



Temas desarrollados en este material

<i>Introducción a los conceptos de bases de datos</i>	4
1.1 Definición de Base de Datos	4
1.2 Objetivos de los sistemas de bases de datos.	6
1.3 Abstracción de la información.	8
1.4 Modelos de datos.	10
1.5 Instancias y esquemas.	15
1.6 Independencia de los datos	16
1.7 Lenguaje de definición de datos	16
1.8 Lenguaje de manipulación de datos	16
1.9 Manejador de Bases de Datos	17
1.10 Administrador de Bases de Datos	19
1.11 Usuarios de las bases de datos.	19
1.12 Estructura general del sistema.	21
<i>Modelo Entidad-Relación</i>	23
2.1 Entidades y conjunto de entidades	23
2.2 Relaciones y conjunto de relaciones.	24
2.3 Limitantes de mapeo.	25
2.4 Llaves primarias.	28
2.5 Diagrama Entidad-Relación	29
2.6 Reducción de diagramas E-R a tablas	31
2.7 Generalización y especialización	33
2.8 Agregación.	35
<i>Modelo relacional</i>	37
3.1 Estructura de las bases de datos relacionales	37
3.2 Lenguajes de consulta formales.	39
Solución a problemas de álgebra relacional.	43
3.3 Lenguajes de consulta comerciales	44
3.4 Modificación de la Base de datos	53

Introducción a los conceptos de bases de datos

1.1 Definición de Base de Datos

Todo buen curso necesita empezar con algunos conceptos básicos para el mejor entendimiento del mismo, por lo tanto empezaremos con las definiciones que involucran a las bases de datos.

Dato:

Conjunto de caracteres con algún significado, pueden ser numéricos, alfabéticos, o alfanuméricos.

Información:

Es un conjunto ordenado de datos los cuales son manejados según la necesidad del usuario, para que un conjunto de datos pueda ser procesado eficientemente y pueda dar lugar a información, primero se debe guardar lógicamente en archivos.

Conceptos básicos de archivos computacionales.

Campo:

Es la unidad más pequeña a la cual uno puede referirse en un programa. Desde el punto de vista del programador representa una característica de un individuo u objeto.

Registro:

Colección de campos de iguales o de diferentes tipos.

Archivo:

Colección de registros almacenados siguiendo una estructura homogénea.

Base de datos:

Es una colección de archivos interrelacionados, son creados con un DBMS. El contenido de una base de datos engloba a la información concerniente(almacenadas en archivos) de una organización, de tal manera que los datos estén disponibles para los usuarios, una finalidad de la base de datos es eliminar la redundancia o al menos minimizarla. Los tres componentes principales de un sistema de base de datos son el hardware, el software DBMS y los datos a manejar, así como el personal encargado del manejo del sistema.

Sistema Manejador de Base de Datos. (DBMS)

Un DBMS es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de una tarea específica.

El objetivo primordial de un sistema manejador base de datos es proporcionar un contorno que sea a la vez conveniente y eficiente para ser utilizado al extraer, almacenar y manipular información de la base de datos. Todas las peticiones de acceso a la base, se manejan centralizadamente por medio del DBMS, por lo que este paquete funciona como interfase entre los usuarios y la base de datos.

Esquema de base de datos:

Es la estructura por la que esta formada la base de datos, se especifica por medio de un

1.2 Objetivos de los sistemas de bases de datos.

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

◆ Redundancia e inconsistencia de datos.

Puesto que los archivos que mantienen almacenada la información son creados por diferentes tipos de programas de aplicación existe la posibilidad de que si no se controla detalladamente el almacenamiento, se pueda originar un duplicado de información, es decir que la misma información sea más de una vez en un dispositivo de almacenamiento. Esto aumenta los costos de almacenamiento y acceso a los datos, además de que puede originar la inconsistencia de los datos - es decir diversas copias de un mismo dato no concuerdan entre sí -, por ejemplo: que se actualiza la dirección de un cliente en un archivo y que en otros archivos permanezca la anterior.

◆ Dificultad para tener acceso a los datos.

Un sistema de base de datos debe contemplar un entorno de datos que le facilite al usuario el manejo de los mismos. Supóngase un banco, y que uno de los gerentes necesita averiguar los nombres de todos los clientes que viven dentro del código postal 78733 de la ciudad. El gerente pide al departamento de procesamiento de datos que genere la lista correspondiente. Puesto que esta situación no fue prevista en el diseño del sistema, no existe ninguna aplicación de consulta que permita este tipo de solicitud, esto ocasiona una deficiencia del sistema.

◆ Aislamiento de los datos.

Puesto que los datos están repartidos en varios archivos, y estos no pueden tener diferentes formatos, es difícil escribir nuevos programas de aplicación para obtener los datos apropiados.

◆ Anomalías del acceso concurrente.

Para mejorar el funcionamiento global del sistema y obtener un tiempo de respuesta más rápido, muchos sistemas permiten que múltiples usuarios actualicen los datos simultáneamente. En un entorno así la interacción de actualizaciones concurrentes puede dar por resultado datos inconsistentes. Para prevenir esta posibilidad debe mantenerse alguna forma de supervisión en el sistema.

◆ Problemas de seguridad.

La información de toda empresa es importante, aunque unos datos lo son más que otros, por tal motivo se debe considerar el control de acceso a los mismos, no todos los usuarios pueden visualizar alguna información, por tal motivo para que un sistema de base de datos sea confiable debe mantener un grado de seguridad que garantice la autenticación y protección de los datos. En un banco por ejemplo, el personal de nóminas sólo necesita ver la parte de la base de datos que tiene información acerca de los distintos empleados del banco y no a otro tipo de información.

A horizontal line of 20 small black squares.
A small blue diamond icon.
Problemas de integridad.

Los valores de datos almacenados en la base de datos deben satisfacer cierto tipo de restricciones de consistencia. Estas restricciones se hacen cumplir en el sistema añadiendo códigos apropiados en los diversos programas de aplicación.

1.3 Abstracción de la información.

Una base de datos es en esencia una colección de archivos relacionados entre sí, de la cual los usuarios pueden extraer información sin considerar las fronteras de los archivos.

Un objetivo importante de un sistema de base de datos es proporcionar a los usuarios una visión *abstracta* de los datos, es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. Sin embargo para que el sistema sea manejable, los datos se deben extraer eficientemente.

Existen diferentes niveles de abstracción para simplificar la interacción de los usuarios con el sistema; Interno, conceptual y externo, específicamente el de almacenamiento físico, el del usuario y el del programador.

◆ Nivel físico.

Es la representación del nivel más bajo de abstracción, en éste se describe en detalle la forma en como se almacenan los datos en los dispositivos de almacenamiento (por ejemplo, mediante señalizadores o índices para el acceso aleatorio a los datos).

◆ Nivel conceptual.

El siguiente nivel más alto de abstracción, describe que datos son almacenados realmente en la base de datos y las relaciones que existen entre los mismos, describe la base de datos completa en términos de su estructura de diseño. El nivel conceptual de abstracción lo usan los administradores de bases de datos, quienes deben decidir qué información se va a guardar en la base de datos.

Consta de las siguientes definiciones:

1. **Definición de los datos:** Se describen el tipo de datos y la longitud de campo todos los elementos direccionables en la base. Los elementos por definir incluyen artículos elementales (atributos), totales de datos y registros conceptuales (entidades).
2. **Relaciones entre datos:** Se definen las relaciones entre datos para enlazar tipos de registros relacionados para el procesamiento de archivos múltiples.

En el nivel conceptual la base de datos aparece como una colección de registros lógicos, sin descriptores de almacenamiento. En realidad los archivos conceptuales no existen físicamente. La transformación de registros conceptuales a registros físicos para el almacenamiento se lleva a cabo por el sistema y es transparente al usuario.

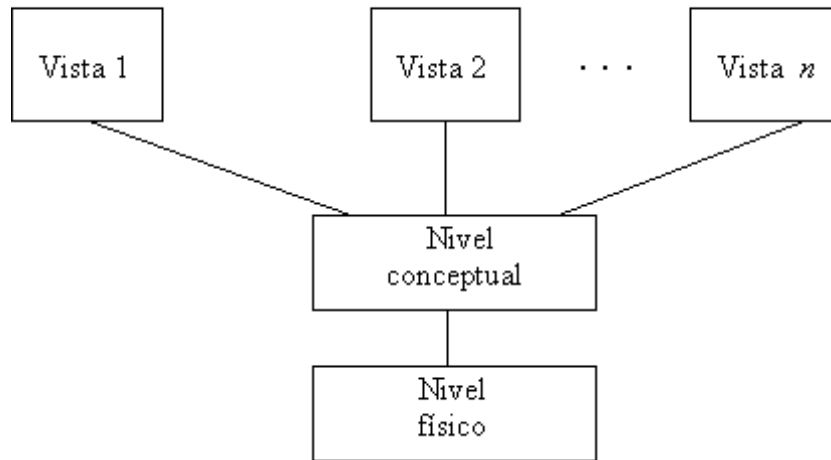
◆ Nivel de visión.

Nivel más alto de abstracción, es lo que el usuario final puede visualizar del sistema terminado, describe sólo una parte de la base de datos al usuario acreditado para verla. El sistema puede proporcionar muchas visiones para la misma base de datos.

La interrelación entre estos tres niveles de abstracción se ilustra en la siguiente figura.



Nivel de visión





1.4 Modelos de datos.

Para introducirnos en este tema, empezaremos definiendo que es un modelo.

modelo:

Es una representación de la realidad que contiene las características generales de algo que se va a realizar. En base de datos, esta representación la elaboramos de forma gráfica.

¿Qué es modelo de datos?

Es una colección de herramientas conceptuales para describir los datos, las relaciones que existen entre ellos, semántica asociada a los datos y restricciones de consistencia.

Los modelos de datos se dividen en tres grupos:

- Modelos lógicos basados en objetos.
- Modelos lógicos basados en registros.
- Modelos físicos de datos.

• **Modelos lógicos basados en objetos.**

Se usan para describir datos en los niveles conceptual y de visión, es decir, con este modelo representamos los datos de tal forma como nosotros los captamos en el mundo real, tienen una capacidad de estructuración bastante flexible y permiten especificar restricciones de datos explícitamente. Existen diferentes modelos de este tipo, pero el más utilizado por su sencillez y eficiencia es el modelo Entidad-Relación.

* **Modelo Entidad-Relación.**

Denominado por sus siglas como: E-R; Este modelo representa a la realidad a través de **entidades**, que son objetos que existen y que se distinguen de otros por sus características, por ejemplo: un alumno se distingue de otro por sus características particulares como lo es el nombre, o el número de control asignado al entrar a una institución educativa, así mismo, un empleado, una materia, etc. Las entidades pueden ser de dos tipos:

- **Tangibles :**
Son todos aquellos objetos físicos que podemos ver, tocar o sentir.
- **Intangibles:**
Todos aquellos eventos u objetos conceptuales que no podemos ver, aun sabiendo que existen, por ejemplo: la entidad materia, sabemos que existe, sin embargo, no la podemos visualizar o tocar.

Las características de las entidades en base de datos se llaman **atributos**, por ejemplo el nombre, dirección teléfono, grado, grupo, etc. son atributos de la entidad alumno; Clave, número

de seguro social, departamento, etc., son atributos de la entidad empleado. A su vez una entidad se puede asociar o relacionar con más entidades a través de **relaciones**.


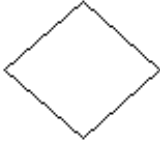


Pero para entender mejor esto, veamos un **ejemplo**:

Consideremos una empresa que requiere controlar a los vendedores y las ventas que ellos realizan; de este problema determinamos que los objetos o entidades principales a estudiar son el empleado (vendedor) y el artículo (que es el producto en venta), y las características que los identifican son:

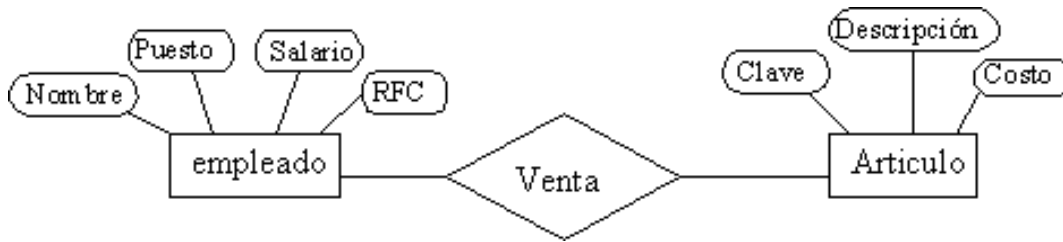
<i>Empleado:</i>	<i>Artículo:</i>
Nombre	Descripción
Puesto	Costo
Salario	Clave
R.F.C.	

La relación entre ambas entidades la podemos establecer como **Venta**.

Bueno, ahora nos falta describir como se representa un modelo E-R gráficamente, la representación es muy sencilla, se emplean símbolos, los cuales son:

Símbolo	Representa
	Entidad
	Relación
	Atributos
	Ligas

Así nuestro ejemplo anterior quedaría representado de la siguiente forma:



Existen más aspectos a considerar con respecto a los modelos entidad relación, estos serán considerados en el tema [Modelo Entidad Relación](#).

Modelos lógicos basados en registros.

Se utilizan para describir datos en los niveles conceptual y físico. Estos modelos utilizan registros e instancias para representar la realidad, así como las relaciones que existen entre estos registros (ligas) o apuntadores. A diferencia de los modelos de datos basados en objetos, se usan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a nivel más alto de la implementación. Los tres modelos de datos más ampliamente aceptados son:

- ◆ Modelo Relacional
- ◆ Modelo de Red
- ◆ Modelo Jerárquico

* Modelo relacional.

En este modelo se representan los datos y las relaciones entre estos, a través de una colección de tablas, en las cuales los renglones (tuplas) equivalen a los cada uno de los registros que contendrá la base de datos y las columnas corresponden a las características (atributos) de cada registro localizado en la tupla;

Considerando nuestro ejemplo del empleado y el artículo:

Tabla del empleado

* Cada una las columnas representa a los atributos de la entidad empleado

Tabla del empleado

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

* Registros que contienen la información de la entidad empleado

Tabla artículo

Clave	Descripción	Costo
C001	Colcha matrimonial	200

Ahora te preguntaras **¿cómo se representan las relaciones entre las entidades en este modelo?**

Existen dos formas de representarla; pero para ello necesitamos definir que es una **llave primaria**: Es un atributo el cual definimos como atributo principal, es una forma única de identificar a una entidad. Por ejemplo, el RFC de un empleado se distingue de otro por que los RFC no pueden ser iguales.

Ahora si, las formas de representar las relaciones en este modelo son:

1. Haciendo una tabla que contenga cada una de las llaves primarias de las entidades involucradas en la relación.

Tomando en cuenta que la llave primaria del empleado es su RFC, y la llave primaria del articulo es la Clave.

La relación de nuestro modelo resulta:

RFC	Clave
PECJ500922XYZ	C001
MEAN761014ABC	B300

2. Incluyendo en alguna de las tablas de las entidades involucradas, la llave de la otra tabla.

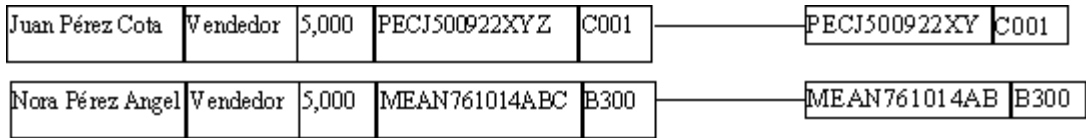
Incrustamos la llave primaria del articulo en la tabla del empleado

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

* Modelo de red.

Este modelo representa los datos mediante colecciones de registros y sus relaciones se representan por medio de ligas o enlaces, los cuales pueden verse como punteros. Los registros se organizan en un conjunto de gráficas arbitrarias.

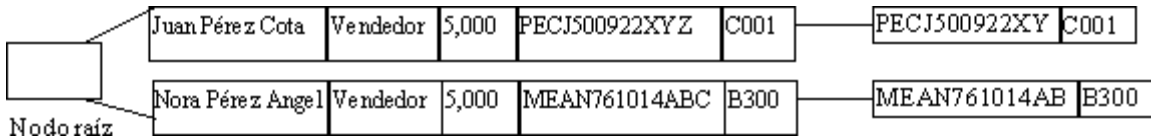
Ejemplo:



Para profundizar en este tema visitemos: [Modelo de datos de red.](#)

*** Modelo jerárquico.**

Es similar al modelo de red en cuanto a las relaciones y datos, ya que estos se representan por medio de registros y sus ligas. La diferencia radica en que están organizados por conjuntos de arboles en lugar de gráficas arbitrarias.



Para profundizar más en el tema visita: [Modelo de datos jerárquico.](#)

Modelos físicos de datos.

Se usan para describir a los datos en el nivel más bajo, aunque existen muy pocos modelos de este tipo, básicamente capturan aspectos de la implementación de los sistemas de base de datos. Existen dos clasificaciones de este tipo que son:

- Modelo unificador
- Memoria de elementos.



1.5 Instancias y esquemas.

Con el paso del tiempo la información que se va acumulando y desechando en la base de datos, ocasiona que está cambie.

Denominamos:

Instancia.

Al estado que presenta una base de datos en un tiempo dado. Veámoslo como una fotografía que tomamos de la base de datos en un tiempo t, después de que transcurre el tiempo t la base de datos ya no es la misma.

Esquema.

Es la descripción lógica de la base de datos, proporciona los nombres de las entidades y sus atributos especificando las relaciones que existen entre ellos. Es un banco en el que se inscriben los valores que irán formando cada uno de los atributos. El esquema no cambia los que varían son los datos y con esto tenemos una nueva instancia.

Ejemplo:

Considerando el ejemplo del vendedor que vende artículos, esquema e instancia según nuestro ejemplo, quedaría:

Esquema:

{ Vendedor : Nombre, puesto, salario, RFC }

{ Artículo : Clave, costo, descripción }

Instancia:

Juan Pérez Cota	Vendedor	5,000	PECJ500922XYZ	—	C001	250	Colcha matrimonial
-----------------	----------	-------	---------------	---	------	-----	--------------------

Como podemos observar el esquema nos muestra la estructura en el cual se almacenaran los datos, en este caso en registros cuyos nombres de campos son: por parte del vendedor (Nombre, puesto, salario, RFC) y por el artículo (Clave, costo, descripción); La instancia representa a una serie de datos almacenados en los registros establecidos por el esquema, estos datos varían, no permanecen fijos en el tiempo.

1.6 Independencia de los datos

Se refiere a la protección contra los programas de aplicación que puedan originar modificaciones cuando se altera la organización física o lógica de la base de datos. Existen 2 niveles de independencia de datos.

◆ **Independencia física de datos:**

Es la capacidad de modificar el esquema físico sin provocar que se vuelvan a escribir los programas de aplicación.

◆ **Independencia lógica de datos:**

Capacidad de modificar el esquema conceptual sin provocar que se vuelvan a escribir los programas de aplicación.

1.7 Lenguaje de definición de datos

El lenguaje de definición de datos, denominado por sus siglas como: DDL(Data definition Language).

Permite definir un esquema de base de datos por medio de una serie de definiciones que se expresan en un lenguaje especial, el resultado de estas definiciones se almacena en un archivo especial llamado diccionario de datos.

1.8 Lenguaje de manipulación de datos

La manipulación de datos se refiere a las operaciones de insertar, recuperar, eliminar o modificar datos; dichas operaciones son realizadas a través del lenguaje de manipulación de datos (DML, Data Manipulation Language), que es quién permite el acceso de los usuarios a los datos.

Existen básicamente 2 tipos de lenguajes de manipulación de datos:

- **Procedimentales:**
Los LMD requieren que el usuario especifique que datos se necesitan y cómo obtenerlos.
- **No procedimentales:**
Los LMD requieren que el usuario especifique que datos se necesitan y sin especificar cómo obtenerlos.

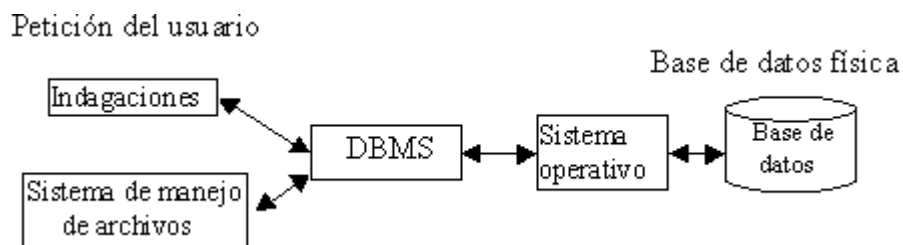
1.9 Manejador de Bases de Datos

El sistema manejador de bases de datos es la porción más importante del software de un sistema de base de datos. Un DBMS es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de alguna tarea específica.

Las funciones principales de un DBMS son:

- Crear y organizar la Base de datos.
- Establecer y mantener las trayectorias de acceso a la base de datos de tal forma que los datos puedan ser accedidos rápidamente.
- Manejar los datos de acuerdo a las peticiones de los usuarios.
- Registrar el uso de las bases de datos.
- Interacción con el manejador de archivos.
Esto a través de las sentencias en DML al comando de el sistema de archivos. Así el Manejador de base de datos es el responsable del verdadero almacenamiento de los datos.
- Respaldo y recuperación.
Consiste en contar con mecanismos implantados que permitan la recuperación fácilmente de los datos en caso de ocurrir fallas en el sistema de base de datos.
- Control de concurrencia.
Consiste en controlar la interacción entre los usuarios concurrentes para no afectar la inconsistencia de los datos.
- Seguridad e integridad.
Consiste en contar con mecanismos que permitan el control de la consistencia de los datos evitando que estos se vean perjudicados por cambios no autorizados o previstos.

El DBMS es conocido también como Gestor de Base de datos.



La figura muestra el DBMS como interfase entre la base de datos física y las peticiones del usuario. El DBMS interpreta las peticiones de entrada/salida del usuario y las manda al sistema operativo para la transferencia de datos entre la unidad de memoria secundaria y la memoria principal.

En sí, un sistema manejador de base de datos es el corazón de la base de datos ya que se encarga del control total de los posibles aspectos que la puedan afectar.

1.10 Administrador de Bases de Datos

Denominado por sus siglas como: DBA, Database Administrator.

Es la persona encargada y que tiene el control total sobre el sistema de base de datos, sus funciones principales son:

■ **Definición de esquema.**

Es el esquema original de la base de datos se crea escribiendo un conjunto de definiciones que son traducidas por el compilador de DDL a un conjunto de tablas que son almacenadas permanentemente en el diccionario de datos.

■ **Definición de la estructura de almacenamiento del método de acceso.**

Estructuras de almacenamiento y de acceso adecuados se crean escribiendo un conjunto de definiciones que son traducidas por el compilador del lenguaje de almacenamiento y definición de datos.

■ **Concesión de autorización para el acceso a los datos.**

Permite al administrador de la base de datos regular las partes de las bases de datos que van a ser accedidas por varios usuarios.

■ **Especificación de limitantes de integridad.**

Es una serie de restricciones que se encuentran almacenados en una estructura especial del sistema que es consultada por el gestor de base de datos cada vez que se realice una actualización al sistema.

1.11 Usuarios de las bases de datos.

Podemos definir a los usuarios como toda persona que tenga todo tipo de contacto con el sistema de base de datos desde que este se diseña, elabora, termina y se usa.

Los usuarios que accesan una base de datos pueden clasificarse como:

■ **Programadores de aplicaciones.**

Los profesionales en computación que interactúan con el sistema por medio de llamadas en DML (Lenguaje de Manipulación de Datos), las cuales están incorporadas en un programa escrito en un lenguaje de programación (Por ejemplo, COBOL, PL/I, Pascal, C, etc.)

■ **Usuarios sofisticados.**

Los usuarios sofisticados interactúan con el sistema sin escribir programas. En cambio escriben sus preguntas en un lenguaje de consultas de base de datos.

■ **Usuarios especializados.**

Algunos usuarios sofisticados escriben aplicaciones de base de datos especializadas que no encajan en el marco tradicional de procesamiento de datos.



■ **Usuarios ingenuos.**

Los usuarios no sofisticados interactúan con el sistema invocando a uno de los programas de aplicación permanentes que se han escrito anteriormente en el sistema de base de datos, podemos mencionar al usuario ingenuo como el usuario final que utiliza el sistema de base de datos sin saber nada del diseño interno del mismo por ejemplo: un cajero.

1.12 Estructura general del sistema.

Un sistema de base de datos se encuentra dividido en módulos cada uno de los cuales controla una parte de la responsabilidad total de sistema. En la mayoría de los casos, el sistema operativo proporciona únicamente los servicios más básicos y el sistema de la base de datos debe partir de esa base y controlar además el manejo correcto de los datos. Así el diseño de un sistema de base de datos debe incluir la interfaz entre el sistema de base de datos y el sistema operativo.

Los componentes funcionales de un sistema de base de datos, son:

◆ **Gestor de archivos.**

Gestiona la asignación de espacio en la memoria del disco y de las estructuras de datos usadas para representar información.

◆ **Manejador de base de datos.**

Sirve de interfaz entre los datos y los programas de aplicación.

◆ **Procesador de consultas.**

Traduce las proposiciones en lenguajes de consulta a instrucciones de bajo nivel. Además convierte la solicitud del usuario en una forma más eficiente.

◆ **Compilador de DDL.**

Convierte las proposiciones DDL en un conjunto de tablas que contienen metadatos, estas se almacenan en el diccionario de datos.

◆ **Archivo de datos.**

En él se encuentran almacenados físicamente los datos de una organización.

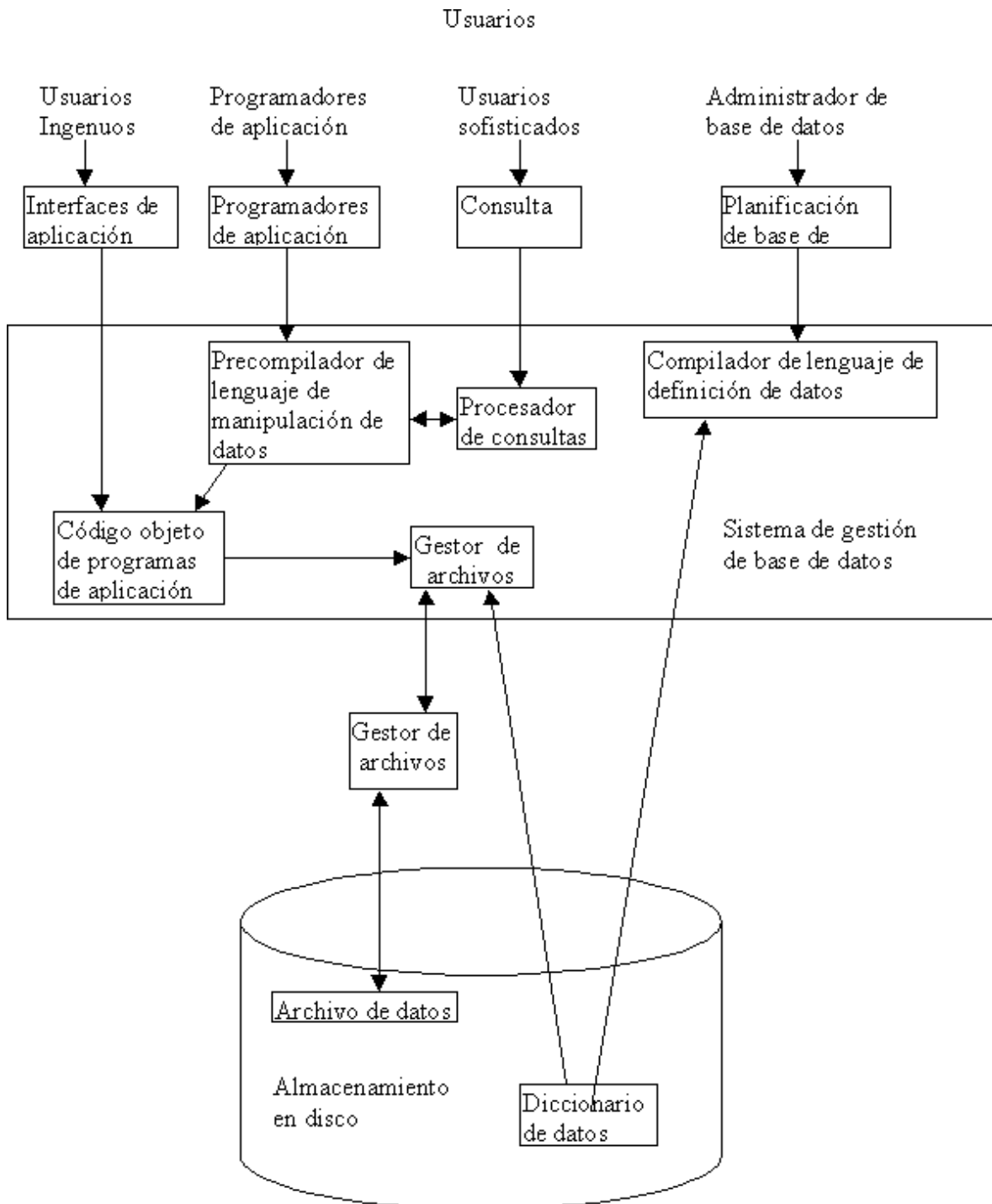
◆ **Diccionario de datos.**

Contiene la información referente a la estructura de la base de datos.

◆ **Indices.**

Permiten un rápido acceso a registros que contienen valores específicos.

Una forma gráfica de representar los componentes antes mencionados y la relación que existe entre ellos sería la siguiente.





Modelo Entidad-Relación

El modelo E-R se basa en una percepción del mundo real, la cual esta formada por objetos básicos llamados entidades y las relaciones entre estos objetos así como las características de estos objetos llamados atributos.

2.1 Entidades y conjunto de entidades

Una **entidad** es un objeto que existe y se distingue de otros objetos de acuerdo a sus características llamadas atributos . Las entidades pueden ser concretas como una persona o abstractas como una fecha.

Un **conjunto de entidades** es un grupo de entidades del mismo tipo. Por ejemplo el conjunto de entidades CUENTA, podría representar al conjunto de cuentas de un banco X, o ALUMNO representa a un conjunto de entidades de todos los alumnos que existen en una institución.

Una entidad se caracteriza y distingue de otra por los **atributos**, en ocasiones llamadas propiedades, que representan las características de una entidad. Los atributos de una entidad pueden tomar un conjunto de valores permitidos al que se le conoce como **dominio** del atributo. Así cada entidad se describe por medio de un conjunto de parejas formadas por el atributo y el valor de dato. Habrá una pareja para cada atributo del conjunto de entidades.

Ejemplo:

Hacer una descripción en pareja para la entidad alumno con los atributos *No_control*, *Nombre* y *Especialidad*.

Nombre_atributo, Valor

No_control , 96310418

Nombre , Sánchez Osuna Ana

Esp , LI

O considerando el ejemplo del Vendedor cuyos atributos son: RFC, Nombre, Salario.

Nombre_atributo, Valor

RFC , COMD741101YHR

Nombre , Daniel Colín Morales

Salario , 3000

2.2 Relaciones y conjunto de relaciones.

Una *relación* es la asociación que existe entre dos a más entidades.

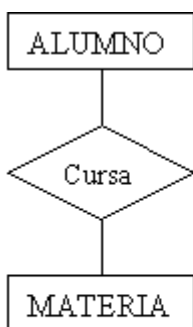
Un *conjunto de relaciones* es un grupo de relaciones del mismo tipo.

La cantidad de entidades en una relación determina el *grado* de la relación, por ejemplo la relación ALUMNO-MATERIA es de grado 2, ya que intervienen la entidad ALUMNO y la entidad MATERIA, la relación PADRES, puede ser de grado 3, ya que involucra las entidades PADRE, MADRE e HIJO.

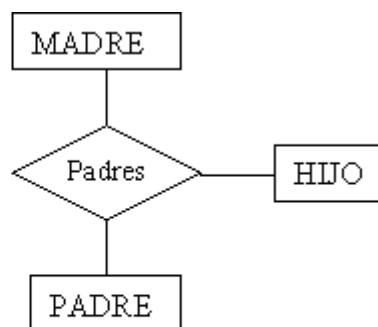
Aunque el modelo E-R permite relaciones de cualquier grado, la mayoría de las aplicaciones del modelo sólo consideran relaciones del grado 2. Cuando son de tal tipo, se denominan relaciones binarias.

La función que tiene una relación se llama *papel*, generalmente no se especifican los papeles o roles, a menos que se quiera aclarar el significado de una relación.

Diagrama E-R (sin considerar los atributos, sólo las entidades) para los modelos ejemplificados:



Ejemplo de relación de grado 2



Ejemplo de relación de grado 3

2.3 Limitantes de mapeo.

Existen 4 tipos de *relaciones* que pueden establecerse entre entidades, las cuales establecen con cuantas entidades de tipo B se pueden relacionar una entidad de tipo A:

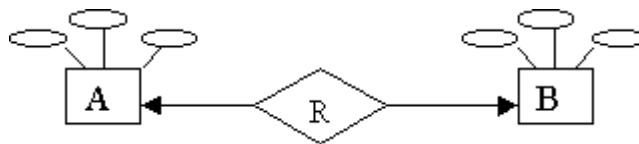
Tipos de relaciones:

▣ *Relación uno a uno.*

Se presenta cuando existe una relación como su nombre lo indica uno a uno, denominado también relación de matrimonio. Una entidad del tipo A solo se puede relacionar con una entidad del tipo B, y viceversa;

Por ejemplo: la relación *asignación de automóvil* que contiene a las entidades EMPLEADO, AUTO, es una relación 1 a 1, ya que asocia a un empleado con un único automóvil por lo tanto ningún empleado posee más de un automóvil asignado, y ningún vehículo se asigna a más de un trabajador.

Es representado gráficamente de la siguiente manera:



A: Representa a una entidad de cualquier tipo diferente a una entidad **B**.

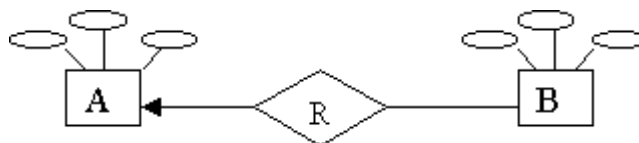
R: en el diagrama representa a la relación que existe entre las entidades.

El extremo de la flecha que se encuentra punteada indica el uno de la relación, en este caso, una entidad A ligada a una entidad B.

▣ *Relación uno a muchos.*

Significa que una entidad del tipo A puede relacionarse con cualquier cantidad de entidades del tipo B, y una entidad del tipo B solo puede estar relacionada con una entidad del tipo A.

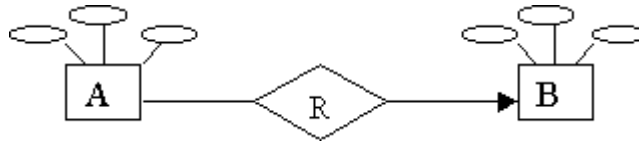
Su representación gráfica es la siguiente:



Nótese en este caso que el extremo punteado de la flecha de la relación de A y B, indica una entidad A conectada a muchas entidades B.

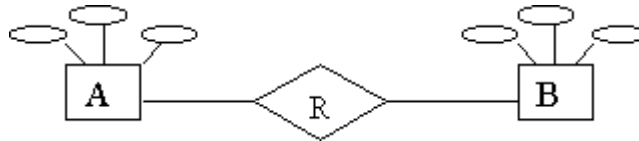
■ **Muchos a uno.**

Indica que una entidad del tipo B puede relacionarse con cualquier cantidad de entidades del tipo A, mientras que cada entidad del tipo A solo puede relacionarse con solo una entidad del tipo B.



■ **Muchas a muchas.**

Establece que cualquier cantidad de entidades del tipo A pueden estar relacionados con cualquier cantidad de entidades del tipo B.



A los tipos de relaciones antes descritos, también se le conoce como **cardinalidad**.

La cardinalidad nos especifica los tipos de relaciones que existen entre las entidades en el modelo E-R y establecer con esto las validaciones necesarias para conseguir que los datos de la instancia (valor único en un momento dado de una base de datos) correspondan con la realidad.

Algunos **ejemplos** de cardinalidades de la vida común pueden ser:

Uno a uno.

El noviazgo, el RFC de cada persona, El CURP personal, El acta de nacimiento, ya que solo existe un solo documento de este tipo para cada una de las diferentes personas.

Uno a muchos.

Cliente – Cuenta en un banco, Padre-Hijos, Camión-Pasajeros, zoológico- animales, árbol – hojas.

Muchos a muchos.

Arquitecto – proyectos, fiesta – personas, estudiante – materias.



2.4 Llaves primarias.

Como ya se ha mencionado anteriormente, la distinción de una entidad entre otra se debe a sus atributos, lo cual lo hacen único. Una *llave primaria* es aquel atributo el cual consideramos clave para la identificación de los demás atributos que describen a la entidad. Por ejemplo, si consideramos la entidad ALUMNO del Instituto Tecnológico de La Paz, podríamos tener los siguientes atributos: Nombre, Semestre, Especialidad, Dirección, Teléfono, Número de control, de todos estos atributos el que podremos designar como llave primaria es el número de control, ya que es diferente para cada alumno y este nos identifica en la institución.

Claro que puede haber más de un atributo que pueda identificarse como llave primaria en este caso se selecciona la que consideremos más importante, los demás atributos son denominados *llaves secundarias*.

Una clave o llave primaria es indicada gráficamente en el modelo E-R con una línea debajo del nombre del atributo.

2.5 Diagrama Entidad-Relación

Denominado por sus siglas como: E-R; Este modelo representa a la realidad a través de un esquema gráfico empleando la terminología de *entidades*, que son objetos que existen y son los elementos principales que se identifican en el problema a resolver con el diagramado y se distinguen de otros por sus características particulares denominadas *atributos*, el enlace que rige la unión de las entidades esta representada por la *relación* del modelo.

Recordemos que un rectángulo nos representa a las entidades; una elipse a los atributos de las entidades, y una etiqueta dentro de un rombo nos indica la relación que existe entre las entidades, destacando con líneas las uniones de estas y que la llave primaria de una entidad es aquel atributo que se encuentra subrayado.

A continuación mostraremos algunos ejemplos de modelos E-R, considerando las cardinalidades que existen entre ellos:

Relación Uno a Uno.

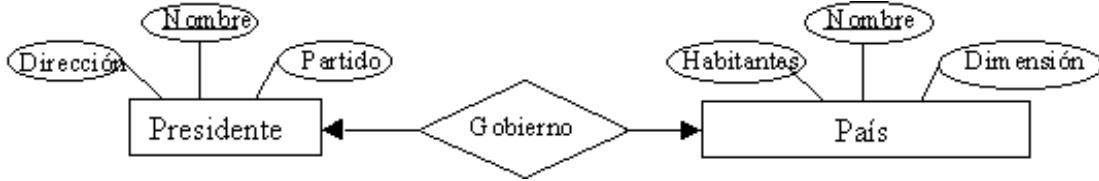
Problema:

Diseñar el modelo E-R, para la relación Registro de automóvil que consiste en obtener la tarjeta de circulación de un automóvil con los siguientes datos:- Automóvil- Modelo, Placas, Color - Tarjeta de circulación -Propietario, No_serie, Tipo.



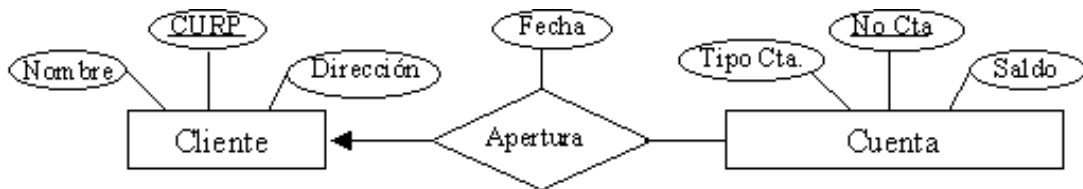
Indicamos con este ejemplo que existe una relación de pertenencia de uno a uno, ya que existe una tarjeta de circulación registrada por cada automóvil.

En este ejemplo, representamos que existe un solo presidente para cada país.



Relación muchos a muchos.

El siguiente ejemplo indica que un cliente puede tener muchas cuentas, pero que una cuenta puede llegar a pertenecer a un solo cliente (Decimos puede, ya que existen cuentas registradas a favor de más de una persona).

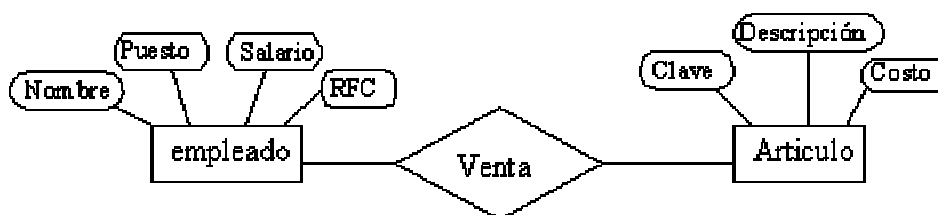


2.6 Reducción de diagramas E-R a tablas

Un diagrama E-R, puede ser representado también a través de una colección de tablas. Para cada una de las entidades y relaciones existe una tabla única a la que se le asigna como nombre el del conjunto de entidades y de las relaciones respectivamente, cada tabla tiene un número de columnas que son definidas por la cantidad de atributos y las cuales tienen el nombre del atributo.

La transformación de nuestro ejemplo Venta en la que intervienen las entidades de Vendedor con los atributos RFC, nombre, puesto, salario y Artículo con los atributos Clave, descripción, costo.

Cuyo diagrama E-R es el siguiente:



Entonces las tablas resultantes siguiendo la descripción anterior son:

Tabla Empleado

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

Tabla artículo

Clave	Descripción	Costo
A100	Abanico	460
C260	Colcha matrimonial	1200



Tabla Venta

RFC	Clave
TEAT701210XYZ	C260
COV741120ABC	A100

Nótese que en la tabla de relación - Venta -, contiene como atributos a las llaves primarias de las entidades que intervienen en dicha relación, en caso de que exista un atributo en las relaciones, este atributo es anexado como una fila más de la tabla;

Por ejemplo si anexamos el atributo fecha a la relación venta, la tabla que se originaría sería la siguiente:

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

2.7 Generalización y especialización

Generalización.

Es el resultado de la unión de 2 o más conjuntos de entidades (de bajo nivel) para producir un conjunto de entidades de más alto nivel. La generalización se usa para hacer resaltar los parecidos entre tipos de entidades de nivel más bajo y ocultar sus diferencias.

La generalización consiste en identificar todos aquellos atributos iguales de un conjunto de entidades para formar una entidad(es) global(es) con dichos atributos semejantes, dicha entidad(es) global(es) quedara a un nivel más alto al de las entidades origen.

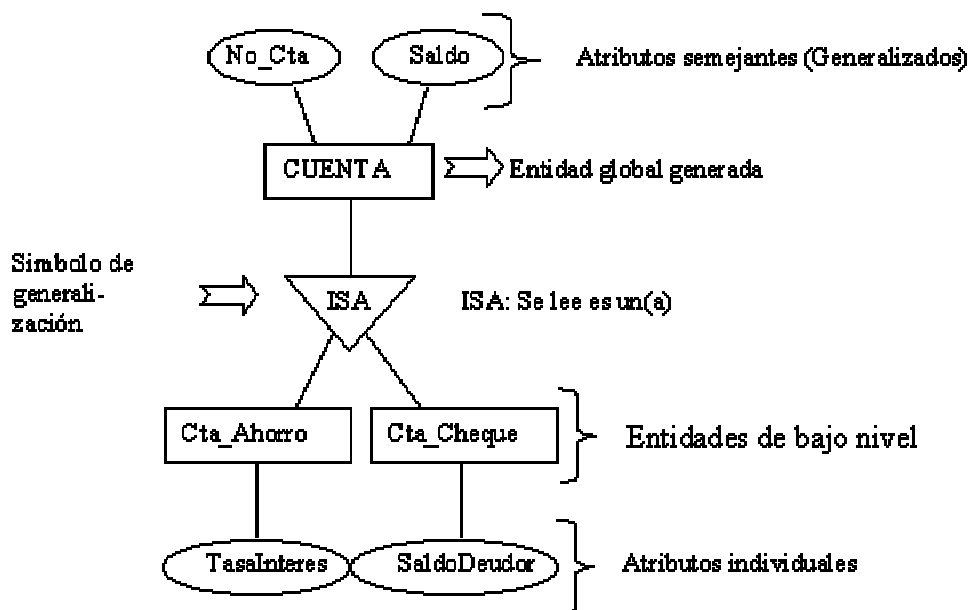
Ejemplo:

Tomando el ejemplo del libro de fundamentos de base de datos de Henry F. Korth.

Donde:

Se tiene las entidades Cta_Ahorro y Cta_Cheques, ambas tienen los atributos semejantes de No_Cta y Saldo, aunque además de estos dos atributos, Cta_Ahorro tiene el atributo Tasa_Interes y Cta_Cheques el atributo Saldo_Deudor. De todos estos atributos podemos juntar (generalizar) No_Cta y Saldo que son iguales en ambas entidades.

Entonces tenemos:



Podemos leer esta gráfica como: La entidad Cta_Ahorro hereda de la entidad CUENTA los atributos No_Cta y saldo, además del atributo de TasaInteres, de forma semejante Cta_cheque tiene los atributos de No_Cta, Saldo y SaldoDeudor.

Como podemos observar la Generalización trata de eliminar la redundancia (repetición) de atributos, al englobar los atributos semejantes. La entidad(es) de bajo nivel cuentan (heredan) todos los atributos correspondientes.

2.8 Agregación.

La agregación surge de la limitación que existe en el modelado de E-R, al no permitir expresar las relaciones entre relaciones de un modelo E-R en el caso de que una relación X se quiera unir con una entidad cualquiera para formar otra relación.

La Generalización consiste en agrupar por medio de un rectángulo a la relación (representada por un rombo) junto con las entidades y atributos involucrados en ella, para formar un grupo que es considerado una entidad y ahora sí podemos relacionarla con otra entidad.

Para ejemplificar lo anterior consideremos el ejemplo del libro de fundamentos de Base de Datos de Henry F. Korth. En donde el problema consiste en que existen trabajando muchos empleados que trabajan en diferentes proyectos, pero dependiendo del trabajo que realiza en pueden llegar a utilizar un equipo o maquinaria; en este problema intervienen 3 entidades: Empleado, Proyecto y Maquinaria, el diagrama E-R correspondiente es:

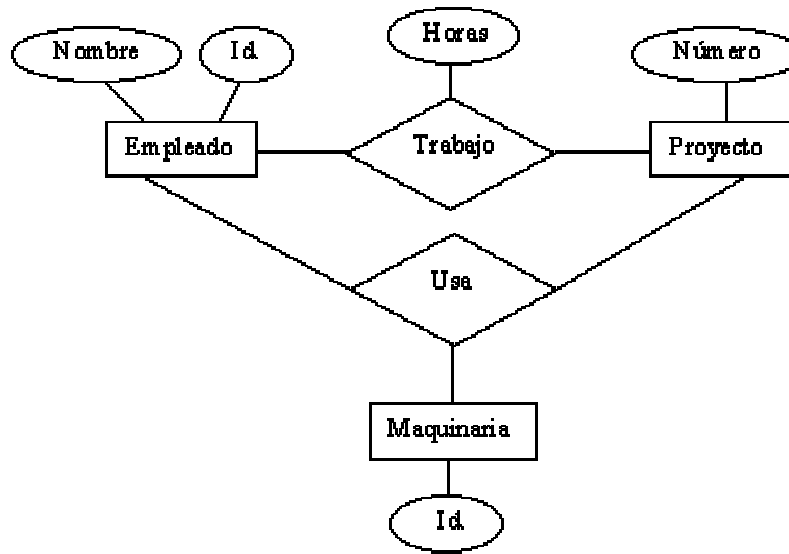


Diagrama E-R con relaciones redundantes

Como el modelo E-R no permite la unión entre dos o más relaciones, la relación trabajo es englobada como si fuera una entidad más de la relación usa, gráficamente queda como:

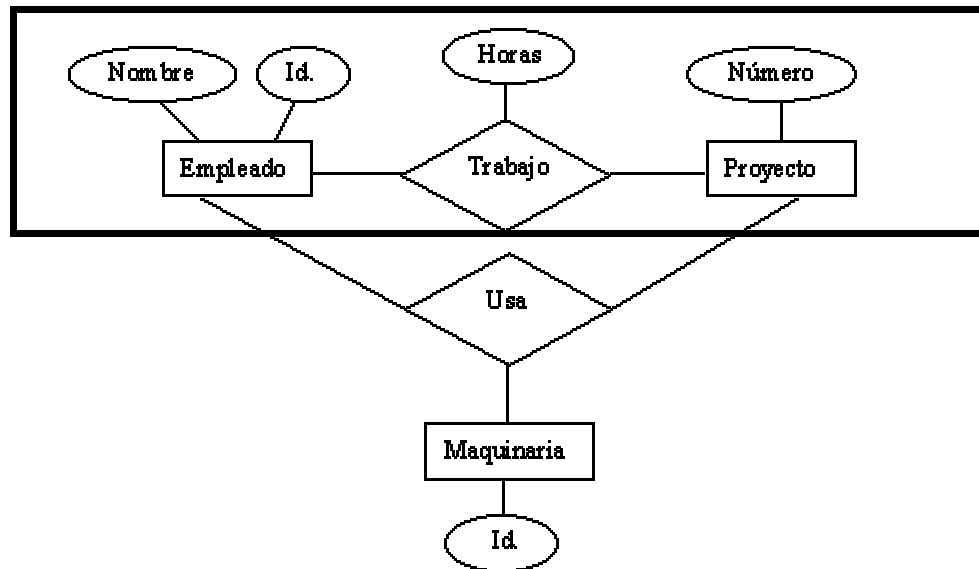


Diagrama E-R con agregación

Ahora podemos decir que la entidad trabajo se relaciona con la entidad maquinaria a través de la relación usar. Para indicarnos que un trabajo usa un determinado equipo o maquinaria según el tipo de trabajo que se trate.

Modelo relacional

La ventaja del modelo relacional es que los datos se almacenan, al menos conceptualmente, de un modo en que los usuarios entienden con mayor facilidad. Los datos se almacenan como tablas y las relaciones entre las filas y las tablas son visibles en los datos. Este enfoque permite a los usuarios obtener información de la base de datos sin asistencia de sistemas profesionales de administración de información.

Las características más importantes de los modelos relacionales son:

- a. Es importante saber que las entradas en la tabla tienen un solo valor (son atómicos); no se admiten valores múltiples, por lo tanto la intersección de un renglón con una columna tiene un solo valor, nunca un conjunto de valores.
- b. Todas las entradas de cualquier columna son de un solo tipo. Por ejemplo, una columna puede contener nombres de clientes, y en otra puede tener fechas de nacimiento. Cada columna posee un nombre único, el orden de las columnas no es de importancia para la tabla, las columnas de una tabla se conocen como atributos. Cada atributo tiene un dominio, que es una descripción física y lógica de valores permitidos.
- c. No existen 2 filas en la tabla que sean idénticas.
- d. La información en las bases de datos son representados como datos explícitos, no existen apuntadores o ligas entre las tablas.

En el enfoque relacional es sustancialmente distinto de otros enfoques en términos de sus estructuras lógicas y del modo de las operaciones de entrada/salida. En el enfoque relacional, los datos se organizan en tablas llamadas relaciones, cada una de las cuales se implanta como un archivo. En terminología relacional una fila en una relación representa un registro o una entidad; Cada columna en una relación representa un campo o un atributo.

Así, una relación se compone de una colección de entidades (o registros) cuyos propietarios están descritos por cierto número de atributos predeterminados implantados como campos.

3.1 Estructura de las bases de datos relacionales

La arquitectura relacional se puede expresar en términos de tres niveles de abstracción: nivel interno, conceptual y de visión.

La arquitectura relacional consta de los siguientes componentes:

1. Modelo relacional de datos:

En el nivel conceptual, el modelo relacional de datos está representado por una colección de relaciones almacenadas. Cada registro de tipo conceptual en un modelo relacional de datos se implanta como un archivo almacenado distinto.

2. Submodelo de datos:

Los esquemas externos de un sistema relacional se llaman submodelos relacionales de datos; cada uno consta de uno a más escenarios (vistas) para describir los datos requeridos por una aplicación dada. Un escenario puede incluir datos de una o más tablas de datos. Cada programa de aplicación está provisto de un buffer ("Área de trabajo de usuario") donde el DBMS puede depositar los datos recuperados de la base para su procesamiento, o puede guardar temporalmente sus salidas antes de que el DBMS las escriba en la base de datos.



3. **Esquema de almacenamiento:**

En el nivel interno, cada tabla base se implanta como un archivo almacenado. Para las recuperaciones sobre las claves principal o secundaria se pueden establecer uno o más índices para acceder un archivo almacenado.

4. **Sublenguaje de datos:**

Es un lenguaje de manejo de datos para el sistema relacional, el álgebra relacional y cálculo relacional, ambos lenguajes son "relacionalmente completos", esto es, cualquier relación que pueda derivarse de una o más tablas de datos, también se puede derivar con u solo comando del sublenguaje. Por tanto, el modo de operación de entrada/Salida en un sistema relacional se puede procesar en la forma: una tabla a la vez en lugar de: un registro a la vez; en otras palabras, se puede recuperar una tabla en vez de un solo registro con la ejecución de un comando del sublenguaje de datos.



3.2 Lenguajes de consulta formales.

Los lenguajes de consultas:

Son los lenguajes en el que los usuarios solicitan información de la base de datos. Estos lenguajes son generalmente de más alto nivel que los lenguajes de programación. Los lenguajes de consulta pueden clasificarse como *procedimentales* y *no procedimentales*;

En el lenguaje del tipo *procedimental* el usuario da las instrucciones al sistema para que realice una secuencia de operaciones en la base de datos para calcular el resultado deseado.

En el lenguaje *no procedimental*, el usuario describe la información deseada sin dar un procedimiento específico para obtener dicha información.

El álgebra relacional es un lenguaje de consulta formal procedimental, el álgebra relacional define operadores que funcionan sobre las tablas (de una manera similar a los operadores +, -, etc. del álgebra común) para llegar al resultado deseado. El álgebra relacional es difícil de utilizar, debido en parte a que es procedimental, esto es, al utilizar el álgebra relacional no sólo debemos *saber* lo que queremos, también *cómo* obtenerlo.

En el proceso de bases de datos comerciales el álgebra relacional se utiliza de manera poco frecuente. Aunque unos cuantos productos exitosos DBMS sí tienen opciones del álgebra relacional, éstas son poco utilizadas en vista de su complejidad.

El álgebra relacional toma dos o más tablas como entrada produce una nueva tabla como resultado de la serie de operaciones. Las operaciones fundamentales en el álgebra relacional son *seleccionar*, *proyectar*, *producto cartesiano*, *renombrar*, *unión* y *diferencia de conjuntos*. Además de las operaciones fundamentales existen otras operaciones como son: *intersección de conjuntos*, *producto natural*, *división* y *asignación*.

**** Operaciones fundamentales ****

Las operaciones seleccionar, proyectar y renombrar, son denominadas operaciones unitarias ya que operan sobre una tabla. Las otras operaciones operan sobre pares de relaciones y, por tanto se llaman operaciones binarias.

*** La operación seleccionar.**

Esta operación selecciona tuplas (filas) que satisfacen una instrucción(condición) dada de una tabla. Se representa por medio de paréntesis.

(nombre_tabla WHERE condición);

La oración de la instrucción después de la cláusula WHERE puede incluir condiciones de igualdad como =, <, >, >=, <=, además que se puede hacer una oración más compleja usando los conectores y (^) y o (v).

*** La operación Proyectar.**

Consiste en identificar las columnas (atributos en el modelo E-R) que nos interesa conocer. Se representa por medio de corchetes. Si este se omite indicara que se desea obtener todas las columnas de la tabla en cuestión.

(nombre_tabla WHERE condición) [Nombre_atributo];

* La operación Producto cartesiano.

Consiste en multiplicar todas las tuplas entre tablas, obteniendo como resultado una tabla que contiene todas las columnas de ambas tablas. Se especifica con la orden **TIMES**.

Nombre_tabla **TIMES** Nombre_tabla;

* La operación Join.

Consiste en obtener el producto (multiplicación) de todas las tuplas de una tabla con las de la otra, para posteriormente evaluar aquellas cuyo campo en común sea igual generando como resultado una nueva tabla que tiene como tuplas (renglones) que cumplen con la condición establecida. Se representa con la orden **JOIN**.

La orden Join es colocada entre las dos tablas a multiplicar después de que la primera especifica la operación de selección y proyección.

(Tabla)[atributo] **JOIN** (Tabla)[Atributo];

* La operación Divide.

Toma dos relaciones, una binaria y la otra unaria, construye una relación formada por todos los valores de un atributo de la relación binaria que concuerdan (en el otro atributo) con todos los valores de la relación unaria. Se representa con la orden **DIVIDEBY**.

NomTablaBin **DIVIDEBY** NomTablaUna

* La operación Diferencia.

Construye una relación formada por todas las tuplas (filas) de la primera relación que no aparezcan en la segunda de las dos relaciones especificadas. Se representa con la orden **MINUS**.

Nom_tablaA **MINUS** NomTablaB;

* La operación Unión.

Construye una relación formada por todas las tuplas de la primera relación y todas las tuplas de la segunda relación. El requisito es que ambas relaciones sean del mismo tipo.

Nom_TablaA **UNION** Nom_tablaB

* La operación intersección.

Construye una nueva tabla compuesta por todas las tuplas que están en la primera y segunda tabla.

Nom_TablaA INTERSEC Nom_tablaB

Ejemplos:

Para ejemplificar las notaciones anteriores consideremos el ejemplo ALUMNO - curso - MATERIA, que tienen los siguientes atributos:

<u>NControl</u>	<u>NControl</u>	<u>Clave</u>
<u>NombreA</u>	<u>Clave</u>	NombreM
Especialidad	Calif	Créditos
Dirección		

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

Tabla alumno:

NControl	NombreA	Especialidad	Dirección

Tabla cursa:

NControl	Clave	Calif

Tabla materia:

Clave	NombreM	Créditos

1.- Obtener el nombre de todos los alumnos que están inscritos en la Institución.

(Alumno) [NombreA];

2.- Obtener el nombre de los alumnos que cursan la materia Base de datos 1 cuya clave es SCB9333

(Alumno) JOIN (Cursa where Clave='SCB9333') [NombreA];

3.- Obtener los nombres de los alumnos de la especialidad de Ing. Sistemas que cursan la materia Base de datos 2.



```
((Alumno)[especialidad,NombreA,NControl]
JOIN (Cursa) where especialidad = 'ISC')[Clave,NombreA]
JOIN (Materia where NombreM='BD2')[NombreA];
```

En el álgebra relacional no solo debemos saber lo que queremos si no también como obtenerlo, al realizar las consultas debemos especificar el nombre de la tabla a utilizar en caso de que deseamos realizar una operación con un atributo que las otras tablas no tienen debemos "arrastrar" dicho atributo para poder utilizarlo, como lo es en el caso anterior, en donde requerimos el nombre del alumno que solamente lo tiene la tabla alumno, pero también deseamos que se cumpla la condición NombreM=BD2, como no podemos relacionar directamente a ambas tablas empleamos la tabla cursa de donde obtenemos la clave de las materias y mantenemos el nombre del alumno (NombreA) finalmente con la orden JOIN se combinan las tablas por el campo común que tienen que es clave así que obtenemos una tabla con todas las materias que cursan los alumnos de ISC, de donde seleccionamos solo aquella que se llame BD2 con la orden Join obtenemos esta nueva tabla de donde por último proyectamos el atributo NombreA que hemos venido "arrastrando".

Ejercicios propuestos:

Considere el modelo E-R del caso Médico - atiende - Paciente.

Realizar:

- * La conversión a tablas del modelo E-R.
- * Las siguientes consultas en álgebra relacional.

- 1.- Obtener el nombre de Todos los médicos.
- 2.- Obtener el nombre de todos los pacientes > de 18 años.
- 3.- Obtener todos los datos de todos los pacientes.
- 4.- Obtener los nombres de todos los pacientes que consultan con el médico con cédula profesional ABC001.
- 5.- Obtener los nombres de los médicos que atienden al paciente John Smith.
- 6.- Suponiendo que el hospital de la Ciudad de la Paz tiene una tabla de pacientes similar a la del hospital de San José, obtener el nombre y la afiliación de estos pacientes.
- 7.- Obtener las combinaciones de pacientes y médicos excepto la de aquellos médicos cuya especialidad sea Oftalmología.

Recuerde que tenemos que indicar las tablas a utilizar entre paréntesis y los atributos a proyectar entre corchetes, después podemos utilizar las ordenes Times, Join, Divide, Minus, Union, Intersec, según sea el caso a resolver; si requiere manipular atributos que no tengan las otras tablas "arrástrelos" proyectando siempre entre cada operación dicho atributo. De cada operación o combinación que realice entre las tablas se genera una tabla nueva que cumple con las condiciones que establece.

Solución a problemas de algebra relacional.

La conversión a tablas del modelo Medico-atende-Pacienre resulta:

Tabla Médico

Cédula	NombreM	Especialidad

Tabla Paciente

Afiliación	NombreP	Sexo	Edad

Tabla Atiende

Cédula	Afiliación	Fecha	Hora

La solución a las consultas son:

- 1.- (Medico)[NombreM];
- 2.- (Paciente where Edad>'18')[NombreP];
- 3.- (Paciente);
- 4.- (Paciente)[NombreP] JOIN (Atiende where Cédula='ABC001')[NombreP];
- 5.- ((Médico)[NombreM] JOIN (Atiende) JOIN (Paciente where NombreP='John Smith')) [NombreM];
- 6.- Consideremos 2 tablas de pacientes, Paciente1 y Paciente2, entonces:

 ((Paciente1) UNION (Paciente2))[NombreP,Afiliación];
- 7.- ((Paciente) TIMES (Médico where especialidad<> 'Oftalmología'));



3.3 Lenguajes de consulta comerciales

Un lenguaje de consulta comercial proporciona una interfaz más amigable al usuario. Un ejemplo de este tipo de lenguaje es el SQL, (Structured Query Lenguaje, Lenguaje de Consulta Estructurado).

Las partes más importantes del SQL son:

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

DML: Lenguaje de manipulación de datos (que nos permite tener acceso a las estructuras para suprimir, modificar e insertar)

En este apartado estudiaremos la forma básica para realizar consultas con SQL, en el apartado 3.4: Modificación de la base de datos, estudiaremos lo que concierne a la modificación de las tablas.

La estructura básica de una expresión en SQL contiene 3 partes, Select, From y Where.

La cláusula **Select** se usa para listar los atributos que se desean en el resultado de una consulta.

From, Lista las relaciones que se van a examinar en la evaluación de la expresión.

Where, es la definición de las condiciones a las que puede estar sujeta una consulta.

La consulta típica de SQL tiene la siguiente forma:

Select A1,A2,A3...An

From r1,r2,r3...rm

Where Condición(es)

Donde:

A1,A2,A3...An: Representan a cada atributo(s) o campos de las tablas de la base de datos relacional.

R1,r2,r3...rm: Representan a la(s) tabla(s) involucradas en la consulta.

Condición: Es el enunciado que rige el resultado de la consulta.

Si se omite la cláusula Where, la condición es considerada como verdadera, la lista de atributos (A1,A2..An) puede sustituirse por un asterisco (*), para seleccionar todos los atributos de todas las tablas que aparecen en la cláusula From.

Funcionamiento del SQL.

El SQL forma el producto cartesiano de las tablas involucradas en la cláusula From, cumpliendo con la condición establecida en la orden Where y después proyecta el resultado con la orden select.

Para nuestros ejemplos consideremos una tabla llamada CURSO, que contiene los siguientes campos:

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

Datos contenidos en la tabla CURSO

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



NumC	NombreC	DescC	Creditos	Costo	Depto
S05	Arquitectura de Computadoras	Para ISC	10	50.00	Sistemas
I01	Base de Datos Relacionales	Para ISC y LI	10	150.00	Informática

Ejemplos de consultas:

OBTENCIÓN DE UNA TABLA ENTERA

- Obtener toda la información disponible sobre un curso donde Costo sea 0.

```
SELECT *
FROM CURSO
WHERE Costo=0.00
```

Resultado de la consulta anterior.

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

Colocamos un * debido a que no nos limitan la información de la tabla, es decir nos piden que mostremos todos los datos atributo de la tabla CURSO.

Como la única condición en la sentencia WHERE es que la tarifa del curso sea igual a 0, esta consulta regresa todas las tuplas donde se encuentre que Costo = 0.00.

Debido a que Costo es un campo numérico, la condición solo puede comparar con campos del mismo tipo. Para representar valores negativos se antepone a la izquierda el signo (-), en este ejemplo se considera solo el signo (=) para establecer la condición, sin embargo otros operadores que se pueden utilizar son:

Menor que <

Mayor que >

Menor o igual que <=

Mayor o igual que >=

Diferente <>

Además de los operadores booleanos AND, NOT, OR.

Cabe señalar que en la sentencia Where cuando se requiere establecer condiciones con cadenas, estas son delimitadas por apóstrofes ("). Las expresiones de cadenas son comparadas carácter por carácter, dos cadenas son iguales solo si coinciden todos los caracteres de las mismas.

Ejemplos de consultas con cadenas:

- Obtener toda la información sobre cualquier curso que ofrezca el departamento de Ciencias.



```
SELECT *
FROM CURSO
WHERE Depto = 'Ciencias';
```

Resultado de la consulta.

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

VISUALIZACIÓN DE COLUMNAS ESPECIFICADAS.

En los ejemplos anteriores obteníamos toda la tabla completa, ahora veremos como mostrar solo algunos atributos específicos de una tabla.

- Obtener los valores NumC, NombreC y Depto, en este orden de toda la tabla curso.

```
SELECT NumC, NombreC, Depto
FROM CURSO;
```

Resultado de la consulta:

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



NumC	NombreC	Depto
I01	Base de Datos Relacionales	Informática

Observamos que en este caso no se tiene la sentencia *Where*, no existe condición, por lo tanto, todas las filas de la tabla *CURSO* se recuperan, pero solo se visualizaran las tres columnas especificadas.

Así mismo, empleamos la (,) para separar los campos que deseamos visualizar.

VISUALIZACIÓN DE UN SUBCONJUNTO DE FILAS Y COLUMNAS

- Seleccionar los valores *NumC*, *Depto* y *Costo* para todos los cursos que tengan un *Costo* inferior a \$100

```
SELECT NumC, Depto, Costo
FROM CURSO
WHERE Costo < 100.00
```

Como resultado de esta consulta se obtendrán todas aquellas tuplas que tengan un costo en *CTARIFA* menor que 100, y se visualizaran solo los campos de *NumC*, *Depto*, *Costo*. Podemos observar que este ejemplo cubre el formato general de una consulta *SQL*.

La palabra clave **DISTINCT**

DISTINCT, es una palabra reservada que elimina las filas que duplicadas en el resultado de una consulta.

- Visualizar todos los departamentos académicos que ofrezcan cursos, rechazando los valores duplicados.

```
SELECT DISTINCT Depto
FROM CURSO;
```

Resultado de la consulta

Depto
Administración
Sistemas
Ciencias
Bioquímica
electromecánica
Arquitectura
Informática

La palabra **DISTINCT** va estrictamente después de la palabra **SELECT**.

De no haberse utilizado la palabra **DISTINCT**, el resultado hubiera mostrado todas las tuplas del atributo *Depto* que se encontraran, es decir, se hubiera visualizado la columna de *Depto* completamente.

EMPLEO DE LOS CONECTORES BOOLEANOS (AND, OR, NOT)

Para emplear las condiciones múltiples dentro de la sentencia *WHERE*, utilizamos los conectores lógicos.



El conector **AND**.

Este conector pide al sistema que seleccione una sola columna únicamente si ambas condiciones se cumplen.

- Obtener toda la información sobre todos los cursos que ofrece el departamento Sistemas que tengan una tarifa igual a 0.

```
SELECT *  
FROM CURSO  
WHERE Depto='Sistemas' AND  
Costo=0.00;
```

El resultado de esta consulta sería todas aquellas tuplas que cumplan exactamente con las dos condiciones establecidas.

El conector **OR**.

Este conector al igual que el AND permite conectar condiciones múltiples en la sentencia WHERE, a diferencia del conector AND, el OR permite la selección de filas que cumplan con una sola de las condiciones establecidas a través de este conector.

- Obtener toda la información existente sobre cualquier curso ofrecido por los departamentos Arquitectura o Bioquímica.

```
SELECT *  
FROM CURSO  
WHERE Depto = 'Arquitectura'  
OR Depto= 'Bioquímica';
```

El resultado de esta consulta será la de visualizar todas aquellas tuplas donde se cumpla cualquiera de las 2 condiciones, es decir mostrara todas las tuplas que tengan en el atributo Depto=Arquitectura o Bioquímica.

El conector **NOT**

Este nos permite marcar aquellas tuplas que por alguna razón no deseamos visualizar.

- Obtener el nombre del curso y del departamento de todos los cursos que no sean ofrecidos por el departamento Sistemas.

```
SELECT NombreC, Depto  
FROM CURSO  
WHERE NOT (Depto='Sistemas');
```

JERARQUIA DE OPERADORES BOOLEANOS.

En orden descendente (de mayor a menor prioridad)

N
O
T

A
N
D



O
R

Existen dos formas para realizar consultas: Join de Querys y Subquerys.

Cuando en la sentencia **From** colocamos los nombres de las tablas separados por comas se dice que efectuamos una consulta de la forma **Join de Querys**, en este caso se requiere anteponer el nombre de la tabla y un punto al nombre del atributo. En el Join de Querys el resultado que se produce con las tablas que intervienen en la consulta es la concatenación de las tablas, en donde los valores de una columna de la primera tabla coinciden con los valores de una segunda tabla, la tabla de resultado tiene una fila por cada valor coincidente que resulte de las dos tablas originales.

Para ejemplificar esto, consideremos 2 tablas: Tabla1 y Tabla2, entonces:

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

Resultado de la operación Join:

C1	C2	C3	CA	CB
A	AAA	10	10	S
D	DDD	20	20	U
E	EEE	20	20	U
F	FFF	90	90	V
F	FFF	90	90	W
F	FFF	90	90	Y
H	HHH	90	90	V
H	HHH	90	90	W
H	HHH	90	90	Y

Como podemos observar, la comparación se efectuó por las columnas C3 y CA, que son donde se encontraron valores iguales, el resultado muestra una tupla por cada coincidencia encontrada.

Cuando las consultas se anidan se conoce como **Subquerys** o subconsultas. Este tipo de consulta obtiene resultados parciales reduciendo el espacio requerido para realizar una consulta.

Nota: Todas las consultas que se resuelven con subqueries pueden resolverse con Join de Querys, pero no todas las consultas hechas con Join de Querys pueden resolverse utilizando Subqueries.

Para ejemplificar lo anterior consideremos el ejemplo

ALUMNO - cursa - MATERIA, que tienen los siguientes atributos:
 NControl NControl Clave
 NombreA Clave NombreM
 Especialidad Calif Creditos
 Dirección

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

Tabla alumno:

NControl	NombreA	Especialidad	Dirección

Tabla cursa:

NControl	Clave	Calif

Tabla materia:

Clave	NombreM	Creditos

- Obtener el nombre de la materia que cursa el alumno con número de control 97310211 con créditos igual a ocho.

```
SELECT NombreA
FROM Materia
WHERE creditos='8' and clave in(SELECT clave
                                FROM cursa
                                WHERE NControl='97310211');
```

- Obtener el número de control del alumno que tenga alguna calificación igual a 100

```
SELECT DISTINCT(NControl)
FROM Cursa
WHERE Calif='100';
```

- Obtener el nombre de las materias que cursa el alumno Salvador Chávez.

```
SELECT NombreM
FROM Materia
WHERE Clave in (SELECT DISTINCT (Clave)
                FROM Cursa
                WHERE NControl in (SELECT NControl)
                                FROM Alumno
                                WHERE NombreA='Salvador
                                Chávez'));
```

FUNCIONES AVANZADAS APLICABLES A CONSULTAS

Existen funciones que permiten la agilización de consultas similares a una hoja de cálculo, ya que trabajan en base a renglones y columnas.



COUNT (): Cuenta el número de tuplas en la columna establecida

MIN (): Localiza el valor mínimo de la columna establecida

MAX (): Localiza el valor máximo de la columna establecida.

AVG (): Obtiene el promedio de valores de la columna establecida

SUM (): Obtiene el valor total que implican los valores obtenidos en la columna establecida.

Ejemplos:

- Obtener el número de alumnos que existen en la carrera de Ingeniería en Sistemas Computacionales.

```
SELECT Count (*)
FROM Alumno
WHERE especialidad='ISC';
```

- Obtener la máximo calificación que ha obtenido J.M. Cadena.

```
SELECT Max(Calif)
FROM Cursa
WHERE NControl IN (SELECT NControl
FROM Alumno
WHERE NombreA= 'J.M. Cadena ');
```

- Obtener el promedio de calificaciones de Salvador Chávez.

```
SELECT Avg (Calif)
FROM Cursa
WHERE NCotrol IN (SELECT NControl
FROM Alumno
WHERE NombreA='Salvador Chávez');
```

- Obtener la suma total de las calificaciones obtenidas por Daniel Colín.

```
SELECT Sum (Calif)
FROM Cursa
WHERE NControl IN (SELECT NControl
FROM Alumno
WHERE NombreA='Daniel Colín');
```

Hasta aquí hemos visto el manejo sencillo de realizar consultas con SQL, hay que destacar que en la realización de consultas anidadas se tiene que poner cuidado a la prioridad de los operadores, teniendo cuidado también al momento de agrupar los paréntesis que involucran las condiciones con los operadores.



3.4 Modificación de la Base de datos

Como se mencionó al inicio de este apartado del SQL, éste cuenta con módulos DDL, para la definición de datos que nos permite crear o modificar la estructura de las tablas.

Las instrucciones para realizar estas operaciones son:

- CREATE TABLE:** Nos permite crear una tabla de datos vacía.
- INSERT:** Permite almacenar registros en una tabla creada.
- UPDATE:** Permite modificar datos de registros almacenados en la tabla.
- DELETE:** Borra un registro entero o grupo de registros de una tabla.
- CREATE INDEX:** Crea un índice que nos puede auxiliar para las consultas.
- DROP TABLE:** Permite borrar una tabla.
- DROP INDEX:** Borra el índice indicado.

Para ejemplificar las instrucciones anteriores consideremos el ejemplo ALUMNO - cursa - MATERIA, que tienen los siguientes atributos:

<u>NControl</u>	<u>NControl</u>	<u>Clave</u>
NombreA	<u>Clave</u>	NombreM
Especialidad	Calif	Creditos
Dirección		

* Estructura de la sentencia CREATE TABLE.

```
CREATE TABLE <Nombre de la tabla>
(
  Atributo1: tipo de dato longitud ,
  Atributo2: tipo de dato longitud ,
  Atributo3: tipo de dato longitud ,
  :
  :
  Atributoñ: tipo de dato longitud ,

  PRIMARY KEY (Opcional) );
```

Los campos pueden definirse como NOT NULL de manera opcional excepto en la llave primaria para lo cual es obligatorio. Además al definir la llave primaria se genera automáticamente un índice con respecto al campo llave; para definir la llave la denotamos dentro de los paréntesis de PRIMARY KEY.

Ejemplo:

Crear la tabla alumno con los atributos antes descritos, tomando como llave el numero de control.

```
CREATE TABLE Alumno
(
NControl char(8) NOT NULL,
NombreA char(20),
Especialidad char(3),
Dirección char(30),
PRIMARY KEY (NControl) );
```

Tabla Alumno:

NControl	NombreA	Especialidad	Dirección

Pueden existir más de una llave primaria, esto es si se requiere, se crearán tantos índices como llaves primarias se establezcan.

Pueden existir tantos campos Not Null (No nulos) como se requieran; En si estructurar la creación de una tabla es siempre parecida al ejemplo anterior.

* Estructura de la sentencia INSERT

```
INSERT
INTO Nombre de la tabla a la que se le va a insertar el registro
VALUES (Conjunto de valores del registro) ;
```

Ejemplo:

Insertar en la tabla Alumno, antes creada los datos del alumno Daniel colín, con numero de control 95310518 de la especialidad de Ingeniería civil, con domicilio Abasolo Norte #45.

```
INSERT
INTO Alumno
VALUES("95310518","Daniel Colín","IC","Abasolo Norte #45") ;
```

Nótese que la inserción de los datos se realiza conforme la estructura que se implanto en la tabla, es decir en el orden en que se creo dicha tabla. En caso de querer omitir un dato que no sean no nulos solamente se ponen las comillas indicando el vacío de la cadena.

* Estructura de la Sentencia CREATE INDEX

```
CREATE INDEX Nombre que se le asignara al índice.
ON Nombre de la taba a la cual se le creara el índice (Campo(s) por el cual se creara el índice);
```

Ejemplo:

Crear un índice de la tabla Alumno por el campo Especialidad.

```
CREATE INDEX Indice1
ON Alumno(Especialidad);
Este índice contendrá a todos los alumnos ordenados por el campo especialidad.
CREATE INDEX UNIQUE INDEX Indice2
ON Alumno (Especialidad);
```

En la creación de este índice utilizamos la sentencia UNIQUE, es un indicador para permitir que se cree un índice único por especialidad, esta sentencia siempre se coloca antes de CREATE INDEX. En este ejemplo se creara un índice que contenga un alumno por especialidad existente.

* Estructura de la sentencia UPDATE

```
UPDATE Nombre de la tabla en donde se modificaran los datos.
SET Valores
WHERE (Condición);
```

Ejemplo:

Modificar el número de control del registro de Daniel Colín de la Tabla alumno por el número 96310518.

```
UPDATE Alumno
SET NControl '96310518'
WHERE NombreA='Daniel Colín';
```

* Estructura de la sentencia DROP TABLE

```
DROP TABLE Nombre de la tabla a borrar ;
```

Ejemplo:

Borrar la tabla Alumno creada anteriormente.

```
DROP TABLE Alumno;
```

* Estructura de la sentencia DROP INDEX

```
DROP INDEX Nombre del índice a borrar;
```

Ejemplo:

Borrar el índice Indice1 creado anteriormente.

```
DROP INDEX Indice1;
```

* Estructura de la sentencia DELETE

```
DELETE
FROM Nombre de la tabla
WHERE Condición;
```

Ejemplos:

- Borrar el registro cuyo número de control es 95310386.

```
DELETE
FROM Alumno
WHERE Control='95310386';
```

- Borrar todos los registros de la tabla alumno.

```
DELETE
FROM Alumno;
```

En el primer ejemplo, se borrara todo el registro(todos los datos), del alumno con número de control = 95310386.

En el segundo ejemplo se borrarán todos los registros de la tabla alumno, pero sin borrar la estructura de la tabla, ya que la orden Delete solo borra registros, la sentencia Drop Table es la que borra toda la estructura de la tabla junto con los registros de la misma.



3.5 Vistas.

Una vista se define en SQL usando la orden CREATE VIEW. Para definir una vista debemos dar a la vista un nombre y declarar la consulta que calcula la vista. Una vez que establecemos una vista, podemos ejecutar una sentencia SELECT que referencie a esa vista. El sistema asociará la vista SQL con una tabla base y extraerá y visualizará, entonces, los datos de la tabla base.

Esto significa que una vista no contiene datos duplicados de una tabla base. No tiene absolutamente ningún dato, puesto que no es una tabla real, todo el proceso se realiza con los datos almacenados en la tabla base. Es decir se percibe como una tabla virtual.

Las ordenes que se utilizan para la manipulación de vistas son:

CREATE VIEW: Crea una tabla virtual.

DROP VIEW: Elimina una vista creada anteriormente.

 Estructura de la sentencia CREATE VIEW.

CREATE VIEW Nombre de la vista AS (Expresión de consulta);

Para nuestros ejemplos consideremos de nuevo la tabla llamada CURSO, que contiene los siguientes campos:

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

Que contiene los siguientes datos:

NumC	NombreC	DescC	Creditos	Costo	Depto
A01	Liderazgo	Para público General	10	100.00	Admón.
S01	Introducción a la inteligencia artificial	Para ISC y LI	10	90.00	Sistemas.
C01	Construcción de torres	Para IC y	8	0.00	Ciencias



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

Ejemplos:

* Crear una vista (tabla virtual), denominada CursosS, que contenga las filas solo correspondientes a cursos ofrecidos por el departamento Sistemas. La vista deberá contener todas las columnas de la tabla CURSO, con la excepción de la columna Depto, la secuencia, de izquierda a derecha de las columnas, deberá ser: NombreC, NumC, Creditos, Costo Y DescC.

```
CREATE VIEW CursosS AS
SELECT NombreC,NumC,Creditos,Costo,DescC
FROM CURSO
WHERE DescC='Sistemas';
```

Observemos que después del nombre de la vista ponemos la sentencia AS, esto para definir la estructura de la vista, la estructura en si de la vista esta formada por la consulta anteriormente vista utilizando la orden SELECT.

* Crear una vista denominada CursosCaros, correspondientes a las filas de la tabla CURSO, en donde la tarifa exceda de \$150, las columnas de la vista deberán tener los nombres ClaveCurso, NombreCurso y CostoCaro.

```
CREATE VIEW CursosSCaros(ClaveCurso,NombreCurso,CostoCaro) As
SELECT NumC,NombreC, Costo
FROM Curso
WHERE Costo > 150;
```


Diseño de Bases de Datos relacionales

4.1 Peligros en el diseño de bases de datos relacionales.

Uno de los retos en el diseño de la base de datos es el de obtener una estructura estable y lógica tal que:

1. El sistema de base de datos no sufra de anomalías de almacenamiento.
2. El modelo lógico pueda modificarse fácilmente para admitir nuevos requerimientos.

Una base de datos implantada sobre un modelo bien diseñado tiene mayor esperanza de vida aun en un ambiente dinámico, que una base de datos con un diseño pobre. En promedio, una base de datos experimenta una reorganización general cada seis años, dependiendo de lo dinámico de los requerimientos de los usuarios. Una base de datos bien diseñada tendrá un buen desempeño aunque aumente su tamaño, y será lo suficientemente flexible para incorporar nuevos requerimientos o características adicionales.

Existen diversos riesgos en el diseño de las bases de datos relacionales que afecten la funcionalidad de la misma, los riesgos generalmente son la redundancia de información y la inconsistencia de datos.

La normalización es el proceso de simplificar la relación entre los campos de un registro. Por medio de la normalización un conjunto de datos en un registro se reemplaza por varios registros que son más simples y predecibles y, por lo tanto, más manejables. La normalización se lleva a cabo por cuatro razones:

- Estructurar los datos de forma que se puedan representar las relaciones pertinentes entre los datos.
- Permitir la recuperación sencilla de los datos en respuesta a las solicitudes de consultas y reportes.
- Simplificar el mantenimiento de los datos actualizándolos, insertándolos y borrándolos.
- Reducir la necesidad de reestructurar o reorganizar los datos cuando surjan nuevas aplicaciones.

En términos más sencillos la normalización trata de simplificar el diseño de una base de datos, esto a través de la búsqueda de la mejor estructuración que pueda utilizarse con las entidades involucradas en ella.

Pasos de la normalización:

1. Descomponer todos los grupos de datos en registros bidimensionales.
2. Eliminar todas las relaciones en la que los datos no dependan completamente de la llave primaria del registro.
3. Eliminar todas las relaciones que contengan dependencias transitivas.

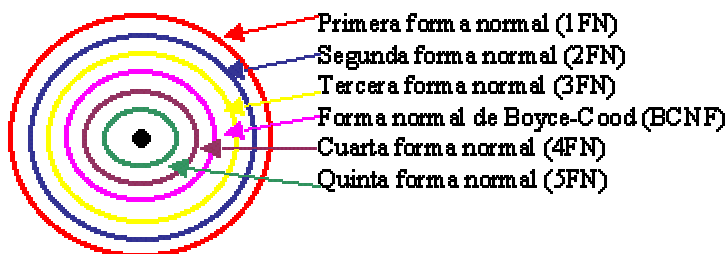
La teoría de normalización tiene como fundamento el concepto de formas normales; se dice que una relación está en una determinada forma normal si satisface un conjunto de restricciones.

4.2 Primera y segunda formas normales.

Formas normales.

Son las técnicas para prevenir las anomalías en las tablas. Dependiendo de su estructura, una tabla puede estar en primera forma normal, segunda forma normal o en cualquier otra.

Relación entre las formas normales:



Primera forma normal.

Definición formal:

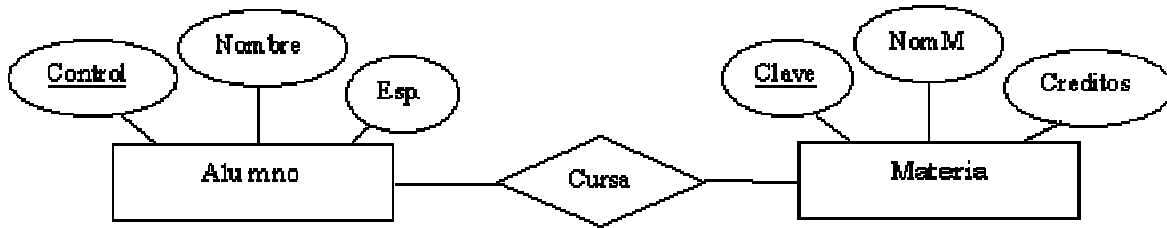
Una relación R se encuentra en 1FN si y solo si por cada renglón columna contiene valores atómicos.

Abreviada como 1FN, se considera que una relación se encuentra en la primera forma normal cuando cumple lo siguiente:

1. Las celdas de las tablas poseen valores simples y no se permiten grupos ni arreglos repetidos como valores, es decir, contienen un solo valor por cada celda.
2. Todos los ingresos en cualquier columna (atributo) deben ser del mismo tipo.
3. Cada columna debe tener un nombre único, el orden de las columnas en la tabla no es importante.
4. Dos filas o renglones de una misma tabla no deben ser idénticas, aunque el orden de las filas no es importante.

Por lo general la mayoría de las relaciones cumplen con estas características, así que podemos decir que la mayoría de las relaciones se encuentran en la primera forma normal.

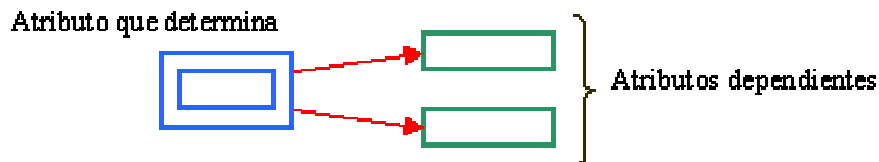
Para ejemplificar como se representan gráficamente las relaciones en primera forma normal consideremos la relación alumno cursa materia cuyo diagrama E-R es el siguiente:



Como esta relación maneja valores atómicos, es decir un solo valor por cada uno de los campos que conforman a los atributos de las entidades, ya se encuentra en primera forma normal, gráficamente así representamos a las relaciones en 1FN.

Segunda forma normal.

Para definir formalmente la segunda forma normal requerimos saber que es una **dependencia funcional**: Consiste en edificar que atributos dependen de otro(s) atributo(s).

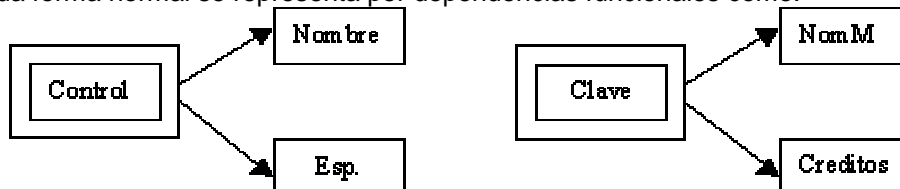


Definición formal:

Una relación R está en 2FN si y solo si está en 1FN y los atributos no primos dependen funcionalmente de la llave primaria.

Una relación se encuentra en segunda forma normal, cuando cumple con las reglas de la primera forma normal y todos sus atributos que no son claves (llaves) dependen por completo de la llave. De acuerdo con esta definición, cada tabla que tiene un atributo único como llave, esta en segunda forma normal.

La segunda forma normal se representa por dependencias funcionales como:



Nótese que las llaves primarias están representadas con doble cuadro, las flechas nos indican que de estos atributos se puede referenciar a los otros atributos que dependen funcionalmente de la llave primaria.



4.3 Tercera forma normal y la forma normal de Boyce Codd.

Para definir formalmente la 3FN necesitamos definir **dependencia transitiva**: En una afinidad (tabla bidimensional) que tiene por lo menos 3 atributos (A,B,C) en donde A determina a B, B determina a C pero no determina a A.

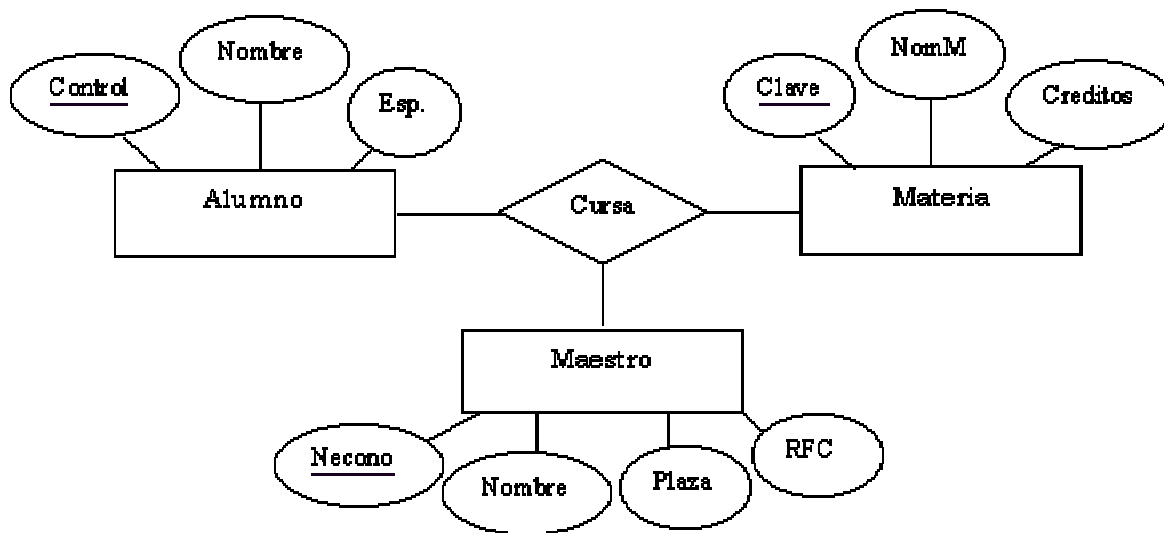
Tercera forma normal.

Definición formal:

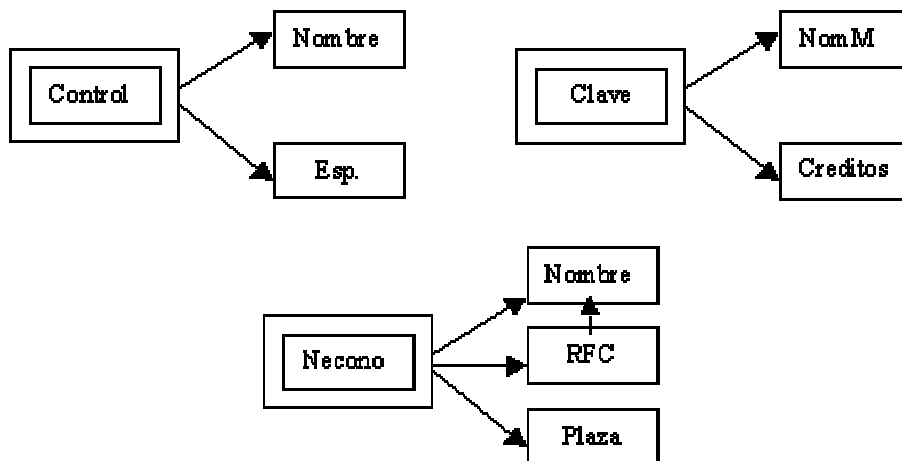
Una relación R está en 3FN si y solo si esta en 2FN y todos sus atributos no primos dependen no transitivamente de la llave primaria.

Consiste en eliminar la dependencia transitiva que queda en una segunda forma normal, en pocas palabras una relación esta en tercera forma normal si está en segunda forma normal y no existen dependencias transitivas entre los atributos, nos referimos a dependencias transitivas cuando existe más de una forma de llegar a referencias a un atributo de una relación.

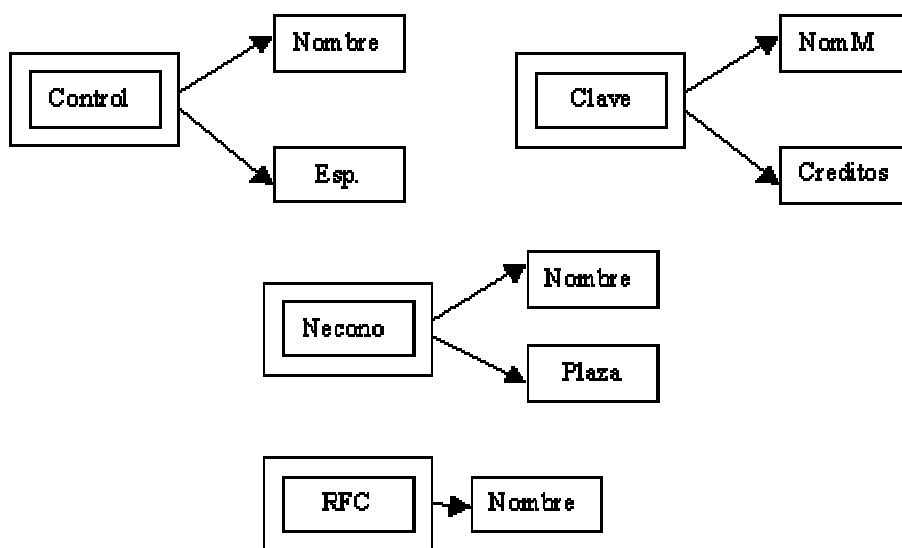
Por ejemplo, consideremos el siguiente caso:



Tenemos la relación alumno-cursa-materia manejada anteriormente, pero ahora consideramos al elemento maestro, gráficamente lo podemos representar de la siguiente manera:



Podemos darnos cuenta que se encuentra graficado en segunda forma normal, es decir que todos los atributos llave están indicados en doble cuadro indicando los atributos que dependen de dichas llaves, sin embargo en la llave Necono tiene como dependientes a 3 atributos en el cual el nombre puede ser referenciado por dos atributos: Necono y RFC (Existe dependencia transitiva), podemos solucionar esto aplicando la tercera forma normal que consiste en eliminar estas dependencias separando los atributos, entonces tenemos:



Forma normal de Boyce Codd.

Determinante: Uno o más atributos que, de manera funcional, determinan otro atributo o atributos. En la dependencia funcional $(A,B) \rightarrow C$, (A,B) son los determinantes.

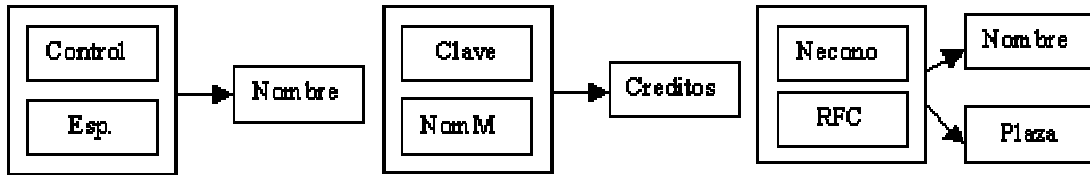
Definición formal:

Una relación R esta en FNBC si y solo si cada determinante es una llave candidato.

Denominada por sus siglas en ingles como BCNF; Una tabla se considera en esta forma si y sólo si cada determinante o atributo es una llave candidato.

Continuando con el ejemplo anterior, si consideramos que en la entidad alumno sus atributos control y nombre nos puede hacer referencia al atributos esp., entonces decimos que dichos atributos pueden ser llaves candidato.

Gráficamente podemos representar la forma normal de Boyce Codd de la siguiente forma:



Obsérvese que a diferencia de la tercera forma normal, agrupamos todas las llaves candidato para formar una global (representadas en el recuadro) las cuales hacen referencia a los atributo que no son llaves candidato.

4.4 Cuarta y quinta formas normales

Cuarta forma normal.

Definición formal:

Un esquema de relaciones R está en 4FN con respecto a un conjunto D de dependencias funcionales y de valores múltiples sí, para todas las dependencias de valores múltiples en D de la forma $X \twoheadrightarrow Y$, donde $X \subseteq R$ y $Y \subseteq R$, se cumple por lo menos una de estas condiciones:

- * $X \twoheadrightarrow Y$ es una dependencia de valores múltiples trivial.
- * X es una superllave del esquema R.

Para entender mejor aún esto consideremos una afinidad (tabla) llamada estudiante que contiene los siguientes atributos: Clave, Especialidad, Curso tal y como se demuestra en la siguiente figura:

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

Suponemos que los estudiantes pueden inscribirse en varias especialidades y en diversos cursos. El estudiante con clave S01 tiene su especialidad en sistemas y Bioquímica y toma los cursos de Natación y danza, el estudiante B01 tiene la especialidad en Bioquímica y toma el curso de Guitarra, el estudiante con clave C03 tiene la especialidad de Civil y toma el curso de natación.

En esta tabla o relación no existe dependencia funcional porque los estudiantes pueden tener distintas especialidades, un valor único de clave puede poseer muchos valores de especialidades al igual que de valores de cursos. Por lo tanto existe **dependencia de valores múltiples**. Este tipo de dependencias produce redundancia de datos, como se puede apreciar en la tabla anterior, en donde la clave S01 tiene tres registros para mantener la serie de datos en forma independiente lo cual ocasiona que al realizarse una actualización se requiera de demasiadas operaciones para tal fin.

Existe una dependencia de valores múltiples cuando una afinidad tiene por lo menos tres atributos, dos de los cuales poseen valores múltiples y sus valores dependen solo del tercer atributo, en otras palabras en la afinidad R (A,B,C) existe una dependencia de valores múltiples si A determina valores múltiples de B, A determina valores múltiples de C, y B y C son independientes entre sí.

En la tabla anterior Clave determina valores múltiples de especialidad y clave determina valores múltiples de curso, pero especialidad y curso son independientes entre sí.

Las dependencias de valores múltiples se definen de la siguiente manera: Clave \twoheadrightarrow Especialidad y Clave \twoheadrightarrow Curso; Esto se lee "Clave multidetermina a Especialidad, y clave multidetermina a Curso"

Para eliminar la redundancia de los datos, se deben eliminar las dependencias de valores múltiples. Esto se logra construyendo dos tablas, donde cada una almacena datos para solamente uno de los atributos de valores múltiples.

Para nuestro ejemplo, las tablas correspondientes son:

Tabla Eespecialidad

Clave	Especialidad
S01	Sistemas
B01	Bioquímica
C03	Civil

Tabla ECurso

Clave	Curso
S01	Natación
S01	Danza
B01	Guitarra
C03	Natación

Quinta forma normal.

Definición formal:

Un esquema de relaciones R está en 5FN con respecto a un conjunto D de dependencias funcionales, de valores múltiples y de producto, si para todas las dependencias de productos en D se cumple por lo menos una de estas condiciones:

- * (R1, R2, R3, ... Rn) es una dependencia de producto trivial.
- * Toda Ri es una superllave de R.

La quinta forma normal se refiere a dependencias que son extrañas. Tiene que ver con tablas que pueden dividirse en subtablas, pero que no pueden reconstruirse.

Modelo de datos de red

5.1 Conceptos básicos.

Una base de datos de red como su nombre lo indica, esta formado por una colección de registros, los cuales están conectados entre sí por medio de enlaces. El registro es similar a una entidad como las empleadas en el modelo entidad-relación.

Un *registro* es una colección de campos (atributos), cada uno de los cuales contiene solamente almacenado un solo valor, el *enlace* es la asociación entre dos registros exclusivamente, así que podemos verla como una relación estrictamente binaria.

Una estructura de datos de red, llamada algunas veces estructura plex, abarca más que la estructura de árbol porque un nodo hijo en la estructura de red puede tener más de un padre. En otras palabras, la restricción de que en un árbol jerárquico cada hijo puede tener un solo padre, se hace menos severa. Así, la estructura de árbol se puede considerar como un caso especial de la estructura de red tal como lo muestra la siguiente figura.

Para ilustrar la estructura de los registros en una base de datos de red, consideremos la base de datos alumno-materia, los registros en lenguaje Pascal entonces quedarían como:

```
type alumno= record
    NombreA:string[30];
    Control:string[8];
    Esp: string[3];
end;
```

```
type materia = record
    Clave:string[7];
    NombreM:string[25];
    Cred=string[2];
end;
```

5.2 Diagramas de estructura de datos.

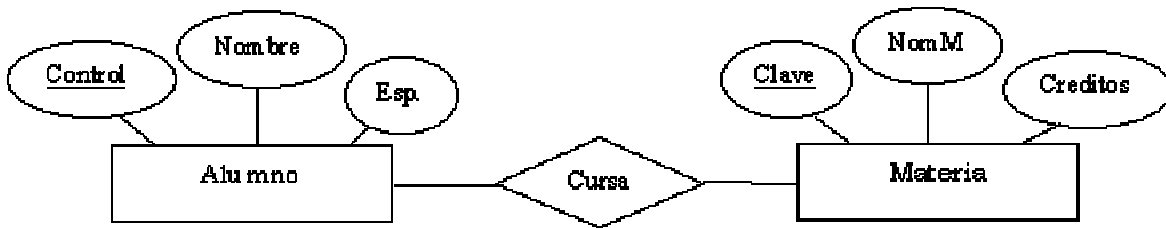
Un diagrama de estructura de datos es un esquema que representa el diseño de una base de datos de red. Este modelo se basa en representaciones entre registros por medio de ligas, existen relaciones en las que participan solo dos entidades (binarias) y relaciones en las que participan más de dos entidades (generales) ya sea con o sin atributo descriptivo en la relación.

La forma de diagramado consta de dos componentes básicos:

- Celdas: representan a los campos del registro.
- Líneas: representan a los enlaces entre los registros.

Un diagrama de estructura de datos de red, especifica la estructura lógica global de la base de datos; su representación gráfica se basa en el acomodo de los campos de un registro en un conjunto de celdas que se ligan con otro(s) registro(s), ejemplificaremos esto de la siguiente manera:

Consideremos la relación alumno-cursa-materia donde la relación cursa no tiene atributos descriptivos :



Las estructuras de datos según la cardinalidad se representan en los siguientes casos:

Cuando el enlace no tiene atributos descriptivos

Caso 1. Cardinalidad Uno a Uno.

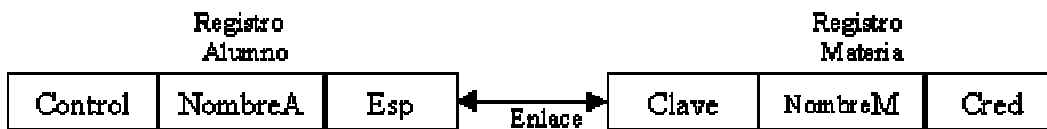


Diagrama de estructura de datos

Caso 2. Cardinalidad Muchos a uno.

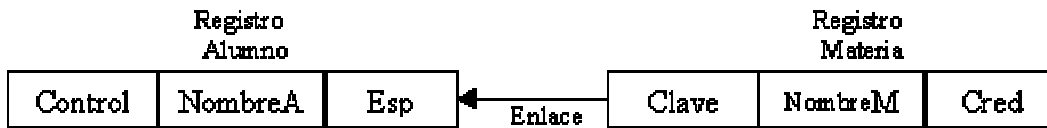


Diagrama de estructura de datos

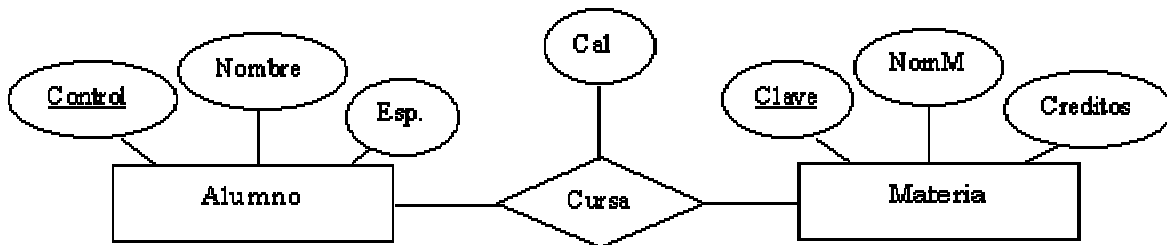
Caso 3. Cardinalidad Muchos a muchos.



Diagrama de estructura de datos

Cuando el enlace tiene atributos descriptivos.

Consideremos que a la relación cursa le agregamos el atributo Cal (calificación), nuestro modelo E-R quedaría de la siguiente manera:



La forma de convertir a diagramas de estructura de datos consiste en realizar lo siguiente:

1. Realizar la representación de los campos del registro agrupándolos en sus celdas correspondientes.
2. Crear nuevo registro, denominado *Calif*, para este caso, con un solo campo, el de cal (calif).
3. Crear los enlaces indicando la cardinalidad de :

- AluCal, del registro Calif al registro *Alumno*.
- MatCal, del registro *Calif* al registro *Materia*.

AluCal y *MatCal* son solo los nombres que emplearemos para identificar el enlace, pueden ser otros y no son empleados para otra cosa.

Los diagramas de estructuras de datos según la cardinalidad se transforman en:

Caso 1. Cardinalidad uno a uno.

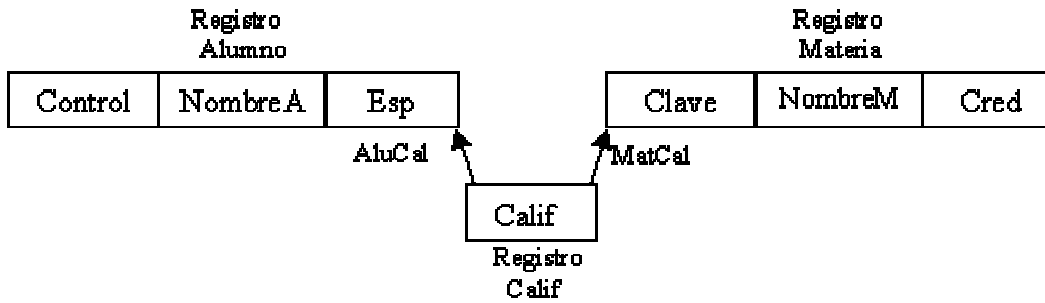


Diagrama de estructura de datos

Caso 2. Cardinalidad Uno a muchos.

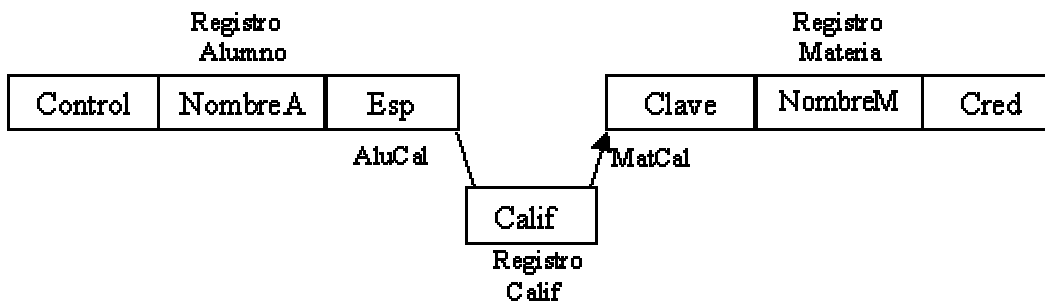


Diagrama de estructura de datos

Caso 3. Cardinalidad Muchos a muchos.

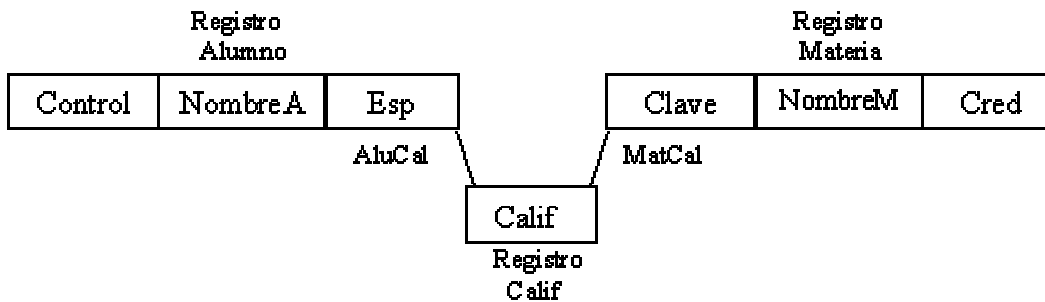
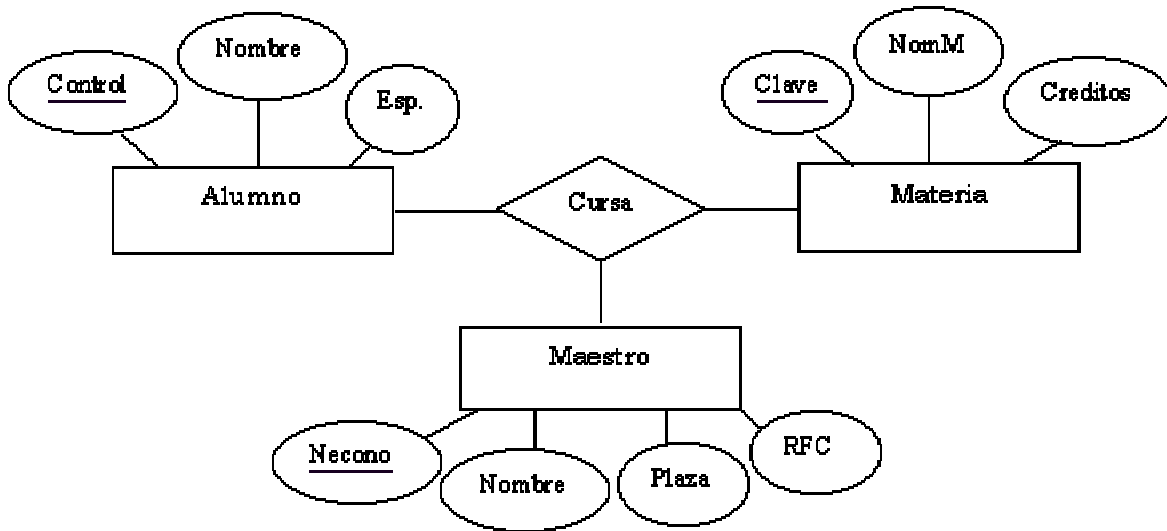


Diagrama de estructura de datos

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

Consideremos que a la relación alumno-cursa-materia le agregamos la entidad maestro, quien es el que imparte dicha materia.

Nuestro diagrama E-R quedaría de la siguiente manera:



La transformación a diagramas de estructura de datos se realiza mediante los siguientes pasos:

1. Crear los respectivos registros para cada una de las entidades que intervienen en el modelo.
2. Crear un nuevo tipo de registro que llamaremos Renlace, que puede no tener campos o tener solo uno que contenga un identificador único, el identificador lo proporcionará el sistema y no lo utiliza directamente el programa de aplicación, a este registro se le denomina también como registro ficticio o de enlace o unión.

Siguiendo los pasos anteriores nuestra estructura finalmente es: (Considerando una relación con cardinalidad Uno a Uno)

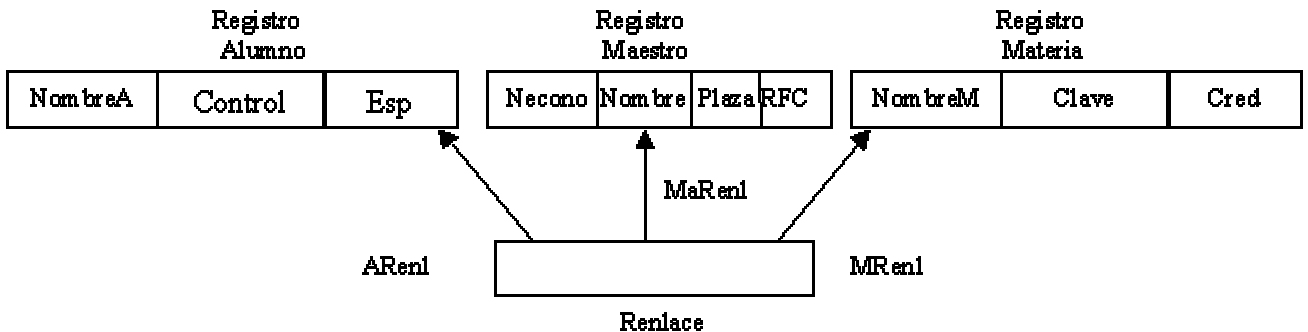


Diagrama de estructura de datos

Ahora si nuestro enlace tuviera atributos descriptivos, se crea el registro con los campos respectivos y se liga indicando el tipo de cardinalidad de que se trate.

En este caso tomamos el ejemplo anterior con cardinalidad uno a uno y le agregamos a la relación el atributo calif. (calificación).

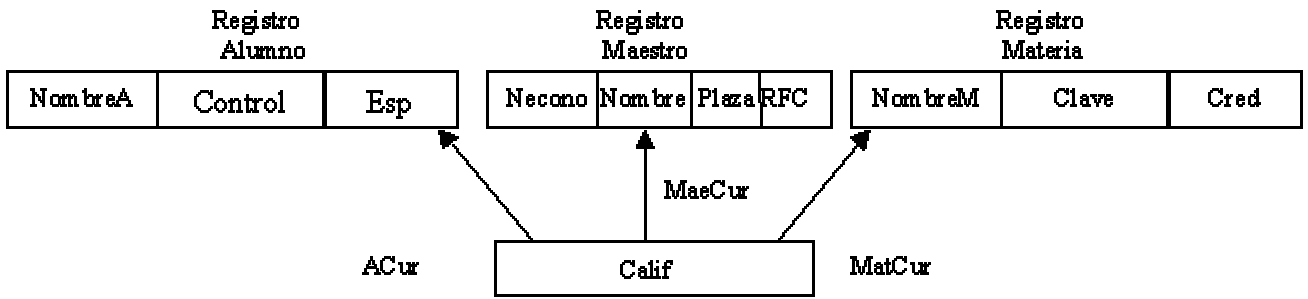
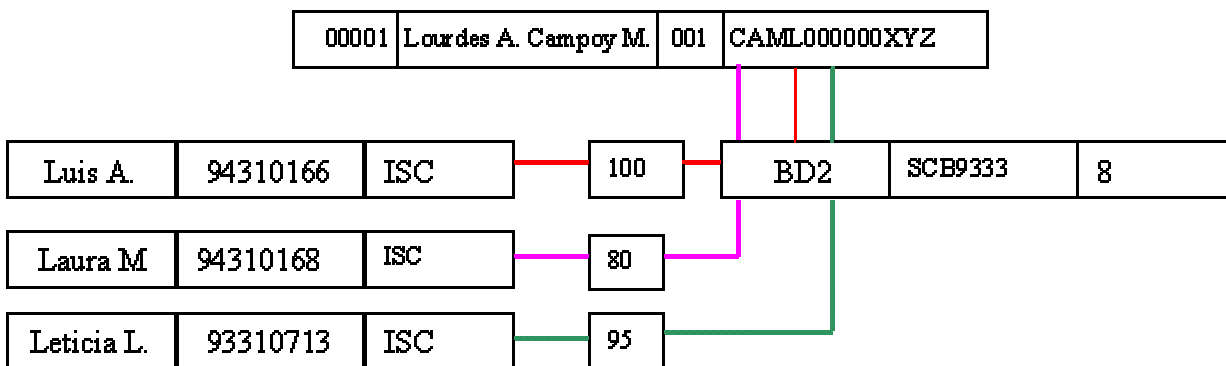


Diagrama de estructura de datos

Considerando el anterior diagrama de estructura de datos, una instancia de este sería:

La estructura quedaría:



Este diagrama nos indica que los alumnos Luis A. Laura M. y Leticia L. cursaron la materia Base de datos 2 con La maestra Ing. Lourdes A. Campoy M obteniendo una calificación de 100,80,95 respectivamente.

Este modelo fue desarrollado en 1971 por un grupo conocido como CODASYL: Conference on Data System Languages, Data Base Task Group, de ahí el nombre; este grupo es el que desarrolló los estándares para COBOL, el modelo CODASYL ha evolucionado durante los últimos años y existen diversos productos DBMS orientados a transacciones, sin embargo hoy día, estos productos están de salida, ya que este modelo es complejo y no cohesivo; los diseñadores y programadores deben de tener mucho cuidado al elaborar bases de datos y aplicaciones DBTG, además este modelo tiene mucho enfoque de COBOL, gran parte a las deficiencias detectadas en la actualidad se le atribuye a que este modelo fue desarrollado muy pronto antes de que se establecieran correctamente los conceptos esenciales de la tecnología de bases de datos.

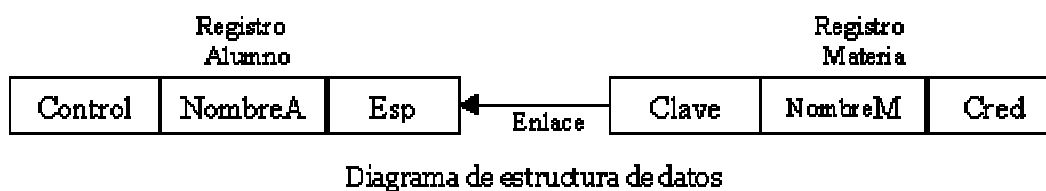
En esta unidad se estará aprendiendo sobre un modelo que fué base para el diseño de muchos productos DBMS orientados a transacciones.

5.3 El modelo CODASYL DBTG.

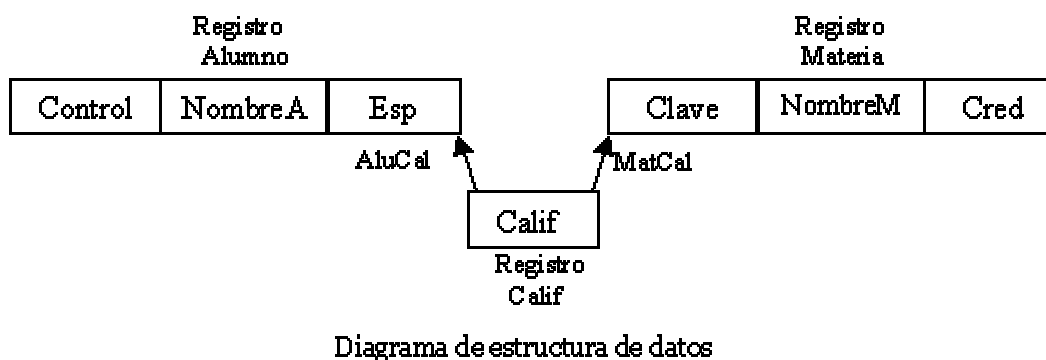
En el modelo DBTG solamente pueden emplearse enlaces uno a uno y uno a muchos. En este modelo existen dos elementos principales que son el dueño y el miembro, donde solo puede existir un dueño y varios miembros, donde cada miembro depende solamente de un dueño.

Empleando el ejemplo de la relación Alumno- cursa-Materia.

- Si la relación es uno a muchos sin atributos descriptivos, entonces el diagrama de estructura de datos apropiado es:



- Si la relación tiene un atributo descriptivo, como el de calif, entonces el diagrama de estructura de datos apropiado es:



- Si la relación fuera de muchos a muchos el algoritmo de transformación sería como el siguiente considerando que la relación no tiene atributos descriptivos, entonces:

1. Crear los registros correspondientes de las entidades involucradas (alumno,materia).
2. Crear un nuevo tipo de registro ficticio, enlace que puede no tener campos o tener sólo uno que contenga un identificador único definido externamente.
3. Crear los enlaces correspondientes muchos a uno.

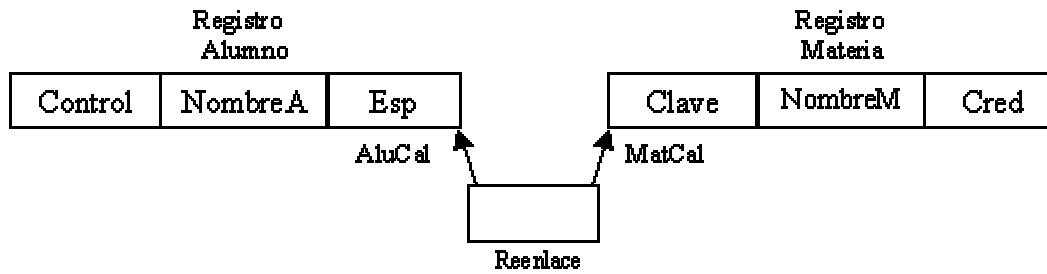


Diagrama de estructura de datos

En el caso de las relaciones generales (es decir, no binarias), el algoritmo de transformación es el mismo empleado para el estructuramiento de los diagramas de los modelos de red donde intervienen más de 2 entidades.

Por ejemplo consideremos la agregación de la entidad maestro, entonces para este caso resulta la estructura siguiente:

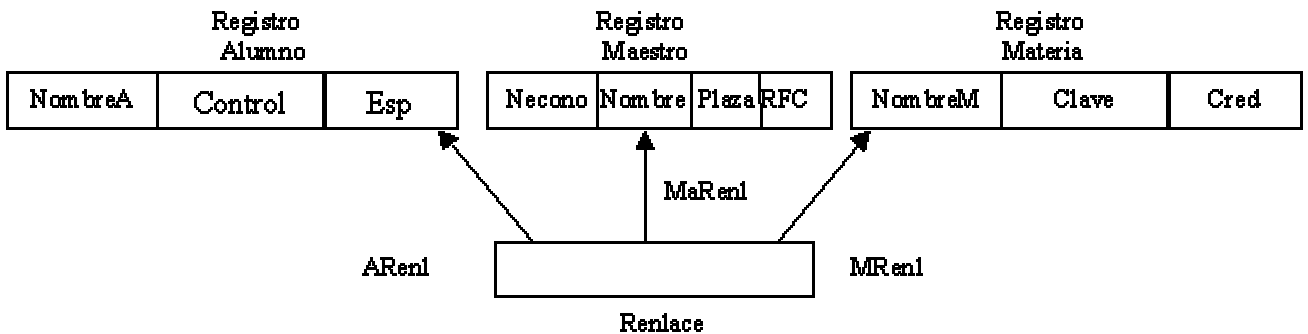
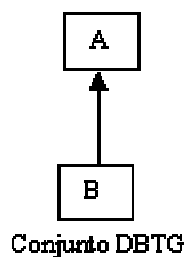


Diagrama de estructura de datos

Conjuntos DBTG

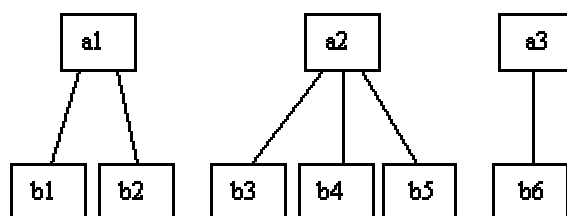
Como se mencionó anteriormente en este modelo solo pueden utilizarse enlaces muchos a uno y uno a uno, así una forma general de este modelo sería:



En el modelo DBTG, esta estructura se denomina *conjunto DBTG*. El nombre que se le asigna al conjunto generalmente es el mismo que el de la relación que une a las entidades.

En todo conjunto DBTG de este tipo, el tipo de registro A se denomina *dueño* (o padre) del conjunto, el tipo de registro B se le denomina *miembro* (o hijo) del conjunto. Cada conjunto DBTG puede tener cualquier número de ocurrencias del conjunto. Puesto que no se permiten enlaces del tipo muchos a muchos, cada ocurrencia del conjunto tiene exclusivamente un dueño y cero o más registros miembros. Además ningún registro puede participar en más de una ocurrencia del conjunto en ningún momento. Sin embargo, un registro miembro puede participar simultáneamente en varias ocurrencias de *diferentes* conjuntos.

Podemos ejemplificar las ocurrencias que se pueden presentar, como:



Tres ocurrencias de un conjunto

Para ilustrarlo, considérese el diagrama de estructura siguiente:

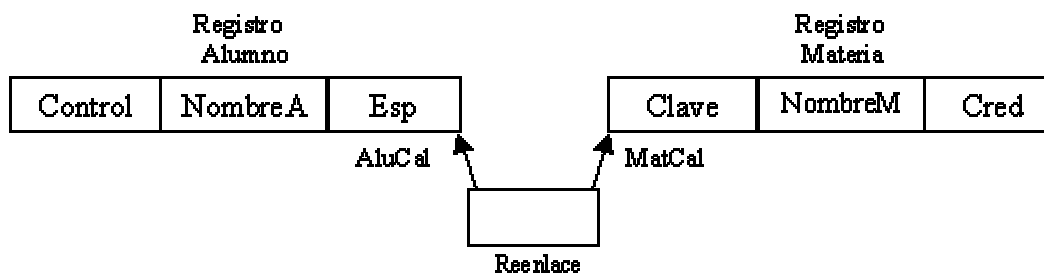


Diagrama de estructura de datos

Existen dos conjuntos DBTG:

- AluCal, cuyo *dueño* es Alumno y cuyo *miembro* es reenlace.
- MatCal, cuyo *dueño* es Materia y el *miembro* reenlace.

Declaración de conjuntos

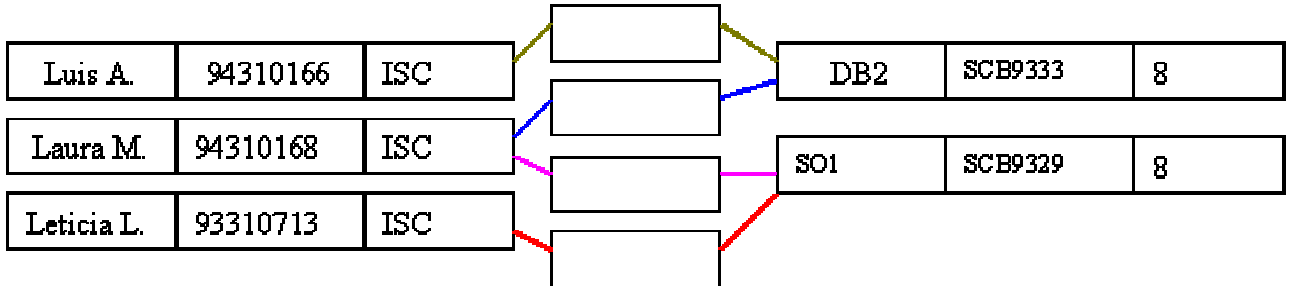
El conjunto *AluCal* se define: y el conjunto *MatCal*:



Set name is AluCal
 owner is Alumno
 member is reenlace

Set name is MatCal
 owner is Materia
 member is reenlace

Una instancia de la base de datos podría ser:



5.4 Recuperación de datos en DBTG

El lenguaje de manipulación de datos de la propuesta DBTG consiste en un número de órdenes que se incorporan en un lenguaje principal.

La mayor parte de los comandos de manejo de datos en DBTG de CODASYL tienen dos pasos. En primer lugar, se emite un comando FIND para identificar el registro sobre el cual se va a actuar. El comando FIND no lee, procesa el registro indicado; sólo identifica un registro para que lo localice el DBMS. Una vez identificado el registro, un segundo comando DML puede emitirse para que se lleve a cabo una operación sobre él. Patrones típicos son FIND, GET, o bien, FIND, MODIFY; o bien FIND, ERASE.

Cuando el programa emite un comando FIND, se localiza un registro y su identidad permanece almacenada en una variable especial llamada **indicador de posición**. Después, cuando se emite un comando GET, MODIFY, ERASE o bien otro, el DBMS hace referencia al indicador de posición para determinar cuál es el registro sobre el que ha de actuar. Los indicadores de posición también se utilizan como puntos de referencia para los comandos del procesamiento secuencial como son FIND NEXT o FIND PRIOR.

Las operaciones utilizadas para la recuperación de datos son :

- **Find:** Que localiza a un registro en la base de datos y da valores a los punteros de actualidad correspondientes

- **Get:** Copia el registro al que apunta el actual de unidad de ejecución de la base de datos a la plantilla adecuada en el área de trabajo de programa.

La orden FIND tiene varias formas, en este caso estudiaremos 2 ordenes find diferentes para localizar los registros individuales de la base de datos; La orden más sencilla es de la forma:

```
find any < tipo de registro > using < campo de registro >
```

Esta orden localiza un registro del tipo <tipo registro> cuyo valor de < campo de registro> es el mismo que el del valor de <campo de registro> en la plantilla del <tipo de registro> en el área de trabajo del programa. Una vez que se encuentra ese registro se modifican los siguientes punteros para que apunten a ese registro:

- El puntero actual de unidad de ejecución
- El puntero de actualidad de tipo de registro que corresponde a <tipo de registro>
- Para cada conjunto en el que participe ese registro, el puntero de actualidad de conjunto apropiado.

Para ilustrar lo anterior, construyamos una consulta en DBTG que nos muestre el número de control del alumno Luis A. (consideremos el modelo alumno-materia).

```
Alumno.nombre="Luis A.";
find any Alumno using NombreA;
get Alumno;
print("Alumno.control");
```

Pueden existir varios registros con el valor especificado. La orden **find** localiza el primero de ellos en un orden previamente especificado. Para localizar otros registros de la base de datos que pudieran tener el mismo campo <campo de registro, utilizamos la orden



find duplicate <tipo de registro> **using** <campo de registro>

Que localiza al siguiente registro (según un orden que depende del sistema) que tiene el valor de <campo registro>.ejemplo:

```
Alumno.NombreA:="Luis A.";
find any Alumno using NombreA;
while DB-Status = 0 do
  begin
    get Alumno;
    print (Alumno.control);
    find duplicate Alumno using NombreA;
  end;
```

Se ha encerrado parte de la consulta en un ciclo **while** ya que no sabemos por adelantado cuántos nombres Luis A. pueden existir. Salimos del bucle cuando DB-Status<>0, esto indica que la última operación **find duplicate** falló, lo que implica que ya hemos agotado todos los registros con esos datos.

5.5 Actualización en DBTG.

Creación de nuevos registros.

La orden para crear un nuevo registro es:

```
Store <tipo de registro>
```

Esta técnica nos permite crear y añadir solamente un registro a la vez.

Para ejemplificar consideremos el caso siguiente:

Deseamos registrar a un nuevo alumno de nombre "Delia J. Siordia", la instrucción para esto es:

```
Alumno.NombreA:="Delia J. Siordia";  
Alumno.control:="94310128";  
Alumno.Esp:="LAE";  
Store Alumno;
```

Modificación de un registro existente.

Para realizar la modificación a los registro primero se debe encontrar ese registro en la base de datos, pasarlo a la memoria principal y efectuar las modificaciones deseadas, posteriormente actualizamos el puntero de actualidad hacia el registro a modificar y ejecutamos la orden Modify.

```
Modify <tipo de registro>
```

Para que el sistema se entere que se va a realizar una modificación se emplea la orden **for update**.

Ejemplo: Deseamos cambiar la Especialidad del alumno de nombre Delia J. Siordia, por LI.

```
Alumno.NombreA:="Delia J. Siordia";  
find for update any Alumno using nombreA;  
get Alumno.Esp :="LI";  
modify Alumno;
```

Las líneas de código antes descritas, primero requieren del dato por el cual se realizará la búsqueda del registro, la orden **find for update any** establecen el tipo de registro que se usara, **using** indica el campo por el cual debe buscar la coincidencia para la modificación, **get** se antepone la línea de modificaciones que se realizaran sobre el registro y finalmente ya que el registro fue modificado se ejecuta la instrucción **modify** para activar las actualizaciones ejecutadas.

Eliminación de registros.

5.6 Procesamiento de conjuntos en DBTG.

* La sentencia de conexión (connect)

Esta sentencia permite la inserción de un registro en un conjunto.

La sintaxis es:

Connect <tipo de registro> **tp** <tipo de conjunto>

La inserción de un registro nuevo se realiza creando el nuevo registro buscar el dueño del conjunto en donde el puntero de actualidad apuntara a el lugar adecuado donde debe insertarse el nuevo registro y posteriormente insertamos el registro ejecutando la sentencia Connect.

Ejemplo: Considérese la consulta DBTG para el modelo alumno-materia para crear el registro de un nuevo alumno en la materia BD1.

```
Alumno.Control:=97310128;
Alumno.NombreA:=Armando Sánchez;
Alumno.Esp:=ISC;
Store Alumno;
Materia.Clave:=SCB9333;
find any Materia using Clave;
connect Alumno to AIren1; (*Conectar el registro al reenlace *);
```

* Sentencia de desconexión Disconnect

Esta sentencia se utiliza para "desconectar" a un registro de un conjunto, es necesario que los punteros de actualidad tanto del registro como del conjunto y del registro apunten a los elementos adecuados. posteriormente se elimina el registro deseado aplicando la sentencia Disconnect.

Sintaxis:

Disconnect <Tipo registro> **From** <Tipo de conjunto>

Esta operación solamente desconecta al registro no lo elimina de la base de datos para eliminarlo totalmente se utiliza la orden Erase.

Sintaxis:

Erase <registro>

* Sentencia de reconexión reconnect.

La función permite mover un registro de un conjunto hacia otro conjunto, para ello se requiere encontrar el registro y el dueño apropiado.

Sintaxis:

reconnect <registro> to <Conjunto>

* **Inserción y retención en conjuntos.**

Cuando se define un nuevo conjunto, debemos especificar cómo se van a insertar los registros miembros. Además de especificarse las condiciones bajo las cuales debe retenerse a un registro.

* **Inserción en conjuntos.**

Un registro recién creado, del tipo registro de un tipo de conjunto puede añadirse a una ocurrencia de un conjunto manual o automáticamente. El modo se especifica al definir el conjunto mediante la orden:

Insertion is <modo de inserción>

donde modo de inserción puede tomar los valores Manual y Automática mediante las siguientes ordenes: **connect** <tipo registro> **tp** <tipo conjunto> y **store** <Tipo registro> respectivamente. En ambos casos el puntero de actualidad del conjunto debe apuntar al conjunto sobre el cual se va a realizar la inserción.

* **Retención en conjuntos.**

Para que se logre la retención de miembros en un conjunto se realiza a través de una serie de restricciones que se especifican en el momento de definir el conjunto.

Sintaxis:

retention is <modo de retención>

Donde modo de retención puede ser de tres formas:

- Fija: Una vez insertado un registro miembro en un conjunto no podrá sacarse del conjunto. En caso de querer cambiar dicho registro primero se tendrá que eliminar el registro y volver a crearlo e insertarlo en el conjunto deseado.
- Obligatoria: Permite la reconexión de registros miembros solo en conjuntos del mismo tipo.
- Opcional: No tiene limitaciones.



Modelo de datos de Jerárquico

6.1 Conceptos Básicos.

Una base de datos jerárquica consiste en una colección de registros que se conectan entre sí por medio de enlaces. Los registros son similares a los expuestos en el modelo de red. Cada registro es una colección de campos (atributos), que contienen un solo valor cada uno de ellos. Un enlace es una asociación o unión entre dos registros exclusivamente. Por tanto, este concepto es similar al de enlace para modelos de red.

Consideremos la base de datos, nuevamente, que contiene la relación alumno - materia de un sistema escolar. Existen dos tipos de registros en este sistema, alumno y materia. El registro alumno consta de tres campos: NombreA, Control y Esp; El registro Materia esta compuesto de tres campos: Clave, NombreM y Cred.

En este tipo de modelos la organización se establece en forma de árbol, donde la raíz es un nodo ficticio. Así tenemos que, una base de datos jerárquica es una colección de árboles de este tipo.

El contenido de un registro específico puede repetirse en varios sitios(en el mismo árbol o en varios árboles).

La repetición de los registros tiene dos desventajas principales:

- * Puede producirse una inconsistencia de datos
- * El desperdicio de espacio.

6.2 Diagramas de estructura de árbol

Un diagrama de estructura de árbol es la representación de un esquema de la base de datos jerárquica, de ahí el nombre, ya que un árbol está desarrollado precisamente en orden descendente formando una estructura jerárquica.

Este tipo de diagrama está formado por dos componentes básicos:

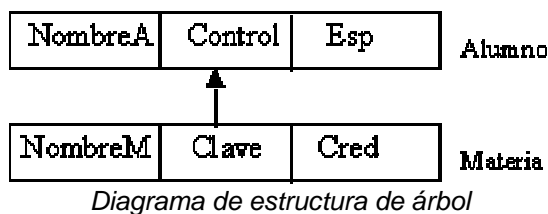
- Rectángulos: que representan a los de registros.
- Líneas: que representan a los enlaces o ligas entre los registros.

Un diagrama de árbol tiene el propósito de especificar la estructura global de la base de datos.

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

Características de las estructuras de árbol:

- El árbol no puede contener ciclos.
- Las relaciones que existen en la estructura deben ser de tal forma que solo existan relaciones muchos a uno o uno a uno entre un padre y un hijo.

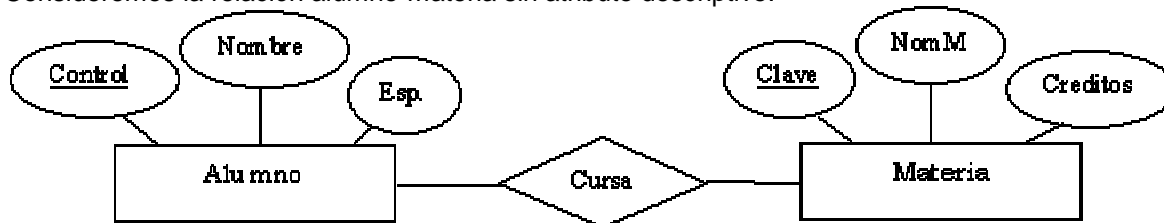


En este diagrama podemos observar que las flechas están apuntando de padres a hijos. Un padre (origen de una rama) puede tener una flecha apuntando a un hijo, pero un hijo siempre puede tener una flecha apuntando a su padre.

El esquema de una base de datos se representa como una colección de diagramas de estructura de árbol. Para cada diagrama existe una única instancia de árbol de base de datos. La raíz de este árbol es un nodo ficticio. Los hijos de ese nodo son instancias de los registros de la base de datos. Cada una de las instancias que son hijos pueden tener a su vez, varias instancias de varios registros.

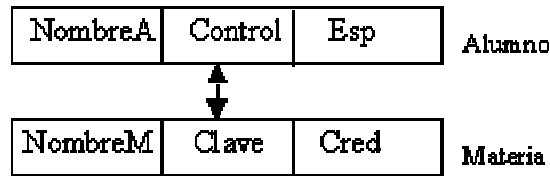
Las representaciones según las cardinalidades son:

Consideremos la relación alumno-materia sin atributo descriptivo.

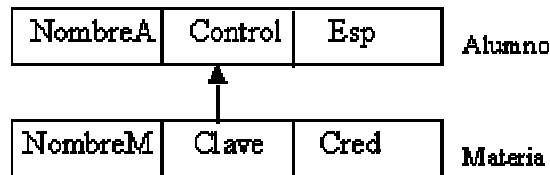


La transformación según las cardinalidades sería:

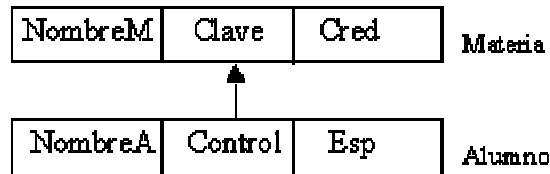
- Cuando la relación es uno a uno.



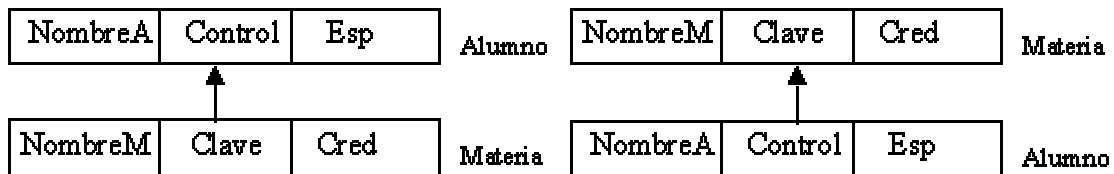
- Cuando la relación es uno a muchos.



- Cuando la relación es muchos a uno.



- Cuando la relación es muchos a muchos.

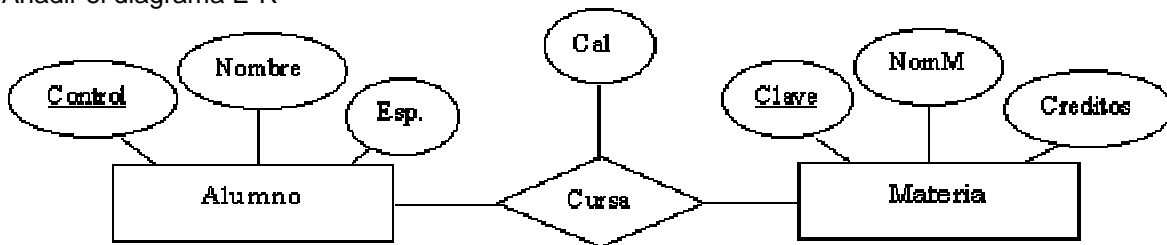


Cuando la relación tiene atributos descriptivos, la transformación de un diagrama E-R a estructura de árbol se lleva a cabo cubriendo los siguientes pasos:

1. Crear un nuevo tipo de registro.
2. Crear los enlaces correspondientes.

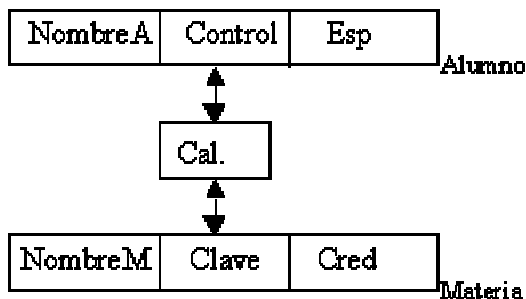
Consideremos que a la relación Alumno-Materia añadimos el atributo Cal a la relación que existe entre ambas, entonces nuestro modelo E-R resulta:

Añadir el diagrama E-R

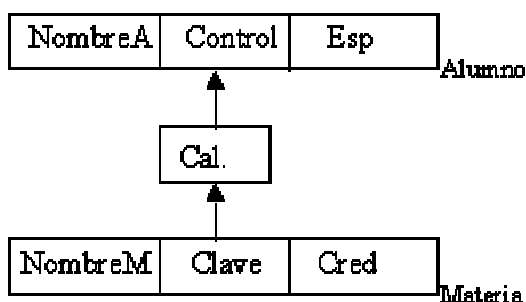


Según las cardinalidades los diagramas de estructura de árbol pueden quedar de la siguiente manera:

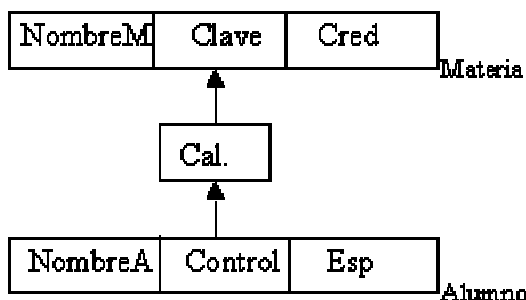
- Cuando la relación es uno a uno.



- Cuando la relación es uno a muchos.



- Cuando la relación es Muchos a uno.



- Cuando la relación es Muchos a Muchos.

Si la relación es muchos a muchos entonces la transformación a diagramas de árbol es un poco más compleja debido a que el modelo jerárquico solo se pueden representar las relaciones uno a uno o uno a muchos.

Existen varias formas distintas de transformar este tipo de relaciones a estructura de árbol, sin embargo todas las formas constituyen la repetición de algunos registros.

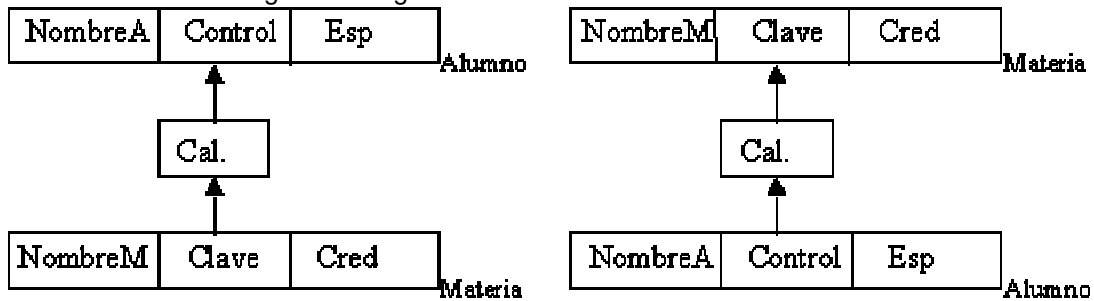
La decisión de qué método de transformación debe utilizarse depende de muchos factores, entre los que se incluyen:

- El tipo de consultas esperadas en la base de datos.
- El grado al que el esquema global de base de datos que se está modelando se ajusta al diagrama E-R dado.

A continuación se describe la forma de transformar un diagrama E-R a estructura de árbol con relaciones muchos a muchos. Suponemos el ejemplo de la relación alumno-materia.

1. Crear dos diagramas de estructura de árbol distintos T1 y T2, cada uno de los cuales incluye los tipos de registro alumno y materia, en el árbol T1 la raíz es alumno y en T2 la raíz es materia.
2. Crear los siguientes enlaces:
 - Un enlace muchos a uno del registro cuenta al registro Alumno, en T1
 - Un enlace muchos a uno del tipo de registro cliente al tipo de registro materia en T2.

Como se muestra en el siguiente diagrama:





6.3 Recuperación de datos

Para manipular la información de una base de datos jerárquico, es necesario emplear un lenguaje de manipulación de datos, el lenguaje consta de varias órdenes que están incorporadas en un lenguaje principal, Pascal.

La orden Get

La recuperación de datos se realiza mediante esta orden, se realizan las siguientes acciones:

- Localiza un registro en la base de datos y actualiza a un puntero que es el que mantiene la dirección del último registro accesado.
- Copia los datos solicitados a un tipo de registro apropiado para la consulta.

La orden Get debe especificar en cual de los árboles de la base de datos se va a buscar.

Para revisar todos los registros de forma consistente, se debe imponer un orden en los registros. El que normalmente se usa es el preorden, el cuál consiste en iniciar la búsqueda por la raíz y continua buscando por los subárboles de izquierda a derecha recursivamente. Así pues, empezamos por la raíz, visitamos el hijo más a la izquierda, y así sucesivamente, hasta que alcancemos un nodo hoja (sin hijos). A continuación movemos hacia atrás al padre de la hoja y visitamos el hijo más a la izquierda no visitado. Continuamos de esta forma hasta que se visite todo el árbol.

Existen dos ordenes Get diferentes para localizar registros en un árbol de base de datos. La orden más simple tiene la forma:

```
get first <Tipo de registro>  
where <Condición>
```

La cláusula where es opcional. La <Condición> que se adjunta es un predicado que puede implicar a cualquier tipo de registro que sea un antecesor de <tipo registro> o el <Tipo de registro> mismo.

La orden get localiza el primer registro (en preorden) del tipo <Tipo registro> en la base de datos que satisfaga la <condición > de la cláusula where. Si se omite la cláusula where, entonces se localiza el primer registro del tipo <Tipo registro>. Una vez que se encuentra ese registro, se hace que el puntero que tiene la dirección del último registro accesado apunte a ese registro y se copia el contenido del registro en un registro apto para la consulta.

Para ilustrar lo anterior veamos los **ejemplos**:

Realicemos una consulta que imprima El nombre del alumno llamado Luis A. (consideremos la relación Alumno-Materia que hemos estado manejando.)

```
get first Alumno  
where Alumno.NombreA="Luis A.";  
print (Alumno.Control)
```

Ahora consideremos que deseamos la consulta de los nombres de las materias en donde el alumno de nombre Luis A. A obtenido una calificación igual a 100 (si es que existe)

```
get first Alumno  
where Alumno.NombreA="Luis A." and Cursa.cal= 100;  
if DB-Status=0 then print (Materia.NombreM);
```




La condición involucra a la variable DB-Status, la cual nos indica si se encontró o no el registro.

La orden **get first**, solo nos muestra el primer registro encontrado que satisfaga la orden de consulta, sin embargo puede haber más de ellos, para localizar a los demás registros empleamos la orden **Get next**.

Cuya estructura es:

```
get next <Tipo de registro>  
where <Condición>
```

La cual localiza el siguiente registro en preorden que satisface la condición. Si se omite la cláusula **where**, entonces se localiza el siguiente registro del tipo <Tipo registro>.



6.4 Actualización de datos.

Creación de nuevos registros.

La orden utilizada para la inserción de registros es:

```
insert <tipo de registro>  
where <Condición>
```

donde: tipo registro contiene los datos de los campos del registro a insertar.

Si se incluye la cláusula *where*, el sistema busca en el árbol de la base de datos (en preorden) un registro que satisfaga la condición dada, una vez encontrado, el registro creado se inserta en el árbol como un hijo más a la izquierda. Si se omite la cláusula *where*, el registro nuevo es insertado en la primera posición (en preorden) en el árbol de la base de datos donde se pueda insertar un registro del mismo tipo que el nuevo.

Ejemplos:

Consideremos que queremos añadir una nueva alumna cuyo nombre es Delia Siordia con número de control 99310168 de la carrera de LI; entonces la inserción del nuevo registro sería de la siguiente manera:

```
Alumno.NombreA:="Delia Siordia";  
Alumno.Control:="99310168";  
Alumno.Esp:="ISC";  
insert Alumno;
```

Consideremos que deseamos crear la alta de la materia de matemáticas 1 a la alumna con número de control 99310168.

```
Materia.NombreM:="Matemáticas 1";  
Materia.Clave:="SCB9334";  
Materia.Cred:=8;  
insert Materia;  
where Alumno.Control="99310168";
```

Modificación de registros existentes.

La instrucción para efectuar cambios a los registros es:

Replace

Esta instrucción no requiere los datos del registro a modificar como argumento, el registro que se afectará será aquel al que este apuntando el puntero de actualidad, que debe ser el registro que se desea modificar.

Ejemplo:



6.5 Registros virtuales.

Como se describió anteriormente, en las relaciones muchos a muchos se requería la repetición de datos para conservar la organización de la estructura del árbol de la base de datos.

La repetición de la información genera 2 grandes problemas:

- La actualización puede generar inconsistencia de los datos.
- Se genera un desperdicio considerable de espacio.

Para solventar estos problemas se introdujo el concepto de registro virtual, el cual no contiene datos almacenados, si no un puntero lógico a un registro físico determinado.

Cuando se va a repetir un registro en varios árboles de la base de datos, se mantiene una sola copia de ese registro en uno de los árboles y empleamos en los otros registros la utilización de un registro virtual que contiene la dirección del registro físico original.



Bases de datos orientadas por objetos.

Las aplicaciones de las bases de datos en áreas como el diseño asistido por computadora, la ingeniería de software y el procesamiento de documentos no se ajustan al conjunto de suposiciones que se hacen para aplicaciones del estilo de procesamiento de datos. El modelo de datos orientado por objetos se ha propuesto para tratar algunos de estos nuevos tipos de aplicaciones.

El modelo de bases de datos orientado por objetos es una adaptación a los sistemas de bases de datos. Se basa en el concepto de encapsulamiento de datos y código que opera sobre estos en un objeto. Los objetos estructurados se agrupan en clases. El conjunto de clases está estructurado en sub y superclases basado en una extensión del concepto ISA del modelo Entidad - Relación. Puesto que el valor de un dato en un objeto también es un objeto, es posible representar el contenido del objeto dando como resultado un objeto compuesto.

El propósito de los sistemas de bases de datos es la gestión de grandes cantidades de información. Las primeras bases de datos surgieron del desarrollo de los sistemas de gestión de archivos. Estos sistemas primero evolucionaron en bases de datos de red o en bases de datos jerárquicas y, más tarde, en bases de datos relacionales.

Estructura de objetos.

El modelo orientado por objetos se basa en encapsular código y datos en una única unidad, llamada objeto. El interfaz entre un objeto y el resto del sistema se define mediante un conjunto de mensajes.

Un objeto tiene asociado:

- un conjunto de variables que contienen los datos del objeto. El valor de cada variable es un objeto.
- Un conjunto de mensajes a los que el objeto responde.
- Un método, que es un trozo de código para implementar cada mensaje. Un método devuelve un valor como respuesta al mensaje.

El término mensaje en un contexto orientado por objetos, no implica el uso de un mensaje físico en una red de computadoras, si no que se refiere al paso de solicitudes entre objetos sin tener en cuenta detalles específicos de implementación.

La capacidad de modificar la definición de un objeto sin afectar al resto del sistema está considerada como una de las mayores ventajas del modelo de programación orientado a objetos.

Jerarquía de clases.

En una base de datos existen objetos que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. Sería inútil definir cada uno de estos objetos por separado por lo tanto se agrupan los objetos similares para que formen una clase, a cada uno de estos objetos se le llama instancia de su clase. Todos los objetos de su clase comparten una definición común, aunque difieran en los valores asignados a las variables.

Así que básicamente las bases de datos orientadas por objetos tienen la finalidad de agrupar aquellos elementos que sean semejantes en las entidades para formar un clase, dejando por separado aquellas que no lo son en otra clase.

