INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

DESARROLLO DE SOFTWARE

Instrumento para facilitar el proceso de enseñanza-aprendizaje de la asignatura

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA
20240001

BASE DE DATOS

Período académico: Primero

Abril - 2024

ING. JUAN DIEGO ROJAS ESCANDÓN, MG.

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA – BASE DE DATOS

INSTITUTO SUPERIOR TECNOLÓGICO TENA

Carrera de Tecnología Superior en Desarrollo de Software

ISTT DSW Segunda Edición – Tena, abril 2024

SIN ISBN

Instituto Superior Tecnológico Tena
Km. 1 1/2 Vía Tena - Archidona
Tena, Ecuador

AUTOR - REDACCIÓN Y FORMULACIÓN DE CONTENIDOS
Ing. Juan Diego Rojas E., Mg.

Profesor del Instituto Superior Tecnológico Tena

REVISIÓN DE PARES

Lcdo. Segundo Calisto Rochina Chileno
Mg. Alvaro Santiago Toalombo Diaz
Mg. Henry Fabian Chango Chango
Ing. Agustín Gonzalo Guanipatin Ramirez

Comisión de revisión técnica de guías de estudio del Instituto Superior Tecnológico Tena

APROBACIÓN

Mg. Danilo Alexander Zamora Núñez
Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Impreso y hecho en Ecuador.

TABLA DE CONTENIDO

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA

DATOS GENERALES DE LA ASIGNATURA							
Carrera	Tecnología en Desarrollo de Software			**Nombre asignatura**		Base de Datos	
Modalidad	Presencial			**Campo de Formación**		Adaptación e Innovación Tecnológica	
Jornada	Matutina/Nocturna			**Unidad de Organización Curricular**		Profesional	
Período académico	Primero			**Código de la asignatura**		DSW-105	
Distribución de horas en las actividades de aprendizaje				**N° Total de horas de la asignatura**		192	
N° de horas Docencia	64	**N° de horas Aprendizaje Práctico Experimental**				**N° de horas Autónomo**	48
		En contacto con docente	32	**Autónomo**	48		

PRERREQUISITOS Y CORREQUISITOS			
Prerrequisitos de la asignatura		**Correquisitos de la asignatura**	
Asignatura	**Código**	**Asignatura**	**Código**

DESCRIPCIÓN DE LA ASIGNATURA

Las bases de datos son el principio fundamental del funcionamiento de un sistema informático, siendo una parte clave del aprendizaje para el Tecnólogo Superior en Desarrollo de Software. La asignatura de Base de datos es parte de la Unidad de Organización Curricular Profesional y en ella se estudia el diseño conceptual, los modelos relacionales, conceptual lógico y fisco conjuntamente con la práctica de laboratorio a través del desarrollo y ejecución de sentencias utilizando herramientas case que facilitan el aprendizaje del estudiante.

OBJETIVO GENERAL

Generar bases de datos a través de un modelo relacional utilizando herramientas CASE que permitan al estudiante Tecnólogo Superior en Desarrollo de Software incorporar almacenes de datos para sistemas de informáticos.

CONTRIBUCIÓN DE LOS RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA AL PERFIL DE EGRESO DE LA CARRERA

Resultados de aprendizaje de la asignatura	Resultados de aprendizaje del perfil de egreso de la carrera	Contribución (alta – media – baja)
Analiza los problemas planeados y genera modelos relacionales para manejo del proceso.	Brinda asistencia técnica en el diseño, modelamiento e ilustración del proceso de base de datos.	Media
Desarrolla scripts de creación de base de datos.	Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.	Media
Describe el proceso mediante el esquema de administración de base de datos.	Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.	Baja
Desarrolla sentencias SQL para manejo y administración de la información.	Brinda asistencia técnica en el diseño de bases de datos mediante procesos de control y seguimiento de las	Media

	operaciones para el manejo adecuado de la información.	
Relaciona la base de datos con el desarrollo del software.	Brinda asistencia técnica en el diseño de bases de datos mediante procesos de control y seguimiento de las operaciones para el manejo adecuado de la información.	Alta
Brinda asistencia técnica en el diseño, modelamiento e ilustración del proceso de base de datos.	Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.	Media
Brinda asistencia técnica en el diseño de bases de datos mediante procesos de control y seguimiento de las operaciones para el manejo adecuado de la información.	Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.	Media

CONTENIDOS DE LA ASIGNATURA (descripción mínima de contenidos de la asignatura)

Unidad 1: Introducción a base de datos
1.1 Conceptos Generales
1.2 Entidades y Atributos
1.3 Relaciones y Cardinalidades

Unidad 2: Modelamiento de base de datos
2.1 Tipos de Modelados
2.2 Modelo Entidad – Relación
2.3 Claves o Identificadores
2.4 Integridad Relacional

Unidad 3: Optimización e implementación de base de datos
3.1 Normalización
3.2 Modelo Conceptual, Lógico y Físico.
3.3 Gestores de Bases de datos.
3.4 Sentencias DML y DDL

Unidad 4: Bases de datos no SQL
4.1 Introducción.
4.2 Características Generales
4.3 Tipos de Bases de datos No SQL
4.4 Principales diferencias con las bases de datos SQL
4.5 Ventajas y Desventajas
4.6 Modelización de bases de datos No SQL
4.7 Bases de datos No SQL y su integración en aplicaciones informáticas

ESTRATEGIAS METODOLÓGICAS Y RECURSOS DIDÁCTICOS

ESTRATEGIAS METODOLÓGICAS	HABILIDADES BLANDAS	FINALIDAD
Activas para la enseñanza y aprendizaje	**Valores vinculados a la autonomía del sujeto**: confianza, crítica y autocrítica, honestidad, integridad	• Generar confianza/ Promover el pensamiento crítico. • Permite a los estudiantes cumplir un rol activo dentro de su formación. • Construye una sociedad participante.

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona ISTT

Aprendizaje y trabajo cooperativo	**Valores elementales de convivencia y civilidad:** crítica y autocrítica, tolerancia, empatía, respeto, justicia, lealtad, paciencia	• Promover un ambiente de colaboración/ trabajo en equipo/ Saber escuchar/Promover el pensamiento crítico/ fomentar el liderazgo/ adaptabilidad. • Mantener una comunicación abierta con el equipo/ tolerancia a los errores, aceptar y aprender de las críticas. • Fomentar el sentido de pertenencia
Aprendizaje individual	**Valores vinculados a la autonomía del sujeto**: responsabilidad, honestidad, integridad, efectividad, autonomía	• Facilitar la asimilación del contenido por parte del estudiante/ Plantear preguntas para promover la comunicación efectiva /Promover el pensamiento crítico • Lectura comprensiva para fijar contenidos/ Promover el pensamiento crítico

RECURSOS DIDÁCTICOS	
MATERIALES CONVENCIONALES	*Material impreso: libros, folletos, fotocopias, periódicos, etc.*
	Tableros didácticos: pizarra
MATERIALES AUDIOVISUALES	*Imágenes fijas proyectables (fotos): diapositivas y fotografías.*
	Materiales audiovisuales (vídeo): vídeos
NUEVAS TECNOLOGÍAS	*Programas informáticos: procesador de palabras, hojas de cálculo, presentaciones*
	Servicios telemáticos: páginas web, plataforma EVA, correo electrónico, google drive. *Aplicativos WEB/móviles: XAMPP, PHPMYADMIN, MYSQLWORKBENCH*

BIBLIOGRAFÍA

Bibliografía Básica de la Asignatura:	**Físico**	**Digital**
Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Programación de base de datos relacionales.* (Quinta Edición). McGraw-hill/Interamericana. España. **ISBN:** 978-84-481-5671-8, **Número de inventario en biblioteca:** ISTT-DS-0097	X	
Martínez López, F. J., & Gallegos Ruiz, A. (2017). *Programación de base de datos relacionales*. (Primera Edición). Ra-ma. Colombia. **ISBN:** 978-958-762-684-1, **Número de inventario en biblioteca:** ISTT-DS-0027	X	
Bibliografía de consulta de la Asignatura:	**Físico**	**Digital**
Castaño, M., Piattini, M., & Esperanza, M. (2000). *Diseño de base de datos relacionales*. (Primera Edición). Alfaomega Grupo Editorial S.A. México. **ISBN:** 970-15-0526-3, **Número de inventario en biblioteca:** ISTT-DS-0031	X	
Pérez Marqués, M. (2016). *Administración básica de base de datos con Oracle 12 c SQL Prácticas y Ejercicios.* (Primera Edición). Alfaomega. Colombia. **ISBN:** 978-958-778-202-8, **Número de inventario en biblioteca:** ISTT-DS-0057	X	

DESCRIPTIVA DE LAS COMPETENCIAS DE LA GUÍA DE BASE DE DATOS

La guía de estudio está orientada al desarrollo de competencias específicas, generales y transversales que permitan al estudiante adquirir los conocimientos y habilidades necesarias para diseñar, implementar y optimizar bases de datos. Estas competencias se han estructurado con base en las unidades temáticas y responden a las demandas actuales del campo profesional.

Competencias Específicas

Unidad 1: Introducción a Bases de Datos

- Identificar los conceptos fundamentales, niveles de abstracción y estructuras de las bases de datos para comprender su funcionamiento e importancia en los sistemas de información.
- Distinguir los diferentes tipos de bases de datos y su aplicabilidad en diversos contextos profesionales.

Unidad 2: Modelamiento de Bases de Datos

- Diseñar modelos conceptuales, lógicos y físicos de bases de datos, aplicando principios de normalización y herramientas de modelado, para responder a las necesidades específicas de sistemas de información.
- Analizar reglas de negocio y su impacto en el modelado de bases de datos.

Unidad 3: Optimización en la Implementación de Bases de Datos

- Aplicar técnicas de optimización para mejorar el rendimiento de bases de datos, reduciendo la redundancia y asegurando la integridad de los datos.
- Evaluar y solucionar problemas relacionados con la consistencia y eficiencia de las bases de datos implementadas.

Unidad 4: Bases de Datos NoSQL

- Diseñar e implementar bases de datos NoSQL adaptadas a necesidades específicas, considerando escalabilidad y flexibilidad para manejar grandes volúmenes de datos.
- Analizar las ventajas y limitaciones de bases de datos relacionales frente a NoSQL en diversos escenarios.

Competencias Generales

- Aplicar herramientas y metodologías tecnológicas para resolver problemas complejos relacionados con la gestión y administración de bases de datos.
- Integrar conocimientos teóricos y prácticos para diseñar soluciones informáticas adaptadas a las necesidades del entorno profesional.
- Trabajar de forma colaborativa y efectiva en proyectos multidisciplinarios relacionados con sistemas de información.

Competencias Transversales

- Desarrollar habilidades de análisis crítico para identificar problemas y plantear soluciones efectivas en el ámbito de bases de datos.
- Adoptar principios éticos y legales en el manejo y almacenamiento de datos, respetando la privacidad y seguridad de la información.
- Utilizar herramientas digitales actuales para gestionar procesos de diseño, implementación y optimización de bases de datos.
- Fomentar la capacidad de autoaprendizaje y actualización constante en tecnologías emergentes relacionadas con bases de datos.

.

GUÍA DE ESTUDIO DE BASE DE DATOS

UNIDAD 1

INTRODUCCIÓN A LAS BASES DE DATOS

Introducción.
Conceptos generales.
Entidades y Atributos.
Relaciones y Cardinalidades.

Resultado de Aprendizaje

El estudiante será capaz de diferenciar los tipos de bases de datos, sus componentes principales y niveles de abstracción, demostrando comprensión sobre su funcionalidad y aplicabilidad en sistemas de información mediante ejemplos prácticos y comparativos

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona ISTT

DIAGRAMA DE APRENDIZAJE

SÍNTESIS

La Unidad 1 introduce los fundamentos esenciales de las bases de datos, enfatizando su importancia como sistemas organizados para almacenar, gestionar y recuperar información de manera eficiente. Se abordan conceptos generales como las bases de datos relacionales, compuestas por tablas que organizan datos en registros y campos, gestionadas mediante Sistemas de Gestión de Bases de Datos (SGBD). En el modelado, se destacan las entidades, que representan objetos o conceptos del mundo real, y sus atributos, que describen sus características. Además, se exploran las relaciones entre entidades, que definen cómo interactúan entre sí, y las cardinalidades, que especifican el número de asociaciones posibles entre las instancias de diferentes entidades, como uno a uno (1:1), uno a muchos (1) o muchos a muchos (M).

Introducción a las Bases de Datos.

Un sistema gestor de base de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programa para acceder a dicho datos. La colección de datos, normalmente denominada base de datos, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto practica como eficiente. Los sistemas de base de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información almacenada, a pesar de las caídas del sistema o de los intentos de accesos no autorizados. Si los datos van a ser compartidos entre diferentes usuarios, el sistema debe evitar posibles resultados anómalos. (Abraham Silberschatz - Hendry F. korth - S. Sudarshan, 2007)

Una **base de datos** (cuya abreviatura es BD) es una entidad en la cual se pueden almacenar datos de manera estructurada, con la menor redundancia posible. Diferentes programas y diferentes usuarios deben poder utilizar estos datos. Por lo tanto, el concepto de base de datos generalmente está relacionado con el de red, ya que se debe poder compartir esta información. Generalmente se habla de un "Sistema de información" para designar a la estructura global que incluye todos los mecanismos para compartir datos. (Carlos Villagómez. , 2017)

De los sistemas de ficheros a las Bases de Datos.

Según (Morrow, 2015). Menciona que:
Los sistema de ficheros, en el SO coexisten distintas visiones del sistema de ficheros. La de más alto nivel es la que se expresa mediante el árbol de ficheros y directorios (que en Unix tiene una sola raiz, /) y la de más bajo nivel concreta cómo se guardan la información fisicamente en el dispositivo que contiene el sistema de ficheros.

Tipos de sistemas de ficheros:
La función básica de un sistema de ficheros (en inglés file system) es preservar la información en un dispositivo de almacenamiento como un disco duro, o un DVD. Esta tarea se puede realizar de diferentes modos en función de la información que se va a guardar, las características del medio y el tipo de accesos que se van a realizar.
No obstante, existen sistemas de ficheros especializados que pueden tener otras funciones, como servir de interfaz entre el administrador y el núcleo del sistema informático, u otras funciones.

Tipos de sistemas de archivos
De disco: Tienen por función guardar ficheros en un dispositivo de almacenamiento. Algunos FS pueden soportar Journaling, una técnica que utiliza un diario para guardar

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona ISTT

los datos necesarios para restablecer un estado consistente del sistema de archivos tras un fallo. Algunos FS de disco son: EXT, FAT, ISO9660, NTFS, JFS, ReiserFS y XFS.

De red: Permiten compartir ficheros entre diferentes dispositivos conectados a una red. Algunos FS de red son: CIFS y NFS.

De base de datos: En lugar de guardar los ficheros de forma jerárquica, se utiliza una base de datos para guardar los ficheros indexados por su meta información (nombre, permisos, tipo de fichero, etc..). Es posible realizar búsquedas de ficheros en SQL o un lenguaje natural.
Algunos FS de base de datos son: BFS, Gnome VFS y WinFS.

De propósito específico: Sistemas de ficheros que, por ejemplo, tienen por función mostrar ficheros de dispositivo (Udev), permitir que el núcleo muestre los procesos que controla (procfs) o permitir que núcleo utilice un espacio de almacenamiento secundario para la gestión de la memoria virtual (swap).

Administración de sistemas de ficheros

Desde el punto de vista del administrador, las operaciones a realizar con los sistemas de ficheros incluyen:

- Creación
- Montado/Desmontado
- Copias de seguridad
- Comprobaciones/reparaciones

Creación de un sistema de ficheros

Antes de utilizar un dispositivo de almacenamiento debe crearse un sistema de ficheros en su interior. Por ejemplo, un disco duro puede dividirse en 2 particiones, cada partición es un dispositivo de almacenamiento, en cada una de ellas se debe crear un sistema de archivos antes de poder escribir/ficheros.

En GNU/Linux se utiliza la herramienta mkfs para crear sistemas de ficheros. mkfs es símplemente una interfaz para llamar a la herramienta encargada de crear el tipo de sistema de ficheros especificado (mkfs.bfs, mkfs.ext2, mkfs.ext3, mkfs.minix, mkfs.msdos, mkfs.vfat, mkfs.xfs,...).
Para cada sistema de ficheros se pueden especificar diferentes opciones durante su creación, las páginas de manual del comando mkfs.* correspondiente enumeran los detalles.

Ventajas e inconvenientes de las Bases de Datos.

Según (TecnoMagazine, 2018). Menciona:

Ventajas:

- ✓ Almacenan grandes cantidades de información.
  Esto es muy útil para las grandes compañías que manejan grandes volúmenes de información.

- ✓ Compartir la información.
  Los usuarios de distintas oficinas pueden compartir datos e información que son de gran importancia para sus departamentos o funciones.

- ✓ Acceso rápido a la información.
  Esta es una gran ventaja, ya que la información siempre estará disponible para los usuarios.

- ✓ Eliminación de información repetida o redundante.
  Los usuarios tendrán la certeza de que la información que están solicitando no está repetida o es redundante.

- ✓ Aumento en la productividad.
  Como la información está disponible y es coherente, los usuarios podrán aumentar su rendimiento al saber que cuentan con una información fiel.

- ✓ Reducción del espacio de almacenamiento.
  Al tener la información en medios electrónicos, se reduce considerablemente el espacio para almacenar de manera tradicional.

- ✓ Mejora la seguridad de la información.
  Existe un acceso reducido para los usuarios, de tal manera que cierta información podrá ser controlada por los administradores de la base de datos.
- ✓ Mejor mantenimiento.
  Al estar la información en medios electrónicos, el darle mantenimiento es mucho más fácil ya que se cuentan con herramientas para este mantenimiento.

Desventajas:

- ✓ Tamaño.
  Entre más grande sea la base de datos, se requiere mayor capacidad e disco duro y más memoria ram para que pueda funcionar adecuadamente.

- ✓ Costo.
  El hardware y software para el correcto funcionamiento de una base de datos es costoso.

- ✓ Actualización.
  Es necesario mantener actualizados a los usuarios ya que las tecnologías van cambiando constantemente.

- ✓ Vulnerabilidad a los fallos.
  Esta es una gran desventaja ya que la base de datos está expuesta a fallos que no están en las manos de los usuarios, como una descarga eléctrica.

Concepto de Base de Datos.

Según (Diego Rafael Llanos Ferraris, 2007) dice que:
La aparición del concepto de base de datos se produce a comienzos de la década de 1960, concretamente en un simposio que se celebró en Santa Mónica (California, EE.UU.) y cuyo título contenía la expresión Data Base.
Una base de datos es un conjunto, colección o depósito de datos almacenados en un soporte informático de acceso directo. Los datos deben estar relacionados y estructurados de acuerdo con un modelo capaz de recoger el contenido semántico de los datos almacenados: Dada la importancia que tiene en el mundo real las relaciones entre los datos, es imprescindible que la base de datos sea capaz de almacenar estas interrelaciones. Esta es una de las principales diferencias respecto a los ficheros tradicionales, en los que no se almacenan dichas relaciones. Además, las bases de datos modernas también almacenadas las restricciones semánticas que están presentes en los datos y a las que se les está concediendo una importancia creciente.

Una base de datos debe cumplir una serie de requisitos:

- En las bases de datos no debe existir redundancia lógica de datos, aunque es admisible cierta redundancia física por motivos de eficiencia. Desde el punto de vista de usuario, los datos solo están almacenados una vez, aunque el sistema los puede replicar para facilitar su acceso de una manera más eficiente. Por tanto, un usuarios actualizará un dato de forma única y si existiera una redundancia física de dato, el sistema se responsabilizará de efectuar todos aquellos cambios en los lugares en los que dicho dato estuviera replicado.

- Las bases datos han de dar soporte a múltiples usuarios y a diferentes aplicaciones simultaneas.

- En las bases de datos debe existir una independencia tanto física como lógica entre datos y su proceso.

- la definición y descripción del conjunto de datos contenidos en la base deben ser únicas y estar integradas con los mismos datos. En los ficheros, los datos se encuentran almacenados en soporte magnéticos y su descripción forma parte de los programas. En las bases de datos, la descripción y la definición de los datos (metadatos) se almacenan junto con los datos, de manera que los datos almacenados están autodocumentados y cualquier cambio que se produzca en dicha documentación se ha de reflejar en la base de datos.

- La base de datos debe asegurar la integridad, seguridad y confidencialidad de sus datos cuando estos se actualizan y recuperan.

Tal y como hemos visto, el termino base de datos hace referencia a una colección especifica de datos, aunque es habitual usarlo, de manera errónea, como sinónimo del software que gestiona dicha colección de datos.

Niveles de Abstracción en una Base de Datos.

Según (Abraham Silberschatz - Hendry F. korth - S. Sudarshan, 2007) menciona que: Par a que el sistema sea útil debe recuperar los datos eficientemente. La necesidad de eficiencia ha llevado a los diseñadores a usar estructuras de datos complejas para la presentación de los datos en la base de datos. Dado que mucho de los usuarios de sistemas de base de datos no tienen formación en informática, los desarrolladores ocultan esa complejidad a los usuarios mediante varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

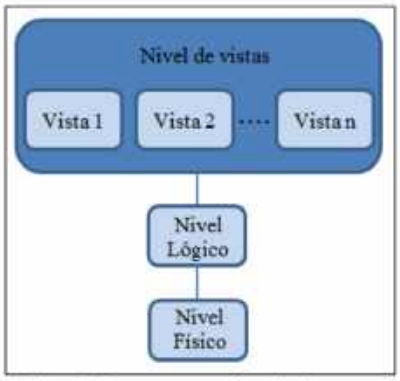


Fig. 1 *Los tres niveles de abstracción de datos.*

Nivel físico: El nivel más bajo de abstracción describe como se almacena realmente los datos. El nivel físico describe en detalle las estructuras de datos complejas de bajo nivel.

Nivel Lógico: El nivel inmediatamente superior de abstracción describe que datos se almacenan en la base de datos y que relaciones existen entre esos datos. El nivel lógico, por lo tanto, describe toda la base de datos en términos de un número pequeño de estructuras realmente simples. Aunque la implementación de esas estructuras simples en el nivel lógico puede involucrar estructuras complejas de nivel físico, los usuarios de nivel lógico no necesitan preocuparse de esta complejidad. Los administradores de base de datos, que deben decidir la información que se guarda en la base de datos, usan el nivel de abstracción lógico.

Nivel de vistas: El nivel más elevado de abstracción solo describe parte de la base de datos. Aunque el nivel lógico usa estructuras más simples, queda algo de complejidad debido a la variedad de información almacenada en las grandes bases de datos. Muchos usuarios de sistema de base de datos no necesitan toda esta información; en su lugar

solo necesitan tener acceso a una parte de la base de datos. El nivel de abstracción de vista existe para simplificar su interacción con el sistema. El sistema puede proporcionar muchas vistas para la misma base de datos.

Una analogía con el concepto de tipos de datos en lenguajes de programación puede clarificar la diferencia entre los niveles de abstracción la mayoría de los lenguajes de programación de alto nivel soportan el concepto de tipo estructurado. Por ejemplo, en los lenguajes tipo Pascal se pueden declarar registros de la manea siguiente:

```
type cliente = record
                 id_cliente: string;
                 nombre_cliente: string;
                 calle_cliente. string;
                 ciudad_cliente: string;
               end;
```

Este código define un nuevo tipo de registro denominado *cliente* con cuatro campos. Cada campo tiene un nombre y un tipo asociados. Una entidad bancaria puede tener varios tipos registros, incluidos:

- *Cuenta,* con los campos *número_cuenta* y *saldo*
- *empleado,* con los campos *nombre_empleado* y *sueldo.*

En el nivel físico, los registros *cliente, cuenta* o *empleado* se pueden describir como bloques de posiciones consecutivas de almacenamiento (por ejemplo, palabras o bytes). El compilador oculta este nivel de detalle a los programadores. De manera parecida, el sistema de base de datos oculta muchos de los detalles de almacenamiento de los niveles inferiores a los programadores de base de datos. Los administradores de base de datos, por otro lado, pueden ser conscientes de ciertos detalles de la organización física de los datos.

En el nivel lógico cada registro de este tipo se describe mediante una definición de tipo, como en el fragmento de código anterior, y también se define la relación entre estos tipos de registros. Los programadores que usan un lenguaje de programación trabajan en este nivel de abstracción. De manera parecida, los administradores de bases de datos suelen trabajar ente nivel de abstracción.

Finalmente, en el nivel de vistas, los usuarios de computadoras ven un conjunto de programas de aplicación que ocultan los detalles de los tipos de datos. De manera parecida, en el nivel de vistas se definen varias vistas de la base de datos y los usuarios de la base de datos pueden verlas. Además de ocultar los detalles de nivel lógico de la base de datos, las vistas también proporcionan un mecanismo de seguridad para evitar que los usuarios tengan acceso a ciertas partes de la base de datos. Por ejemplo, *los cajeros de un banco solo ven la parte de la base de datos que contiene información de las cuentas de los clientes; no pueden tener acceso a la información referente a los sueldos de los empleados.*

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona ISTT

Diseño conceptual.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. Esas grandes cantidades de información no existen aisladas. Forman parte del funcionamiento de alguna empresa, cuyo producto final puede que sea la información obtenida de la base de datos o algún dispositivo o servicio para el que la base de datos solo desempeña un papel secundario. El diseño de base de datos implica principalmente el diseño del esquema de las bases de datos. el diseño de un entorno completo de aplicaciones para la base de datos que satisfaga las necesidades de la empresa que se está modelando exige prestar atención a un conjunto de aspectos más amplio. Este texto se centrara inicialmente en la escritura de las consultas a la base de datos y en el diseño de los esquemas de las bases de datos. (Abraham Silberschatz - Hendry F. korth - S. Sudarshan, 2007)

Entidades y Atributos.

Entidades:

Según (Adoracion de Miguel - Mario Piattini - Esperanza Marcos, 2000). Afirma que:
Se puede definir una entidad como cualquier objeto (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos. HALL (1976) la define como "algo con realidad objetiva que existe o puede ser pensado". Según ANSI (1977), es "una persona, lugar, cosas, conceptual o suceso, real o abstracto, de interés para la empresa". La estructura genérica que describe un conjunto de entidades aplicando la abstracción de clasificación se denomina tipo de entidad, mientras que entidades cada uno de los ejemplares de ese tipo de entidad; por tanto, el tipo de entidades es el resultado de la clasificación de un conjunto de entidades. Así, CURSO es un tipo de entidad que describe las características comunes de un conjunto de curso; un ejemplar del tipo de entidad CURSO será, por ejemplo, "Diseño de Base de Dato Relacionales" y otro "Introducción a los Sistemas de Base de Datos"

Fig. 2 *Representación de Entidad.*

Atributo:

Cada una de las propiedades o características que tienen un tipo de entidad o un tipo de interrelación se denomina ATRIBUTO; los atributos toman valores de uno o varios dominios. Por tanto, podemos decir que el atributo le da una determinada interpretación al dominio (o a los dominios) en el contexto de un tipo de entidad o de un tipo de interrelación (Marcos., 2000).
Un atributo puede ser identificado cuando su nombre esta entro o alado de un circulo o óvalo.

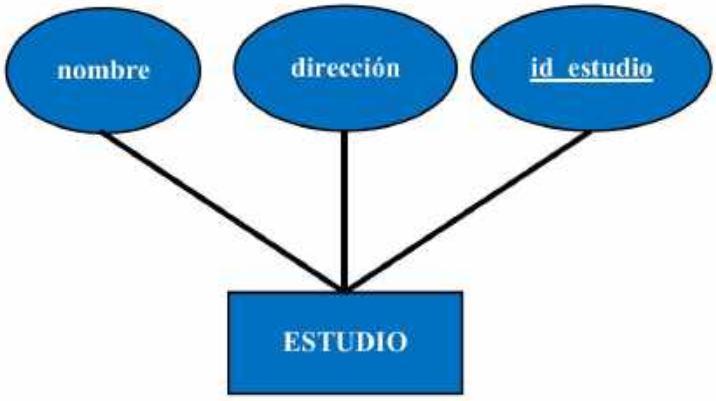


Fig. 3 *Representación de Entidad y Atributo.*

Relaciones y carnalidades.

Según (Pedro Gutiérrez, 2013) indica que:

Podemos encontrar distintos tipos de relaciones según como participen en ellas las entidades. Es decir, en el caso anterior cada empleado puede tener un cargo, pero un mismo cargo lo pueden compartir varios empleados.

Esto complementa a las representaciones de las relaciones, mediante un intervalo en cada extremo de la relación que especifica cuantos *objetos* o *cosas* (de cada entidad) pueden intervenir en esa relación.

Uno a uno: Una entidad se relaciona únicamente con otra y viceversa. Por ejemplo, si tuviésemos una entidad con distintos chasis y otra con matrículas deberíamos de determinar que cada chasis solo puede tener una matrícula (y cada matrícula un chasis, ni más en ningún caso).

Fig. 4 *Entidad uno a uno.*

Uno a varios o **varios a uno**: determina que un registro de una entidad puede estar relacionado con varios de otra entidad, pero en esta entidad existir solo una vez. Como ha sido en el caso anterior del trabajador del taller.

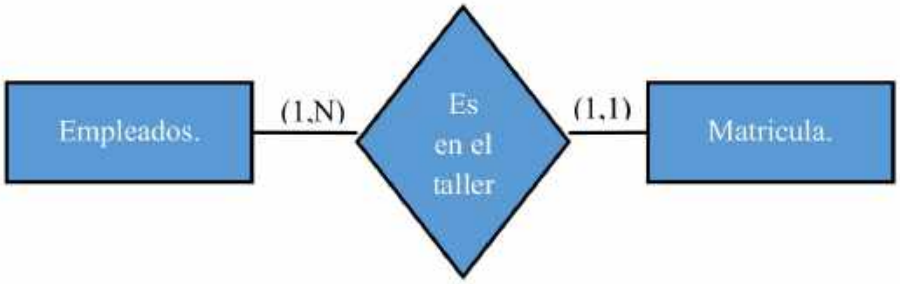


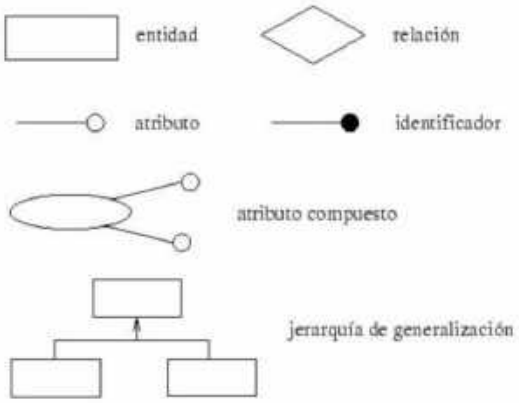
Fig. 5 *Entidad uno a varios o varios a uno.*

Varios a varios: determina que una entidad puede relacionarse con otra con ninguno o varios registros y viceversa. Por ejemplo, en el taller un coche puede ser reparado por varios mecánicos distintos y esos mecánicos pueden reparar varios coches distintos.

Fig. 6 *Entidad varios a varios.*

Los indicadores numéricos indican el primero el número mínimo de registros en una relación y posteriormente el máximo (si no hay límite se representa con una "*n*").

secretaria.general@itstena.edu.ec km 1 1/2 vía Tena- Archidona ISTT



GUÍA DE ESTUDIO DE BASE DE DATOS

Unidad II

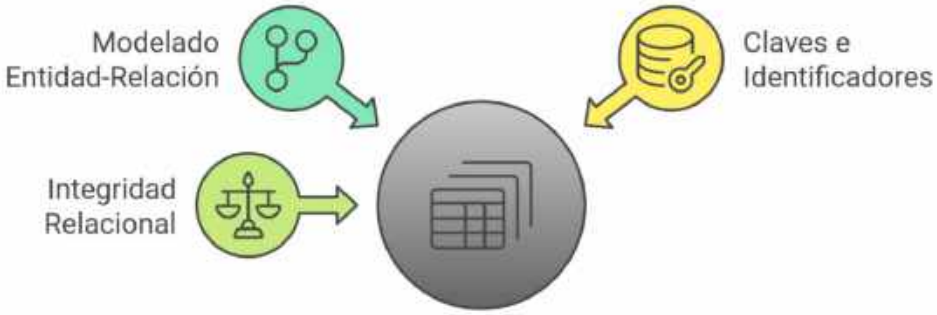
MODELAMIENTO DE BASE DE DATOS

Tipos de modelado.
Modelado Entidad Relación.
Claves o Identificadores.
Integridad Relacional.

Resultado de Aprendizaje

El estudiante será capaz de diseñar modelos conceptuales, lógicos y físicos de bases de datos utilizando herramientas de modelado y aplicando reglas de negocio, para resolver problemáticas específicas en sistemas de información reales.

DIAGRAMA DE APRENDIZAJE.



SÍNTESIS.

La Unidad 2 profundiza en el **modelado de bases de datos**, una etapa crucial en el diseño de sistemas de información que garantiza una estructura eficiente y coherente. Se exploran diversos **tipos de modelado**, destacando el **Modelo Entidad-Relación (ER)**, ampliamente utilizado para representar gráficamente las entidades, sus atributos y las relaciones entre ellas, facilitando la comprensión y comunicación del diseño de la base de datos. Se enfatiza la importancia de las **claves o identificadores**, como las claves primarias, que aseguran la unicidad de cada registro en una tabla, y las claves foráneas, que establecen vínculos entre tablas, manteniendo la integridad referencial. Además, se aborda la **integridad relacional**, que engloba reglas y restricciones destinadas a preservar la exactitud y consistencia de los datos, garantizando que las relaciones entre tablas sean válidas y que las operaciones en la base de datos no comprometan su coherencia. Comprender y aplicar estos conceptos es esencial para diseñar bases de datos robustas y confiables que satisfagan las necesidades de los usuarios y las aplicaciones.

Modelo Entidad – Relación.

Según (Pedro Gutiérrez, 2013) certifica que:

Es solo y exclusivamente un método del que disponemos para diseñar estos esquemas que posteriormente debemos de implementar en un gestor de BBDD (bases de datos). Este modelo se representa a través de diagramas y está formado por varios elementos. Este modelo habitualmente, además de disponer de un diagrama que ayuda a entender los datos y como se relacionan entre ellos, debe de ser completado con un pequeño resumen con la lista de los atributos y las relaciones de cada elemento.

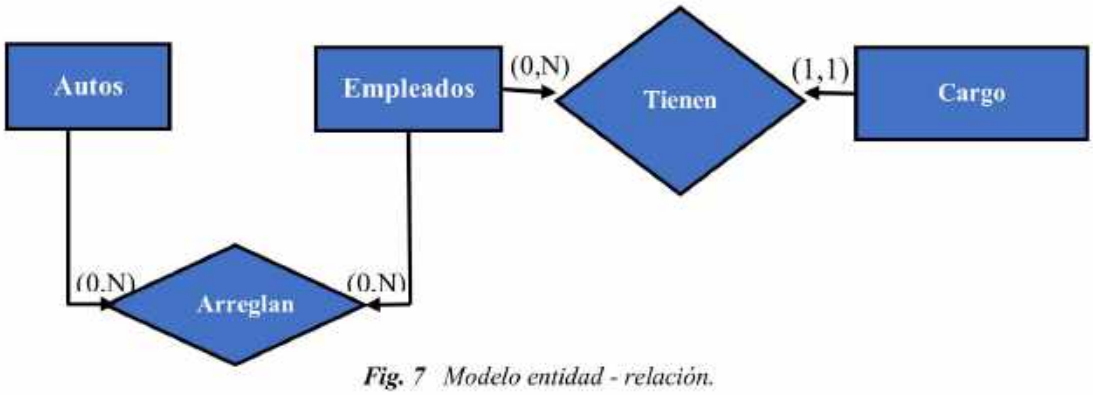


Fig. 7 *Modelo entidad - relación.*

Entidad:

Las entidades representan *cosas* u *objetos* (ya sean reales o abstractos), que se diferencian claramente entre sí.

Para poder seguir un ejemplo durante el artículo añadiré ejemplos sobre un taller mecánico, donde se podría crear las siguientes entidades:

- **Coches** (objeto físico): contiene la información de cada taller.
- **Empleado** (*objeto* físico): información de los trabajadores.
- **Cargo del empleado** (*cosa* abstracta): información de la función del empleado.

Estas entidades se representan en un diagrama con un rectángulo, como los siguientes.

Fig. 8 *Entidades representadas.*

Relación:

Es un vínculo que nos permite definir una dependencia entre varias entidades, es decir, nos permite exigir que varias entidades compartan ciertos atributos de forma indispensable.

Por ejemplo, los empleados del taller (de la entidad "**Empleados**") tienen un cargo (según la entidad "**Cargo del empleado**"). Es decir, un atributo de la entidad "*Empleados*" especificará que cargo tiene en el taller, y tiene que ser idéntico al que ya existe en la entidad "*Cargo del empleado*".

Las relaciones se muestran en los diagramas como rombos, que se unen a las entidades mediante líneas.

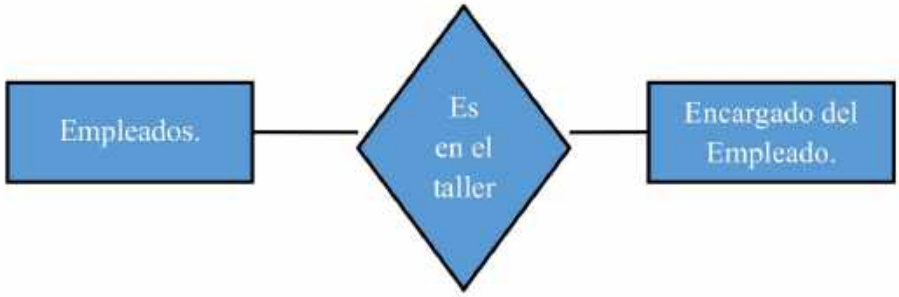


Fig. 9 *Entidades relacionadas.*

Restricciones

Según (Adoreccion de Miguel - Mario Piattini - Esperanza Marcos, 2000) indica que:

El modelo E/R tiene como restricción inherente que solo permite establecer interrelaciones entre entidades, no estando admitidas entre entidades e interrelaciones ni entre interrelaciones. También obliga el modelo a que todas las entidades tengan un identificador, lo que asimismo podría considerarse una restricción inherente. El no tener apenas restricciones inherentes dota al modelo de una gran flexibilidad para la representación del mundo real.

En cuanto a restricciones de integridad, únicamente consideramos as estricciones especificas, distinguiendo entre las restricciones sobre valores y las estructurales.

Las restricciones sobre valores se establecen mediante la definición de dominio, la cual permite limitar los valores del dominio y, por tanto, los de los atributos sobre el definidos, a los de un determinado tipo de datos, o restringirlos a los comprendidos en un rango, o bien declarar los valores posibles en el caso de que la definición se haga por extensión. Las restricciones estructurales se refieren tanto a atributos como a interrelaciones; estas últimas las analizaremos más adelante cuando tratemos la semántica de las interrelaciones, mientras que de las que atañen a los atributos nos ocupamos a continuación.

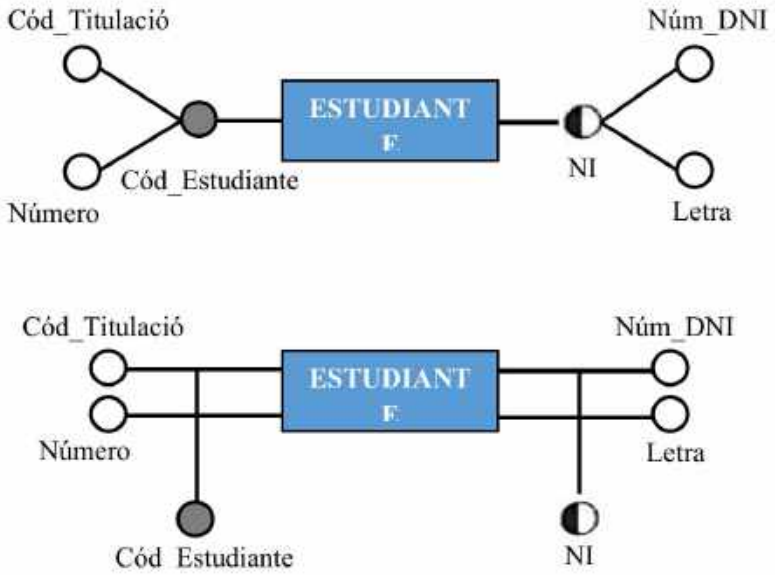


Fig. 10 *Representación gráfica de IP y IA.*

Entre todos los atributos de un tipo de entidad han de existir uno a varios (simples y/o compuesto) que identifiquen unívocamente cada una de os ejemplares de ese tipo de entidad. Cada una de estos conjuntos de atributos se denomina Identificador Candidato (IC). Cuando un IC es compuesto, el número de los atributos que lo componen debe ser mínimo, en el sentido de que la eliminación de cualquiera de ellos le haría perder su carácter identificador.

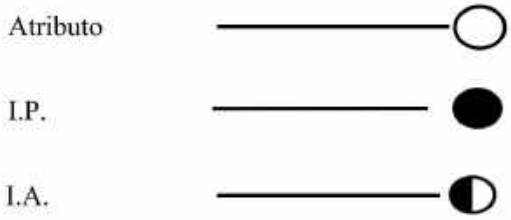


Fig. 11 *Ejemplo de IP y de IA compuesto.*

Luego todo IC debe cumplir a condición de ser univoco y mínimo. Entre los IC se eligen uno como Identificador Principal (IP) y el resto será Identificador Alterno (IA).

Los identificadores principales (o alternos) compuestos se pueden representar de forma análoga a la de los atributos compuestos tal como se muestra en el ejemplo de la figura 11.

El modelo E/R permite también atributos multivaluados y opcionales (nulo o "faltante"). En general un atributo toma, para cada ejemplar de entidad, un único valor de cada dominio (o dominios) subyacente(s) (un libro tiene un único título, un único ISBN, etc.), pero también existen atributos que pueden tomar más de un valor (un curso puede impartirse en más de un idioma, o un profesor puede tener más de un teléfono); estos

atributos reciben el nombre de multivaluados frente a los univaluados que toman un solo valor. Por otro lado, puede obligarse a una tributo de un tipo de entidad a que tome, como mínimo, un valor del (o de los) dominio/(s subyacentes/s para cada ejemplar de entidad; es decir, el valor de ese atributo es obligatorio (no puede ser nulo) para todo ejemplar de la entidad. La prohibición de valores nulos para un atributo (no admitir la opcionalidad) y la de que un atributo pueda tomar más de un valor (no admitir que sea multivaluado) son restricciones específicas sobre la estructuras de los atributos, al igual que la declaración de atributos identificadores. En la figura 12 se muestre una forma de representar los atributos multivaluados / univaluados y opcionales/obligatorios.

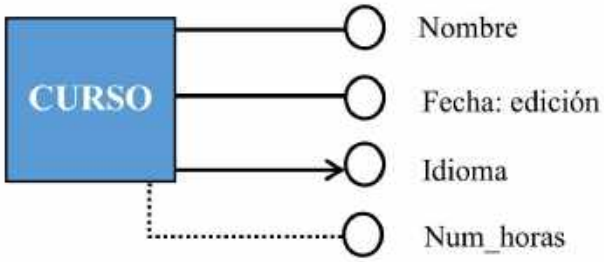


Fig. 12 *Ejemplo de atributos multivaluados (Idioma) y opcional (Núm_Horas).*

Se puede observar en la figura que, en lugar de representar la existencia de restricción (univaluacion u obligatoriamente de un atributo), lo que se representa con un símbolo especial (línea discontinua o punta de flecha) es la ausencia de restricción,; la razón es que lo más habitual es que un atributo sea univaluado y obligatorio , por lo que son estas las características que se toman por defecto y, por tanto, son las contrarias las que se representan con símbolos especiales.

Todas estas restricciones pueden definirse basándose en el concepto de cardinalidad de un atributo en el tipo de entidad o de interrelación al cual pertenece. Se entiende por cardinalidad mínima (o máximo) de un atributo el número mínimo (o máximo) de valores que pueden tomar ese atributo en cada ejemplar del tipo de entidad al cual pertenece; las cardinalidades se representan asociando un par de números enteros (mín, máx) al correspondiente atributo. En la figura 13 aparecen los cuatro tipos posibles de cardinalidades, junto con la forma de representación que mostrábamos en el ejemplo de la figura 12.

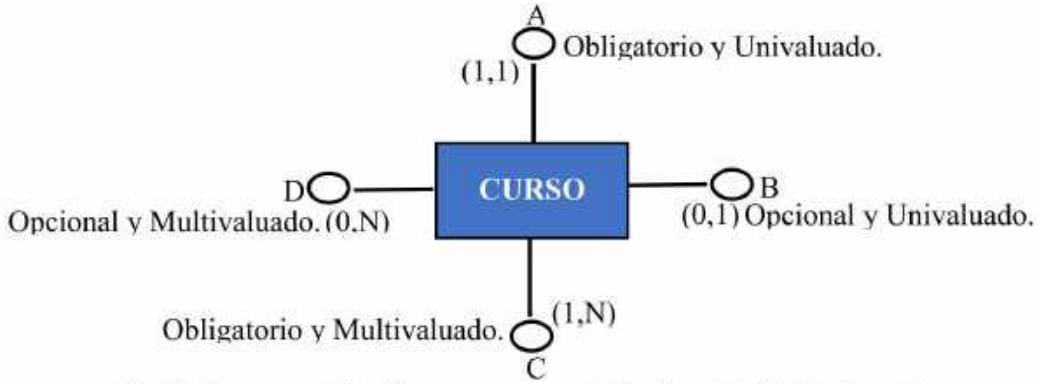


Fig. 13 *Representación de los cuatro tipos posibles de cardinalidades de atributos.*

También las cardinalidades, pero no del atributo sino del tipo de entidad respecto al atributo, permite representar otra restricción que es la unicidad, por la cual se obliga a que valores de un atributo no puedan repetirse en distintos ejemplares de un tipo entidad, en cuyo caso la cardinalidad máxima de esa entidad respecto al atributo es uno. Debemos observar que para todo identificador de un tipo de entidad una cardinalidad máxima de uno respecto a ese atributo. Sin embargo, la reciproca no es cierta, ya que la unicidad de un atributo no implica que sea un identificador, porque si el atributo es compuesto es preciso exigir, además, la condición de minimalidad; y, en todo caso, sea o no compuesto, se debe imponer también que se cumplan las restricciones de obligatoriedad y de univaluación. La cardinalidad mínima de la entidad respecto al atributo no tiene sentido, pero si lo tiene respecto al dominio sobre el cual este definido el atributo que no aparezcan en el atributo para ningún ejemplar del tipo de entidad, mientras que un valor de 1 indica que todos los valores del dominio deben aparecer como valores del atributo en algunas de las instancias del tipo entidad. En la figura 14 aparecen las Cardinalidades del identificador de la entidad E.

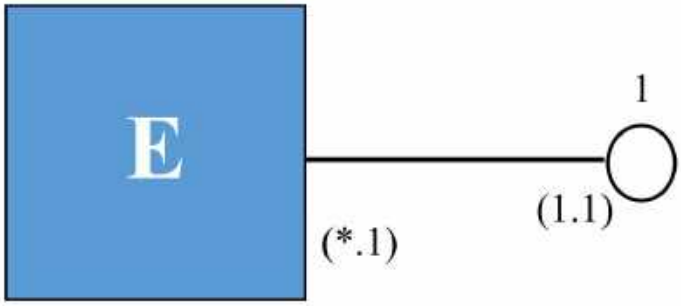


Fig. 14 *Cardinalidades del identificador de la entidad E*

En la figura 15 observamos el tipo de entidad CURSO con algunos de sus atributos y un ejemplar que tome valores de diferentes dominios.

Con independencia de las restricciones que acabamos de ver , y de las restricciones estructurales sobre interrelaciones que estudiamos posteriormente (todas ellas restricciones de condición especifica), el modelo E/R no proporciona instrumentos para la declaración de otras restricciones (restricciones de condición general), las cuales solo podrían ser formuladas mediante un lenguaje general de definición de restricciones, ajeno al modelo E/R o por medio de comentarios que acompañen al diagrama.

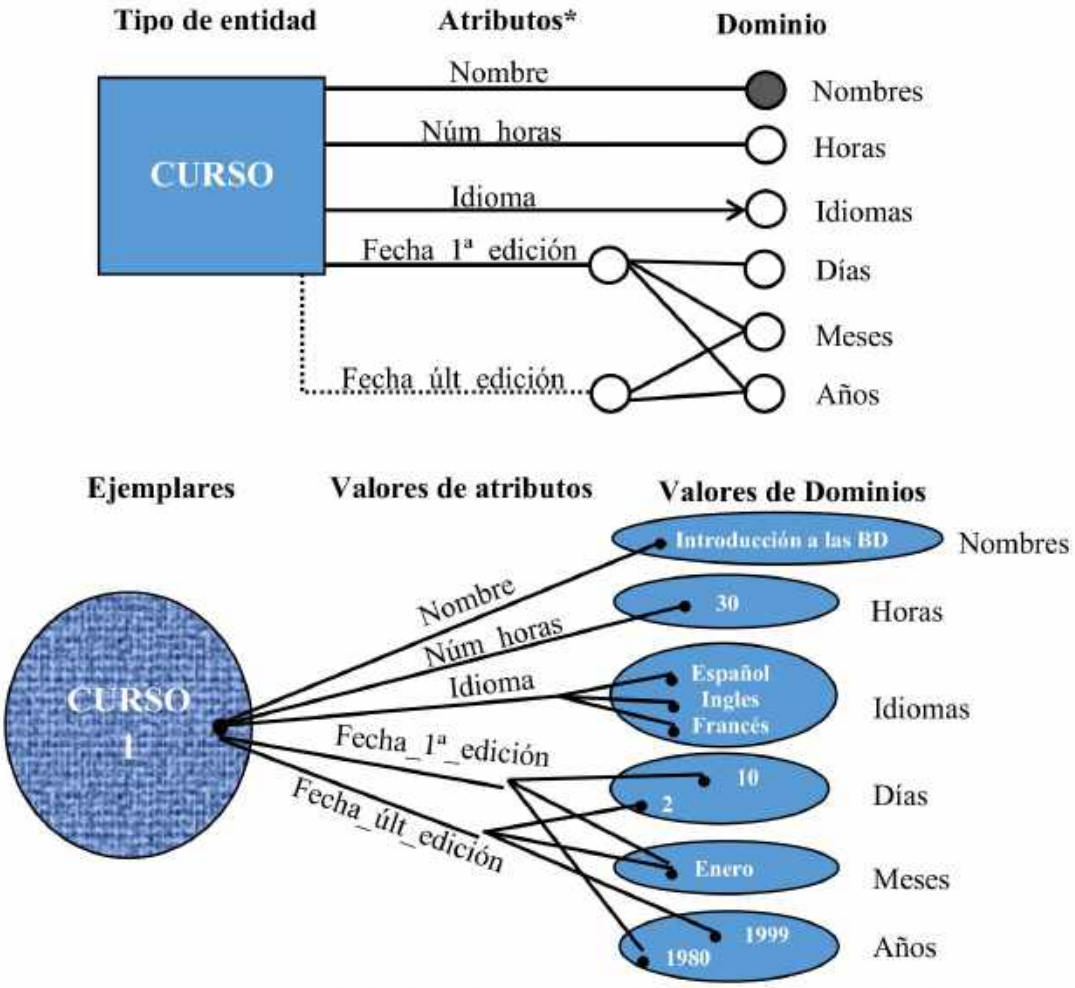


Fig. 15 *Ejemplo del tipo de entidad CURSO, con alguno de sus atributos, y de un ejemplar de CURSO con sus valores.*

Según (Korth , Silberschatz) cita que :

La cardinalidad de asignación expresa el número de entidades con las que se puede asociar otra entidad a través de un conjunto de relaciones.

En una relación binaria entre las entidades A y B, la cardinalidad debe ser una de las siguientes:

Fig. 16 *Cardinalidad.*

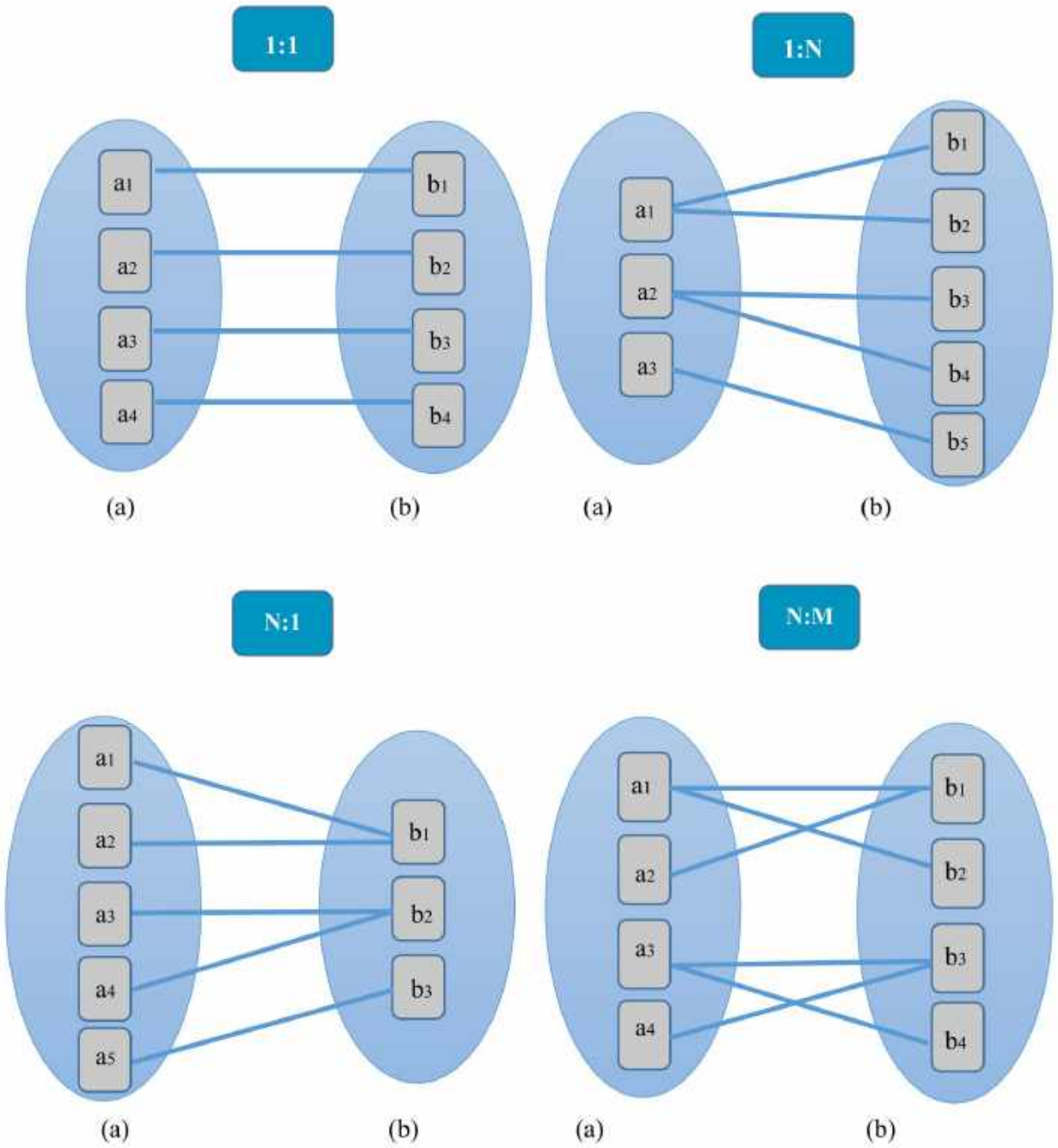


Fig. 17 *Ejemplo de cardinalidad.*

La cardinalidad depende del mundo real que se está modelando.
Ejemplo:

Para la relación CtaCli.

- *Si una cuenta puede pertenecer únicamente a un cliente, y un cliente puede tener varias cuentas.* ⇒⇒ *1:N de Cliente a Cuenta.*

- *Si una cuenta puede pertenecer a varios clientes, y un cliente puede tener varias cuentas.* ⇒⇒ *N:N*

Claves o identificadores (primarios, alternos, foráneos).

Según (Pedro Gutiérrez, 2013) menciona que:

Claves:

Es el atributo de una entidad, al que le aplicamos una restricción que lo distingue de los demás registros (no permitiendo que el atributo específico se repita en la entidad) o le aplica un vínculo (exactamente como comentábamos en las relaciones). Estos son los distintos tipos:

Superclave: aplica una clave o restricción a varios atributos de la entidad, para así asegurarse que en su conjunto no se repitan varias veces y así no poder entrar en dudas al querer identificar un registro.

Ejemplos:

Cliente (nombre-cliente, seguridad-social, calle, ciudad-cliente)
–{nombre-cliente, seguridad-social}
–seguridad-social•

Si K es una superclave, también lo será cualquier superconjunto de K. (Korth , Silberschatz)

Clave primaria: identifica inequívocamente un solo atributo no permitiendo que se repita en la misma entidad. Como sería la matrícula o el número de chasis de un coche (no puede existir dos veces el mismo).

Clave Primaria

ID_Estudiante	**Código_ Estudiante**	**Nombre_ Estudiante**	**Apellido_Estudiante**
C001	B0001	Jhonathan	Chávez
C002	B0002	Carlos	Arteaga
C003	B0003	Miriam	Figueroa

Tabla 1*. Ejemplo de clave primaria*

Una clave primaria se ajusta a la definición de *identificador*, en cuanto a que determina de forma única una instancia de una entidad. (Teorey, T. J., Lightstone, S. S., Nadeau, T., & Jagadish, H., 2011)

Una clave primaria es un campo o grupo de campos que identifica de forma única a cada registro dentro de una tabla (Hernandez, M. J., 2013)

La clave primaria se utiliza para identificar a un registro de manera única. También se le conoce como identificador de la entidad. Cuando más de un elemento dato se utiliza para identificar a un registro, se le denomina *clave concatenada*. (Singh, S. K., 2011)

Clave externa o **clave foránea**: este campo tiene que estar estrictamente relacionado con la clave primaria de otra entidad, para así exigir que exista previamente esa clave. Anteriormente hemos hablado de ello cuando comentábamos que un empleado indispensablemente tiene que tener un cargo (que lo hemos representado numéricamente), por lo cual si intentásemos darle un cargo inexistente el gestor de bases de datos nos devolvería un error. (Pedro Gutiérrez, 2013)

Una clave foránea es una columna o grupo de columnas de una tabla que contiene valores que coinciden con la clave primaria de otra tabla. Las claves foráneas se utilizan para unir tablas. La figura siguiente muestra las claves primaria y foránea de las tablas **customer** y **orders** de la base de datos de demostración.

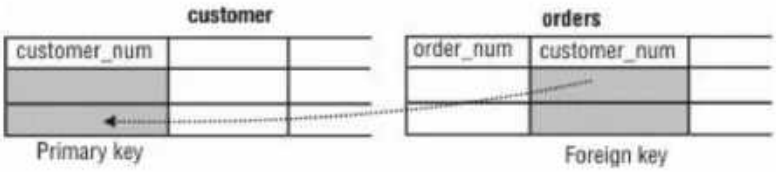


Fig. 18 *Claves primaria y foránea en las relaciones de cliente y pedido*

Las claves foráneas se indican cuando aparecen en el modelo porque su presencia puede restringir su capacidad para suprimir filas de tablas. Para poder suprimir una fila de forma segura, debe suprimir todas las filas que hacen referencia a la misma mediante claves foráneas o debe definir la relación con una sintaxis especial que le permita suprimir filas de columnas de clave primaria y de clave foránea con un solo mandato delete. El servidor de bases de datos no permite realizar supresiones que violen la integridad de referencia.
Para conservar la integridad de referencia, suprima todas las filas de clave foránea antes de suprimir la clave primaria a la que hacen referencia. Si impone restricciones de referencia en la base de datos, el servidor de bases de datos no le permite suprimir claves primarias que tengan claves foráneas coincidentes.

Tampoco le permite añadir un valor de clave foránea que no haga referencia a un valor de clave primaria existente. (IBM DNS Admin, 2000)

Extensiones.

Según (Adoracion de Miguel - Mario Piattini - Esperanza Marcos, 2000). Menciona que:

Es el conjunto de tuplas que, en un instante determinado, satisfacen el esquema de relación y se encuentran almacenadas en la base de datos; es lo que se suele llamar, simplemente, relación. La extensión varía en el transcurso del tiempo.

La extensión en un determinado momento será el conjunto de ejemplares que, en ese momento, pertenecen al tipo y están almacenados en la base de datos.

La extensión del esquema relacional, constituido por una colección de relaciones, es la base de datos relacional.

Lista de extensiones:

Extensión	Descripción
$er	GroupWise Database
4dd	4th Dimension Database Data File
4dl	4th Dimension Database Log File
dtsx	DTS Settings File
dwn	FrameWork database file
dxl	Domino XML Language File
eco	ECCO Database File
ecx	ECCO Corrupted Database File
v12	All The Right Type Database File
vis	Visual Importer Script
wdb	Microsoft Works Database
wmdb	Windows Media Database File
xld	Excel Database File
xsl	Microsoft SharePoint WorkSpace Groove database file
^^^	Pervasive.SQL Database File

***Tabla 2**. Lista de extensiones.*

Integridad relacional.

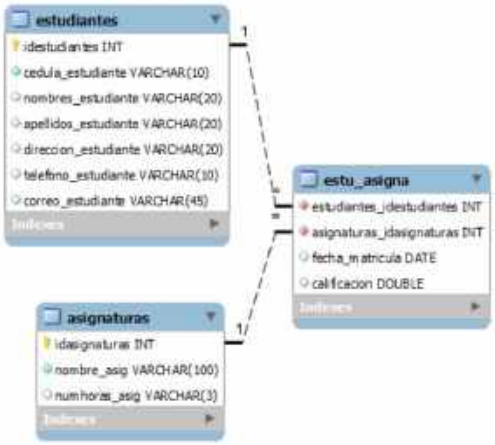
En un momento dado, los valores de los datos en una base de datos son una representación de un fragmento de la realidad. Es decir, si tenemos una tabla con los atributos de personas y entre ellos el peso o la edad, estos no pueden ser negativos, porque en el mundo real, esto no es posible. Si añadimos una restricción de este tipo a una base de datos, estamos incluyéndole una *regla de integridad*.

Por ejemplo, si tenemos una base de datos alumnos, profesores y cursos para una escuela o facultad, algunas reglas de integridad serían:

- Las claves de los alumnos son de la forma AL*aaaannnn* donde *aaaa* son los cuatro dígitos del año de ingreso y *nnnn* son cuatro dígitos que representan un número secuencial.
- Las claves de los profesores son de la forma AC*mmnn* donde *mm* es la clave del departamento al que está asociado y *nn* es un secuencial.
- Las claves de cursos son de la forma MA*mmnnaa* donde *mm* es la clave del departamento, *nn* es la clave de la materia y *aa* son los dos dígitos menos significativos del año.
- Un alumno no puede estar inscrito en más de cinco materias.
- Un maestro no puede dar más de tres materias.
- Un curso no puede tener menos de cinco alumnos ni más de doce.
- Un maestro no puede dar la misma materia más de dos semestres seguidos.
- Un alumno que no aprueba una materia en la segunda oportunidad será dado de baja.
- Los departamentos vienen de una determinada lista.
- Las materias tienen que existir en otra lista.
- Las calificaciones no pueden tomar valores fuera del rango 81#81.

Algunas de estas reglas son arbitrarias y para fines de ejemplificar el concepto y es inmediato notar que se aplican a tablas en específico.

Sin embargo, las bases de datos relacionales, tienen dos reglas *generales* de integridad que se aplican a las llaves primarias y a las llaves foráneas.



GUÍA DE ESTUDIO DE DE BASE DE DATOS

Unidad III

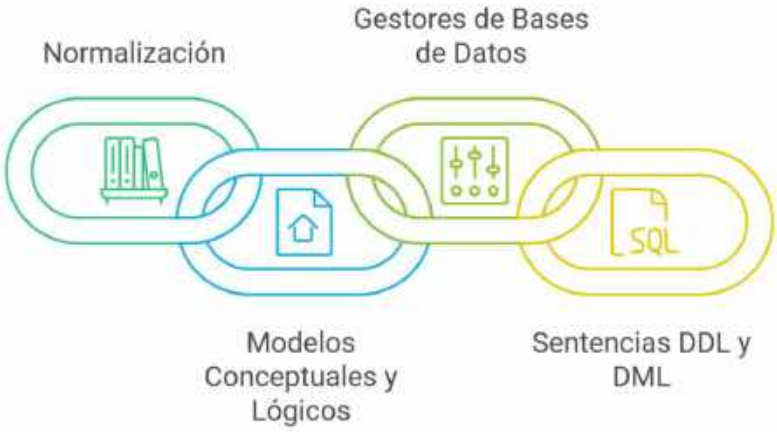
OPTIMIZACIÓN E IMPLEMENTACIÓN DE BASES DE DATOS

Normalización.
Modelo Conceptual y Lógico.
Gestores de base de datos.
Sentencias DDL y DML

Resultado de Aprendizaje

El estudiante será capaz de **evaluar** el rendimiento de bases de datos mediante técnicas de optimización y normalización, identificando y **corrigiendo** problemas de redundancia y consistencia de datos para asegurar la eficiencia en su implementación.

DIAGRAMA DE APRENDIZAJE.



SÍNTESIS.

La Unidad 3 se centra en la optimización y diseño eficiente de bases de datos mediante procesos clave. Se introduce la normalización, una técnica que elimina redundancias y garantiza la consistencia de los datos al estructurarlos en tablas interrelacionadas, mejorando el rendimiento y la integridad del sistema. Además, se exploran los modelos conceptual y lógico, etapas fundamentales en el diseño de bases de datos que transforman requerimientos del negocio en estructuras claras y organizadas para su posterior implementación física. También se estudian los gestores de bases de datos (DBMS), herramientas que permiten crear, administrar y consultar bases de datos, con ejemplos populares como MySQL, PostgreSQL y MongoDB. Finalmente, se profundiza en el uso de sentencias DDL (Data Definition Language) y DML (Data Manipulation Language), esenciales para la definición, manipulación y consulta de los datos, abarcando operaciones como la creación de tablas, inserción, actualización y eliminación de registros. Esta unidad prepara al estudiante para implementar y gestionar bases de datos eficientes y funcionales.

Normalización.

Según (Valero, 2016) menciona que:

La normalización de base de datos es una técnica de modelado consistente en **designar** y **aplicar** una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional.

Objetivos

- Evitar redundancia
- Simplificar la actualización de datos
- Garantizar la integridad referencial

Requerimientos:

Para que una tabla sea considerada una relación tiene que cumplirse lo siguiente:

- Cada tabla tiene que tener un nombre único

- ❖ No pueden haber dos filas iguales – No se permiten duplicados
- ❖ Todos los datos en una columna deben ser del mismo tipo

Conceptos:

A continuación vamos a definir una serie de conceptos fundamentales para comprender las formas normales que explicaremos después:

- ❖ **Dependencia Funcional**: Es una conexión entre uno o más atributos DNI → Nombre y apellidos.
- ❖ **Dependencia Funcional Reflexiva**: Si "Y" está incluido en "X" entonces X → Y

  Si dirección y nombre están incluidos en DNI entonces con el DNI se puede recuperar la dirección y el nombre.

- ❖ **Dependencia Funcional Aumentativa:** Si "X" → "Y" entonces "XZ" → «YZ" DNI → Nombre DNI, Dirección → Nombre, Dirección.

- ❖ **Dependencia Funcional Transitiva:** Si "X" → "Y" → «Z" entonces "X" → "Z"

  Fecha de nacimiento → Edad, Edad → Conducir, Fecha de nacimiento → Edad → Conducir.

Ejemplo:
Para explicar las formas normales vamos a poner una tabla de ejemplo de Empleados. La siguiente tabla muestra la información de una empresa cuyos puestos de trabajo están regulados por el Estado, de modo que el salario de cada empleado depende del puesto. Datos empleados: ID, nombre, puesto salario e email, siendo ID la clave primaria.

ID (Pk)	Nombre	Puesto	Salario	Emails
1	Juan Pérez	Jefe de área	3000	juan@test.com;jefe1@test.com
2	José Sánchez	Administrativo	1500	jsanchez@test.com

***Tabla 3**. Empleados*

Primera Forma De Normalización.

Una tabla está en primera forma normal si:

- Sus atributos contienen valores atómicos (esto quiere decir que tienen que ser indivisibles)

En el ejemplo podemos ver que no se cumple para el atributo «Emails»

ID (Pk)	Nombre	Puesto	Salario	Emails	
1	Juan Pérez	Jefe de área	3000	juan@test.com;jefe1@test.com	X

2	José Sánchez	Administrativo	1500	jsanchez@test.com	
3	Ana Díaz	Administrativo	1500	adiaz@test.com;admin@test.com	X

***Tabla 4**. Empleados*

Para solucionarlo existen 2 opciones:

1. Duplicar registros con valores repetidos:
   - Se elimina el atributo «Emails» que violaba la condición
   - Se incluye un nuevo atributo «Email» que sí sea indivisible.
   - Por lo que se crea una nueva clave primaria con este nuevo atributo.

La nueva clave primaria será «ID-Email»

ID (Pk)	Nombre	Puesto	Salario	Emails (Pk)
1	Juan Pérez	Jefe de área	3000	juan@test.com
1	Juan Pérez	Jefe de área	3000	jefe1@test.com
2	José Sánchez	Administrativo	1500	jsanchez@test.com
3	Ana Díaz	Administrativo	1500	adiaz@test.com
3	Ana Díaz	Administrativo	1500	admin@test.com

***Tabla 5**. Empleados (a)*

2. Separar atributo «Email» en otra tabla:
   - Se crea una nueva tabla Empleados (b) que no contenga el atributo «Email»

ID (Pk)	Nombre	Puesto	Salario
1	Juan Pérez	Jefe de área	3000
1	Juan Pérez	Jefe de área	3000
2	José Sánchez	Administrativo	1500

***Tabla 6**. Empleados (b)*

   - Se crea una nueva tabla EMAILS con clave primaria ID-Email. Las tablas Emails y Empleados se relacionan por el campo ID

ID	Emails (Pk)
1	juan@test.com
1	jefe1@test.com
2	jsanchez@test.com
3	adiaz@test.com
3	admin@test.com

***Tabla 7**. Email*

Segunda Forma De Normalización.

Una tabla está en 2FN si:

- Está en 1FN
- Todos los atributos que no son clave primaria tienen dependencia funcional completa con respecto a todas las claves existentes en el esquema. Para recuperar un atributo no clave, se necesita acceder por la clave completa, no por una subclave.
- Las 2FN aplican a las relaciones con claves primarias compuestas por dos o más atributos

ID (Pk)	Emails (Pk)	Nombre	Puesto	Salario
1	juan@test.com	Juan Pérez	Jefe de área	3000
1	jefe1@test.com	Juan Pérez	Jefe de área	3000
2	jsanchez@test.com	José Sánchez	Administrativo	1500
3	adiaz@test.com	Ana Díaz	Administrativo	1500
3	admin@test.com	Ana Díaz	Administrativo	1500

***Tabla 8**. Empleados (a)*

En la tabla de Empleados (a) se pueden ver las dependencias de los atributos:

ID → Nombre, puesto, salario
Puesto → Salario

Observamos que los atributos nombre, puesto y salario dependen únicamente del campo ID, por lo que **no cumple la 2FN**.
Para solucionarlo:
→ Actuar sobre los atributos con dependencias incompletas:

- Eliminar los atributos con dependencias incompletas

- Crear nueva tabla con los atributos y la clave de la que depende

ID (Pk)	Nombre	Puesto	Salario
1	Juan Pérez	Jefe de área	3000
1	Juan Pérez	Jefe de área	3000
2	José Sánchez	Administrativo	1500

***Tabla 9**. Empleados (b)*

ID	Emails (Pk)
1	juan@test.com
1	jefe1@test.com
2	jsanchez@test.com

3	adiaz@test.com
3	admin@test.com

***Tabla 10**. Email*

Se llega a la misma solución que con la 1FN.

Tercera Forma De Normalización.

Una tabla está en 3FN si:

- Está en 2FN
- Todos los atributos que no son clave primaria no dependen transitivamente de ésta

Por tanto hay que buscar dependencias funcionales entre atributos que no estén en la clave.

ID (Pk)	Nombre	Puesto	Salario
1	Juan Pérez	Jefe de área	3000
2	José Sánchez	Administrativo	1500
3	Ana Díaz	Administrativo	1500

***Tabla 11**. Empleados (b)*

Las dependencias transitivas son:

ID → Puesto

Puesto → Salario

Observamos como la dependencia Puesto – Salario tiene dependencia transitiva con la clave primaria.

Para solucionarlo:

→ Actuar sobre los atributos con dependencias transitivas

- Separar en una tabla adicional los atributos que tienen dependencia transitiva con la clave (Salario) y establecer como Pk el campo que define la transitividad (Puesto).

Puesto (Pk)	Salario
Jefe de área	3000
Administrativo	1500

***Tabla 12**. Puestos*

ID (Pk)	Nombre	Puesto (Pk)
1	Juan Pérez	Jefe de área
2	José Sánchez	Administrativo
3	Ana Díaz	Administrativo

***Tabla 13**. Empleados (c)*

- Se añade el campo «Puesto» como *Foreign Key:*

ID (Pk)	Nombre	Puesto (Fk)
1	Juan Pérez	Jefe de área
2	José Sánchez	Administrativo
3	Ana Díaz	Administrativo

***Tabla 14**. Emails*

Modelo Conceptual, Lógico y físico.

Según (Wild West Domains, 2009) menciona que:

Los modelos de datos definen cómo se modela la estructura lógica de una base de datos. Los modelos de datos son entidades fundamentales para introducir la abstracción en una base de datos.

Los modelos de datos definen cómo los datos se conectan entre sí y cómo se procesan y almacenan dentro del sistema.

El primer modelo de datos fue el modelo de datos planos, donde todos los datos utilizados se mantendrían en el mismo plano.

numero_cedula
1500814242
0157267382
0915544110
0966397462

***Tabla 15**. Modelo de datos planos*

Los primeros modelos de datos no eran tan científicos, por lo tanto, eran propensos a introducir muchas anomalías de duplicación y actualización.

Un modelo de datos puede ser concreto o abstracto, y están representados por la notación de modelado de datos, que a menudo se presenta en formato gráfico.

Su enfoque principal es apoyar y ayudar a los sistemas de información mostrando el formato y la definición de los diferentes datos involucrados.

También ayudan a evitar la redundancia de datos. La información almacenada en los modelos de datos es de gran importancia para las empresas porque dicta las relaciones entre las tablas de la base de datos, las claves externas y los eventos involucrados.

Modelo Conceptual.

Un modelo conceptual de datos identifica las relaciones de más alto nivel entre las diferentes entidades.

Las características del modelo conceptual de datos incluyen:

- Incluye las entidades importantes y las relaciones entre ellas.
- No se especifica ningún atributo.
- No se especifica ninguna clave principal.

La siguiente figura es un ejemplo de un modelo conceptual de datos.

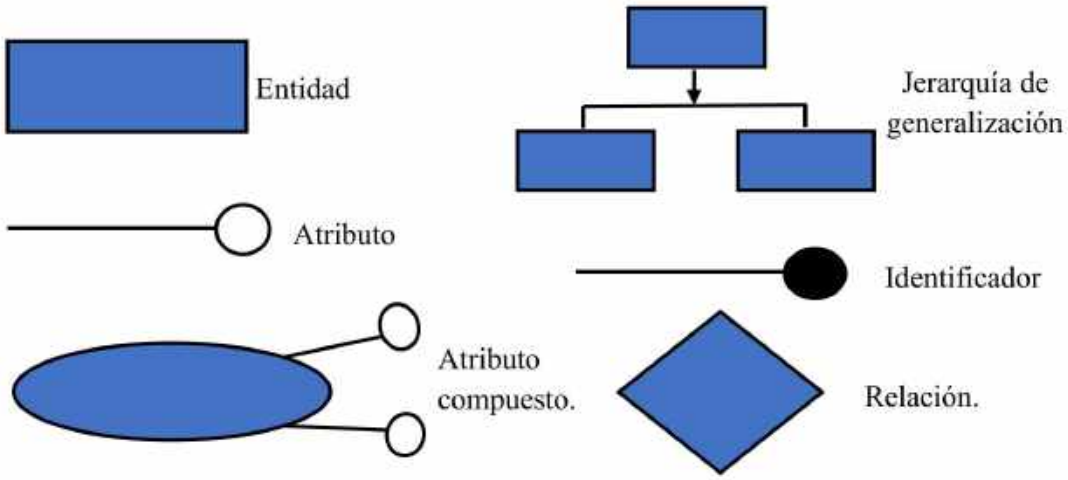


Fig. 19 Modelo conceptual de datos.

Modelo Lógico.

El modelo de datos físicos representa cómo se construirá el modelo en la base de datos. Un modelo de base de datos física muestra todas las estructuras de tabla, incluidos el nombre de columna, el tipo de datos de columna, las restricciones de columna, la clave principal, la clave externa y las relaciones entre las tablas.

Las características de un modelo de datos físicos incluyen:

- Especificación de todas las tablas y columnas.
- Las claves externas se usan para identificar relaciones entre tablas.
- La desnormalización puede ocurrir según los requisitos del usuario.

Las consideraciones físicas pueden hacer que el modelo de datos físicos sea bastante diferente del modelo de datos lógicos. El modelo de datos físicos será diferente para diferentes Sistemas de Gestión de Base de datos. Por ejemplo, el tipo de datos para una columna puede ser diferente entre MySQL y SQL Server. Los pasos básicos para el diseño del modelo de datos físicos son los siguientes:

- Convertir entidades en tablas.
- Convertir relaciones en claves externas.
- Convertir atributos en columnas.
- Modificar el modelo de datos físicos en función de las restricciones / requisitos físicos.

La siguiente figura es un ejemplo de un modelo de datos físicos.

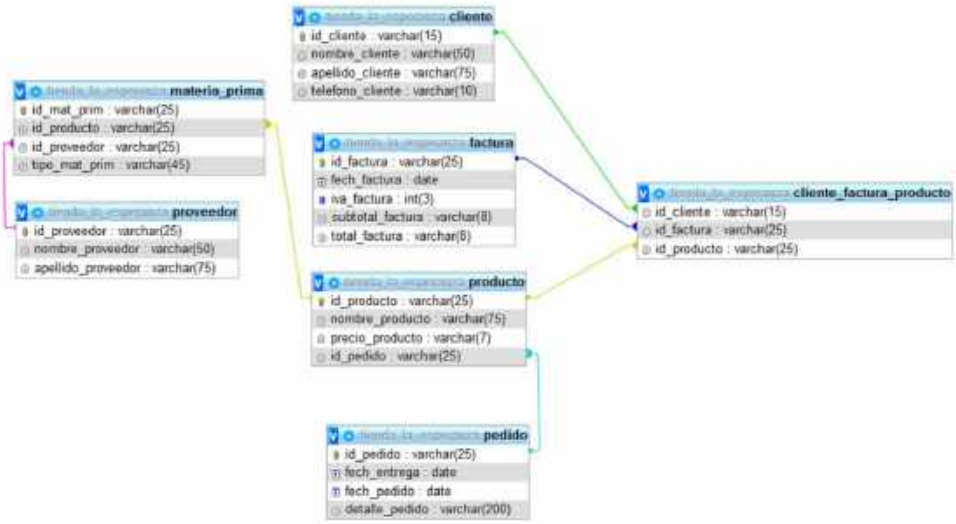


***Fig. 20** Modelo de datos físicos*

Modelo Físico.

Un modelo de datos lógicos describe los datos con el mayor detalle posible, independientemente de cómo se implementarán físicamente en la base de datos.
Las características de un modelo de datos lógicos incluyen:

- Incluye todas las entidades y relaciones entre ellos.
- Todos los atributos para cada entidad están especificados.
- La clave principal para cada entidad está especificada.
- Se especifican las claves externas (claves que identifican la relación entre diferentes entidades).
- La normalización ocurre en este nivel.

Los pasos para diseñar el modelo de datos lógicos son los siguientes:

- Especifique claves primarias para todas las entidades.
- Encuentra las relaciones entre diferentes entidades.
- Encuentra todos los atributos para cada entidad.
- Resuelva las relaciones de muchos a muchos.
- Normalización.

La siguiente figura es un ejemplo de un modelo de datos lógicos.
Aquí comparamos estos tres tipos de modelos de datos. La tabla a continuación compara las diferentes características:

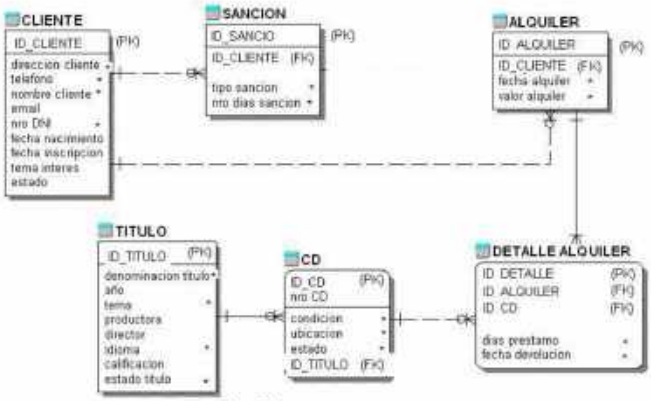


Fig. 21 *Modelo de datos lógicos*

Característica	Conceptual	Lógico	Física
Nombres de entidades	✓	✓	
Relaciones de entidades	✓	✓	
Atributos		✓	
Teclas principales		✓	✓
Foreign Keys		✓	✓
Nombres de tabla			✓
Nombres de columnas			✓
Tipos de datos de columna			✓

Tabla 16*. Ejemplo de comparación de característica conceptual, lógica y física.*

Ventajas y desventajas de los modelos de datos

Ventajas:

- El objetivo principal de un modelo de datos es asegurarse de que los objetos de datos ofrecidos por el equipo funcional se representen con precisión.
- El modelo de datos debe ser lo suficientemente detallado para ser utilizado para construir la base de datos física.
- La información en el modelo de datos se puede utilizar para definir la relación entre tablas, claves primarias y externas y procedimientos almacenados.
- El modelo de datos ayuda a las empresas a comunicarse dentro y entre las organizaciones.
- El modelo de datos ayuda a documentar las asignaciones de datos en el proceso ETL.
- Ayuda a reconocer las fuentes de datos correctas para poblar el modelo.

Desventajas:

- Para desarrollar el modelo de datos se deben conocer las características físicas de los datos almacenados.

- Incluso los cambios más pequeños realizados en la estructura requieren modificaciones en toda la aplicación.
- No hay un lenguaje de manipulación de modelos establecido en DBMS.

Gestores de Bases de datos

Los Gestores de Bases de Datos o Sistemas de Gestión de Bases de Datos (SGBD) son programas diseñados para crear, administrar y manipular bases de datos de manera eficiente. Estos sistemas permiten a los usuarios almacenar, modificar y recuperar datos de forma estructurada, facilitando la gestión de grandes volúmenes de información en diversas aplicaciones y sectores.

Funciones principales de un SGBD:

- Definición de datos: Permite la creación y modificación de la estructura de la base de datos, incluyendo tablas, campos y relaciones.
- Manipulación de datos: Facilita operaciones como inserción, actualización, eliminación y consulta de datos almacenados.
- Control de acceso: Gestiona los permisos y restricciones para garantizar la seguridad y privacidad de la información.
- Integridad de datos: Asegura la consistencia y validez de los datos mediante restricciones y reglas definidas.

- Recuperación ante fallos: Proporciona mecanismos para restaurar la base de datos a un estado consistente después de errores o fallos del sistema.

Tipos de SGBD:

- Relacionales (RDBMS): Organizan los datos en tablas con relaciones predefinidas. Utilizan el lenguaje SQL para la gestión de datos. Ejemplos: MySQL, PostgreSQL, Oracle Database.
- NoSQL: Diseñados para manejar grandes volúmenes de datos no estructurados o semi-estructurados. Se clasifican en:
- Orientados a documentos: Almacenan datos en formatos como JSON o BSON. Ejemplo: MongoDB.
- Basados en columnas: Optimizados para consultas en columnas específicas. Ejemplo: Apache Cassandra.
- Orientados a grafos: Especializados en gestionar relaciones complejas entre datos. Ejemplo: Neo4j.
- Basados en clave-valor: Almacenan datos como pares clave-valor. Ejemplo: Redis.

Selección de un SGBD:

La elección de un gestor de bases de datos depende de diversos factores, como el tipo de datos a manejar, el volumen de información, los requerimientos de rendimiento, escalabilidad y las necesidades específicas de la aplicación o negocio. Es esencial evaluar las características de cada SGBD para determinar cuál se adapta mejor a las necesidades particulares.

Gestor MySQL

MySQL es un Sistema de Gestión de Bases de Datos Relacional (RDBMS) de código abierto ampliamente utilizado en el desarrollo de aplicaciones web y empresariales. Fue creado por MySQL AB en 1995 y es actualmente propiedad de Oracle Corporation. MySQL es conocido por su alto rendimiento, flexibilidad y facilidad de uso, lo que lo convierte en una opción popular para gestionar datos en una amplia gama de proyectos.

Características principales:

1. **Relacional:**
   Organiza los datos en tablas relacionadas mediante claves primarias y foráneas, permitiendo una estructura lógica y organizada.
2. **Lenguaje SQL:**
   Soporta el lenguaje SQL estándar, utilizado para definir, manipular y consultar datos en la base de datos.
3. **Código abierto:**
   Aunque ofrece una versión comercial, la edición comunitaria de MySQL es gratuita y de código abierto, con una amplia comunidad de desarrolladores y soporte técnico.

4. **Escalabilidad:**
   Es capaz de manejar desde pequeños proyectos con pocos registros hasta grandes aplicaciones con millones de transacciones diarias.
5. **Compatibilidad multiplataforma:**
   Funciona en sistemas operativos como Windows, Linux y macOS, ofreciendo flexibilidad para diversos entornos de desarrollo.
6. **Integración:**
   Es compatible con una variedad de lenguajes de programación como PHP, Java, Python y .NET, lo que facilita su integración en múltiples proyectos.
7. **Alta disponibilidad y replicación:**
   MySQL incluye características como replicación de bases de datos y clústeres para garantizar la disponibilidad de los datos y la tolerancia a fallos.
8. **Seguridad:**
   Ofrece autenticación de usuarios, cifrado de datos y configuración de permisos para garantizar la protección de la información.

Ventajas de MySQL:

- Facilidad de instalación y configuración.
- Gran soporte de la comunidad y documentación extensa.
- Optimizaciones para lectura rápida de datos.
- Herramientas gráficas como MySQL Workbench para diseño, desarrollo y administración de bases de datos.

Desventajas:

- Aunque es eficiente, puede no ser tan robusto como otras soluciones comerciales para aplicaciones de misión crítica.
- Algunas funciones avanzadas solo están disponibles en su versión comercial.

Casos de uso:

- Desarrollo de sitios web dinámicos.
- Gestión de aplicaciones empresariales.
- Implementación de sistemas de comercio electrónico.
- Análisis de datos para pequeñas y medianas empresas.

DDL

Según (Anita Ortiz, 2015) señaló que:
Un Data Definition Language o Lenguaje de descripción de datos (DDL) es un lenguaje de programación para definir estructura de datos . El término DDL fue introducido por primera vez en relación con el modelo de base de datos CODASYL, donde el esquema de la base de datos ha sido escrito en un lenguaje de descripción de datos que describe los registros, los campos, y "conjuntos" que conforman el usuario modelo de datos.

SQL

A diferencia de muchos lenguajes de descripción de datos, SQL utiliza una acción de versos imperativo cuyo efecto es modificar el esquema de la base de datos, añadiendo, cambiando o eliminando las declaraciones se pueden mezclar libremente con otras

sentencias SQL, por lo que el DDL no es realmente una lengua independiente. La declaración más común es CREATE TABLE. El lenguaje de consulta SQL, el más difundido entre los gestores de bases de datos, admite las siguientes sentencias de definición: CREATE, DROP y ALTER, cada una de las cuales se puede aplicar a las *tablas*, *vistas*, *procedimientos almacenados* y *triggers* de la base de datos.

Sentencia CREATE

Create - Sirve para crear una nueva base de datos, tabla, índice, o procedimiento almacenado. Una sentencia CREATE en SQL crea un objeto dentro de un sistema de gestión de bases de datos relacionales (RDBMS). Los tipos de objetos que se pueden crear dependen del RDBMS que esté siendo utilizado, pero la mayoría soportan la creación de tablas, índices, usuarios, sinónimos y bases de datos. Algunos sistemas (como PostgreSQL) permiten CREATE, y otros comandos DDL, en el interior de una transacción y por lo tanto puede ser revertido. Otras que se incluyen dentro del DDL, pero que su existencia depende de la implementación del estándar SQL que lleve a cabo el gestor de base de datos son GRANT y REVOKE, los cuales sirven para otorgar permisos o quitarlos, ya sea a usuarios específicos o a un rol creado dentro de la base de datos.

Sentencia CREATE TABLE

Un comando CREATE muy común es el CREATE TABLE. El uso típico es:

CREATE TABLE[nombre de la tabla] ([definiciones de columna]) [parámetros de la tabla]

Sentencia DROP

Sirve para borrar en forma sencilla distintos objetos dentro del [SGBD] como por ejemplo base de datos, tablas, índices. Su sentencia es:

```
DROP objeto_a_eliminar;
DROP TABLE    myTable;
DROP SEQUENCE mySequence;
DROP INDEX    myIndex;
```

Para eliminar una tabla de una base de datos tenemos la sentencia DROP TABLE. Con ella quitamos una o varias definiciones de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos que tengan esas tablas. Las vistas o procedimientos almacenados que hagan referencia a la tabla quitada se deben quitar explícitamente con DROP VIEW o DROP PROCEDURE. Su sintaxis es:

```
DROP TABLE [nbBaseDatos.[nbEsquema].|nbEsquema.]nbTabla[ ,...n ] [ ; ]
```

Sentencia ALTER

La sentencia ALTER TABLE es usada para agregar, borrar o modificar columnas en una tabla existente

Sintaxis de SQL ALTER TABLE

Para agregar una columna a una tabla, se debe usar la siguiente sintaxis:

```
ALTER TABLE nombre_tabla
ADD column_name tipo_datos
```

Para eliminar una columna en una tabla, se debe seguir la siguiente sintaxis (algunas bases de datos no permiten borrar columnas)

```
ALTER TABLE nombre_tabla
DROP COLUMN nombre_columna
```

DML

Lenguaje de Manipulación de Datos (**D**ata **M**anipulation **L**anguage, DML) es un lenguaje proporcionado por los sistemas gestores de bases de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en las Bases de Datos del Sistema Gestor de Bases de Datos.

Elementos del lenguaje de manipulación de datos

SELECT

La sintaxis básica de **select** es la siguiente utilizando el estándar de **SQL**:

select columna **from** tabla;

Donde se sustituye la palabra columna por el nombre del campo a consultar y la palabra tabla por el nombre de la tabla que contiene el campo mencionado.

Insert

La estructura básica para la sentencia **insert** utilizando el estándar de **SQL** es la siguiente:

insert into usuario (nombre, apellidos, edad, carrera) **values** ("Martín", "Bastida Godínez", "23", "Ingeniería en TI");

Tomando como ejemplo si se tuviera una tabla llamada 'usuario' con los campos de tipo cadena de caracteres (nombre, apellidos, edad, carrera), donde se inserta los valores que se encuentran en después de la palabra **values,** los valores se insertan en el orden correspondiente a como se hizo la llamada de los campos, los valores van separados por

comas, las comillas dobles indican que se está insertando datos de tipo cadena de caracteres.

Delete

Para eliminar los registros de una tabla usamos el comando **"delete"**:

```
delete from usuarios;
```

La ejecución del comando indicado en la línea anterior borra TODOS los registros de la tabla.

Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles, para ello utilizamos el comando **"delete"** junto con la clausula **"where"** con la cual establecemos la condición que deben cumplir los registros a borrar. Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es 'Martín':

delete from usuarios **where** nombre='Martín';

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, no se borrarán registros, pues no encontró registros con ese dato.

Update

Para modificar uno o varios datos de uno o varios registros utilizamos **"update"** (actualizar).

Por ejemplo, en nuestra tabla "usuarios", queremos cambiar los valores de todas las claves, por "Sevilla":

```
update usuarios set clave='Sevilla';
```

Utilizamos **"update"** junto al nombre de la tabla y **"set"** junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con **"where".**

Por ejemplo, queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado 'Martín', queremos como nueva clave 'Boca', necesitamos una condición **"where"** que afecte solamente a este registro:

```
update usuarios set clave='Boca'
where nombre='Martín';
```

Si no encuentra registros que cumplan con la condición del "where", ningún registro es afectado.

Las condiciones no son obligatorias, pero si omitimos la cláusula "where", la actualización afectará a todos los registros.

También se puede actualizar varios campos en una sola instrucción:

```
update usuario set nombre='MarceloDuarte', clave='Marce'
where nombre='Marcelo';
```

Clasificación de los DML

Se clasifican en dos grandes grupos de:

- lenguajes de consulta procedimentales

Lenguajes procedimentales. En este tipo de lenguaje el usuario da instrucciones al sistema para que realice una serie de procedimientos u operaciones en la base de datos para calcular un resultado final.

- lenguajes de consulta no procedimentales

En los lenguajes no procedimentales el usuario describe la información deseada sin un procedimiento específico para obtener esa información.

GUÍA DE ESTUDIO DE DE BASE DE DATOS

Unidad IV

Resultado de Aprendizaje

El estudiante será capaz de **analizar** y **crear** soluciones de bases de datos NoSQL adaptadas a necesidades específicas, utilizando herramientas tecnológicas actuales y aplicando criterios de escalabilidad y flexibilidad en el manejo de grandes volúmenes de datos.

BASES DE DATOS NO SQL

Introducción.
Características generales.
Tipos de Bases de datos no SQL.
Principales Diferencias con las Bases de datos SQL.
Ventajas y Desventajas.
Modelización de bases de datos no SQL.
Bases de datos no sql y su interacción en aplicaciones Informáticas.

DIAGRAMA DE APRENDIZAJE.

SÍNTESIS.

La Unidad 4 se centra en las bases de datos NoSQL, una alternativa a las bases de datos relacionales tradicionales, diseñada para manejar grandes volúmenes de datos no estructurados y aplicaciones que requieren alta escalabilidad y flexibilidad. Estas bases de datos se caracterizan por su capacidad para almacenar datos en formatos diversos, como documentos, pares clave-valor, columnas o grafos, permitiendo una estructura más adaptable y eficiente para ciertos tipos de aplicaciones. A diferencia de las bases de datos SQL, que utilizan un esquema fijo y relaciones tabulares, las bases de datos NoSQL ofrecen una mayor flexibilidad en el modelado de datos, lo que facilita su adaptación a necesidades cambiantes y estructuras de datos complejas. Entre las ventajas de las bases de datos NoSQL se incluyen su escalabilidad horizontal, rendimiento optimizado para grandes volúmenes de datos y flexibilidad en el esquema; sin embargo, también presentan desventajas, como la falta de estandarización y posibles inconsistencias en los datos. La modelización en bases de datos NoSQL implica diseñar estructuras que se ajusten a las necesidades específicas de la aplicación, aprovechando la flexibilidad que ofrecen estos sistemas. En el ámbito de las aplicaciones informáticas, las bases de datos NoSQL son especialmente útiles para gestionar datos en tiempo real, big data y aplicaciones que requieren alta disponibilidad y rendimiento.

Introducción.

Las bases de datos NoSQL han emergido como una respuesta a las limitaciones de las bases de datos relacionales tradicionales, especialmente en contextos que demandan alta escalabilidad,

flexibilidad y eficiencia en el manejo de grandes volúmenes de datos no estructurados o semiestructurados. A diferencia de las bases de datos SQL, que almacenan información en tablas con esquemas rígidos, las bases de datos NoSQL ofrecen modelos de datos más flexibles, permitiendo adaptarse a las necesidades específicas de aplicaciones modernas, como redes sociales, sistemas de recomendación y análisis de big data. Esta unidad explora las características generales de las bases de datos NoSQL, sus tipos principales, diferencias con las bases de datos SQL, ventajas y desventajas, modelización y su interacción en aplicaciones informáticas.

Tipos de Bases de datos no SQL

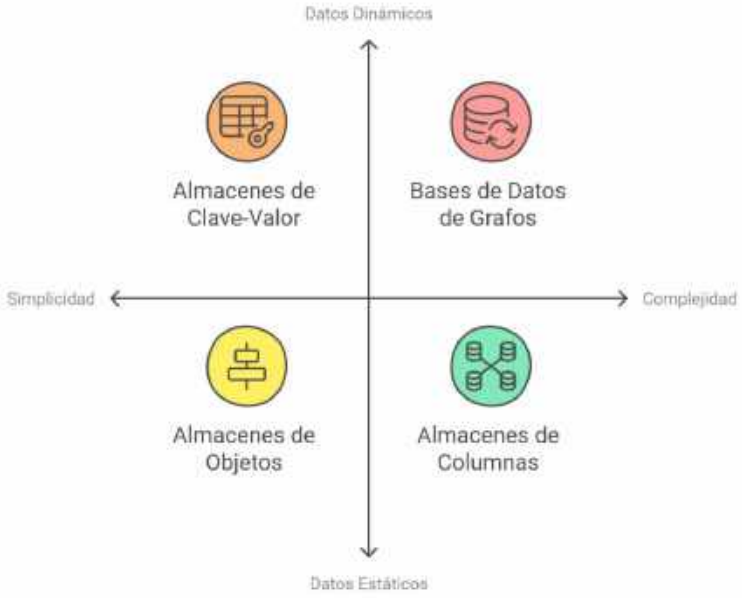


Fig. 22 *Tipos de bases de datos No SQL*

Las bases de datos NoSQL se han desarrollado para abordar las limitaciones de las bases de datos relacionales tradicionales, especialmente en aplicaciones que requieren alta escalabilidad, flexibilidad y manejo eficiente de grandes volúmenes de datos no estructurados o semiestructurados. A continuación, se detallan los principales tipos de bases de datos NoSQL:

1. Almacenes de Clave-Valor: Estos sistemas almacenan datos como pares clave-valor, donde cada clave es única y se asocia a un valor específico. Son altamente eficientes para operaciones de lectura y escritura simples y se utilizan comúnmente en aplicaciones que requieren respuestas rápidas, como cachés y sesiones de usuario.
Ejemplos: Redis, Amazon DynamoDB.

2. Bases de Datos de Documentos: Almacenan datos en documentos, generalmente en formatos como JSON, BSON o XML. Cada documento es autónomo y puede tener una estructura flexible, lo que facilita la adaptación a cambios en los requisitos de la aplicación. Son ideales para aplicaciones que manejan datos complejos y jerárquicos.
Ejemplos: MongoDB, CouchDB.

3. Almacenes de Columnas: Diseñados para manejar grandes volúmenes de datos distribuidos en múltiples servidores, almacenan datos en columnas en lugar de filas. Esto permite una compresión eficiente y un acceso rápido a columnas específicas, siendo útiles en aplicaciones de análisis de big data.

Ejemplos: Apache Cassandra, HBase.

4. Bases de Datos de Grafos: Especializadas en representar y consultar relaciones complejas entre datos, utilizan nodos, aristas y propiedades para modelar estructuras de grafos. Son adecuadas para aplicaciones como redes sociales, motores de recomendación y análisis de fraudes.

Ejemplos: Neo4j, Amazon Neptune.

5. Almacenes de Objetos: Permiten almacenar datos en forma de objetos, alineándose con la programación orientada a objetos. Cada objeto contiene datos y métodos, facilitando la integración con aplicaciones desarrolladas en lenguajes orientados a objetos.

Ejemplos: Db4o, ObjectDB.

6. Almacenes de Series Temporales: Optimizados para manejar datos que varían con el tiempo, como lecturas de sensores o datos financieros. Ofrecen operaciones eficientes para insertar y consultar datos basados en el tiempo.

Ejemplos: InfluxDB, TimescaleDB.

7. Almacenes de Datos en Memoria: Almacenan datos en la memoria principal para operaciones de lectura y escritura ultrarrápidas. Son útiles en aplicaciones que requieren baja latencia, como sistemas de comercio electrónico y juegos en línea.

Ejemplos: Memcached, Redis.

La elección del tipo de base de datos NoSQL adecuado depende de las necesidades específicas de la aplicación, incluyendo el tipo de datos, los patrones de acceso y los requisitos de escalabilidad y rendimiento.

Principales Diferencias

Las bases de datos SQL y NoSQL presentan diferencias fundamentales que las hacen adecuadas para distintos tipos de aplicaciones y necesidades. A continuación, se detallan las principales diferencias entre ambas:

Aspecto	SQL	NoSQL
Modelo de Datos	Relacional basado en tablas con esquemas predefinidos.	No relacional: documentos, clave-valor, columnas o grafos.
Esquema	Estructura rígida y predefinida.	Esquema flexible y dinámico.
Lenguaje de Consulta	SQL estándar para definición y manipulación de datos.	API o lenguajes específicos según el sistema.

Escalabilidad	Escalabilidad vertical (mejorando el hardware del servidor).	Escalabilidad horizontal (añadiendo más servidores).
Consistencia y Modelo	Garantiza consistencia estricta con el modelo ACID.	Sigue el teorema CAP, a menudo prioriza disponibilidad.
Integridad Referencial	Relaciones definidas entre tablas con claves primarias y foráneas.	No siempre soporta relaciones; manejadas en la lógica de la aplicación.
Optimización de Consultas	Ideal para transacciones complejas y consultas ad hoc.	Optimizado para grandes volúmenes y operaciones rápidas.
Casos de Uso	Sistemas financieros, ERP, CRM, aplicaciones estructuradas.	Redes sociales, big data, IoT, sistemas distribuidos.
Consistencia	Alta consistencia en transacciones.	Eventual consistencia en algunos casos.
Ejemplos	MySQL, PostgreSQL, Oracle Database, SQL Server.	MongoDB, Cassandra, Redis, Neo4j.

Tabla 17*. Diferencias entre SQL y No SQL.*

Ventajas y Desventajas

Las bases de datos NoSQL han ganado popularidad debido a su capacidad para manejar grandes volúmenes de datos y adaptarse a las necesidades de aplicaciones modernas. A continuación, se presentan sus principales ventajas y desventajas:

Ventajas:

1. **Escalabilidad Horizontal:** Las bases de datos NoSQL están diseñadas para escalar horizontalmente, permitiendo agregar más servidores al sistema para manejar incrementos en la carga de trabajo sin comprometer el rendimiento.
2. **Flexibilidad de Esquema:** No requieren un esquema fijo, lo que facilita la incorporación de nuevos tipos de datos y cambios en la estructura sin afectar la integridad de la base de datos.
3. **Alto Rendimiento:** Optimizadas para operaciones de lectura y escritura rápidas, son ideales para aplicaciones que demandan baja latencia y alta velocidad de procesamiento.
4. **Manejo de Datos No Estructurados:** Capaces de almacenar y procesar datos no estructurados o semiestructurados, como documentos, imágenes y registros de redes sociales, ofreciendo mayor versatilidad.
5. **Alta Disponibilidad y Tolerancia a Fallos:** Implementan replicación y distribución de datos, asegurando la disponibilidad continua y resistencia ante fallos del sistema.

Desventajas:

1. **Consistencia Eventual:** Al priorizar la disponibilidad y la partición, algunas bases de datos NoSQL pueden no garantizar una consistencia inmediata de los datos, lo que puede ser problemático para ciertas aplicaciones.

2. **Falta de Estandarización:** La ausencia de un lenguaje de consulta estándar como SQL puede dificultar la adopción y el aprendizaje, ya que cada sistema NoSQL puede tener su propia sintaxis y métodos de interacción.
3. **Limitaciones en Consultas Complejas:** No están optimizadas para realizar consultas complejas o uniones entre múltiples conjuntos de datos, lo que puede limitar su uso en aplicaciones que requieren análisis avanzados.
4. **Madurez y Soporte:** Algunas bases de datos NoSQL son relativamente nuevas y pueden carecer de la madurez, documentación y soporte que ofrecen las bases de datos relacionales tradicionales.
5. **Gestión de Transacciones:** No todas las bases de datos NoSQL soportan transacciones ACID completas, lo que puede ser una limitación para aplicaciones que requieren operaciones transaccionales estrictas.

Al evaluar la adopción de una base de datos NoSQL, es esencial considerar estas ventajas y desventajas en función de los requisitos específicos de la aplicación y el entorno operativo.

Modelización de bases de datos no SQL

La modelización de bases de datos NoSQL difiere significativamente de la de las bases de datos relacionales tradicionales, debido a la flexibilidad y diversidad de modelos de datos que ofrecen. A continuación, se detallan los aspectos clave a considerar en este proceso:

1. Comprensión del Modelo de Datos:

Fig. 23 *Comprensión del modelo datos*

Las bases de datos NoSQL se clasifican en varios tipos, cada uno con su propio modelo de datos:

- **Almacenes de Clave-Valor:** Almacenan datos como pares clave-valor, donde cada clave es única y se asocia a un valor específico. Son ideales para aplicaciones que requieren acceso rápido a datos mediante una clave única.
- **Bases de Datos de Documentos:** Almacenan datos en documentos, generalmente en formatos como JSON o BSON. Cada documento es autónomo y puede tener una estructura flexible, lo que facilita la adaptación a cambios en los requisitos de la aplicación.

- **Almacenes de Columnas:** Diseñados para manejar grandes volúmenes de datos distribuidos en múltiples servidores, almacenan datos en columnas en lugar de filas. Esto permite una compresión eficiente y un acceso rápido a columnas específicas, siendo útiles en aplicaciones de análisis de big data.
- **Bases de Datos de Grafos:** Especializadas en representar y consultar relaciones complejas entre datos, utilizan nodos, aristas y propiedades para modelar estructuras de grafos. Son adecuadas para aplicaciones como redes sociales, motores de recomendación y análisis de fraudes.

2. Diseño Basado en Consultas:
A diferencia de las bases de datos relacionales, donde el diseño se centra en la normalización y eliminación de redundancias, en NoSQL el diseño se orienta a las consultas que realizará la aplicación. Es esencial identificar los patrones de acceso y modelar los datos para optimizar el rendimiento en las operaciones más frecuentes.

3. Desnormalización y Duplicación de Datos:
En las bases de datos NoSQL, es común desnormalizar los datos, es decir, almacenar información redundante para optimizar el rendimiento de las consultas. Esta práctica reduce la necesidad de operaciones de unión (joins) y mejora la eficiencia en la recuperación de datos.

4. Consistencia y Disponibilidad:
Según el teorema CAP, las bases de datos distribuidas deben equilibrar consistencia, disponibilidad y tolerancia a particiones. En NoSQL, es común priorizar la disponibilidad y la tolerancia a particiones, aceptando una consistencia eventual. Es fundamental definir los requisitos de consistencia de la aplicación para elegir el modelo de datos adecuado.

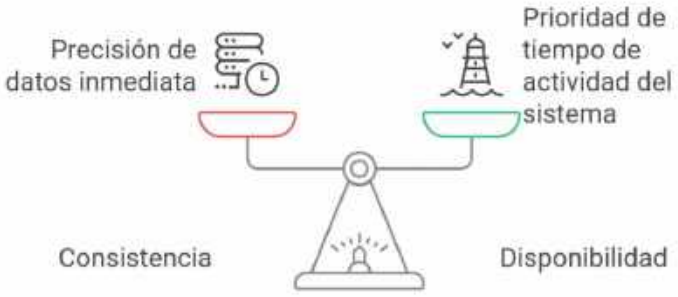


Fig. 24 *Disponibilidad de datos No SQL*

5. Modelado de Relaciones:
Aunque las bases de datos NoSQL no están diseñadas para manejar relaciones complejas como las relacionales, es posible modelar relaciones mediante técnicas como:

- **Incrustación (Embedding):** Incorporar documentos o datos relacionados dentro de un documento principal. Es útil cuando los datos relacionados se acceden juntos con frecuencia.

- **Referencias (Referencing):** Almacenar referencias a documentos relacionados mediante identificadores únicos. Se utiliza cuando los datos relacionados son grandes o se acceden por separado.

6. Herramientas y Prácticas de Modelado:

Existen diversas herramientas y metodologías para el modelado de datos en NoSQL. Por ejemplo, en MongoDB, se recomienda utilizar diagramas de entidad-relación adaptados al modelo de documentos, y considerar patrones de diseño específicos para optimizar el rendimiento y la escalabilidad.

El modelado de datos en bases de datos NoSQL requiere herramientas y prácticas específicas que se adapten a la flexibilidad y diversidad de estos sistemas. A continuación, se detallan algunas herramientas destacadas y prácticas recomendadas para un modelado efectivo:

Herramientas de Modelado para Bases de Datos NoSQL:

1. **NoSQL Workbench para Amazon DynamoDB:** Esta herramienta proporciona una interfaz visual para diseñar y modelar datos en DynamoDB. Permite crear modelos de datos nuevos o basados en existentes, visualizar relaciones y patrones de acceso, y realizar operaciones de importación y exportación de modelos.
2. **Archi:** Archi es una herramienta de modelado de arquitectura empresarial que admite la creación de modelos de datos para diversas bases de datos, incluidas las NoSQL. Ofrece una interfaz intuitiva y soporte para múltiples vistas y perspectivas del modelo de datos.
3. **DbSchema:** DbSchema es un diseñador y administrador de bases de datos visual que soporta bases de datos SQL, NoSQL y en la nube. Permite diseñar e interactuar visualmente con el esquema de la base de datos, trabajar en equipo e implementarlo en múltiples bases de datos.
4. **Hackolade:** Hackolade es una herramienta de modelado de datos especializada en bases de datos NoSQL y esquemas de almacenamiento en la nube. Ofrece capacidades de diseño inverso, generación de documentación y validación de esquemas para garantizar la coherencia y calidad del modelo de datos.

Prácticas Recomendadas para el Modelado de Datos en NoSQL:

1. **Diseño Orientado a Consultas:** A diferencia de las bases de datos relacionales, donde el diseño se centra en la normalización, en NoSQL es esencial modelar los datos según los patrones de acceso y las consultas que realizará la aplicación. Esto optimiza el rendimiento y la eficiencia en las operaciones más frecuentes.
2. **Desnormalización Controlada:** La desnormalización, o duplicación de datos, es común en NoSQL para reducir la necesidad de operaciones de unión (joins) y mejorar la velocidad de las consultas. Sin embargo, debe aplicarse con precaución para evitar inconsistencias y redundancias innecesarias.
3. **Uso de Patrones de Diseño:** Implementar patrones de diseño específicos para NoSQL, como la incrustación (embedding) y las referencias (referencing), ayuda a modelar relaciones entre datos de manera eficiente. La elección entre estos patrones depende de factores como el tamaño de los datos y la frecuencia de acceso conjunto.

4. **Consideración del Teorema CAP:** El teorema CAP establece que una base de datos distribuida no puede garantizar simultáneamente consistencia, disponibilidad y tolerancia a particiones. Es fundamental determinar cuáles de estos aspectos son prioritarios para la aplicación y modelar los datos en consecuencia.
5. **Validación y Pruebas:** Después de diseñar el modelo de datos, es crucial validarlo y probarlo con datos reales o simulados para identificar posibles problemas de rendimiento o inconsistencias. Las herramientas mencionadas ofrecen funcionalidades para realizar estas validaciones y ajustes necesarios.

La combinación de herramientas especializadas y prácticas de modelado adecuadas es esencial para diseñar bases de datos NoSQL que sean eficientes, escalables y alineadas con los requisitos específicos de cada aplicación.

Bases de datos no SQL y su Interacción en aplicaciones informáticas

Las bases de datos NoSQL han emergido como una solución eficaz para gestionar grandes volúmenes de datos y satisfacer las demandas de aplicaciones informáticas modernas. A continuación, se exploran las principales interacciones de las bases de datos NoSQL en diversas aplicaciones:

1. Aplicaciones Web y Móviles: Las aplicaciones que requieren alta disponibilidad y capacidad para manejar grandes cantidades de usuarios simultáneos, como redes sociales, plataformas de streaming y aplicaciones de mensajería, se benefician de la escalabilidad horizontal y la flexibilidad de esquema que ofrecen las bases de datos NoSQL. Por ejemplo, Facebook utiliza HBase para almacenar mensajes y datos de usuarios, aprovechando su capacidad para gestionar datos distribuidos.

Ejemplo: Una aplicación de mensajería como WhatsApp utiliza **Cassandra** para almacenar mensajes de texto y multimedia de usuarios. Gracias a su capacidad para escalar horizontalmente, puede manejar millones de mensajes diarios de manera rápida y eficiente.

2. Big Data y Análisis en Tiempo Real: Las bases de datos NoSQL son fundamentales en el procesamiento y análisis de grandes volúmenes de datos en tiempo real. Herramientas como Apache Cassandra y MongoDB permiten almacenar y consultar datos de manera eficiente, facilitando análisis rápidos y decisiones informadas en sectores como finanzas, salud y comercio electrónico.

Ejemplo: Una plataforma de análisis de redes sociales utiliza **MongoDB** para recopilar datos de publicaciones, comentarios y menciones en tiempo real. Estos datos se procesan para generar informes instantáneos sobre tendencias y análisis de sentimientos.

3. Internet de las Cosas (IoT): El IoT genera una cantidad masiva de datos provenientes de dispositivos conectados. Las bases de datos NoSQL, como Redis y Couchbase, son ideales para manejar la ingesta y procesamiento de estos datos debido a su baja latencia y capacidad para escalar horizontalmente. Por ejemplo, empresas de domótica utilizan bases de datos NoSQL para monitorear y controlar dispositivos en tiempo real.

Ejemplo: Un sistema de monitoreo de vehículos conectados utiliza **Redis** para almacenar datos de sensores en tiempo real, como velocidad, ubicación y estado del motor. Esto permite alertas instantáneas en caso de fallos o condiciones anómalas.

4. Sistemas de Recomendación: Plataformas como Netflix y Amazon implementan sistemas de recomendación personalizados basados en el comportamiento del usuario. Las bases de datos de grafos, como Neo4j, permiten modelar y analizar relaciones complejas entre usuarios y productos, mejorando la precisión de las recomendaciones.
Ejemplo: Un servicio de streaming como Spotify utiliza **Neo4j** para modelar relaciones entre usuarios, listas de reproducción y géneros musicales, generando recomendaciones personalizadas basadas en patrones de escucha.

5. Comercio Electrónico: Las tiendas en línea requieren gestionar catálogos de productos, carritos de compra y transacciones de manera eficiente. Las bases de datos NoSQL, como DynamoDB de Amazon, ofrecen alta disponibilidad y escalabilidad, asegurando una experiencia de usuario fluida incluso durante picos de tráfico.
Ejemplo: Un mercado en línea utiliza **DynamoDB** para gestionar datos de productos, carritos de compra y transacciones de usuarios, asegurando alta disponibilidad y tiempos de respuesta rápidos durante eventos de alto tráfico, como ventas flash.

6. Gestión de Contenido: Sistemas de gestión de contenido (CMS) que manejan diversos tipos de datos, como texto, imágenes y videos, se benefician de la flexibilidad de las bases de datos NoSQL. Por ejemplo, empresas de medios utilizan bases de datos orientadas a documentos para almacenar y servir contenido multimedia de manera eficiente.
Ejemplo: Un portal de noticias utiliza **CouchDB** para almacenar artículos y contenido multimedia en formato JSON, lo que permite a los editores acceder rápidamente a versiones actualizadas y adaptarse a múltiples dispositivos y plataformas.

7. Aplicaciones Financieras: Aunque las bases de datos relacionales son comunes en el sector financiero, las NoSQL se utilizan para análisis de fraude y gestión de riesgos, donde se requiere procesar grandes volúmenes de datos en tiempo real. Por ejemplo, instituciones financieras implementan bases de datos de grafos para detectar patrones sospechosos en transacciones.
Ejemplo: Un sistema de detección de fraudes bancarios utiliza **Amazon Neptune** para identificar patrones sospechosos en transacciones, modelando relaciones entre cuentas y transacciones para detectar actividades fraudulentas en tiempo real.

ELABORACIÓN, REVISIÓN Y APROBACIÓN DE PARES	
Profesor	
Ing. Juan Diego Rojas, Mg.	
Fecha de elaboración: 01/03/2024	
Comisión de revisión de pares de guías de estudio del Instituto Superior Tecnológico Tena	
Lcdo. Segundo Calisto Rochina Chileno	Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango	Ing. Agustin Gonzalo Guanipatin Ramirez
Fecha de revisión: 26/03/2024	
Coordinador de Investigación, Desarrollo Tecnológico e Innovación	
Mg. Danilo Alexander Zamora Núñez	
Fecha de aprobación: 03/04/2024	