INSTITUTO SUPERIOR
TECNOLÓGICO PELILEO

BASE DE DATOS AVANZADA EN EL NUEVO SIGLO

BASE DE DATOS AVANZADA EN EL NUEVO SIGLO

Directorio editorial institucional
Dr. Rodrigo Mena Mg. Rector
Mg. Sandra Cando Coordinadora Institucional
Mg. Oscar Toapanta Coordinador de I+D+i
Ing. Johanna Iza Líder de Publicaciones

Diseño y diagramación
Mg. Belén Chávez
Mg. Santiago Mayorga

Revisión técnica de pares académicos
Mg. Juan Carlos Pico
IST PELILEO
Correo: jcpico7@gmail.com
Mg. Darwin Fabricio Sánchez
IST PELILEO
Correo: fabricifabrici.05@gmail.com

ISBN: 978-9942-686-51-0
DOI:

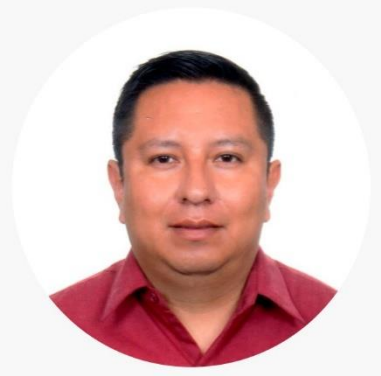
Primera edición
Agosto 2024
https://istp.edu.ec

AUTORES

Ing. Hernán Urquizo, Esp.

DOCENTE

Ingeniero en Sistemas y Computación. Profesional especializado en Base de Datos desarrollo, implementación y mantenimiento de sistemas informáticos y tecnológicos que satisfacen las necesidades de una organización. Su trabajo abarca una amplia gama de actividades relacionadas con la tecnología de la información, conocimiento en áreas como Análisis y diseño de Sistemas, análisis de datos, entre otras. Docente actualmente en el Instituto Superior Tecnológico Pelileo carrera de Desarrollo de Software.



Ing. Fernando Beltrán.

DOCENTE

Ingeniero en Sistemas, ex Docente del Instituto Tecnológico Superior Bolívar, Analista Provincial de Procesos Electorales Consejo Nacional Electoral

Actualmente Docente del Instituto Superior Tecnológico Pelileo.

Ing. Fredy Morales, Mg.

DOCENTE

Ingeniero de Sistemas y Computación en Pontificia Universidad Católica del Ecuador, Magister en Educación Mención en Innovación y Liderazgo educativo por Universidad Tecnológica Indoamericana, Magister en Tecnologías de la Información Mención en Seguridad de Redes y Comunicaciones por universidad Técnica de Ambato. Freddy Morales ha dedicado su carrera a formar futuros profesionales en el campo de la tecnología, combinando una profunda comprensión teórica con una práctica constante en entornos reales. Su experiencia abarca la implementación de proyectos de software utilizando principios de diseño orientado a objetos, así como la aplicación de metodologías ágiles y otras técnicas de programación.

Ing. Fernando Pico Mg.

DOCENTE

Destacado profesional por su capacidad de integrar soluciones tecnológicas en el ámbito empresarial y educativo, ha desarrollado materiales educativos innovadores en redes y seguridad informática. Actualmente, es docente en el Instituto Superior Tecnológico Pelileo, donde imparte materias relacionadas con la Ingeniería de Requerimientos, Integración de Sistemas y Pensamiento Computacional. Su experiencia en el sector privado y en la docencia superior lo consolidan como un experto en la implementación de tecnologías avanzadas y entornos virtuales de aprendizaje, contribuyendo al desarrollo de la próxima generación de profesionales en tecnología.

Ing. Javier Quinde Mg.

DOCENTE

Ingeniero en sistemas e Informática. Profesional especializado en el diseño, desarrollo, implementación y mantenimiento de sistemas informáticos y tecnológicos que satisfacen las necesidades de una organización. Mi trabajo abarca una amplia gama de actividades relacionadas con la tecnología de la información y la gestión de sistemas complejos, conocimiento en áreas como inteligencia artificial, análisis de datos, ciberseguridad, entre otras. Docente actualmente en el Instituto Superior Tecnológico Pelileo carrera de Desarrollo de Software.

Ing. Diego Sanchez Mg.

DOCENTE

Ingeniero en sistemas e Informática. Profesional especializado en el diseño, desarrollo, implementación y mantenimiento de sistemas informáticos y tecnológicos que satisfacen las necesidades de una organización. Mi trabajo abarca una amplia gama de actividades relacionadas con la tecnología de la información y la gestión de sistemas complejos, conocimiento en áreas como inteligencia artificial, análisis de datos, ciberseguridad, entre otras. Docente actualmente en el Instituto Superior Tecnológico Pelileo carrera de Desarrollo de Software.

PRÓLOGO

En el vertiginoso mundo de la tecnología de la información, las bases de datos han emergido como el pilar fundamental que sostiene la gestión y el análisis de grandes volúmenes de información. Desde su concepción, las bases de datos relacionales han revolucionado la manera en que almacenamos, manipulamos y recuperamos datos.

Este paradigma ha demostrado ser robusto y eficiente en una amplia gama de aplicaciones, desde sistemas empresariales hasta plataformas de servicios en la nube.

Este texto está diseñado para guiar a los lectores a través de la evolución y el estado del arte en las bases de datos, explorando tanto los fundamentos de los sistemas relacionales como las innovaciones más recientes en el campo.

A lo largo de estas páginas, los lectores encontrarán un balance entre teoría y práctica, con ejemplos que ilustran cómo los conceptos se aplican en escenarios del mundo real. Esperamos que este texto no solo enriquezca su conocimiento técnico, sino que también inspire nuevas ideas y enfoques en el manejo y explotación de datos.

Bienvenidos a un viaje que abarca desde las bases sólidas de las bases de datos relacionales hasta las alturas de las técnicas avanzadas que están redefiniendo el campo. Este es un recurso indispensable para cualquier profesional, académico o entusiasta que desee entender y dominar el presente y futuro de la gestión de datos.

INSTITUTO SUPERIOR
TECNOLÓGICO PELILEO

TOMO 1:

Base de Datos Relacional

Ing. Hernán Urquizo, Esp.

CONTENIDOS

01

CAPÍTULO UNO
Definición de base de datos.
Objetivos de los sistemas de base de datos.
Abstracción de la información.
Modelo de datos
Manejador de base de datos.
Administrador de base de datos.
Usuarios de las bases de datos.
Estructura general del sistema de base de datos

02

CAPÍTULO DOS
Entidades y conjunto de entidades.
Relaciones y conjunto de relaciones.
Limitantes de mapeo.
Llaves primarias.
Diagrama entidad de relación.
Reducción del diagrama entidad de relación a tablas
Generalización y especialización

03

CAPÍTULO TRES
Estructura de las bases de datos relacionales.
Lenguajes de consulta formales.
Lenguajes de consulta comerciales.
Modificación de la base de datos
Vistas.

04

CAPÍTULO CUATRO
Peligros en el diseño de la base de datos relacionales
Primera y segunda forma normal
Tercera forma normal
Cuarta y quinta forma normal

05

CAPÍTULO CINCO
Conceptos básicos
Diagrama de estructura de datos
El modelo Codossyl DBTG.

06

CAPÍTULO SEIS
Conceptos básicos.
Diagrama de estructura de árbol
Recuperación de datos
Actualización de datos
Registros virtuales

CONCLUSIÒN
BIBLIOGRAFÍA

GENERALIDADES

INTRODUCCIÒN

Hoy en día, la información utilizada en cualquier campo, ya sea empresarial, gubernamental, comercial o educativo, debe automatizarse para evitar la duplicación de información, el acceso lento y la dificultad para encontrarla.

Por lo tanto, el propósito de este curso es brindarles a los estudiantes la oportunidad de comprender los conceptos básicos del diseño de sistemas de bases de datos y aprender lo más posible cómo prevenir problemas que hayan surgido en el pasado, y esta es una de las actividades informáticas en las que Los sistemas de bases de datos son los más utilizados. . . Abrir un campo más amplio.

JUSTIFICACIÒN

Hoy en día, la globalización ha afectado a todas las disciplinas, y la informática no es la excepción, pues el procesamiento de la información debe ser un proceso que simplifique los tiempos y costos de la empresa, por ello P.T. B. En este curso obtendrás una licenciatura en ciencias de la computación y adquirirás las habilidades y destrezas necesarias para comprender cómo diseñar sistemas de bases de datos para minimizar las causas anteriores.

OBJETIVO GENERAL

Utilizar sistemas de gestión de bases de datos para crear bases de datos para el almacenamiento, acceso, recuperación, mantenimiento y preservación de información.

01

INTRODUCCIÒN A LOS CONCEPTOS DE BASE DE DATOS

1.1. Definición de base de datos

Cualquier buen curso debe comenzar con algunos conceptos básicos para comprenderlo mejor, por eso comenzamos con una definición que incluye repositorio.

Dato:

Un conjunto de caracteres que tiene un significado específico pueden ser números, letras o caracteres alfanuméricos.

Información:

Es un conjunto organizado de datos que se gestiona según las necesidades del usuario. Para que un conjunto de datos se procese de manera eficiente y genere informes, primero debe almacenarse de manera lógica en un archivo.

Conceptos básicos de archivos informáticos.

Sitio web:

Es la unidad más pequeña a la que se puede hacer referencia en un programa. Desde el punto de vista de un programador, representa las características de una persona u objeto.

Registro:

Una colección de campos del mismo o de diferentes tipos.

Archivo:

Una colección de registros almacenados en una estructura uniforme.

Base de datos:

Es una colección de archivos interrelacionados creados utilizando un DBMS. El contenido de la base de datos contiene información sobre la organización (almacenada en archivos) para que los usuarios puedan utilizar los datos. Los tres componentes principales de un sistema de base de datos son el hardware, el software DBMS y los datos que se administrarán y las personas responsables de administrar el sistema. Sistema de administración de base de datos.

(Sistema de administración de base de datos)

Un DBMS es una colección de rutinas de software interrelacionadas, cada una de las cuales es responsable de una tarea específica. El objetivo principal de un sistema de gestión de bases de datos es proporcionar un marco que sea práctico y eficiente para recuperar, almacenar y manipular información de una base de datos. Todas las solicitudes de acceso a la base de datos se gestionan centralmente a través del DBMS, por lo que la suite

actúa como una interfaz entre el usuario y la base de datos.

Esquema de base de datos:

Es la estructura que conforma una base de datos y se especifica mediante un conjunto de definiciones expresadas en un lenguaje especial llamado lenguaje de definición de datos. (DDDL)

Administrador de Base de Datos (DBA):

Es un profesional o equipo de profesionales responsables del control y gestión de sistemas de bases de datos, generalmente con experiencia en DBMS, diseño de bases de datos, sistemas operativos, comunicaciones de datos, hardware y programación. El sistema de base de datos está diseñado para procesar grandes cantidades de información. Las operaciones de datos incluyen definir estructuras de almacenamiento de información y proporcionar mecanismos para las operaciones de información. Además, el sistema de base de datos también debe implementar un mecanismo de seguridad para proteger los datos.

El objetivo principal de un sistema de base de datos es proporcionar a los usuarios finales una visión abstracta de los datos, lo que se logra ocultando ciertos detalles de cómo se almacenan y mantienen los datos.

1.2. Objetivos de un sistema de base de datos.

El principal objetivo del sistema de base de datos es reducir los siguientes aspectos:

Duplicación he inconsistencia de datos.

Debido a que los archivos que almacenan información son creados por diferentes tipos de aplicaciones, si el almacenamiento no se controla en detalle, puede ocurrir duplicación de información, es decir, la misma información se utiliza varias veces. Esto aumenta el costo de almacenamiento y acceso a los datos y también puede causar inconsistencia en los datos, es decir, varias copias de los mismos datos se contradicen, por ejemplo: la dirección del cliente está en un archivo y el archivo anterior permanece en otros archivos.

Dificultad para acceder a los datos.

Los sistemas de bases de datos deben considerar un entorno de datos que facilite la gestión de datos a los usuarios.

Considere un banco donde el gerente necesita encontrar los nombres de todos los clientes que viven en la ciudad con

código postal 78733. El gerente le pide al departamento de procesamiento de datos que cree una lista coincidente.

Debido a que el sistema no fue diseñado para anticipar esta situación, no existía ninguna aplicación de consulta que permitiera este tipo de solicitudes, lo que generó un error en el sistema.

Aislamiento de datos.

Dado que los datos se distribuyen en varios archivos y los archivos no pueden estar en diferentes formatos, es difícil escribir nuevas aplicaciones para recuperar los datos relevantes.

Excepción de acceso simultáneo.

Para mejorar el rendimiento general del sistema y lograr tiempos de respuesta más rápidos, muchos sistemas permiten que varios usuarios actualicen datos simultáneamente.

En este entorno, las interacciones entre actualizaciones simultáneas pueden generar datos contradictorios. Para evitar esta posibilidad,
se debe mantener alguna forma de monitoreo en el sistema.

Problema de seguridad.

La información de cualquier empresa es importante, aunque algunos datos son más importantes que otros, se debe considerar el control de acceso, no todos los usuarios pueden ver cierta información, por lo que se deben mantener los datos para que el sistema de base de datos sea confiable.

Autenticación y protección de datos hasta cierto punto.

Por ejemplo, en un banco, los empleados de nómina necesitan ver sólo la parte de la base de datos que contiene información sobre varios empleados del banco y no otro tipo de información.

Cuestiones de integridad.

Los valores de datos almacenados en la base de datos deben cumplir ciertos tipos de restricciones de coherencia. Estas restricciones se implementan en el sistema agregando el código apropiado en varias aplicaciones.

1.3 Abstracción de información.

Una base de datos es esencialmente una colección de documentos relacionados de los cuales los usuarios pueden obtener información independien temente de los límites de los documentos.

Un objetivo importante de un sistema de base de datos es proporcionar a los

usuarios una vista abstracta de los datos, es decir, el sistema oculta cierta información sobre cómo se almacenan y mantienen los datos.

Pero para que el sistema sea manejable, los datos deben adquirirse de manera eficiente.

Existen diferentes niveles de abstracción para simplificar la interacción del usuario con el sistema interna, conceptual y externamente, en particular el almacenamiento del dispositivo, el almacenamiento del usuario y el almacenamiento del programador.

Nivel físico.

Es el nivel más bajo de representación abstracta que describe cómo se almacenan los datos en un dispositivo de almacenamiento (por ejemplo, utilizando punteros o índices para acceder Aleatoriamente a los datos).

Nivel conceptual.

El segundo nivel más alto de abstracción describe los datos reales a lmacenados en la base de datos y las relaciones que existen entre ellos, y describe toda la base de datos en términos de su estructura de diseño.

Este nivel de abstracción conceptual utiliza administradores de bases de datos que deben decidir qué información almacenar en la base de datos.

Consta de las siguientes definiciones:

1. Definición de datos: describe el tipo de datos y la longitud del campo de todos los elementos direccionales de la base de datos. Los elementos definidos incluyen elementos básicos (atributos), agregados de datos y elementos conceptuales (entidades).

2. Relaciones de datos: las relaciones de datos se definen para asociar tipos de registros relacionados para procesar múltiples documentos.

A nivel conceptual, una base de datos aparece como una colección de registros lógicos sin descripciones de almacenamiento. De hecho, el documento conceptual en realidad no existe. La conversión de registros de conceptos a registros de unidades la realiza el sistema y es transparente para los usuarios.

Nivel de visión.

El nivel más alto de abstracción del sistema final visible para los usuarios finales describe sólo una parte de la base de datos para los usuarios que tienen permiso para ver la base de datos. El sistema puede proporcionar múltiples vistas para la misma base de datos.

1.4. Modelo de datos.

Para profundizar en este tema, primero debemos definir qué es un modelo.

Modelo:

Es una representación de la realidad que contiene características generales de lo que se está haciendo. En la base de datos creamos esta representación gráficamente.

¿Qué es un modelo de datos? Es una colección de herramientas conceptuales que describen datos, las relaciones que existen entre datos, la semántica asociada con los datos y las restricciones de coherencia. Los modelos de datos se dividen en tres grupos:

- Modelos lógicos de objetos.
- Un modelo lógico basado en registros.
- Modelo de datos físicos.

Modelo de lógica de objetos.

Se utilizan para describir datos a nivel conceptual y visual, es decir, mediante el uso de un modelo representamos los datos de la misma forma como se perciben en el mundo real. Tienen opciones de estructuración bastante flexibles y permiten una especificación de datos precisa. Existen varios modelos de este tipo, pero el modelo entidad-relación es el más utilizado por su simplicidad y eficiencia.

- **Modelo entidad-relación.**

Debe su nombre a su abreviatura: el modelo E-R representa la realidad a través de entidades, que son objetos que existen y se diferencian de otros objetos por sus propiedades, por ejemplo: un estudiante se caracteriza por sus características específicas (por ejemplo, cuando ingresa a una institución educativa), asignado; nombre o número de control) para distinguirlo de otro estudiante. Las unidades pueden ser de dos tipos:

Activos tangibles:

Son objetos físicos que podemos ver, tocar o sentir.

Activos intangibles:

Todos aquellos eventos u objetos conceptuales que no podemos ver aunque sabemos que existen, ejemplo: una entidad física que sabemos que existe pero que no podemos imaginar ni tocar.

Las características de las entidades en una base de datos se denominan atributos, como nombre, dirección, número de teléfono, grado, grupo, etc. Estos son atributos de entidades

estudiantiles, contraseñas, números de seguro social, departamentos, etc. Una entidad, por otro lado, puede estar relacionada o relacionada con múltiples entidades a través de relaciones.

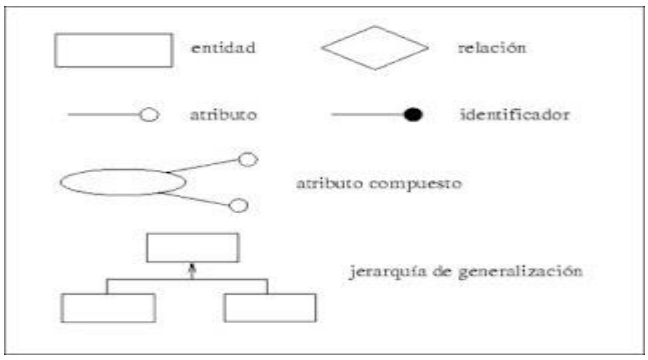
Pero para entenderlo mejor veamos un ejemplo:

Consideremos una empresa que tiene que controlar a los vendedores y sus ventas, a partir de este problema determinamos que los principales objetos o sujetos a estudiar son los empleados (vendedores) y los productos (es decir, bienes para la venta), y sus características identificativas son:

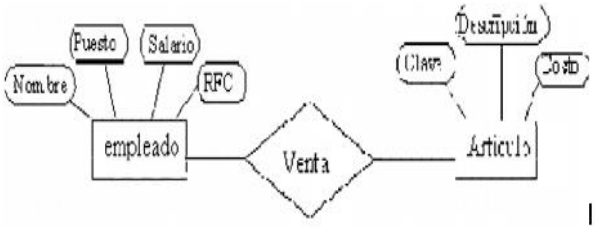
Empleados	**Proyectos**
Nombre	**Descripción**
Costos	**Posición**
Pago	**Clave**
R.F.C.	

Podemos crear una relación entre las dos entidades como ventas. Bien, ahora describiremos cómo se representa gráficamente el modelo E-R. La representación es muy sencilla utilizando los siguientes símbolos:

Símbolo y Representación



Entonces nuestro ejemplo anterior se expresaría como:



Hay muchos más aspectos a considerar sobre el modelo entidad-relación, que se cubrirán en el tema del modelo entidad-relación.

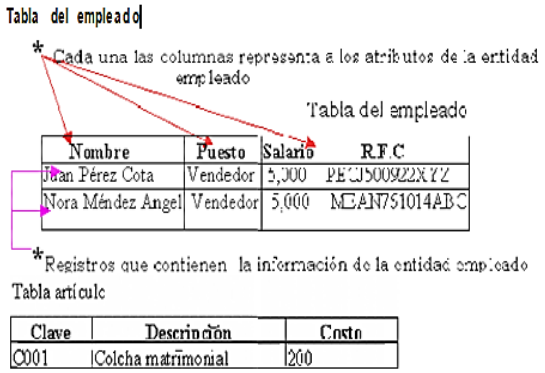
Un modelo lógico basado en registros.

Se utilizan para describir datos a nivel conceptual y físico. Estos modelos utilizan registros e instancias para representar la realidad y las relaciones que existen entre esos registros (vínculos) o indicadores. A diferencia de los modelos de datos de objetos, se utilizan para especificar la estructura lógica general de la base de datos y proporcionar una descripción de implementación de nivel superior. Los tres modelos de datos más aceptados son:

- Modelo relacional
- Modelo de red
- Modelo jerárquico

• Modelo de relación.

En este modelo los datos y sus relaciones se representan a través de una colección de tablas, donde las filas (filas) corresponden a cada registro que contendrá la base de datos, y las columnas corresponden a las características (atributos) de los registros. En las notas se analizan ejemplos de nuestros empleados y proyectos:



Ahora quizás te preguntes, **¿cómo representa este modelo las relaciones entre entidades?**

Hay dos formas de representarlo; pero para ello necesitamos definir cuál es la clave primaria: es la propiedad que definimos como propiedad primaria y es la única forma de identificar la entidad. Por ejemplo, el RFC de un empleado es diferente al RFC de otro empleado porque los RFC no pueden ser iguales. La forma de representar las relaciones en este modelo es la siguiente:

1. Cree una tabla que contenga cada clave principal para las entidades involucradas en la relación.

Considere que la clave primaria de un empleado es su RFC y la clave primaria de un artículo es la clave.

El resultado del modelo:

RFC	Clave
PECJ500922XYZ	C001
MEAN761014ABC	B300

2. Incluido en cualquier tabla de las entidades participantes.

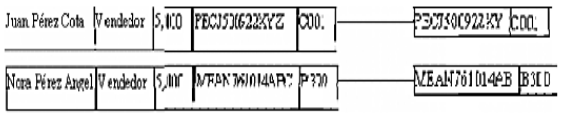
Añadimos la llave primaria en la tabla empleado

Nombre	Puesto	Salario	R.F.C	Clave
Juan Pérez Cota	Vendedor	5,000	PECJ500922XYZ	C001
Nora Méndez Angel	Vendedor	5,000	MEAN761014ABC	B300

Modelo de red.

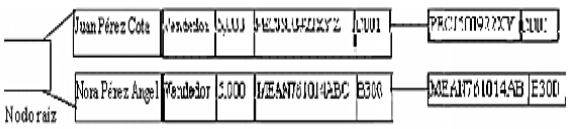
Este modelo representa datos mediante colecciones de registros, y sus relaciones están representadas por alianzas o vínculos que pueden considerarse como indicadores. Estos registros están organizados en un conjunto de gráficos arbitrarios.

Ejemplo:



Modelo jerárquico.

Se asemeja a un modelo de red en términos de relaciones y datos, ya que están representados por registros y sus vínculos. La diferencia es que están organizados mediante árboles en grupos en lugar de gráficos arbitrarios.



Modelo de datos físicos.

Se utilizan para describir el nivel más bajo de datos. Aunque existen pocos modelos de este tipo, cubren prácticamente todos los aspectos de la implementación de un sistema de base de datos. Hay dos clasificaciones de este tipo:

- Memoria de elementos
- Modelo Único

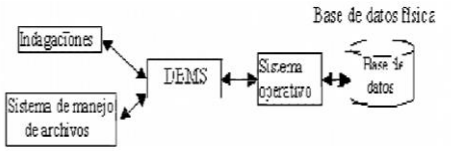
1.5. Administrador de base de datos

Un sistema de gestión de bases de datos es la parte más importante del software del sistema de bases de datos. Un DBMS es una colección de rutinas de software interrelacionadas, cada una de las cuales es responsable de una tarea específica.

Las principales funciones de un DBMS son:

- Crear y organizar bases de datos.
- Crear y mantener rutas de acceso a bases de datos para un acceso rápido a los datos.
- Gestionar los datos según los requerimientos del usuario.
- Uso de base de datos documental.
- **Cooperar con el archivero.** Esto se logra mediante el uso de declaraciones DML para comandos del sistema de archivos. Por lo tanto, el administrador de la base de datos es responsable del almacenamiento de datos real.
- **Copia de seguridad y restaurar.** Dispone de mecanismos para facilitar la colocación de datos en caso de fallo del sistema de base de datos. Permitir la recuperación
- **Control de concurrencia.** Esto incluye controlar las interacciones entre usuarios simultáneos para evitar inconsistencias en los datos.
- **Seguridad e integridad.** Incluye mecanismos que le permiten controlar la coherencia de los datos y protegerlos de cambios planificados o no autorizados. Un

DBMS también se conoce como administrador de bases de datos.



Este diagrama muestra el DBMS como la interfaz entre la base de datos de la entidad y las solicitudes de los usuarios. El DBMS interpreta las solicitudes de entrada/salida del usuario y las envía al sistema operativo para la transferencia de datos entre el dispositivo de almacenamiento secundario y la memoria principal.

El sistema gestor de bases de datos en sí es el núcleo de la base de datos porque es el responsable del control total sobre los posibles aspectos que pueden afectar a la base de datos.

1.6. Administrador de base de datos

Su abreviatura es:

DBA, administrador de base de datos.

Es responsable del sistema de base de datos y tiene control total sobre el sistema de base de datos.

- **Definición del programa.**

  El esquema de base de datos inicial se crea escribiendo un conjunto de definiciones que el compilador DDL traduce en un conjunto de tablas que se almacenan persistentemente en el diccionario de datos.

- **Una estructura de almacenamiento que define los métodos de acceso.**

  Cree estructuras de acceso y almacenamiento adecuadas escribiendo un conjunto de definiciones traducidas por el compilador del lenguaje de definición y almacenamiento de datos.

- **Proporcionar acceso a los datos.**

  Esto permite al administrador de la base de datos regular a qué partes de la base de datos deben acceder varios usuarios.

- **Especificación de restricciones de integridad.**

  Es un conjunto de restricciones almacenadas en una estructura especial del sistema que el administrador de la base de

datos verifica cada vez que se realiza una actualización del sistema.

1.7. Usuarios de bases de datos.

Podemos definir a un usuario como cualquier persona que tenga algún tipo de interacción con un sistema de base de datos desde su diseño, desarrollo, finalización y uso.

Los usuarios que acceden a la base de datos se pueden dividir en:

- **Diseñador de aplicaciones.**
  Los profesionales de la informática interactúan con los sistemas mediante llamadas DML (Lenguaje de manipulación de datos) que se incluyen en programas escritos en lenguajes de programación como COBOL, PL/I, Pascal, C, etc.

- **Usuarios experimentados.**
  Los usuarios experimentados pueden interactuar con el sistema sin necesidad de codificar. En cambio, escriben preguntas en lenguaje
  de consulta de bases de datos.
- **Usuarios profesionales.**
  Algunos usuarios avanzados escriben aplicaciones de bases de datos especializadas que no forman parte de los sistemas de procesamiento de datos tradicionales.
- **Usuario ingenuo.**
  Un usuario no calificado interactúa con el sistema llamando a una de las aplicaciones persistentes previamente registradas en el sistema de base de datos.

  Por ejemplo, podemos referirnos a un usuario ingenuo como un usuario final que utiliza un sistema de base de datos sin comprender su diseño interno.

1.8 Estructura general del sistema.

El sistema de base de datos está dividido en varios módulos, cada módulo controla parte de la responsabilidad general del sistema.

En la mayoría de los casos, el sistema operativo proporciona sólo los servicios más básicos y el sistema de base de datos debe construirse sobre esta base y controlar el

correcto procesamiento de los datos.

Por lo tanto, el diseño del sistema de base de datos debe incluir la interfaz entre el sistema de base de datos y el sistema operativo.

Los componentes de funciones del sistema de base de datos son:

- **Gestión de archivos.**
  Controla la asignación de espacio de memoria en disco y las estructuras de datos utilizadas para representar la información.

  Administrador de base de datos.
  Actúa como una interfaz entre datos y aplicaciones.

  Procesador de consultas.
  Traduzca sugerencias de lenguaje de consulta en instrucciones de bajo nivel.

  También convierte la solicitud del usuario en una forma más eficiente.
- **Compilador DDL.**
  Convierte declaraciones DDL en un conjunto de tablas que contienen metadatos almacenados en un diccionario de datos.

- **Archivo de datos.**
  Es donde se almacenan físicamente los datos de la organización.
- **Diccionario de datos.**
  Contiene información sobre la estructura de la base de datos.
- **Indicador.**
  Proporcionan acceso rápido a registros que contienen valores específicos.
- **Modelo de entidad-relación**
  El modelo E-R se basa en la percepción del mundo real, que consta de objetos básicos llamados entidades, relaciones entre estos objetos y propiedades de estos objetos llamadas atributos.

02

DIAGRAMA DE ENTIDAD RELACIÓN

2.1. Unidades y conjuntos de unidades

Las entidades son objeto que existen y se distinguen de otros objetos en función de sus propiedades, llamadas propiedades. Las entidades pueden ser tan concretas como una persona o tan abstractas como una fecha.

Un conjunto de entidades es un grupo de entidades del mismo tipo. Por ejemplo, el conjunto de entidades CUENTAS puede representar el conjunto de cuentas bancarias X, o el conjunto de entidades ESTUDIANTE puede representar el conjunto de todos los estudiantes de la institución.

Una entidad se caracteriza y se distingue de otra entidad por atributos (a veces llamados características) que representan las características de la entidad.

Los atributos de una entidad pueden adoptar un conjunto de valores permitidos llamado dominio.

Por lo tanto, cada entidad se describe mediante un conjunto de pares de atributos y valores de datos.
Cada atributo del conjunto de entidades tiene un par.

Ejemplo:

Cree algunas descripciones para la entidad Estudiante utilizando los atributos No control, Nombre y Especialidad.

Nombre del atributo	**valor**
Sin cheque	**96310418**
Nombre:	**Sánchez Osuna Ana**
ESP	**LI**

O considere un ejemplo con un vendedor cuyos atributos son: RFC, Nombre, Salario.

Nombre del atributo	**valor**
RFC	**COMD741101YHR**
Nombre:	**Daniel Colin Morales**
Salario	**3000**

2.2. Relaciones y conjuntos de relaciones.

Una relación es una asociación que existe entre dos o más entidades.

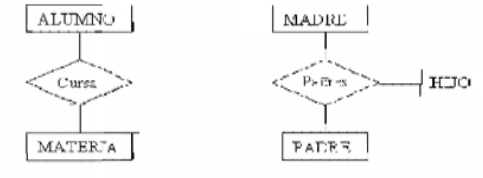
Un conjunto de relaciones es un grupo de un tipo de relación.

El número de entidades de relación determina el grado de la relación, por ejemplo la RELACIÓN ESTUDIANTE-SUJETO es de nivel 2 porque involucra a las entidades ESTUDIANTE y SUJETO, mientras que la relación de APRENDIZAJE puede ser de nivel 3 porque involucra a las entidades PADRE, MADRE e HIJO.

Aunque el modelo E-R permite relaciones de cualquier grado, la mayoría de las aplicaciones del modelo sólo consideran relaciones de grado 2.

La funcionalidad de una relación se denomina rol, o los roles generalmente no se especifican a menos que desee especificar qué significa la relación.

Un diagrama E-R de un modelo de ejemplo (ignora atributos, solo entidades):



2.3. Restricciones de mapeo.

Se pueden crear cuatro tipos de relaciones entre entidades, que determinan cuántas entidades de tipo A se pueden vincular a entidades de tipo B:

Tipo de relación:

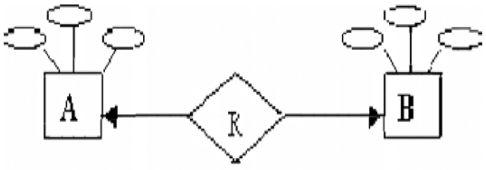
Relación uno a uno.

Como su nombre indica, esto ocurre cuando existe una relación uno a uno, también conocida como relación matrimonial.

Las unidades de tipo A sólo se pueden asociar con unidades de tipo B y viceversa;

Por ejemplo: una relación de asignación de automóvil que involucra las entidades EMPLEADO y AUTO es una relación 1 a 1 porque asocia un empleado con

un automóvil, por lo que a ningún empleado se le puede asignar más de un automóvil y a ningún vehículo se le puede asignar más de uno. . Un coche. Esto se representa gráficamente de la siguiente manera:

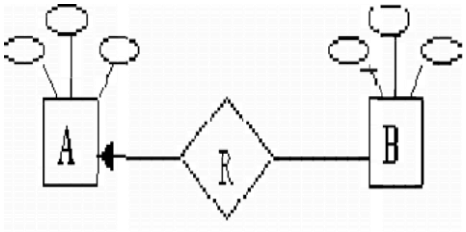


A: representa cualquier tipo de unidades excepto las unidades B.

El extremo discontinuo de la flecha representa una de las relaciones; en este caso, la entidad A está relacionada con la entidad B.

Relación uno a muchos.

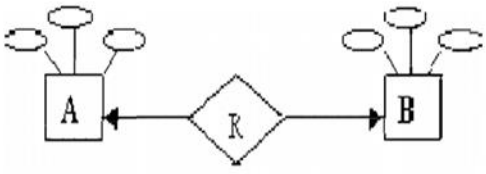
Esto significa que una entidad de tipo A puede asociarse con un número ilimitado de entidades de tipo B, pero una entidad de tipo B puede asociarse solo con una entidad de tipo A.
Su representación gráfica es la siguiente:



Tenga en cuenta que, en este caso, el extremo discontinuo de la flecha para la relación entre A y B significa que la entidad A está relacionada con muchas entidades B. Muchos a uno. Indica que una entidad de tipo B puede asociarse con un número ilimitado de entidades de tipo A, y cada entidad de tipo A puede asociarse solo con una entidad de tipo B.

Muchos a uno.

Indica que una entidad de tipo B puede asociarse con un número ilimitado de entidades de tipo A, y cada entidad de tipo A puede asociarse solo con una entidad de tipo B.



Muchos a muchos.

Dice que cualquier número de unidades de tipo A puede

asociarse con cualquier número de unidades de **tipo** B.

Los tipos de relaciones anteriores también se denominan cardinalidades.

La cardinalidad define los tipos de relaciones que existen entre entidades en el modelo E-R y así crea la validación necesaria para garantizar que el perfil de instancia (un valor en el repositorio en un momento dado) corresponda a la realidad. Algunos ejemplos de gastos de vida normales pueden incluir: Uno después del otro.

Derechos, RFC para cada persona, CURP personal, actas de nacimiento, porque cada uno tiene un solo documento de ese tipo.

Uno a muchos

Cliente - cuenta bancaria, padre - hijo, camión - pasajero, zoológico - animal, árbol - hoja.

Muchos a muchos.

Arquitecto - proyecto, fiestas - personas, estudiantes - tema.

Nota:

Cabe mencionar que la cardinalidad de cada grupo de entidades depende del punto de vista del modelo estudiado, el cual por supuesto está sujeto a la realidad.

Otro tipo de limitación es la presencia de dependencias.

Refiriéndose a las mismas entidades A y B, decimos que si la entidad A depende de la existencia de la entidad B, entonces A depende de la existencia de la entidad B, si eliminamos B, debemos eliminar la entidad A, en este caso es la entidad dominante de B .A es la entidad dependiente.

2.4. Clave primaria.

Como se mencionó anteriormente, un dispositivo se distingue de otro por sus características que lo hacen único.

Un clave primaria es una propiedad que consideramos clave para identificar otras propiedades que describen una entidad.

Por ejemplo, si consideramos la entidad "Estudiante" del Politécnic o de La Paz, podemos tener los siguientes atributos: nombre,

semestre, especialidad, dirección, teléfono, número de control, y entre todos estos atributos podemos especificar como principal.

La clave del atributo es el número de Control, porque es diferente para cada estudiante y nos identifica en la institución.

Por supuesto, puede haber múltiples atributos que puedan identificarse como claves primarias, en cuyo caso se elige el atributo que creemos que es más importante y los demás atributos se denominan claves secundarias.

Las claves o claves primarias en el modelo E-R se representan gráficamente con una línea debajo del nombre del atributo.

2.5. Relación diagrama de entidad

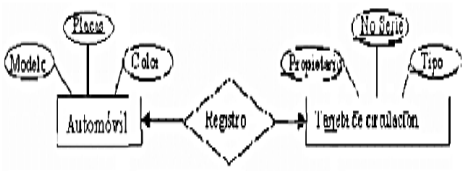
Debe su nombre a sus siglas: E-R representa la realidad a través de un esquema gráfico, utilizando la terminología de entidades, que son objetos que existen y son los principales elementos identificados en la solución de un problema mediante diagramas, y con propiedades especiales llamadas atributos de otros elementos. Los vínculos en un modelo representan relaciones.

Recuerde que el rectángulo representa el atributo de entidad de la entidad, la etiqueta en el diamante representa la relación que existe entre las entidades, su conexión está marcada con una línea y la clave principal de la entidad es el atributo subrayado;

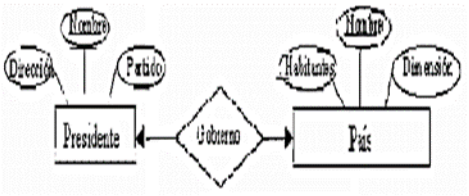
A continuación, mostraremos algunos ejemplos de modelos E-R y consideraremos la cardinalidad que existe entre ellos:

Relación uno a uno:

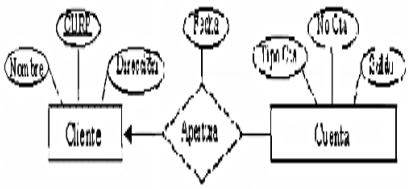
Desarrollar un modelo E-R para una relación de registro de automóvil que implica recibir una tarjeta de registro de automóvil con los siguientes datos: - Auto - Modelo, marca, color - Tarjeta de registro - Propietario, número de serie, tipo.



Usamos este ejemplo para mostrar que existe una relación de membresía individual porque cada vehículo está registrado con una tarjeta de registro.

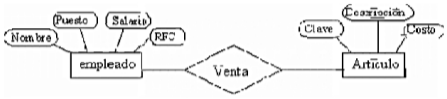


Este ejemplo muestra que un cliente puede tener varias cuentas, pero solo una puede llegar
Propiedad de un cliente (decimos posible porque hay cuentas registradas por varias personas).



2.6. Simplifica el diagrama E-R en una tabla

Los diagramas E-R también se pueden representar mediante una colección de tablas. Cada entidad y relación tiene una tabla única a la que se le asignan nombres de conjunto de entidades y relaciones respectivamente.



Entonces las tablas resultantes siguiendo la descripción anterior son:

Tabla Empleado

Nombre	Puesto	Salario	RFC
Teófilo	Vendedor	2000	TEAT701210XYZ
Cesar	Auxiliar ventas	1200	COV741120ABC

Tabla artículo

Clave	Descripción	Costo
A100	Abanico	460
C260	Colcha matrimonial	1200

Tabla Venta

RFC	Clave
TEAT701210XYZ	C260
COV741120ABC	A100

Tenga en cuenta que en la tabla relacional - Ventas - los atributos contienen las claves primarias de las entidades involucradas en dicha relación.
Si p.e. Al agregar un atributo de fecha a la relación de ventas, la tabla resultante se verá así:

RFC	Clave	Fecha
TEAT701210XYZ	C260	10/12/96
COV741120ABC	A100	11/12/96

2.7. Generalización y especialización.

Generalización.

Es el resultado de combinar 2 o más conjuntos de entidades (de orden inferior) para formar un conjunto de entidades de orden superior. La generalización se utiliza para
resaltar similitudes entre tipos de

entidades de nivel inferior y ocultar sus diferencias.

La generalización implica identificar todos los mismos atributos para un conjunto de entidades para crear un universo de entidades con dichos atributos similares que estarán en un nivel superior al origen de la entidad.

Ejemplo:

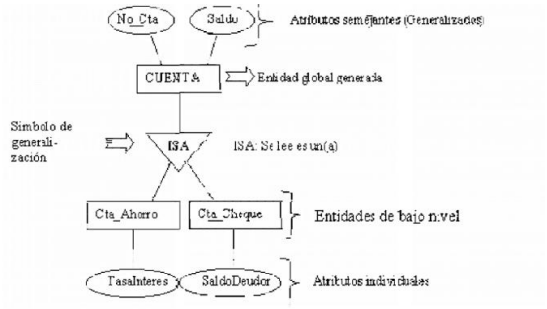
Tomemos como ejemplo el libro "Conceptos básicos de bases de datos" de Henry F.

Dónde:

Hay entidades Account_Savings y Account_Cheques, las cuales tienen propiedades similares no_account y balance,

Aunque además de estas dos propiedades, la Cuenta de Ahorros también tiene una propiedad Tasa de Interés y una propiedad Saldo Deudor para Cuenta_Cheque. De todos estos atributos podemos combinar Account_No y Balance (generalización), que es el mismo en ambas entidades.

Entonces tenemos:



Podemos interpretar este diagrama de la siguiente manera: la entidad Cuenta Ahorró hereda las propiedades No Cuenta y Saldo de la entidad CUENTA,

Además, el atributo de tasa de interés está Dentro del Balance.

Check_account es similar

Propiedades para No_Cta, Saldo deudor.

Como hemos visto, la generalización intenta evitar la duplicación de atributos al incluir atributos similares. La entidad de nivel inferior enumera (hereda) todas las propiedades correspondientes.

Especialización:

Es el resultado de que un subconjunto de entidades de orden superior forme un conjunto de entidades de orden inferior. * En resumen, cualquier entidad de orden superior también debe tener una entidad

de orden inferior. La especialización no tiene esta limitación.

* Está representado por un triángulo llamado "ISA", que se diferencia de la generalización en el grosor de la línea que conecta el triángulo y la entidad.

* La especialización representa la diferencia entre un conjunto de unidades de orden superior y un conjunto de unidades de orden inferior. 2.8.

Compilación.

La agregación surge de una limitación que existe en el modelado E-R, ya que no permite expresar relaciones entre relaciones en el modelo E-R en el caso de que la relación X quiera unirse a una entidad para formar otra relación. La generalización implica el uso de rectángulos para agrupar relaciones (representadas por diamantes) y las entidades y atributos involucrados en ellas para crear un grupo que se considera una entidad y que ahora podemos relacionar con otra entidad.

Para ilustrar lo anterior, miremos a Henry F. Coth. El problema es que hay muchos trabajadores trabajando en diferentes proyectos, pero dependiendo del trabajo que estén realizando pueden utilizar equipos o maquinaria. Este problema involucra 3 entidades: Trabajador, Proyecto y Máquinas, y el diagrama E-R correspondiente es:

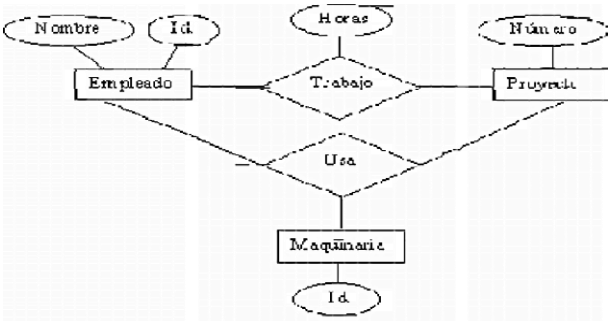


Diagrama E-R con relaciones redundantes

Ahora podemos decir que la entidad de trabajo está relacionada con la entidad mecánica a través de una relación de uso. Indica que se utiliza un tipo específico de equipo o maquinaria para el trabajo, dependiendo del tipo de trabajo involucrado.

Modelo relacional

- 3.1. Creación de una base de datos relacional.
- 3.2. El idioma común oficial.
- 3.3. Lenguaje de consulta empresarial.
- 3.4 Cambio de base de datos.
- 3.5 Pantalla.

Modelo relacional

La ventaja del modelo relacional es que, al menos conceptualmente, los datos se almacenan de la manera más fácil de

entender para los usuarios. Los datos se almacenan como tablas y los datos muestran relaciones entre filas y tablas. Este enfoque permite a los usuarios extraer información de la base de datos sin la ayuda de sistemas profesionales de gestión de información. Las características más importantes del modelo relacional son:

a.) Es importante saber que las entradas de una tabla tienen un solo valor (son valores atómicos no están permitidos, por lo que la intersección de una fila y una columna tiene un solo valor, no un conjunto de valores).

b.) Todas las entradas de una columna son del mismo tipo. Por ejemplo, una columna puede contener nombres de clientes, mientras que otra columna puede contener fechas de nacimiento. Cada columna tiene un nombre único,

Para una tabla cuyas columnas se denominan atributos, el orden de los municipios es irrelevante. Cada atributo tiene un dominio, que es una descripción física y lógica de los valores permitidos.

C.) No hay 2 filas idénticas en la tabla.

d. La información de la base de datos se representa como datos claros sin referencias ni enlaces entre tablas. y. Los métodos relacionales son fundamentalmente diferentes de otros métodos en términos de su estructura lógica y patrón de operaciones de entrada/salida. En el enfoque relacional, los datos se organizan en tablas llamadas relaciones y cada tabla se implementa como un archivo. En términos relacionales, una fila en una relación representa un registro o entidad, cada columna en una relación representa un campo o atributo.

Por tanto, una relación consta de una colección de entidades (o registros) cuyos propietarios se describen mediante una serie de atributos predefinidos implementados como campos.

03

ESTRUCTURA DE BASE DE DATOS RELACIONAL

La arquitectura relacional se puede representar utilizando tres niveles de abstracción: interna, conceptual y visual.

La arquitectura relacional consta de los siguientes elementos:

Modelo de datos relacionales:

A nivel conceptual, un modelo de datos relacional está representado por una colección de relaciones almacenadas. Cada registro de concepto en el modelo de datos relacionales se implementa como un archivo de almacenamiento independiente.

Submodelo de datos:

El esquema externo de un sistema relacional se denomina submodelo de datos relacionales. Cada vista contiene uno o más escenarios (vistas) que describen los datos necesarios para una aplicación particular. Una escena puede contener datos de una o más tablas de datos. Cada aplicación tiene un búfer ("espacio de trabajo del usuario") donde el DBMS puede almacenar datos recuperados de la base de datos para su procesamiento, o donde su salida puede almacenarse temporalmente antes de que el DBMS la escriba en los datos de la base de datos.

Plan de inventario:

Internamente, cada tabla base se implementa como un archivo de almacenamiento. Para la recuperación de claves primarias o secundarias, se pueden crear uno o más índices para acceder a los archivos almacenados.

Sublenguaje de datos:

Es un lenguaje de gestión de datos para sistemas relacionales, álgebra relacional y cálculo relacional. Ambos lenguajes son "relacionales completos", es decir, cualquier relación que pueda derivarse de una o más tablas de datos también puede derivarse de un solo subconjunto. Comando Exportar idioma. Por lo tanto, las operaciones de E/S en sistemas relacionales se pueden procesar una tabla a la vez en lugar de un registro a la vez; en otras palabras, las tablas se pueden recuperar ejecutando comandos de sublenguaje de datos en lugar de registros individuales.

Lenguaje de consulta formal.

Lenguaje de consulta:

Estos son los lenguajes que utilizan los usuarios para solicitar información de una base de datos. Estos lenguajes suelen

estar en un nivel superior a los lenguajes de programación. Los lenguajes de consulta se pueden clasificar en:

Procesales y no procesales;

En un lenguaje de procedimientos, el usuario da instrucciones al sistema para que realice una serie de operaciones en la base de datos para calcular los resultados deseados. En un lenguaje no procesal, los usuarios describen la información que necesitan sin especificar un procedimiento específico para obtener esa información.

El álgebra relacional es un lenguaje de consulta de procedimiento formal que define operadores que operan en tablas (similar a operadores como, -). Álgebra ordinaria) para lograr los resultados deseados. El álgebra relacional es difícil de usar, en parte porque es procedimental, es decir, cuando usamos el álgebra relacional, no sólo necesitamos saber lo que queremos, sino también cómo conseguirlo. El álgebra relacional rara vez se utiliza en el procesamiento de bases de datos empresariales. Aunque algunos productos DBMS exitosos tienen capacidades de álgebra relacional, rara vez se utilizan debido a su complejidad. El álgebra relacional toma dos o más tablas como entrada y crea una nueva tabla mediante una serie de operaciones. Las operaciones básicas del álgebra relacional son selección, proyección, multiplicación cartesiana, cambio de nombre, unión y diferencia de conjuntos. Además de las operaciones básicas, existen otras operaciones como cortar conjunto, producto natural, dividir y asignar, etc.

Operación básica

Las operaciones de selección, diseño y cambio de nombre se denominan operaciones de entidad porque operan en tablas. Otras operaciones operan en pares relacionales y, por lo tanto, se denominan operaciones binarias.

Seleccione una acción.

Esta operación selecciona tuplas (filas) de la tabla que satisfacen la declaración (condición) dada. Está representado entre paréntesis. (nombre de la tabla DONDE condición);

Las expresiones después de la cláusula WHERE pueden contener condiciones de igualdad, por ejemplo

Además, puedes usar los símbolos de conjunción y (^) y o (v) para hacer oraciones más complejas.

Gestión de proyectos.

Esto implica identificar las columnas (atributos del modelo E-R) que nos

interesa comprender. Está representado entre paréntesis. Si se omite, significa que desea que todos los campos estén en la tabla de datos correspondiente. (Nombre de tabla DONDE condición) [Nombre de atributo];

Operación del producto cartesiano. Consiste en multiplicar todas las filas entre las tablas, dando como resultado una tabla que contiene todas las columnas de ambas tablas. Esto está especificado por la directiva TIMES. Nombre de la tabla TIMES nombre de la tabla;

Operación de conexión.

Consiste en el producto (multiplicación) de todas las tuplas de una tabla con todas las tuplas de otra tabla, para luego evaluar las tuplas con el mismo campo común, dando como resultado una nueva tabla que contiene tuplas (filas) que cumplen con lo especificado. criterios) Estado de salud. Esto está representado por la declaración JOIN.

Se coloca una declaración de unión entre dos tablas para multiplicarlas. Primero especifique la operación de selección y proyección (Tabla) [atributo] JOIN (Tabla) [Atributo];

Funcionamiento del departamento.

Tome dos relaciones, una binaria y otra unaria, y cree una relación donde todos los valores de los atributos en la relación binaria (atributo de la otra) coincidan con todos los valores de la relación unaria. Esto está representado por la directiva DIVIDEBY. Nombre de la tabla Bin DIVIDEBY Nombre de la tabla A

Operación diferencial.

Construye una relación que consta de todas las tuplas (filas) de la primera relación que no aparecen en la segunda relación de las dos relaciones especificadas. Está representado por el comando MENOS. Nombre de la tabla A menos nombre de la tabla B;

Alianza operativa.

Construye una relación que consta de todas las tuplas de la primera relación y todas las tuplas de la segunda relación. El requisito es que ambas relaciones sean del mismo tipo. ÚNETE A TABLE_NAMEA EN TABLE_NAMEB

Operación de intersección.

Cree una nueva tabla que consta de todas las tuplas de la primera y segunda tabla. Nombre_tablaA intersección Nombre_tablaB

ejemplo:

Para ilustrar la notación anterior, veamos el ejemplo ESTUDIANTE – curso –

ASIGNATURA, que tiene las siguientes propiedades:

Control	Tecla	Contraseña
Nombre A	Contraseña	Nombre M
Esp.	Nota	Créditos

1.Obtenga los nombres de todos los estudiantes registrados en la institución.

(Estudiante) [Nombre A];

2.- Obtener el nombre del alumno en la base de datos 1 materia, la clave es SCB9333(Estudiante) UNIRSE (Curso donde clave=\'SCB9333\') [NombreA];

3.-Obtener nombres de estudiantes de ingeniería.

Sistema de base de datos temática 2.

JOIN (Cursa) donde profesional = \'ISC\'[Clave, NombreA]) JOIN (Asunto donde NombreM=\'BD2\')[NombreA];

En álgebra relacional, necesitamos saber no solo lo que queremos, sino también cómo obtenerlo; al ejecutar una consulta, debemos especificar el nombre de la tabla a usar si queremos realizar una operación que tenga propiedades que otras tablas.

no.es, necesitamos "extraer" este atributo para usarlo, como en la situación anterior, necesitamos usar el nombre del estudiante, que solo está disponible en la tabla de estudiantes, pero también esperamos satisfacer la condición MName=BD2, p.ej.

no se puede vincular directamente a estas dos tablas, usamos la tabla del curso de la cual obtenemos la clave de la materia y almacenamos el nombre del estudiante (Nombre A) Finalmente, usando el comando JOIN, unimos las tablas según los campos comunes que tienen.

clave para que obtengamos una tabla con los nombres de los estudiantes de ISC.

La tabla de temas de la cual seleccionamos solo el tema llamado BD2 usando el comando Join, de donde finalmente proyectamos el atributo NameA que tenemos y así obtuvimos esta nueva tabla.

Lenguaje de consulta empresarial

Business Query Language proporciona una interfaz más fácil de usar. Un ejemplo

de este tipo de lenguaje es SQL (lenguaje de consulta estructurado).

Las partes principales de SQL son:

DDL: Lenguaje de definición de datos (nos permite crear estructuras).

DML: Lenguaje de manipulación de datos (nos da acceso a estructuras para borrar, modificar e insertar)

En esta sección, veremos los métodos básicos para ejecutar consultas usando SQL, mientras que

3.4. En la sección:

Modificación de bases de datos veremos cosas relacionadas con la modificación de tablas.

La estructura básica de una expresión SQL consta de 3 partes:

Select, from y Where.

Se utiliza una expresión de selección para especificar los atributos requeridos en los resultados de la consulta. A partir de la expresión de la deuda, las relaciones a comprobar en la valoración. Donde está la definición de las condiciones seguidas por la consulta.

Una consulta SQL típica tiene la siguiente forma:

- **Select:** A1, A2, A3...Un
- **From:** r1,r2,r3...rm
- **Where:(**Condiciones**).**

Dónde:

- ✓ **A1, A2, A3...An:** Representan cada atributo o campo en una tabla de base de datos relacional.
- ✓ **R1,r2,r3...rm:** Representan las tablas involucradas en la consulta.
- ✓ **Condición:** Declaraciones que cont rolan los resultados de una consulta. Sí se omite la cláusula Where, la condición se considera verdadera y la lista de atributos (A1, A2..An) se puede reemplazar con un asterisco (*) para seleccionar todos los atributos de todas las tablas que aparecen en la cláusula From.

Operaciones SQL.

SQL crea un producto cartesiano de la tabla contenida en la cláusula From, coincide con las condiciones de la directiva Where y luego usa la directiva select para proyectar el resultado.

En nuestro ejemplo, consideramos una tabla llamada CURSO con los siguientes campos:

Nombre del campo	Descripción
NumC	Número del curso, único para identificar cada curso
NombreC	Nombre del curso, también es único
DescC	Descripción del curso
Creditos	Créditos, número de estos que gana al estudiante al cursarlo
Costo	Costo del curso.
Depto	Departamento académico que ofrece el curso.

Información contenida en la tabla CURSO

NumC	NombreC	DescC	Creditos	Costo	Depto
A01	Liderazgo	Para público General	10	100.00	Admón.
S01	Introducción a la inteligencia artificial	Para ISC y LI	10	90.00	Sistemas.
C01	Construcción de torres	Para IC y Arquitectura	8	0.00	Ciencias
B01	Situación actual y perspectivas de la alimentación y la nutrición	Para IB	8	80.00	Bioquímica
E01	Historia presente y futuro de la energía solar	IE e II	10	100.00	Electromecánica.
S02	Tecnología OLAP	Para ISC y LI	8	100.00	Sistemas
C02	Tecnología del concreto y de las Estructuras	Para IC	10	100.00	Ciencias
B02	Metabolismo de lípidos en el camarón	Para IB	10	0.00	Bioquímica
E02	Los sistemas eléctricos de potencia	Para IE	10	100.00	Electromecánica
S03	Estructura de datos	Para ISC y LI	8	0.00	Sistemas
A01	Diseño bioclimático	Para Arquitectura	10	0.00	Arquitectura
C03	Matemáticas discretas	General	8	0.00	Ciencias
S04	Circuitos digitales	Para ISC	10	0.00	Sistemas
S05	Arquitectura de Computadoras	Para ISC	10	50.00	Sistemas
I01	Base de Datos Relacionales	Para ISC y LI	10	150.00	Informática

Consulta de ejemplo:

Obtención de una tabla completa

• Obtén toda la información disponible sobre cursos 0 costo.

- **SELECT:**
- **FROM CURSO:**
- **Tarifa de cursos = 0.00**

Resultados de la última consulta.

NumC	NombreC	DescC	Creditos	Costo	Depto
C01	Construcción de torres	Para IC y Arquitectura	8	0.00	Ciencias
B02	Metabolismo de lípidos en el camarón	Para IB	10	0.00	Bioquímica
S03	Estructura de datos	Para ISC y LI	8	0.00	Sistemas
A01	Diseño bioclimático	Para Arquitectura	10	0.00	Arquitectura
C03	Matemáticas discretas	General	8	0.00	Ciencias

Insertamos * porque no limitan la información en la tabla, es decir, requieren que mostremos toda la información de los atributos en la tabla CURSO.

Dado que la única condición en la cláusula WHERE es que el costo del curso sea igual a 0, la consulta devuelve todas las tuplas donde se encuentra costo = 0,00. Dado que el precio es un campo numérico, la condición sólo se puede comparar con campos del mismo tipo. Se agrega un nuevo prefijo de símbolo (-) a la izquierda para representar valores negativos.

- Ø menor que <
- Ø mayor que >
- Ø Menor o igual a <=
- Ø Mayor o igual a >=
- Ø Varios<>
- Ø Además de los operadores booleanos AND, NOT y OR.

Tenga en cuenta que en la cláusula Where, cuando utiliza cadenas para definir condiciones, esas condiciones están separadas por apóstrofes ('). Las expresiones de cadena se comparan carácter por carácter; dos cadenas son iguales sólo si todos los caracteres coinciden. Consulta de ejemplo con una cadena:

• Obtener toda la información sobre cualquier curso ofrecido por el Departamento de Ciencias.

- **Select:**
- **From Curso:**
- **Where Depto:**'Ciencias'

Resultado

NumC	NombreC	DescC	Creditos	Costo	Depto
C01	Construcción de torres	Para IC y Arquitectura	8	0.00	Ciencias
C02	Tecnología del concreto y de las Estructuras	Para IC	10	100.00	Ciencias
S04	Circuitos digitales	Para ISC	10	0.00	Sistemas

Muestra la columna especificada.

En el ejemplo anterior recuperamos la tabla completa, ahora veremos cómo mostrar solo algunas propiedades específicas de la tabla.

• Obtener los valores de CNum, CNname y Dept secuencialmente de todo el programa de estudios. En el curso, seleccione CNum, CNname,

- **Select:** CNum, CNname y Dept
- **From Curso:**

Resultado Obtenido:

NumC	NombreC	Depto
A01	Liderazgo	Admón.
S01	Introducción a la inteligencia artificial	Sistemas.
C01	Construcción de torres	Ciencias
B01	Situación actual y perspectivas de la alimentación y la nutrición	Bioquímica
E01	Historia presente y futuro de la energía solar	Electromecánica.
S02	Tecnología OLAP	Sistemas
C02	Tecnología del concreto y de las Estructuras	Ciencias
B02	Metabolismo de lípidos en el camarón	Bioquímica
E02	Los sistemas eléctricos de potencia	Electromecánica
S03	Estructura de datos	Sistemas
A01	Diseño bioclimático	Arquitectura
C03	Matemáticas discretas	Ciencias
S04	Circuitos digitales	Sistemas
S05	Arquitectura de Computadoras	Sistemas
I01	Base de Datos Relacionales	Informática

Si una columna de la primera tabla coincide con un valor de la segunda tabla, la tabla resultante contendrá una fila para cada valor coincidente de las dos tablas originales.

Para ilustrar esto con un ejemplo, veamos 2 tablas: Tabla1 y Tabla2, luego: Notamos que en este caso no hay cláusula Where, ni condición, por lo que se restauran todas las filas de la tabla CURSO, pero solo las tres filas.

Ver subconjunto de filas y columnas.

•Seleccione los valores de NumC, Departamento y Costo para todos los cursos que cuesten menos de $100

SELECT:

NumC, Departamento, Costo	FROM CURSO	WHERE Costo<100.00

El resultado de esta consulta es que se recuperarán todas las tuplas de CTARIFA con un costo inferior a 100 y solo se mostrarán los campos NumC, Dept, Cost.

Podemos ver que este ejemplo cubre el formato general de consultas SQL.

palabra clave DIFERENTE

DISTINCT es una palabra reservada que se utiliza para evitar filas duplicadas en los resultados de la consulta.

• Ver todos los departamentos académicos que ofrecen cursos y rechazar valores duplicados

Select Distinct Depto	**From Curso:**

Resultado

Depto
Administración
Sistemas
Ciencias
Bioquímica
electromecánica
Arquitectura
Informática

DISTINCT viene directamente después de la palabra SELECT.

Si no se utiliza la palabra DISTINCT, el resultado serán tuplas de todos los atributos de departamento encontrados, es decir columna Departamento se mostrará en su totalidad.

Utilización de los conectores booleanos (AND, OR, NOT)

Para utilizar múltiples condiciones en la cláusula WHERE,
utilizamos combinaciones lógicas.

El Conector **AND**

Este conector le pedirá al sistema que seleccione una columna solo si se cumplen dos condiciones.

• Obtener toda la información de todos los cursos ofrecidos por el

departamento de
sistemas que tienen una tarifa igual a 0.

- SELECT
- FROM CURSO
- WHERE departamento = "sistema" y precio = 0,00;

El resultado de esta consulta serán todas las tuplas que coincidan exactamente con las dos coinciden condiciones dadas.

El conector **OR.**

Al igual que An esta Unión permite combinar múltiples condiciones en una cláusula WHERE.

- Obtener toda la información disponible sobre todos los cursos Ofrecidos por el departamento de arquitectura o el departamento de bioquímica.

Elige uno de los cursos *

Select*,**from**,curso,**where,**Depto='Arquitectura' **OR** Depto='Bioquímica'

Esta consulta devolverá todas las tuplas. Que coincidan con una de las condiciones, es decir, mostrará todas las tuplas que tengan el atributo departamento=arquitectura o bioquímica.

Conector NOT

Esto nos permite marcar aquellas tuplas que no queremos mirar por algún motivo.

- Obtener nombres de cursos y departamentos para todos los Cursos fuera del Departamento de sistemas

Desde Cursos, seleccione CNumber, Departamento

- ✓ SELECT NombreC, Depto
- ✓ From curso
- ✓ WHERE NOT (Depto=Sistemas);

Jerarquía de operadores booleanos.

Ordenar en orden descendente (de mayor a menor prioridad)

NOT,AND,OR.

Hay dos maneras de realizar consultas, Join de Querys y Subquerys.

Si separamos los nombres de las tablas con comas en la cláusula from significa que estamos ejecutando la consulta en. Forma de consulta. En este caso necesitamos tener el nombre de los atributos. Antes del nombre de la tabla y el punto. En una combinación de consultas el resultado de las tablas involucradas. En la consulta es una combinación de tablas. Donde los valores son los siguientes:

C1	C2	C3		CA	CB
A	AAA	10		35	R
B	BBB	45		10	S
C	CCC	55		65	T
D	DDD	20		20	U
E	EEE	20		90	V
F	FFF	90		90	W
G	GGG	15		75	X
H	HHH	90		90	Y
				35	Z

Cuando las consultas están anidadas, se denominan subconsultas o subconsultas. Este tipo de consulta proporciona resultados parciales, lo que reduce el espacio necesario para ejecutar la consulta.

Nota. Todas las consultas analizadas mediante subconsultas se pueden analizar mediante consultas de combinación, pero no todas las consultas analizadas mediante consultas de combinación se pueden analizar mediante subconsultas.

Para ilustrar lo anterior, consideremos el siguiente ejemplo.

ALUMNO - cursa - MATERIA, que tienen los siguientes atributos:

NControl NControl Clave
NombreA Clave NombreM

Especialidad Calif Creditos

Dirección

Representando en tablas a los atributos quedarían de la siguiente forma:

Tabla alumno:

NControl	NombreA	Especialidad	Dirección

Tabla cursa:

NControl	Clave	Calif

Tabla materia:

Clave	NombreM	Creditos

Obtenga el nombre de la materia que está cursando el estudiante, el número de control es 97310211 y los puntos son iguales a 8. De las materias

Select: NombreA

- ✓ WHERE créditos='8' and clave in (SELECT:clave)
- ✓ From: Cursa
- ✓ WHERE NControl='97310211';

Obtener el control del estudiante que tenga un aporte =100

- ✓ SELECT DISTINC(NCONTROL)
- ✓ FFROM Cursa
- ✓ WHERE Calif='100'

• Obtener el nombre de la materia estudiada por el estudiante Salvador Chávez.

- ✓ Select NameM
- ✓ From Materia
- ✓ Where Clave in (SELECT DISTINC (clave)
- ✓ From Cursa
- ✓ WHERE NControl in (SELECT NControl)
- ✓ From estudiante
- ✓ WHERE NombreA='Salvador
- ✓ Chávez'));

Funciones adicionales para consultas

Algunas funciones pueden acelerar las consultas, como las hojas de cálculo, porque funcionan en filas y columnas.

- **COUNT ():** Cuenta el número de filas en una columna de colección
- **MIN ():** Encuentra el valor mínimo de la columna creada
- **MAX ():** Coloca el valor máximo de la columna creada. AVG (): obtiene el valor promedio de la columna creada
- SUMA (): Obtiene la suma de los valores resultantes de los valores del campo creado.

Ejemplo:

✓ Número de estudiantes de la carrera de ingeniería en sistemas informáticos.

Select Count (*) **From** estudiantes, **Where** ESP = 'ISC';

• Recibió la calificación más alta jamás obtenida por J.M. cadena.

Select Max (Limitado)

From Curso

Where NControl in (seleccione NControl **From** Estudiante Where NombreA = 'J.M. cadena');

• Obtuve el promedio de Salvador Chávez.

Select Avg (Calif From curso **WHERE** NControl in (select NControl **From estudiantes WHERE** Nombre A='Salvador Chávez');

• Obtener el número total de resultados obtenidos por Daniel Collin.

Select cantidad (Calif) from curso

WHERE NControl in (seleccione NControl **FROM** estudiante **WHERE** NombreA='Daniel Colin');

Hasta ahora hemos visto el proceso simple de ejecutar consultas usando SQL. Es importante señalar que al realizar consultas anidadas, debemos tener en cuenta la precedencia de los operadores y también de los paréntesis.

3.4 Cambios en la base de datos

Como se mencionó al principio de esta sección de SQL, contiene un módulo DDL para definir datos, que nos permite crear o cambiar la estructura de las tablas de datos.

Las instrucciones para realizar estos pasos son:

- **CREATE TABLE:** Le permite crear una tabla de datos vacía.
- **INSERT:** Le permite almacenar registros en la tabla creada.

- **UPDATE:** Le permite cambiar los datos de registro almacenados en la tabla.
- **DELETE:** Eliminar un registro completo o un grupo de registros de una tabla.
- **CREATE ÍNDEX:** Crea un índice para ayudarnos a ejecutar consultas.
- **DROP TABLE:** Le permite eliminar una mesa.
- **DROP INDEX:** elimina el índice especificado.

Para ilustrar la explicación anterior, veamos el ejemplo ESTUDIANTE – curso – ASIGNATURA, el cual tiene las siguientes propiedades:

NControl	CLAVE	NControl
Nombre A	Contraseña	Nombre M
Esp	Calif	Credito

* **La estructura de la declaración CREATE TABLE. Crear tabla de tabla>**

(

Atributo 1: tipo de datos de longitud
Atributo 2: tipo de datos de longitud
Atributo 3: tipo de datos de longitud

Atributos: longitud del tipo de datos,

PRIMARY KEY (opcional);

Opcionalmente, los campos se pueden definir como NOT NULL a menos que se requiera NOT NULL en la clave principal. Además, al definir una clave primaria se generará automáticamente el índice del campo clave para definir la clave, lo expresamos entre paréntesis PRIMARY KEY.

* Estructura de sentencia CREATE INDEX

CREAR ÍNDICE: El nombre que se le dará al índice.

ON El nombre de la tabla a indexar (la columna a través de la cual se indexará);

* **Estructura de la declaración *UPDATE**

UPDATE: El nombre de la tabla cuyos datos se cambiarán.

SET: valor

WHERE (Estado-condición);

Ejemplo:

Cambie el número de control del registro de Daniel Collin en la tabla de estudiantes a 96310518.

UPDATE Estudiante

SET NControl '96310518'

WHERE NomA='Daniel Colín';

* **Estructura de declaraciones DROP TABLE**

DROP TABLE Nombre de la tabla que se va a eliminar;

Ejemplo:

Elimina la tabla de estudiantes creada anteriormente.

DROP TABLE Estudiante;

* **Estructura de la declaración DROP INDEX**

DROP INDEX Nombre del índice que se eliminará;

Ejemplo:

Elimine el índice Index1 creado anteriormente.

DROP INDEX índice 1;

* Estructura DELETE

DELETE

FROM Nom de la table

WHERE Condición;

Ejemplo:

- Eliminar registro con número de control 95310386.

FROM estudiantes

WHERE control='95310386';

- Eliminar todas las entradas en la tabla de estudiantes.

DELETE

FROM estudiantes;

El primer ejemplo eliminará todo el registro (todos los datos) del estudiante con número de control = 95310386. Debido a la eliminación.

3.5 VISTA.

Las vistas se definen en SQL utilizando la directiva CREATE VIEW. Para definir una vista, debemos darle un nombre a la vista y declarar una consulta que calcule la vista. Una vez creada la vista, podemos ejecutar una instrucción SELECT que haga referencia a la vista. El sistema asocia la vista SQL con la tabla base y luego recupera y muestra los datos de la tabla base.

Esto significa que la vista no tiene datos duplicados de la tabla base. No tiene absolutamente ningún dato porque no es una tabla real, todo el proceso se realiza utilizando los datos almacenados en la tabla base. Es decir, se considera una mesa virtual.

Los comandos utilizados para manipular las vistas son:

CREAR VISTA: Crea una tabla virtual. ELIMINAR VISTA: Elimina una vista creada previamente.

Estructura de la declaración CREATE VIEW. CREATE VIEW nombre-vista AS (operador de consulta);

En nuestro ejemplo, veamos nuevamente la tabla denominada CURSO, que contiene los siguientes campos:

Nombre del campo	Descripción
NumC	Número del curso, único para identificar cada curso
NombreC	Nombre del curso, también es único
DescC	Descripción del curso
Creditos	Créditos, número de estos que gana al estudiante al cursarlo
Costo	Costo del curso.
Depto	Departamento académico que ofrece el curso.

Create View Cursos **AS**

SELECT NOMBRE, Cnum, Puntos, Costo, CDesc FROM curso WHERE DESCC='sistema';

Notemos que después del nombre de la vista ponemos una declaración AS que define la estructura de la vista, y la estructura de la vista en sí está formada por las consultas que vimos anteriormente usando el comando SELECT.

VER LA VISTA

Creamos una tabla virtual que contiene los datos que queremos consultar, ahora necesitamos visualizar estos datos, para ello usamos SELECT y ejecutamos la consulta:

SELECT *

FROM cursos costosos;

A partir de esta consulta, podemos ver que se muestran todos los campos de la vista y, aunque solo podemos ver algunos de ellos, también podemos ver que estamos reemplazando el nombre de la vista con el nombre de la tabla junto a la cláusula FROM, que Esto se debe a que la vista es una tabla virtual, pero contiene datos como cualquier tabla normal.

Eliminar vista

Al igual que una tabla normal, una vista también se puede eliminar, para ello utilizamos la instrucción DROP VIEW. Estructura de la declaración DROP VIEW.

DROP VIEW Nombre de la vista A eliminar;

Ejemplo: Eliminar la vista CursosCaros creada previamente.

DROP VIEW CursosCaros;

Diseño de bases de datos relacionales

4.1. Amenazas del diseño de bases de datos relacionales.

4.2 Primera y segunda forma normal.
4.3. Tercera forma normal y forma normal de Boyce-Cood.

4.4. Cuarta y quinta forma normal.

04

DISEÑO DE BASES DE DATOS RELACIONALES.

Amenazas del diseño de bases de datos relacionales.

Uno de los desafíos del diseño de bases de datos es lograr una estructura lógica sólida para:

1. No habrá excepciones de almacenamiento en el sistema de base de datos.

2. El modelo lógico se puede modificar fácilmente para adaptarse a nuevos requisitos.

Incluso en un entorno dinámico, las bibliotecas implementadas en modelos bien diseñados tienen una vida útil más larga que las bibliotecas mal diseñadas. En promedio, cada seis años la base de datos sufre una importante reestructuración en función de las necesidades de los usuarios. Un almacén bien diseñado funcionará bien incluso a medida que crezca en tamaño y sea lo suficientemente flexible como para incorporar nuevos requisitos o funcionalidades adicionales.

El diseño de bases de datos relacionales tiene varios riesgos que afectan su funcionalidad. Estos riesgos suelen ser la duplicación de información y la inconsistencia de los datos. La normalización es el proceso de simplificar las relaciones entre campos de un registro. La normalización reemplaza un único conjunto de datos en un solo registro con múltiples registros que son más simples, más predecibles y, por lo tanto, más fáciles de administrar.

Hay cuatro razones para la estandarización:

- Estructura los datos para que puedas representar las relaciones entre los datos.
- Capacidad para recuperar fácilmente materiales en respuesta a preguntas y solicitudes de informes.
- Simplifique el mantenimiento de datos con actualizaciones, inserciones y eliminaciones.
- Reduzca la necesidad de reestructurar o reorganizar datos a medida que surgen nuevas aplicaciones.

En resumen, la normalización intenta simplificar el diseño de la base de datos al encontrar la mejor estructura que pueda usarse con las entidades involucradas en la base de datos.

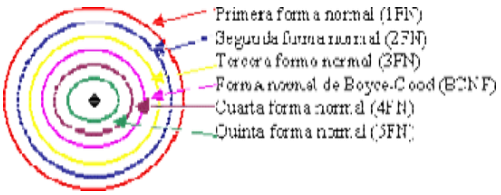
Pasos de estandarización:

1. Divida todos los grupos de datos en registros bidimensionales.
2. Elimine todas las relaciones que no dependan completamente de la clave principal del registro.
3. Elimine cualquier relación que tenga dependencias transitivas.

La teoría de la normalización se basa en el concepto de forma normativa, se dice que una relación está en un paradigma determinado si se ajusta a un conjunto de restricciones.

Primera y segunda forma normal. Forma normal.

Estos son métodos para manejar excepciones en tablas. Dependiendo de su estructura, una tabla puede estar en primera forma normal, segunda forma normal o cualquier otra forma. Vínculo entre paradigmas:



✓ **Primera forma normal.**

Definición formal:

Una relación R existe en 1FN si y sólo si cada fila de columna contiene un valor atómico. Abreviada como 1NF, se dice que una relación está en primera forma normal si se cumplen las siguientes condiciones:

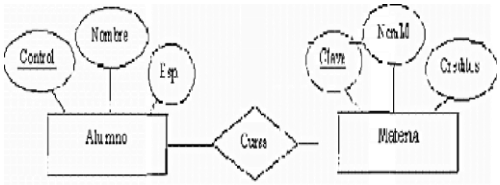
1. Las celdas de la tabla tienen valores simples y no se permiten valores como grupos o arreglos repetidos. cada celda contiene un valor.

2. Todas las entradas en cualquier campo (atributo) deben ser del mismo tipo.

3. Cada columna debe tener un nombre único, no importa el orden de las columnas en la tabla.

4. No pueden dos filas o filas de una misma tabla ser idénticas, aunque da igual el orden de las filas.

En general, la mayoría de las relaciones satisfacen estas propiedades, por lo que podemos decir que la mayoría de las relaciones están en primera forma normal. Para dar un ejemplo de cómo se puede representar gráficamente una relación en el primer paradigma, considere una relación entre un estudiante y un curso cuyo diagrama E-R se ve así:

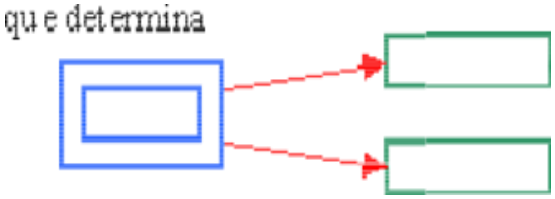


Como esta relación trata de valores atómicos, es decir, de un valor para

cada campo que compone los atributos de la entidad, ya está en primera forma normal, que es, gráficamente hablando, como representamos las relaciones en 1NF.

✓ La segunda forma normal.

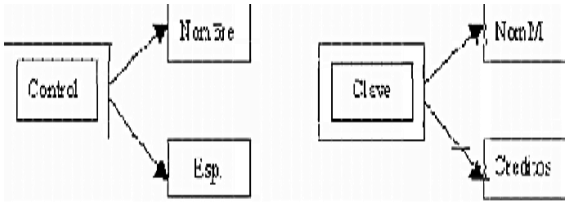
Para definir formalmente la segunda forma normal, necesitamos saber qué es la dependencia funcional: implica especificar qué propiedades dependen de otras propiedades.



Definición formal:

Una relación R es 2NF si y sólo si es 1NF y los atributos no primarios dependen funcionalmente de la clave primaria.

Una relación está en segunda forma normal si sigue las reglas de la primera forma normal y todos sus atributos no clave (claves) dependen completamente de la clave. Según esta definición, cada tabla con un atributo único como clave está en segunda forma normal. La segunda forma normal está representada por dependencias funcionales como:



Tenga en cuenta que la clave principal está representada por un cuadro doble y las flechas indican que estas propiedades pueden hacer referencia a otras propiedades que dependen funcionalmente de la clave principal.

Tercera forma normal y forma normal del Código Boyce.

Para definir formalmente 3NF, necesitamos definir dependencias transitivas: una relación (en la Tabla 2D) con al menos 3 atributos (A, B, C) donde A determina B y B determina C pero no A.

✓ Tercera forma normal.

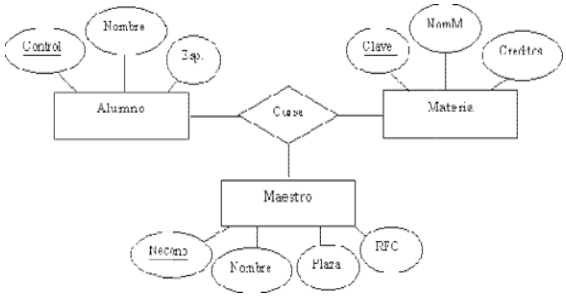
Definición formal:

Una relación R es 3NF si y sólo si es 2NF y todos sus atributos no primarios dependen invariantemente de la clave primaria.

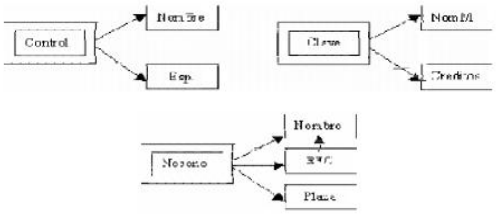
Esto incluye eliminar dependencias transitivas almacenadas en otra forma normal. En resumen, una relación está en tercera forma normal si está en segunda forma normal y no hay

dependencias transitivas entre las relaciones.

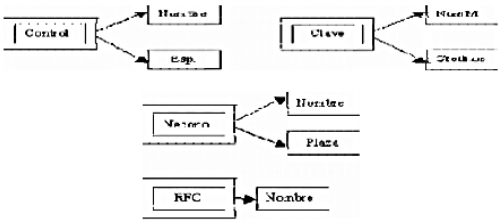
Propiedades Hablamos de dependencias transitivas cuando hay más de una forma de llegar a una referencia a una propiedad relacional. Por ejemplo, considere la siguiente situación:



Anteriormente nos fijamos en la relación entre alumno, curso y materia, pero ahora nos fijamos en el elemento principal, que podemos representar gráficamente de la siguiente manera:



Podemos ver que está dibujado en segunda forma normal, es decir, todas las propiedades de la clave se muestran en un cuadro doble, indicando las propiedades que dependen de la clave, pero en la clave Necono tiene 3 atributos de dependencia, donde puedes referirte al nombre con dos atributos: Necono y RFC (existe una dependencia transitiva), este problema lo podemos solucionar aplicando la tercera forma normal, que consiste en eliminar estas dependencias separando los atributos, y luego tenemos:



Forma normal de Boyce Corder.

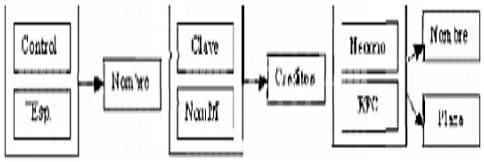
Determinante: Uno o más atributos que determinan funcionalmente otro atributo o atributos. En la dependencia funcional (A, B)-->C, (A,B) es el factor decisivo.

Definición formal:

Una relación R pertenece al CBNF si y sólo si cada determinante es una clave candidata. Las tablas BCNF de esta forma, denominadas así por la abreviatura en inglés, se consideran si y sólo si cada determinante o propiedad es una clave candidata.

Siguiendo con el ejemplo anterior, si creemos que el nombre del atributo control y entidad estudiante puede

hacer referencia a atributos específicos, entonces decimos que esos atributos pueden ser claves candidatas. Podemos representar gráficamente la forma normal de Boyce Codd de la siguiente manera:



Tenga en cuenta que, a diferencia de la tercera forma normal, agrupamos todas las claves candidatas para formar una clave candidata global (representada en un cuadro) que se refiere a atributos de clave no candidatas.

El cuarto y quinto paradigma

✓ Cuarta forma normal.

Definición formal:

Un esquema de relación R está en formato 4NF e incluye un conjunto de funciones y una dependencia multivalor D. Sí, para todas las dependencias multivalor de la forma X->->Y D, donde X<=R e Y<=R, al menos uno DEBEN cumplirse una de las siguientes condiciones:

* X->->Y es una dependencia simple de valores múltiples.

*X es la superclave del esquema R.

Para entender esto mejor, consideremos una afinidad (tabla) llamada Estudiante, que contiene las siguientes propiedades: clave, especialidad, curso, como se muestra en la siguiente figura:

Clave	Especialidad	Curso
S01	Sistemas	Natación
S01	Bioquímica	Danza
S01	Sistemas	Natación
B01	Bioquímica	Guitarra
C03	Civil	Natación

Aceptamos que los estudiantes puedan inscribirse en múltiples especialidades y múltiples cursos. Un estudiante con código S01 estudia sistemas y bioquímica y estudia cursos de natación y danza con especialización en bioquímica y un estudiante con código C03 estudia ingeniería y estudia cursos de natación;

No hay dependencias funcionales en esta tabla o relación porque los estudiantes pueden tener diferentes especialidades y un valor clave puede tener múltiples valores principales, así como valores del curso. Entonces hay una dependencia de múltiples valores.

Esta dependencia dará como resultado la duplicación de datos, como se muestra en la tabla anterior, donde la clave S01 tiene tres entradas para mantener series de datos de forma independiente, lo que genera demasiadas operaciones durante la actualización.

Una dependencia multivaluada existe cuando una afinidad tiene al menos tres propiedades, dos de las propiedades tienen múltiples valores y sus valores dependen solo de la tercera propiedad, en otras palabras, en la afinidad R (A,B),C) Si A determina varios valores de B, y A determina múltiples valores de C, y B y C son independientes entre sí, existe una dependencia multivaluada.

En la tabla anterior, la clave define múltiples valores para especialidades y la clave define múltiples valores para cursos, pero las especialidades y los cursos son independientes entre sí.

Las dependencias múltiples se definen como Clave->->Principal y Clave->->Curso. Se lee "clave principal de determinación múltiple, clave de curso de determinación múltiple";

Se deben resolver múltiples dependencias de valores para evitar la duplicación de datos. Esto se logra creando dos tablas, cada una de las cuales almacena datos para solo uno de los atributos multivalor. Las tablas correspondientes a nuestro ejemplo son:

Tabla principal Tabla ECours

Tabla Especialidad

Clave	Especialidad
S01	Sistemas
B01	Bioquímica
C03	Civil

Tabla ECurso

Clave	Curso
S01	Natación
S01	Danza
B01	Guitarra
C03	Natación

✓ Quinta forma normal.

Definición formal:

Un esquema de relación R es 5NF con respecto a un conjunto de dependencias de características, valores múltiples y productos D si al menos una de las siguientes condiciones es verdadera para todas las dependencias de productos en D:

*(R1, R2, R3, ... Rn) es una dependencia de producto simple.

* Todas las Ri son súper claves para R.

La quinta forma normal se refiere a dependencias no relacionadas. Esto está relacionado con tablas que se pueden dividir en subtablas pero que no se pueden reconstruir.

05

DIAGRAMA DE ESTRUCTURA DE DATOS

Conceptos básicos.

Como sugiere el nombre, una base de datos en red consta de una colección de registros conectados entre sí mediante enlaces. Este registro se parece a una entidad, como la utilizada en el modelo entidad-relación.

Un registro es una colección de campos (atributos), cada campo contiene solo un valor almacenado y un enlace es una asociación exclusiva de dos registros, por lo que podemos considerarlo como una relación estrictamente binaria. Las estructuras de datos de red (a veces denominadas estructuras compuestas) son más extensas que las estructuras de árbol porque los nodos secundarios de una estructura de red pueden tener varios padres. En otras palabras, la restricción de que cada hijo pueda tener sólo un padre en un árbol jerárquico se vuelve menos estricta. Por lo tanto, una estructura de árbol puede considerarse un caso especial de una estructura de red, como se muestra en la siguiente figura.

Para ilustrar la estructura de los registros en una base de datos web, veamos la base de datos de materias de los estudiantes. Una entrada en Pascal se ve así:

type estudiantes= record

NomA: string [30]; Control: string [8]; Especialidad: string [3];

end;

type materia = record

Clave: string [7]; NomM: string [25]; Cred=string [2];

end;

Diagrama de estructura de datos.

Un diagrama de estructura de datos es un diagrama esquemático que muestra el diseño de una base de datos de red. Este modelo se basa en la representación de registros mediante enlaces, que son relaciones que involucran sólo dos entidades (binarias) y relaciones que involucran más de dos entidades (genéricas), con o sin atributos descriptivos.

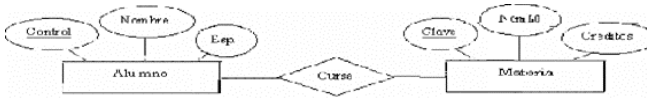
Un formulario de diseño consta de dos componentes básicos:

- **Celda:** representa un campo de registro.

Línea: Representa una conexión entre registros. Un diagrama de estructura

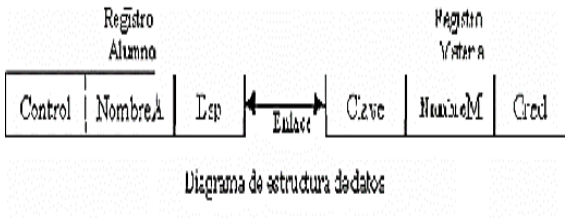
de datos de red muestra la estructura lógica general de una base de datos, una representación gráfica de la misma basada en la disposición de los campos de un registro en un conjunto de celdas que están relacionadas con otro registro. Ilustraremos esto con un ejemplo:

Veamos la relación estudiante, curso y materia, donde la relación del curso no tiene propiedades descriptivas:

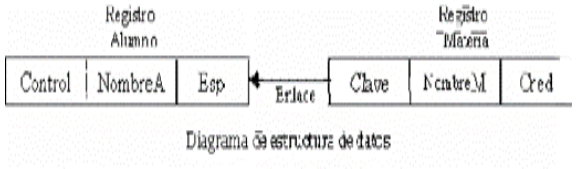


✓ Las estructuras de datos basadas en cardinalidad se representan en las siguientes situaciones: Cuando los enlaces no tienen propiedades descriptivas

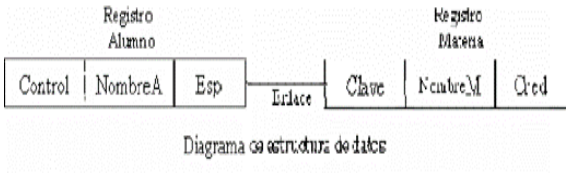
Caso 1. Cardinalidad uno a uno.



Caso 2. Cardinalidad muchos a uno.

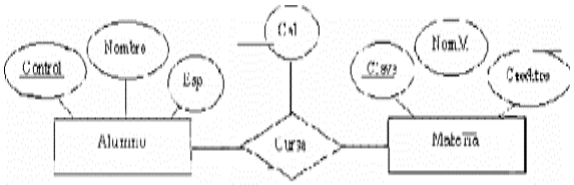


Caso 3. Cardinalidad de muchos a muchos.



Si el enlace tiene propiedades descriptivas.

Digamos que agregamos un atributo Cal (calificación) a la relación actual y nuestro modelo E-R se verá así:



El método para convertir a un diagrama de estructura de datos es el siguiente:

1. La representación de los campos de registro se realiza agrupando los campos de registro en las celdas correspondientes.

2. En este ejemplo, cree un nuevo registro llamado Calif usando un solo campo cal (calif).

3. Cree un enlace especificando la siguiente base:

- AluCal, de registro de California a registro de Estudiante.
- MatCal, desde la entrada de California hasta la entrada de Materia.

AluCal y MatCal son solo nombres que usamos para identificar enlaces, pueden ser otros nombres y no se usan para ningún otro propósito.

✓ **Transformación de grafos de estructura de datos basada en:**

Caso 1. Cardinalidad uno a uno.

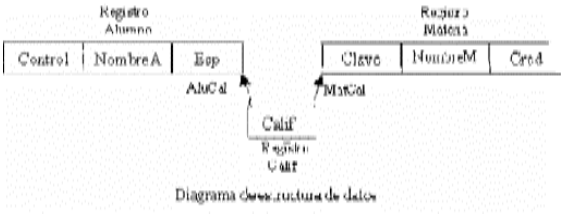


Diagrama de estructura de datos

Caso 2. Cardinalidad uno a muchos.

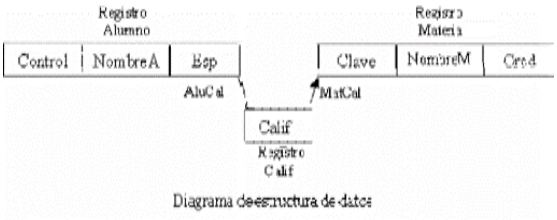


Diagrama de estructura de datos

Caso 3. Cardinalidad de muchos a muchos.

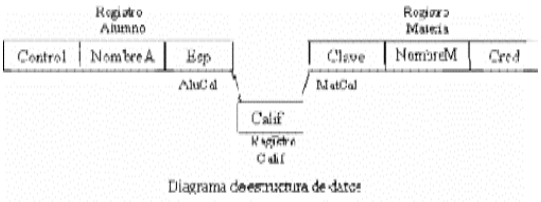
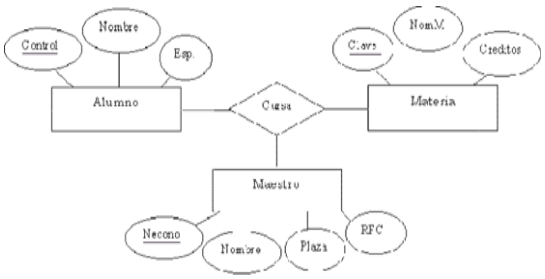


Diagrama de estructura de datos

Diagrama de estructura de datos cuando intervienen más de dos entidades y el enlace no tiene atributos descriptivos.

Supongamos que hemos agregado una entidad profesora, la entidad que imparte dicha materia, a la relación estudiante, curso y materia.

Nuestro diagrama E-R se ve así:



La conversión del diagrama de estructura de datos se completa realizando los siguientes pasos:

1. Crear registros apropiados, para cada unidad modelo de intervención.

2. Crea un nuevo tipo de publicación que llamamos enlace. Puede que no contenga ningún campo o que solo contenga un campo que contenga un identificador único. Este identificador será proporcionado por el sistema y no será utilizado directamente por la aplicación. Ficción o conexión o asociación.

Después de los pasos anteriores, nuestra estructura final es: (dada una relación de cardinalidad uno a uno)

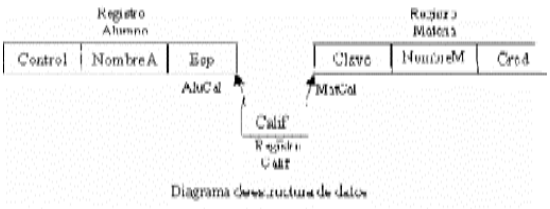


Diagrama de estructura de datos

Ahora, si nuestro enlace tiene un atributo descriptivo, se creará un registro con los campos adecuados y un tipo de enlace indicando la cardinalidad adecuada. En este ejemplo, tomamos el ejemplo anterior de cardinalidad uno a uno y agregamos las propiedades correspondientes a la relación. (calificación).

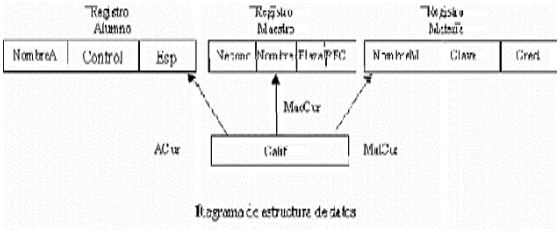
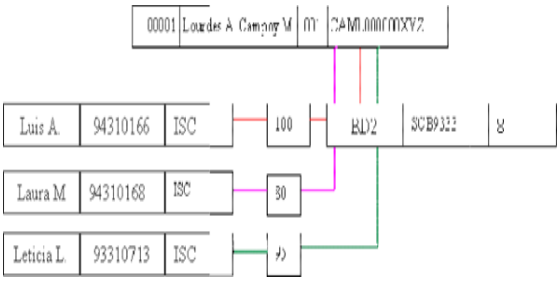


Diagrama de estructura de datos

Dado el diagrama de estructura de datos anterior, el siguiente es un ejemplo:



La foto muestra que el estudiante Luis A. Laura M. Leticia L. aprendió el tema de la segunda base de datos junto con el Ing. Lourdes A. Campoy M obtuvo 100, 80 y 95 respectivamente.

Este modelo fue desarrollado en 1971 por el grupo CODASYL: CODASYL: Conference on Data Systems Languages, Database Task Group, de ahí que el grupo fuera nombrado desarrollador del estándar COBOL, y el modelo CODASYL ha seguido evolucionando a lo largo de los años, y las transacciones. Han surgido productos DBMS orientados, pero hoy en día estos productos se están volviendo obsoletos porque el modelo es muy complejo y poco confiable. Los diseñadores y programadores deben tener mucho cuidado al desarrollar bases de datos y aplicaciones DBTG. Recién desarrollado.

En este capítulo, aprenderá sobre el modelo que subyace al desarrollo de muchos productos DBMS orientados a transacciones.

5.3 Modelo CODASYL DBTG.

En el modelo DBTG sólo se pueden utilizar conexiones uno a uno y uno a muchos. Este modelo tiene dos elementos principales: propietarios y miembros, donde sólo puede existir un propietario, y miembros múltiples, donde cada miembro depende de un solo propietario.

Ejemplos de uso de relaciones estudiante-curso-materia.

Si la relación es de uno a muchos y no tiene atributos descriptivos, los datos relevantes son:

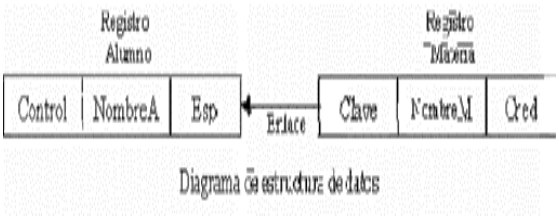
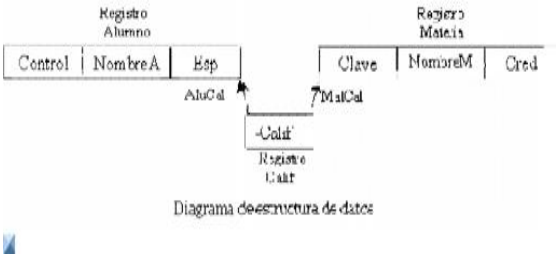


Diagrama de estructura

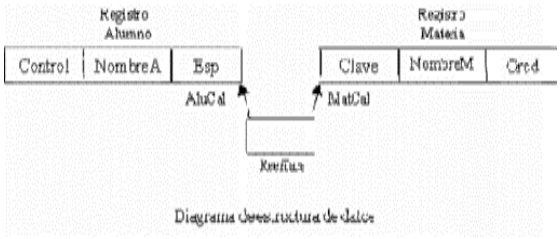


-. Si la relación es de muchos a muchos, dado que la relación no tiene propiedades descriptivas, el algoritmo de conversión se verá así:

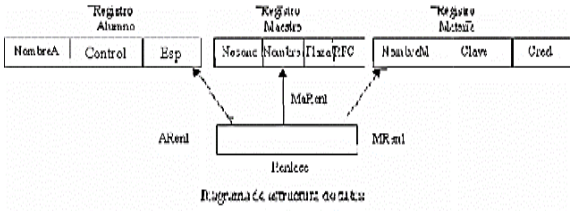
1. Crear registros adecuados que involucren a los sujetos (estudiantes, sujetos).

2. Cree un nuevo tipo de registro virtual y enlace. Este tipo de registro puede no contener ningún campo o puede contener solo un campo que contenga un identificador único definido externamente.

3. Cree vínculos entre pares de varios a uno.

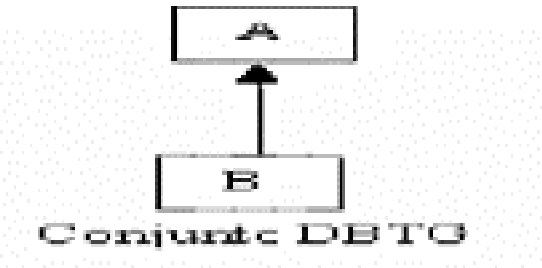


Para relaciones generales (es decir, no binarias), el algoritmo de transformación es el mismo que se utiliza para crear gráficos de modelos de red que involucran más de 2 entidades. Por ejemplo, considere la agregación de la unidad principal, por lo que en este caso se obtiene el siguiente resultado estructural:



Paquete DBTG

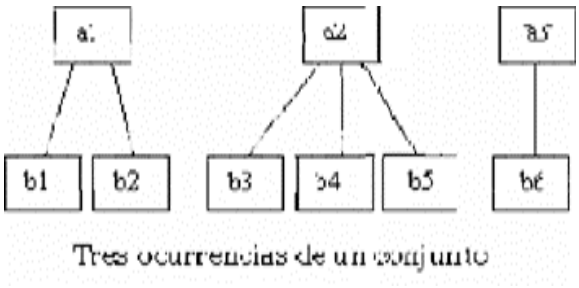
Como se mencionó anteriormente, en este patrón solo se pueden usar enlaces muchos a uno y uno a uno, por lo que la forma general del patrón es la siguiente:



En el modelo DBTG, esta estructura se denomina conjunto DBTG. Este nombre suele ser el mismo que el nombre de la relación de la entidad federada.

En cada conjunto DBTG de este tipo, el tipo de registro A se denomina propietario (o padre) del conjunto, y el tipo de registro B se denomina miembro (o descendiente) del conjunto. Se muestra el conjunto. Dado que no se permiten enlaces de muchos a muchos, cada instancia de colección tiene solo un propietario y cero o más registros de miembros. Además, ninguna inscripción podrá participar en más de una actuación grupal a la vez. Sin embargo, un registro de miembro puede participar en varias instancias de diferentes colecciones al mismo tiempo.

Podemos dar ejemplos de lo que puede pasar, como, por ejemplo:



Tres ocurrencias de un conjunto

Para ilustrar esto, considere el siguiente diagrama de estructura:

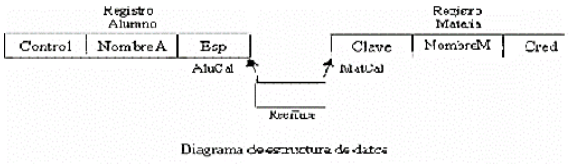


Diagrama de estructura de datos

Hay dos colecciones DBTG:

* AluCal, propiedad de Alumno y cuyos miembros son enlaces.
* MatCal, el propietario es Materia y el miembro es relink.

Declaración de conjuntos

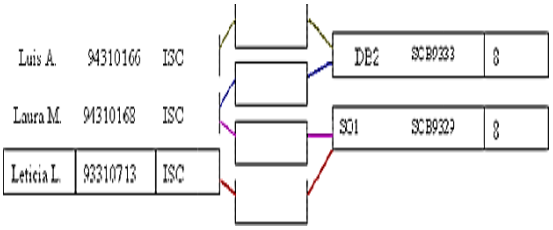
Definición de **conjunto AluCal**:

Set name is Alucal **owner is** Estudiante **nember is** reenlace

Definición de **conjunto AsigCal**:

Set **name is** Alucal **owner is** Asignatura **nember is** reenlace

La unidad de ejecución de la base de datos puede ser:



Recuperación de datos en DBTG

El lenguaje de procesamiento de datos de la propuesta DBTG consta de una serie de comandos entrelazados en el lenguaje principal.

La mayoría de los comandos de manipulación de datos en CODASYL DBTG tienen dos operaciones. Primero, emita el comando FIND para identificar el registro que se va a operar. El comando FIND no lee, procesa el registro especificado, solo identifica un registro que el DBMS puede encontrar. Una vez identificado el registro, el segundo

Se pueden emitir comandos DML para operar con ellos. Buscar y modificar o buscar y eliminar artículos.

Generalmente FIND, MODIFY; o FIEN ERASE.

Cuando el programa emite el comando FIND, se encuentra el registro y su identificador se almacena en una variable especial llamada indicador de ubicación. Después de emitir una instrucción GET, MODIFIC, ERASE u otra instrucción, el DBMS mira el indicador de posición para decidir en qué registro operar. Los indicadores de posición también se utilizan como puntos de referencia para comandos de procesamiento sucesivos como FIND NEXT o FIND PRIOR.

Los siguientes pasos se utilizan para la recuperación de datos:

FIND: busque una entrada en la base de datos y asigne el indicador actual correspondiente.

GET: copie el registro al que apunta la unidad de ejecución de la base de datos actual en la plantilla adecuada en el espacio de trabajo del programa.

El comando FIND tiene diferentes formas, en este ejemplo veremos 2 comandos de búsqueda diferentes para buscar registros individuales en la base de datos, la secuencia más simple es la siguiente:

Find any <tipo de registro> **using** <campo de registro>

Este comando busca registros de tipo de registro> cuyo valor de de registro> sea el mismo que el valor de de registro> en la plantilla del espacio de

trabajo de la aplicación. Cuando se encuentra un registro, los siguientes indicadores cambian para indicar el registro:

- puntero a la unidad de ejecución actual
- Un puntero al tipo de registro actual que coincide con
- Para cada colección en la que participa este registro, la métrica de publicación de colección correspondiente.

Para ilustrar lo anterior, creemos una consulta en DBTG que muestre el número de cheque del estudiante Louis A (considere el modelo estudiante-sujeto).

Nombre. Estudiante: ="Luis A.";

Find any <Estudiante> **using** <NomA>

Get estudiantes:

print ("estudiante. Control");

Pueden existir varias entradas con el valor especificado. El comando Buscar encontrará el primero de ellos en el orden anterior. Encuentre otros registros en la base de datos que puedan tener los mismos campos. Usamos el siguiente orden:

Find any <tipo de registro> **using** <campo de registro>

Encuentra los siguientes registros con valores de (según el orden del sistema):

Estudiante. NombreA: ="Luis A.";

FIND ANY estudiante **USING** NombreA;

WHILE DB Status= 0 do

BEGIN

GET estudiantes;

print (estudiante. Control);

FIND DUPLICATE Estudiante **USING** NomA;

END;

Parte de la consulta está contenida en un bucle porque no sabemos de antemano cuántos nombres podría tener Luis A.. Salimos del ciclo cuando DB-Status<>0, lo que indica que la última operación de búsqueda de duplicados falló, lo que significa que hemos agotado todos los registros que contienen este perfil.

Actualización en DBTG.

Establece un nuevo registro.

El orden en que se crean los nuevos registros es el siguiente:

Store<Tipo de registro>

Esta tecnología nos permite crear y agregar solo un registro a la vez.

Para ilustrar, considere la siguiente situación:

Queremos inscribir a una nueva estudiante llamada "Delia J. Siordia" y las instrucciones son:

Student.NameA: ="Delia J. Siordia".

Control. Estudiante: ="94310128"; Estudiante.Esp: ="LAE";

Store Estudiante;

Editar registros existentes.

Para modificar un registro, primero buscamos el registro en la base de datos, lo transferimos a la memoria principal y realizamos los cambios necesarios, luego actualizamos el puntero actual al registro que se va a modificar y ejecutamos el comando de modificación.

Editar <Tipo de registro>

Utilice la directiva de (UPDATE) actualización para informar al sistema que es necesario realizar un cambio.

Ejemplo: Queremos cambiar la especialización de una estudiante llamada Delia J. Siordia, en Lee.

Estudiante. NombreA="Delia J. Horda;

FIND FOR UPDATE ANY estudiante **USING** nombre A;

GET estudiantes. Esp :="LI";

MODIFY estudiante;

En las líneas de código anteriores, primero necesitamos datos para buscar un registro, cualquier comando de búsqueda para actualización crea un tipo de registro que se utilizará, el uso indica que se deben buscar cambios en los campos correspondientes del elemento, colóquese delante de la cola de modificación que se realizará en el registro y, finalmente, debido a que el registro fue modificado, se ejecuta la instrucción de modificación para iniciar la actualización realizada.

Registro eliminado.

Para eliminar una entrada, el indicador actual debe apuntar a la entrada que queremos eliminar y luego ejecutar en el siguiente orden: **ERASE** < Tipo de registro>.

Esta declaración es la misma que la declaración de modificación y la declaración de actualización debe declararse en la declaración de búsqueda. Ejemplo. Digamos que queremos eliminar un alumno con

número de control 94310166 perteneciente al alumno Luis A.

Fin: =Falso;

Estudiante. NombreA: ="Luis A.";

find any Estudiante using NomA; while DB-Estado=0 and not fin do begin

get Estudiante;

if Estudiante.control=94310166 then begin

erase Estudiante; fin:=true;

end else

find for update next estudiante with Alren1(*Registro reenlace*)

end;

Se puede eliminar una secuencia de ensamblaje completa buscando a su propietario; para ello, ejecute el siguiente comando:

Eliminar todo

Esto eliminará al propietario de la colección y a todos sus miembros, y si el miembro de la colección posee otra colección, los miembros de esa colección también serán eliminados. Ejemplo. Digamos que queremos eliminar a un estudiante llamado Louis A. y todas tus publicaciones temáticas, entonces:

Estudiante. NombreA: ="Luis A.";

Find for update any Estudiante using NomA

Erase all estudiante;

Procesamiento de cobros en DBTG.

* Notificación de conexión (connect)

Esta declaración le permite insertar registros en la colección.

La sintaxis es:

Connect <tipo de registro> tp <tipo de conjto >

La inserción de nuevos registros se realiza creando un nuevo registro y buscando el propietario de la colección, donde el cursor existente apuntará al lugar apropiado para insertar el nuevo registro, luego se inserta el registro ejecutando la instrucción Connect.

Ejemplo. Considere una consulta DBTG para el modelo de materia Estudiante para crear un nuevo registro de estudiante en la materia BD1.

Estudiantes. Control: =97310128; Estudiante. NombreA: =Armando Sánchez; Estudiante.Esp: =ISC;

ShopStudent; Topic.Key:=SCB9333;

Buscar cualquier tema por clave;

Conecte al estudiante a AlRen1 (*Revinculación del registro de conexión*);

* **Sentencia de desconexión Disconect**

Esta declaración se utiliza para "liberar" un registro de una colección. El registro actual y el índice de la colección deben apuntar al elemento correspondiente. Luego elimine las entradas necesarias usando la instrucción Desconectar.

Sintaxis:

Disconnect<tipo de regidstro> FROM <Tipo de conjunto>

Esta acción sólo detiene el registro y no lo elimina de la base de datos.

sintaxis:

Erase <registro>

*** Vuelva a agregar la notificación de reconexión. (RECONECCIÒN)**

Esta función le permite mover un registro de un grupo a un grupo dorado, lo que requiere encontrar el registro y el propietario correspondiente.

Sintaxis:

Reconnect <registro> to <Conjunto>

*** Inserción y retención de conjuntos**

Cuando definimos una nueva colección, debemos especificar cómo insertar los registros de miembros. Además, aclarar las condiciones bajo las cuales se debe llevar la contabilidad.

*** Poner en colección.**

Los registros de tipo de registro de tipo colección recién creados se pueden agregar manual o automáticamente a los eventos de colección. Especifique el esquema cuando defina una colección usando el siguiente comando:

Insertion is <modo de inserciòn>

En el modo de inserción, puede utilizar los siguientes comandos para obtener los valores "manual" y "automático", respectivamente: agregar de registro> tp de configuración> y guardar. En ambos casos, el puntero actual de la colección debe apuntar a la colección en la que se insertará.

- **Aplicación:** Sólo se permite la reincorporación de registros pertenecientes a miembros de colecciones del mismo tipo.

06

MODELO DE DATOS JERÁRQUICOS

Conceptos básicos.

Una base de datos jerárquica consta de una colección de registros conectados entre sí mediante enlaces. Estas entradas son similares a las que se ven en el modelo de red.

Cada registro es una colección de campos (propiedades), y cada campo contiene un valor. Una unión es la única asociación o unión entre dos registros. Por tanto, el concepto es similar al modelo de red de conexión. Echemos otro vistazo a la base de datos que contiene las relaciones estudiante-sujeto del sistema escolar.

Hay dos tipos de registros en este sistema: estudiantes y materias. Los registros de estudiantes constan de tres campos: NombreA, Control y Esp. Los registros de materias constan de tres campos: Contraseña, MName y Cred.

En este tipo de modelo, la organización se construye en forma de árbol, donde las raíces son nodos ficticios. Entonces sabemos que una base de datos jerárquica es una colección de árboles de este tipo.

El contenido de un registro particular puede repetirse en varios lugares (en un árbol o en varios árboles). El registro duplicado tiene dos desventajas principales:

*Pueden producirse inconsistencias en los datos.

* Perdida de espacio.

Diagrama de estructura de árbol

Un diagrama de árbol es una representación de un esquema de base de datos jerárquico, de ahí el nombre, ya que un árbol se desarrolla en una estructura jerárquica en orden descendente preciso. Este tipo de gráfico consta de dos componentes básicos:

• Rectángulo: representa un registro.

• Línea: Representa un vínculo o conexión entre registros.

Un diagrama de árbol pretende indicar la estructura general de una base de datos.

Un diagrama de estructura de árbol es similar a un diagrama de estructura de datos en un modelo de red. La principal diferencia es que, en el modelo de red, los registros se organizan en forma de gráfico arbitrario, mientras que, en el modelo estructurado en árbol, los registros se

organizan en forma de árbol con raíces.

Características de la estructura de madera:

• Un árbol no puede contener ciclos.

• Las relaciones dentro de la estructura deben ser relaciones de muchos a uno o de uno a uno entre padre e hijo únicamente.

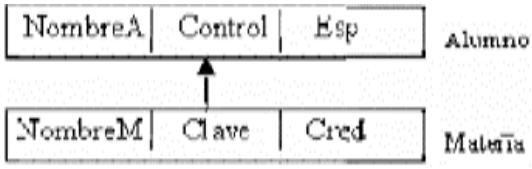
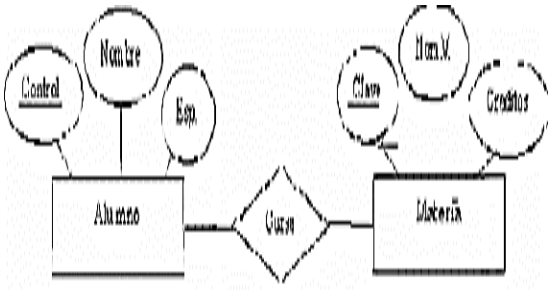


diagrama de estructura de árbol

En esta imagen vemos una flecha que apunta de padre a hijo. Un padre (origen de rama) puede tener una flecha apuntando a un hijo, pero un hijo siempre puede tener una flecha apuntando a un padre.

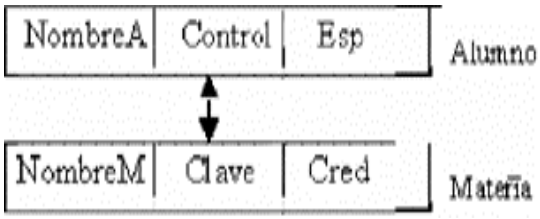
Un esquema de base de datos se representa como una colección de diagramas estructurados en árbol. Cada gráfico tiene una instancia del árbol del repositorio. La raíz de este árbol es un nodo virtual. Los nodos secundarios de este nodo son instancias de registros de base de datos. Cada instancia derivada puede tener múltiples instancias de múltiples registros.

En términos de cardinalidad, la representación es la siguiente: Considere la relación estudiante-sujeto sin propiedades descriptivas.

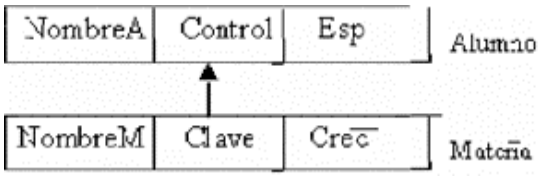


La conversión por base es:

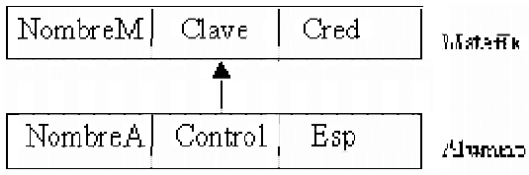
• Cuando la relación es uno a uno.



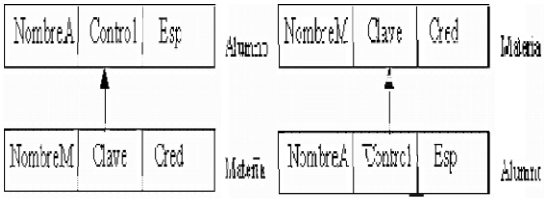
• Cuando la relación es de uno a muchos.



• Si la razón es muchos a uno.



• Si la relación es de muchos a muchos.



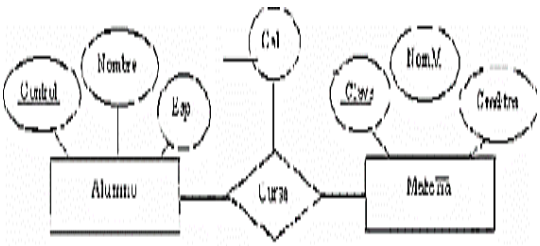
Cuando las relaciones tienen atributos descriptivos, la conversión de diagramas E-R en estructuras de árbol implica los siguientes pasos:

1. Cree un nuevo tipo de registro.

2. Crea enlaces relevantes.

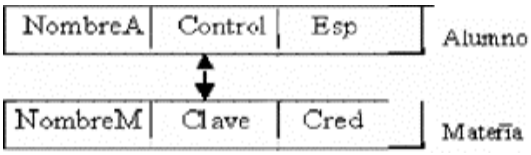
3. Considere que a la relación Estudiante-Sujeto le sumamos el atributo Cal a la relación que existe entre ambos, y luego nuestro modelo E-R resulta:

Diagrama E-R adjunto

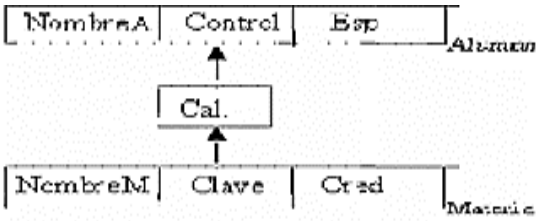


Dependiendo de la cardinalidad, el diagrama de árbol puede verse así:

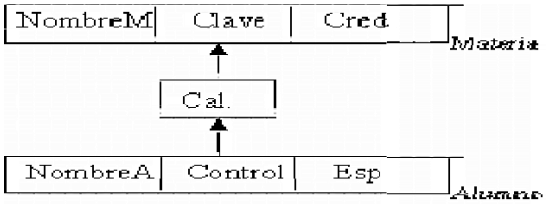
• Cuando la relación es uno a uno.



• Cuando la relación es de uno a muchos.



• Si la razón es muchos a uno.



• Si la relación es de muchos a muchos. Si la relación es de muchos a muchos, la conversión a un diagrama de árbol es un poco más complicada porque el modelo jerárquico solo puede representar relaciones de uno a uno o de uno a muchos.

Hay varias formas diferentes de traducir este tipo de relación a una estructura de árbol, pero todas las formas son duplicados de algunos registros.

Decidir qué método de conversión utilizar depende de muchos factores, entre ellos:

• Tipos de consultas esperadas en la base de datos.

• En qué medida el esquema de la base de datos global a modelar corresponde al diagrama E-R dado.

A continuación, se describe cómo convertir un diagrama E-R en una estructura de árbol de muchos a muchos. Asumimos un ejemplo de relación estudiante-sujeto.

1. Cree dos diagramas de estructura de árbol diferentes, T1 y T2. Cada diagrama de estructura de árbol contiene tipos de registros estudiante y materia. La raíz del árbol T1 es estudiante y la raíz de T2 es materia.

2. Crea este enlace:

• Enlace de muchos a uno desde el recuento de registros al registro de estudiante T1

• Enlace muchos a uno desde el tipo de registro de cliente al tipo de registro de transacción T2. Como se muestra abajo:

Recuperación de datos

La manipulación de información en una base de datos jerárquica requiere un lenguaje de manipulación de datos que consta de una serie de comandos entretejidos en el lenguaje central Pascal.

La orden GET

Utilice este comando para realizar la recuperación de datos y realice los siguientes pasos:

• Buscar un registro en la base de datos y actualizar el índice que mantiene la dirección del último registro accedido.

• Copie los datos solicitados en el tipo de registro apropiado para la consulta.

El comando GET debe especificar el árbol de la base de datos en el que buscar.

Para revisar consistentemente todos los registros, se deben ordenar los registros. Normalmente se utiliza el orden previo, que implica buscar desde la raíz y atravesar recursivamente los subárboles de izquierda a derecha. Entonces comenzamos en la raíz, visitamos el nodo hijo más a la izquierda y así sucesivamente hasta llegar a un nodo hoja (uno sin hijos). A continuación, movemos el padre de la hoja de trabajo hacia atrás y accedemos al hijo no visitado más a la izquierda. Seguimos de esta manera hasta visitar todo el árbol.

Hay dos comandos Obtener diferentes para buscar registros en el árbol de la base de datos.

La secuencia más simple tiene la siguiente forma:

GET FIRST <Tipo de registro>

WHERE<CONDITION)

Donde la cláusula WHERE es opcional. La <CONDICIÒN> es un predicado que puede hacer referencia a cualquier tipo de registro que sea el antepasado del de registro> o al de registro> mismo. El comando Obtener busca el primer registro de tipo en la base de datos que cumple las siguientes condiciones (en preorden) de donde cláusulas. Si se omite la cláusula donde, se encuentra el primer registro de tipo de registro>. Cuando se encuentra un registro, un cursor con la dirección del último registro al que se accedió apunta al registro y el contenido del registro se copia en el registro consultado.

Para ilustrar lo anterior, veamos un ejemplo:

Ejecutemos una consulta que imprima los nombres de los estudiantes llamados Louis A. (Piense en la relación que creamos entre el estudiante y la materia).

GET FIRST estudiante

WHERE Estudiante. NombreA="Luis A.";

PRINT (estudiante. Control)

Ahora digamos que queremos preguntar el nombre de la materia (si la hay) por la cual el estudiante Luis A. A tiene 100 puntos.

GET FIRST estudiante

WHERE Estudiante. NombreA="Luis A." **AND** Curso. Cal= 100;

IF DB-Status=0 entonces PRINT (Matter.MName);

La condición incluye una variable DB-Status que indica si se ha encontrado el registro. El comando obtener primero solo mostrará el primer registro encontrado que coincida con la secuencia de consulta, pero puede haber más registros y usamos el comando obtener siguiente para buscar otros registros.

Su estructura es:

GET NEXT <Tipo de registro)

WHERE <Condición**>**

Encuentra la siguiente entrada en la secuencia anterior que coincide con la condición. Si se omite la cláusula

donde, se encuentra el siguiente registro de tipo.

Actualización de datos.

✓ Establecer nuevos récords. El comando para insertar registros es: insert <Tipo de registro>

WHERE>Condición)

Entre ellos: el tipo de registro contiene los datos del campo del registro a insertar.

Si se incluye una cláusula donde, el sistema buscará en el árbol (secuencia) de la base de datos registros que coincidan con los criterios dados. Una vez encontrada, la entrada creada se colocará en el árbol como el elemento secundario más a la izquierda. Si se omite la cláusula donde, el nuevo registro se inserta en la primera posición en el árbol de la base de datos donde se puede insertar un registro del mismo tipo que el nuevo registro (en preorden).

✓ Editar registros existentes.

El comando para cambiar la entrada es el siguiente:

REPLACE

Este comando no requiere que los datos del registro se cambien como parámetro. El registro afectado será el registro al que apunta el indicador actual y debe ser el registro que se está cambiando.

REPLACE;

Se agrega la palabra "reservado" para informarle al sistema que se cambiará el registro.

✓ Eliminación de registro

Para eliminar una entrada, apunte el indicador actual a la entrada y luego ejecute el comando

Para eliminar, al igual que el comando reemplazar, debe ingresar el comando Mantener.

Registro Virtual.

Como se mencionó anteriormente, las relaciones de muchos a muchos requieren datos duplicados para mantener la organización de la estructura de árbol de la base de datos.

La duplicación de mensajes crea dos problemas importantes:

• Las actualizaciones pueden causar inconsistencias en los datos.

• Provoca una pérdida importante de espacio.

Para solucionar estos problemas se introduce el concepto de registro virtual. Un registro virtual no contiene ningún dato almacenado, sino que es un puntero lógico que apunta a un registro físico específico.

Si es necesario duplicar un registro en varios árboles de bases de datos, se almacena una copia del registro en uno de los árboles y utilizamos un registro ficticio que contiene la dirección del registro de entidad original en los otros registros.

Base de datos orientada a objetos.

Las aplicaciones de bases de datos en áreas como el diseño asistido por computadora, el desarrollo de software y el procesamiento de documentos no se ajustan a varios supuestos sobre las aplicaciones basadas en bases de datos.

Procesamiento de datos.

Se han propuesto modelos de datos orientados a objetos para abordar algunos de estos nuevos tipos de aplicaciones.

El modelo de base de datos orientada a objetos es una adaptación de los sistemas de bases de datos. Se basa en el concepto de encapsular datos y código que opera sobre ellos en objetos. Los objetos estructurales se agrupan en categorías. Basado en una extensión del concepto ISA a un modelo entidad-relación, los conjuntos de categorías se estructuran en subcategorías y super categorías. Dado que el valor de los datos en el objeto también es un objeto, puede representar el contenido del objeto, formando así un objeto compuesto.

El propósito de un sistema de base de datos es procesar grandes cantidades de información. Las primeras bases de datos aparecieron con el desarrollo de los sistemas de gestión de archivos. Estos sistemas primero evolucionaron hacia bases de datos en red o bases de datos jerárquicas y luego hacia bases de datos relacionales.

Estructura del objeto.

El modelo orientado a objetos se basa en encapsular código y datos en una unidad llamada objeto. La interfaz entre el objeto y el resto del sistema está definida por un conjunto de mensajes. La instalación está asociada con:

• Un conjunto de variables que contienen los datos del objeto. Cada valor de variable es un objeto.

• Un conjunto de mensajes a los que responde el objeto.

• Un método es un fragmento de código que implementa cada mensaje. El método devuelve un valor en respuesta al mensaje.

El término "mensajería" en un contexto orientado a objetos no se refiere al uso físico de mensajes en una red informática, sino más bien a la transferencia de solicitudes entre objetos sin tener en cuenta detalles de implementación específicos.

La capacidad de cambiar la definición de un objeto sin afectar al resto del sistema se considera una de las principales ventajas del modelo de programación orientada a objetos.

Jerarquía de clases.

La base de datos contiene objetos que responden al mismo mensaje, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. No tiene sentido definir estos objetos por separado, por lo que los objetos similares se agrupan en una clase y cada objeto se denomina instancia de su propia clase. Todos los objetos de esta categoría comparten una definición común, aunque difieren en los valores asignados a las variables.

Básicamente, el objetivo de una base de datos orientada a objetos es agrupar elementos similares en una entidad en una clase, dejando elementos diferentes separados en otra clase. Por ejemplo: volvemos a la relación estudiante-curso-materia y agregamos una entidad docente que toma los atributos de cada estudiante: nombre, dirección, número de teléfono, especialización, semestre, especialización: nombre, dirección, número de teléfono, finanzas; número, ubicación, asunto RFC: nombre, crédito, contraseña.

Los atributos Nombre, Dirección y Teléfono se repiten en las entidades Estudiante y Profesor, por lo que podemos agrupar estos elementos creando una clase Persona con estos campos. Solicitudes separadas para estudiantes: especialización, semestre, grupo. Y maestro: finanzas, área y tema RFC no forman parte de la agrupación (categoría de persona) porque esta categoría solo indica datos personales, por lo que sigue siendo una categoría de tema.

HERENCIA.

Las categorías en un sistema orientado a objetos se representan en una estructura jerárquica como se muestra arriba, por lo que los atributos o características del elemento Persona estarán contenidos (heredados) por los elementos Estudiante y Profesor. Decimos que las entidades Estudiante y Profesor son subclases de la clase Persona. Este concepto es similar al utilizado en las clases de especialización (relaciones ISA) del modelo E-R.

Para simplificar el modelo se pueden crear muchas agrupaciones (categorías), por lo que la jerarquía (en forma gráfica) puede ser muy amplia, en estos casos necesitamos tener buenos separadores que intervengan entre las categorías y los objetos que las heredan.

CONSULTAS ORIENTADAS A OBJETOS:

Los lenguajes de programación orientados a objetos requieren que toda interacción con los objetos se realice mediante el envío de mensajes. Veamos el ejemplo de estudiante-curso-materia, donde queremos consultar la base de datos para 1 estudiante que tome esta materia.

Por lo tanto, el lenguaje de consulta de un sistema de base de datos orientado a objetos debe incluir un modelo para transferir información de un objeto a otro y un modelo para transferir información de una colección a otra.

CAMBIA LA COMPLEJIDAD.

Los siguientes cambios son posibles en bases de datos orientadas a objetos:

• Agregar una nueva categoría: Para realizar este proceso, la nueva categoría debe ubicarse en la jerarquía de categorías o subcategorías y manejar las variables o métodos heredados relevantes.

• Eliminar una categoría: requiere varias operaciones, manejar elementos heredados de una categoría a otras categorías y reestructurar la jerarquía.

Por sí solo, construir modelos orientados a objetos simplifica la estructura al evitar elementos o variables repetidos dentro de cada unidad, pero el costo de hacerlo es que, si el modelo es complejo, se requiere un manejo cuidadoso de las relaciones objeto-categoría. la complejidad de la modificación y eliminación de clases, ya que

recuperar variables de otros objetos requiere una reorganización que involucra una serie compleja de operaciones.

BIBLIOGRAFÍA

1. Procesamiento de Bases de Datos, Fundamentos, diseño e instrumentación Quinta edición David M. Kroenke Ed. Prentice Hall México 1996
2. Sistemas de Bases de Datos, Administración y uso Alice Y. H. Tsai Editorial Prentice Hall México 1990
3. DB2/SQL, Manual para programadores Tim Martyn Tim Hartley Editorial Mc.Graw Hill España 1991
4. Fundamentos de Bases de Datos Segunda edición Henry F. Korth Abraham Silberschatz Editorial Mc.Graw Hill
5. http://www.itlp.edu.mx/publica/tutoriales/basedat1/index.htm

INSTITUTO SUPERIOR
TECNOLÓGICO PELILEO

TOMO 2:

Base de Datos Avanzada

CONTENIDOS

01

CAPÍTULO UNO
Definición de base de datos.
Objetivos de los sistemas de base de datos.
Abstracción de la información.
Modelo de datos
Manejador de base de datos.
Administrador de base de datos.
Usuarios de las bases de datos.
Estructura general del sistema de base de datos

02

CAPÍTULO DOS
Entidades y conjunto de entidades.
Relaciones y conjunto de relaciones.
Limitantes de mapeo.
Llaves primarias.
Diagrama entidad de relación.
Reducción del diagrama entidad de relación a tablas
Generalización y especialización

03

CAPÍTULO TRES
Estructura de las bases de datos relacionales.
Lenguajes de consulta formales.
Lenguajes de consulta comerciales.
Modificación de la base de datos
Vistas.

04

CAPÍTULO CUATRO
Peligros en el diseño de la base de datos relacionales
Primera y segunda forma normal
Tercera forma normal
Cuarta y quinta forma normal

CAPÍTULO CINCO
Conceptos básicos
Diagrama de estructura de datos
El modelo Codossyl DBTG.

BIBLIOGRAFÍA

01

TRIGGERS (DISPARADORES) EN UNA BASE DE DATOS.

¿QUÉ ES UN TRIGGER?

Desde la aparición de la versión 5.0.3 de la base de datos MySQL, se ha implementado la posibilidad de desarrollar triggers para las tablas de una base de datos. Recordar que un trigger es un objeto asociado a una tabla que es ejecutado cuando sucede un evento en la tabla propietaria. Son aquellas sentencias (INSERT, UPDATE, DELETE) que modifican los datos dentro de una tabla. Sólo puede haber un trigger de cada insert, update y delete por tabla o vista.

Pero los triggers tienen una ventaja obvia. Es código que se ejecuta en el servidor de la base de datos, y no en la máquina cliente.

Un Trigger o Disparador es un programa almacenado (stored programo SP), creado para ejecutarse automáticamente cuando ocurra un evento en una tabla o vista de la base de datos. Dichos eventos son generados por los comandos INSERT, UPDATE y DELETE.

Un Disparador nunca se llama directamente, en cambio, cuando una aplicación o usuario intenta insertar, actualizar, o anular una fila en una tabla, la acción definida en el disparador se ejecuta automáticamente (se dispara).

Las ventajas de los triggers son varias:

1. La entrada en vigor automática de restricciones de los datos, hace que los usuarios entren sólo valores válidos.

2. El mantenimiento de la aplicación se reduce, los cambios a un disparador se reflejan automáticamente en todas las aplicaciones que tienen que ver con la tabla sin la necesidad de recompilar o enlazar.

3. Logs automáticos de cambios a las tablas. Una aplicación puede

guardar un registro corriente de cambios, creando un disparador que se active siempre que una tabla se modifique.

4. La notificación automática de cambios a la Base de Datos con alertas de evento en los disparadores.

Decimos que los Triggers se invocan para ejecutar un conjunto de instrucciones que protejan, restrinjan, actualicen o preparen la información de las tablas, al momento de manipular la información. Para crear triggers son necesarios los privilegios SUPER y TRIGGER.

SINTAXIS DE UN TRIGGER

Usamos la siguiente sintaxis:

CREATE
[DEFINER={usuario | CURRENT_USER}]

TRIGGER nombre_del_trigger
{BEFORE | AFTER}
{UPDATE | INSERT | DELETE} ON
nombre_de_la_tabla

FOR EACH ROW

<bloque_de_instrucciones>

Obviamente la sentencia CREATE es conocida para crear nuevos objetos en la base de datos. Explicación de las partes de la definición:

DEFINER={usuario | CURRENT_USER}: Indica al gestor de bases de datos qué usuario tiene privilegios en su cuenta, para la invocación de los triggers cuando surjan los eventos DML. Por defecto esta característica tiene el valor CURRENT_USER que hace referencia al usuario actual que está creando el Trigger.

- nombre_del_trigger: Indica el nombre del trigger. Por defecto existe una nomenclatura práctica para nombrar un trigger, la cual da mejor legibilidad en la administración de la base de datos. Primero, escriba el nombre

de tabla, luego especifique con la inicial de la operación DML y seguido usamos la inicial del momento de ejecución (AFTER o BEFORE). Por ejemplo:

BEFORE INSERT clientes_BI_TRIGGER

- BEFORE | AFTER: Especifica si el Trigger se ejecuta antes o después del evento DML.
- UPDATE | INSERT | DELETE: Aquí elija que sentencia usa para que se ejecute el Trigger.
- ON nombre_de_la_tabla: En esta sección establece el nombre de la tabla asociada.
- FOR EACH ROW: Establece que el Trigger se ejecute por cada fila en la tabla asociada.
- <bloque_de_instrucciones>: Define el bloque de sentencias que el Trigger ejecuta al ser invocado.

IDENTIFICADORES NEW Y OLD EN TRIGGERS

Si requiere relacionar el trigger con columnas específicas de una tabla debemos usar los identificadores OLD y NEW.

OLD indica el valor antiguo de la columna y NEW el valor nuevo que pudiese tomar. Por ejemplo: OLD.idproducto o NEW.idproducto.

Si usa la sentencia UPDATE se refiere a un valor OLD y NEW, ya que modifica registros existentes por los valores. En cambio, si usa INSERT sólo usa NEW, ya que su naturaleza es únicamente de insertar nuevos valores a las columnas. Y con DELETE usa OLD debido a que borra valores existentes.

TRIGGERS BEFORE Y AFTER

Estas cláusulas indican si el Trigger se ejecuta BEFORE (antes) o AFTER (después) del evento DML. Hay ciertos eventos que no son compatibles con estas sentencias.

Sí tiene un Trigger AFTER que se ejecuta en una sentencia UPDATE, es ilógico editar valores nuevos NEW, sabiendo que el evento ya ocurrió. Igual ocurre con la sentencia INSERT, el Trigger tampoco podría hacer referencia a valores NEW, ya que los valores que en algún momento fueron NEW, han pasado a ser OLD.

¿QUÉ UTILIDADES TIENEN LOS TRIGGERS?

Con los Triggers implementamos varios casos de uso que mantengan la integridad de la tabla de la base de datos, como Validar información, Calcular atributos derivados, Seguimiento de movimientos de datos en tablas de la base de datos, etc.

Cuando surja una necesidad para que se ejecute una acción implícitamente (sin ejecución manual) sobre los registros de una tabla, considera el uso de un Trigger.

Aunque los triggers sirven en su mayoría para mantener la integridad de la base de datos también pueden usarse para:

AUDITORIA

- Los triggers se usan para llenar tablas de auditoría, en donde se registren ciertos tipos de transacciones o accesos hacia tablas.

REGLAS DE NEGOCIO

Cuando ciertas reglas de negocio son muy elaboradoras y necesitan ser expresadas a nivel base de datos, es necesario que además de reglas y restricciones se utilicen triggers.

VALIDACIÓN DE DATOS

- Los triggers pueden controlar ciertas acciones, por ejemplo: pueden rechazar o modificar ciertos valores que no cumplan determinadas reglas,

para prevenir que datos inválidos sean insertados en la base.

SEGURIDAD

- Refuerzan las reglas de seguridad de una aplicación.
- Ofrecen verificación de seguridad basada en valores.
- Integridad de los datos mejorada.
- Fuerzan restricciones dinámicas de integridad de datos y de integridad referencial.
- Aseguran que las operaciones relacionadas se realizan juntas de forma implícita.
- Respuesta instantánea ante un evento auditado.
- Ofrece un mayor control sobre la Base de Datos.

DESVENTAJAS

•Hay que definir con anticipación la tarea que realizara el trigger

•Peligro de pérdida en Reorganizaciones.

•Hay que programarlos para cada DBMS.

•Un Trigger nunca se llama directamente.

•Los triggers no se desarrollan pensando en un sólo registro, los mismos deben funcionar en conjunto con los datos ya que se disparan por operación y no por registro.

•Por funcionalidad, no hay que poner en uno solo las funciones de INSERT, UPDATE y DELETE.

•Utilizar moderadamente los triggers.

- No se pueden utilizar en tablas temporales.

02

VISTAS Y TRANSACCIONES EN UNA BASE DE DATOS

DEFINICIONES DE VISTAS

Una vista es una tabla virtual derivada de otras tablas (que pueden ser tablas base o también otras vistas).

Sus tuplas no se almacenan, sino que se generan a partir de las tablas de las que depende (no necesariamente existen en forma física, por ello se las considera tablas virtuales)

Son útiles para usar, como si fueran tablas, consultas que se efectúan frecuentemente, y también para cuestiones de seguridad.

SQL3:

CREATE VIEW <name> [(<column> [, ...])]

AS <query >

[WITH [CASCADE | LOCAL] CHECK OPTION] ;

PROVEEDOR (id_prov, nombreAp, situación, ciudad) PRODUCTO (id_prod, nombre, color, peso, ciudad) ENVIO (id_prov, id_prod, cant)

Crear una vista de los proveedores cuya situación sea superior a 15.

CREATE VIEW Buenos_proveedores (prov, sit, ciudad) AS

SELECT id_prov, situación, ciudad FROM PROVEEDORES

WHERE situación > 15;

Crear una vista que corresponda a los proveedores cuya situación sea superior a 15 pero que no sean de Capital Federal.

CREATE VIEW Buenos_proveedores_noCapital AS

SELECT *

FROM Buenos_proveedores WHERE ciudad <> 'Capital Federal';

Los nombres de columna deberán especificarse explícitamente si:

Cualquier columna se deriva de una función, una expresión operacional o un literal.

Dos o más columnas de la vista reciben el mismo nombre

CREATE VIEW EnvioProveedor (id_prov, cantTotal) AS

SELECT id_prov, SUM (cant)

FROM ENVIO

GROUP BY id_prov;

CREATE VIEW Pareja_de_ciudades (ciudadProv, ciudadProd) AS

SELECT DISTINCT PROVEEDORES.ciudad, PRODUCTO.ciudad

FROM PROVEEDORES, PRODUCTO, ENVIO

WHERE PROVEEDORES.id_prov = ENVIO.id_prov AND ENVIO.id_prod = PRODUCTO.id_prod ;

Consulta: como si fuera una tabla base, por ejemplo SELECT *

FROM Buenos_proveedores WHERE prov < 30

ORDER BY ciudad;

DROP VIEW <Nombre-vista > [CASCADE | RESTRICT];

Ejemplo:

DROP VIEW

Buenos_proveedores;

Si la vista es actualizable y está definida con:

WITH CHECK OPTION: todos los INSERT y UPDATE sobre la vista son chequeados para asegurar que los datos satisfacen la condición de definición de la Vista.

WITH LOCAL CHECK OPTION: chequea la integridad sobre la vista.

WITH CASCADE CHECK OPTION : chequea la integridad sobre la vista y cualquier vista dependiente.

CREATE VIEW Buenos_proveedores (prov, sit, ciudad) AS SELECT id_prov, situación, ciudad FROM PROVEEDORES

WHERE situación > 15 WITH CHECK OPTION ;

Insert into Buenos_proveedores (prov, sit, ciudad) values (10, 20, 'Paris'); □n

Update Buenos_proveedores set situacion = 5 where prov = 20; X

Insert into Buenos_proveedores (prov, sit, ciudad) values (8, 5, 'Roma'); X

No toda vista es actualizable. Es SQL-actualizable si:

- Está definida a partir de una única tabla
- Conserva la clave (primaria o alternativa)
- No está definida mediante agrupación o funciones de agregación
- No incluye la cláusula Distinct
- No incluye subconsultas
- No se define mediante unión, intersección, diferencia

No todas las vistas son actualizables:

CREATE VIEW Proveedor_ciudad (id_prov, ciudad)AS

SELECT id_prov, ciudad

FROM PROVEEDORES;

CREATE VIEW Situación_ciudad (situación, ciudad)AS

SELECT situación, ciudad

FROM PROVEEDORES; ACTUALIZABLE

NO ACTUALIZABLE

Insert into Proveedor_ciudad values (1,'Roma');n

Insert into Situación_ciudad values (40,'Roma');X

CREATE VIEW Proveedor_Londres AS SELECT id_prov, nombreAp, situacion, ciudad FROM PROVEEDORES

WHERE ciudad = 'Londres';

ACTUALIZABLE

Insert into Proveedor_Londres values (15,'Martin Lopez', 5, 'Tandil'); □n

CREATE VIEW EnvioProveedor (id_prov, cantTotal) AS

SELECT id_prov, SUM (cant)

FROM ENVIO

GROUP BY id_prov; NO

Insert into EnvioProveedor (1,100); X

ACTUALIZABLE

Motivos por los que una vista no es actualizable en Oracle:

Contiene operadores de conjuntos (UNION, INTERSECT,…)

Contiene el operador DISTINCT.

Contiene funciones agregadas (SUM, AVG, ..).

Contiene una cláusula GROUP BY

Se define mediante una subconsulta en una lista SELECT

Se define mediante JOIN (con excepción de key-preserved table *)

* Una tabla base o una vista es considerada una key-preserved table si cada valor clave (primary-key o unique-key) en cada tabla base es también clave en el resultado de la Join View.

TRANSACCIONES

Una transacción es una unidad lógica de trabajo o procesamiento (ejecución de un programa que incluye operaciones de acceso a la base de datos).

Una transacción es una secuencia de operaciones que llevan la base de datos desde un estado de consistencia1 a otro estado de consistencia, por esto suele decirse también que la transacción es una unidad lógica de integridad.

Cuando múltiples transacciones son introducidas en el sistema por varios usuarios, es necesario evitar que interfieran entre ellas de forma tal que provoquen que la BD quede en un estado no consistente; desde este punto de vista, podemos ver una transacción como una unidad lógica de concurrencia.

Cuando ocurre un fallo que provoca la caída del sistema, en el momento en el que había varias transacciones en curso de ejecución, muy probablemente dejará erróneos los datos en la BD (estado inconsistente); en estas circunstancias, se debe garantizar que la BD pueda ser recuperada a un estado en el cual su contenido sea consistente, por esto una transacción es considerada también una unidad lógica de recuperación.

La idea clave es que una transacción debe ser atómica, es decir, las operaciones que la componen deben ser ejecutadas en su totalidad o no ser ejecutadas en absoluto.

Una sentencia de definición o manipulación de datos ejecutada de forma interactiva (por ejemplo utilizar el SQL*Plus de Oracle para realizar una consulta) puede suponer el inicio de una transac- ción. Asimismo, la ejecución de una sentencia SQL por parte de un programa que no tiene ya una transacción en progreso, supone la iniciación de una transacción.

Toda transacción finaliza con una operación de commit (confirmar) o bien con una operación de rollback (anular, abortar o revertir).

Tanto una operación como la otra puede ser de tipo explícito (si la propia transacción (su código) contiene una sentencia COMMIT o ROLLBACK) o implícito (si dicha operación es realizada por el sistema de forma automática, por ejemplo, tras detectar una terminación normal (éxito) o anormal (fallo) de la transacción).

Por defecto, una vez finalizada una transacción, si todas sus operaciones se han realizado con éxi- to, se realiza un COMMIT implícito de dicha transacción; y si alguna de ellas tuvo problemas, se lleva a cabo un ROLLBACK implícito de la transacción (es decir, se deshacen todas las operaciones que había realizado hasta el momento del fallo).

El conjunto de valores almacenados en la base de datos cumple toda restricción de integridad especificada en el es- quema, así como cualesquiera otras restricciones que deban cumplirse en la base de datos.

PROPIEDADES (deseables) DE UNA TRANSACCIÓN

Se suele hacer referencia a estas como las propiedades ACID (por sus iniciales en inglés).

1. Atomicidad

Todas las operaciones de la transacción son ejecutadas por completo, o no se ejecuta ninguna de ellas (si se ejecuta la transacción, se hace hasta el final).

2. Consistencia

Una transacción T transforma un estado consistente de la base de datos en otro estado consis- tente, aunque T no tiene por qué preservar la consistencia en todos los puntos intermedios de su ejecución. Un ejemplo es el de la transferencia de una cantidad de dinero entre dos cuentas bancarias.

3. Aislamiento (Isolation)

Una transacción está aislada del resto de transacciones.

Aunque existan muchas transacciones ejecutándose a la vez, cualquier modificación de datos que realice T está oculta para el resto de transacciones hasta que T sea confirmada (realiza COMMIT).

Es decir, para cualesquiera T1 y T2, se cumple que

- T1 ve las actualizaciones de T2 después de que T2 realice COMMIT, o bien

- T2 ve las modificaciones de T1, después de que T1 haga un COMMIT Pero nunca se cumplen ambas cosas al mismo tiempo.

Nota: esta propiedad puede no imponerse de forma estricta2; de hecho, suelen definirse niveles de aislamiento de las transacciones.

4. Durabilidad

Una vez que se confirma una transacción, sus actualizaciones sobreviven cualquier fallo del sistema. Las modificaciones ya no se pierden, aunque el sistema falle justo después de realizar dicha confirmación.

El Subsistema de Recuperación del SGBD es el encargado de conseguir el cumplimiento de las propiedades de atomicidad y durabilidad de las transacciones.

La conservación de la consistencia es una propiedad cuyo cumplimiento han de asegurar, por un lado, los programadores de base de datos, y por otro el Subsistema de Integridad del SGBD.

El Subsistema de Control de Concurrencia es el encargado de conseguir el aislamiento de las tran- sacciones.

OPERACIONES DE UNA TRANSACCIÓN

Para hacer posible el control de la concurrencia de las transacciones, así como la recuperación del sistema tras fallos o caídas del mismo, es necesario almacenar cuándo se inicia, termina, se con- firma o se aborta cada transacción, y además qué modificaciones de qué elementos de BD realiza cada una.

- **INICIO DE TRANSACCIÓN**

Operación que marca el momento en el que una transacción comienza a ejecutarse.

- **LEER o ESCRIBIR**

Operaciones de lectura / escritura de elementos de la base de datos, que se realizan como parte de una transacción.

- **FIN DE TRANSACCIÓN**

Las operaciones de LEER y ESCRIBIR han terminado. En este punto se verifica si la transacción debe abortarse por alguna razón (si viola el control de concurrencia, por ejemplo), o bien si los cambios realizados por la transacción (hasta ahora en buffers de memoria volátil) pueden apli- carse permanentemente a la base de datos en disco (es decir, si puede realizarse una operación de CONFIRMAR (COMMIT)).

- **CONFIRMAR**

La transacción terminó con éxito, todos los cambios que ha realizado se pueden confirmar sin peligro en la BD y ya no serán cancelados.

- **ABORTAR**

La transacción terminó sin éxito y toda actualización que ha realizado se debe cancelar. Algunas técnicas de recuperación necesitan operaciones adicionales como las siguientes:

- **DESHACER**

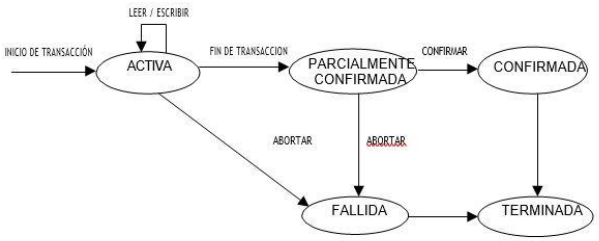
Similar a ABORTAR, pero se aplica a una sola operación y no a una transacción completa.

- **REHACER**

Especifica que algunas de las operaciones realizadas por una transacción deben repetirse, para asegurar que todas las operaciones realizadas por una transacción que ha sido CONFIRMADA se hayan aplicado con éxito a la BD (es decir, los cambios hayan quedado grabados físicamente en disco).

ESTADOS DE UNA TRANSACCIÓN

El siguiente es el diagrama de transición de estados para la ejecución de transacciones:



Una transacción entra en el estado ACTIVA justo después de iniciar su ejecución y, en este estado, puede realizar operaciones LEER y ESCRIBIR.

Cuando la transacción finaliza, pasa al estado PARCIALMENTE CONFIRMADA. En este punto, el Subsistema de Control de Concurrencia puede efectuar verificaciones para asegurar que la transacción no interfiera con otras transacciones en ejecución. Además, el Subsistema de Recuperación puede anotar qué operaciones (qué cambios) ha realizado que la transacción en un fichero del sistema (bitácora3), con el objetivo de garantizar que los cambios realizados por la transacción terminada queden permanentes, a pesar de fallos del sistema.

Una vez realizadas con éxito ambas verificaciones, la transacción ha llegado a su punto de confirmación y pasa al estado CONFIRMADA (ha concluido su ejecución con éxito).

Si una de las verificaciones falla o la transacción se aborta mientras está en estado ACTIVA, pasa al estado FALLIDA. En este caso, es posible que la transacción deba ser cancelada (anulada, revertida, abortada) para anular los efectos de sus operaciones ESCRIBIR sobre la BD.

El estado TERMINADA indica que la transacción ha abandonado el sistema.

Las transacciones fallidas (abortadas) pueden ser reiniciadas posteriormente, ya sea de forma automática por parte del sistema, o bien sea el usuario el que las reintroduzca como si fueran nuevas transacciones.

03

INDICES, CURSORES Y USUARIOS EN UNA BASE DE DATOS

DEFINICIÓN Y USO

Un índice es un puntero a una fila de una determinada tabla de nuestra base de datos. Pero… ¿qué significa exactamente un puntero? Pues bien, un puntero no es más que una referencia que asocia el valor de una determinada columna (o el conjunto de valores de una serie de columnas) con las filas que contienen ese valor (o valores) en las columnas que componen el puntero.

Los índices mejoran el tiempo de recuperación de los datos en las consultas realizadas contra nuestra base de datos. Pero los índices no son todo ventajas, la creación de índices implica un aumento en el tiempo de ejecución sobre aquellas consultas de inserción, actualización y eliminación realizadas sobre los datos afectados por el índice (ya que tendrán que actualizarlo). Del mismo modo, los índices necesitan un espacio para almacenarse, por lo que también tienen un coste adicional en forma de espacio en disco.

La construcción de los índices es el primer paso para realizar optimizaciones en las consultas realizadas contra nuestra base de datos. Por ello, es importante conocer bien su funcionamiento y los efectos colaterales que pueden producir.

¿Para qué los usa MySQL?

MySQL emplea los índices para encontrar las filas que contienen los valores específicos de las columnas empleadas en la consulta de una forma más rápida. Si no existiesen índices, MySQL empezaría buscando por la primera fila de la tabla hasta la última buscando aquellas filas que cumplen los valores establecidos para las columnas empleadas en la consulta. Esto implica que, cuantas más filas tenga la tabla, más tiempo tardará en realizar la consulta. En cambio, si la tabla contiene índices en las columnas empleadas en la consulta, MySQL tendría una referencia directa hacia los datos sin necesidad de recorrer secuencialmente todos ellos.

En general, MySQL emplea los índices para las siguientes acciones:

Encontrar las filas que cumplen la condición WHERE de la consulta cuyas columnas estén indexadas.

Para recuperar las filas de otras tablas cuando se emplean operaciones de tipo JOIN. Para ello, es importante que los índices sean del mismo tipo y tamaño ya que aumentará la eficiencia de la búsqueda. Por ejemplo: una operación de tipo JOIN sobre dos columnas que tengan un índice del tipo INT(10).

Disminuir el tiempo de ejecución de las consultas con ordenación (ORDER BY) o agrupamiento (GROUP BY) si todas las columnas presentes en los criterios forman parte de un índice.

Si la consulta emplea una condición simple cuya columna de la condición está indexada, las filas serán recuperadas directamente a partir del índice, sin pasar a consular la tabla.

¿Qué tipos de índices hay?

A continuación, vamos a analizar los distintos tipos de índices que se pueden crear y las condiciones que deben cumplir cada uno de ellos:

INDEX (NON-UNIQUE): este tipo de índice se refiere a un índice normal, no único. Esto implica que admite valores duplicados para la columna (o columnas) que componen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea

simplemente para mejorar el tiempo de ejecución de las consultas.

UNIQUE: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único. Esto implica que no admite valores duplicados para la columna (o columnas) que componen el índice. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.

PRIMARY: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único (al igual que en el caso del índice UNIQUE) pero con la limitación de que sólo puede existir un índice PRIMARY en cada una de las tablas. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.

FULLTEXT: estos índices se emplean para realizar búsquedas sobre texto (CHAR, VARCHAR y TEXT). Estos índices se componen por todas las palabras que están contenidas en la columna (o columnas) que contienen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea simplemente para mejorar el tiempo de ejecución de las consultas. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.

SPATIAL: estos índices se emplean para realizar búsquedas sobre datos que componen formas geométricas representadas en el espacio. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.

Es importante destacar que todos estos índices pueden construirse empleando una o más columnas. Del mismo modo, el orden de las columnas que se especifique al construir el orden es relevante para todos los índices menos para el FULLTEXT (ya que este índice mira en TODAS las columnas que componen el índice).

Para crear un índice, se empleará la siguiente estructura:

«CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name ON table_name (index_col_name…) index_type;»

Donde:

index_name: es el nombre del índice.

table_name: es el nombre de la tabla donde se va a crear el índice.

index_col_name: nombre de la columna (o columnas) que formarán el índice.

index_type: es el tipo del índice. Se emplea con USING [BTREE | HASH].

Un ejemplo sería:

CREATE UNIQUE INDEX mi_indice_unico ON mi_tabla (mi_columna) USING HASH;

¿En qué estructuras se almacenan los índices?

Una vez hemos visto los tipos de índices, vamos a ver los distintos tipos de estructuras que se pueden crear para almacenar los índices junto con las características de cada uno de ellas:

B-TREE: este tipo de índice se usa para comparaciones del tipo =, >, <, >=, <=, BETWEEN y LIKE (siempre y cuando se utilice sobre constantes que no empiecen por %). Para realizar búsquedas empleando este tipo de índice, se empleará cualquier columna (o conjunto de columnas) que formen el prefijo del índice. Por ejemplo: si un índice está formado por las columnas [A, B, C], se podrán realizar búsquedas sobre: [A], [A, B] o [A, B, C].

HASH: este tipo de índice sólo se usa para comparaciones del tipo = o <=>. Para este tipo de operaciones son muy rápidos en comparación a otro tipo de estructura. Para realizar búsquedas empleando este tipo de índice, se emplearán todas las columnas que componen el índice.

Un índice puede ser almacenado en cualquier tipo de estructura, pero, en función del uso que se le vaya a dar, puede interesar crear el índice en un tipo determinado de estructura o en otro. Por norma general, un índice siempre se creará con la estructura de B-TREE, ya que es la estructura más empleada por la mayoría de operaciones.

¿Por qué es bueno utilizar índices?

Como hemos visto en los apartados anteriores, los índices permiten optimizar las consultas sobre los elementos de la base de datos. Esto desemboca en una reducción considerable del tiempo de ejecución de la consulta. Esta reducción de tiempo es visible sobre tablas de gran tamaño. Sobre tablas pequeñas, el uso de índices no aporta una disminución drástica del tiempo de ejecución de la consulta ya que, prácticamente, MySQL tarda lo mismo en acceder al índice que en acceder de forma secuencial al contenido de la tabla para buscar la fila deseada.

¿por qué no crear índices para todas las columnas?

No todas las soluciones son perfectas y tampoco lo iba a ser la creación de índices. La creación de índices también tiene efectos negativos. Estos efectos negativos es bueno conocerlos ya que pueden ocasionar efectos colaterales no deseados.

Uno de ellos es que las operaciones de inserción, actualización y eliminación que se realicen sobre tablas que tengan algún tipo de índice (o índices), verán aumentado su tiempo de ejecución. Esto es debido a que, después de cada una de estas operaciones, es necesario actualizar el índice (o los índices) presentes en la tabla sobre la que se ha realizado alguna de las operaciones anteriores.

Otro efecto negativo es que los índices deben ser almacenados en algún lugar. Para ello, se empleará espacio de disco. Por ello, el uso de índices aumenta el tamaño de la base de datos en mayor o menor medida.

Conclusiones

Como se ha visto a lo largo del tutorial, los índices tienen aspectos positivos y aspectos negativos. Los aspectos positivos permiten que las consultas de tipo WHERE se ejecuten de forma más rápida siempre y cuando se empleen columnas con índices en las condiciones de la consulta. Por otra parte, empeora el tiempo sobre las consultas de inserción, actualización y eliminación sobre las tablas que contengan índices. Por norma general, las consultas de tipo WHERE son más predominantes que el resto de consultas e interesa que éstas se ejecuten lo más rápido posible. En cambio, las consultas de inserción, actualización y eliminación pueden ver disminuido su tiempo de ejecución sin comprometer demasiado la usabilidad del sistema.

CURSORES EN UNA BASE DE DATOS

En base de datos un Cursor es un mecanismo el cual nos permite procesar fila por fila el resultado de una consulta.

Como sabemos SQL es un lenguaje orientado a conjuntos. Si nosotros deseamos alterar ciertos elementos en nuestra colección tendremos que hacerlo mediante condicione. Única y exclusivamente los elementos que cumpla con dichas condiciones podrán ser alterados. Con los cursores podremos trabajar con cada uno de los elementos (filas) de nuestra consulta sin tener que obtener nuevos conjuntos. Esto nos permitirá ser mucho más flexibles al momento de manipular la información.

Para nosotros poder hacer uso de un cursor será necesario seguir los siguientes pasos.

Crear un cursor a partir de una sentencia SQL.

Apertura del cursor.

Acceso a datos.

Cierre del cursor.

Los cursores en SQL son estructuras que permiten a los desarrolladores y administradores de bases de datos procesar de manera controlada y fila por fila el conjunto de resultados de una consulta SQL. En lugar de manejar todo el resultado de una consulta a la vez, los cursores permiten recorrer cada fila individualmente, facilitando operaciones complejas que requieren un procesamiento detallado de cada fila.

Características principales de los cursores:

1. Procesamiento fila por fila: Permiten iterar sobre un conjunto de resultados fila por fila.

2. Control sobre el conjunto de datos: Ofrecen un mayor control sobre el procesamiento de datos, especialmente útil para operaciones que no pueden realizarse fácilmente con comandos SQL estándar.

3. Declaración y uso explícito: Deben ser declarados, abiertos, utilizados y cerrados explícitamente.

Tipos de cursores:

Existen varios tipos de cursores en SQL, cada uno con características específicas:

1. Cursores implícitos: Son manejados automáticamente por el sistema de base de datos y se utilizan para operaciones básicas como la ejecución de comandos SELECT.

2. Cursores explícitos: Son definidos y controlados por el usuario. Estos cursores deben ser declarados y gestionados manualmente.

3. Cursores de solo avance (forward-only): Solo permiten moverse hacia adelante en el conjunto de resultados.

4. Cursores sensibles e insensibles: Determinan si los cambios en la base de datos son visibles durante la iteración.

5. Cursores dinámicos: Permiten ver los cambios en el conjunto de resultados mientras se recorre.

Cómo utilizar cursores en SQL

El uso de cursores en SQL sigue una serie de pasos básicos: declaración, apertura, obtención de datos y cierre.

Declaración de un cursor

Primero, se declara un cursor definiendo el conjunto de resultados que manejará.

```
DECLARE cursor_name CURSOR FOR

SELECT column1, column2

FROM table_name

WHERE condition;
```

Ejemplo práctico

A continuación, se muestra un ejemplo completo que ilustra el uso de cursores en SQL.

```
-- Declarar el cursor
DECLARE employee_cursor CURSOR FOR
SELECT employee_id, employee_name
FROM employees
WHERE department = 'Sales';
-- Declarar variables para almacenar los datos
DECLARE @employee_id INT, @employee_name NVARCHAR(50);
-- Abrir el cursor
OPEN employee_cursor;
-- Obtener y procesar cada fila
FETCH NEXT FROM employee_cursor INTO @employee_id, @employee_name;
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Operaciones a realizar con los datos obtenidos
    PRINT 'Employee ID: ' + CAST(@employee_id AS NVARCHAR(10));
    PRINT 'Employee Name: ' + @employee_name;

    -- Obtener la siguiente fila
    FETCH NEXT FROM employee_cursor INTO @employee_id, @employee_name;
END;
-- Cerrar y liberar el cursor
CLOSE employee_cursor;
DEALLOCATE employee_cursor;
```

Ventajas y desventajas de los cursores

Ventajas:

Control detallado: Permiten un control fino sobre el procesamiento de filas individuales.

Flexibilidad: Útiles para operaciones complejas que no pueden ser resueltas fácilmente con consultas SQL estándar.

Desventajas:

Rendimiento: Pueden ser menos eficientes que las operaciones en conjunto, especialmente con grandes volúmenes de datos.

Complejidad: Añaden complejidad al código SQL y pueden ser más difíciles de mantener.

Conclusión:

Los cursores en SQL son herramientas poderosas que permiten el procesamiento detallado de filas en un conjunto de resultados. Aunque pueden impactar en el rendimiento y añadir complejidad, su capacidad para manejar operaciones fila por fila los hace indispensables en ciertos escenarios. Con un uso adecuado, los cursores pueden ser extremadamente útiles para gestionar datos en bases de datos SQL.

Permisos de usuario de base de datos predeterminados.

el servidor de base de datos permite que más de un usuario pueda trabajar con los recursos del servidor (registros, tablas, bases de datos, funciones, etc..).

Así los únicos administradores del servidor (algo que muy pocas veces sucede) no deberíamos tener problemas en seguir utilizando root, sin embargo, si más personas trabajarán con el servidor, será necesario que generemos nuevos usuarios y asignemos los permisos pertinentes.

Agregar nuevos usuarios.

Para que nosotros generemos un nuevo usuario lo primero que debemos de hacer es autenticarnos en el servidor.

mysql -u root -p -h localhost<ip>

Posteriormente debemos de generar un nuevo Usuario, Para esto ejecutamos la siguiente sentencia.

CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'password';

Hasta este punto, nosotros ya podemos autenticarnos con el servidor utilizando el nuevo usuario, sin embargo, una vez autenticado las acciones que podemos hacer son mínimas, debido a que este usuario no posee los permisos necesarios para trabajar con las bases de datos.

Asignar permisos

Para poder establecer permisos, las siguientes sentencias deben de ejecutarse utilizando el usuario root.

Si queremos que el nuevo usuario tenga permisos de administrador (Todos los permisos), debemos de ejecutar la siguiente sentencia.

GRANT ALL PRIVILEGES ON *.* TO 'nombre_usuario'@'localhost';

Los asteriscos indican que los permisos serán asignados a todas las bases de datos y a todas las tablas (primer asteriscos bases de datos, segundo asterisco tablas).

Si queremos asignar permisos para ciertas acciones, la sentencia quedaría de la siguiente manera. Reemplazamos ALL PRIVILEGES y colocamos las acciones que queremos asignar.

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP

-> ON codigofacilito.*

-> TO 'nombre_usuario'@'localhost';

En esta ocasión estamos indicando que el nuevo usuario podrá consultar, crear, actualizar y eliminar registros, así como podrá crear o eliminar elementos (tablas, índices, columnas, funciones, stores, etc ...).

Todos estos permisos serán válidos únicamente en la base de datos * y se aplicarán a todas las tablas.

Si queremos asignar permisos únicamente a una tabla, reemplazamos el asteriscos por el nombre de la tabla.

Una vez hayamos finalizado con los permisos, el último paso será refrescarlos.

FLUSH PRIVILEGES;

Permisos

Aquí un listado de algunos permisos que podemos asignar.

CREATE permite crear nuevas tablas o bases de datos.

DROP permite eliminar tablas o bases de datos.

DELETE permite eliminar registros de tablas.

INSERT permite insertar registros en tablas.

SELECT permite leer registros en las tablas.

UPDATE permite actualizar registros en las tablas.

GRANT OPTION permite remover permisos de usuarios.

SHOW DATABASE Permite listar las bases de datos existentes.

Existen diferentes tipos de Privilegios Se distinguen dos tipos:

- De sistema: Permite realizar determinadas acciones en la base de datos (Por ejemplo, crear espacios de almacenamiento, crear usuarios, ...) o en cualquier esquema.
- Sobre objetos: Permite a un usuario acceder y manipular o ejecutar objetos concretos (tablas, vistas, secuencias, procedimientos, funciones o paquetes).

Otorgar privilegios

Para que un usuario pueda otorgar un privilegios de sistema bien debe haberse otorgado con ADMIN OPTION, permite a aquel a quien se le concede el privilegio poder otorgarlo o haber sido concedido el privilegio GRANT ANY PRIVILEGE .

Al especificar ALL PRIVILEGES se otorgan todos los privilegios de sistema. La cláusula PUBLIC

otorga el privilegio a todos los usuarios.

Quitar privilegios

REVOKE <privilegio>/ALL_PRIVILEGES FROM <usuario>/<rol>/PUBLIC;

Cualquier usuario con la opción ADMIN OPTION sobre un privilegio puede revocarlo. Quien lo hace no tiene porque ser el usuario que originalmente lo otorgo. Al retirar ciertos privilegios determinados objetos pueden quedar inconsistentes (procedimientos o vistas consultadas merced al privilegio SELECT ANY TABLE).

En el caso de ADMIN OPTION no hay un efecto en cascada cuando se retira un privilegio referente a operaciones DDL (por ej. CREATE TABLE); si lo hay cuando se revoca un privilegio referente a operaciones DML (por ejemplo SELECT ANY TABLE). Si se retira un privilegio de sistema de PUBLIC , pero existen usuarios a los que se ha otorgado aquel directamente o a través de roles, estos siguen pudiéndolo usar.

ROLES

Es un grupo de privilegios, de sistema o sobre objetos, a los que se les da un nombre y pueden ser asignados a otros usuarios y roles.

Características de los roles:

- Pueden otorgarse a cualquier usuario o rol, pero no a sí mismo y tampoco de forma circular.
- Pueden tener contraseña.
- Su nombre es único en la bd, distinto a cualquier otro nombre de usuario o rol.
- No pertenecen a ningún esquema.
- Simplifican el manejo de privilegios. Los permisos pueden asignarse a un rol y este a los diferentes usuarios.
- Manejo de privilegios dinámico. Si se modifican los privilegios asociados al rol, todos los usuarios que lo posean los adquieren de forma inmediata.

Roles predefinidos:

Es recomendable crear roles específicos en cada bd y asignarles los permisos necesarios, evitando el uso de roles predefinidos.

Creación de Roles

Debe poseerse el privilegio CREATE ROLE. El nombre debe ser diferente a cualquier nombre de rol o usuario existente.

Antes verificamos quienes tienen privilegios para crear roles:

Ahora asignamos a un usuario al rol creado:

SQL> gran fcld to aperpian

La cláusula IDENTIFIED BY indica como debe ser autorizado antes de usarse por un usuario al que se la ha otorgado.

Modificación de roles.

Un rol solo puede modificarse para cambiar su método de autenticación. Debe poseerse el privilegio de sistema ALTER ANY ROLE o haber sido sido otorgado el rol con la opción ADMIN. No se ven afectadas las sesiones en las que el rol está ya activo.

Para que un usuario pueda otorgar un rol debe habérsele concedido con ADMIN OPTION , poseer el privilegio GRANT ANY ROLE , o haberlo creado. El usuario que crea el rol implícitamente lo tiene asignado con ADMIN OPTION .

Roles por defecto.

Un rol por defecto es aquel que automáticamente se activa al conectarse. Con la sentencia ALTER USER se limitan los roles por defecto asignados a un usuario. La cláusula puede sólo indicar roles otorgados directamente al usuario con una sentencia GRANT.

Revocar un rol.

Puede hacerlo cualquier usuario con la opción ADMIN OPTION para un rol, también aquellos usuarios con el privilegio GRANT ANY ROLE (pueden revocar cualquier rol), con PUBLIC se les asigna el rol de todos los usuarios.

REVOKE <rol1>, …<roln> FROM <usuario> | <rol> | PUBLIC, ...

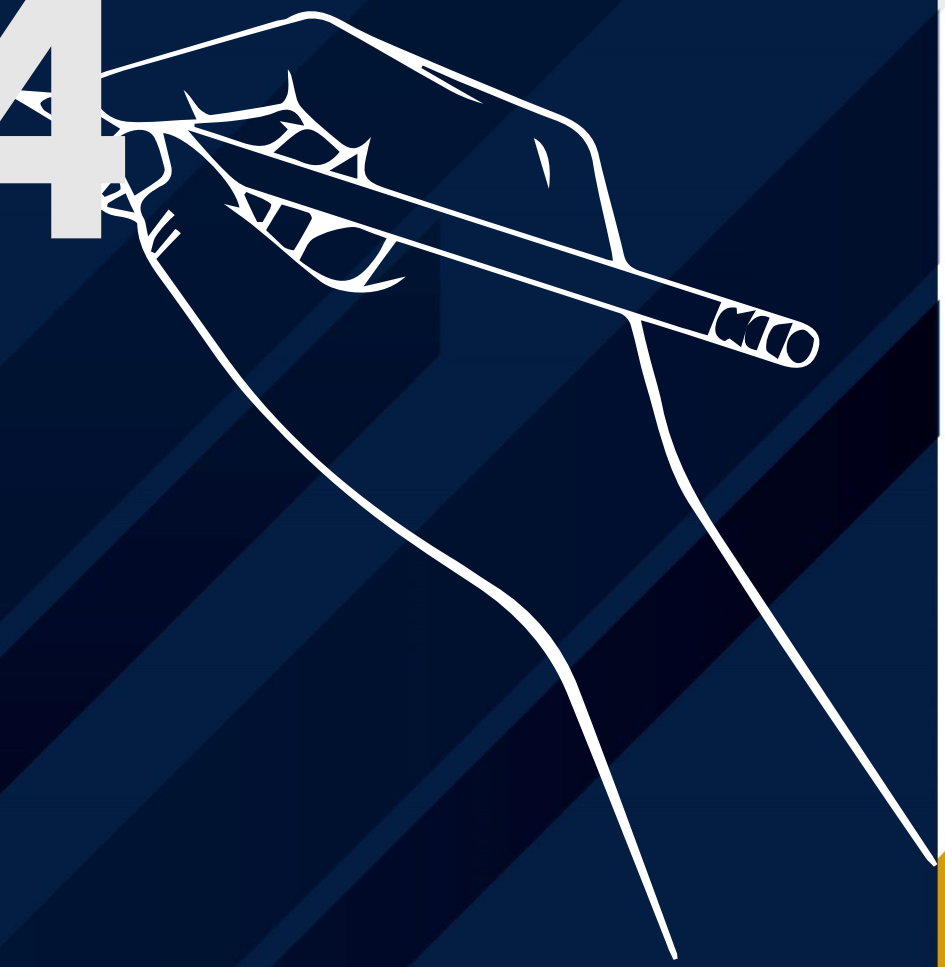
Eliminación de roles.

Debe poseerse el privilegio DROP ANY ROLE o haber ido concedido el rol con ADMIN OPTION.

DROP ROLE <rol>;

Al borrar un rol se asigna de todos los usuarios y roles, y se elimina de la base de datos. Las sesiones en las que el rol está activo no se ven afectadas, pero ninguna otra lo podrá usar.

04



Arquitectura del Data WareHouse

El Data Warehouse es una tecnología para el manejo de la información construido sobre la base de optimizar el uso y análisis de la misma utilizado por las organizaciones para adaptarse a los vertiginosos cambios en los mercados. Su función esencial es ser la base de un sistema de información gerencial, es decir, debe cumplir el rol de integrador de información proveniente de fuentes funcionalmente distintas (Bases Corporativas, Bases propias, de Sistemas Externos, etc.) y brindar una visión integrada de dicha información, especialmente enfocada hacia la toma de decisiones por parte del personal jerárquico de la organización.

Es un sitio donde se almacena de manera integrada toda la información resultante de la operatoria diaria de la organización. Además, se almacenan datos estratégicos y tácticos con el objetivo de obtener información estratégica y táctica que pueden ser de gran ayuda para aplicar sobre los mismos técnicos de análisis de datos encaminadas a obtener información oculta (Data Mining).

Esta información incluye movimientos que modifican el estado del negocio, cualquier interacción que se tenga con los clientes y proveedores, y cualquier dato adicional que ayude a comprender la evolución del negocio.

Esta tecnología ayuda a la organización a responder preguntas esenciales para la toma de decisiones que le permitan obtener ventajas competitivas y mejorar su posición en el mercado en el que operan. Algunas de las preguntas podrían ser:

- Cuál es el perfil de mis clientes?
- Cómo es su comportamiento?
- Cuál es la rentabilidad que me deja?
- Cuál es el riesgo que corro con él?

Es un modelo multidimensional basado en tecnología OLAP, incluyendo variables claves y los indicadores claves para el proceso de toma de decisiones.

Algunas ventajas de la construcción del Data Mart:

- Son más simples de implementar que un

Data Warehouse Pequeños conjuntos de datos y, en consecuencia, menor necesidad de recursos

- Se encuentran más rápidamente las necesidades de las Unidades de Negocio
- Queries más rápidos por menor volumen de datos

Como desventaja se puede decir que, en algunos casos, añaden tiempo al proceso de actualización.

En síntesis, son pequeños Data Warehouse centrados en un tema o un área de negocio específico. En muchos casos, los Data Warehouse comienzan siendo Data Marts con el objetivo de minimizar los Herramienta de soporte para la toma de decisiones. Incorpora reglas de decisión y análisis de datos no predefinidos en las posibilidades de un EIS.

- Sistemas de presentación
- Sistemas Interrogativos
- Sistemas de Simulación
- Sistemas funcionales
- Sistemas Expertos

Diferencias entre OLTP y OLAP

Mientras que las aplicaciones OLTP se caracterizan por estar actualizadas constantemente por varios usuarios a través de transacciones operacionales sobre datos individuales, las aplicaciones OLAP son utilizadas por personal de niveles ejecutivos que requieren datos con alto grado de agregación y desde distintas perspectivas (dimensiones), como ser: totales de venta por región, por producto, por período de tiempo, etc. (Ver figura 3).

OLTP	OLAP
Atomizado	Sumarizado
Datos Históricos	Datos Actuales
Un registro a la vez	Muchos registros a la ve
Orientado a la información operativa	Orientado a la información estratégica
Datos relacionales	Datos Multidimensional
Consultas simples predefinidas	Consultas ad-hoc
Volumen de datos acotados	Grandes volúmenes de datos

Figura #3

Concepto de datos multidimensionales

En el análisis multidimensional, los datos se representan mediante dimensiones como producto, territorio y cliente. En general, las dimensiones se relacionan en jerarquías, por ejemplo, ciudad, estado, región, país y continente. El tiempo es también una dimensión estándar con sus propias jerarquías tales como: día, semana, mes, trimestre y año. (Figura 4).

No es común que, por ejemplo, alguien dentro de la organización se pregunte: "¿cuánto vendí?".

Figura 4 – Estructura multidimensional de los datos

En general, un Gerente de Ventas podría preguntarse: ¿Cuánto vendí del producto "A" en el períodos "X" en la región "Y"? (Figura 5).

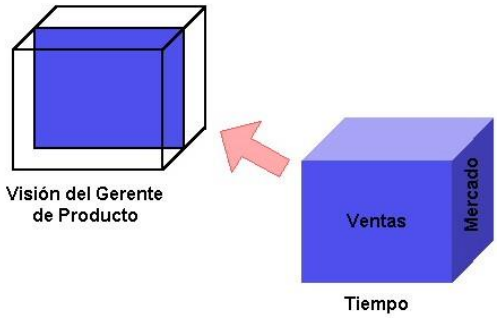


Figura 5 – Estructura multidimensional de los datos

En general, un Gerente de Ventas podría

preguntarse: ¿Cuánto vendí del producto "A" en el períodos "X" en la región "Y"? (Figura 5).

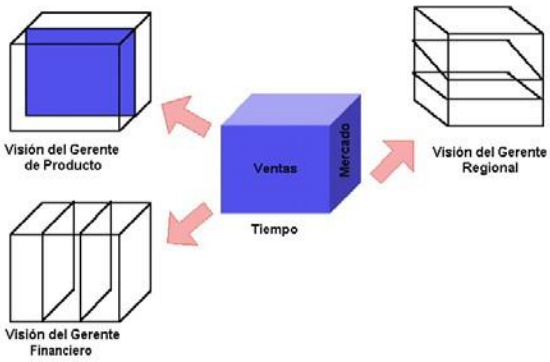


Figura 6 – Visión de los gerentes financiero y regional

Implementación de un Data Warehouse

La estructura adoptada para el almacén de datos se debe realizar de tal modo que satisfaga las necesidades de la empresa, dicha elección es clave en la efectividad del Data Warehouse. Existen tres formas básicas de estructura del almacén:

Data Warehouse central

La implementación consta de un solo nivel con un solo almacén que soporta los requerimientos de información de toda la empresa.

Data Warehouse distribuido

Es una estructura de un solo nivel que se particiona para distribuirlo a nivel departamental.

Data Warehouse de dos niveles

Es una combinación de los anteriores que soporta requerimientos de información tanto a nivel empresarial como departamental.

Costos del Data Warehouse

Uno de los puntos más importantes a tener en cuenta en el momento de decidir implementar un Data Warehouse es el costo que trae aparejado. A grandes rasgos los costos asociados a un proyecto Data Warehouse son el Costo de Construcción y el costo de Mantenimiento y Operación una vez construido.

Costo de Construcción

Es similar a al Costo de Construcción de cualquier sistema de Tecnología. Se pueden clasificar en tres tipos:

RECURSOS HUMANOS: Es necesario contar con conocimiento sobre el perfil y cualidades del personal ya que el desarrollo de esta tecnología requiere de la participación tanto del personal técnico como de los especialistas de negocios, estos dos grupos trabajarán juntos durante todo el desarrollo del Data Warehouse.

TIEMPO: Además de los tiempos de construcción y entrega del Data Warehouse, se debe tener en cuenta los tiempos de planificación del proyecto y de definición de la Arquitectura.

TECNOLOGÍA: El costo de la nueva tecnología introducida por el Data Warehouse se debe considerar solo como el costo inicial de la implementación.

Costo de Operación y Mantenimiento

Es necesario, una vez que se ha finalizado la construcción y se ha entregado el producto se debe dar soporte que es una fuente continua de costos. Los costos de operación se dividen en:

Costo de Evolución

Es necesario realizar ajustes continuos del Data Warehouse a través del tiempo, muchas veces estos cambios se deben al aprendizaje mediante el uso.

Costo de Crecimiento

Incrementos de volúmenes de datos, de cantidad de usuarios accediendo al Data Warehouse desembocará en un aumento en los recursos necesarios para que los tiempos de respuesta y recuperación de datos, principalmente, sigan siendo óptimos.

Costo producido por cambios

El Data Warehouse necesita soportar los cambios en el origen de datos que utiliza como así también soportar los cambios de la información que produce. Por ejemplo, si el cambio se produce en el ambiente empresarial, seguramente, cambiarán las necesidades de información de los usuarios serán necesarios, entonces, cambios en las Aplicaciones DSS y EIS. Si por el contrario cambio viene dado por el sector tecnológico y éste afecta el modo de almacenamiento de los datos, implicaría ajustes en los procesos de Extracción, Soporte y Carga para adaptarse a las variaciones.

Impactos de implementación del Data Warehouse

El éxito del Data Warehouse no está en la construcción sino en utilizarlo para mejorar los procesos empresariales, operacionales y de toma de decisiones, para que esto suceda se deben tener en cuenta los impactos producidos en los siguientes ámbitos:

Impacto en la gente

La construcción requiere de la participación activa de quienes utilizarán el Data Warehouse, depende tanto de la realidad de la empresa como de las condiciones que existan en ese momento, las cuales determinarán cuál será su contenido.

El Data Warehouse provee los datos que posibilitará a los usuarios a acceder a su propia información en el momento que la necesitan. Esta posibilidad para entregar información presenta varias implicancias:

Los usuarios deberán adquirir nuevas destrezas.

Se eliminará los largos tiempos de análisis y programación para obtener información. Como la información estará lista para ser utilizada, probablemente, aumenten las expectativas. Pueden existir nuevas oportunidades en la comunidad empresarial para los especialistas de información. Se reducirá hasta casi

eliminarse la gran cantidad de reportes en papel.

La madurez del Data Warehouse dependerá del uso activo y retroalimentación de sus usuarios.

Impactos en los procesos empresariales y de toma de decisiones

Mejora del proceso de toma de decisiones por medio de la disponibilidad de la información. Las decisiones se toman más rápidamente por gente más informada.

Los procesos empresariales pueden ser optimizados, se elimina el tiempo de espera de información que, generalmente, es incorrecta o no se encuentra.

Se reducen los costos de los procesos y muchas veces se aclaran sus conexiones y dependencias, aumentando así la eficiencia en dichos procesos.

El Data Mining y su relación con el DataWarehouse

Las técnicas de Data Mining son el resultado de un largo proceso de investigación y desarrollo de productos orientados al almacenamiento, extracción análisis de datos. Esta evolución comenzó cuando los datos de negocios fueron almacenados por primera vez en computadoras, y continuó con mejoras en el acceso a los datos, y más recientemente con tecnologías generadas para permitir a los usuarios navegar a través de los datos en tiempo real. Data Mining está soportado por las siguientes tecnologías:

- Soportes de almacenamiento masivo de datos
- Potentes computadoras con multiprocesadores
- Data Warehouse
- Algoritmos de Data Mining

Data Mining es la extracción de información oculta y predecible de grandes bases de datos.

Un sistema Data Mining es una tecnología de soporte para usuario final cuyo objetivo es extraer conocimiento útil y utilizable a partir de la información contenida en las bases de datos de las empresas.

Las herramientas de Data Mining sirven para predecir tendencias y comportamientos, de esta manera permiten a las organizaciones tomar decisiones proactivas para adaptarse rápidamente a los cambios del mercado obteniendo así ventajas competitivas.

Las herramientas de Data Mining pueden responder a preguntas de negocios que tradicionalmente consumen demasiado tiempo para poder ser resueltas por consultas en un sistema tradicional de soporte operacional. La potencialidad de estas herramientas reside en la capacidad de explorar las bases de datos en busca de patrones ocultos, encontrando

información predecible que para un experto sería casi imposible debido al gran volumen de información.

Una vez que las herramientas de Data Mining fueron implementadas en computadoras cliente servidor de alto performance o de procesamiento paralelo, pueden analizar bases de datos masivas para brindar respuesta a preguntas tales como, "¿Cuáles clientes tienen más probabilidad de responder al próximo mailing promocional, y por qué? y presentar los resultados en formas de tablas, con gráficos, reportes, texto, hipertexto, etc.

El origen de la información que utilizan los algoritmos de Data Mining, por lo general, son datos históricos que se encuentran almacenados en un Data Warehouse. El partir de un Data Warehouse simplifica la etapa previa a la etapa de preparación de los datos ya que se construye en base a la integración de fuentes de datos múltiples y heterogéneas Bases de Datos relacionales.

La mejor forma de aplicar las técnicas de Data Mining es que éstas se encuentren totalmente integradas con el Data Warehouse así como también con herramientas flexibles e interactivas para el análisis de negocios. Varias herramientas de Data Mining actualmente operan fuera del Data Warehouse, requiriendo pasos extra para extraer, importar y analizar los datos. Además la integración con el Data Warehouse permite que ni bien los cambios originados en las bases de datos operacionales son replicados al Data Warehouse pueden ser analizados directamente y monitoreados mediante las técnicas de Data Mining.

Bilbiografía:

Data Mining y Data Warehousing en www.kdnuggets.com
Data Warehouse Terminology, 2003. En http://www.credata.com/research/terminology.html
On Line Analitical Processing en http://altaplana.com/olap/
http://lucas.hispalinux.es/Postgresqles/web/navegable/programmer/triggers.html
http://dev.mysql.com/doc/refman/5.0/es/server-side-scripts.html
http://dev.mysql.com/doc/refman/5.0/es/instance-manager.html
http://dev.mysql.com/doc/refman/5.0/es/configuring-mysql.html
http://dev.mysql.com/doc/refman/5.0/es/server-shutdown.html
http://dev.mysql.com/doc/refman/5.0/es/security.html
http://dev.mysql.com/doc/refman/5.0/es/mysql.html
http://dev.mysql.com/doc/refman/5.0/es/mysqladmin.html
http://dev.mysql.com/doc/refman/5.0/es/features.html
http://dev.mysql.com/doc/refman/5.0/es/log-files.html

INSTITUTO SUPERIOR TECNOLÓGICO PELILEO

1994

INSTITUTO SUPERIOR TECNOLÓGICO PELILEO

ISBN: 978-9942-686-51-0

Educación gratuita y de calidad