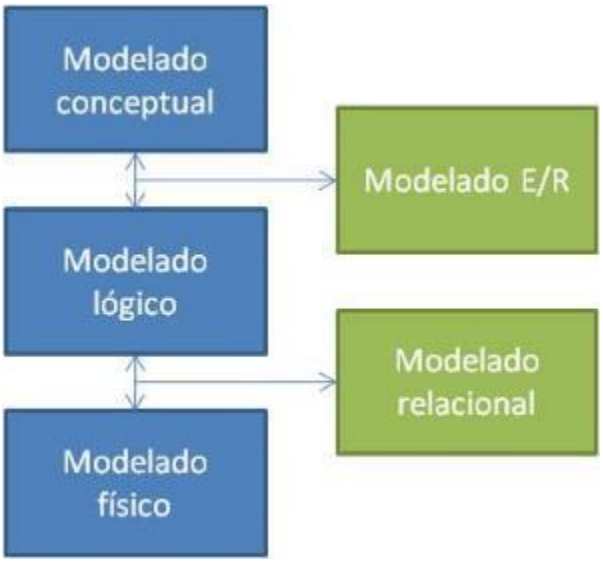

MINIMUNDO
RECOLECCIÓN Y ANÁLISIS DE REQUERIMIENTO
Requerimientos funcionales
Requerimientos de la base de datos
ANÁLISIS FUNCIONAL
DISEÑO CONCEPTUAL
Especificaciones de transacciones de alto nivel
Esquema conceptual (en un modelo de datos de alto nivel)
Independiente del SGBO
Específico para cada SGBO
DISEÑO LÓGICO (TRANSFORMACIÓN DE MODELO DE DATOS)
Esquema conceptual (lógico) (en un modelo de datos de un SGBO)
DISEÑO DE PROGRAMAS DE APLICACIÓN
DISEÑO FÍSICO
IMPLEMENTACIÓN DE TRANSACCIONES
Esquema interno (para el mismo SGBO)
Programas de aplicación

4.1. Introducción al diseño

El diseño de bases de datos consiste en traducir un conjunto de datos inmersos en una realidad a un modelo manejable en una base de datos. Esta traducción debe decirnos la estructura lógica de las estructuras para almacenar los datos y las restricciones sobre estos.

Diseñar es más un arte que una ciencia, o al menos no es posible encuadrarlo en rigurosos principios científicos -si acaso el proceso de normalización que se revisó en la Unidad 2. En las siguientes secciones conocerás una metodología de diseño. Esta metodología estará basada en el desarrollo de un modelado semántico a través de un modelo entidad-relación y su posterior transformación en un modelo relacional. El proceso de diseño de base de datos puede verse como una interacción de tres etapas generales:

Proceso del diseño de base de datos



4.2. Modelo semántico

El modelo relacional ha demostrado ser un modelo muy útil para el desarrollo de sistemas de información organizacionales y en algunas otras áreas de la actividad humana. Sin embargo, desde sus inicios ha sido criticado porque no representa mucha de la semántica de la realidad. El hecho de estar basado en relaciones y nada más que relaciones, le impide captar el significado de las interacciones entre éstas, su jerarquía y las restricciones asociadas a estas interacciones.

Por esta razón, entre otras, fue propuesta una novedosa manera de modelar la realidad. Su principal característica era que intentaba representar en buena medida la semántica de la realidad. La propuesta se conoce como “Modelado Semántico”. En palabras de CJ Date (2001, p. 419), lo que se trató de resolver es que “por lo regular los sistemas de bases de datos sólo tienen una comprensión muy limitada de lo que significa la información de la base de datos”.

El modelado semántico tuvo su principal desarrollo en un modelo que veremos adelante: el modelo Entidad-Relación (E/R). Este modelo materializó el objetivo del modelado semántico y tuvo las ventajas necesarias para convertirse en el más utilizado en nuestro tiempo. Claro que resulta importante aclarar que tampoco capta todos los sentidos de la realidad que representa, pero sí la gran mayoría.

4.3. Modelo Lógico

Mediante un modelado semántico de la realidad es posible obtener un modelo de datos en el nivel conceptual que nos permitirá representar la realidad en una forma dirigida al almacenamiento de datos. Como ya lo habíamos mencionado, es el modelo Entidad-Relación la manera más utilizada para hacerlo. Los elementos y características de este modelo se verán en la siguiente sección.

A partir del modelo conceptual se deriva un modelo lógico específico para un tipo de base de datos: orientado a objetos, relacional, jerárquico, etc. En nuestro caso, el modelo lógico se basará en el modelo relacional.

4.3.1. E/R

El modelo Entidad-Relación (E/R) ayuda a realizar un diseño de bases de datos sin atender a un modelo en especial (jerárquico, relacional, orientado a objetos).

Fue propuesto por Peter Chen en el artículo “The entity-relationship model - Toward a unified view of data” (1976). En éste, Chen propone utilizar un enfoque más natural del mundo real basado en entidades e interrelaciones. Hoy en día, podríamos decir que más bien existe una familia de modelos, ya que muchos autores han realizado propuestas que han enriquecido al modelo E/R.

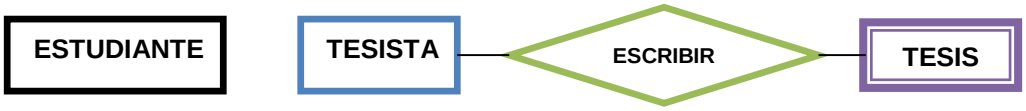
El modelo E/R permite visualizar la base de datos desde un alto nivel de abstracción. Los elementos interesantes de la realidad que queremos modelar son las entidades, además modelamos sus atributos y las interacciones entre ellas.

Una ventaja del modelo E/R es que utiliza una representación gráfica conocida como Diagrama Entidad-Relación (DER). Es importante mencionar que existen distintas representaciones de un DER en las que cambian los aspectos gráficos, pero se modelan los mismos elementos. A continuación, se detallan los elementos del modelo E/R, así como su representación gráfica en el DER.

Entidad

Una entidad es cualquier “objeto (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos” (De Miguel y Marcos, 2000, p. 49). Las entidades se agrupan en tipos de entidades, de los cuales podemos identificar ejemplares, por ejemplo, el Sr. Ruiz sería un ejemplar del tipo de entidad PERSONA. Los tipos de entidades se representan con un rectángulo con el nombre del tipo de entidad en su interior. Existen entidades llamadas débiles, cuya existencia depende de que exista otra entidad, denominada fuerte o regular; las entidades débiles se representan con doble rectángulo.

Algunos ejemplos son los siguientes. El primer ejemplo corresponde a la entidad ESTUDIANTE. El segundo, se refiere a una interrelación (ESCRIBIR) entre una entidad fuerte, TESISTA, y una entidad débil, TESIS. Ésta última es débil porque su existencia depende de la entidad TESISTA.



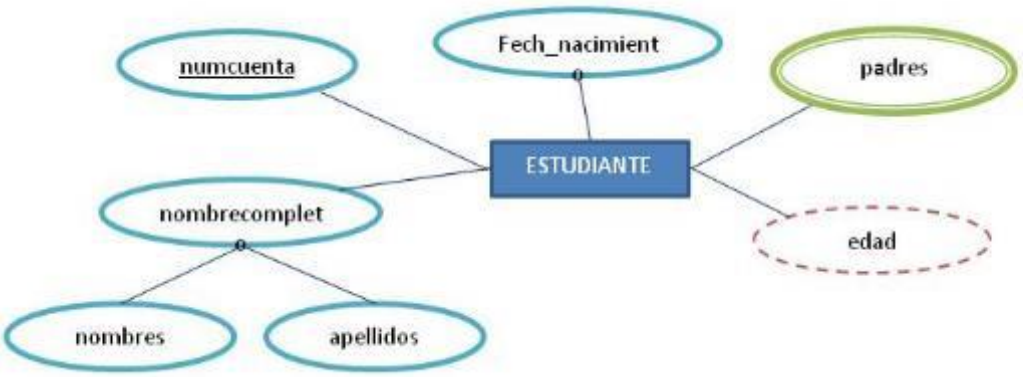
Entidades

Atributo

Un atributo es una característica o propiedad de un tipo de entidad o interrelación, que puede tomar distintos valores. Al conjunto de valores se le distingue como dominio. Los atributos se representan con elipse que incluye el nombre en su interior.

Entre los atributos tenemos a los:

a) Atributos compuestos
b) Atributos clave
c) Atributos multivaluados
d) Atributos derivados



Tipos de atributos en una entidad

Interrelación

Una interrelación es “una asociación, vinculación o correspondencia entre entidades” (De Miguel, 2001, p. 51). Igual que las entidades, las interrelaciones se agrupan en diferentes tipos. Por ejemplo, el tipo de interrelación IMPARTE es la vinculación entre las entidades PROFESOR y CURSO. Un ejemplar de esta interrelación sería la vinculación entre el profesor Sr. Ruiz y el curso Bases de datos. Las interrelaciones se representan gráficamente con un rombo que incluye el nombre del tipo de interrelación en su interior.

En algunos casos de diseño, será posible ver una interrelación como un tipo especial de entidad débil. Para decidir si lo modelaremos como entidad o como interrelación dependerá de la conveniencia en el modelo y de la realidad que analicemos.

Las interrelaciones cuentan con varias características:

a) Grado
b) Tipo
c) Papel (rol)
d) Cardinalidad

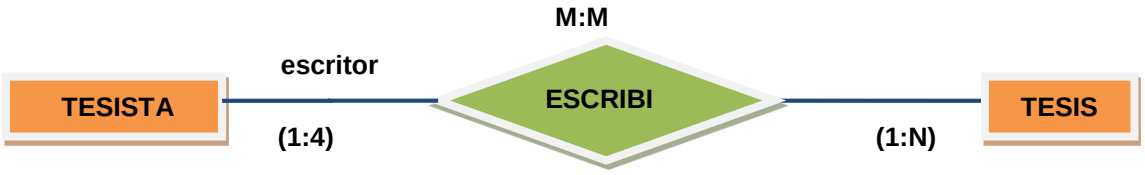
Las interrelaciones cuentan con varias características:

Grado	"Es el número de tipos de entidad que participan en un tipo de interrelación" (De Miguel, 2001, p. 61). Una interrelación de grado dos se refiere a la vinculación de dos tipos de entidades. Un tipo especial de interrelación de grado dos es la reflexiva, que asocia un tipo de entidad consigo misma.
Tipo	"Es el número máximo de ejemplares de un tipo de entidad que pueden estar asociados" (De Miguel, 2001, p. 62). Los tipos son uno a uno (1:1), uno a muchos (1:M) y muchos a muchos (M:M).
Papel (rol)	"Es la función que cada uno de los tipos de entidad realiza en el tipo de interrelación" (De Miguel, 2001, p. 63). Cuando la función de una entidad en la interrelación es ambigua o no se puede inducir de manera clara, se recomienda colocar el papel (rol) en la línea que conecta a la entidad con la interrelación.
Cardinalidad	"Se define como el número máximo y mínimo de ejemplares de un tipo de entidad que pueden estar interrelacionados con un ejemplar del otro tipo" (De Miguel y Marcos, 2000, p. 63). Los

	valores que podrían tomar son (0,1), (1,1), (0,N) o (1,N), donde N significa muchos ejemplares.

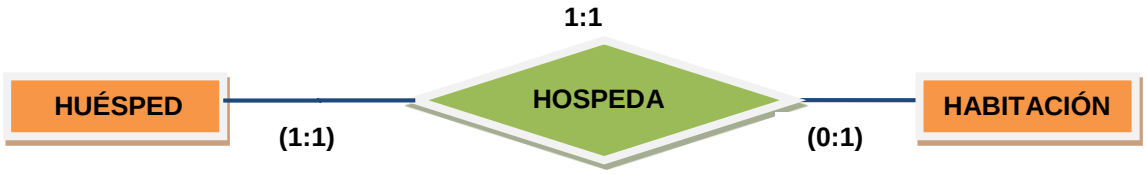
Para comprender mejor las interrelaciones, veamos el siguiente ejemplo. En éste podemos observar una interrelación de tipo muchos a muchos (M:M), entre TESISTA y TESIS. Esto significa que muchos tesistas escriben una tesis, pero también que muchas tesis son escritas por un tesista. Pero, si miramos bien, tenemos una restricción establecida por la cardinalidad en TESISTA, ésta nos indica que pueden participar en la relación al menos 1 tesista y como máximo 4; en otras palabras, una tesis es escrita por mínimo 1 y máximo 4 tesistas.

En el caso de la cardinalidad de TESIS, ésta nos indica que un estudiante escribe 1 o muchas tesis, sin restricción de cuántas. También se indica el papel o rol del tesista en la interrelación, consistiendo su rol en “escritor”.



Ejemplo 1. Interrelaciones

En otro ejemplo, las situaciones cambian. De acuerdo con este tipo de relación, una habitación sólo puede estar ocupada por un huésped y un huésped sólo puede ocupar una habitación (1:1). La cardinalidad nos indica que un huésped sólo puede tomar una habitación como máximo (1:1), pero una habitación puede estar desocupada o puede estar ocupada por máximo un huésped, por lo que la cardinalidad de su interrelación es 0:1.



Ejemplo 2. Interrelaciones

4.3.2. E/R extendido

El modelo E/R puede captar la mayoría de las características semánticas para la base de datos, pero es posible extenderlo para captar otros aspectos. Esta extensión es conocida como modelo Entidad-Relación extendido. Las características de modelo extendido son:

* Especialización
* Generalización
* Agregación
* Herencia de atributos

(Véase, Silberschatz y otros, 2006, pp. 190-196)

4.4. Modelo físico

El modelo Entidad-Relación nos permite representar la realidad sobre la cual vamos a almacenar información. Mediante el Diagrama Entidad-Relación (DER), captamos el significado de todo aquello que queremos almacenar en la base de datos. Gracias a esto, podemos pasar de un modelo conceptual a un modelo lógico y finalmente a uno físico.

Para realizar lo anterior, es necesario determinar el modelo de datos con el que construiremos nuestra base. En nuestro caso, utilizaremos el modelo relacional, que como recordarás, está basado en relaciones o tablas. En el modelo físico se determinan aspectos físicos de almacenamiento como: registros, punteros, direccionamiento y asignación de espacio para memorias intermedias.

La realidad es que hoy en día son los manejadores de bases de datos los que se encargan de esto y el diseñador de bases de datos no juega un papel decisivo en estos aspectos físicos del almacenamiento. La parte donde sí se involucra el experto en bases de datos es en la modificación de parámetros de rendimiento del manejador.

Dado que los RDBMS (Sistema Administrador de Bases de Datos Relacionales) son los más utilizados hoy en día, es común que el proceso de diseño de una base de datos se realice inicialmente con el modelo E/R a través de un DER y después se realice un mapeo o trasformación a relaciones o tablas de un modelo relacional. A continuación veremos cómo se hace esto.

4.4.1. Implementación de un modelo E/R al modelo relacional

Una vez realizado el modelado para obtener el Diagrama Entidad-Relación, es necesario seguir un conjunto de pasos para convertirlo en un modelo relacional. Las tres reglas generales para realizar esto son:

1) Los tipos de entidades se convierten en relaciones.
2) Las interrelaciones N:M se transforman en relaciones.
3) Para las interrelaciones 1:N se realiza una propagación de clave a partir del lado de 1 hacia el lado de N.

A continuación, explicaremos detalladamente los pasos necesarios para esta derivación. Es necesario recordar que, de forma ideal, se debe mantener la semántica del modelo conceptual en el modelo lógico. Desafortunadamente, no siempre es posible hacerlo. Las reglas específicas para derivar un modelo Entidad-Relación son:

I. Derivación de dominios

Debe crearse los dominios de todos los atributos. Para esto utilizamos la sentencia CREATE DOMAIN, por ejemplo:

```
CREATE DOMAIN edo_civil AS CHAR(1)
CHECK (VALUE IN ('S', 'C'));
```

II. Derivación de entidades

Cada tipo de entidad se transforma en una relación o tabla. Para ello, utilizamos la sentencia CREATE TABLE.

III. Derivación de atributos

Todo atributo de una entidad se transforma en una columna de su relación correspondiente.

a) Atributos identificadores (claves o principales). Se transforman en claves primarias. Para ello se usa la restricción de integridad PRIMARY KEY. Por ejemplo:

cod_alumno INTEGER CONSTRAINT pk_cod_alumno PRIMARY KEY

b) Atributos identificadores alternativos (claves candidatas). Se les aplica la restricción de integridad UNIQUE. Por ejemplo:

rfc CHAR(13) CONSTRAINT un_rfc UNIQUE

c) Atributos no identificadores (no clave, no principales). Se les aplica la restricción de integridad de NOT NULL, sólo si es necesario. Por ejemplo:

nombre VARCHAR(30) NOT NULL

IV. Derivación de interrelaciones

a) Interrelaciones N:M. La interrelación se transforma en una relación que tendrá como superclave los atributos identificadores (claves o principales) de las entidades que relaciona. Los atributos de la relación resultante (relación o tabla de interrelación) se convierten en llaves foráneas (claves ajenas) mediante la restricción FOREIGN KEY. Para esta restricción es necesario indicar la tabla padre y la llave primaria en esa tabla. Además, podemos indicar las restricciones de borrado y actualización en cascada. Por ejemplo:

```
CREATE TABLE imparte
(
cod_curso INTEGER,
cod_profesor INTEGER,
PRIMARY KEY (cod_curso, cod_profesor),/*Primaria
compuesta*/
CONSTRAINT fk_cod_curso FOREIGN KEY
(cod_curso)
        REFERENCES curso(cod_curso)
        ON DELETE CASCADE
        ON UPDATE CASCADE, /* Foránea con
borrado y actualización en cascada*/
CONSTRAINT fk_cod_profesor FOREIGN KEY
(cod_profesor)
        REFERENCES profesor(cod_profesor)
        ON DELETE CASCADE
        ON UPDATE CASCADE /* Foránea con
borrado y actualización en cascada*/
)
```

Otro aspecto importante por derivar es la cardinalidad de las entidades participantes. Como sabemos, existen casos en los que hay restricciones de negocio que impiden que un determinado número de ejemplares de una entidad interactúe con ejemplares de otra entidad. En este caso es necesario agregar una restricción de aserción. Desafortunadamente no todos los manejadores de bases de datos incluyen esta posibilidad. En ese caso, debemos utilizar disparadores (*triggers*) de integridad. Por ejemplo, supongamos que un curso sólo puede ser impartido por cuatro profesores, es decir, cardinalidad (0, 4). En ese caso tendríamos que crear la siguiente aserción:

```
CREATE ASSERTION profesor_curso
CHECK NOT EXIST (SELECT COUNT(*)
                 FROM imparte
                 GROUP BY cod_curso
                 HAVING COUNT(*)>=4);
```

b) Interrelaciones 1:N. La manera general de transformar esta interrelación consiste en propagar (pasar) el atributo identificador (clave o principal) de la entidad con cardinalidad 1 hacia la entidad con cardinalidad N. Como resultado la interrelación se pierde, es decir, no se convierte en entidad como en el caso de N:M.

El atributo propagado se convierte en llave foránea, para el cual debemos indicar si existe borrado en cascada. La cardinalidad de la interrelación también debe ser implementada como ya establecimos, esto es, utilizando restricciones de aserción o triggers. Adicionalmente, si la cardinalidad indica que el número de ejemplares puede ser 0, la llave foránea aceptará valores nulos. Si, por el contrario, indica que al menos un ejemplar debe estar relacionado con uno o más ejemplares (cardinalidad 1), la llave foránea debe tener restricción de NOT NULL.

c) Interrelaciones 1:1. No hay regla fija para la transformación de esta interrelación. Lo más común es aplicar la regla 4.2 (interrelaciones N:M), aunque puede ser posible la aplicación de la regla 4.1 (interrelaciones 1:N). Es importante notar que al aplicar la regla 4.2 necesitamos decidir en qué dirección se propaga la llave primaria. Para decidirlo podemos tomar en cuenta al menos dos aspectos:

→ Pasar la llave primaria de una entidad fuerte a una débil es generalmente más conveniente.

→ Pasar la llave primaria de la entidad con cardinalidad (1, 1) a la entidad con cardinalidad (0, 1) permite aplicar NOT NULL a la llave foránea.

V. Atributos de interrelaciones

Si la interrelación se transforma en relación, los atributos de la interrelación se transforman en columnas de la relación resultante. En caso de que se aplicase una propagación de identificador (clave), los atributos de la interrelación pasan en la misma dirección que dicha clave.

VI. Otras restricciones

Podemos encontrar restricciones de valores posibles para un atributo. Para transformarlas al modelo relacional utilizamos la restricción CHECK. Además, si el manejador de bases de datos no acepta creación de dominios, con el CHECK es posible restringir los valores de un dominio.

VII. Transformación de dependencias de existencia e identificación

En este caso se realiza una propagación de identificador (clave) asegurándonos de poner en la llave foránea, si es necesaria, la restricción de borrado y actualización en cascada. Cuando se trate de una dependencia de identificación, la llave primaria propagada se combina con algún atributo de la entidad débil para formar una superclave.

VIII. Derivación de tipos y subtipos

No hay una manera de conservar la semántica de los tipos y subtipos en el modelo relacional. Lo más recomendable es crear relaciones para cada supertipo y para cada subtipo. Después, propagar la clave principal del supertipo en cada relación de los subtipos. Esta clave propagada será, además de llave foránea, la llave primaria de cada subtipo. Finalmente, si queremos asegurarnos de que un tipo no pueda aparecer en varios subtipos, tendremos que usar disparadores (triggers) que revisen que la clave del tipo no exista ya en algún subtipo.

4.5. Modelo de clases (UML)

El modelo de clases es una herramienta para modelado de sistemas orientado a objetos. Muestra la estructura estática del sistema mediante clases. Estas clases representan objetos involucrados en el sistema; pueden mantener relaciones entre ellas, ser especializaciones de otras clases o tener dependencias.

Algunas de estas clases pueden terminar almacenadas en la base de datos, pero no necesariamente. El diagrama de clases es producto del modelado de sistemas orientado a objetos, de tal modo que si la base de datos es orientada a objetos, sería de esperarse que las clases del sistema sean persistentes de forma transparente.

El problema radica en si la implementación se realiza en una base de datos relacional. En este caso parece necesario, además de este diagrama, elaborar un Diagrama Entidad-Relación y luego pasarlo a tablas. El diagrama de clases ayuda a modelar las entidades que intervienen en el sistema junto con sus atributos, pero recordemos que no está orientado al modelado de datos.

Existen estándares de representación de este tipo de modelos de clases. Uno muy conocido y utilizado es el Lenguaje de Modelado Unificado (UML). Se trata de una norma de modelado mediante aspectos gráficos auspiciada por el Grupo de Administración de Objetos (*Object Management Group*, OMG, dedicado al desarrollo de especificaciones y estándares para crear componentes de software.

Hemos visto un acercamiento al diseño de base de datos llamado modelado semántico. Éste se realiza mediante un modelo Entidad-Relación que se representa en un Diagrama Entidad Relación. Una vez hecho, hemos obtenido un modelo conceptual de la realidad que queremos almacenar en la base de datos.

Después, siguiendo las reglas que se propusieron en la sección correspondiente, se transformó el modelo E/R en un modelo específico de base de datos. En nuestro caso, se transformó en un modelo relacional, el cual representa el modelo lógico de la base de datos que se implementa en un manejador de base de datos específico mediante instrucciones de SQL. Es precisamente éste último paso, el objetivo de la siguiente unidad.

RESUMEN

En esta Unidad se vieron otros modelos para obtener una aplicación más representativa de la realidad y que se ajuste a las necesidades del usuario.

Asimismo, pudimos tener en cuenta que el diseño de bases de datos es más que un arte, refiriéndose más a un conjunto de actividades con las cuales se representa una solución para alguna aplicación. El proceso de diseño de bases de datos está integrado por tres etapas a saber: el modelo conceptual, el modelo lógico y el modelo físico, y sus dos refinaciones los modelos E/R y el Relacional.

También pudimos ver que los RDBMS (Sistema Administrador de Bases de Datos Relacionales) son los más utilizados hoy en día; es común que el proceso de diseño de una base de datos se realice inicialmente con el modelo E/R a través de un DER y después se realice un mapeo o trasformación a relaciones o tablas de un modelo relacional.

De igual forma, el Modelo de Clases UML para sistemas orientados a objetos. Muestra la Estructura Estática del sistema con clases, pero no todas las clases pueden terminar almacenadas en la base de datos. Entre los estándares de representación de este tipo de modelos de clases más conocido es el Lenguaje de Modelado Unificado (UML). Se trata de una norma de modelado mediante aspectos gráficos auspiciada por el Grupo de Administración de Objetos (Object Management Group, OMG) dedicado al desarrollo de especificaciones y estándares para crear componentes de software.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo
Chen, Peter P. (1976),	Completo
Churcher, C. (2007).	Completo
De Miguel, A., Piattini, M y Marcos, E. (2000).	Completo
Johnson, J. L. (1997).	Completo
Silberschatz, A; Korth, HF. & Sudarshan, S. (2006).	87-192

Licenciatura: Informática

UNIDAD 5

CONSTRUCCIÓN

UNAM FCA
CEDIGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno construirá una base de datos relacional, producto de un modelo E/R y cada uno de los elementos que componen dicho modelo, haciendo uso de un manejador de bases de datos relacional.

TEMARIO DETALLADO

(10 horas)

5. Construcción

5.1. Roles del implementador
5.2. Tablas
5.3. Integridad
5.4. Índices
5.5. Vistas
5.6. Triggers
5.7. Stored Procedures
5.8. Manejo de Transacciones
5.9. Recuperación

INTRODUCCIÓN

Una vez que hemos realizado el diseño de la base de datos y obtenido el modelo relacional (unidad 4), ha llegado el momento de implementarlo (programarlo) en un manejador de base de datos específico. Para realizar esto será necesario determinar cuál nos conviene. Hoy en día existe una gran variedad de ellos, desde los que cuentan con licencia de uso comercial hasta los basados en la postura del software libre.

Algunos de los aspectos relevantes para la selección son: el volumen de información que pueden almacenar (generalmente medido en *megabytes* o *terabytes*); el número de usuarios que pueden acceder al mismo tiempo; sus ventajas en el manejo de transacciones; su velocidad de respuesta a un número considerable de transacciones, y sus características para imponer seguridad a los datos.

Como recordarás, la programación en una base de datos relacional se realiza con el lenguaje SQL, y a lo largo de esta unidad conocerás de manera general el uso de este lenguaje. El objetivo de la materia de bases de datos no consiste en que aprendas a programar con SQL, por lo que sólo haremos mención de los comandos más importantes y necesarios. Te explicaremos para qué sirve cada uno de los objetos de una base de datos, cuál es su objetivo y sus principales características.

SSMSE 2005

Afortunadamente el lenguaje SQL es un estándar y, si bien hay diferencias de programación entre los distintos manejadores de bases de datos, lo que revisemos en esta unidad aplicará para cualquiera de ellos. En otras palabras, desarrollaremos la unidad sin pensar en un software específico. En caso de que sea necesario, haremos alguna anotación adicional. Si te interesa conocer al detalle aspectos de los principales manejadores de bases de datos puedes revisar los capítulos 26 al 29 del libro de Silberschatz y otros (2006, pp. 807-921), en los que presentan cuatro manejadores: PostgreSQL, Oracle, DB2 UDB y SQL Server.

Revisemos entonces los objetos de base de datos que utilizamos para implementar una base de datos útil en un sistema de información.

5.1. Roles del implementador

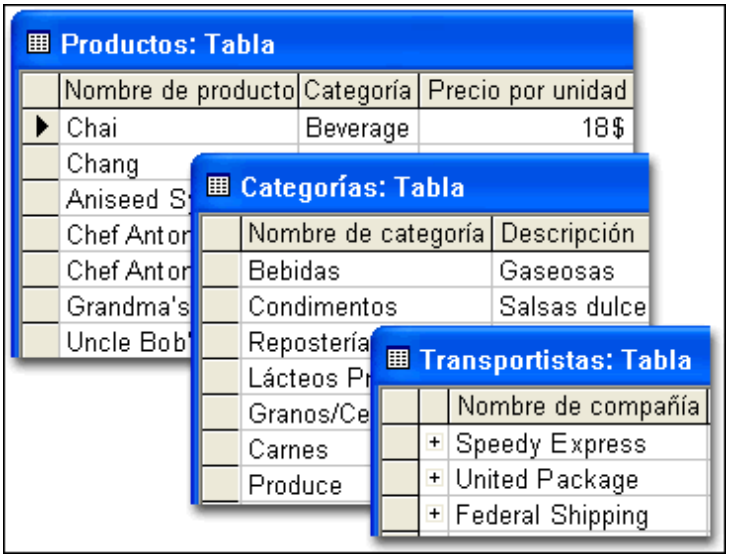
El diseñador de bases de datos entrega el modelo lógico de tablas al implementador o programador de bases datos para que construya la base de datos en el sistema manejador. Sus principales roles o actividades son:

1. Programar las estructuras de almacenamiento (tablas).
2. Implementar las reglas de integridad mediante restricciones o triggers.
3. Agilizar las consultas mediante la creación de índices.
4. Encapsular consultas en vistas.
5. Programar procedimientos almacenados para implementar la lógica de procesamiento de datos dentro de la base de datos.

Para que conozcas a fondo estas actividades, en los siguientes temas revisaremos cada uno de los objetos programables en una base de datos.

5.2. Tablas

Las tablas son un conjunto de filas y columnas que nos permiten almacenar los datos bajo el enfoque de un modelo relacional. Como sabes, el término tabla se conoce de manera formal como relación, al renglón como tupla y a la columna como atributo. Son en éstas donde almacenamos los datos mediante instrucciones en DML (Lenguaje de Manipulación de Datos).



Instrucciones en DML

Para crear una tabla utilizamos el comando CREATE TABLE, con la siguiente sintaxis general[5]:

[5] La sintaxis que presentamos está simplificada en comparación con la que puedes encontrar en los manuales de programación de SQL de los sistemas de base de datos. Ya que nuestro objetivo no es presentar la manera de programar sino el entendimiento de los objetos programables de una base de datos, creemos que es necesario ser más específicos. Para entenderla mejor, recuerda que los [ ] encierran elementos que pueden o no incluirse en la programación y que el uso del carácter | es para proponer distintas opciones agrupadas entre llaves { }.

```
CREATE  TABLE nombre_tabla
(
nombre_columna tipo_dato
[DEFAULT valor_default]
[CONSTRAINT nombre_constraint TIPO (condición)],
nombre_columna tipo_dato...,
nombre_columna tipo_dato...
)
```

Para ejemplificar la manera de crear una tabla veamos un ejemplo. Pensemos que deseamos crear la siguiente tabla:

Autor		
Idautor	Nombres	Apellidos
1	Ignacio Manuel	Altamirano
2	Manuel	Payno
3	Jorge Luis	Borges
4	Sor Juana Inés	De la Cruz
5	Julio	Cortázar

Siguiendo la plantilla de sintaxis expuesta arriba, podemos crear la tabla de la siguiente manera:

```
CREATE TABLE autor
(
idautor integer CONSTRAINT pkautor PRIMARY KEY,
nombres varchar(40) NOT NULL,
apellidos varchar(40) NOT NULL
);
```

Como puedes observar en el código anterior, creamos una tabla de nombre auto con tres columnas. Cada una de ellas se declara por separado y termina su declaración con una coma, la última columna no tiene una coma al final ya que no le sigue ninguna columna más. Cada definición de columna se compone de nombre, por ejemplo: **idautor**, su tipo de dato, en este caso entero **integer**, y la declaración de una restricción de integridad o **constraint**.

El *constraint* se forma por la palabra reservada **CONSTRAINT**, seguida de un nombre para esa restricción, *pkauto*, y el tipo de restricción, en este caso de clave primaria, **PRIMARY KEY**. Las otras columnas se definen de la misma forma y sólo cambian de tipo de dato y de restricción por una de **NOT NULL**.

Otras operaciones que podemos realizar con una tabla son:

- Renombrarla.
- Renombrar una columna.
- Agregar o eliminar columnas.
- Cambiar el tipo de dato de una columna.
- Agregar o eliminar una restricción (*constraint*).
- Agregar o eliminar un DEFAULT.

Estas operaciones se realizan con el comando ALTER TABLE.

Un DEFAULT es un valor predefinido para una columna que se inserta automáticamente si no definimos un valor específico. Otra operación es la de eliminar una tabla con todo y sus datos que ejecutamos con el comando DROP TABLE.

5.3. Integridad

La integridad de una base de datos se establece con restricciones, en inglés *constraints*. Ponemos restricciones en las columnas de una tabla para que acepten o rechacen ciertos valores. Con ello evitamos que los datos ingresados a la base sean incorrectos o inapropiados. La siguiente lista incluye posibles restricciones de una columna y la descripción de los valores que acepta o rechaza.

RESTRICCIÓN	VALORES QUE ACEPTA	VALORES QUE RECHAZA
NOT NULL	Distintos a Null	Valor Null
CHECK	Los que cumplen con la condición establecida por la restricción	Los que no cumplen con la condición establecida por la restricción.
UNIQUE	Valores únicos en la columna	Valores repetidos en la columna.
PRIMARY KEY	Valores únicos distintos a Null	Valores repetidos y valor Null.
FOREIGN KEY	Valores que existan previamente en la clave primaria	Valores que no existan previamente en la clave primaria.

Tabla de restricciones de una columna

Las restricciones se programan dentro de la definición de cada columna en un CREATE TABLE. En la siguiente tabla puedes ver ejemplos de la programación de cada tipo de *constraint*.

Restricción	Ejemplo
NOT NULL	titulo text NOT NULL
CHECK	sexo char(1) CONSTRAINT chksexo CHECK (sexo IN ('F', 'M'))
UNIQUE	rfc char(13) CONSTRAINT unirfc UNIQUE
PRIMARY KEY	idlibro integer CONSTRAINT pklibro PRIMARY KEY
FOREIGN KEY	REFERENCES autor (idautor) ON DELETE CASCADE ON UPDATE CASCADE

Ejemplos de programación de restricciones

El *constraint* de FOREIGN KEY es particularmente más elaborado. La parte de REFERENCES hace referencia a la tabla padre de la relación, es decir, aquella donde está la clave primaria a la que hace referencia la clave foránea que se está declarando. La restricción incluye dos cláusulas:

1. ON DELETE action
2. ON UPDATE action

Éstas indican la acción que se va a ejecutar con los valores de la clave foránea, en caso de que los valores de la clave primaria sean borrados o actualizados. El parámetro *action dependiente de cláusulas,* se refiere a las siguientes posibles acciones:

NO ACTION	• En caso de borrado o actualización, no hacer nadar.
RESTRICT	• En caso de borrado o actualización, producir error
CASCADE	• En caso de borrado o actualización, los valores de la clave foránea son borrados o actualizados al mismo valor de la clave primaria.
SET NULL	• En caso de borrado o actualización, asignar nulos a los valores de la clave foránea.
SET DEFAULT	• En caso de borrado o actualización, asigna los valores por *default* a la columna que es clave foránea.

En seguida puedes ver el código completo para crear una tabla con todos los tipos de *constraints*.

```
CREATE TABLE empleado
(
idempleado integer CONSTRAINT pkempleado PRIMARY
KEY,
nombres varchar(40) NOT NULL,
apellidos varchar(40) NOT NULL,
rfc char(13) CONSTRAINT unirfc UNIQUE,
idarea integer REFERENCES area(idarea)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

5.4. Índices

Los índices permiten incrementar el rendimiento de la base de datos haciendo más rápida la ejecución de consultas. En otras palabras, una cláusula SELECT con WHERE encontraría más rápido los registros que cumplen la condición.

Se hacen sobre una o más columnas de una tabla. Es importante resaltar que debemos poner índices sólo en columnas usadas frecuentemente en nuestras expresiones de comparación con la cláusula WHERE.

Para crear un índice utilizamos la cláusula CREATE INDEX y para borrarlo se hace con el comando DROP INDEX, por ejemplo:

```
CREATE INDEX idx_nombres ON empleado(nombres);
DROP INDEX idx_nombres;
```

Los Índices de Tablas asociados a campos de tipo numérico y carácter, pueden ser compuestos en virtud de que se pueden vincular dos campos para formar un solo índice como por ejemplo en la Tabla “Producto” y los Atributos Cod_Prod y Suc_Prod se pueden combinar para formar el índice compuesto Pedido. El comando sería el siguiente:

```
CREATE INDEX Pedido ON Producto(Cod-Prod, Suc_Prod);
```

Para ahondar más en el tema, favor de consultar los sitios de Internet, en particular el video de Intersystem (2008). “El rendimiento de la base de datos” de 6 partes.

5.5. Visitas

Una vista es un objeto de la base de datos que almacena una consulta. Funciona como una tabla, pero no existe físicamente en la base de datos, se genera de forma dinámica. Una vista nos permite encapsular consultas que utilizamos de forma recurrente, nos evita escribirlas de nuevo. También nos ayuda a manipular consultas muy complejas de una forma más sencilla.

La instrucción SQL para consultar datos es SELECT. La sintaxis básica general de esta instrucción es:

```
SELECT nombre_columna, nombre_columna,...
FROM nombre_tabla
WHERE condición
```

A partir de este tipo de consultas se crean las vistas con el comando CREATE VIEW, de la siguiente forma:

```
CREATE VIEW nombre_vista AS
SELECT ...
```

Veamos algunos ejemplos de vistas creadas a partir de diversas consultas:

Seleccionar todos los renglones y todas las columnas:	Seleccionar algunas columnas:	Seleccionar renglones a partir de una condición expresada en la cláusula WHERE:
CREATE VIEW vempleados AS SELECT * FROM empleado;	CREATE VIEW vemp AS SELECT idempleado, nombres, apellidos FROM empleado;	CREATE VIEW vempleado2 AS SELECT nombres, apellidos FROM empleado WHERE idempleado = 2;

Recuperar registros de una sola tabla es muy inusual. En la realidad siempre obtenemos datos de diferentes tablas, a veces de muchas. Es importante entonces conocer la manera de "unirlas" para poder obtener valores almacenados en columnas de unas y de otras. Esto lo realizamos con la operación relacional llamada junta o *join*. Existen varios tipos de *join*, entre ellos podemos mencionar:

1. Cross join

El resultado es un producto cartesiano, es decir, una combinación de todos los valores de una tabla contra todos los valores de otra tabla.

2. Inner join

El resultado es un conjunto de registros que resultan de la combinación de dos o más tablas, siempre y cuando existan columnas en común y los valores de dichas columnas coincidan. Este *join* necesita de una cláusula ON que iguale las columnas en común.

3. Outer join

El resultado es un *inner join* que además incluye aquellos valores donde no hay coincidencia en el origen del lado izquierdo (LEFT OUTER JOIN) o del lado derecho (RIGHT OUTER JOIN) o aquellos que no coinciden en ningún lado (FULL OUTER JOIN).

A continuación, un ejemplo de consulta de dos tablas.

```
CREATE VIEW vlibroautor AS
SELECT libro.titulo, autor.nombres, autor.apellidos
FROM libro INNER JOIN autor
ON (libro.idautor = autor.idautor);
```

El resultado de este *join* no incluiría los libros sin autor en caso de que no haya coincidencia entre las claves idautor de las dos tablas. Esto sucede porque el *inner join* rescata sólo aquellos que cumplen la condición de igualdad entre columnas en común. Si quisiéramos rescatar los libros que sí tienen autor y los que no tienen, deberíamos utilizar un *outer join*.

```
CREATE VIEW vlibroautor AS
SELECT libro.titulo, autor.nombres, autor.apellidos
FROM libro LEFT OUTER JOIN autor
ON (libro.idautor = autor.idautor);
```

Se trata de un *left outer join* porque la tabla libro está a la izquierda del *join*. Si lo que quisiéramos fuera todos los autores con y sin libro asociado, entonces tendríamos que programar un *right outer join*.

```
CREATE VIEW vlibroautor AS
SELECT libro.titulo, autor.nombres, autor.apellidos
FROM libro RIGHT OUTER JOIN autor
ON (libro.idautor = autor.idautor);
```

5.6. Triggers

Un *trigger* ejecuta un determinado procedimiento almacenado en la base de datos cuando se realiza cualquier modificación en una tabla. Un *trigger* es una función que se "dispara" antes o después de la instrucción que actualiza la tabla a la que está asociado. Son usados para implementar reglas de integridad de datos, auditoría de tablas importantes y actividades de mantenimiento de datos.

Para crear un *trigger* debemos usar la siguiente sintaxis general:

```
CREATE TRIGGER nombre_trigger
{ BEFORE | AFTER } { DELETE OR UPDATE OR INSERT }
ON nombre_tabla FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE nombre_procedimiento;
```

Las opciones de BEFORE o AFTER permiten definir si el trigger se ejecutará antes o después de un evento INSERT, DELETE o UPDATE. La cláusula FOR EACH ROW indica que el procedimiento que dispara el *trigger* será ejecutado por cada renglón actualizado por el evento INSERT, DELETE o UPDATE. Si por el contrario se indica FOR EACH STATEMENT, el procedimiento será disparado una sola vez.

Para borrar un *trigger* contamos con la sentencia:

```
DROP TRIGGER nombre_trigger ON nombre_tabla;
```

Siendo redundantes, un *trigger* dispara un procedimiento almacenado, que puede realizar cualquier acción en la base de datos. De esta manera, podemos crear un *trigger* para los siguientes casos:

- Después de borrar o actualizar en la tabla ventas, actualizar en la tabla venta_total el total de venta.
- Después de modificar, eliminar o actualizar la tabla venta, registrar en la tabla venta_bitacora el usuario que modificó la venta y la hora de modificación.
- Antes de actualizar o eliminar en la tabla cliente, revisar si el cliente no tiene registros en la tabla clientes_morosos, en caso de tener registros detener la actualización.
- Después de eliminar un registro de la tabla préstamo, insertar en la tabla prestamo_historico el registro eliminado.

5.7. Stored Procedures

Un procedimiento almacenado (*stored procedure*) es un conjunto de instrucciones en un lenguaje de programación que se almacenan como un objeto de la base de datos y puede ser ejecutado en cualquier momento. Por lo general estos procedimientos almacenados están escritos en un lenguaje peculiar que combina un lenguaje procedimental (*procedural*) con el SQL.

Este lenguaje se vuelve bastante poderoso en tanto que une declaración de variables, manejo de excepciones, expresión de ciclos y condicionales con las capacidades del SQL. Todos los manejadores de bases de datos incluyen un lenguaje de este tipo, por ejemplo: Oracle tiene el pl/sql, PostgreSQL el pl/pgsql, y SQL Server cuenta con el Transact-sql.

Todo procedimiento almacenado sigue una estructura basada en tres secciones:

1. Sección **declarativa**.
Permite declarar variables y cursores.

2. Sección **ejecutiva**.
Es donde se expresan las instrucciones que procesan los datos.

3. Sección **de excepciones**.
Aquí podemos manejar excepciones producidas por el procedimiento.

Hay tres tipos de **procedimientos almacenados**. Los procedimientos como tales, que no regresan valor; las funciones, que sí regresan algún valor; y los *triggers*, que como ya vimos, se ejecutan automáticamente cuando se actualiza la tabla a la que están vinculados. Tanto las funciones como los procedimientos reciben parámetros que utilizan dentro de la sección ejecutiva. Dependiendo del lenguaje que utilicemos podremos contar con parámetros de entrada, salida o entrada-salida.

La programación de procedimientos almacenados varía de manejador en manejador. Además, es muy importante, ya que nos permite encapsular la lógica del procesamiento de datos al interior de la base de datos. Este hecho es una ventaja en comparación con programar el procesamiento de datos en los programas de aplicación, ya que los *stored procedures* se ejecutan más rápido y están pre-compilados y probados desde antes.

Es posible programar dentro de la base de datos, desde cálculos completos de impuestos y procesamientos estadísticos hasta modificaciones complejas de datos. Desafortunadamente, por el desconocimiento de estos objetos de bases de datos, en muchos equipos de desarrollo se continúa programando fuera de la base de datos.

5.8. Manejo de transacciones

Una transacción de base de datos es un conjunto de dos o más instrucciones que modifican la información de la base (UPDATE, DELETE o INSERT), las cuales son tratadas como una unidad, de tal forma que se realizan todas o no se realiza ninguna. Una transacción puede terminar en una actualización de los datos (*commit*) o puede terminar sin actualización regresando la base de datos al estado consistente con el cual empezó (*rollback*).

Una transacción comienza con la palabra BEGIN TRANSACTION, todas las instrucciones siguientes a ella forman parte de esa transacción. Para que los cambios sean permanentes debe terminar con el comando COMMIT TRANSACTION, en caso de querer deshacer los cambios utilizaríamos ROLLBACK TRANSACTION.

Las transacciones se utilizan en los casos en los que las empresas requieren que las operaciones se completen como una unidad. La falta de actualización completa implicaría una falta de consistencia en la información. El ejemplo más claro es el de un cargo y su respectivo abono, sabemos que de no completarse las dos operaciones sufriríamos de un problema en los resultados financieros. Toda base de datos es susceptible de sufrir un error, originado por diversas fuentes, que impida que las operaciones se completen en conjunto.

El peligro de realizar sólo parte de la transacción es que algunos registros queden actualizados y otros no, dando como resultado información errónea. Para evitar estos problemas de consistencia, los manejadores de bases de datos implementan automáticamente un mecanismo de recuperación que revisaremos en la siguiente sección.

La siguiente información y comandos son de Librerías basadas en SQL SERVER 2005 del sitio de MSDN.

> Una transacción explícita es aquella en que se define explícitamente el inicio y el final de la transacción.
>
> Las aplicaciones de DB-Library y las scripts Transact-SQL utilizan las instrucciones BEGIN TRANSACTION, COMMIT TRANSACTION, COMMIT WORK, ROLLBACK TRANSACTION o ROLLBACK WORK de Transact-SQL para definir transacciones explícitas.
>
> **BEGIN TRANSACTION**. Marca el punto de inicio de una transacción explícita para una conexión.
>
> **COMMIT TRANSACTION o COMMIT WORK**. Se utiliza para finalizar una transacción correctamente si no hubo errores. Todas las modificaciones de datos realizadas en la transacción se convierten en partes permanentes de la base de datos. Se liberan los recursos ocupados por la transacción.
>
> **ROLLBACK TRANSACTION** o **ROLLBACK WORK**. Se utiliza para eliminar una transacción en la que se encontraron errores. Todos los datos modificados por la transacción vuelven al estado en el que estaban al inicio de la transacción. Se liberan los recursos ocupados por la transacción.

También puede utilizar transacciones explícitas en OLE DB. Llame al método **ITransactionLocal::StartTransaction** para iniciar una transacción. Llame al método **ITransaction::Commit** o **ITransaction::Abort** con ***fRetaining*** establecido en FALSE para finalizar la transacción sin iniciar otra automáticamente. En ADO, se utiliza el método **BeginTrans** en un objeto **Connection** para iniciar una transacción explícita. Para finalizar la transacción, se llama a los métodos **CommitTrans** o **RollbackTrans** del objeto **Connection**.

En el proveedor administrado de **SqlCliente** de ADO.NET, utilice el método **BeginTransaction** en un objeto **SqlConnection** para iniciar una transacción explícita. Para finalizar la transacción, llame a los métodos **Commit()** o **Rollback()** del objeto **SqlTransaction**.

La API de ODBC no acepta transacciones explícitas, sólo acepta transacciones de confirmación automática y transacciones implícitas.

El modo de transacciones explícitas se mantiene solamente durante la transacción. Cuando la transacción termina, la conexión vuelve al modo de transacción en que estaba antes de iniciar la transacción explícita, es decir, el modo implícito o el modo de confirmación automática. (MSDN, Transacciones explícitas)

5.9. Recuperación

Como se estudió en el tema anterior, la recuperación es un mecanismo automático de un sistema de bases de datos que entra en acción cuando ocurre un error en medio de una transacción y ésta no se ha completado. El principio fundamental de la recuperación es que se hacen todas las operaciones o ninguna; si ya se habían registrado algunas operaciones, éstas deben deshacerse. Es mejor que los datos no se actualicen a que se actualicen de manera incompleta y, por tanto, errónea.

En seguida verás dos esquemas que muestran los dos comportamientos de una transacción, uno termina con un commit y el otro con rollback. En el segundo caso, los datos no se actualizan ya que ocurrió un error antes de terminar la transacción.

Estado consistente inicial			Transacción	Estado consistente final		
			1) Update saldo = 3000			
Id	Nombre	Saldo	Where id =1;	Id	Nombre	Saldo
1	Alberto	2000		1	Alberto	3000
2	Alma	5000	2) Update saldo = 2500	2	Alma	2500
			Where id =2;			

Cuadro 5.1. Transacción sin error que termina en *commit*

Estado consistente inicial			Transacción	Estado consistente final		
			1) Update saldo = 3000 Where id =1;			
Id	Nombre	Saldo		Id	Nombre	Saldo
1	Alberto	2000		1	Alberto	2000
2	Alma	5000	ERROR	2	Alma	5000
			2) Update saldo = 2500 Where id =2;			

Cuadro 5.2. Transacción con error que termina en rollback

En el primer esquema (Cuadro 5.1. Transacción sin error que termina en *commit*), dado que no existió error en la transacción, ésta finaliza con un *commit* y los datos se actualizan. Pero en el segundo caso (Cuadro 5.2. Transacción con error que termina en *rollback*), observa que ningún registro es actualizado a pesar de que la operación (1) sí se realizó; al presentarse el error ésta es desecha y la tabla queda como en el estado inicial.

Hemos descrito una serie de objetos de bases de datos que son programables y que permiten implementar el sistema relacional de almacenamiento de datos junto con sus restricciones. Con estos, el programador construye una base de datos funcional para un sistema de información. En la siguiente unidad revisaremos cómo administrar la base de datos una vez que está construida.

RESUMEN

En esta unidad se vieron las actividades del Diseñador de la Base de Datos, en qué consisten y su impacto en la entrega del diseño de la Base de Datos; la aplicación de comandos o instrucciones con las cuales se crean tablas o Bases de Datos Complejas como son CREATE TABLE y los valores por DEFAULT así como sus restricciones CONSTRAINT. También se definen las Llaves Primarias y Foráneas con las cuales se vinculan las tablas sobre criterios establecidos.

Se estudiaron los valores de NOT NULL, CHECK, UNIQUE, PRIMARY KEY, FOREING KEY, y su aplicación en los atributos específicos de las Tablas para conservar la integridad de las Tablas en la Base de Datos; de igual forma de vieron los Comandos SELECT con WHERE, CREATE INDEX y la creación de archivos temporales para realizar los movimientos y actualizaciones de las Tablas.

Se crearon Vistas para guardar consultas efectuadas con anterioridad, como una forma de no emplear más de un archivo y ahorrar el empleo de memoria interna (con el empleo del comando CREATE VIEW y el nombre de la vista). Se abordó la forma de Almacenamiento de los Procedimientos para poder guardar procedimientos empleando comandos de SQL como son BEGIN TRANSACTION, COMMIT TRANSACTION y ROLLBACK TRANSACTION para confirmar, deshacer o confirmar una transacción en las Tablas dentro de la Base de Datos.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo
Date, C. J. (2001).	448-471
Geschwinde, E. y Schönig, H. (2002).	445-510
Mendelzon, A. (2000).	Completo.
Melton J. y Eisenberg, A. (2004)	Completo
Pérez, C. (2004).	Completo

UNIDAD 6

ADMINISTRACIÓN

UNAM FCA
CEDIGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno conocerá y realizará las actividades de un Administrador de Bases de Datos a fin de administrar óptimamente las bases de datos a su cargo y hacer un resguardo adecuado de la información.

TEMARIO DETALLADO

(12 horas)

6. Administración

6.1. Roles del administrador
6.2. Seguridad
6.3. Respaldo
6.4. Otras actividades

INTRODUCCIÓN

En la unidad 1 se mencionó que uno de los usuarios más importantes de un sistema de bases de datos es el administrador o DBA (Database Administrator). Un DBA es el experto responsable de asegurar la continua funcionalidad y operación eficiente de las bases de datos de una organización y de las aplicaciones que acceden a ellas.

Imagen predeterminada Office, control

La actividad del DBA es de suma importancia, ya que en estos días las bases de datos son vitales para las organizaciones. Este almacenaje de datos tiene la finalidad de ofrecer una ventaja competitiva a las organizaciones mediante el acceso a información oportuna y veraz. Veamos en seguida las actividades de este personaje.

6.1. Roles del administrador

Las principales tareas que realiza el DBA son:

a) Administración del software (servidor) de bases de datos
- Instalación
- Configuración
- Monitoreo del sistema manejador de bases de datos
- Actualización del sistema manejador de bases de datos

b) Implementación de medidas de seguridad

c) Operaciones de respaldo y recuperación

d) Exportación y recuperación de datos

e) Ajustes de rendimiento (*Performance and Tunning*)

6.2. Seguridad

Actualmente el tema de la seguridad es un aspecto fundamental que no debe omitirse en todo sistema de cómputo y, en especial, en un sistema de bases de datos. Esta actividad consiste en garantizar que los datos sean accesibles únicamente al personal autorizado. Asimismo, debe tener especial cuidado en reducir los riesgos y vulnerabilidades del sistema de bases de datos, a fin de impedir cualquier intrusión de usuarios no autorizados que puedan extraer o modificar los datos. En esta época, en la que la información se ha vuelto un activo crítico en las organizaciones, la seguridad de bases de datos es una gran responsabilidad para el DBA.

Divisoft: Pass-Security

El DBA implementa un esquema de seguridad basado en los siguientes elementos principales: un login para cada usuario, un password, un nombre de usuario asociado al login y grupos de usuarios. Para determinar el esquema se toman en cuenta los siguientes aspectos:

1. Determinar las tareas que los usuarios van a realizar en la base de datos.

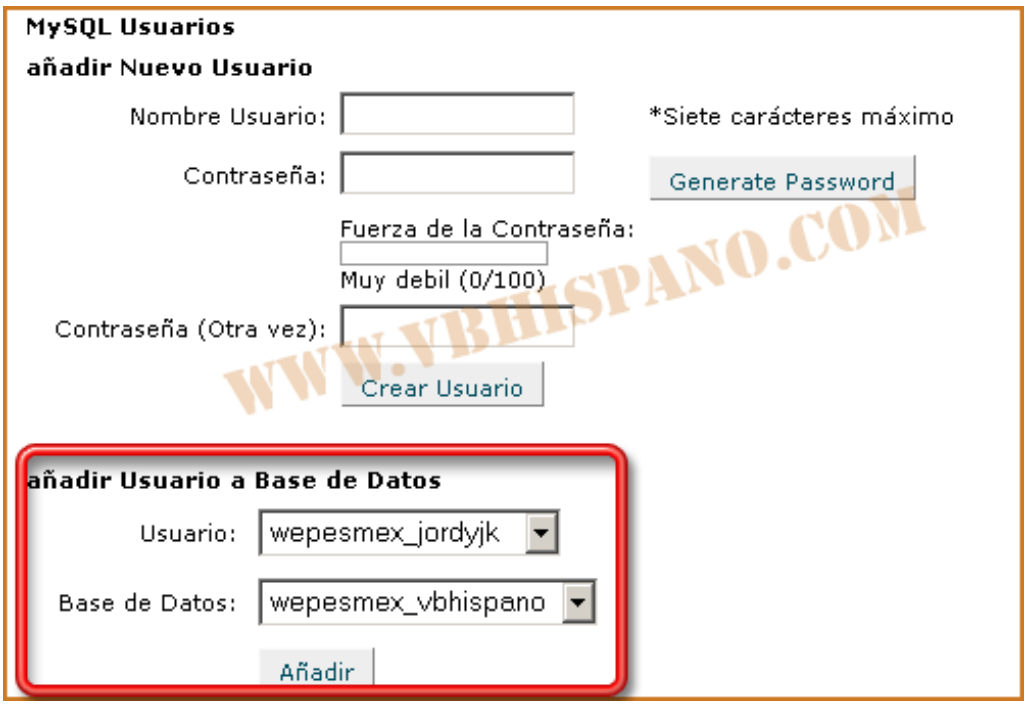
Se establecen las funciones de cada usuario y las acciones que le estarán permitidas realizar: actualizar, capturar o consultar los datos.

2. Agrupar de manera lógica a los usuarios con tareas comunes.

Estos grupos se basan en lo que harán los usuarios con la base.

Una vez abordados estos aspectos, la implementación del esquema de seguridad se lleva a cabo de la siguiente manera:

a) Crear grupos por cada base de datos usando nombres acordes con la organización de la empresa.
b) Crear un login por cada usuario.
c) Asignar una base de datos por defecto a cada login.
d) Asociar a cada login un nombre de usuario.
e) Asignar cada nombre de usuario a uno de los grupos determinados.



Ejemplo de cómo agregar nuevos usuarios a MySQL (vBulletin Hispano)

Otro aspecto de seguridad de bases de datos por considerar son los privilegios que cada usuario o grupo tienen sobre los datos. Estos privilegios son:

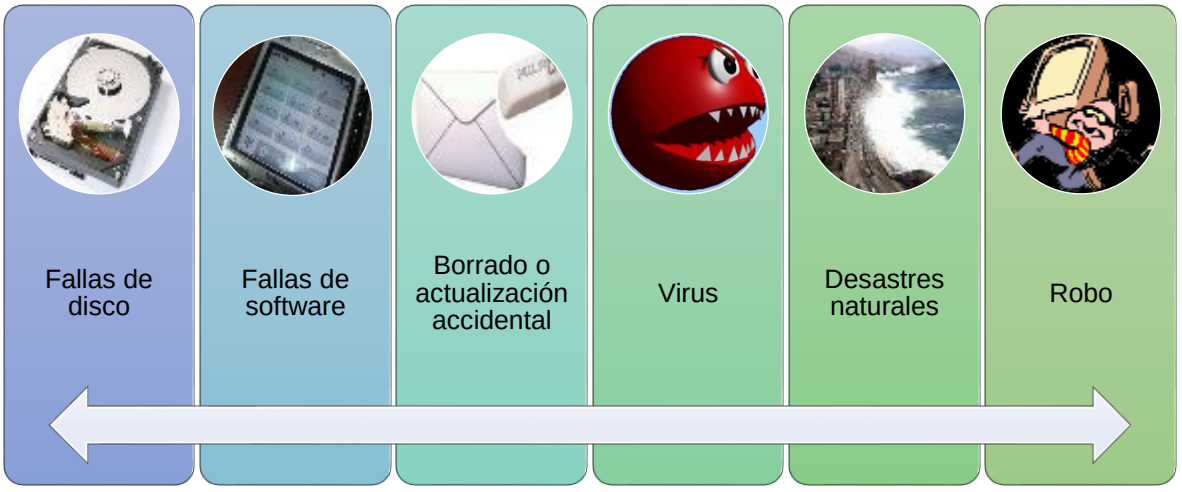
Privilegio	Acción que permite
SELECT	Seleccionar datos de una tabla, vista o columna.
INSERT	Insertar nuevos datos a una tabla o vista.
UPDATE	Actualizar datos existentes en una tabla, vista o columna.
DELETE	Borrar datos de una tabla o vista.
EJECUTAR	Ejecutar procedimientos almacenados.

En el siguiente cuadro se describe las actividades de administración de usuarios y privilegios con su correspondiente comando SQL.

Actividad	Comando SQL
Crear usuarios	CREATE USER
Modificar usuarios	ALTER USER
Eliminar usuarios	DROP USER
Crear grupos	CREATE GROUP
Borrar grupos	DROP GROUP
Asignar privilegios	GRANT
Quitar privilegios	REVOKE

6.3. Respaldo

Un respaldo de bases de datos consiste en hacer copias de las tablas de sistema, los objetos creados por el programador (tablas, vistas, procedimientos almacenados, restricciones, etc.) y los datos del usuario. Es el DBA el encargado de realizar esta labor mediante un esquema de respaldo (*backup*). El respaldo es lo que permite al administrador recuperar los datos en caso de una contingencia, como puede ser:



Para establecer su esquema, el DBA toma en cuenta los siguientes aspectos:

a) ¿Con qué periodicidad debe realizarse el respaldo?
b) ¿Qué se debe respaldar?
c) ¿Qué medio electrónico se debe usar para el respaldo?
d) ¿Debe efectuarse en línea o fuera de línea?
e) ¿Existe un mecanismo para asegurarnos que el respaldo se hizo correctamente?
f) ¿Dónde se almacenarán los respaldos?
g) ¿Cuánto tiempo deben conservarse los respaldos?

h) ¿Deben ser hechos de forma manual o automática?
i) Si son automáticos ¿cómo se verifican?
j) Cuando ocurre una falla ¿cuánto tiempo toma restaurar las bases de datos?

Algunas de las respuestas las encontramos al conocer el volumen de transacciones que se realizan sobre las bases de datos. Pensemos que un respaldo toma tiempo y distrae al procesador de su actividad normal, por lo que el DBMS (Sistema de gestión de base de datos) deja de atender con la misma velocidad las transacciones de los usuarios. Por ello muchos respaldos se hacen por la noche o los fines de semana y de manera automática. El principio básico consiste en hacer los respaldos en horas en las que se efectúe el menor número de transacciones.

Por otro lado, es primordial verificar que los respaldos se hayan copiado correctamente, ya que puede suceder que al momento de necesitarse no funcionen. Para probar su efectividad, se puede hacer un simulacro de falla en el DBMS, de esta manera se prueban los respaldos y se determina el tiempo que tarda el DBA en restaurar el servicio de la base de datos.

6.4. Otras actividades

Otras de las actividades que realiza el DBA son:

a) Determinación de los requerimientos de espacio lógico para la base de datos.
b) Monitoreo del espacio disponible para la base de datos.
c) Importar y exportar datos.
d) Mantener un sistema de tareas automáticas y alertas en caso de problemas con el DBMS.
e) Ajustes de configuración de rendimiento
f) Monitoreo constante del sistema de base de datos.

RESUMEN

En esta unidad se revisaron: Las principales tareas que realiza un DBA como la administración del software (servidor) de bases de datos, implementación de medidas de seguridad, operaciones de respaldo, recuperación, importación y exportación de datos y ajustes de rendimiento. De igual forma se abordaron otras actividades que realiza el DBA como la determinación de los requerimientos de espacio lógico para la base de datos, monitoreo del espacio disponible para la base de datos, importar y exportar datos, mantener un sistema de tareas automáticas y alertas en caso de problemas con el DBMS, ajustes de configuración de rendimiento, monitoreo constante del sistema de base de datos.

Asimismo, las medidas de seguridad aplicables a los usuarios de forma individual y en conjunto (asignación de login, password y privilegios y aspectos que implican estas medidas).

El orden de las actividades para ejecutar todo lo anterior y los comandos del lenguaje de SQL para implementar las medidas de seguridad. También los privilegios tienen su expresión en Comandos de SQL.

Los respaldos, elementos muy necesarios para salvaguardar el contenido e integridad de la base de datos contra contingencias como fallas de discos, virus, robos, etc., así como las tareas cotidianas para realizarlos como la frecuencia de los respaldos, horarios, datos dentro y fuera de línea y mecanismos de los respaldos entre otros.

Estas actividades también las efectuará el Administrador de la Base de Datos sobre y cada una de las bases de datos, sus tablas y su entorno.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo
Date, C.J. (2001).	440-471
Mullins, C. (2002).	203-660
Johnson, J.L. (1997).	Completo
Silberschatz, A; Korth, HF. & Sudarshan, S. (2006)	511-567

UNIDAD 7

NUEVAS TECNOLOGÍAS

UNAM FCA
CEDIGEC
Centro de Educación a Distancia y Gestión del Conocimiento

Plan 2012 2016 actualizado

OBJETIVO PARTICULAR

El alumno conocerá y utilizará nuevas tendencias en bases de datos para una explotación óptima de la información dentro de las organizaciones.

TEMARIO DETALLADO

(6 horas)

7. Nuevas tecnologías

7.1. Minería de datos

7.2. Dataware Housing

INTRODUCCIÓN

Las necesidades de información del ser humano y de las organizaciones evolucionan con el tiempo. Con el surgimiento de las bases de datos se pudo resolver el problema de registrar información útil para su futura recuperación; fueron posibles cálculos más rápidos y se le dio confiabilidad al procesamiento de grandes cantidades de transacciones. Hoy en día, podemos decir que estas necesidades están prácticamente cubiertas.

De esta forma, las necesidades de información se han vuelto más complejas. Las cantidades de información que guardan las bases de datos empresariales son enormes y su análisis se ha vuelto complicado. La primera necesidad ya no consiste en el almacenamiento y procesamiento, sino en el análisis. Por tanto, las nuevas tecnologías de bases de datos se están orientando al análisis automático de información para brindar soporte a las decisiones y generar conocimiento.

Antes, las necesidades de información eran del estilo: ¿cuánto fue vendido en Nuevo León en septiembre del año pasado?, cuya solución era aplicada con una consulta simple de SQL (*query*). Hoy en día, éstas son: ¿cuántas unidades se vendieron en Nuevo León con respecto a Guadalajara el año pasado, con respecto a este año? La solución ya no es un solo *query* de SQL; o por ejemplo, con base en las ventas de los últimos cinco años, ¿cuántas podrían ser las unidades vendidas este año? Esta situación obliga a usar métodos que permitan predecir el comportamiento de los datos.

Por otro lado, no es raro encontrar organizaciones que utilicen las bases de datos como centros de almacenaje del acontecer diario de la organización, ya que por ejemplo, esto les permite facturar y llevar control de stocks, pero pocas veces las usan para producir un análisis.

Entonces, ¿qué necesitan las organizaciones para utilizar este análisis automático de información y predecir el comportamiento? Primero, un almacén de datos construido con ese fin, un Data Warehouse, y después, un conjunto de estrategias y métodos de análisis, es decir, la minería de datos. Este tema presenta las características generales de ambas tecnologías.

7.1. Minería de datos

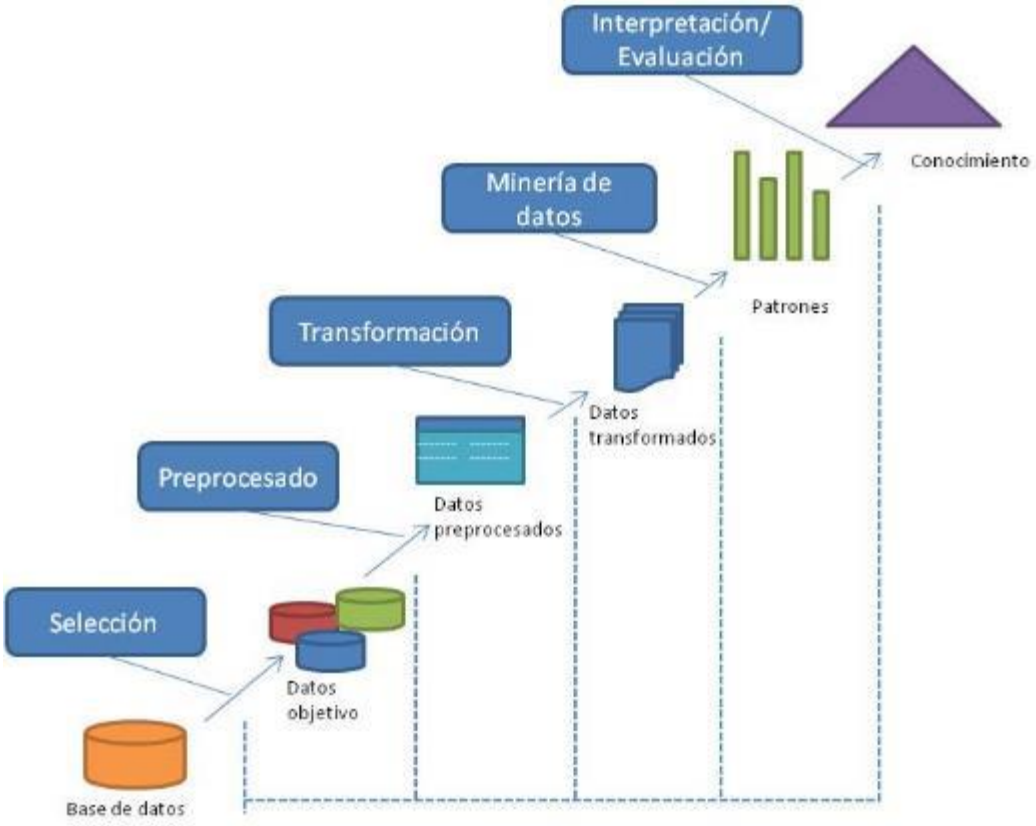
El volumen de datos que guardan actualmente las bases de datos se ha convertido en un recurso importante que debe analizarse. Este análisis permitiría describir y entender los procesos económicos y financieros de las organizaciones. Además, se cuenta con enormes bases de datos científicas como las relacionadas al genoma o la astronomía, que encierran conocimiento que debe ser descubierto.

Una manera de obtener este análisis es mediante el descubrimiento de patrones en los datos. Un patrón es una serie de características o de eventos que presenta alguna regularidad, suceden cada cierto tiempo, en las mismas circunstancias o con los mismos efectos. Si vemos una base de datos, será difícil percibir a simple vista si existe un patrón, de aquí que necesitemos utilizar técnicas especializadas llamadas en conjunto: minería de datos.

Definición

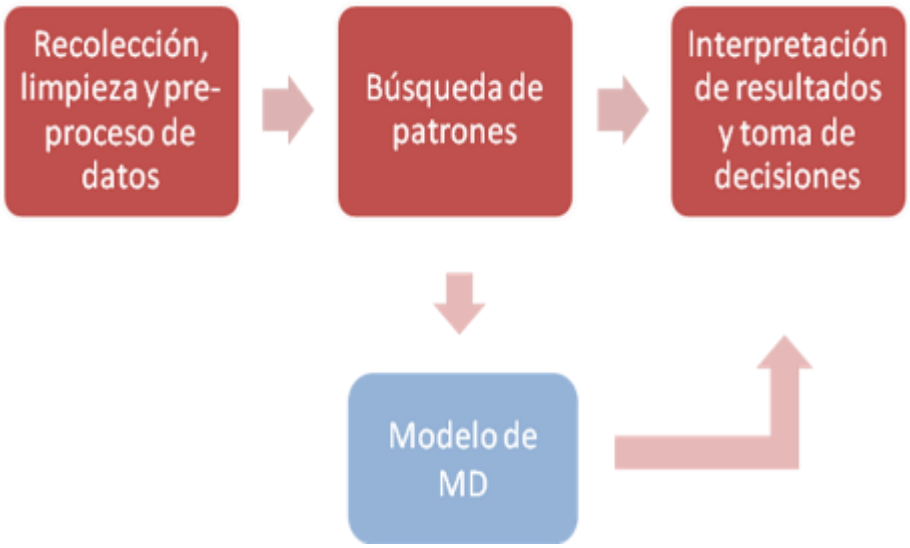
La minería de datos (**MD**) es también conocida como Descubrimiento de Conocimiento en Bases de Datos (*Knowledge Discovery in Databases, KDD*). Podemos definirla como la aplicación de técnicas estadísticas y de aprendizaje automático para encontrar patrones no triviales en bases de datos, que resulten de interés para el experto de un dominio determinado.

La MD tiene dos enfoques. El primero es descriptivo y consiste en encontrar patrones que nos permitan describir la situación actual de la organización. El segundo es predictivo y trata de obtener modelos que pronostiquen, a partir de los patrones, algún comportamiento interesante en el futuro.



Pasos generales de la minería de datos

La MD se lleva a cabo siempre bajo la idea de cooperación entre el experto del dominio (finanzas, mercadotecnia, ventas) y el personal de informática. A continuación, lo pasos generales de la MD:



Pasos generales de la Minería de Datos

El proceso de MD produce un modelo de MD, que puede ser descriptivo o predictivo, según los patrones identificados en los datos. Con él se analizan posibles cursos de acción para tomar decisiones. Este modelo puede ser desde una gráfica hasta una red neuronal. La recolección de datos se realiza a partir de las bases de datos transaccionales o de un Data Warehouse. Finalmente, debes saber que el modelo de MD puede ser estadístico o de aprendizaje automático.

Proceso de minería de datos

Primer paso
El proceso de MD comienza con la definición del objetivo de la minería. Esto se hace entre el experto del dominio y el experto en minería de datos. El objetivo puede ser explorar los datos de forma general (conocer las características de los clientes), obtener un modelo descriptivo o clasificador (obtener un modelo que clasifique a los clientes en sujetos de crédito o no), o demostrar una hipótesis (¿es cierto que los cursos más rentables se dan en Guadalajara a personas mayores de 40 años con puestos gerenciales y con ingresos superiores a $20,000?).
Segundo paso
Una vez identificado el objetivo, será necesario definir las fuentes de datos y los datos relevantes para el análisis. Estos datos pueden estar resumidos o acumulados.
Tercer paso
En seguida, se puede hacer un acercamiento preliminar a los datos. Por lo general se tratan de aplicar medidas estadísticas y obtener gráficas representativas de los datos, por ejemplo: promedios, máximos, mínimos, histogramas, diagramas de barras, pareto o análisis de correlación.
Cuarto paso
En esta etapa se deciden los métodos estadísticos o de aprendizaje automático que serán aplicados a los datos. Estos se verán en los apartados de estrategias y técnicas de minería.
Quinto paso
Este paso consiste en obtener los datos y adaptarlos al formato de entrada de nuestro método. Es común que se llegue a una sola tabla con datos no normalizados como insumo de la minería. Esta tabla suele llamarse “vista minable”.

Sexto paso
Se corren los procesos de aprendizaje automático para obtener un modelo a partir de los datos. Por lo general se utiliza un 70% de los datos originales, como datos de entrenamiento, y el resto se emplea para evaluar si el modelo obtenido es bueno.
Séptimo paso
Evaluación del modelo obtenido mediante medidas especiales (*precision* y *recall*). Esto nos permitirá saber si el modelo obtenido es bueno.
Octavo paso
Evaluar los resultados del proceso de MD con el experto del dominio para saber si son útiles y no triviales.

Estrategias de minería de datos

Las estrategias de minería de datos pueden clasificarse en: supervisadas (clasificación y estimación), no supervisadas (agrupamiento o clustering) y análisis de canasta.

1. Supervisadas

Clasificación

Construye modelos que puedan asignar o clasificar nuevos ejemplos a un conjunto de clases, definidas previamente. Los modelos de clasificación pueden servirnos para:

a) Clasificar personas con riesgo de sufrir paro cardiaco.
b) Clasificar aspirantes con bajo o alto riesgo crediticio.

Estimación

Consiste en determinar el valor de un atributo numérico. Ejemplos:

a) Estimar las ventas de próximo mes.
b) Estimar la probabilidad de que una tarjeta de crédito sea usada de manera fraudulenta.

2. No supervisadas

Agrupamiento

En este caso, no existen categorías predefinidas para clasificar ejemplos o instancias. Estas clases (clusters) se proponen como resultado del proceso de minería de datos. A este modelo de agrupamiento le acompañan medidas de cercanía entre los ejemplos o instancias agrupadas. Algunos ejemplos son:

a) Clustering de clientes de acuerdo con su historial de ventas.
b) Agrupamiento de nuevas especies de flores.

3. Análisis de canasta

Este análisis nos permite encontrar relaciones entre productos de acuerdo con sus ventas. Para este análisis suelen utilizarse algoritmos que producen reglas de asociación. Una regla sería la siguiente: “Cuando un cliente compra pan marca ‘Bready’ también compra lecha ‘Milky’”.

Técnicas de minería de datos

Las técnicas de MD se usan para aplicar una estrategia determinada a un conjunto de datos. Éstas cuentan generalmente con un algoritmo y una estructura de conocimiento. Las principales técnicas de minería son:

a) Reglas de Producción

Son reglas con la forma:

IF antecedente

THEN consecuencia

b) Regresión Lineal

Permite crear ecuaciones matemáticas con más de una variable independiente.

c) Árboles de Decisión

Es un conjunto de nodos que representan preguntas con las que se clasifica un ejemplo o instancia en una categoría predefinida.

d) Clustering

Como lo habíamos mencionado, consiste en encontrar clusters, también llamados cúmulos, nubes, agrupamientos o categorías en conjuntos de datos. Lo empleamos cuando no estamos seguros de las categorías que existen en ellos.

e) Reglas de asociación (análisis de canasta)

Representan asociaciones entre atributos contenidos en las bases de datos.

Aplicaciones de la minería de datos

Algunas de las actividades en las que resulta útil el uso de la minería de datos:

- ❖ Identificar patrones de fraudes en el uso de tarjetas de crédito.
- ❖ Identificar buenos y malos clientes.
- ❖ Predecir los clientes que podrían cambiar de banco.
- ❖ Encontrar relaciones entre indicadores financieros.
- ❖ Desarrollar modelos para diagnóstico o identificación de enfermedades.
- ❖ Predecir el perfil de los clientes que podrían adquirir nuevos productos.

7.2. Dataware Housing

Los avances en la tecnología de bases de datos y el desarrollo de un conjunto variado de sistemas manejadores han brindado invaluables beneficios al procesamiento automatizado de información. Así, las organizaciones se han convertido en continuos desarrolladores de bases de datos transaccionales para apoyar sus actividades económico-administrativas.

Junto con la observación anterior es necesario rescatar dos aspectos que no han sido favorables para la misma organización:

Primero, se volvió práctica común que distintos departamentos de una misma organización decidieran trabajar con distintos sistemas manejadores de bases de datos, incluso no resulta raro encontrar en una misma área distinto software de base de datos; situación que ha generado problemas para el análisis y procesamiento consolidado de datos.

Segundo, las organizaciones se preocuparon mucho - y con justa razón - por crear sistemas robustos para captar las operaciones diarias de la organización, y no contemplaron la labor del análisis de los datos. De esta manera, es normal encontrar en muchas organizaciones enormes bases de datos transaccionales y procesos de análisis de datos manuales basados en reportes automatizados, eso sí hay buena suerte.

Con el afán de responder a estas problemáticas, se propusieron nuevos modelos de bases de datos que permiten contar con un gran almacén consolidado de datos listo para aplicarle procesos de análisis automático. A esta nueva tecnología se le llamó *data warehousing*.

Conceptos básicos

Para Chaudhuri y Dayal (1997) el *data warehousing* consiste en una colección de tecnologías que permiten mejores y más rápidas decisiones:

> Data warehousing is a collection of decision support technologies, aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions.

El objetivo del *data warehousing* es brindar la tecnología necesaria para obtener resúmenes complejos de información y conocimiento. Asimismo, empleando las respectivas tecnologías, herramientas y metodologías, se podría crear, usar y mantener un *Data Warehouse*.

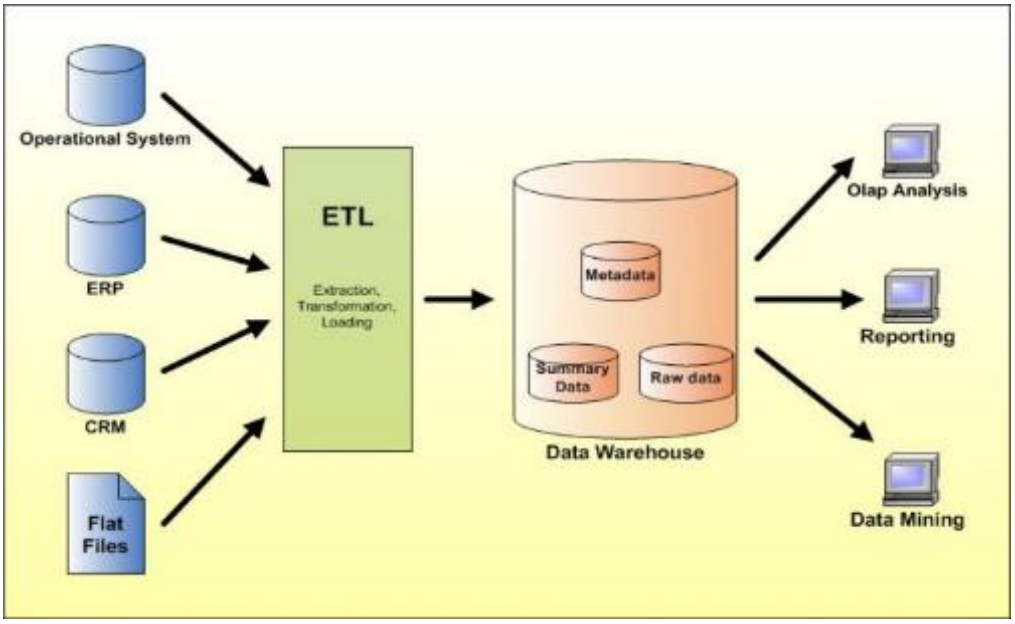


Diagrama de Data Warehose

Por su parte, el concepto de *Data Warehouse* (DW) se puede entender como un depósito centralizado de datos que ayuda al análisis del negocio. Es un almacén que unifica las bases de datos empresariales o departamentales sin importar el sistema manejador en el que se encuentren. Uno de los precursores de esta tecnología indica: “A data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management’s decisions” (Inmon, 2002).

Al respecto, también se suele escuchar el concepto de *Data Mart*. Vamos a entender un *Data Mart* como un *Data Warehouse* departamental, con las mismas características y componentes, pero a nivel de un departamento de la organización.

Es más, podríamos decir que un *Data Warehouse* se forma de todos los *Data Mart* de la organización.

Características de un Data Warehouse

A partir de la definición del llamado padre del *Data Warehouse*, William H. Inmon, podemos identificar sus características:

- **Orientados a temas**

Los datos de un DW deben estar recolectados para proporcionar información sobre los temas importantes de la empresa y no sobre sus operaciones diarias.

- **Integrado**

Los datos están integrados a partir de una variedad de bases de datos transaccionales provenientes de los sistemas OLTP de la organización. Los datos deben ofrecer una imagen corporativa general de ella.

- **No volátil**

Los datos de un DW no son eliminados ni modificados, ya que se quiere representar la historia de la organización.

- **Variante en el tiempo**

Los datos están asociados a periodos de tiempo, específicos y bien identificados.

Componentes de un DW

Los principales componentes de un DW incluyen el Data Warehouse como tal y algunos elementos adicionales como las fuentes y destinos del mismo.

- **Bases de datos transaccionales y otros recursos externos**

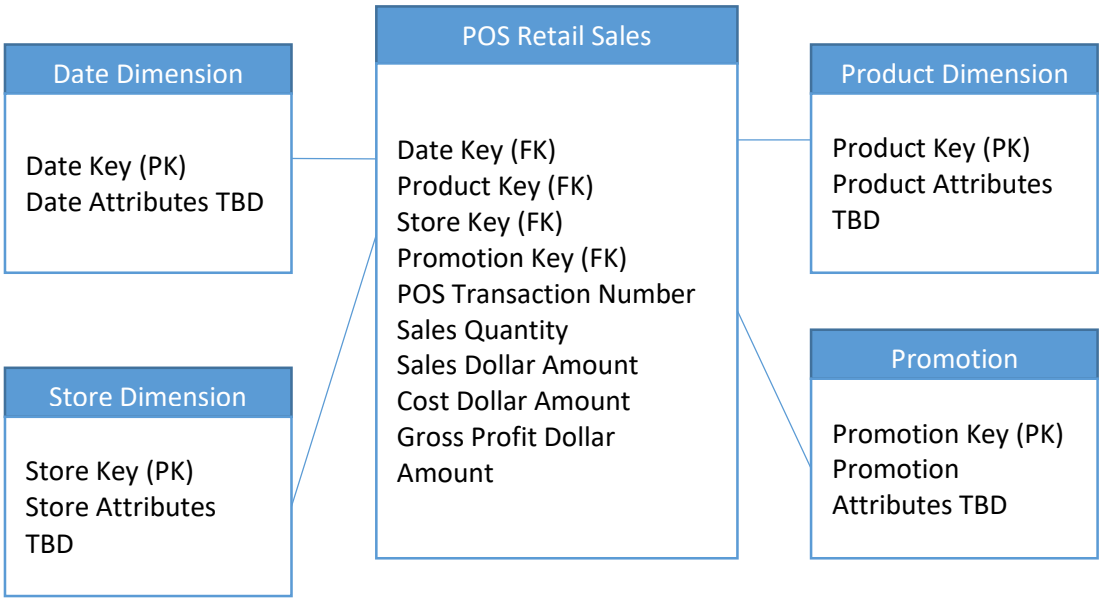
Son las fuentes de origen de los datos las que forman el DW. De ellas se extraen los datos y se cargan en las estructuras del DW. Es común que los datos se tengan que limpiar y transformar. Este proceso es conocido como ETL (*extract, tranform and load*).

- **Metadatos**

Son datos que describen el contenido del DW. Entre estos se incluyen el origen de los datos, el responsable y las descripciones sobre los tipos de resúmenes de datos.

- **La base de datos del DW**

Es la base de datos que almacena los datos del DW. Es normal observar que el almacenamiento se haya realizado bajo un modelo dimensional y no relacional. Este modelo consiste en un grupo de tablas de dimensiones que guardan una relación de uno a muchos con una tabla principal o tabla de hechos.



Measured facts in the retail sales schema

Ejemplo de un Modelo dimensional de almacenamiento (Kimball, 2002, p. 36)

- **Herramientas de consulta**

La construcción de un DW conlleva la creación de herramientas de procesamiento y análisis de los datos contenidos en él. Estas herramientas pueden ser parte de lo que se conoce como un sistema OLAP (*On Line Analytical Processing*) o ser herramientas de minería de datos.

- **Los usuarios**

Son el último componente de un DW y son los que explotan sus beneficios, ya sea mediante el resultado de la minería de datos o de sistemas de consulta para soporte de las decisiones.

Áreas que usan un DW

A continuación, se listan las diversas áreas que en la actualidad han adoptado la tecnología del DW para mejorar sus negocios:

Servicios financieros	Comercio minorista
Análisis de rentabilidad	Análisis y planificación de distribución
Gestión de riesgo y prevención de fraude	Telecomunicaciones
Análisis de marketing	Gobierno
Seguros	Salud

La DW se está convirtiendo en la base de los procesos analíticos de negocios y es importante que tú, como estudiante de informática, conozcas sus aspectos fundamentales, que hemos expuesto a lo largo de este tema.

RESUMEN

En esta unidad vimos que las Organizaciones requieren contar con información inmediata que ayude a la toma de decisiones, a fin de incrementar su productividad, encontrar hábitos de sus clientes y proveedores, planear las condiciones del mercado, responder a la competencia, etc.

Parte de esto se logra mediante el descubrimiento de ciertos patrones en los datos que contiene la Organización en sus bases de datos, a través de la aplicación de pasos y siguiendo estrategias, lo cual se denomina como *Minería de Datos* (MD).

Los avances en la tecnología de bases de datos y el desarrollo de un conjunto variado de sistemas manejadores han brindado invaluables beneficios al procesamiento automatizado de información. Así, las organizaciones se han convertido en continuos desarrolladores de bases de datos transaccionales para apoyar sus actividades económico-administrativas.

Con el afán de responder a estas problemáticas, se han propuesto nuevos modelos de bases de datos que permiten contar con un gran almacén consolidado de datos listo para aplicarle procesos de análisis automático. A esta nueva tecnología se le llamó *Data Warehouse* (DW), que consiste en una colección de tecnologías que permiten mejores y más rápidas decisiones.

BIBLIOGRAFÍA

SUGERIDA

Autor	Capítulo
Chaudhuri, S., Dayal, U. (1997).	Completo
Gutiérrez Hernández, G. y Barranco Serrano, V. (2008).	Completo
Ibáñez Carranza, A. R. (2008).	Completo
Inmon, William H. (2002).	Completo
Kimball, R. (2002).	Completo
Mallach, E. (2000).	Completo
Reyes García, C. T. (2007).	Completo

REFERENCIAS BIBLIOGRÁFICAS

BIBLIOGRAFÍA SUGERIDA

Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. y Zdonik, S. (1992). "The Object-Oriented Database System Manifesto", in *Proceedings of the First International Conference on Deductive and Object-Oriented* Databases, Kyoto, Diciembre.

Bertino, E. y Martino, L. (1995). *Sistemas de bases de datos orientadas a objetos. Conceptos y arquitectura*. Wilmington, Delaware: Addison-Wesley / Díaz de Santos.

Codd. EF. (1970). *Introducción a los sistemas de bases de datos*. (7ª ed.) México: Pearson Education.

Chaudhuri, S., Dayal, U. (1997). "An Overview of Data Warehousing and OLAP Technology". *ACM SIGMOD Record*, no. 26-1.

Chen, Peter P. (1976), "The entity-relationship model - toward a unified view of data". *ACM Transactions on Database Systems*.

Churcher, C. (2007). *Beginning Database Design. From Novice to Professional.* Berkeley, CA: Apress.

Date, C. J. (2001). *Sistemas de Bases de Datos* (7ª ed.) México: Pearson Education.

De Miguel, A., Piattini, M y Marcos, E. (2000). *Diseño de bases de datos relacionales.* Madrid: Alfaomega / Ra-Ma.

Elmasri, R. y Navathe. (2002). SB. (2002). *Fundamentos de sistemas de bases de datos.* México: Pearson Educación / Addison-Wesley.

Geschwinde, E. y Schönig, H. (2002). González, A. (2001). *SQL Server, programación y administración.* Madrid: Alfaomega / Ra-Ma.

Gutiérrez Hernández, G. y Barranco Serrano, V. (2008). *Minería de datos dentro del proceso de KDD aplicado a la base de datos de circulación bibliográfica de la Biblioteca Central*. Tesis de licenciatura [Licenciado en Informática]. México: Facultad de Contaduría y Administración, UNAM.

Hughes, J.G. (1991). *Object-Oriented databases.* Nueva York: Prentice Hall.

Ibáñez Carranza, A. R. (2008). *Data Warehouse. Guía para el diseño de un Data Warehouse*. Tesis de licenciatura [Licenciado en Informática]. México: Facultad de Contaduría y Administración, UNAM.

Inmon, William H. (2002). *Building the data warehouse*. (3rd ed.) Nueva York: John Wiley & Sons.

Johnson, James L. (1997). *Bases de datos*. *Modelos, lenguajes, diseño.* México: Oxford University Press (OUP).

Kimball, R. (2002). *The data warehouse toolkit: the complete guide to dimensional modeling*. (2nd ed.) Nueva York: John Wiley & Sons.

Mallach, E. (2000). *Decision support and data warehouse systems*. Nueva York: McGraw-Hill.

Melton J. y Eisenberg, A. (2004) *SQL y JAVA. Guía para SQLJ, JDBC y tecnologías relacionadas.* México: Alfaomega / Ra-Ma.

Mendelzon, A. (2000). *Sistemas de bases de datos relacionales.* México: Pearson Education.

Mullins, C. (2002). Database administration: the complete guide to practices and procedures. Boston: Addison-Wesley.

Pérez, C. (2004). *Domine Microsoft SQL Server 2000, administración y análisis de bases de datos.* México: Alfaomega / Ra-Ma.

Reyes García, C. T. (2007). *La Minería de Datos como Herramienta para la Toma de Decisiones en el Proceso de Calendarización de Cursos de Cómputo*. Tesis de licenciatura. [Licenciado en Informática]. México: Facultad de Contaduría y Administración, UNAM.

Silberschatz, A; Korth, HF. & Sudarshan, S. (2006). *Fundamentos de bases de datos* (5ª ed.) Madrid: McGraw-Hill.

BIBLIOGRAFÍA BÁSICA

Cano, M. J. (2013). *Inseguridad de la información: una visión estratégica*. Colombia: Alfaomega.

Casas, J. (2013). *Diseño conceptual de bases de datos en UML*. España: Editorial UOC.

Coronel, C. Rob, P., Morris, S., & Romo, J. H. (2011). *Bases de datos: diseño, implementación y administración.* México: Cengage Learning.

Jiménez, M. Y. (2014). *Bases de datos relacionales y modelado de datos. Antequera*, España: IC Editorial.

Joyanes, L. (2013). *Big data: análisis de grandes volúmenes de datos en organizaciones*. México: Alfaomega.

López, I., Castellano, M. J., & Ospino, J. (2013*). Bases de datos.* México: Alfaomega.

Millán, M. E. (2012). *Fundamentos de bases de datos: notas de referencia.* Colombia: Programa Editorial Universidad del Valle.

Mora, A. (2014). *Bases de datos: diseño y gestión.* España: Editorial Síntesis.

Piñero, J. M. (2013). *Bases de datos relacionales y modelado de datos.* España: Paraninfo.

BIBLIOGRAFÍA COMPLEMENTARIA

Grupo Alfaomega. (2014). *Base de datos.* México: Alfaomega.

Hueso, L. (2012). *Bases de datos*. España: Ra-Ma.

Lorenzo, C. (2013). *Bases de datos.* España: Centro de Estudios Financieros UDIMA.

Reinosa, E. Maldonado, C. Muñoz, R. Damiano, L. & Abrutsky. M. (2012). *Bases de datos*. México: Alfaomega.

Silberschatz, A., Korth, H. F., Sudarshan, S., & Moreno, P. (2014). *Fundamentos de bases de datos*. Madrid: McGraw Hill

Spona, H. (2010). *Programación de bases de datos con MySQL y PHP*. México: Alfaomega.

Plan 2012

2016

actualizado