Lenguaje de programación

Un lenguaje de programación es un conjunto de símbolos y de reglas para combinarlos, que se usan para expresar algoritmos. Se caracterizan por:

- Ser independientes de la arquitectura física del computador
- Una sentencia en un lenguaje de alto nivel da lugar, tras el proceso de traducción, varias instrucciones en lenguaje máquina.
- Algo expresado en un lenguaje de alto nivel utiliza notaciones más cercanas a las habituales en el ámbito en que se usan.

Un **traductor** es un programa que recibe como entrada un texto en un lenguaje de programación concreto y produce, como salía, un texto en lenguaje máquina equivalente.

Entrada → lenguaje fuente, que define a una máquina virtual.

Salida → lenguaje objeto, que define a una máquina real.

La forma en la que un programa escrito para una máquina virtual es posible ejecutarlo en una máquina real puede ser gracias a un compilador o a un intérprete.

Un **compilador** traduce la especificación de entrada a lenguaje máquina **incompleto** y con instrucciones máquina **incompletas**, originando la necesidad de un enlazador.

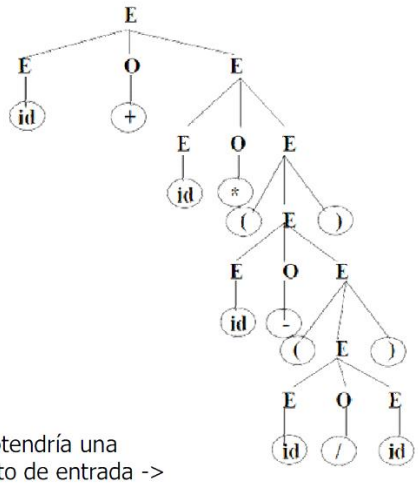
Un **enlazador** realiza el enlazado de los programas completando las instrucciones máquina necesarias y generando un programa ejecutable para la máquina real.

Gramática

Ejemplo

Dada la gramática siguiente:

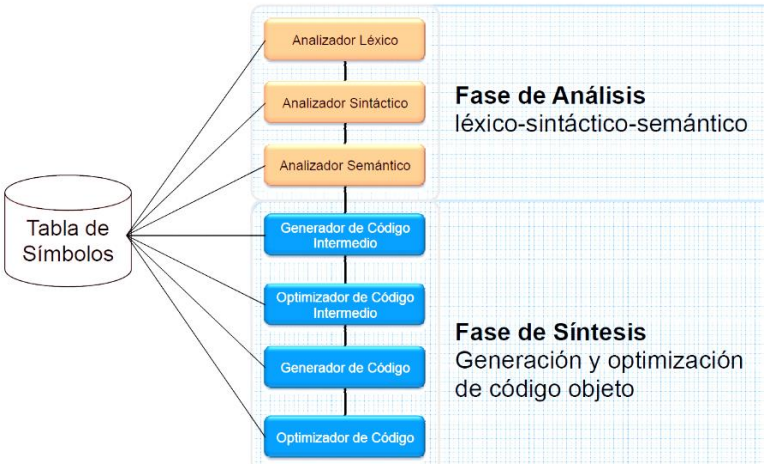
$$
\begin{aligned}
P = \{\ & E \rightarrow E\ O\ E \\
& \quad\ \mid (\ E\ ) \\
& \quad\ \mid id \\
& O \rightarrow + \mid - \mid * \mid / \\
\}& \\
V_N &= \{E,O\} \\
V_T &= \{(,),id,+,-,*,/\} \\
S &= E
\end{aligned}
$$



Y el texto de entrada: **id+id*(id-(id/id))**

Usando las reglas de formación gramatical, se obtendría una representación que valida la construcción del texto de entrada -> verificación sintáctica correcta.

Fases de traducción



Análisis léxico

Su función es leer los caracteres de la entrada del programa fuente, agrupados en lexemas (palabras) y producir como salida una secuencia de tokens para cada lexema en el programa fuente.

- Un lexema es una secuencia de caracteres del alfabeto con significado propio.
- Un token es un concepto asociado a un conjunto de lexemas que, según la gramática del programa fuente, tienen la misma misión sintáctica.
- Un patrón es una descripción de la forma que pueden tomar los lexemas de un token.

En muchos lenguajes de programación, se cubren la mayoría de los siguientes tokens:

- Un token para cada palabra reservada (if, do while, else, …)
- Los tokens para los operadores (individuales o agrupados)
- Un token que representa a todos los identificadores tanto de variables como de subprogramas
- Uno o más tokens que representan a las constantes (números y cadenas de literales)
- Tokens para cada signo de puntuación (paréntesis, llaves, coma, punto…)

El error léxico se producirá cuando el carácter de la entrada no tenga asociado a ninguno de los patrones disponibles en nuestra lista de tokens (ej.: whi¿le)

Análisis sintáctico

Su objetivo es analizar las secuencias de tokens y comprobar que son correctas sintácticamente. Beneficios:

- Una gramática proporciona una especificación sintáctica precisa de un lenguaje de programación.
- A partir de ciertas clases gramaticales, es posible construir de manera automática un analizador sintáctico eficiente.
- Permite revelar ambigüedades sintácticas y puntos problemáticos en el diseño del lenguaje.
- Una gramática permite que el lenguaje pueda evolucionar o se desarrolle de forma iterativa agregando nuevas construcciones.

A partir de una secuencia de tokens, el analizador sintáctico devuelve:

- Si la secuencia es correcta o no sintácticamente (debe existir un conjunto de reglas gramaticales aplicables para poder estructurar la secuencia de tokens)
- El orden en el que hay que aplicar las producciones de la gramática para obtener la secuencia de entrada (árbol sintáctico).

Si no se encentra un árbol sintáctico para una secuencia de entrada, entonces la secuencia de entrada es incorrecta sintácticamente.

Una gramática definida como $G = (V_N, VT, P, S)$, donde:

- V_N es el conjunto de símbolos no terminales
- V_T es el conjunto de símbolos terminales
- P es el conjunto de producciones
- S es el símbolo inicial

se dice que es una gramática **libre de contexto** cuando el conjunto de producciones P sólo admiten tener un símbolo no terminal en su parte izquierda. Una gramática es **ambigua** cuando admite más de un árbol sintáctico para una misma secuencia de símbolos de entrada. El analizador sintáctico no hace uso de la tabla de símbolos (el registro de los nombres y tipos de cada variable).

Análisis semántico

La semántica de un lenguaje de programación es el significado dado a las distintas construcciones sintácticas. En los lenguajes de programación, el significado está ligada a la estructura sintáctica de las sentencias.

Durante la fase de análisis semántico se producen errores cuando se detectan construcciones sin un significado correcto (variable no declarada, tipos incompatibles en una asignación, llamada a un procedimiento incorrecto o con número de argumentos incorrectos, ...)

En lenguaje C es posible realizar asignaciones entre variables de distintos tipos (algunos compiladores devuelven warnings de que algo puede realizarse mal), otros como Pascal lo impiden.

Generación de código

En esta fase se genera un archivo con un código en lenguaje objeto (generalmente lenguaje máquina) con el mismo significado que el texto fuente. En algunos, se intercala una fase de generación de código intermedio para proporcionar independencia de las fases de análisis con respecto al lenguaje máquina (portabilidad del compilador)

Optimización de código

Esta fase existe para mejorar el código mediante comprobaciones locales a un grupo de instrucciones (bloque básico) o a nivel global. Se pueden realizar optimizaciones de código tanto al código intermedio (si existe) como al código objeto final. Generalmente, las optimizaciones se aplican a códigos intermedios.

Intérpretes

Un intérprete lee un programa fuente escrito para una máquina virtual, realiza la traducción de manera interna y **ejecuta una a una** las instrucciones obtenidas para la máquina real. Consecuencias inmediatas:

- No se crea un archivo o programa objeto almacenable en memoria para posteriores ejecuciones.
- La ejecución del programa escrito en lenguaje fuente está supervisada por el intérprete.

Es útil cuando:

- El programador trabaja en un entorno interactivo y se desean obtener los resultados de la ejecución de una instrucción antes de ejecutar la siguiente.
- El programador lo ejecuta escasas ocasiones y el tiempo de ejecución no es importante.
- Las instrucciones del lenguaje tienen una estructura simple y pueden ser analizadas fácilmente.
- Cada instrucción será ejecutada una sola vez.

En cambio, no es eficiente cuando:

- Las instrucciones del lenguaje son complejas
- Los programas van a trabajar en modo de producción y la velocidad es importante
- Las instrucciones serán ejecutadas con frecuencia.

Modelos de memoria de un proceso

Los elementos responsables de la gestión de memoria son: un lenguaje de programación, un compilador, un enlazador, un sistema operativo, la MMU…

La gestión de la memoria se divide en diferentes niveles:

- Nivel de procesos: Responsabilidad del SO, se encarga del reparto de memoria entre los procesos.
- Nivel de regiones: Gestionado por el SO, distribuye el espacio asignado a un proceso a las regiones del mismo
- Nivel de zonas: Gestionado por el lenguaje de programación con soporte del SO, se encarga del reparto de una región entra las diferentes zonas de esta.

Un proceso necesita:

- Tener un espacio lógico independiente, protegido del resto de procesos.
- La posibilidad de compartir memoria, con soporte de diferentes regiones.
- Facilidad de depuración.
- Uso de un mapa amplio de memoria, y de diferentes tipos de objetos de memoria.
- Persistencia de datos.
- Desarrollo modular y carga dinámica de módulos.

Existen distintos tipos de datos: estáticos; globales; constantes o variables; con o sin valor inicial; datos dinámicos asociados a una función (que son almacenados en pila en un registro de activación); datos dinámicos controlados por el programa (heap).

Código independiente de la posición (PIC)

Un fragmento de código cumple esta propiedad si puede ejecutarse en cualquier parte de la memoria. Para ello es necesario que todas sus referencias a instrucciones o datos no sean absolutas sino relativas a un registro, por ejemplo, el contador del programa.

Ciclo de vida de un programa

A partir de un código fuente, un programa debe pasar por varias fases antes de poder ejecutarse:

Compilación

El compilador procesa cada uno de los archivos de código fuente para generar el correspondiente archivo objeto.

1. Genera código objeto y calcula cuánto espacio ocupan los diferentes tipos de datos.
2. Asigna direcciones a los símbolos estáticos (instrucciones o datos) y resuelve las referencias bien de forma absoluta o relativa (necesita reubicación).
3. Las referencias a símbolos dinámicos se resuelven usando direccionamiento relativo a la pila para dato relacionados a la invocación de una función, o con direccionamiento indirecto para el heap. No necesitan reubicación al no aparecer en el archivo objeto.
4. Por último, se genera la Tabla de símbolos e información de depuración.

Enlazado

El enlazador debe agrupar los archivos objetos de la aplicación y las bibliotecas, resolviendo las referencias entre ellos. En ocasiones, debe realizar reubicaciones dependiendo del esquema de gestión de memoria utilizado.

1. Se completa la etapa de resolución de símbolos externos utilizando la tabla de símbolos.
2. Se agrupan las regiones de similares características de los diferentes módulos en **regiones** (código, datos inicializados o no)
3. Se realiza reubicación de módulos, transformando las referencias de un módulo a referencias dentro de las regiones. Tras esta fase cada archivo objeto tiene una lista de reubicación que contiene los nombres de los símbolos y los desplazamientos dentro dela archivo que deben aún parchearse.
4. En sistemas paginados, se realiza la reubicación de regiones, es decir, transformar direcciones de una región en direcciones del mapa del proceso.

Existen varios tipos de enlazado:

- Externo: de ámbito global. - Interno: con ámbito dentro del fichero. - Sin enlazado: con visibilidad de bloque.

//Reglas de enlazado

- Cualquier objeto/identificador que tenga ámbito global deberá tener enlazado interno si su declaración contiene el especificador static.
- Si el mismo identificador aparece con enlazados externo e interno, dentro del mismo fichero, tendrá enlazado externo.
- Si en la declaración de un objeto o función aparece el especificador de tipo de almacenamiento extern, el identificador tiene el mismo enlazado que cualquier declaración visible del identificador con ámbito global. Si no existiera tal declaración visible, el identificado tiene enlazado externo.
- Si una función es declarada sin especificador de tipo de almacenamiento, su enlazado es el que correspondería si se hubiese utilizado extern
- Si un objeto (que no sea una función) de ámbito global a un fichero es declarado sin especificar un tipo de almacenamiento, dicho identificador tendrá enlazado externo (ámbito de todo programa). Como excepción, los objetos declarados const que no hayan sido declarados explícitamente extern tienen enlazado interno.

Carga y ejecución

La reubicación del proceso se realiza en la carga o ejecución. Existen tres tipos según el esquema de gestión de memoria:

- El cargador copia el programa en memoria sin modificarlo. Es la MMU la encargada de realizar la reubicación en ejecución.
- En paginación, el hardware es capaz de reubicar los procesos en ejecución por lo que el cargador lo carga sin modificación.
- Si no usamos hardware de reubicación, ésta se realiza en la carga.

Los archivos objeto (resultado de la compilación) y ejecutable (resultado del enlazado) se diferencian en:

- En el ejecutable la cabecera del archivo contiene el punto de inicio del mismo, es decir, la primera instrucción que se cargará en el PC.
- En cuanto a las regiones, sólo hay información de reubicación si ésta se ha de realizar en la carga.

Bibliotecas

Una biblioteca es una colección de objetos, normalmente relacionados entre sí. Estas favorecen modularidad y reusabilidad de código, y se pueden clasificar en:

- Estáticas: Se enlazan con el programa en la **compilación**. Se trata de un conjunto de archivos que se copian en un único archivo.
- Dinámicas: Se enlazan en **ejecución**. El código de la biblioteca debe estar en todos los ejecutables que la usan, y al actualizar la biblioteca se debe recompilar los programas que las usan.
  Estas se integran en ejecución, a través de reubicación de módulos. A diferencia del ejecutable, tienen tabla de símbolos, información de reubicación y no tienen punto de entrada. Pueden ser:
  - Compartidas de carga dinámica: La reubicación se realiza en tiempo de **enlazado**.
  - Compartidas enlazadas dinámicamente: El enlazado se realiza en **ejecución**.

Tema 4

Un **archivo** es un conjunto de información sobre un mismo tema, tratada como una unidad de almacenamiento y organizada de forma estructurada para la búsqueda de un dato individual.

Los **registros** son las estructuras homogéneas o unidades que forman el archivo y que contienen la información correspondiente a cada elemento individual. Un **campo** es un dato que forma parte de un registro y representa una información unitaria e independiente.

El sistema operativo permite que el usuario pueda aludir al archivo mediante un nombre, independientemente de la forma en que se almacene en el dispositivo, además de suministrar órdenes para: crear/copiar/borrar/renombrar un archivo; abrir/cerrar; leer/escribir…

El sistema operativo transporta, cada vez que se accede al dispositivo, una cantidad fija de información que depende de las características físicas de éste: bloque o registro físico, de longitud distinta al tamaño del registro.

El sistema operativo transforma la dirección lógica usada en los programas de usuario en la dirección física con la que se direcciona el soporte.

Un archivo es una estructura de datos externa al programa; en las operaciones de lectura/escritura se transfiere información a/desde un buffer de memoria principal asociado a las entradas/salidas sobre el archivo.

Clasificación de archivos según el tipo de registros

- Longitud fija
- Longitud variable:
  - **Con delimitador**: Un determinado carácter llamado delimitador **marca** el fin de un registro (ej.: salto de línea)
  - **Con cabecera:** Cada registro contiene un campo **inicial** que almacena el número de bytes del registro.
- Indefinido: El archivo no tienen realmente ninguna estructura interna, el SO no realiza ninguna gestión sobre la longitud de los registros. En cada operación de E/S se transfiere una determinada subcadena del archivo, siendo el programa de usuario quien se encargue de localizar el principio y final de cada registro.

Concepto de directorio

El SO permite que el conjunto de archivos sea visible para el usuario según una estructura jerárquica en que aparece el concepto de directorio como agrupación de otros archivos o directorios.

Organización secuencial: Los registros están almacenados físicamente **contiguos**.

Las distintas operaciones que se pueden realizar son:

- Añadir: Sólo es posible escribir al **final** del archivo. La información se graba en el archivo escribiendo los registros en forma secuencia, en el orden en que se desea que estén en el archivo.
- Consulta: Se realiza en orden secuencial, es decir, para leer el registro **n** es necesario leer previamente los **n-1** registros anteriores.

- Inserción, modificación y eliminación: La modificación de un registro necesita que no se aumente su longitud. No es posible eliminar un registro del archivo, en cambio, se usa el **borrado lógico** (marcarlo de tal forma que al leer se identifique como no válido)

Si el archivo se encuentra en un soporte direccionable y los registros son de longitud fija, entonces es posible determinar la dirección de comienzo de cada registro a partir de su posición relativa dentro del archivo. Por tanto, se puede acceder a un registro conociendo su posición relativa sin necesidad de leer secuencialmente desde el principio.

Adecuada para cuando se necesita únicamente un acceso secuencial y se procesarán la mayor parte de los registros.

Ventajas: Buen aprovechamiento del espacio; sencilla de utilizar; bajo precio.

Organización secuencial encadenada: Junto a cada registro se almacena un puntero con la dirección siguiente.

- Consulta: La consulta es secuencial.
- Inserción:
  - Se localiza entre qué dos registros se quiere insertar el nuevo registro.
  - Se escribe el registro en una zona libre y como puntero asignamos la dirección del registro que debe ser el siguiente.
  - Se actualiza el puntero del anterior de forma que contenga la dirección del nuevo registro.
- Borrado: Se asigna al puntero del registro anterior la dirección del registro siguiente
- Modificación: Si el cambio no implica aumentar la longitud del registro, éste se puede reescribir. En caso contrario se inserta un registro y se borra el anterior

La principal ventaja es la facilidad de inserción y borrado; su principal inconveniente, que su limitación es la consulta secuencial

Organización secuencial indexada: El archivo se constituye de la zona de registros y la zona de índices.

Zona de registros: Se encuentran los registros en sí, ordenados según el valor de la llave; está dividida en diferentes tramos (conjuntos de registros consecutivos)

Por cada tramo de la zona de registros hay un registro en la zona de índices que contiene el valor de la llave del tramo (del último registro) y la dirección del primer registro del tramo.

- Consulta: Esta organización de archivo permite realizar consultas por llave sin necesidad de leer los registros que le anteceden en el archivo. El procedimiento a seguir es:
  - Leer en secuencia las llaves en la zona de índices hasta encontrar una **mayor o igual** a la del registro buscado. Obtener la dirección de comienzo del tramo donde está el registro.
  - Leer en secuencia en la zona de registros a partir de la dirección obtenida en la zona de índices hasta encontrar el registro buscado o uno con valor de llave mayor que el buscado. (En este último caso el registro no se encuentra en el archivo)
- Inserción: Dado que ambas zonas son secuenciales, no es posible insertar un registro en archivos con esta organización. En algunos casos se permite la escritura de nuevos registros al final de la zona de registros. Estos registros, como es lógico, no podrán ser consultados por llave con el procedimiento antes descrito.
- Borrado y modificación: Al estar los registros escritos en secuencia no es posible borrar un registro. La única forma de eliminar la información contenida en un registro es marcándolo, lo que se conoce como borrado lógico. Las modificaciones son posibles tan sólo si el registro no aumenta de longitud al ser modificado y no se altera el valor de la llave del mismo.

Organización directa o aleatoria: Un archivo escrito sobre un soporte de acceso directo para el cual existe una transformación conocida que genera la dirección de cada registro en el archivo a partir de un campo que se usa como llave.

El nombre de "aleatorio" se debe a que normalmente no existe ninguna vinculación aparente entre el orden lógico de los registros y su orden físico. Un problema fundamental de esta organización es elegir adecuadamente de la transformación o método de direccionamiento que se ha de utilizar. Situaciones no deseadas que pueden ocurrir:

- Hay direcciones que no se corresponden con ninguna llave y, por tanto, habrá zonas de disco sin utilizar.
- Hay direcciones que corresponden a más de una llave. En este caso se dice que las llaves son sinónimas para esa transformación.

Hay dos formas de resolver el problema de los sinónimos, siempre a costa de complicar la estructura del archivo:

- Cuando se asocia a una llave una dirección ya ocupada por un registro distinto (esto es, por un sinónimo de esta llave), se busca en el archivo hasta encontrar una posición libre donde escribir el registro.

- Se reserva una zona de desbordamientos donde se escribirán los registros que no se pueden escribir en la posición que les corresponde según la transformación, a la que denominaremos zona principal.

Hay tres métodos de direccionamiento para los archivos de organización directa:

- Direccionamiento directo: Se utiliza como dirección la propia llave
  Sólo es factible cuando la llave es numérica y su rango de valores no es mayor que el rango de direcciones en el archivo.
  En algunos casos pueden quedar lagunas de direcciones sin utilizar, en lugares conocidos de antemano. En este caso se pueden ocupar dichas direcciones desplazando las direcciones superiores.
  Por ejemplo, el archivo de habitaciones de un hotel puede organizarse en forma aleatoria con direccionamiento directo, sin más que hacer la dirección igual al número de habitación.
- Direccionamiento asociado: El direccionamiento asociado se puede utilizar para cualquier tipo de llave. Si se utiliza este método debe construirse una tabla en la que se almacena para cada llave la dirección donde se encuentra el registro correspondiente.
- Direccionamiento calculado ("hashing"): La dirección de cada registro se obtiene realizando una transformación sobre la llave. Se utiliza cuando:
  - La llave no es numérica; se usará una conversión previa para obtener un número a partir de ella. Por ejemplo, se usa el equivalente decimal al propio código binario del carácter (al carácter A le correspondería el 65…)
  - La llave es numérica, pero toma valores en un rango inadecuado para usarse directamente como dirección.

Operaciones sobre un archivo con organización directa:

- Consulta: Se realiza por llave.
  Para leer un registro debe aplicarse a la llave el algoritmo de transformación y posteriormente leer el registro en la dirección resultante.
  Si el registro con la llave buscada no se encuentra allí se procederá según cómo se haya resuelto gestionar los sinónimos.
- Borrado: Siempre se realiza un borrado lógico, pudiéndose reutilizar el espacio del registro eliminado.
- Modificación e inserción: Siempre se puede modificar o insertar un nuevo registro, realizándola transformación de la llave correspondiente.

La organización directa es útil para archivos donde los accesos deben realizarse por llave, accediéndose siempre a registros concretos. Si la información se va a procesar en conjunto, con frecuencia puede ser más rentable una organización secuencial indexada.

Bases de datos: concepto y tipos

En una aplicación convencional con archivos aparecen los siguientes problemas:

- Dificultad de mantenimiento: Si hay archivos con información parcialmente duplicada, realizar las actualizas necesarias es un problema complejo y costoso. Normalmente, es necesario actualizar varios archivos con diferentes organizaciones. Si la actualización no se realiza correctamente se tendrán informaciones incoherentes.
- Redundancia: Se dice que hay redundancia si un dato se puede reducir a partir de otros datos (se dan los problemas explicados en el caso anterior)
- Rigidez de búsqueda: El archivo se concibe para acceder a los datos de un determinado modo. Sin embargo, en la mayoría de los casos es necesario (o al menos deseable) combinar acceso secuencial y directo por varias claves.
- Dependencia con los programas: En un archivo no están reflejadas las relaciones existentes entre campos y registros. Es el programa, que trabaja con el archivo, el que determina en cada caso dichas relaciones. Esto implica que cualquier modificación de la estructura de un archivo obliga a modificar todos los programas que lo usen. Esto ocurre aun en el caso de que la alteración sea ajena al programa. Así, por ejemplo, si se aumenta la longitud de un campo habrá que modificar incluso los programas que no lo usan.
- Seguridad: Si se está trabajando con archivos el control deberá realizarlo el propio programa.

Concepto de base de datos

Una base de datos es un sistema formado por un **conjunto de datos y un paquete software** para gestión de dicho conjunto de datos de tal modo que se controla el almacenamiento de datos redundantes, los datos resultan independientes de los programas que los usan, se almacenan las relaciones entre los datos junto con éstos y se puede acceder a los datos de diversas formas.

En una base de datos se almacenan las **relaciones** entre datos junto a los datos. Al utilizar como unidad de almacenamiento el campo además de un registro, es el fundamento de la independencia con los programas de aplicación.

Requisitos que debe cumplir un buen sistema de base de datos:

- **Diversos usuarios** pueden acceder a la base de datos, sin que se produzcan conflictos.
- Cada usuario podrá tener una **imagen** o visión particular de la estructura de la base de datos.
- Se podrán usar **distintos métodos** de acceso, con tiempos de respuesta razonablemente pequeños.
- Se controlará el acceso a los datos (a nivel de campo), impidiéndolo a usuarios no autorizados.
- Deben existir mecanismos concretos de recuperación en caso de fallo de la computadora.
- Se puede cambiar el soporte físico de la base de datos sin que esto repercuta en ella ni los programas que la utilizan.
- Se pueden **modificar** los datos contenidos en la base, las relaciones existentes entre ello o incluir nuevos datos sin afectar a los programas que la usan.
- Los datos se almacenan **una sola vez**.
- Existe una forma sencilla y cómoda de utilizar la base al menos desde un lenguaje de programación de alto nivel.
- (query) Existe una utilidad que permite el acceso a los datos de forma interactiva.

Estructura de una base de datos

En una base de datos se alacena información de una serie de objetos o elementos. Estos objetos reciben el nombre de entidades, siendo una entidad cualquier ente sobre el que se almacena información.

De cada entidad se almacenan una serie de datos que se denominan atributos de la entidad. Puede ser atributo de una entidad cualquier característica o propiedad de ésta.

Las entidades y los atributos son conceptos abstractos. En una base de datos la información de cada entidad se almacena en registros, y cada atributo en campos de dicho registro, de forma análoga al almacenamiento de archivos. Sin embargo, en una base de datos hay diferentes tipos de registros, uno por cada entidad.

Se dice que un conjunto de atributos, de una entidad es un identificador o clave primaria de dicha entidad si el valor de dichos atributos determina de forma unívoca cada uno de los elementos de dicha entidad, y no existe ningún subconjunto de él que sea identificador de la entidad.

En una base de datos se almacenan además de las entidades, las relaciones existentes entre ellas. Estas relaciones se almacenan con punteros que insertan automáticamente el sistema de gestión de la base de datos.

Las relaciones entre entidades pueden ser de estos tipos:

- Relación 1 a 1: Si a cada elemento de una entidad A le corresponde un único elemento de B.
- Relación 1 a N: Si a cada elemento de A le pueden corresponder más de uno de B, y a cada uno de B le corresponde un único de A.
- Relación N a N: Si a cada elemento de A le pueden corresponder más de uno de B, y a cada uno de B le puede corresponder más de uno de A.

Al definir la base de datos se deben especificar cada uno de los registros que la integran, indicando los campos que los componen y las relaciones que los ligan. Tal conjunto de especificaciones recibe el nombre de esquema de la base de datos.

El esquema de una base de datos es la definición de su estructura lógica, suele representarse en forma gráfica o por simple enumeración.

Para que un programa concreto pueda acceder a una base de datos es necesario especificar la estructura lógica de la parte de base de datos que éste va a usar. La descripción de la estructura lógica de una parte de la base de datos que va a ser utilizada por uno o más programas recibe el nombre de vista o subesquema.

El subesquema permite que varios usuarios utilicen distintas “visiones” de una misma base de datos. También permite limitar el acceso a distintas partes de la base de datos, para realizar tan solo determinadas acciones.

Tipos de bases de datos

- Jerárquicas: Se establece una relación entre los datos en forma de árbol, no existen las relaciones muchos a muchos.
- En red: Se trata de un modelo similar al jerárquico que permite las relaciones muchos a muchos
- Relacional: Está formada por tablas (estructura bidimensional formada por una sucesión de registros del mismo tipo). Si se imponen ciertas condiciones a las tablas, se pueden tratar como relaciones matemáticas:
    - Todos los registros de una tabla son del mismo tipo
    - En ninguna tabla aparecen campos repetidos
    - No existen registros duplicados

- El orden es indiferente
- En cada tabla hay una llave

Se denomina sistema de gestión de la base de datos al conjunto de software destinado a la creación, control y manipulación de la información de una base de datos. Permite:

- Acceso a los datos desde algún lenguaje de alto nivel.
- Interrogación directa en modo conversacional.
- Definición del esquema de la base de datos y de los distintos subesquemas.
- Organización física de la base de datos y recuperación tras fallos del sistema

El sistema actúa como intermediario entre los programas de aplicación y el SO, lo que permite que los programas sean independientes de la estructura física de los datos.