Proyecto Compilando Conocimiento

Programación

Bases de Datos

Una Pequeña (Gran) Introducción

Autores:
Rosas Hernandez Oscar Andrés
Lopez Manriquez Angel

Índice general

Parte I

Parte Abstracta

Capítulo 1

Definiciónes

1.1. Repositorio de Datos

Son un conjunto de datos, donde definio a un dato como cualquier información que sea váliosa.

1.2. Base de Datos

Son un conjunto de datos interrelacionado definido por un modelo de datos, esto no lo tiene necesariamente un repositorio de datos. Así como progrmas que nos permitan acceder y manipular esa información.

Podemos definir de manera alterna como: Una colección de registros el cual es almacenada en una computadora de una forma sistemática (estructurada), de tal forma que un programa de computadora pueda consultarlo para responder consultas.

1.3. Proposito para una Base de Datos

Todo muy bien, pero ¿porque debería importarme un comino?¿Porqué preferir una base de datos sobre simplemente guardar los datos de manera "común"?

Antes de la aparición de los SGBD, las organizaciones normalmente almacenaban la información en Sistemas de Procesamiento de Archivos Típicos (Sistemas de Archivos).

Un sistema de archivos es un conjunto de programas que prestan servicio a los usuarios finales, donde cada programa define y maneja sus propios datos, los cuales presentan los siguientes inconvenientes:

- **Redundancia de datos e inconsistencia**: Ya que tu no vas a programar un sistema entero habra muchas maneras en que los demás programadores crearán aplicaciones y sobretodo en como van a guardar los datos.

  Peor aún, ¿Qué pasa cuando tengamos un motón de archivos con casi la misma información? Es decir cuando tengamos un montón de archivos con tu mismo número de telefono, con tu misma información de contacto.

- **Problemas para acceder a la información que queriamos en primer lugar**: Supongamos que queremos acceder a los datos, digamos que tenemos un montón de registros de sobre alumnos.

  ¿Como hariamos para tener todos los alumos que hayan reprobado? No hay forma fácil de hacerlo, incluso la forma mas "correcta" sería desarrollar un pequeño programa que se encargue de hacer lo que queremos.

  Y esto podría servir muy bien.... Hasta que necesitemos algo más. Entonces tenemos que estar haciendo programas y programas ¡Que cansado!

- **Aislamiento de la Información**: Ya que puede estar repartida por todos lados, no se exactamente por donde tengo que empezar a buscar. Y esto se vuelve un verdadero desastre cuando intentamos modificar la información guardada.

- **Problemas con Integridad**: Los valores de los datos almacenados en la base de datos, deben satisfacer ciertos tipos de restricciones de consistencia.

  Por ejemplo, el saldo de cierto tipo de cuentas bancarias no pueden ser nunca inferior a una cantidad predeterminada, digamos 4000 dolares.

  Los desarrolladores deben cumplir estas restricciones en el sistema añadiendo el código correspondiente en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan.

- **Problemas con Atomicidad**: Si ocurriera un problema en el sistema al momento de estar modificando la información me gustaria que al volver a arrancar todo, este debería regresar a un punto de respaldo.

  Como si el sistema fallara justo al hacer una transacción bancaria, me interesa que se haya realizado o no, pero que no le haya quitando dinero a una cuenta pero no se la haya dado al otro cliente.

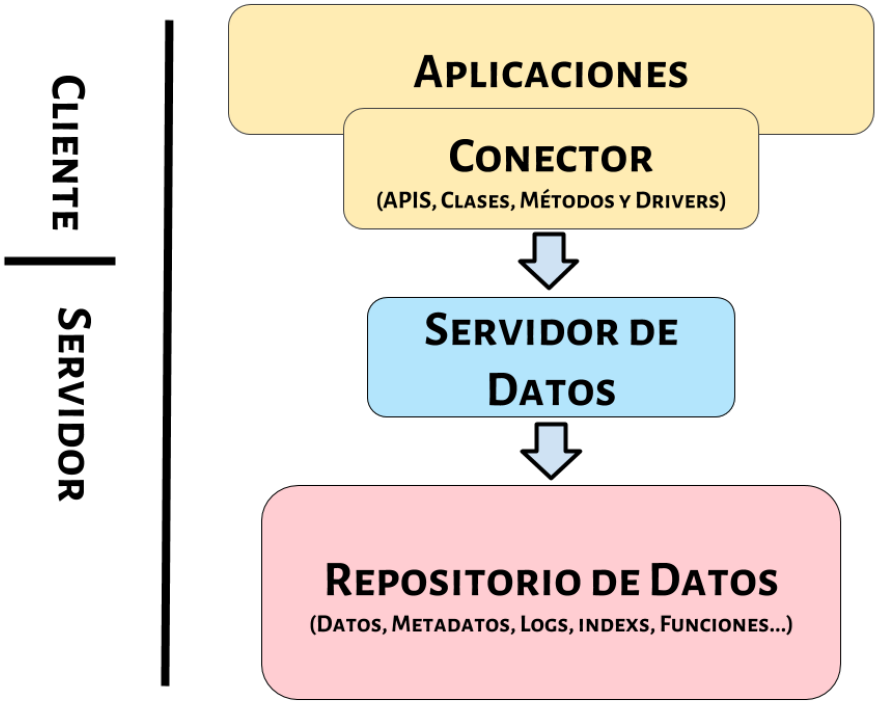
- **Errores del Acceso Concurrente**: Para aumentar el rendimiento global del sistema y obtener una respuesta más rápida, muchos sistemas permiten que varios usuarios actualicen los datos simultaneamente. En realidad hoy en día, los principales sitios de comercio electrónico en internet pueden tener millones de accesos diarios de compradores a sus datos. En tales entornos es posible la interacción de actualizaciones concurrentes y puede dar lugar a datos inconsistentes.

- **Problemas de seguridad**: No todos los usuarios de un sistema de base de datos deben tener acceso a todos los datos. Ya que los programas de aplicación se añaden al sistema de procesamiento de datos de un forma adhoc, es difícil hacer cumplir tales restricciones de seguridad.

Podemos resumir esto con que a diferencia de un sistema de archivos, una base de datos busca:

- Evitar o miniza la redundancia
- Evitar inconsistencias en los datos
- Eliminar inconsistencias en los datos
- Comunicacion con distintos repositorios de datos
- Control de concurrencia

1.4. Elementos de un Sistema de Base de Datos



- **Aplicaciones:** Es la interfaz entre la base de datos y el usuario, estas pueden ser desarrolladas por un lenguaje de alto nivel
- **Conector:** Son los componentes que permiten el enlace entre el SGBD y las interfaces desarrolladas en un lenguaje de programación, estas contienen las clases y/o funciones necesarias para llevar a cabo la comunicación entre las aplicaciones con el Sistema Gestor de Base de Datos.
- **Sistema Gestor de Base de Datos:** Son el software especialízado que nos permite manipular inteligentemente nuestro datos:

  Es la aplicación que permite a los usuarios definir, crear y mantener la base de datos y proporciona acceso controlado a la misma.

  - Creación de Repositorios
  - Creación de Cuentas de Usuarios
  - Se encarga de crear archivos lógicos, físicos y objetos de la BD.
  - Se encarga de administrar las transacciones, bloqueos , etc.

1.5. Abstracción de los Datos

- **Nivel Físico**:

  Es el nivel base de abstracción que describe como es que la información es guardade de manera actual. Esto nos permite describir como de complejo es que son las estructuras en la realidad.

- **Nivel Lógico**:

  Es el nivel que nos muestra como es que se almacena la información dentro de la base de datos y como es que se da la relaciones entre la información.

  Aunque la implementación de las estructuras simples en el nivel lógico puede implicar complejas estructuras de nivel físico, el usuario de este nivel no necesita ser consciente de esta complejidad. Esto se conoce como independencia de datos físicos. Los administradores de bases de datos, que deben decidir qué información debe conservar en la base de datos, utilizan el nivel lógico de abstracción.

- **Nivel Visual**:

  El nivel más alto de abstracción describe sólo una parte de la base de datos completa.

  Aunque el nivel lógico utiliza estructuras más simples, la complejidad se mantiene debido a la variedad de información almacenada en una base de datos grande.

  Muchos usuarios del sistema de base de datos no necesitan toda esta información, sólo necesitan acceder a una parte de la base de datos. El nivel de vista de la abstracción existe para simplificar su interacción con el sistema.

1.6. Usuarios de una Base de Datos

Hay cuatro grupos de personas que intervienen en el entorno de un sistema de base de datos: el administrador de la base de datos, los diseñadores de la base de datos, los programadores de aplicaciones y los usuarios.

- **Diseñadores de la Base de Datos**
  - Encargado de grabar los propios Módelos de Datos
  - Esquema de la Base de Datos
  - Diseño lógico de la Base de Datos
- **Administrador de la Base de Datos**

  Encargado de:
  - Monitorear el Performance
  - Diseño físico de la base de datos y de su implementación.
  - Herramientas Administrativas
    - Creacion de cuentas de usuarios
    - Objetos accedidos.
    - Matriz de autorizacion.
  - Definir tiempos de respaldo
  - Reorganización físico
  - Llevar a cabo las tecnicas de recuperación
- **Programadores de Aplicaciones**

  Son los que se encargan de implementar los programas de aplicación que servirán a los usuarios finales. Estos programas son los que permiten consultar datos, insertarlos, actualizarlos y eliminarlos.
  - Interfaces de los usuarios finales
    - Facilitar el acceso a ciertos .ºbjetos"de la BD
  - Interfaces para la gestión de la aplicaciones
    - Operaciones de escritura sobre altas ó bajas
    - IDE desarrollo (Java, .NET)
    - Lenguaje Scripts
    - Conectividad servidores de datos (API s)

1.6.1. Usuarios Finales

Los usuarios finales son los clientes de la base de datos, son las personas que requieren acceso a la base de datos para realizar consultas, actualizaciones e informes. Los usuarios se pueden clasificar en varias categorías:

- **Casuales** Estos acceden ocasionalmente a la base de datos, pero pueden necesitar una información diferente en cada momento.
  - Data Minning
  - Big Data
- **Principiantes - Paramétricos** Constituyen una parte considerable de los usuarios finales de los sistemas de bases de datos. Su labor principal gira entorno a la consulta y actualización constantes de la BD.
- **Sofisticados**
  - Experiencia en aplicaciones
  - Programadores de aplicación
  - DBA
  - Investigadores
- **Independientes (Stand Alone)**
  - Sistemas Escolares
  - Sistemas de prueba
  - Sin conectividad a otros nodos

1.7. DML vs DDL

- **DML**

  Data Manipulation Languaje, es un lenguaje que nos permite modificar los datos guardados.

  Los tipos de acceso que tenemos disponibles son:

  - Recuperar información
  - Insertar nueva información
  - Eliminar la información
  - Modificación de la información

  Una consulta o query es una sentencia que solicita la recuperación de información. La parte de un DML que implica recuperación de información se denomina lenguaje de consulta.

  Aunque técnicamente incorrecto, solemos utilizar los términos lenguaje de consulta y lenguaje de manipulación de datos como sinónimos.

- **DDL**

  Data Definition Languaje, es Lenguaje de definición de datos.

  Especificamos un esquema de base de datos mediante un conjunto de definiciones expresadas por un lenguaje especial denominado DDL.

  El DDL también se utiliza para especificar propiedades adicionales de los datos.

Podemos simplificar en el sentido de SQL que:

DDL: Data Definition Language, mediante el cual puede definir nuevos objetos de base de datos, como Table, Views, Stored Procedures, etc. Algunos comandos comunes son:

- CREATE
- ALTER
- DROP
- etc...

DML: Data Manipulation Language, mediante el cual puede realizar cambios en los objetos creados anteriormente por DDLs. Algunos comandos comunes son:

- INSERT
- UPDATE
- DELETE
- SELECT
- etc...

1.8. Arquitectura del Sistema Gestor

Podemos acceder al Sistema Gestor de nuestra Base de Datos de muchas maneras, desde formularios web, aplicaciones de escritorio e interpretes de SQL, todos se comunican con la Base de Datos mediante SQL.

- **Motor de Evaluación de Consultas**
  - Analizador: Este se encarga de analizar las sentencias SQL a nivel sintactico y lexico, así como una validación de que existan dichas relaciones y atributos. Es lo que mucha gente conoce como un compilador de DDL ó DML.
  - Evaluador de Operaciones: Este se encarga de crear el árbol canónico, es decir, la forma formal que tendría el árbol necesario para acceder a la información.
  - Optimizador: Se encarga de tomar el árbol canónico y optimizarlo para que se tenga que hacer la menor cantidad de operación a nivel lógico, este se conoce como árbol de consulta.
  - Ejecutor de Planes: Se encarga de planear la mejor estrategia para ejecutar la consulta de tal manera que se optimize de manera física.

- **Gestores**

  - Gestor de Transacciones: Este es el que se encarga de organizar varias sentencias SQL para ejecutarlas como una transacción.
  - Gestor de Bloqueos: Este es el que se encarga de ver si es que cierta relación esta bloqueada porque esta ocurriendo una transacción en ese momento.

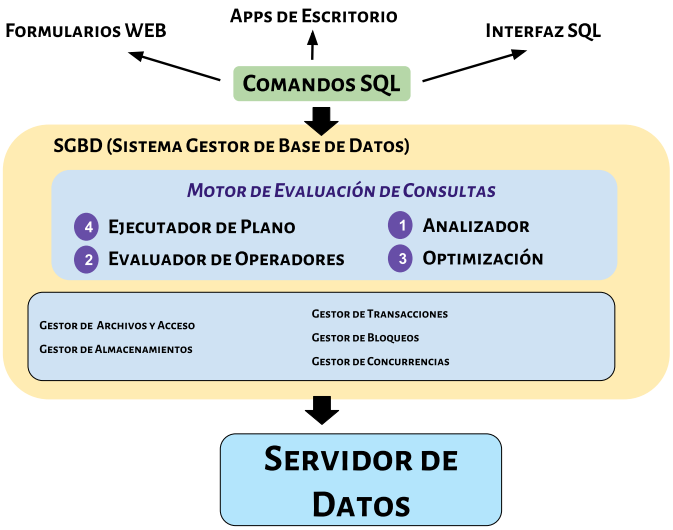
    Los bloqueos son una parte muy importatne de este gesto.

    Podemos definirlos en principalmente 2:

    - Binario: Es decir permite que una relación este o bine bloqueada o no para lectura o escritura.
    - Múltiplos Niveles: Es decir nos permite que existan bloqueos de lectura, escritura diferentes, ayudando a evitar las esperas si no son necesarias.
  - Gestor de Recuperación: Este es el que se encarga de definir todas las técnicas de recuperación sobre la base, estas mismas las podemos definir en dos:

    - Inmediatas: Son las que podemos representar como rollback o rollfoward
    - Técnicas en Frío: Usa una copia de seguridad
  - Gestor de Archivos y Métodos de Acceso Todos los elementos, metadatos, logs, funciones, etc... tienen que estar almacenada de manera física.

    Este se encarga de dependiendo de las querys como es que debe acceder de manera eficiente a los datos.
  - Gestor de Memoria Intermedia Es como una RAM, nos permite guardar consultas bastante frecuentes para aumentar la velocidad sobre la RAM



1.9. Arquitecturas de las Bases de Datos

- **Cliente - Servidor**

  Mientras que el servidor es aquel que contiene toda la información importante, el cliente solo se encarga de solicitar servicios.

  Podemos seperar estos como:

  - Cliente sin Disco: Aquel que puede acceder a la información pero no puede modificarlo.
  - Cliente con Disco: Aquel que puede acceder a la información y puede modificarlo.

- **Multibase**

  Esta se caracterisa porque cada base de datos interna que tienes no se relaciona entre si, es decir, no hay comunación entre ellas.

Capítulo 2

Sistema Entidad-Relación

2.1. ¿Qué es?

Fue creado por Peter Pin-Shan Chen en 1976, llamado "The Entity Relationship Model Toward Unified View of Data"

Es un modelo conceptual, es decir, creado en lenguaje natural y se se ayuda de DER (gráficas) para hablar de las relaciones.

Permite construir el modelo conceptual de datos.

- Es una representación de la estructura y contenido de una base de datos.
- Es independiente del software (como el SMBD).
- Permite establecer las restricciones de la BD.
- Esta asociado al modelo de datos que es usado para implementar la BD.

Siendo una publicado por la ACM (Association for Computing Machinery), considera los siguientes puntos:

- Incorpora información semántica del mundo real.
- Introduce una técnica gráfica como herramienta para el diseño de bases de datos (Diagrama Entidad Relación).
- Es una representación lógica de los datos de una organización o un área de negocio.
- Normalmente es expresado como un DER, siendo este la representación gráfica del Modelo ER.

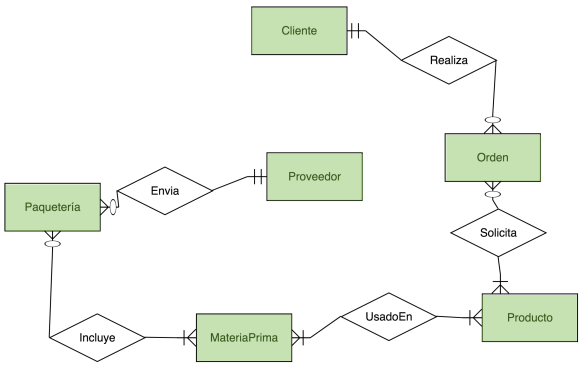


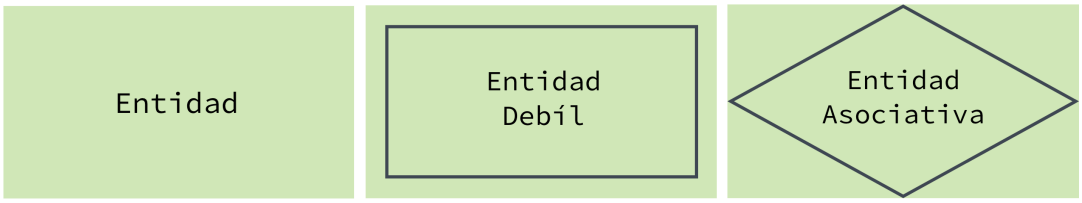
Figura 2.1: Ejemplo de un Diagrama Entidad-Relación

2.2. Entidades

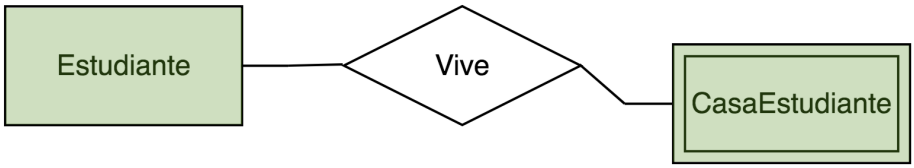
Son objetos que existen en el mundo real y que son distingubles (gracias a sus características) de otros objetos.

El tipo de entidad es el esquema que tiene la entidad en la base de datos, es decir son todos los atributos o características que definien a la entidad.

Piensa en las entidades como sustantivos. Ejemplos: cliente, estudiante, automóvil o producto.



- **Fuerte:**
  Es aquella entidad cuya existencia no depende de otras entidades.
- **Debíl:**
  Es aquella entidad cuya existencia depende de otras entidades.
  - Esta depende de de una o varias entidades fuertes
  - Una entidad debíl jamas se debe asociar con una entidad debíl
  - Una entidad debíl no tiene identificador propio, pero si parciales
  - Usa una relación identificada para asociar una entidad fuerte con una debíl



- **Asociativa:**
  Se usa cuando tienes una relación y resulta que quieres usar esta relación como conexión (por ejemplo la relación entre alumno y clase) así que haces esa relación una entidad asociativa (por ejemplo para asociar esta nueva entidad con profesor).

  Relación Identificada: Es la relación entre una entidad débil y el identificador propio.

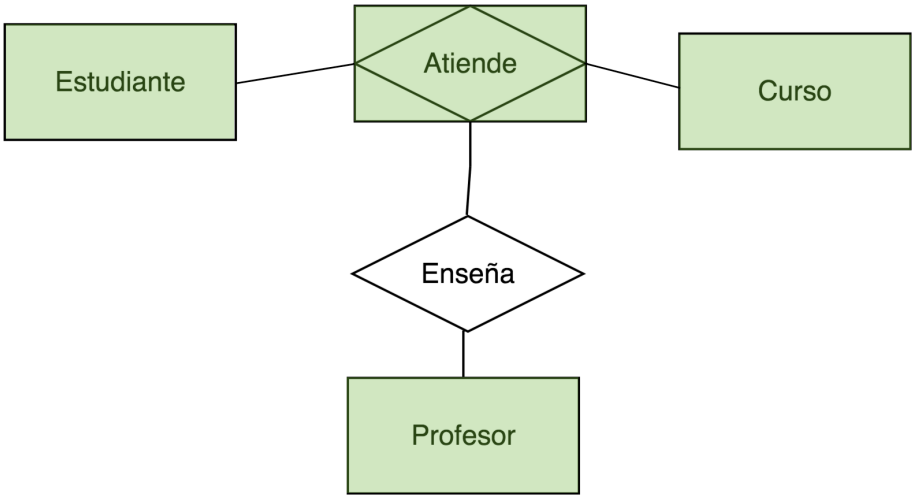


Figura 2.2: Ejemplo

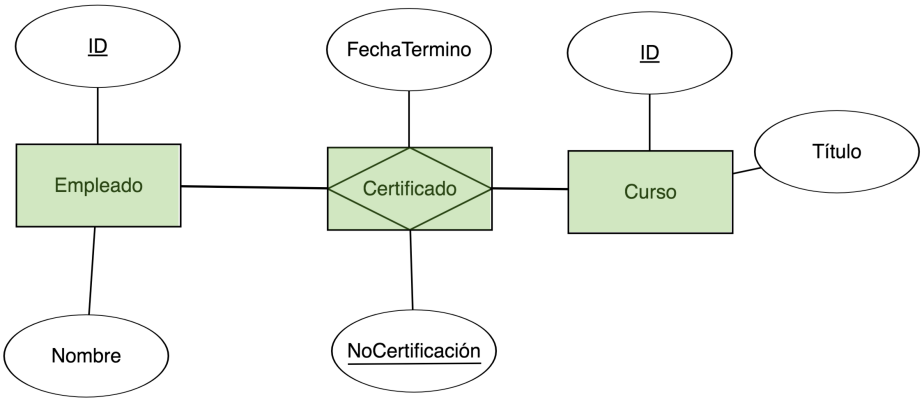
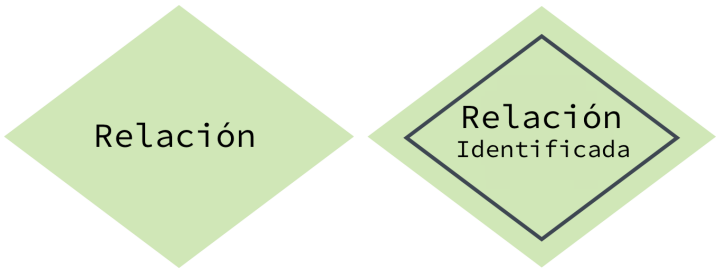


Figura 2.3: Otro Ejemplo

2.3. Relaciones

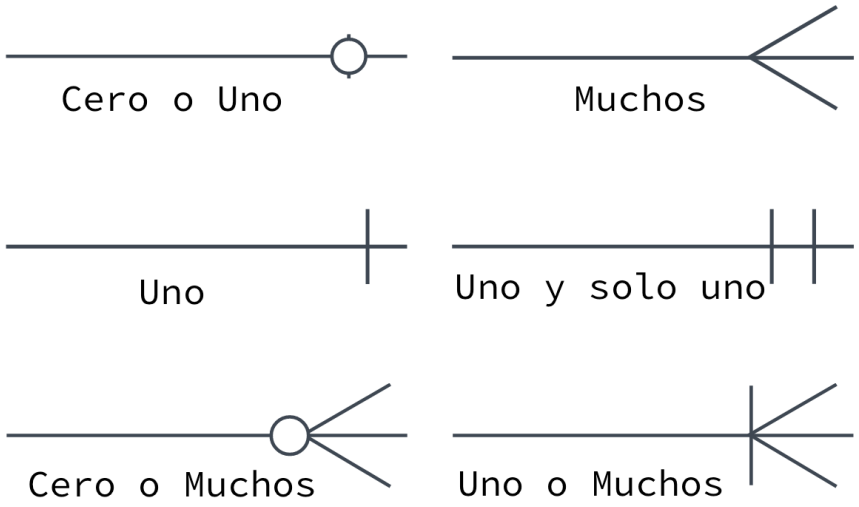
Esto es como las entidades actúan unas sobre otras o están asociadas entre sí. Piensa en las relaciones como verbos.

Por ejemplo, el estudiante nombrado puede registrarse para un curso. Las dos entidades serían el estudiante y el curso, y la relación representada es el acto de inscribirse, conectando las dos entidades de esa manera.



2.3.1. Cardinalidad

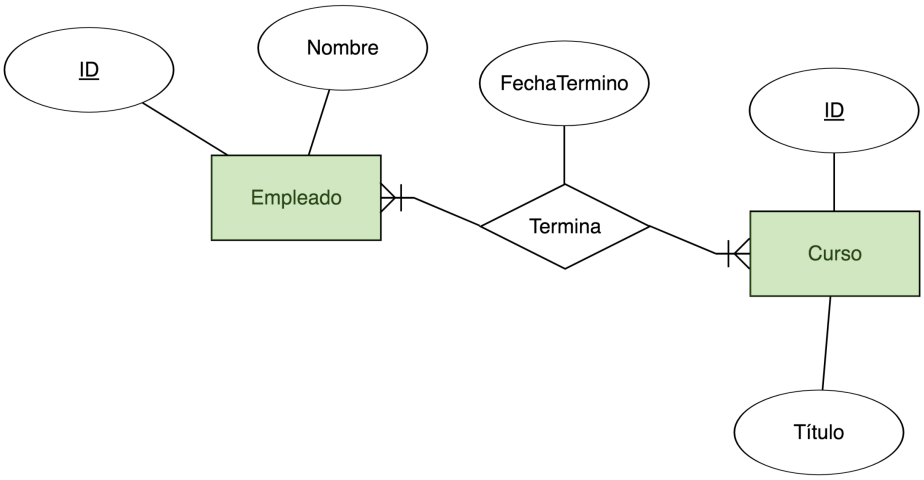
Nos dice la cantidad de atributos que se relacionan.



2.3.2. Atributos en las Relaciones

Atributos en relaciones: Los atributos pueden ser asociados con relaciones como una entidad.

En este caso, se requiere almacenar la Fecha (mes y año) cuando un empleado completa un curso.

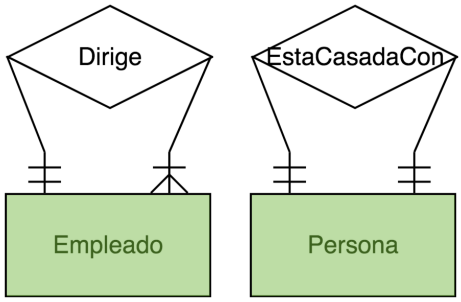


2.3.3. Grado de una Relación

El número de entidades que participan en una relación. Las Relaciones de acuerdo al grado de relación

Relaciones Unarias:

Cuando una única relación interviene en la asociación

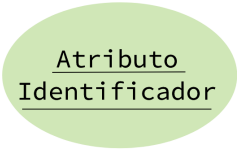


2.4. Atributos

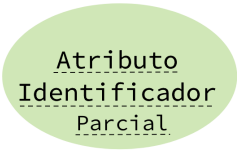
Son las propiedades o características de una entidad

- **Simples:** Son aquelllas propiedades atomicas que ya no se pueden subdividir

- **Identificador:** Es aquel atributo que nos permite identificar de forma unica cada una de las instancias de la entidad.



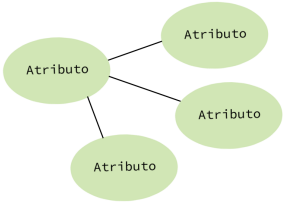
- **Identificador Parcial:**



- **Derivado:** Es aquel atributo cuyo valor puede ser calculado a partir de otros valores de atributos (posiblemente de datos que no se encuentren en la BD's, tal como, la fecha del sistema, etc)

- **Multivalor:** Aquellos que tienen mas de un valor

- **Compuestos:** Aquellos que estan formados por varios subatributos

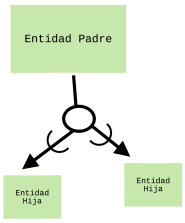


Capítulo 3

Sistema Entidad-Relación Extendido

3.1. Introducción

El Modelo de Relación de Entidad Extendida es un modelo más complejo y de más alto nivel que extiende un diagrama E-R para incluir más tipos de abstracción y para expresar más claramente las restricciones.



Todos los conceptos de un diagrama E-R están incluidos en el modelo EE-R.

Vamos a ver las diferencias más importantes con respecto a nuestro clásico amigo el diagrama entidad relación.

- **Herencia: Super y Subclases**

  Como el nombre sugiere, podemos tener entidades que consideramos padres y que heredan a otras entidades hijas.

  Por ejemplo podemos podemos tener una entidad empleado puede tener subentidades que se dividen por el trabajo que hacen, por ejemplo diseñadores, gerentes y ayudantes.

  Todas nuestras subentidades van a heredar las propiedades o atributos de nuestra superentidad.

  Esto nos permite que nuestra entidad padre sea una idea general y que nuestras subclases sean especializaciones de la misma, por esto decimos que esta clase de diagramas permite la especialización.

  Es también importante remarcar que alguna de nuestras entidades puede heredar de una o mas entidades

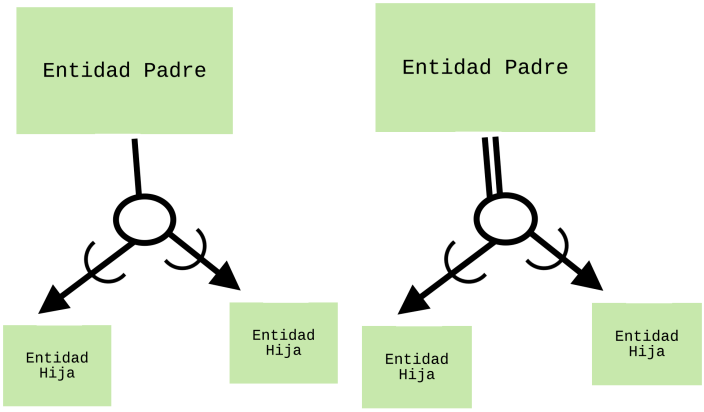
- **Generalización y Especialización**

  Como habiamos hablando antes, esto nos permite que si tenemos una gran cantidad de entidades con atributos comunes podemos crear una entidad general y hacer que las demás hereden de ella, con esto también propiciamos que nuestras entidades que heredan se puedan especializaciar.

- **Restricciones** Tambien es común que se les conozca como superclass constraint
  - Entidades Disconjuntas
  - Entidades Superpuestas
- **Discriminadores de Superentidad**

  Un discriminador de entidades es un atributo que indica el subtipo de una entidad.

  Es decir podemos indicar con este diagrama si las divisiones que hacemos son parciales (es decir, es posible que una instancia de la sueprclase pueda existir sin tener que ser parte de una subentidad) o total en la que decimos que cualquier instancia tiene que pertencer a alguna subentidad



Capítulo 4

Sistema Relacional

4.1. Introducción

El modelo Relacional fue introducido en 1970 por E.F. Codd.

La mayoría de los sistemas de bases de datos utilizados hoy en día son relacionales.

4.2. Definiciones Básicas

En el modelo relacional, los datos se dividen en diferentes **tablas**. Una tabla funciona como una matriz o una hoja de cálculo.

- **Relación**: Una tabla con columnas y filas
- **Atributo**: Nombre de una columna de una relación
- **Dominio**: Es el conjunto de valores legales de el atributos
- **Tupla**: Es una fila de una relación
- **Grado de una Relación**: Es el número de atributos que contiene la relación
- **Cardinalidad de una Relación**: Es el número de tuplas que contiene

4.3. Relaciones

Para evitar la redundancia de datos la clave esta en crear diversas tablas e ir relacionandolas.

Las relaciones tienen las siguientes características:

- Cada tupla dentro de la tabla tiene que ser única
- El orden de las tuplas no importa
- Cada relación tiene un nombre único en la Base de Datos
- Los atributos de cada tabla tienen que tener un nombre único en la tabla
- No importa el orden de los atributos
- Los valores de los atributos son atómicos en cada tupla, es decir, cada atributo toma un solo valor. Se dice que las relaciones están normalizadas.

Valores Nulos

Cuando en una tupla un atributo es desconocido, se dice que es nulo.

Un nulo no representa el valor cero ni la cadena vacía, éstos son valores que tienen significado.

El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

4.3.1. Llaves Relaciones

Para lograr esto tenemos que crear relaciones, y es imposible hablar de relaciones sin hablar de llaves:

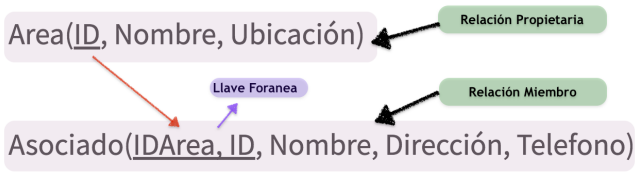
- **Primary Key:** La idea de una llave primaria es crear un atributo (o un conjunto de atributos) que tiene, tiene pero tiene que ser único y es recomendable que casi no cambie.

  Una llave compuesta es una llave primaria que consiste de más de un atributo

  Asociado(ID, Nombre, Apellido1, Apellido2, Salario)

  Socio(ID, Email, Nombre, Dirección, Telefono)

- **Foreign Key:** La idea de una llave secundaria es crear un atributo (o mas comunmente varios atributos) que en alguna otra tabla sea la llave primaria. Eso es todo.



4.3.2. Reglas de Integridad

Una vez definida la estructura de datos del modelo relacional, pasamos a estudiar las reglas de integridad que los datos almacenados en dicha estructura deben cumplir para garantizar que son correctos.

El definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos para cada atributo. A este tipo de restricciones se les denomina restricciones de dominios.

Hay además dos reglas de integridad muy importantes que son restricciones que se deben cumplir en todas las bases de datos relacionales y en todos sus estados o instancias (las reglas se deben cumplir todo el tiempo).

- **Ninguno de los atributos que componen la Llave Primaria puede ser nunca nulo**

  Por definición, una clave primaria es un identificador irreducible que se utiliza para identificar de modo único las tuplas, si se permite que parte de la clave primaria sea nula, se está diciendo que no todos sus atributos son necesarios para distinguir las tuplas, con lo que se contradice la irreducibilidad.

  Nótese que esta regla sólo se aplica a las relaciones propietarias

- **Si en una relación hay alguna Llave Foránea, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos**

4.3.3. Reglas de Negocio

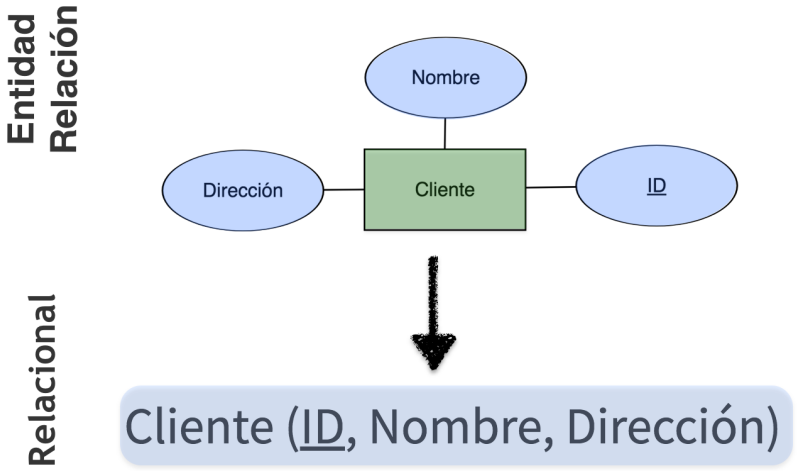
Además de las dos reglas de integridad anteriores, los usuarios o los administradores de la base de datos pueden imponer ciertas restricciones específicas sobre los datos, denominadas reglas de negocio.

Por ejemplo, si en una oficina de la empresa inmobiliaria sólo puede haber hasta veinte empleados, el SGBD debe dar la posibilidad al usuario de definir una regla al respecto y debe hacerla respetar. En este caso, no debería permitir dar de alta un empleado en una oficina que ya tiene los veinte permitidos.

4.4. Normalización

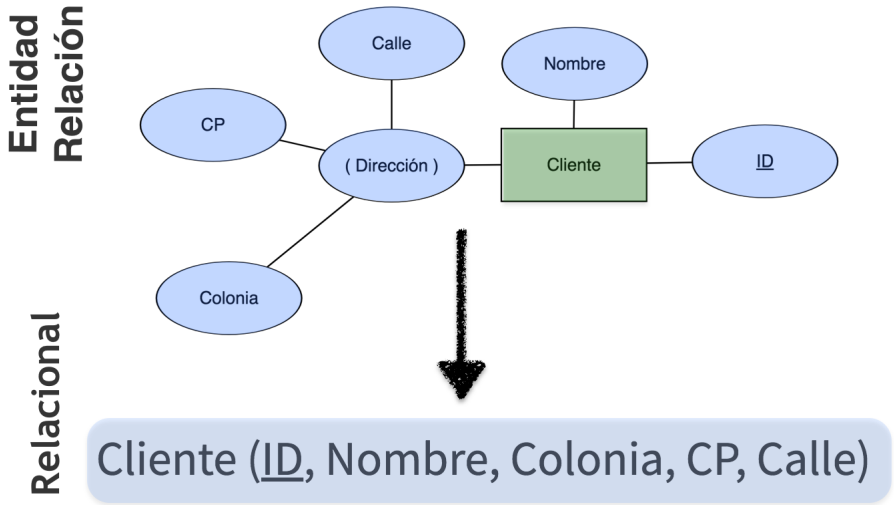
4.4.1. Mapear Entidades

- Cada entidad en un Diagrama Entidad Relación (DER) es transformado en una Relación.
- Cada atributo en una entidad se convierte en atributo de la relación.
- El atributo identificador de la entidad se convierte en la llave primaria de la relación correspondiente.



4.4.2. Mapear Atributos Compuestos

Cuando una entidad tiene atributos compuestos, solo los componentes del atributo compuesto son incluidos en la nueva relación.

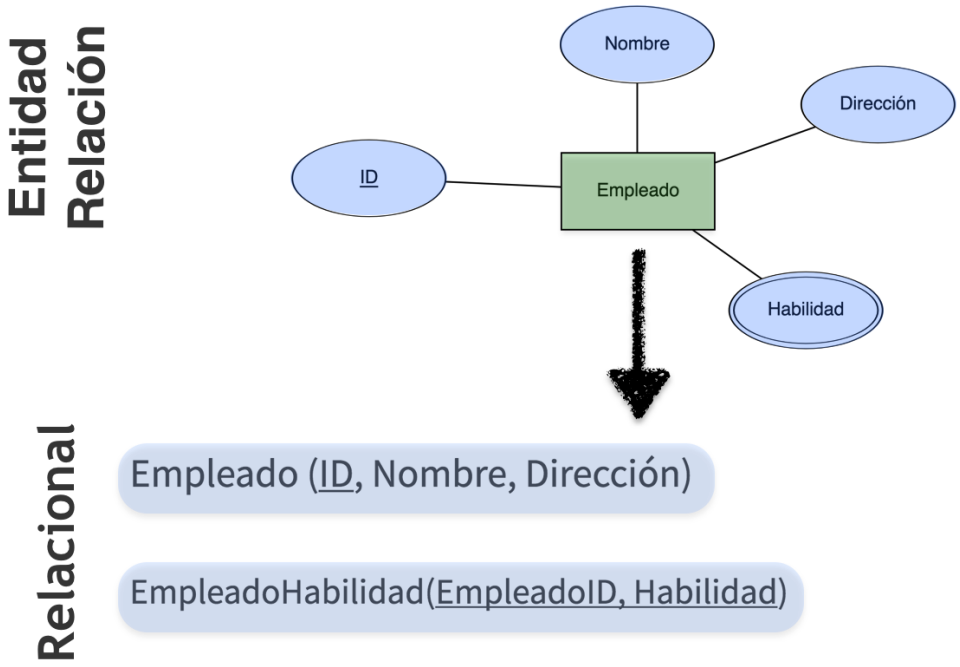


4.4.3. Mapear Atributos MultiValor

Cuando una entidad contiene atributos multivalor, dos nuevas relaciones son creadas.

- La primera relación contiene todos los atributos de la entidad excepto el atributo multivalor.
- La segunda relación contiene dos atributos que forman la llave primaria (llave primaria compuesta) de la segunda relación.

  El primero de estos atributos es la llave primaria de la primera relación, la cual se convierte en la llave foránea en la segunda relación. El segundo atributo es el atributo multivalor. El nombre de la segunda relación debe ser representativo del atributo multivalor.

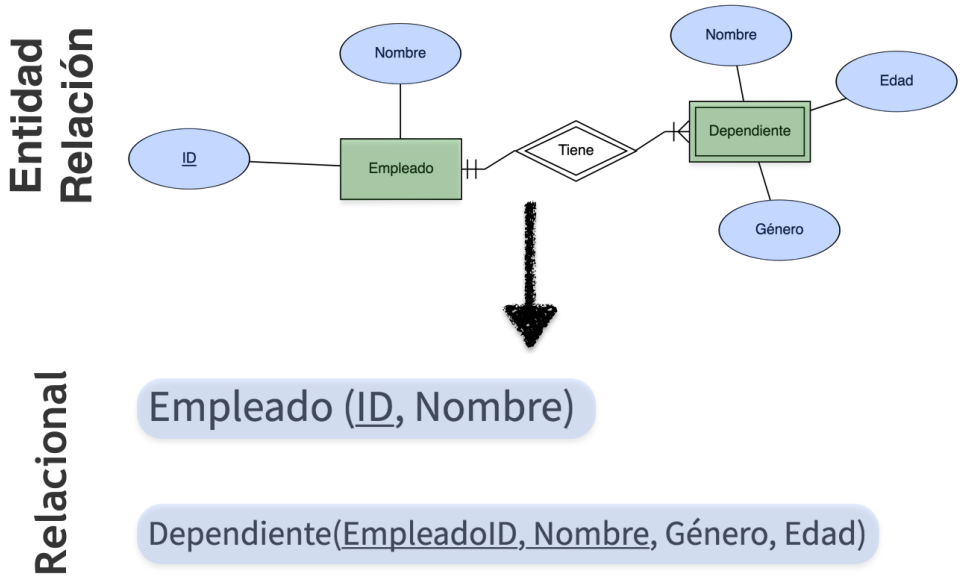


4.4.4. Mapear de Entidades Débiles

Por cada par de Entidades fuertes y debiles tenemos que crear dos entidades:

- Se crea una nueva relación basada en la entidad fuerte e incluye todos los atributos simples (o atributos simples de atributos compuestos) como atributos de esta relación.
- Se crea otra relación basada en la entidad debíl, esta incluye la llave primaria de la entidad fuerte o relación identificada como llave foránea en esta nueva relación.

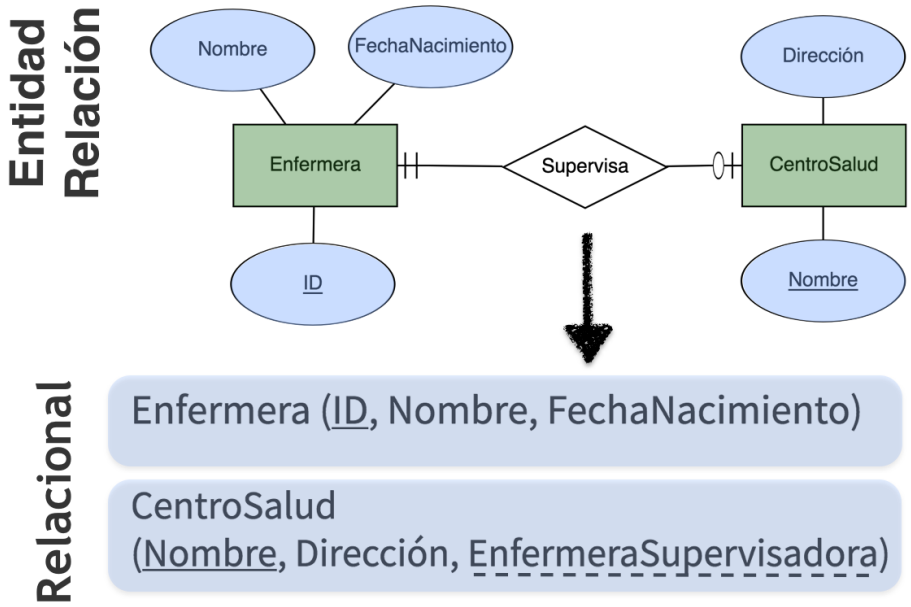
  La llave primaria de la nueva relación es la combinación de la llave primaria de la entidad fuerte y el identificador parcial de la entidad débil.



4.4.5. Mapear Relaciones Binarias: Uno a Uno

Para una relación binaria uno a uno (1:1) se debe crear 2 relaciones, una para cada entidad involucrada.

- La llave primara de una de las relaciones resultantes es incluida como llave foránea de la otra entidad.
- Si la relación binaria es opcional en una dirección y obligatoria hacia la otra dirección (como se muestra en el ejemplo), la llave foránea debe de ir del lado de la relación opcional, esto para evitar valores nulos.

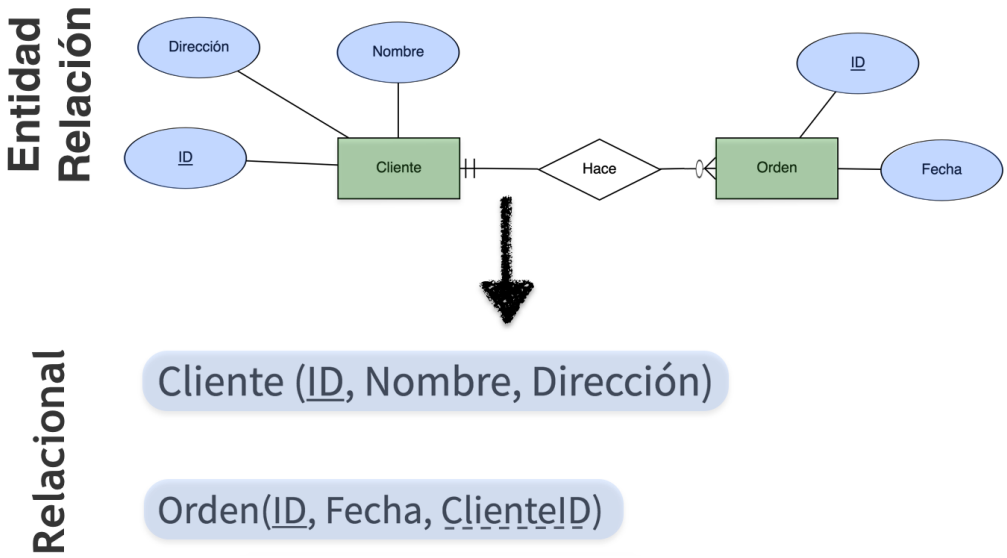


4.4.6. Mapear Relaciones Binarias: Uno a Muchos

Para cada relación uno a muchos, primero se crea una relación para cada una de las dos entidades participantes en la relación, usando el procedimiento para mapear entidades.

Se debe incluir la llave primaria de la entidad del lado de la cardinalidad uno como llave foránea en la relación que está del lado de cardinalidad muchos de la relación.

En otras palabras, la llave primaria se pasa como llave foránea al lado de los muchos.

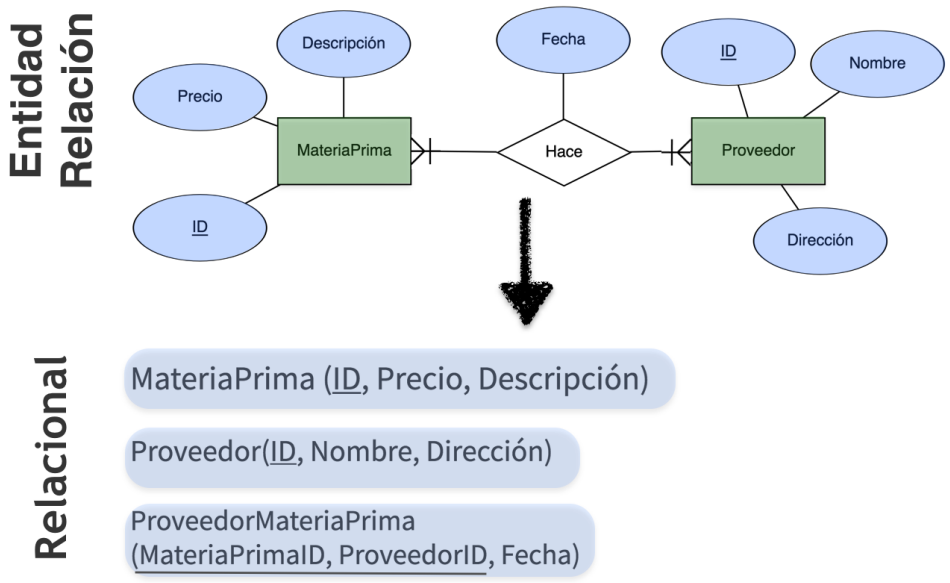


4.4.7. Mapear Relaciones Binarias: Muchos a Muchos

Suponga la relación entre dos entidades A y B. Para hacer el mapeo se deben generar 2 relaciones una para cada una de las entidades (A y B) y una adicional para la relación M:N entre las entidades, que llamaremos C.

Esta nueva relación C tendrá estas carácteristicas:

- Se debe incluir como llaves foráneas de la relación C, la llave primaria de cada una de las dos entidades (A y B).
- Estos mismos atributos (llaves foráneas) se convierten en la llave primaria de la relación C, cualquier atributo asociado a la relación M:N se debe incluir en la relación C.



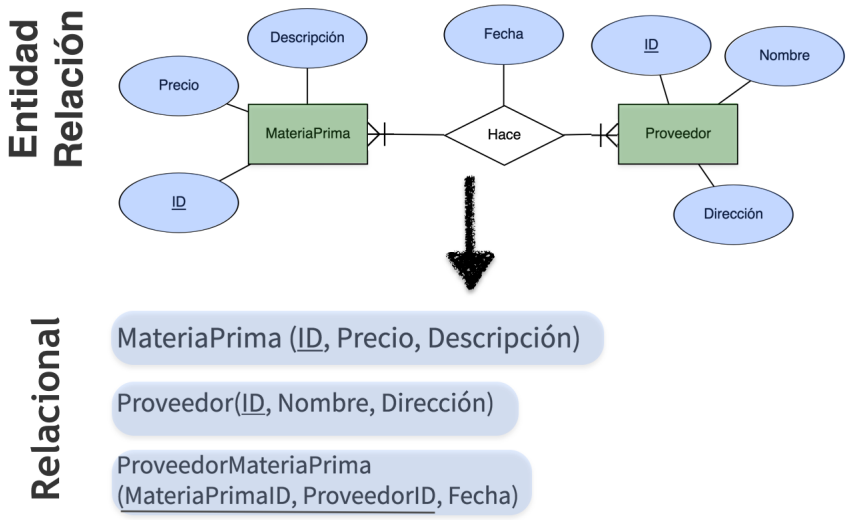
4.4.8. Mapear Entidades Asociativas

Se crean 3 relaciones, una para cada una de las entidades, incluyendo la entidad asociativa (la cual se denomina relación asociativa al hacer el mapeo).

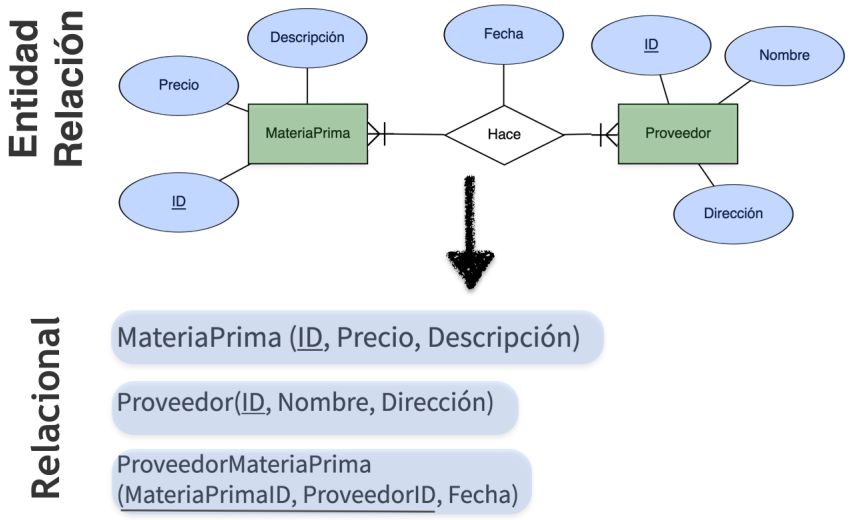
Existen dos casos al hacer el mapeo al modelo relacional, el primero es si no existe un identificador asignado a la entidad asociativa y el segundo es si existe un identificador asignado a la entidad asociativa.

- Si no existe un identificador asignado, la llave primaria de la relación asociativa se va a componer de dos atributos (llave compuesta), los cuales son las llaves primarias de las otras dos relaciones.

  Los atributos simples asociados a la entidad asociativa se agregan a la relación asociativa como atributos adicionales.



- Si existe un identificador asignado a la entidad asociativa, se crean tres relaciones, incluyendo la relación asociativa, la llave primaria de la relación asociativa va a ser el atributo identificador asociado a la misma y no será una llave compuesta como en el caso anterior. Las llaves primarias de las otras dos entidades participantes son incluidas como llaves foráneas en la relación asociativa resultante.



Capítulo 5

Almacenamiento de Información

5.1. Indices

5.1.1. Introducción

Muchas consultas hacen referencia solo a una pequeña proporción de los registros en un archivo.

Por ejemplo, una consulta como "Buscar todos los instructores en el departamento de Física" o "Encontrar el número total de créditos obtenidos por el estudiante con ID 22201" hace referencia a solo una fracción de los registros del estudiante.

Es ineficiente que el sistema lea cada tupla en la relación del instructor para verificar si el departamento es ineficiente para leer toda la relación estudiantil solo para encontrar la única tupla para la ID "32556".

Idealmente, el sistema debería poder ubicar estos registros directamente. Para permitir estas formas de acceso, diseñamos estructuras adicionales que asociamos con los archivos.

Un índice para un archivo en un sistema de base de datos funciona de forma muy parecida al índice en este libro de texto. Si queremos aprender sobre un tema en particular (especificado por una palabra o una frase) en este libro de texto, podemos buscar el tema en el índice al final del libro, encontrar las páginas donde se produce y luego leer las páginas para encontrar la información para la que estamos buscando.

Las palabras en el índice están ordenadas, lo que facilita encontrar la palabra que queremos. Además, el índice es mucho más pequeño que el libro, reduciendo aún más el esfuerzo necesario.

Los índices del sistema de base de datos cumplen el mismo rol que los índices de libros en las bibliotecas.

Por ejemplo, para recuperar un registro de alumno con una ID, el sistema de base de datos buscaría un índice para encontrar en qué bloque de disco reside el registro correspondiente, y luego buscará el bloque de disco, para obtener el registro de estudiante apropiado.

Mantener una lista ordenada de ID de los estudiantes no funcionaría bien en bases de datos muy grandes con miles de estudiantes, ya que el índice sería muy grande; Además, aunque mantener el índice ordenado reduce el tiempo de búsqueda, encontrar un estudiante puede ser bastante lento.

Los índices en las BBDD pretenden aligerar las consultas, a modo de simil podríamos verlos como un índice de libro. Buscar un capítulo en un libro sin un índice implicaría recorrer el libro entero hasta que nos topasemos con el mientras que encontrarlo con un índice supondría recorrer este último y luego ir directamente a la página en la que se encuentra lo que nos interesa.

En general podemos decir que: Un índice es una estructura de disco asociada con una tabla o una vista que acelera la recuperación de filas de la tabla o de la vista. Un índice contiene claves generadas a partir de una o varias columnas de la tabla o la vista.

Pro:

- Las búsquedas recorren muchas menos tuplas.
- Menos carga.
- Mas velocidad.

Contras:

- Evidentemente los índices tienen que generarse y posteriormente actualizarse con cada escritura, esto supone carga.
- Los índices ocupan espacio, se que parece de perogrullo, pero cuando superan en tamaño a la BBDD tienes un problema.

5.1.2. Indices Primarios

Para obtener acceso aleatorio rápido a los registros en un archivo, podemos usar una estructura de índice. Cada estructura de índice está asociada con una clave de búsqueda en particular.

Al igual que el índice de un libro o un catálogo de biblioteca, un índice ordenado almacena los valores de las claves de búsqueda en orden ordenado y asocia con cada clave de búsqueda los registros que lo contienen.

Un archivo puede tener varios índices, en diferentes claves de búsqueda. Si el archivo que contiene los registros se ordena secuencialmente, un índice de agrupamiento es un índice cuya clave de búsqueda también define el orden secuencial del archivo.

El término índice primario puede parecer que denota un índice en una clave primaria, pero tales índices pueden de hecho construirse en cualquier clave de búsqueda. La clave de búsqueda de un índice de agrupamiento suele ser la clave principal, aunque no necesariamente. Los índices cuya clave de búsqueda especifica un orden diferente del orden secuencial del archivo se denominan índices no agrupados o índices secundarios.

5.1.3. Indices Secundarios

Los índices secundarios deben ser densos, con una entrada de índice para cada valor de clave de búsqueda y un puntero para cada registro en el archivo. Un índice de agrupamiento puede ser escaso, almacenando solo algunos de los valores de la clave de búsqueda, ya que siempre es posible encontrar registros con valores de clave de búsqueda intermedios mediante un acceso secuencial a una parte del archivo, como se describió anteriormente. Si un índice secundario almacena solo algunos de los valores de la clave de búsqueda, los registros con valores de clave de búsqueda intermedios pueden estar en cualquier lugar del archivo y, en general, no podemos encontrarlos sin buscar el archivo completo.

Un índice secundario en una clave candidata se parece exactamente a un índice de agrupamiento denso, excepto que los registros apuntados por valores sucesivos en el índice no se almacenan secuencialmente.

En general, sin embargo, los índices secundarios pueden tener una estructura diferente de los índices de agrupamiento. Si la clave de búsqueda de un índice de clústeres no es una clave candidata, basta con que el índice apunte al primer registro con un valor particular para la clave de búsqueda, ya que los otros registros pueden obtenerse mediante un escaneo secuencial del archivo.

5.1.4. Indices Hash

Una desventaja de la organización de archivos secuenciales es que debemos acceder a una estructura de índice para localizar datos, o debemos usar la búsqueda binaria, y eso da como resultado más operaciones de E / S. Las organizaciones de archivos basadas en la técnica de hashing nos permiten evitar el acceso a una estructura de índice.

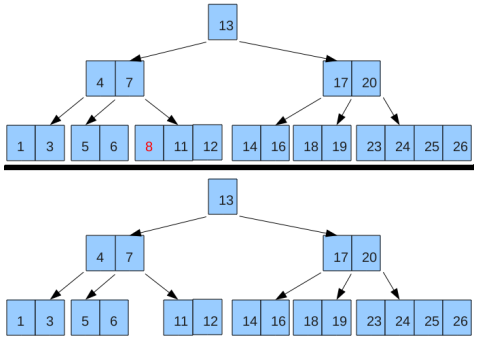
Hashing también proporciona una forma de construir índices. Estudiamos organizaciones de archivos e índices basados en hashing en las siguientes secciones.

5.1.5. Indices Arboles B

La principal desventaja de la organización de archivos index-sequential es que el rendimiento se degrada a medida que el archivo crece, tanto para las búsquedas de índices como para los escaneos secuenciales a través de los datos. Aunque esta degradación se puede remediar mediante la reorganización del archivo, las reorganizaciones frecuentes son indeseables.

La estructura de índice de árbol B + es la más utilizada de varias estructuras de índice que mantienen su eficiencia a pesar de la inserción y eliminación de datos. Un índice de árbol B toma la forma de un árbol equilibrado en el que cada ruta desde la raíz del árbol hasta una hoja del árbol es de la misma longitud.

Cada nodo sin hoja en el árbol tiene entre [n / 2] y n hijos, donde n se fija para un árbol en particular.



Parte II

Parte Practica

Capítulo 6

SQL

6.1. SQL como Lenguaje

SQL **no** es tan potente como una máquina universal de Turing.

Es decir, hay algunos cálculos que son posibles utilizando un lenguaje de programación de propósito general, pero no son posibles con SQL. SQL también no admite acciones como la entrada de usuarios, la salida a las pantallas ó la comunicación a través de la red. Dichas computaciones y acciones deben escribirse en un lenguaje principal (C, C++ ó Java) con consultas SQL incorporadas que acceden a los datos de la base de datos

Los programas de aplicación son programas que se utilizan para interactuar con la base de datos de esta manera.

Capítulo 7

Queries en SQL

7.1. SELECT FROM

Empecemos por la sentencia más básica para un Query en SQL:

```
1 SELECT FielName FROM TableName;
```

Ahora empecemos por aquí:

7.1.1. Distint vs All

- Distint (Default): Esta solo muestra solo los atributos diferentes

```
1 SELECT DISTINT FielName FROM TableName;
```

- All: Esta muestra todos los atributos, incluso duplicados

```
1 SELECT ALL FielName FROM TableName;
```

7.2. Funciones de Agregación

Las funciones de agregación son funciones que toman una colección como entrada y regresan un simple valor. SQL ofrece estas 5 funciones por default:

- "AVG()": Promedio
- "MIN()": Minimo
- "MAX()": Máximo
- "COUNT()": Cuenta la cantidad
- "SUM()": Suma todo el resultado

7.3. WHERE

Esta nos permite seleccionar solo las filas en el resultado del quey que satisface nuestros criterios.

```
SELECT FielName FROM TableName
    WHERE
        (TableName.Thing1 < Something) AND
        (TableName.Thing2 = "Some_Strings");
```

Tenemos a nuestra dispoción los siguientes comparadores:

- Lógicos: AND, OR, NOT
- Aritmeticos: <, >, <=, >=, =, <>

7.4. Comparación de Strings

Podemos comparar strings gracias a lo que conocemos como comodínes usando el comando "LIKE"

- " %": Que sirve como comodín para cualquier cantidad de caracteres
- "_": Que sirve como comodín para un caracter

7.5. Orden de las Tuplas

Podemos controlar el orden en que se nos retornan las tuplas del query que estamos haciendo muy facilmente con la instructiva ORDER BY.

Por default se usa la opción "ASC" que las ordena de manera ascendente mientras que "DESC" la ordena de manera descendente.

Por default esta se coloca la última línea de manera implicita:

```
SELECT DISTINT FielName1, FielName2, FielName3 FROM TableName
    ORDER BY FielName1 ASC, FielName2 ASC, FielName3 ASC;
```

También podemos simplemente poner el número en que las pusimos para indicar el orden:

```
SELECT DISTINT FielName1, FielName2, FielName3 FROM TableName
    ORDER BY 1,2, 3;
```

7.6. Views en SQL

En teoría de bases de datos, una vista es una consulta que se presenta como una tabla (virtual) a partir de un conjunto de tablas en una base de datos relacional.

Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo **se almacena de ellas la definición, no los datos**. Los cambios aplicados a los datos en una tabla se reflejan en los datos mostrados en invocaciones posteriores de la vista. En algunas bases de datos NoSQL, las vistas son la única forma de consultar datos.

Los datos que se recuperan mediante una consulta a una vista se presentarán igual que los de una tabla.

Una vista se crea a través de una expresión de consulta (una sentencia SELECT) que la calcula y que puede realizarse sobre una o más tablas.

7.6.1. Ventajas

Las vistas pueden ofrecer ventajas sobre las tablas:

- Las vistas pueden representar un subconjunto de los datos contenidos en una tabla. En consecuencia, una vista puede limitar el grado de exposición a las tablas al mundo exterior: un usuario dado puede tener permiso para consultar la vista, mientras que se deniega el acceso al resto de la tabla base.
- Las vistas pueden ocultar la complejidad de los datos. Por ejemplo, una vista podría aparecer como Sales2000 o Sales2001, particionando de forma transparente la tabla subyacente real.
- Las vistas cuestan muy poco espacio para almacenar; la base de datos contiene sólo la definición de una vista, no una copia de todos los datos que presenta.

7.6.2. Sintaxis

Empecemos por la sentencia general de las vistas:

```
1 CREATE VIEW ViewName AS
2     SELECT FielName
3         FROM TableName
4         WHERE Conditions
5         ORDER BY FielName;
```