

**Materia: Base de Datos**

**Técnico Superior en Ciencia de Datos.**

Base de Datos

Contenido

Introducción 2

Sistemas de Gestión de Bases de Datos (DBMS) 2

Concepto 2

Propósito de los DBMS 3

Funciones de los DBMS 5

Tecnología de Base de Datos vs. Tecnología Tradicional 5

Clasificación de Bases de Datos 7

Modelo de la relación de entidad (E-R) 10

Caso de Estudio 10

Entidades 10

Identificación de Entidades 11

Atributos 11

Clasificación de Atributos 12

Reglas de Integridad 14

Relaciones 14

Cardinalidad y Opcionalidad de las relaciones 14

Tipos de cardinalidad y opcionalidad 14

Lenguaje de Consulta SQL 19

Creación de una base de datos 19

Definición de tablas y campos 19

Normalización de las tablas 19

Utilidades 19

Tipos de datos que se pueden almacenar 19

Propiedades de los campos 19

Introducción de datos 19

Ordenamiento de datos 19

Filtros 19

Definición de la clave primaria y otros índices 19

Definición de relaciones entre tablas 19

Definición de reglas de integridad. 19

# Introducción

Durante las últimas cuatro décadas del siglo veinte, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaban directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaban indirectamente con bases de datos (con informes impresos como los extractos de las tarjetas de crédito, o mediante agentes como los cajeros de los bancos y los agentes de reservas de las líneas aéreas). Después llegaron los cajeros automáticos y permitieron a los usuarios interactuar directamente con las bases de datos. Pero a partir de la revolución de Internet a finales de los años noventa aumentó significativamente el acceso directo del usuario a las bases de datos.

**Una base de datos** (DB, en inglés Data Base) **es una colección de datos relacionados**. Con la palabra datos nos referimos a los hechos conocidos que se pueden grabar y que tienen un significado implícito. Por ejemplo, los nombres, números de teléfono y direcciones de las personas que conoce. Puede tener todos estos datos grabados en un libro de direcciones indexado o los puede tener almacenados en el disco duro de un computador. Esta colección de datos con un significado implícito es una base de datos.

Una base de datos tiene las siguientes propiedades implícitas:

* Una base de datos representa algún **aspecto del mundo real**, lo que en ocasiones se denomina mini-mundo o universo de discurso. Los cambios introducidos en el mini-mundo se reflejan en la base de datos.
* Una base de datos es una **colección de datos lógicamente coherente** con algún tipo de significado inherente. No es correcto denominar base de datos a un surtido aleatorio de datos.
* Una base de datos se diseña, construye y rellena con datos para un **propósito específico**. Dispone de un grupo pretendido de usuarios y algunas aplicaciones preconcebidas en las que esos usuarios están interesados.

En otras palabras, una base de datos tiene algún origen del que se derivan los datos, algún grado de interacción con eventos del mundo real y un público que está activamente interesado en su contenido.

Con el objetivo de que una base de datos sea en todo momento precisa y fiable, debe ser un reflejo exacto del mini-mundo que representa; por consiguiente, en la base de datos deben reflejarse los cambios tan pronto como sea posible.

Una base de datos puede ser de **cualquier tamaño y complejidad**.

Una base de datos se puede generar y mantener manualmente o estar automatizada.

# Sistemas de Gestión de Bases de Datos (DBMS)

## Concepto

Una base de datos, por sí misma, no es más que una forma de almacenamiento de datos; por lo tanto, para poder acceder a dichos datos existe un conjunto de programas que facilitan la interacción con dichos datos.

El objetivo principal de este conjunto de programas denominado: Sistema de Administración de Base de Datos (DBMS, en inglés DataBase Management System) es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea práctica y eficiente.

Estos sistemas de administración de DB están diseñados para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de administración de bases de datos deben garantizar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o de los intentos de acceso no autorizados. Si los datos van a ser compartidos entre diferentes usuarios, el sistema debe evitar posibles resultados anómalos.

Podemos definir un sistema de administración de datos como **una colección de programas que permite a los usuarios crear y mantener una base de datos**. El DBMS es un sistema de software de propósito general que facilita los procesos de definición, construcción, manipulación y compartición de bases de datos entre varios usuarios y aplicaciones. Definir una base de datos implica especificar los tipos de datos, estructuras y restricciones de los datos que se almacenarán en la misma.

## Propósito de los DBMS

Las bases de datos y los sistemas de administración de bases de datos surgieron en respuesta a los primeros métodos de gestión informatizada de los datos comerciales. A modo de ejemplo de dichos métodos, típicos de los años sesenta, considérese parte de una entidad bancaria que, entre otros datos, guarda información sobre todos los clientes y todas las cuentas de ahorro. Una manera de guardar la información en la computadora es almacenarla en archivos del sistema operativo. Para permitir que los usuarios manipulen la información, el sistema tiene varios programas de aplicación que gestionan los archivos, incluyendo programas para:

* Efectuar cargos o abonos en las cuentas.
* Añadir cuentas nuevas.
* Calcular el saldo de las cuentas.
* Generar los extractos mensuales.

Estos programas de aplicación los han escrito programadores de sistemas en respuesta a las necesidades del banco.

Se añaden nuevos módulos al sistema según surgen las necesidades. Por ejemplo, supóngase que un banco decide ofrecer además de caja de ahorro, cuentas corrientes. En consecuencia, se crean nuevos archivos permanentes que contienen información acerca de todas las cuentas corrientes abiertas en el banco y puede que haya que escribir nuevos programas de aplicación para afrontar situaciones que no se dan en las cajas de ahorro, como los descubiertos. Así, con el paso del tiempo, se añaden más archivos y programas de aplicación al sistema.

Los sistemas operativos convencionales soportan este sistema de procesamiento de archivos típico. El sistema almacena los registros permanentes en varios archivos y necesita diferentes programas de aplicación para extraer y añadir a los archivos correspondientes. Antes de la aparición de las bases de datos, las organizaciones normalmente almacenaban la información en sistemas de este tipo.

Guardar la información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

* **Redundancia e inconsistencia de los datos**. Debido a que los archivos y programas de aplicación los crean diferentes programadores en el transcurso de un período de tiempo, es probable que los diversos archivos tengan estructuras diferentes y que los programas estén escritos en varios lenguajes de programación diferentes. Además, puede ser que la información esté duplicada en varios lugares (archivos). Por ejemplo, la dirección y el número de teléfono de un cliente dado pueden aparecer en un archivo que contenga registros de cuentas de ahorros y en un archivo que contenga registros de cuentas corrientes. Esta redundancia conduce a costos de almacenamiento y de acceso más elevados. Además, puede dar lugar a la inconsistencia de los datos; es decir, puede ser que las diferentes copias de los mismos datos no coincidan. Por ejemplo, es posible que el cambio en la dirección de un cliente esté reflejado en los registros de las cuentas de ahorro, pero no en el resto del sistema.
* **Dificultad en el acceso a los datos.** Supóngase que uno de los empleados del banco necesita averiguar los nombres de todos los clientes que viven en domicilio con un mismo código postal. El empleado pide al departamento de procesamiento de datos que genere esa lista. Debido a que esta petición no fue prevista por los diseñadores del sistema original, no hay un programa de aplicación a mano para satisfacerla. Hay, sin embargo, un programa de aplicación que genera la lista de todos los clientes. El empleado del banco tiene ahora dos opciones: obtener la lista de todos los clientes y extraer manualmente la información que necesita, o bien pedir a un programador de sistemas que escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias. Supóngase que se escribe el programa y que, varios días más tarde, el mismo empleado necesita reducir esa lista para que incluya únicamente a aquellos clientes que tengan una cuenta con saldo igual o superior a $10.000. Como se puede esperar, no existe ningún programa que genere tal lista. De nuevo, el empleado tiene que elegir entre dos opciones, ninguna de las cuales es satisfactoria. La cuestión aquí es que los entornos de procesamiento de archivos convencionales no permiten recuperar los datos necesarios de una forma práctica y eficiente. Hacen falta sistemas de recuperación de datos más adecuados para el uso general.
* **Aislamiento de datos.** Como los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos correspondientes.
* **Problemas de integridad.** Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de restricciones de consistencia. Por ejemplo, el saldo de ciertos tipos de cuentas bancarias no puede nunca ser inferior a una cantidad predeterminada (por ejemplo, $25). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código correspondiente en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema se complica cuando las restricciones implican diferentes elementos de datos de diferentes archivos.
* **Problemas de atomicidad.** Los sistemas informáticos, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallos. En muchas aplicaciones es crucial asegurar que, si se produce algún fallo, los datos se restauren al estado consistente que existía antes del fallo. Considérese un programa para transferir $10.000 desde una cuenta A a una Cuenta B. Si se produce un fallo del sistema durante la ejecución del programa, es posible que los $10.000 fueran retirados de la cuenta A, pero no acreditados en la cuenta B, dando lugar a un estado inconsistente de la base de datos. Evidentemente, resulta esencial para la consistencia de la base de datos que tengan lugar tanto el débito (descuento del saldo de la cuenta A) como el crédito (cargo en la Cuenta B), o que no tenga lugar ninguno. Es decir, la transferencia de fondos debe ser atómica—debe ocurrir en su totalidad o no ocurrir en absoluto. Resulta difícil asegurar la atomicidad en los sistemas convencionales de procesamiento de archivos.
* **Anomalías en el acceso concurrente.** Para aumentar el rendimiento global del sistema y obtener una respuesta más rápida, muchos sistemas permiten que varios usuarios actualicen los datos simultáneamente. En realidad, hoy en día, los principales sitios de comercio electrónico de Internet pueden tener millones de accesos diarios de compradores a sus datos. En tales entornos es posible la interacción de actualizaciones concurrentes y puede dar lugar a datos inconsistentes. Considérese una cuenta bancaria A, que contenga $5000. Si dos clientes retiran fondos (por ejemplo, $500 y $1000, respectivamente) de la cuenta A aproximadamente al mismo tiempo, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto (o inconsistente). Supóngase que los programas que se ejecutan para cada retiro de dinero leen el saldo anterior, reducen su valor en el importe que se retira y luego escriben el resultado. Si los dos programas se ejecutan concurrentemente, pueden leer el valor $500, y escribir después $4500 y $4000, respectivamente. Dependiendo de cuál escriba el valor en último lugar, la cuenta puede contener $4500 o $4000, en lugar del valor correcto, $3500. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Pero es difícil ofrecer supervisión, ya que muchos programas de aplicación diferentes que no se han coordinado con anterioridad pueden tener acceso a los datos.
* **Problemas de seguridad.** No todos los usuarios de un sistema de bases de datos deben poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas (Empleados de la empresa), sólo necesita ver la parte de la base de datos que contiene información acerca de los diferentes empleados del banco. No necesitan tener acceso a la información acerca de las cuentas de clientes. Pero, como los programas de aplicación se añaden al sistema de procesamiento de datos a medida que se requieren, es difícil hacer cumplir tales restricciones de seguridad.

Estas dificultades, entre otras, motivaron el desarrollo de las bases de datos y los sistemas de administración de las mismas.

## Funciones de los DBMS

* **Metadatos**: es la definición o información descriptiva de una base de datos, relacionada con su estructura y los datos que contiene, también se almacena, en forma de catálogo o diccionario de la base de datos.
* **Construcción de la base de datos**: es el proceso consistente en almacenar los datos en algún medio de almacenamiento controlado por el DBMS.
* **Manipulación de una base de datos**: son aquellas funciones como la consulta de la base de datos para recuperar datos específicos, actualizar la base de datos para reflejar los cambios introducidos en el mini-mundo y generar informes a partir de los datos.
* **Compartir una base de datos**: permite que varios usuarios y programas accedan a la base de datos de forma simultánea.
* **Consultas**: una aplicación accede a la base de datos enviando consultas o solicitudes de datos al DBMS. Una consulta normalmente provoca la recuperación de algunos datos.
* **Transacciones**: una transacción puede provocar la lectura o la escritura de algunos datos en la base de datos.
* **Protección**: incluye la protección del sistema contra el funcionamiento defectuoso del hardware o el software y la protección de la seguridad contra el acceso no autorizado o malintencionado.
* **Mantenimiento**: una gran base de datos típica puede tener un ciclo de vida de muchos años, por lo que el DBMS debe ser capaz de mantener el sistema de bases de datos permitiendo que el sistema evolucione según cambian los requisitos con el tiempo.

## Tecnología de Base de Datos vs. Tecnología Tradicional

Unas cuantas características distinguen la tecnología de bases de datos de la tecnología tradicional de programación con archivos.

|  |  |
| --- | --- |
| Procesamiento tradicional de archivos | Base de Datos |
| * Cada usuario define e implementa los archivos necesarios para una aplicación concreta como parte de la programación de esa aplicación. * Cada aplicación tiene libertad para asignar un nombre independientemente a los elementos de datos. | * Se mantiene un único almacén de datos, que se define una sola vez, y al que acceden varios usuarios. * Los nombres o etiquetas de los datos se definen una vez, y son utilizados para consultas, transacciones y aplicaciones. |

Las principales características de las bases son las siguientes:

##### Naturaleza auto descriptiva de un sistema de bases de datos.

Una característica fundamental de la tecnología de bases de datos es que el sistema de bases de datos no sólo contiene la propia base de datos, sino también una completa definición o descripción de la estructura de la base de datos y sus restricciones. Esta definición se almacena en el catálogo DBMS, que contiene información como la estructura de cada archivo, el tipo y el formato de almacenamiento de cada elemento de datos, y distintas restricciones de los datos. La información almacenada en el catálogo se denomina, como vimos anteriormente: metadatos y describe la estructura de la base de datos principal. El software DBMS y los usuarios de la base de datos utilizan el catálogo cuando necesitan información sobre la estructura de la base de datos. Un paquete de software DBMS de propósito general no se escribe para una aplicación de base de datos específica. Por consiguiente, debe referirse al catálogo para conocer la estructura de los archivos de una base de datos específica, como el tipo y el formato de los datos a los que accederá. El software DBMS debe funcionar igual de bien con cualquier cantidad de aplicaciones de bases de datos (por ejemplo, la base de datos de una universidad, la base de datos de un banco o la base de datos de una empresa), siempre y cuando la definición de la base de datos esté almacenada en el catálogo.

##### Aislamiento entre programas y datos, y abstracción de datos.

En el procesamiento de archivos tradicional, la estructura de los archivos de datos está incrustada en las aplicaciones, por lo que los cambios que se introducen en la estructura de un archivo pueden obligar a realizar cambios en todos los programas que acceden a ese archivo. Por el contrario, los programas que acceden a un DBMS no necesitan esos cambios en la mayoría de los casos. La estructura de los archivos de datos se almacena en el catálogo DBMS, independientemente de los programas de acceso. Llamaremos a esta propiedad independencia programa-datos.

##### Soporte de varias vistas de los datos.

Normalmente una base de datos tiene muchos usuarios, cada uno de los cuales puede necesitar una perspectiva o vista diferente de la base de datos. Una vista puede ser un subconjunto de la base de datos o puede contener datos virtuales derivados de los archivos de la base de datos pero que no están explícitamente almacenados.

Algunos usuarios no tienen la necesidad de preocuparse por si los datos a los que se refieren están almacenados o son derivados. Un DBMS multiusuario cuyos usuarios tienen variedad de diferentes aplicaciones debe ofrecer facilidades para definir varias vistas.

##### Compartición de datos y procesamiento de transacciones multiusuario.

Un DBMS multiusuario, como su nombre indica, debe permitir que varios usuarios puedan acceder a la base de datos al mismo tiempo. Esto es esencial, si los datos destinados a varias aplicaciones serán integrados y mantenidos en una sola base de datos. El DBMS debe incluir software de control de la concurrencia para que esos varios usuarios que intentan actualizar los mismos datos, en el mismo momento, lo hagan de un modo controlado para que el resultado de la actualización sea correcto. Por ejemplo, si varios agentes de viajes intentan reservar un asiento en un vuelo, el DBMS debe garantizar que en cada momento sólo un agente tiene acceso a la asignación de ese asiento para un pasajero. Estos tipos de aplicaciones se denominan, por lo general, aplicaciones de procesamiento de transacciones en línea (OLTP, online transaction processing). Un papel fundamental del software DBMS multiusuario es garantizar que las transacciones concurrentes operan correcta y eficazmente.

El concepto de transacción es cada vez más importante para las aplicaciones de bases de datos. Una transacción es un programa en ejecución o proceso que incluye uno o más accesos a la base de datos, como la lectura o la actualización de los registros de la misma. Se supone que una transacción ejecuta un acceso lógicamente correcto a la base de datos si lo ejecutó íntegramente sin interferencia de otras transacciones. El DBMS debe implementar varias propiedades de transacción. La propiedad aislamiento garantiza que parezca que cada transacción se ejecuta de forma aislada de otras transacciones, aunque puedan estar ejecutándose cientos de transacciones al mismo tiempo. La propiedad de atomicidad garantiza que se ejecuten o todas o ninguna de las operaciones de bases de datos de una transacción.

## Clasificación de Bases de Datos

En la actualidad, las empresas manejan una gran cantidad de datos que debe tener almacenados en una base de datos para poder consultarlos o modificarlos mediante un producto de software o sistema. Sin esta funcionalidad, resultaría imposible administrar en su totalidad los datos que usa la empresa y se perdería mucho tiempo y dinero.

Uno de los pasos cruciales en la construcción de una aplicación que maneje una base de datos es el diseño de la base de datos, en donde lo más importante es la correcta elección del Modelo de Datos.

Son muchas las consideraciones a tener en cuenta al momento de hacer el diseño de la base de datos, entre las que se pueden mencionar:

* La velocidad de acceso
* El tamaño de la base de datos
* El tipo de los datos
* Facilidad de acceso a los datos
* Facilidad para extraer los datos requeridos

Un modelo de datos es un conjunto de conceptos utilizados para organizar los datos de interés y describir su estructura en forma comprensible para un sistema informático.

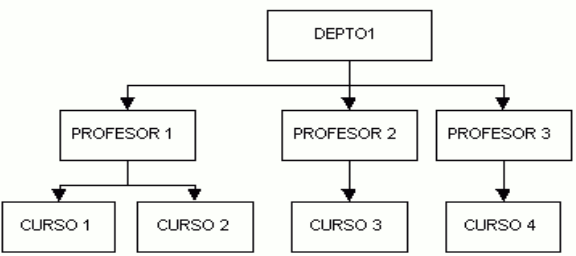
El diseño de bases de datos es el proceso por el que se determina la organización de una base de datos, incluidos su estructura, contenido y las aplicaciones que se han de desarrollar.

Un modelo de datos es entonces una serie de conceptos que puede utilizarse para describir un conjunto de datos y las operaciones para manipularlos. Hay dos tipos de modelos de datos: los modelos conceptuales y los modelos lógicos. Los **modelos conceptuales** se utilizan para representar la realidad a un alto nivel de abstracción. Mediante los modelos conceptuales se puede construir una descripción de la realidad fácil de entender. En los **modelos lógicos**, las descripciones de los datos tienen una correspondencia con la estructura física de la base de datos. En el diseño de bases de datos se usan primero los modelos conceptuales para lograr una descripción de alto nivel de la realidad, y luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad.

Coincidiendo con la evolución histórica de las bases de datos éstas han utilizado distintos modelos:

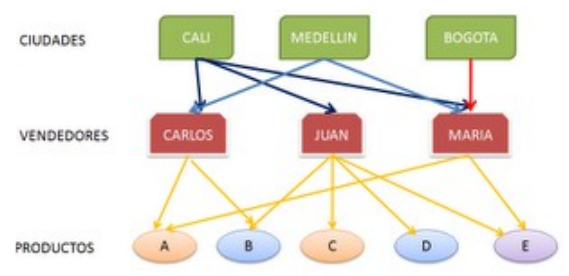
##### Modelo Jerárquico

En el modelo jerárquico los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de datos puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas.



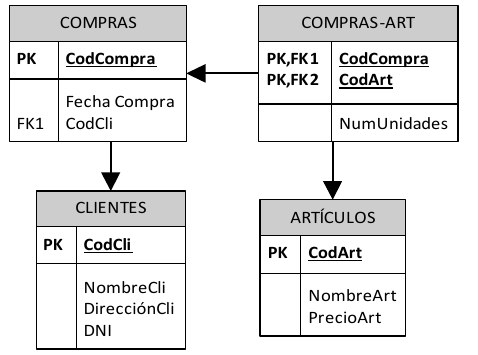
##### Modelo de red

Este modelo es ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres.



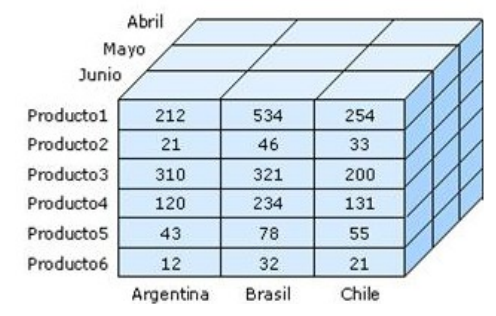
##### Modelo Relacional

Es el modelo más utilizado en la actualidad. Almacena los datos en filas o registros (tuplas) y columnas o campos (atributos). Estas tablas pueden estar conectadas entre sí por claves comunes.



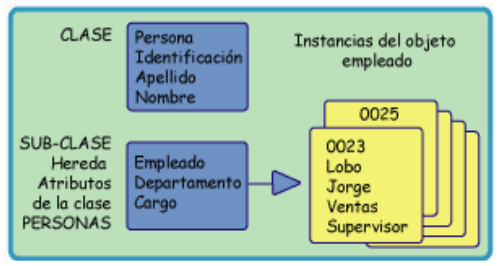
##### Modelo Multidimensional

La estructura multidimensional tiene parecidos a la del modelo relacional, pero en vez de las dos dimensiones filas-columnas, tiene N dimensiones. Las bases de datos multidimensionales suelen representarse gráficamente como cubos.



##### Modelo Orientado a Objetos

La estructura orientada a objetos está diseñada siguiendo el paradigma de los lenguajes orientados a objetos. De este modo soporta los tipos de datos gráficos, imágenes, voz y texto de manera natural. Esta estructura tiene gran difusión en aplicaciones web para aplicaciones multimedia.



##### Modelo Orientado a Documentos

Este modelo permite la indexación a texto completo, es decir que posibilita crear estructuras de datos similares a los árboles de búsqueda para, en líneas generales, realizar búsquedas más potentes.



# Modelo de la relación de entidad (E-R)

## Caso de Estudio

Antes de comenzar a desarrollar el tema de modelo de Entidad Relación, vamos a presentar el caso de estudio “Sistema de Gestión de Ventas y Reservas de Entradas para un Complejo de Cines”

* Un complejo de cines está integrado por varios cines ubicados principalmente en los centros comerciales de la ciudad.
* Cada cine cuenta con una cantidad de salas, que son las que exhiben las películas en las distintas funciones cinematográficas.
* La programación de las salas se renueva en forma semanal, existiendo la posibilidad de que algunas salas queden sin uso. Cabe mencionar que no todas las salas tienen la misma capacidad (cantidad de butacas).
* La programación es la que determina qué películas van a proyectarse y los horarios para cada función de cada una de las salas, para todos los cines. Esta programación se realiza en forma centralizada, desde la administración del Complejo, tomándose como base la información de las películas próximas a estrenar, que envía el INCAA (Instituto Nacional de Cines y Artes Audiovisuales). La programación implica el diseño de las funciones y sus horarios en forma anticipada, debiendo el responsable de la misma, habilitar cada función en el momento que desee permitir la reserva y/o venta de entradas para la misma.
* La que se le entrega al cliente representa el comprobante de venta y como tal debe cumplir con lo reglamentado en la Ley de Facturación vigente, debiendo contener como datos: nro. de venta, fecha de venta, número de función, sala en la que se proyecta la película, el nombre de la película, fecha y hora de la función, el precio, el tipo de entrada (si es mayor, menor, jubilado) y la calificación de la película, que según especificaciones de la Ley de Cine Nro. 17.741, debe ser informada tanto en la entrada como al inicio de la película. Es importante destacar que la entrada es válida únicamente para la fecha, hora y función indicadas en la misma.
* Los tipos de entradas y los días y horarios de proyección son los que determinan el precio de la entrada, que también pueden variar en cada cine del complejo. Las funciones admiten ciertos tipos de entradas y otros no, dependiendo de factores como: horarios, calificación de las películas, etc. Por ejemplo: si una película está calificada como para mayores de 16 años, para esa función no se pueden vender entradas de TIPO = MENOR. Cada función tiene asociado un tipo de función, que determina si la función es un pre-estreno, un estreno, una función normal.

## Entidades

El elemento básico representado por el modelo entidad relación es una entidad, que es una cosa del mundo real con una existencia independiente.

* Una entidad puede ser un elemento con una existencia física (por ejemplo, una persona en particular, un coche, una casa o un empleado) o puede ser un elemento con una existencia conceptual (por ejemplo, una venta, un trabajo o un curso universitario).
* Cada entidad tiene atributos (propiedades particulares que la describen).

Una entidad es una abstracción de un conjunto de cosas del mundo real tal que:

* Las cosas de ese conjunto tienen las mismas características o comportamiento.
* Las cosas de ese conjunto están sujetas y conformes a las mismas reglas.

Las entidades que se tendrán en cuenta al modelar un sistema son aquellas que representan "cosas" de las que el sistema necesita almacenar ciertos datos.

Ejemplo:

El conjunto: {ruleman 8705, tornillo 5 mm, filtro de aire, etc.}

forma la entidad => REPUESTO

## Identificación de Entidades

Para identificar entidades vemos cuál es el sistema que se está analizando (identificando sus límites) y nos preguntamos:

¿Cuáles son las cosas de este sistema de las que nos interesa tener sus datos?

Para facilitar la identificación de las entidades, debemos tener en cuenta que existen cuatro categorías, de acuerdo a lo que ellas representen:

* **Cosas tangibles**: Artículo, Repuesto, Rodado.
* **Roles desempeñados por personas u organizaciones**: Cliente, Proveedor, Personal.
* **Incidentes**: Usado para representar la ocurrencia de un hecho (en un sistema de una compañía de seguros: Siniestros; en una empresa de transporte: Viajes).
* **Interacciones**: Representan alguna transacción (Compra, Pedido, Venta, Pago).

Es importante una buena elección del nombre dado a una entidad para la legibilidad y el entendimiento del modelo de datos. Luego de identificar las entidades, se analizan sus atributos.

## Atributos

Un atributo es una abstracción que identifica características, propiedades que posee una entidad. Los atributos de una entidad deben ser:

* Completos: capturar toda la información que interesa del objeto, desde el punto de vista del sistema.
* Plenamente elaborados: cada atributo captura un aspecto separado de la entidad.
* Mutuamente independientes: cada atributo debe tomar un valor independientemente de los valores asumidos por otros atributos.

Con respecto a nuestro caso de estudio, podemos identificar la entidad principal PELICULA y sus atributos, de la siguiente manera:

* Entidad: PELICULA
* Atributos: nombre, titulo original, año de estreno, disponible, duración, fecha de ingreso.

Podemos ver esta entidad representada en un diagrama de entidad-relación de la siguiente manera:

**Pelicula**

anio\_estreno

disponible

duracion

fecha\_ingreso

nombre

titulo\_original

Así podemos tener como ejemplo de entidades PELICULA, a dos instancias de película, con los siguientes atributos:

Película 1: {2013, true, 143, 11/07/2013, "El Gran Gatsby", "The Great Gatsby"}

Película 2: {2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes"}

### Clasificación de Atributos

##### Atributos descriptivos

Son las características intrínsecas de cada instancia de la entidad; como lo dice su nombre, describen a la entidad, representan sus propiedades. En la entidad película sería los atributos: anio\_estreno, disponible

duración, fecha\_ingreso, nombre y titulo\_original.

##### Atributo Identificador Único

Se denomina identificador a uno o más atributos que identifican unívocamente cada instancia de una entidad; es conocido también como "clave candidata". Es decir, nunca puede existir dos instancias de una entidad con el mismo valor de su atributo identificador. En una entidad puede haber varios atributos posibles para ser elegidos como "identificador".

Por ejemplo, en el caso de una entidad EMPLEADO podríamos tener dos atributos que cumplen con la condición de identificar unívocamente a cada instancia de la entidad: documento y CUIT.

En estos casos, para elegir el atributo identificador debemos tener en cuenta dos reglas:

* Que la clave sea mínima: Es decir elegir la alternativa en la que se necesiten menos atributos para conformar la clave.
* Elegir el atributo más significativo dentro del dominio del problema que se está modelando.

En el caso de ejemplo mencionado anteriormente, en la entidad Empleado se debe elegir un atributo para que sea identificador, en este caso conviene elegir el CUIT ya que ocasionalmente pueden existir dos personas con el mismo DNI, pero nunca con el mismo CUIT.

Una definición más formal podría ser la siguiente:

“El atributo A, o el conjunto de atributos, de una entidad, es un posible atributo identificador si y solo si satisface dos propiedades:

* Unicidad: en cualquier momento dado no existen 2 instancias con el mismo valor de A.
* Minimidad: Si A es compuesto (es decir el atributo identificador está formado por más de un atributo) no será posible eliminar ningún componente de A sin destruir la propiedad de unicidad.

Toda entidad tiene por lo menos un atributo como posible atributo identificador. El o los atributos identificadores se señalan con el símbolo "@"(arroba), o de lo contrario con la sigla PK (clave primaria).

Para mejorar el desempeño de la base de datos se recomienda utilizar identificadores numéricos; por lo tanto, si una entidad no posee un atributo identificador numérico, se debería agregar un atributo, comúnmente llamado id (abreviación de identificador) seguido por el nombre de la entidad, como se muestra en la entidad Película, donde podemos observar que si bien el nombre de la película no debe repetirse en el negocio, no es un atributo numérico; como consecuencia es más conveniente agregar uno, de la siguiente manera:

**Pelicula**

@id\_pelicula

anio\_estreno

disponible

duracion

fecha\_ingreso

nombre

titulo\_original

Película 1: {1, 2013, true, 143, 11/07/2013, "El Gran Gatsby", "The Great Gatsby"}

Película 2: {2, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes"}

Este atributo agregado id\_película, no es más que un número identificador que crece secuencialmente a medida que se agregan nuevas películas: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,..n.

##### Atributo Referencial

Un atributo referencial se utiliza para poder establecer relaciones entre diferentes entidades de un modelo entidad- relación. Se denominan REFERENCIALES ya que hacen referencia al ATRIBUTO IDENTIFICADOR de la entidad con que se relacionan. Se dice que un atributo j, ó un conjunto de atributos, de una entidad B es un atributo referencial si y solo si satisface dos propiedades:

* Cada valor j es nulo del todo o no nulo del todo. En caso de ser un atributo compuesto, formado por más de un atributo.
* Existe una entidad A con atributo identificador j tal que:
  + Cada valor no nulo de j es en la entidad B idéntico al valor j en alguna instancia de la entidad A. Es decir que si en B el atributo j tiene valor es porque existe ese mismo valor de j en la entidad A.

Podemos ver un ejemplo relacionado con el caso de estudio del Complejo de Cines, con las entidades: PELICULA y PAIS DE ORIGEN. Se utiliza el símbolo “# ” (numeral) para señalar que un atributo es referencial, o de lo contrario con la sigla FK (clave foránea).

Esto se logra de la siguiente manera:

**PaisDeOrigen**

@id\_pais\_origen

idioma

nombre

**Pelicula**

@id\_pelicula

anio\_estreno

disponible

duracion

fecha\_ingreso

nombre

titulo\_original

#id\_pais\_origen

Como podemos ver, la entidad Película se relaciona con la entidad PaisDeOrigen, donde país de origen tiene como atributo identificador, señalado con @, que es: id\_pais\_de\_origen.

La entidad Película tiene un atributo referencial, señalado con #, denominado: id\_pais\_de\_origen, que para cada instancia de película referenciará a un único país de origen.

Por ejemplo:

Si tuviéramos las siguientes entidades en PaisDeOrigen:

País de Origen 1: {1, “Español”, Argentina}

País de Origen 1: {2, “Inglés”, Estados Unidos}

País de Origen 1: {3, “Francés”, Francia}

Entonces la entidad película, tendría en el atributo referencial id\_pais\_de\_origen, el valor 1, que referencia a la instancia Argentina de la entidad PaisDeOrigen, como se ve a continuación:

Película 2: { 1, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes",1}

## Reglas de Integridad

Existen dos reglas denominadas “Reglas de Integridad” que son generales y se aplican a todo el modelo de datos relacional y que son:

1. Integridad de Entidades: Ningún componente del atributo identificador en una entidad aceptará NULOS (nulo se considera que es inexistente, es decir, ausencia de valor).

Ejemplo:

**X** Película: {**NULL**, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes”,1}

**√** Película: {**2**, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes",1}

1. Integridad Referencial: Un modelo de datos no debe contener valores en sus atributos referenciales para los cuales no exista un valor concordante en el (ó los) atributos identificadores en la entidad objetivo pertinente.

Ejemplo:

**X** Película: {2, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes",**4**}

No existe en la tabla PaisDeOrigen una instancia con id\_pais\_de\_origen = 4.

**√** Película: {**2**, 2014, true, 122, 01/08/2014, "Relatos Salvajes", "Relatos Salvajes",**1**}

## Relaciones

* Una relación es la abstracción de un conjunto de asociaciones que existen entre las instancias de dos entidades, por ejemplo, existe una relación entre Película y PaisDeOrigen (Ver figura 14).
* Las relaciones tienen sentido bidireccional.
* Las relaciones existen ya que las entidades representan aspectos del mundo real y en este mundo los componentes no están aislados, sino que se relacionan entre sí; es por esto que es necesario que existan las relaciones entre las entidades.

## Cardinalidad y Opcionalidad de las relaciones

**Cardinalidad: Indica para una instancia de una entidad A con cuántas instancias de la entidad B, se relaciona. Las posibilidades son: 0, 1 o muchos, que se representan con O ; | respectivamente**

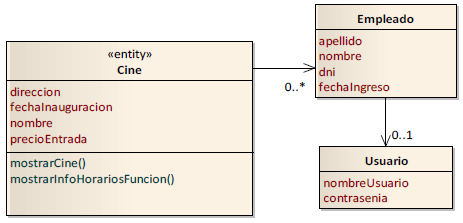
**Opcionalidad**: Indica para una instancia de una entidad A, si la relación con instancias de la entidad B, es opcional u obligatoria. Las posibilidades son: 0 o 1 que se representan con O o |, 🡨; respectivamente.

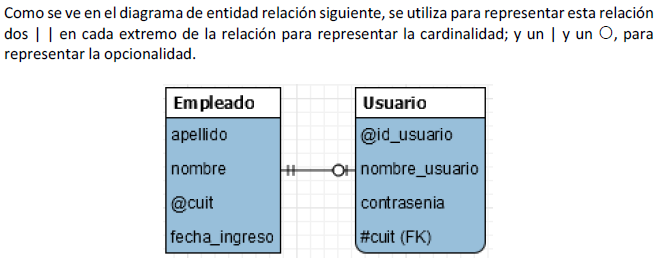
### Tipos de cardinalidad y opcionalidad

##### Relación Uno a uno

Por ejemplo, en el caso de estudio añadiremos las clases que se muestran a continuación para explicar esta relación:

Entonces, si llevamos estas clases a entidades, vemos que: un empleado puede tener o no un usuario y si ese usuario existe, es para un único empleado.



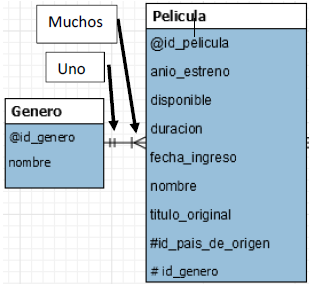


Esto se lee, de la siguiente manera:

**“Un empleado puede tener opcionalmente un único usuario o no tener ningún usuario asociado, un usuario está asociado de manera obligatoria a un único empleado”.**

##### Relación Uno a muchos

Por ejemplo, una película tiene un único género, pero un género como “Drama” puede estar asignado a muchas películas:

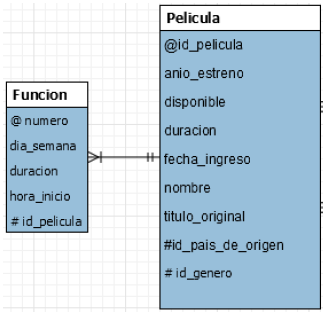


Esto se lee, de la siguiente manera:

**“Un género puede estar asignado a muchas películas, pero una película tiene un único género”**

##### Muchos a Uno

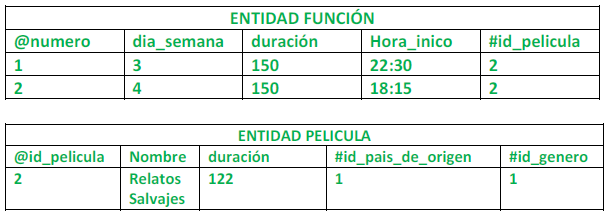
Por ejemplo, una película tiene muchas funciones asignadas, pero una función es para una única película.



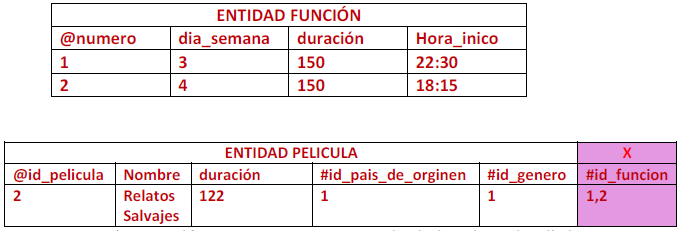
**ACLARACIÓN**: Para los casos anteriores (uno a muchos o muchos a uno) siempre el atributo referencial se debe colocar en la entidad donde está la cardinalidad MUCHOS. Esto se justifica debido a que los atributos deben tener valores atómicos en cada celda, es decir un único valor.

A continuación, se muestran 2 ejemplos para explicar lo mencionado anteriormente:

Caso 1: Relación entre Función y Película, con Función referenciando a Película: en este caso la función tiene una única película asociada porque se proyecta una película por función:



Caso 2: Relación entre Función y Película, con Película referenciando a Función: en este caso como la película puede proyectarse en muchas funciones, deberíamos poner en la columna identificada como #id-funcion, más de un atributo referencias, rompiendo la propiedad de atomicidad de los atributos:

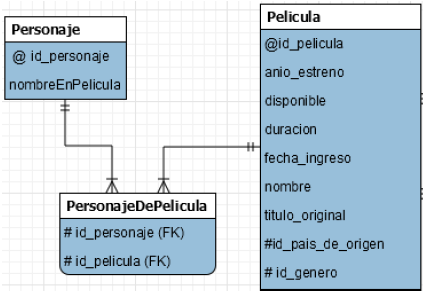


##### Muchos a muchos

Por ejemplo, un personaje puede pertenecer a varias películas, por ejemplo: Harry Potter y la piedra filosofal, Harry Potter y la cámara secreta, Harry Potter y el prisionero de Azkaban… y una película puede tener varios personajes.

En este caso, en donde ambas entidades tienen la cardinalidad muchos, nos encontramos con la limitación de no poder dónde colocar el atributo referencial en ninguna entidad, sin romper la unicidad.

Cuando la relación es de muchos a muchos, la forma de solucionarlo y respetar con la propiedad de unicidad es por medio de la creación de una entidad intermedia, denominada Entidad Asociativa, que relacione un personaje con una película de la siguiente manera:



Como se puede ver en la figura 19, la tabla intermedia creada, PersonajeDePelicula, se encarga de establecer una relación uno a uno, entre un personaje y una película; de esta manera siguiendo con el ejemplo de Harry Potter, tendríamos:

Entidades de Personajes:

{1,“Harry Potter”}

{2,“ Ron Weasley”},

{3,“ Hermione Granger”}

Instancias de la Entidad Pelicula:

{5, 2001, true, 152, 03/03/2001, “Harry Potter y la piedra filosofal”, “Harry Potter and the

Philosopher's Stone”, 1, 3}

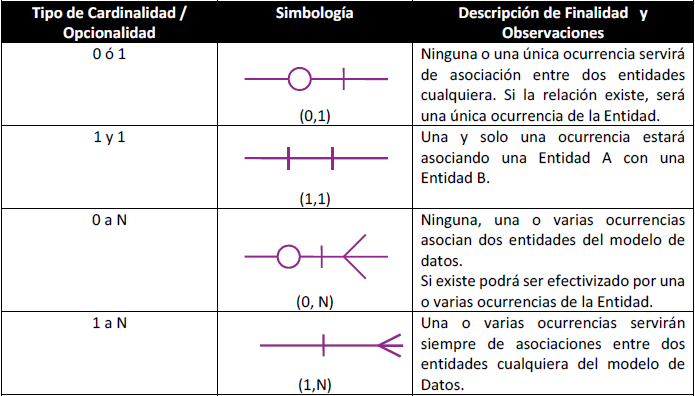
{6, 2002, true, 152, 03/06/2002, “Harry Potter y la cámara secreta”, “Harry Potter and the

Chamber of Secrets”, 1, 3}

Instancias de la Entidad PersonajeDePelicula:

{1,5},{ 2,5},{3,5},{1,6},{ 2,6},{3,6}

La siguiente tabla presenta un resumen de la Simbología usualmente adoptada en la construcción de Diagramas de Entidad - Relación para representar relaciones entre entidades:



# Lenguaje de Consulta SQL

El nombre SQL significa Lenguaje de consulta estructurado (Structured Query Language). SQL fue establecido como el lenguaje de alto nivel estándar para sistemas de base de datos relacionales. Los responsables de publicar este lenguaje como estándar fueron ANSI (Instituto Americano de Normalización) e ISO (Organismo Internacional de Normalización).

SQL utiliza los términos tabla (relación), fila (tupla) y columna (atributo).

SQL es agrupa 3 tipos de sentencias con objetivos particulares, en los siguientes lenguajes:

* Lenguaje de definición de datos: DDL (Data Definition Language).
* Lenguaje de manipulación de datos: DML (Data Manipulation Language).
* Lenguaje de control de datos: DCL (Data Control Language).

En esta sección nos centraremos en:

* DDL: grupo de sentencias SQL que soportan la definición de los objetos de la base de datos. Cuando nos referimos a objetos tales como: database, table, view, index, procedure, trigger, rule, domain y valores por defecto (default). Utiliza las siguientes instrucciones:
  + CREATE
  + DROP
  + ALTER
* DML: grupo se sentencias SQL para manipular los datos que están almacenados en la base de datos, a nivel de filas (tupla) y/o columnas (atributos). Ya sea que requiera que los datos sean modificados, eliminados, consultados o que se agreguen nuevas filas a las tablas. Utiliza las siguientes instrucciones:
  + SELECT: Consulta registros de la base de datos que satisfagan un criterio determinado.
  + INSERT: Carga lotes de datos en la base de datos en una única operación.
  + UPDATE: Modifica los valores de los campos y registros especificados.
  + DELETE: Elimina registros de una tabla de una base de datos.
* DCL: grupo de sentencias de SQL para controlar las funcionas de administración que realiza el DBMS, tales como la atomicidad y seguridad. Utiliza las siguientes instrucciones:
  + COMMIT TRANSACTION
  + ROLLBACK TRANSACTION
  + GRANT
  + REVOKE

## Tipos básicos de datos

La norma SQL soporta gran variedad de tipos de dominio predefinidos, entre ellos:

* char(n). Una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario. También se puede utilizar la palabra completa character.
* varchar(n). Una cadena de caracteres de longitud variable con una longitud máxima n especificada por el usuario. La forma completa, character varying, es equivalente.
* int. Un entero (un subconjunto finito de los enteros dependiente de la máquina). La palabra completa, integer, es equivalente.
* smallint. Un entero pequeño (un subconjunto dependiente de la máquina del tipo de dominio
* entero).
* numeric(p; d). Un número de coma fija, cuya precisión la especifica el usuario. El número está formado por p dígitos (más el signo), y de esos p dígitos, d pertenecen a la parte decimal. Así, numeric( 3,1) permite que el número 44:5 se almacene exactamente, pero ni 444:5 ni 0:32 se pueden almacenar exactamente en un campo de este tipo.
* real, double precision. Números de coma flotante y números de coma flotante de doble precisión, con precisión dependiente de la máquina.
* float(n). Un número de coma flotante cuya precisión es, al menos, de n dígitos.
* date. Una fecha de calendario que contiene el año (de cuatro cifras), el mes y el día del mes.
* time. La hora del día, en horas, minutos y segundos. Se puede usar una variante, time(p), para especificar el número de cifras decimales para los segundos (el valor predeterminado es 0). También es posible almacenar la información del huso horario junto a la hora especificando time with timezone.
* timestamp. Una combinación de date y time. Se puede usar una variante, timestamp(p), para especificar el número de cifras decimales para los segundos (el valor predeterminado es seis). También se almacena información sobre el huso horario si se especifica with timezone.

Los valores de fecha y hora se pueden especificar de esta manera:

date ’2001-04-25’

time ’09:30:00’

timestamp ’2001-04-25 10:29:01.45’

La fecha se debe especificar en el formato de año seguido del mes y del día, tal y como se muestra. El campo segundos de time y timestamp puede tener parte decimal, como puede verse en el ejemplo anterior.

Se pueden usar expresiones de la forma cast e as t para convertir una cadena de caracteres (o una expresión de tipo cadena de caracteres) e al tipo t, donde t es de tipo date, time o timestamp. La cadena de caracteres debe tener el formato adecuado, como se indicó al comienzo de este párrafo. Si es necesaria, la información sobre el huso horario puede deducirse de la configuración del sistema.

Para extraer campos concretos de un valor d de date o de time se puede utilizar extract (campo from d), donde campo puede ser year, month, day, hour, minute o second. La información sobre el huso horario puede obtenerse mediante timezone\_hour y timezone\_minute.

SQL también define varias funciones útiles para obtener la fecha y la hora actuales. Por ejemplo, current\_date devuelve la fecha actual, current\_time devuelve la hora actual (con su huso horario) y localtime devuelve la hora local actual (sin huso horario). Las marcas de tiempo (fecha y hora) se obtienen con current\_timestamp (con huso horario) y localtimestamp (fecha y hora locales sin huso horario).

SQL permite realizar operaciones de comparación sobre todos los tipos de datos que se han mencionado aquí, así como operaciones aritméticas y de comparación sobre los diferentes tipos de datos numéricos. SQL también proporciona un tipo de datos denominado interval y permite realizar cálculos basados en fechas, horas e intervalos. Por ejemplo, si x e y son del tipo date, entonces x ¡ y es un intervalo cuyo valor es el número de días desde la fecha x hasta la y. De forma análoga, al sumar o restar un intervalo a una fecha o a una hora se obtiene como resultado otra fecha u hora, respectivamente.

A menudo resulta útil comparar valores de diferentes tipos de datos compatibles. Por ejemplo, supóngase que el tipo de datos de nombre\_cliente es una cadena de caracteres de longitud 20 y el tipo de datos de nombre\_sucursal es una cadena de caracteres de longitud 15. Aunque la longitud de las cadenas sea diferente, la norma SQL considera que los dos tipos de datos son compatibles. Como ejemplo adicional, como cada entero perteneciente al tipo smallint es también un entero, las comparaciones x < y, donde x es de tipo smallint e y es de tipo int (o viceversa), son válidas. Este tipo de comparación se lleva a cabo transformando primero el número x al tipo int. Las transformaciones de este tipo se denominan coerción. La coerción de tipos se emplea de manera habitual en los lenguajes de programación comunes, así como en los sistemas de bases de datos.

## Definición de Datos

#### CREATE DATABASE

CREATE DATABASE [IF NOT EXIST] database\_name

* Esta sentencia sirve para crear una base de datos con un nombre específico.
* Para poder crear una base de datos, el usuario que la crea debe tener privilegios de creación asignados.
* IF NOT EXISTS significa: SI NO EXISTE, por lo tanto, esto es útil para validar que la base de datos sea creada en caso de que no exista, si la base de datos existe y se ejecuta esta sentencia, se genera error.

#### CREATE TABLE

CREATE TABLE [IF NOT EXIST] table\_name

column\_name <data\_type> [ NULL | NOT NULL ]

[ DEFAULT default\_value ] 🡪 [ CONSTRAINT constraint\_name [ DEFAULT constant\_expression ] ]

[ AUTO\_INCREMENT] 🡪 [ IDENTITY [ (seed,increment) ]

[ UNIQUE | PRIMARY KEY ]

[ CONSTRAINT [nombre\_relación] FOREIGN KEY (nombre\_columna) REFERENCES

nombre\_de\_tabla (nombre\_columna)]

[ ON DELETE opciones\_de\_referencia]

[ ON UPDATE opciones\_de\_referencia]

***opciones\_de\_referencia:***

RESTRICT | CASCADE | SET NULL | NO ACTION

en el manual de SQL es

[ CONSTRAINT constraint\_name ]

{ { PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

[

WITH FILLFACTOR = fillfactor

| WITH ( < index\_option > [ , ...n ] )

]

[ ON { partition\_scheme\_name ( partition\_column\_name )

| filegroup | "default" } ]

| [ FOREIGN KEY ]

REFERENCES [ schema\_name . ] referenced\_table\_name [ ( ref\_column ) ]

[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ NOT FOR REPLICATION ]

}

#### ALTER TABLE

#### CROP TABLE

#### CREATE INDEX

## Creación de una base de datos

## Definición de tablas y campos

## Normalización de las tablas

## Utilidades

## Tipos de datos que se pueden almacenar

## Propiedades de los campos

## Introducción de datos

## Ordenamiento de datos

## Filtros

## Definición de la clave primaria y otros índices

## Definición de relaciones entre tablas

## Definición de reglas de integridad.