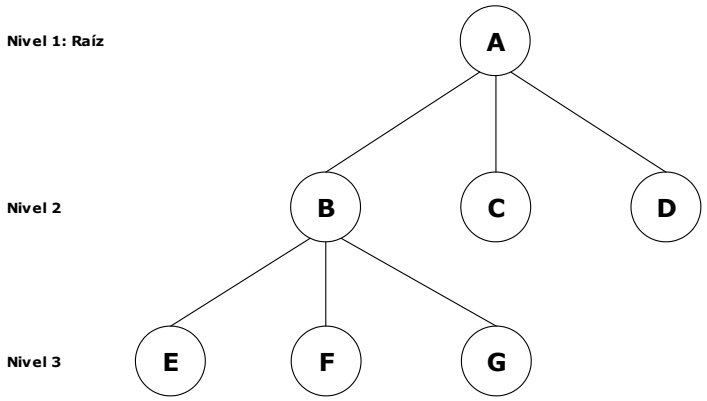
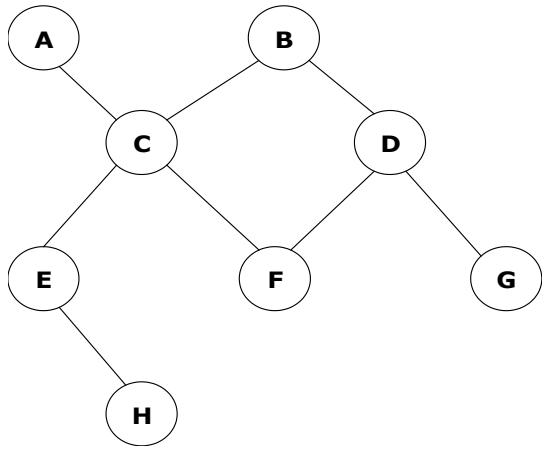
Universidad Nacional de la Patagonia San Juan Bosco
Facultad de Ingeniería

Cátedra: **Bases de datos I**

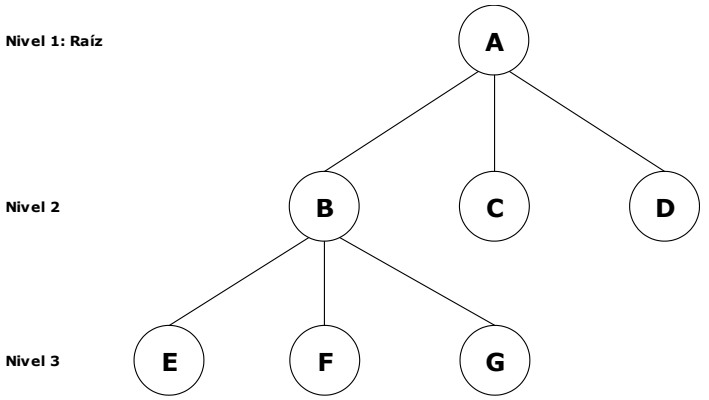
Evolución de los sistemas de Gestión de Bases de Datos





- Sistemas de información
  - Recopilan, estructuran, centralizan y almacenan datos. El objetivo es recuperar estos mismos datos u otros derivados de ellos en cualquier momento, sin necesidad de volverlos a recopilar, paso que suele ser el más costoso o incluso irrepetible. El objetivo final de un sistema de información → proporcionar a los usuarios información sobre el dominio que representan, con el objetivo de tomar decisiones y realizar acciones más pertinentes que las que se realizarían sin dicha información.
- Primeros Sistemas de Base de Datos
  - Finales de los ’50: evolucionados de los sistemas de archivos, obligaban a que el usuario visualizara los datos de manera muy parecida a como se almacenaban.
  - Durante los ’60: empezaron a aparecer distintos modelos de datos para describir la estructura de la información en una BD, con el objetivo de conseguir una independencia un poco mayor entre las aplicaciones y la organización física de los datos. Esto se consiguió inicialmente mediante la abstracción entre varios esquemas externos para las aplicaciones frente a la organización física de los mismos.
  - Los modelos más popularizados en esa época fueron el **modelo jerárquico** o basado en árboles, y el **modelo en red** o basado en grafos.

- Modelo Jerárquico
  - Fue el primero en aparecer y se caracterizó por organizar la información a través de una estructura de árboles, donde las relaciones entre registros se expresaban mediante una jerarquía. Dicha jerarquía distribuía y ordenaba los datos mediante un recorrido en Preorden (Consiste en visitar el nodo actual, y después visitar el subárbol izquierdo y una vez visitado, visitar el subárbol derecho. Es un proceso recursivo por naturaleza).

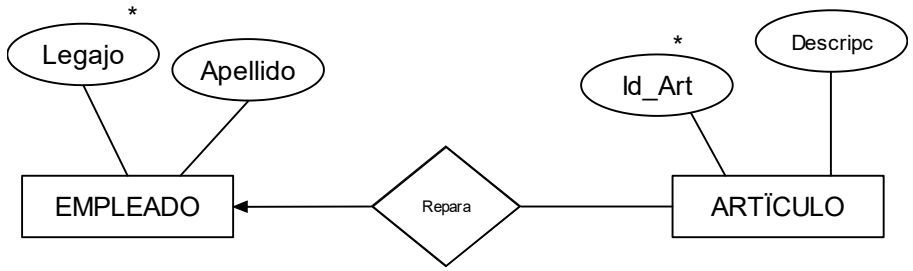


  - Las BD jerárquicas consisten en conjuntos de registros conectados entre si mediante enlaces. Cada registro consta de un conjunto de atributos que contiene un valor. Los enlaces son asociaciones entre dos registros.
  - Cada nodo del árbol representa la instancia de un registro constituido por un número de campos que lo describen mediante las propiedades o atributos de las entidades.

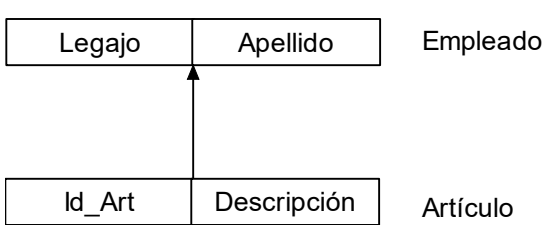
- Características de las BD jerárquicas:
  - Los registros están dispuestos en forma de árbol.
  - Los registros están enlazados mediante relaciones uno a muchos.
  - Cada nodo consta de uno o más campos.
  - Cada ocurrencia de un registro padre puede tener distinto número de ocurrencias de registros hijos.
  - Cuando se elimina un registro padre se deben eliminar todos los registros hijos, es la forma de garantizar la integridad de los datos.
  - Todo registro debe tener un único registro padre excepto la raíz.

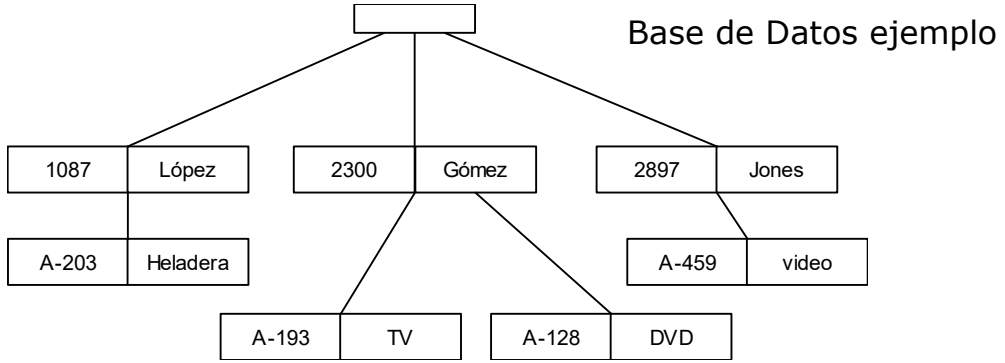
- Diagrama de Estructura de Árbol
  - Permiten representar la estructura lógica global de las BD jerárquicas.
  - Árbol con raíz:
  - ✓ El grafo no puede contener ciclos.
  - ✓ Las relaciones formadas en el grafo deben ser de tal forma que sólo existan relaciones uno a muchos o uno a uno entre un padre y un hijo.
  - El esquema de BD se representa como una colección de diagramas de estructura de árbol. La raíz de este árbol es un nodo ficticio. Los hijos de ese nodo son instancias del tipo de registro adecuado. Cada una de esas instancias hijo puede tener, a su vez, varias instancias de varios tipos de registro, según se especifica en el diagrama de estructura de árbol correspondiente.

- Para comprender cómo están formados los diagramas de estructura de árbol, veremos cómo transformar los diagramas E-R a sus correspondientes diagramas de estructura de árbol.
- Relaciones únicas. Considere el diagrama E-R siguiente:



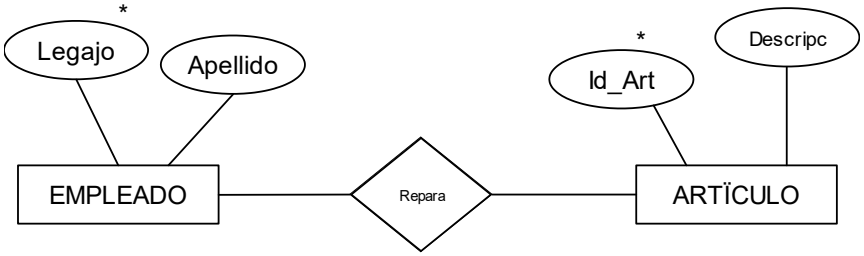
- El diagrama de estructura de árbol correspondiente es el siguiente:





- El tipo de registro *Empleado* corresponde al conjunto de entidades *EMPLEADO*. Incluye dos campos: *Legajo*, *Apellido*. De manera similar, *Artículo* es el tipo de registro que corresponde al conjunto de entidades *ARTICULO*. Incluye dos campos: *Id_Art* y *Descripción*. Finalmente, la relación *REPARA* se ha sustituido por el enlace *Repara*, con una flecha apuntando al tipo de registro *Empleado*.

– Si la relación REPARA es M:M:

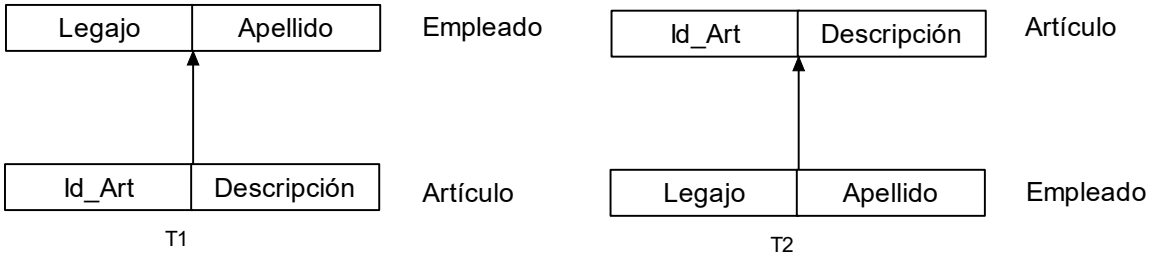


– La transformación es más complicada, se debe a que en el modelo jerárquico sólo pueden representarse directamente las **relaciones 1:M y 1:1**.
– Existen varias formas de transformar este diagrama E-R en un diagrama de estructura de árbol, todos estos diagramas comparten la propiedad de que el o los **árboles** de BD tendrán **registros repetidos**.
– Para transformar diagrama E-R anterior debemos hacer lo siguiente:

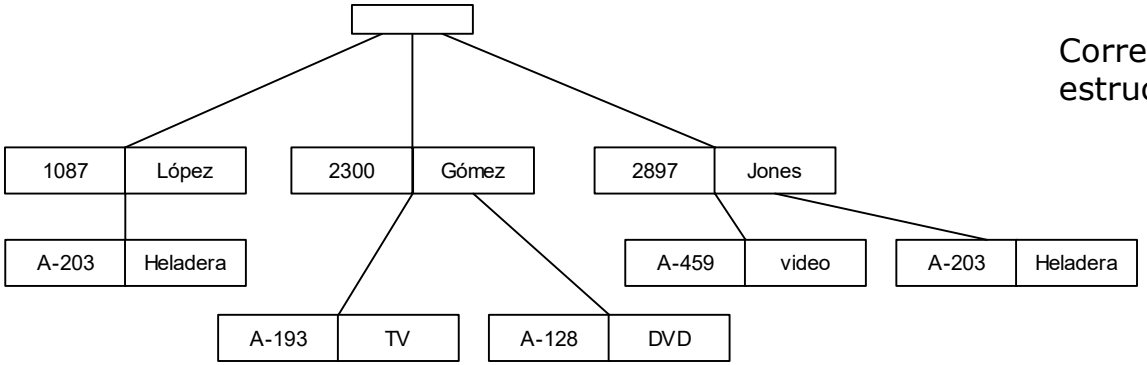
1. Crear dos diagramas de estructura de árbol distintos, T1 y T2, incluyendo los tipos de registro Empleado y Artículo. En el árbol T1, la raíz es Empleado, mientras que en el árbol T2 la raíz es Artículo.
2. Crear los dos enlaces siguientes:

✓ en T1, un enlace M:1 desde el registro Artículo al registro Empleado
✓ en T2, un enlace M:1 desde el registro Empleado al registro Artículo.

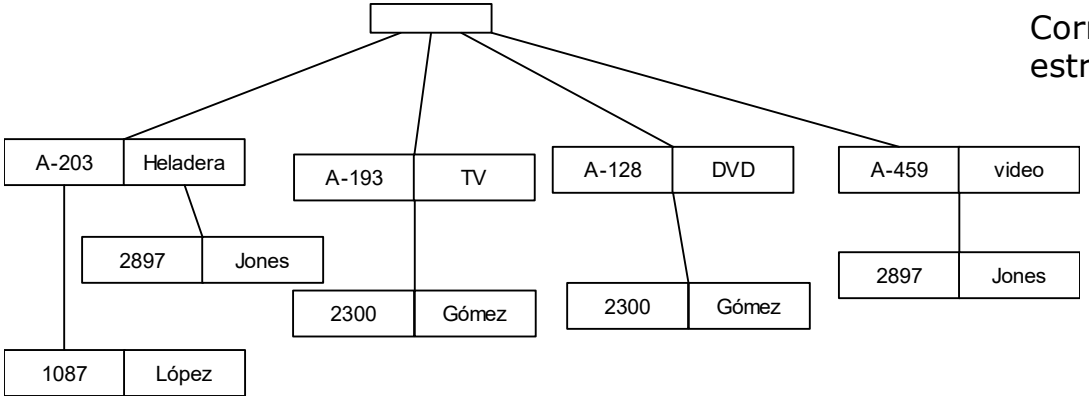
– El diagrama de estructura de árbol resultante es la siguiente:



– Un ejemplo de BD correspondiente a este diagrama es el siguiente:



Corresponde al diagrama de estructura de árbol T1



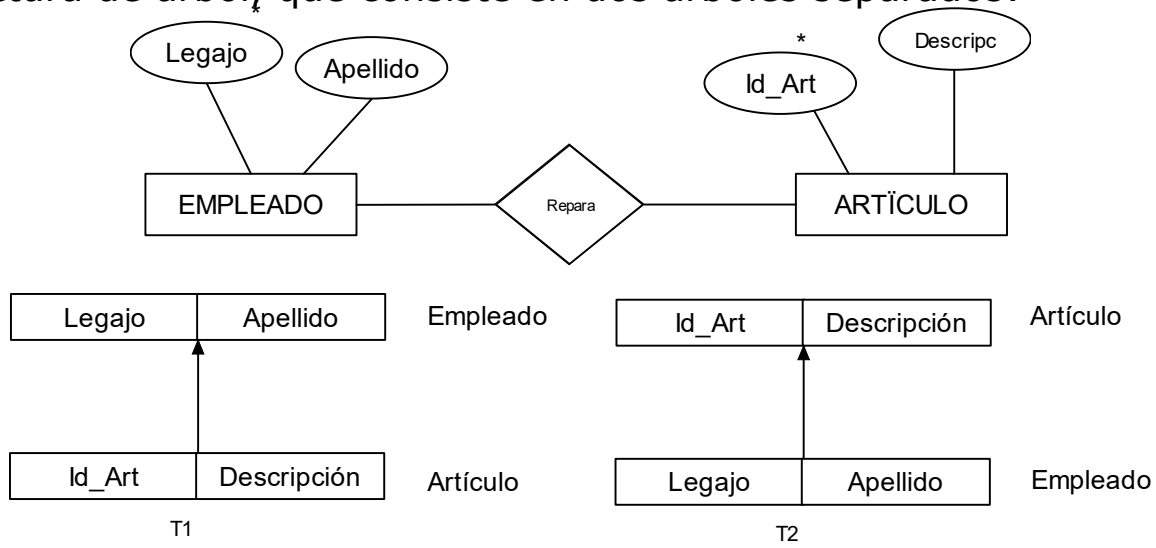
Corresponde al diagrama de estructura de árbol T2

– Todos los registros Empleado y Artículo están repetidos en ambos árboles de BD. Además, el registro artículo A-203 aparece dos veces en el primer árbol, mientras que los registros empleado Jones y Gómez aparecen dos veces en el segundo árbol.
– Si una relación incluye también un atributo descriptivo, la transformación de un diagrama E-R a un diagrama de estructura de árbol es más complicada. Esto se debe a que un enlace no puede contener el valor de un dato. En este caso se necesita crear un nuevo tipo de registro y establecer los enlaces adecuados.

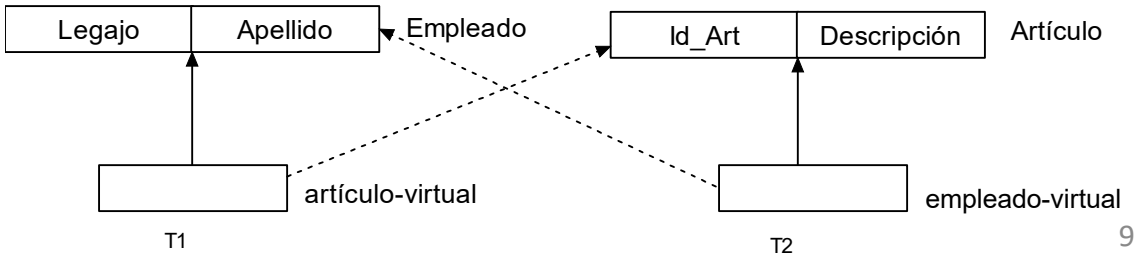
- Varias relaciones
  - El esquema que se acaba de describir para transformar un diagrama E-R en un diagrama de estructura de árbol garantiza que para cada relación sola, la transformación resultará en diagramas que tienen la forma de árbol con raíz.
  - Aplicar esta transformación de manera individual a cada una de las relaciones de un diagrama E-R no resulta necesariamente en diagramas que son árboles con raíz. La técnica para resolver el problema es dividir los diagramas en varios diagramas, cada uno de los cuales es un árbol con raíz.

- Registros Virtuales
  - Hemos visto que en el caso de las relaciones M:M es necesario repetir registros para conservar la organización de estructura de árbol de la BD. La repetición de registros tiene dos inconvenientes importantes:
  - ✓ La actualización puede producir inconsistencia en los datos
  - ✓ El desperdicio de espacio es inevitable
  - Para eliminar la repetición de registros necesitamos relajar el requisito de que la organización lógica de los datos debe tener forzosamente estructura de árbol. Sin embargo hay que tener cuidado al hacerlo, de lo contrario, estaríamos convirtiéndolo en un modelo red.
  - La solución es introducir el concepto de **registro virtual**. Un registro virtual no contiene valores, sino un **puntero lógico** a un registro físico determinado. Cuando se va a repetir un registro en varios árboles de la BD, mantenemos una sola copia de ese registro en uno de los árboles y sustituimos cada uno de los otros registros por uno virtual que contiene un puntero a ese registro físico.

– Consideremos nuevamente el diagrama E-R y su correspondiente diagrama de estructura de árbol, que consiste en dos árboles separados:



– Para eliminar la repetición de datos creamos dos tipos de registros virtuales: *empleado-virtual* y *artículo-virtual*. A continuación sustituimos el tipo de registro *Artículo* por el tipo de registro *artículo-virtual* en el primer árbol y el tipo de registro *Empleado* por el tipo de registro *empleado-virtual* en el segundo árbol. También añadimos una línea discontinua desde el registro *artículo-virtual* al registro *Artículo* y otra línea discontinua desde el registro *empleado-virtual* al registro *Empleado*, para especificar la asociación entre un registro virtual y su correspondiente registro físico. El diagrama de estructura de árbol resultante se muestra como sigue:



- Inconvenientes del modelo Jerárquico

  - La utilización de la estructura jerárquica en la actualidad es prácticamente nula. Cuando la información almacenada en la BD es muy grande, se vuelve inmanejable.

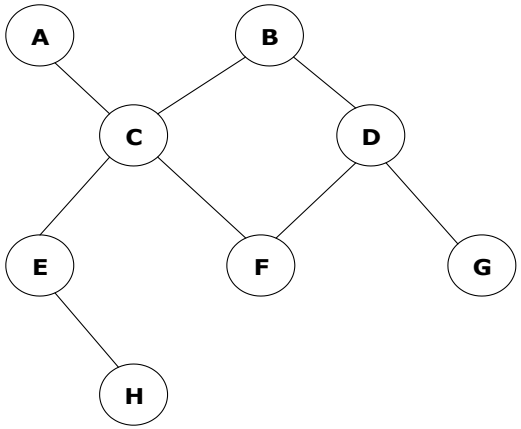
  - En cuanto a limitaciones lógicas cabe destacar que este modelo sólo soporta relaciones del tipo 1:M, las de tipo M:M requieren un inaceptable nivel de redundancia y no pueden ser implementadas de forma directa. Además, si un registro tipo aparece como hijo en más de dos relaciones, se debe replicar. Esto puede producir **problemas de integridad y consistencia de los datos**.

  - Los **lenguajes de manipulación** asociados son fuertemente **navegacionales**, actúan registro a registro y hace falta un lenguaje de programación anfitrión en el que se inserten los operadores que deben permitir al usuario moverse (navegar) por la BD.

  - Las reglas de integridad en el modelo jerárquico prácticamente se reducen a la ya mencionada **eliminación en cadena**.

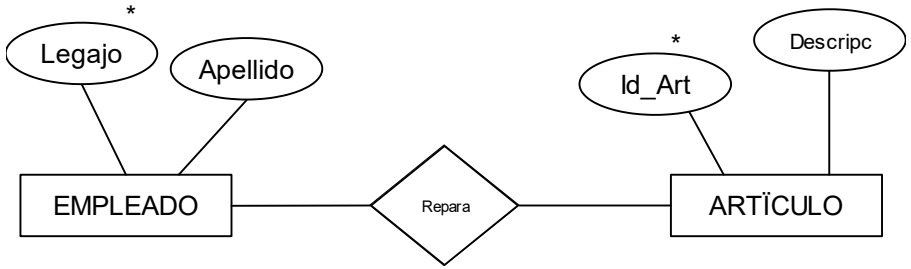
  - El modelo jerárquico no tiene una historia demasiado bien documentada. Se deriva de los sistemas de gestión de información de los cincuenta y los sesenta.

- Modelo Red
  - Una estructura de datos en red es muy similar a una estructura jerárquica. Al igual que en la estructura jerárquica, cada nodo puede tener varios hijos pero, a diferencia de ésta, también puede tener varios padres.
  - La figura muestra una estructura de datos en red. En esta representación, los nodos C y F tienen dos padres, mientras que los nodos D, E, G y H tienen sólo uno.

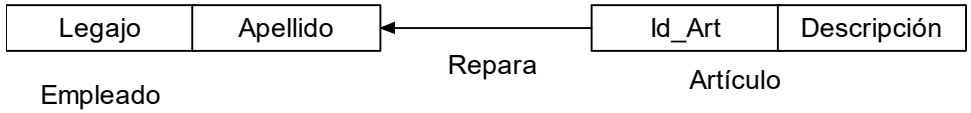


- Diagramas de Estructura de Datos
  - Un diagrama de estructura de datos es un esquema que representa el diseño de una BD de red. Este diagrama consta de dos componentes básicos:
  - ✓ Cajas, que corresponden a tipos de registro.
  - ✓ Líneas, que corresponden a enlaces.
  - Un diagrama de estructura de datos tiene el mismo propósito que uno de E-R, especificar la estructura lógica global de la BD.
  - Para comprender cómo se estructuran estos diagramas, utilizaremos el mismo enfoque que usamos para el modelo jerárquico: mostraremos cómo transformar los diagramas de E-R en los de estructura de datos correspondientes.

- Relación binaria
  - De acuerdo al diagrama E-R se presenta su correspondiente diagrama de estructura de datos :



  - En el diagrama de estructura de datos el tipo de registro *Empleado* corresponde al conjunto de entidades *EMPLEADO*. Incluye dos campos: *Legajo* y *Apellido*. *Artículo* es el tipo de registro correspondiente al conjunto de entidades *ARTICULO*. Incluye los campos *Id_art* y *Descripción*. Finalmente, la relación *REPARA* se ha sustituido por el enlace *Repara*.
  - La relación *REPARA* es M:M, si fuera de 1:M de *EMPLEADO* a *ARTICULO*, el enlace *REPARA* tendría una flecha apuntando al tipo de registro *Empleado*.



  - De manera similar, si la relación REPARA fuera de 1:1, el enlace Repara tendría dos flechas, uno apuntando al tipo de registro *Empleado* y otra apuntando al tipo de registro *Artículo*.

– Si la relación incluye atributos, la transformación de un diagrama E-R en un diagrama de estructura de datos es algo más complicada. Esto se debe al hecho de que un enlace no puede contener valores de datos. En este caso se necesita crear un nuevo tipo de registro y establecer los enlaces.

– Si agregamos el atributo *Fecha* a la relación *REPARA* del diagrama M:M entre *EMPLEADO* y *ARTICULO* (que indica la fecha en que el empleado reparó el artículo), para transformar este diagrama en uno de estructura de datos se debe:

1. Sustituir los conjuntos entidades *EMPLEADO* y *ARTICULO* por los tipos de registro *Empleado* y *Artículo*.
2. Crear un tipo de registro nuevo, *fecha*, con un solo campo para representar la fecha.
3. Crear los siguientes enlaces M:1:

✓ *EmpleadoFecha* del tipo de registro *fecha* al tipo de registro *Empleado*

✓ *ArtículoFecha* del tipo de registro *fecha* al tipo de registro *Artículo*

– El diagrama de estructura de datos es el siguiente:

- El Modelo CODASYL DBTG
  - La primera especificación estándar de una BD, llamada informe **CODASYL** (**CO**nference on **DA**ta **SY**stems and **L**anguages) **DBTG 1971**, fue escrita por el Grupo de Trabajo sobre Bases de Datos (Database Task Group). Desde entonces se han sugerido varios cambios a ese informe, el último oficial fue en 1978. En 1981 se publicó una nueva propuesta. Hemos elegido esta proposición como la fuente principal para nuestro tratamiento del modelo de DBTG.

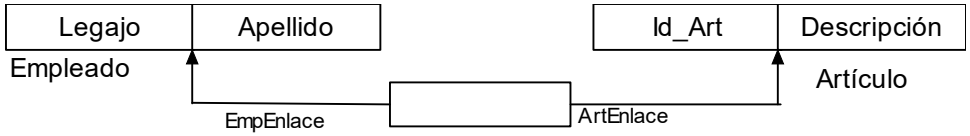
- Restricción de enlaces
  - En el modelo DBTG solamente pueden emplearse enlaces 1:M. Los enlaces M:M se prohíben para simplificar la implementación. Los enlaces 1:1 se representan utilizando un enlace 1:M. Estas restricciones implican que los diversos algoritmos anteriores para transformar un diagrama E-R en uno de estructura de datos deben revisarse.
  - Consideremos una relación binaria que sea 1:M o 1:1, en este caso puede aplicarse directamente el algoritmo de transformación definido anteriormente. Para la BD del ejemplo, si la relación *REPARA* es 1:M sin atributos descriptivos, entonces el diagrama de estructura de datos apropiado es el que vimos anteriormente.
  - Sin embargo, si la relación *REPARA* es M:M sin atributos descriptivos, el algoritmo de transformación debe redefinirse como sigue:

1. Sustituir los conjuntos entidades *EMPLEADO* y *ARTICULO* por los tipos de registro *Empleado* y *Artículo*.
2. Crear un tipo de registro nuevo ficticio, *Enlace*, que puede no tener campos o tener un campo con un identificador.
3. Crear los siguientes enlaces 1:M:

✓ *EmpEnlace* del tipo de registro *Enlace* al tipo de registro *Empleado*

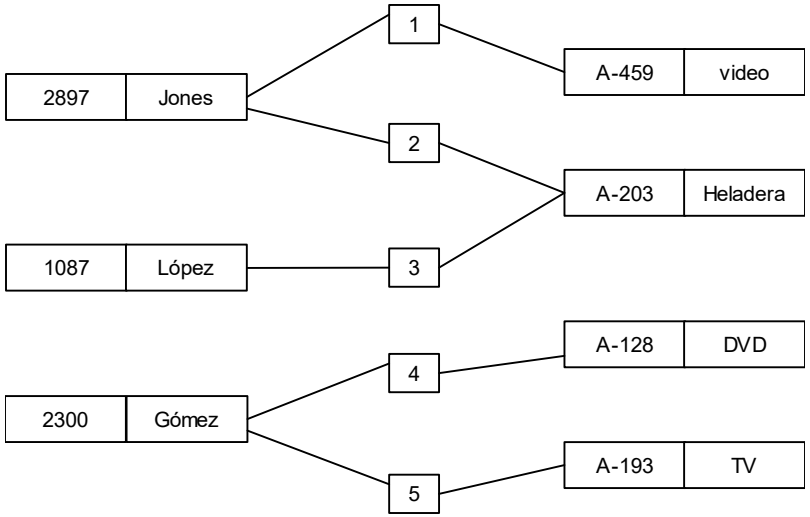
✓ *ArtEnlace* del tipo de registro *Enlace* al tipo de registro *Artículo*

– El diagrama de estructura de datos correspondiente es el siguiente:

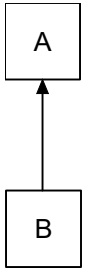


– Si la relación *REPARA* es M:M con un atributo descriptivo (p. e. fecha), entonces el algoritmo de transformación es similar al que se acaba de describir, pero el nuevo tipo de registro *Enlace* ahora contiene el campo *fecha*.

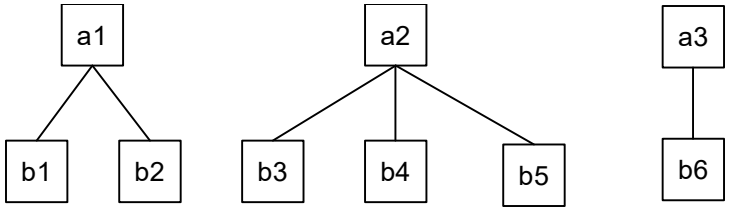
– Un ejemplo al diagrama anterior sería:



- Conjuntos DBTG
  - Dado que solamente pueden utilizarse enlaces 1:M en el modelo DBTG, un diagrama de estructura de datos consta de dos tipos de registro enlazados entre sí y tiene la forma general:

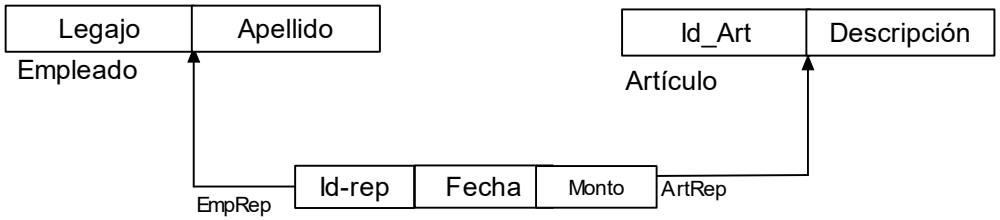


  - En el modelo DBTG, ésta estructura se denomina **Conjunto DBTG**. Como nombre del conjunto, generalmente se elige el nombre del enlace que conecta los dos tipos de registro. En un conjunto DBTG de este tipo, el tipo de **registro A** se denomina **dueño** (o padre) del conjunto, y el tipo de **registro B** se denomina **miembro** (o hijo) del conjunto. Cada conjunto DBTG puede tener cualquier número de ocurrencias, es decir, instancias reales de registros enlazados. Por ejemplo, tenemos tres ocurrencias del conjunto (instancias):



  - Puesto que no se permiten enlaces del tipo de M:M, cada ocurrencia del conjunto tiene exclusivamente un dueño y cero o más registros miembros. Además, ningún registro miembro 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.

– Para ilustrarlo, considérese el diagrama de estructura de datos siguiente:



– Existen dos conjuntos DBTG:

✓ *EmpRep*, cuyo dueño es *Empleado* y cuyo miembro es *Reparación*

✓ *ArtRep*, cuyo dueño es *Artículo* y cuyo miembro es *Reparación*

- Inconvenientes del modelo Red
  - Este modelo intenta superar las deficiencias del enfoque jerárquico, convirtiendo una relación M:M en dos relaciones 1:M mediante la inserción de un nuevo tipo de registro. Su inconveniente es la complejidad que alcanza el entramado de enlaces entre las instancias cuando se almacena gran cantidad de datos, así como la hostilidad de los lenguajes de programación y control de estas BD.
  - El lenguaje de manipulación de datos de CODASYL es de tipo navegacional y tiene que estar embebido en un lenguaje de programación; en él se distingue la selección o localización de la acción: recuperación o actualización.
  - 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 este último los tipos de registro se organizan en forma de un **grafo arbitrario**, mientras que en el primero se organizan en forma de **árbol con raíz**.

  ↳ Los dos modelos permiten únicamente operaciones y facilidades navegacionales primitivas.

- Bases de Datos Relacionales

  - Ni los productos CODASYL, ni los sistemas jerárquicos satisfacían las exigencias de un buen SGBD. Edgar F. Codd comenzó a trabajar en una nueva forma de organizar y acceder a los datos. A partir de estos trabajos publicó el artículo "**A Relational Model of Data for Large Shared Data Banks**" en 1970.

  - Codd propuso que los SGBD deberían presentarse a los usuarios con una visión de los datos organizados en estructuras llamadas relaciones (Tablas). Las relaciones se definen como conjuntos de tuplas, no como series o secuencias de objetos, con lo que el orden no es importante y un conjunto de campos (columnas) → detrás de una relación puede haber cualquier estructura de datos compleja que permita una respuesta rápida a una variedad de consultas.

  - El usuario de un sistema relacional no tenía que preocuparse de la estructura de almacenamiento, sólo debía preocuparse por **qué** consultar y no **cómo**.

  - Las consultas podían expresarse en un lenguaje de muy alto nivel, lo que incrementaba en gran medida la eficiencia de los programadores sobre BD.

  - En **resumen**, Codd concibió un sistema donde el usuario sería capaz de acceder a la información con comandos parecidos al lenguaje natural y donde la información estuviera almacenada en tablas.

- Pese a sus virtudes, la aceptación del modelo relacional no fue inmediata, debido en parte a su base matemática que, aunque muy simple, no era común para la industria de BD de la época. Además, se dudaba de la eficiencia del modelo y se había invertido una gran cantidad de esfuerzo y dinero en los productos ya existentes. La nueva tecnología relacional debía demostrar que era mucho mejor que la existente para cambiar la situación.

- Sin embargo la Universidad de California en Berkeley creyó en la idea del modelo relacional y obtuvo financiación para desarrollar un sistema relacional, **Ingres**. El producto se revisó continuamente durante toda la década en base a los comentarios y retroalimentación de muchos usuarios de otras Universidades y centros que lo adaptaban. Ingres incluía su propio lenguaje de consultas, llamado **QUEL**.

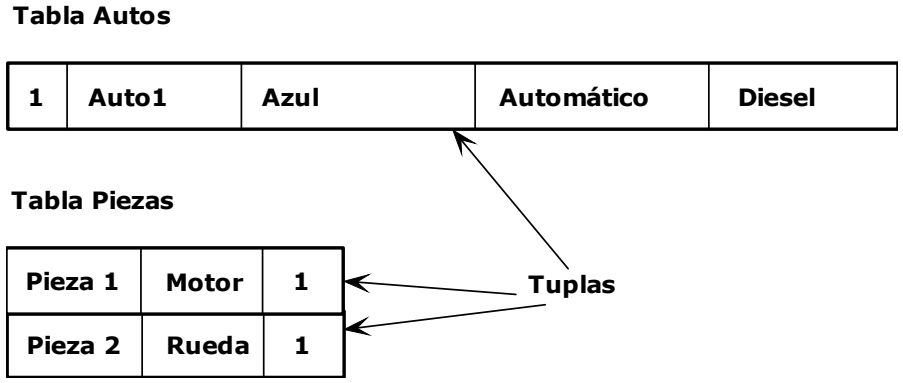
- El grupo de investigación de IBM puso en marcha el desarrollo de otro sistema relacional, el **System R**. Más adelante se trabajó en el rediseño de System R como un sistema multiusuario, con un lenguaje de consulta estructurado, el **SEQUEL** que luego pasaría a llamarse **SQL** (**S**tructured **Q**uery **L**anguage).

- Durante estos desarrollos se produjo la publicación de la separación en tres niveles de los SGBD (Arquitectura ANSI/SPARC): nivel de visión, conceptual y físico.

- Respecto a los lenguajes de consulta, Codd introdujo el **álgebra relacional** y consideró el uso de la lógica para realizar consultas, que llevaría a lenguajes lógicos como el Cálculo Relacional de Tuplas o Dominios. (Los que desarrollaremos en unidades posteriores)

- Necesidad de un Sistema de Gestión de Bases de Datos OO

  - El Modelo Relacional es muy diferente del OO. Por una parte, el modelo relacional sólo se ocupa de la parte estática de la aplicación (datos) y no de la parte dinámica (procesos). Por ejemplo, un auto tiene ciertas características: color, tipo de caja de cambios, tipo de combustible, etc. y puede realizar determinadas acciones: acelerar, frenar, etc. En el modelo relacional, sólo interesan las características del auto y no las acciones que se pueden hacer con él. Este énfasis en los datos es lógico en un modelo cuyo objetivo es modelar la parte estática de la aplicación, es decir, la BD. Además, la forma en que el modelo relacional trata los datos es muy diferente a cómo lo hace el modelo OO. Mientras en este último, los datos son modelados en forma de objetos, en el modelo relacional son modelados como tuplas, las cuales son una serie de datos pertenecientes a una misma entidad de la vida real. Una tupla difiere de un objeto en que sólo modela datos.
  - Las tuplas similares se agrupan en relaciones. Cada tupla contendría los datos de un determinado auto (color rojo, caja de cambios automática, diesel, etc.) y las tablas no serían más que listas de autos con todos sus datos.



- Si un auto contiene piezas, el objeto Auto contiene objetos Pieza, en el modelo relacional las tuplas de los autos y piezas están por separado y se relacionan por claves. El programador debe saber que hay una relación entre los autos y las piezas, y debe programar de acuerdo a ello.
- Los SGBDR actuales no están concebidos para manejar grandes cantidades de datos, ni datos complejos, aunque la mayor parte de los productos relacionales ofrezcan la posibilidad de definir BLOB's (Binary Large Objects) para almacenar este tipo de datos, pero este mecanismo no posibilita expresar la semántica asociada a los objetos multimedia, ni realizar accesos por determinados componentes de ellos (color, forma u otras características) → El modelo relacional no refleja completamente la estructura de la realidad.
- Una aplicación actual, generalmente, está formada por un programa y una BD que se comunican entre sí. Programa → modelo OO y la BD → modelo relacional, esto introduce una dificultad importante, porque los dos formatos de datos (objetos y tuplas) son incompatibles entre sí. Así, cuando el programa quiere guardar o recuperar los datos en disco, lo hace en forma de objetos, sin embargo, la BD no puede guardar o recuperar objetos, porque está diseñada para tuplas. Resumiendo, los formatos de datos que utilizan la aplicación y la BD son incompatibles entre sí, produciendo una **desadaptación de impedancias**.
- La solución más obvia a este problema es hacer que uno de los componentes use el formato de datos del otro. Así, la primera opción que se presenta es que el programa y la BD estén diseñados para tratar con datos relacionales … **¿?**
- Actualmente podemos distinguir dos enfoques en la integración de la OO en las BD. En primer lugar, de manera natural aparecen los Sistemas de Gestión de Bases de Datos Objeto Relacional (SGBDOR), que son una extensión de las Bases de Datos Relacionales. En segundo lugar aparecen los Sistemas de Gestión de Bases de Datos Orientados a Objeto (SGBDOO).

- Bases de Datos Objeto Relacionales

  - El modelo objeto relacional es un desarrollo más reciente y parece haber tenido bastante efecto. No es una tecnología en sí, sino una cohesión de los modelos relacional y OO.

  - Este modelo surgió dada la necesidad de la industria y de sus usuarios de tratar con nuevos tipos de datos: audio, imágenes y video, además de tipos definidos por el usuario con sus propiedades. Por otra parte, las organizaciones eran reticentes a migrar de un SGBDR a un SGBDOO por diversos motivos. Y el mantenimiento de los SGBDR empezaba a crear desadaptación de impedancias, con un más generalizado uso de los lenguajes OO.

  - Aquí es donde el modelo objeto relacional puede demostrar su valor, éste se define como una **extensión OO del modelo relacional**, permitiendo la **definición de nuevos tipos** y de **relaciones de herencia**, entre otras cosas. Además, permite a las organizaciones continuar usando sus sistemas existentes y sus datos, sin realizar prácticamente cambios y les permiten empezar a utilizar gradualmente características OO, especialmente si se hace en conjunción con aplicaciones desarrolladas en entornos también OO.

  - Uno de los ejemplos de esta combinación es JDBC (Java DataBase Connectivity), aunque pensado para interrelacionar BDR con Java, va incorporando algunos detalles objeto relacionales. De una manera más precisa, JDBC es una API (Interfaz de los programas de aplicación) desarrollada para conectar Java con las BD.

- Bases de Datos Orientadas a Objetos

  - Alrededor de la mitad de los ochenta, algunas aplicaciones exigían mayor expresividad en los datos con los que trabajaban. Por ejemplo, las BD médicas, BD multimedia y algunas BD científicas requerían mayor flexibilidad en la manera en que los datos se representaban y eran accedidos.

  - Coincidiendo con la entrada de los lenguajes OO como Smalltalk o C++, los investigadores se plantearon reflejar estas ideas en las BD y permitir que el tipo de datos marcara cómo se representaba y se manipulaba, dependiendo de los métodos que se definían para dicho tipo o clase. La idea de una BDOO se articuló por primera vez con el sistema prototipo **GemStone**. Al principio de los noventa, los primeros SGBDOO empezaron a aparecer en el mercado, introducidos por compañías como **Objectivity**.

  - En este modelo, la información sobre una entidad se almacena como un objeto persistente y no como una fila en una tabla. Esto lo hace más eficiente en términos de requerimientos de espacio y asegura que los usuarios puedan manipular los datos sólo de las maneras en las que el programador haya especificado. A esto hay que sumar las ventajas derivadas del modelo OO, mayor expresividad y adecuación para almacenar muchos tipos de datos diferentes.

  - ¿Las BDOO deberían haber superado en la práctica a las relacionales? A veces se denominan postrelacionales y no obstante, después de más 30 años, el mercado de las BDOO no supone más de un pequeño porcentaje del mercado de las relacionales.

- Razones: **Primero**, las BDOO acarrean consigo algunas de las propiedades no deseables de los modelos prerelacionales. El programador tiene que tener mucha información sobre la estructura de los datos. Lo que preocupa o interesa es almacenar los atributos de los objetos y relacionar los valores de estos atributos, aspecto en el que el modelo relacional es más sencillo. **Segundo** y quizás más importante es que las organizaciones tiendan a ser conservadoras en relación con las BD, uno de sus activos más valiosos. Muchas organizaciones aunque utilizan lenguajes OO para las aplicaciones, desconfían de los lenguajes OO en general por no considerarlos suficientemente estables para trabajar con información crucial para la organización.

- Mito o realidad, el hecho es que no acaban de decidirse por cambiar por un SGBDOO y siguen aferrados al SQL para realizar sus informes y al SQL embebido para interrelacionar las aplicaciones con el SGBD, manteniendo una separación que consideran imprescindible.

- Del mismo modo que evolucionaba el modelo OO para el análisis y diseño de los sistemas a modelar, debían también evolucionar o desarrollarse nuevos modelos conceptuales y metodologías OO para el diseño de BD. El modelo E-R se mostró insuficiente para la etapa de diseño conceptual de BDOO y se importaron lenguajes de modelado e incluso gran parte de las metodologías utilizadas en ingeniería del software OO (UML). Aunque pensadas para el desarrollo de software OO, se han ido adaptando o importando al campo de las BD como herramienta de modelado conceptual de BDOO.

- UML fue aceptado como estándar por el **ODMG** (**O**bject **D**ata **M**anagement **G**roup). Dado que puede utilizarse para cualquier sistema OO, es apropiado para el **diseño de BDOO**.

- Especialmente para la interacción entre analistas y programadores.

- Finalmente hay que recordar que UML es sólo un lenguaje y no una metodología, con lo que también es posible utilizar UML como notación para expresar el modelo entidad-relación.

- ODMG es un grupo de vendedores y usuarios de BD que desarrollan estándares para los SGBDOO.

- El primer documento fue el **ODMG 1.0** en 1993, que incluía el **OQL** (Object Query Language), un lenguaje de consultas OO inspirado en SQL.

- Aunque a principios de los noventa tuvo bastante fuerza, hoy su importancia es menor debido a que **SQL3** incluye muchas características OO.

- Además del OQL, el **ODMG v.3.0** incluye un modelo de objetos estándar y enlaces para los tres lenguajes OO más populares: C++, Smalltalk y Java.

- Gestores de Objetos

  - Los gestores de objetos, también llamados almacenes de objetos persistentes, proporcionan **menos funcionalidad** que los SGBDOO.

  - Están diseñados para gestionar el **almacenamiento persistente de un número de objetos relativamente pequeño y con bajos requerimientos de concurrencia**. Son por tanto, poco apropiados para muchos usuarios.

  - Normalmente, ofrecen una distribución de datos limitada, no proporcionan evolución de esquemas y carecen de un lenguaje de consulta o de un lenguaje de programación.

  - La mayoría de estos gestores vienen en **forma de API** (Application Program Interface) que será utilizada desde el lenguaje de programación correspondiente (C++ o Java principalmente), y actualmente existe una gran cantidad de ellos, sobre todo para Java (PSE Pro, etc.).

- Evolución de SQL

  - **SQL** (**S**tandar **Q**uery **L**enguaje) es un lenguaje que interactúa con BD. Es bastante sencillo y se utiliza para la creación y manipulación de una BD.
  - SQL es un lenguaje de acceso a BD que explota la flexibilidad y potencia de los sistemas relacionales, permitiendo gran variedad de operaciones. Es un **lenguaje declarativo de alto nivel** o de no procedimiento, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación. De esta forma una sola sentencia puede equivaler a uno o más programas que utilizasen un lenguaje de bajo nivel orientado a registro.
  - Sus orígenes están ligados a los de las BD relacionales, en 1970 Codd propone el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados.
  - Basándose en estas ideas los laboratorios de IBM definen el lenguaje **SEQUEL** (**S**tructured **E**nglish **QUE**ry **L**anguage) y entre 1976 y 1977, se realizaron revisiones del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo **System R**, basado en este lenguaje, se adoptó y utilizó internamente en IBM algunos de sus clientes elegidos. Gracias al éxito de este sistema, no comercializado aún, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL, fue **Oracle** quien lo introdujo por primera vez en 1979 en un programa comercial. En el curso de los años ochenta, numerosas compañías comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las BD relacionales.

- SQL es una versión evolucionada de SEQUEL y pasa a ser el lenguaje por excelencia de los diversos SGBD relacionales surgidos en los años siguientes y es estandarizado en 1986 por el ANSI, dando lugar a la **primera versión estándar** de este lenguaje, el **SQL-86** o **SQL1**. Al año siguiente este estándar es también adoptado por la ISO.

- El hecho de tener un estándar definido por un lenguaje para BD relacionales abre potencialmente el camino a la intercomunicabilidad entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa en la propia BD sólo el **corazón del lenguaje SQL**, extendiéndolo de manera individual según la propia visión.

- Debido a que este primer estándar no cubre todas las necesidades de los desarrolladores, en 1992 se lanza un **nuevo estándar ampliado** y revisado del SQL llamado **SQL-92** o **SQL2**. En la actualidad el SQL2 es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares es amplia, el soporte al estándar SQL-92 es general y muy amplio.

- En el año 1999 un nuevo proceso de revisión del lenguaje por parte de los comités ANSI e ISO dio lugar al estándar **SQL3**. En el año 2003 se produce una revisión menor, en la que se incorporan características de XML.

- Revisiones del ANSI SQL:

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor. Define el estándar para el SQL embebido
1992	SQL-92	SQL2	Revisión mayor. Nuevos operadores, mejor tratamiento de errores y normas para el SQL embebido.
1999	SQL:1999	SQL3	Se agregaron expresiones regulares, tipos de datos, consultas recursivas (para relaciones jerárquicas), triggers, roles de usuarios y algunas características OO.
2003	SQL:2003		Introduce algunas características de XML,cambios en las funciones, estandarización del objeto sequence y de las columnas autonuméricas.
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras importar y guardar datos XML en una base dedatos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades quepermiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL ydocumentos XML.
2008	SQL:2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.

- Revisiones del ANSI SQL:
  - XML es un método para introducir datos estructurados en un archivo de texto.
  - Cuando pensamos en "datos estructurados" pensamos en cosas tales como hojas de cálculo, libretas de direcciones, transacciones financieras, dibujos técnicos, etc.
  - Los programas que producen esta clase de datos a menudo también los guardan en disco, por lo que pueden usar tanto un formato binario como un formato texto. El último formato permite, si es necesario, ver los datos sin el programa que los ha producido.
  - XML consiste en una serie de reglas para planificar formatos texto para tales datos, de manera que produzcan archivos que sean fácilmente generados y leídos (por una computadora).