TIPOS DE LENGUAJES DE PROGRAMACION

LENGUAJES DE MÁQUINA

El lenguaje máquina de una computadora consta de cadenas de números binarios (ceros y unos) y es el único que "entienden" directamente los procesadores. Todas las instrucciones preparadas en cualquier lenguaje de máquina tienen por lo menos dos partes. La primera es el comando u operación, que dice a la computadora cuál es la función que va a realizar. Todas las computadoras tienen un código de operación para cada una de sus funciones. La segunda parte de la instrucción es el operando, que indica a la computadora dónde hallar o almacenar los datos y otras instrucciones que se van a manipular; el número de operandos de una instrucción varía en las distintas computadoras. En una computadora de operando único, el equivalente binario de "SUMAR 0814" podría hacer que se sume el valor que se encuentra en la localidad de almacenamiento o dirección 0814 al valor que se encuentra en la unidad aritmética lógica. En una máquina de dos operandos, la representación binaria de "SUMAR 0814 8672" podría hacer que se sume el valor que está en la localidad 8672 al valor que está en la dirección 0814. El formato de operando único es popular en las microcomputadoras más pequeñas; la estructura de dos operandos se encuentra en casi todas las demás máquinas.

Según los estándares actuales, las primeras computadoras eran poco tolerantes. Los programadores tenían que traducir las instrucciones de manera directa a la forma de lenguaje de máquina que comprendían las computadoras. Por ejemplo, un programador que escribiera la instrucción "SUMAR 0814" para una de las primeras máquinas IBM hubiera escrito:

00010000000000000000000000010111000

Además de recordar las docenas de códigos numéricos para los comandos del conjunto de instrucciones de la máquina, el programador tenía que conocer las posiciones donde se almacenan los datos y las instrucciones. La codificación inicial muchas veces requería meses, por lo que era costosa y era frecuente que originara errores. Revisar las instrucciones para localizar errores era casi tan tedioso como escribirlas por primera vez. Además, si era necesario modificar un programa posteriormente, la tarea podía llevarse meses.

1.3. LENGUAJES ENSAMBLADORES

A principios de la década de 1950, y con el fin de facilitar la labor de los programadores, se desarrollaron códigos nemotécnicos para las operaciones y direcciones simbólicas. La palabra nemotécnico se refiere a una ayuda para la memorización. Uno de los primeros pasos para mejorar el proceso de preparación de programas fue sustituir los códigos de operaciones numéricos del lenguaje de máquina por símbolos alfabéticos, que son los códigos nemotécnicos. Todas las computadoras actuales tienen códigos nemotécnicos aunque, naturalmente, los símbolos que se usan varían en las diferentes marcas y modelos. La computadora sigue utilizando el lenguaje de máquina para procesar los datos, pero los programas ensambladores traducen antes los símbolos de código de operación especificados a sus equivalentes en lenguaje de máquina.

Este procedimiento preparó avances posteriores. Si la computadora era capaz de traducir símbolos convenientes en operaciones básicas, ¿por qué no hacer también que realizara otras funciones rutinarias de codificación, como la asignación de direcciones de almacenamiento a los datos? La técnica de direccionamiento simbólico permite expresar una dirección no en términos de su localización numérica absoluta, sino en términos de símbolos convenientes para el programador.

Durante las primeras etapas del direccionamiento simbólico, el programador asigna un nombre simbólico y una dirección real a un dato. Por ejemplo, el programador podría asignar el valor total de mercancía adquirida durante un mes por un cliente de una tienda de departamentos a la dirección 0063, y darle el nombre simbólico TOTAL. Se podría asignar el valor de la mercancía devuelta sin usar durante el mes a la dirección 2047 y dársele el nombre simbólico CRÉDITO. Así, durante el resto del programa, el programador se referirá a los nombres simbólicos, más que a las direcciones, cuando fuera preciso procesar estos datos. Por ejemplo, se podría escribir la instrucción "S CRÉDITO TOTAL" para restar el valor de las mercancías devueltas del importa total de compras para obtener el importe de la factura mensual del cliente. A continuación, el programa ensamblador traduciría la instrucción simbólica a esta cadena de bits:

011111	011111111111	000000111111
Código nemotécnico de operación	2047	0063
(S)	(CRÉDITO)	(TOTAL)

Más adelante se hizo otra mejora. Se dejó a la computadora la tarea de asignar y recordar las direcciones de las instrucciones. Lo único que tenía que hacer el programador era indicar a la computadora la dirección de la primera instrucción, y el programa ensamblador se encargaba de almacenar, de manera automática, todas las demás en forma secuencial a partir de ese punto. Así, si se agregaba más tarde otra instrucción al programa, no era necesario modificar las direcciones de todas las instrucciones que seguían al punto de inserción (como tendría que hacerse en el caso de programas escritos en lenguaje de máquina). En vez de ello, el procesador ajustaba automáticamente las localidades de memoria la próxima vez que se ejecutaba el programa.

En la actualidad, los programadores no asignan números de dirección reales a los datos simbólicos, simplemente especifican dónde quieren que se coloque la primera localidad del programa, y el programa ensamblador se encarga de lo demás: asigna localidades tanto para las instrucciones como para los datos.

Estos programas de ensamble, o ensamblador, también permite a la computadora convertir las instrucciones en lenguaje ensamblador del programador en su propio código de máquina. Un programa de instrucciones escrito en lenguaje ensamblador por un programador se llama programa fuente. Después de que el ensamblador convierte el programa fuente en código de máquina a éste se le denomina programa objeto. Para los programadores es más fácil escribir instrucciones en un lenguaje ensamblador que en códigos de lenguajes de máquina, pero es posible que se requieran dos corridas de computadora antes de que se puedan utilizar las instrucciones del programa fuente para producir las salidas deseadas.

Los lenguajes ensambladores tienen ventajas sobre los lenguajes de máquina. Ahorran tiempo y requieren menos atención a detalles. Se incurren en menos errores y los que se cometen son más fáciles de localizar. Además, los programas en lenguaje ensamblador son más fáciles de modificar que los programas en lenguaje de máquina. Pero existen limitaciones. La codificación en lenguaje ensamblador es todavía un proceso lento. Una desventaja importante de estos lenguajes es que tienen una orientación a la máquina. Es decir, están diseñados para la marca y modelo específico de procesador que se utiliza, y es probable que, para una máquina diferente, se tengan que volver a codificar los programas.

1.4. LENGUAJES DE ALTO NIVEL

Los primeros programas ensambladores producían sólo una instrucción en lenguaje de máquina por cada instrucción del programa fuente. Para agilizar la codificación, se desarrollaron programas ensambladores que podían producir una cantidad variable de instrucciones en lenguaje de máquina por cada instrucción del

programa fuente. Dicho de otra manera, una sola macroinstrucción podía producir varias líneas de código en lenguaje de máquina. Por ejemplo, el programador podría escribir "LEER ARCHIVO", y el programa traductor produciría una serie detallada de instrucciones al lenguaje de máquina previamente preparadas, con lo que se copiaría un registro del archivo que estuviera leyendo el dispositivo de entrada a la memoria principal. Así, el programador no se tenía que ocupar de escribir una instrucción por cada operación de máquina realizada.

El desarrollo de las técnicas nemotécnicas y las macroinstrucciones condujo, a su vez, al desarrollo de lenguajes de alto nivel que a menudo están orientados hacia una clase determinada de problemas de proceso. Por ejemplo, se han diseñado varios lenguajes para procesar problemas científico-matemático, asimismo han aparecido otros lenguajes que hacen hincapié en las aplicaciones de proceso de archivos.

A diferencia de los programas de ensamble, los programas en lenguaje de alto nivel se pueden utilizar con diferentes marcas de computadores sin tener que hacer modificaciones considerables. Esto permite reducir sustancialmente el costo de la reprogramación cuando se adquiere equipo nuevo. Otras ventajas de los lenguajes de alto nivel son:

- Son más fáciles de aprender que los lenguajes ensambladores.
- Se pueden escribir más rápidamente.
- Permiten tener mejor documentación.
- Son más fáciles de mantener.

Un programador que sepa escribir programas en uno de estos lenguajes no está limitado a utilizar un solo tipo de máquina.

2. LENGUAJES COMPILADOS

Naturalmente, un programa que se escribe en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores. Éstos, como los programas ensambladores avanzados, pueden generar muchas líneas de código de máquina por cada proposición del programa fuente. Se requiere una corrida de compilación antes de procesar los datos de un problema.

Los compiladores son aquellos cuya función es traducir un programa escrito en un determinado lenguaje a un idioma que la computadora entienda (lenguaje máquina con código binario).

Al usar un lenguaje compilado (como lo son los lenguajes del popular Visual Studio de Microsoft), el programa desarrollado nunca se ejecuta mientras haya errores, sino hasta que luego de haber compilado el programa, ya no aparecen errores en el código.

3. LENGUAJES INTERPRETADOS

Se puede también utilizar una alternativa diferente de los compiladores para traducir lenguajes de alto nivel. En vez de traducir el programa fuente y grabar en forma permanente el código objeto que se produce durante la corrida de compilación para utilizarlo en una corrida de producción futura, el programador sólo carga el programa fuente en la computadora junto con los datos que se van a procesar. A continuación, un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente en lenguaje de máquina conforme vaya siendo necesario durante el proceso de los datos. No se graba el código objeto para utilizarlo posteriormente.

La siguiente vez que se utilice una instrucción, se le debe interpretar otra vez y traducir a lenguaje máquina. Por ejemplo, durante el procesamiento repetitivo de los pasos de un ciclo, cada instrucción del ciclo tendrá que volver a ser interpretado cada vez que se ejecute el ciclo, lo cual hace que el programa sea más lento en tiempo de ejecución (porque se va revisando el código en tiempo de ejecución) pero más rápido en tiempo de diseño (porque no se tiene que estar compilando a cada momento el código completo). El intérprete elimina la necesidad de realizar una corrida de compilación después de cada modificación del programa cuando se quiere agregar funciones o corregir errores; pero es obvio que un programa objeto compilado con antelación deberá ejecutarse con mucha mayor rapidez que uno que se debe interpretar a cada paso durante una corrida de producción.

4. LENGUAJES DE PROGRAMACIÓN DECLARATIVOS

Se les conoce como lenguajes declarativos en ciencias computacionales a aquellos lenguajes de programación en los cuales se le indica a la computadora qué es lo que se desea obtener o qué es lo que se esta buscando, por ejemplo: Obtener los nombres de todos los empleados que tengan más de 32 años. Eso se puede lograr con un lenguaje declarativo como SQL.

La programación declarativa es una forma de programación que implica la descripción de un problema dado en lugar de proveer una solución para dicho problema, dejando la interpretación de los pasos específicos para llegar a dicha solución a un intérprete no especificado. La programación declarativa adopta, por lo tanto, un enfoque diferente al de la programación imperativa tradicional.

En otras palabras, la programación declarativa provee el "qué", pero deja el "cómo" liberado a la implementación particular del intérprete. Por lo tanto se puede ver que la programación declarativa tiene dos fases bien diferenciadas, la declaración y la interpretación.

Es importante señalar que a pesar de hacer referencia a intérprete, no hay que limitarse a "lenguajes interpretados" en el sentido habitual del término, sino que también se puede estar trabajando con "lenguajes compilados".

4.1. CARACTERÍSTICAS DE LOS LENGUAJES DE PROGRAMACIÓN DECLARATIVOS

Los lenguajes declarativos están orientados a buscar la solución del problema, sin preocuparse por la forma de llegar a ello; es decir, el programador debe concentrarse en la lógica del algoritmo, más que en el control de la secuencia.

Los programas están formados por un conjunto de definiciones o ecuaciones, las cuales describen lo que debe ser calculado, no en sí la forma de hacerlo.

Las variables sólo pueden tener asignado un solo valor a lo largo de la ejecución del programa, lo cual implica que no puede existir asignación destructiva. Debido a esto, cobra especial importancia el uso del anidamiento y la recursividad.

Las listas representan la estructura fundamental de datos.

El orden de la ejecución no resulta importante debido a que no existen efectos colaterales; es decir, que al calcular un valor, resulta imposible afectar el cálculo de otros y con esto se puede afirmar que cualquier secuencia de ejecución deberá conducir al mismo resultado.

Las expresiones o definiciones pueden ser usadas como valores y por lo tanto se pueden tratar como argumentos de otras definiciones.

El control de la ejecución no es responsabilidad del programador.

4.2. DESVENTAJAS DE LA PROGRAMACIÓN DECLARATIVA

La principal desventaja de la programación declarativa es que no puede resolver cualquier problema dado, sino que está restringida al subconjunto de problemas para los que el intérprete o compilador fue diseñado.
Otra desventaja de la programación declarativa está relacionada con la eficiencia. Dado que es necesaria una fase de interpretación extra, en la cual se deben evaluar todas las consecuencias de todas las declaraciones realizadas, el proceso es relativamente más lento que en la programación imperativa, en que los cambios de estado del sistema están dados por instrucciones particulares y no por un conjunto de condiciones arbitrariamente grande.

4.3. VENTAJAS DE LA PROGRAMACIÓN DECLARATIVA

A pesar de lo anterior existen algunas ventajas en el uso de la programación declarativa. Entre las ventajas se destaca que la solución de un problema se puede realizar con un nivel de abstracción considerablemente alto, sin entrar en detalles de implementación irrelevantes, lo que hace a las soluciones más fácil de entender por las personas. La resolución de problemas complejos es resuelta por el intérprete a partir de la declaración de las condiciones dadas.

La programación declarativa es muy usada en la resolución de problemas relacionados con inteligencia artificial, bases de datos, configuración, y comunicación entre procesos; sin embargo, ningún leguaje declarativo se aproxima en popularidad a los lenguajes imperativos.

4.4. EJEMPLOS DE LENGUAJES DECLARATIVOS

Algunos lenguajes declarativos que se pueden mencionar son:

- PROLOG
- SQL
- HTML
- WSDL (Web Services Description Language)
- XML Stylesheet Language for Transformation

4.5. PROGRAMACIÓN LÓGICA

La idea fundamental de la programación lógica consiste en emplear la lógica como lenguaje de programación.
La lógica no es imperativa porque no sirve para indicar cómo resolver un problema (órdenes). La lógica es declarativa porque sirve para especificar qué problema resolver (condiciones).

En la programación lógica, se especifican las condiciones que satisfacen las soluciones, se deducen las soluciones a partir de las condiciones y el énfasis de todo está en qué problema resolver. El problema se describe especificando qué caracteriza a sus posibles soluciones.

La programación lógica, junto con la funcional, forma parte de lo que se conoce como programación declarativa. En los lenguajes tradicionales, la programación consiste en indicar cómo resolver un problema mediante sentencias; en la programación lógica, se trabaja de forma descriptiva, estableciendo relaciones entre entidades, indicando no cómo, sino qué hacer. Se establece entonces que la idea esencial de la programación lógica es: algoritmos = lógica + control. Es decir, un algoritmo se construye especificando conocimiento en un lenguaje formal (lógica de primer orden), y el problema se resuelve mediante un mecanismo de inferencia (control) que actúa sobre aquél.

Al hacer un recorrido por la programación lógica, aparece como uno de sus lenguajes más representativos, Prolog, que es un clásico de la inteligencia artificial y que se aplica de múltiples formas en el desarrollo de software comercial.

4.6. PROGRAMACIÓN FUNCIONAL

La programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas. El objetivo de la programación funcional es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo (pues la programación funcional es declarativa), sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al permitir conceptos tomados de los lenguajes imperativos, como las secuencias de instrucciones o la asignación de variables. En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje híbrido.

4.7. PROGRAMACIÓN ORIENTADA A BASES DE DATOS

Las bases de datos son programas que administran información y hacen más ordenada la información, aparte de hacer la fácil de buscar y por supuesto de encontrar.

Las características de las bases de datos pueden ser ventajosas o desventajosas: pueden ayudar a almacenar, organizar, recuperar, comunicar y manejar información en formas que serían imposibles sin las computadoras, pero también afecta de alguna manera ya que existen enormes cantidades de información en bases de datos de las que no se tiene control del acceso.

Las bases de datos tienen muchos usos: facilitan el almacenamiento de grandes cantidades de información; permiten la recuperación rápida y flexible de información, con ellas se puede organizar y reorganizar la información, así como imprimirla o distribuirla en formas diversas.

Es claro que los lenguajes orientados a bases de datos son declarativos y no imperativos, pues el problema es "qué" se quiere hacer o "qué" se necesita buscar y encontrar en la base de datos, y no se enfatiza el "cómo".
Una base de datos también se puede definir como un banco de datos o conjunto de datos que pertenecen al mismo contexto, almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

En la actualidad, y gracias al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos.

Los sistemas gestores de bases de datos (SGBD) permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

4.7.1. TIPOS DE BASES DE DATOS

Las bases de datos pueden clasificarse de varias maneras, de acuerdo al criterio elegido para su clasificación, que puede ser según la variabilidad de los datos almacenados o según el contenido.

4.7.1.1 CLASIFICACIÓN DE BASES DE DATOS SEGÚN LA VARIABILIDAD DE LOS DATOS ALMACENADOS

Esta clasificación depende de cómo los datos cambian o varían dentro de la base de datos, o si se mantienen estáticos sin sufrir modificaciones o cambios. Dentro de esta clasificación se tiene lo siguiente:

BASES DE DATOS ESTÁTICAS: Éstas son bases de datos de sólo lectura, utilizadas primordialmente para almacenar datos históricos que posteriormente se pueden utilizar para estudiar el comportamiento de un conjunto de datos a través del tiempo, realizar proyecciones y tomar decisiones.

BASES DE DATOS DINÁMICAS: Éstas son bases de datos donde la información almacenada se modifica con el tiempo, permitiendo operaciones como actualización y adición de datos, además de las operaciones fundamentales de consulta. Un ejemplo de esto puede ser la base de datos utilizada en un sistema de información de una tienda de abarrotes, una farmacia, un videoclub, etc.

4.7.1.2. CLASIFICACIÓN DE BASES DE DATOS SEGÚN CONTENIDO

Esta clasificación basa su enfoque en el contenido o lo que almacena la base de datos, de esta manera:

BASES DE DATOS BIBLIOGRÁFICAS: Solo contienen un "representante" de la fuente primaria, que permite localizarla. Un registro típico de una base de datos bibliográfica contiene información sobre el autor, fecha de publicación, editorial, título, edición, de una determinada publicación, etc. Puede contener un resumen o extracto de la publicación original, pero nunca el texto completo, porque sino se estaría en presencia de una base de datos a texto completo (o de fuentes primarias). Como su nombre lo indica, el contenido son cifras o números. Por ejemplo, una colección de resultados de análisis de laboratorio, entre otras.

BASES DE DATOS DE TEXTO COMPLETO: Almacenan las fuentes primarias, como por ejemplo, todo el contenido de todas las ediciones de una colección de revistas científicas.

DIRECTORIOS: Un ejemplo son las guías telefónicas en formato electrónico.

BANCO DE IMÁGENES, AUDIO, MULTIMEDIA, ETC.

4.7.2. MODELOS DE BASES DE DATOS

Además de la clasificación por la función de las bases de datos, éstas también se pueden clasificar de acuerdo a su modelo de administración de datos.

Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.
Algunos modelos con frecuencia utilizados en las bases de datos son:

BASES DE DATOS JERÁRQUICAS: Éstas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información 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. Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

BASES DE DATOS DE RED: Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico). Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.

BASES DE DATOS RELACIONAL: Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información. El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales. Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización (reglas fundamentales para el buen funcionamiento de una base de datos relacional) de una base de datos. Durante los años '80 (1980-1989) la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no

utilizaba SQL como lenguaje base para su gestión. La normalización de una base de datos relacional consiste en las siguientes reglas:

1. **Primera Forma Normal (1FN)**: se eliminan los grupos repetitivos. Los atributos han de ser atómicos. Gráficamente las celdas de la tabla contienen solo un valor, en cada uno de los atributos sólo se puede incluir un dato, aunque sea compuesto, pero no se puede incluir una lista de datos. Se trata de que cada atributo guarde la menor cantidad de información posible. Para eliminar los grupos repetitivos se pone cada uno de ellos en una tabla aparte, esa nueva tabla hereda la clave primaria de la relación en la que se encontraban, se crea una clave foránea para la nueva tabla.
2. **Segunda Forma Normal (2FN):** dependencia completa. Está en 2FN si está en 1FN y si sus atributos no principales dependen de forma completa de la clave principal. Toda relación que tenga como clave sólo un atributo está en 2FN.
3. **Tercera Forma Normal (3FN):** dependencia transitiva. Está en segunda forma normal y todo atributo no primo es implicado por la clave primaria en una secuencia no transitiva.
4. Forma Normal de Boyce-Codd: una tabla está en FNBC, sí y sólo sí las únicas dependencias funcionales elementales son aquellas en las que la Clave primaria determina un atributo.
5. **Cuarta Forma Normal (4FN):** está en forma normal de Boyce-Codd y se eliminan las dependencias multivaluadas y se generan todas las relaciones externas con otras tablas u otras bases de datos.
6. **Quinta Forma Normal (5FN):** está en cuarta forma normal y toda dependencia-join viene implicada por claves candidatas.

BASES DE DATOS ORIENTADAS A OBJETOS: este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (estado y comportamiento). Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos: encapsulación (propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos), herencia (propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases y polimorfismo (propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos. En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos.

BASES DE DATOS DOCUMENTALES: permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes.

BASE DE DATOS DEDUCTIVAS: un sistema de base de datos deductivas, es un sistema de base de datos pero con la diferencia de que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. También las bases de datos deductivas son llamadas base de datos lógica, a raíz de que se basan en lógica matemática.

5. LENGUAJES DE PROGRAMACIÓN IMPERATIVOS

En ciencias de la computación se llama lenguajes imperativos a aquellos en los cuales se le ordena a la computadora cómo realizar una tarea siguiendo una serie de pasos o instrucciones, por ejemplo:
Paso 1, solicitar número.
Paso 2, multiplicar número por dos.
Paso 3, imprimir resultado de la operación.
Paso 4, etc,

El proceso anterior se puede realizar con un lenguaje imperativo como por ejemplo BASIC, C, C++, Java, Clipper, Dbase, C#, PHP, Perl, etc.

Dentro de la programación imperativa, se tiene un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

Los lenguajes imperativos se basan en comandos u órdenes que se le dan a la computadora para que haga algo, con el fin de organizar o cambiar valores en ciertas partes de la memoria.

La ejecución de estos comandos se realiza, en la mayor parte de ellos, secuencialmente, es decir, hasta que un comando no ha sido ejecutado no se lee el siguiente.

Según el dominio, o mejor dicho con el propósito que se utiliza el programa, se puede hablar de lenguajes de dominio específico y de dominio general.

5.1. LENGUAJES IMPERATIVOS PROCEDURALES

En los lenguajes tradicionales o procedurales, es la aplicación quien controla qué porciones de código se ejecuta, y la secuencia en que este se ejecuta. La ejecución de la aplicación se inicia con la primera línea de código, y sigue una ruta predefinida a través de la aplicación, llamando procedimientos según sea necesario.

Los lenguajes procedurales están fundamentados en la utilización de variables para almacenar valores y en la realización de operaciones con los datos almacenados. Algunos ejemplos son: FORTRAN, PASCAL, C, ADA, ALGOL,…

En este tipo de lenguajes, la arquitectura consta de una secuencia de celdas, llamadas memoria, en las cuales se pueden guardar en forma codificada, lo mismo datos que instrucciones; y de un procesador, el cual es capaz de ejecutar de manera secuencial una serie de operaciones, principalmente aritméticas y booleanas, llamadas comandos. En general, un lenguaje procedural ofrece al programador conceptos que se traducen de forma natural al modelo de la máquina. El programador tiene que traducir la solución abstracta del problema a términos muy primitivos, cercanos a la máquina.

Con un lenguaje procedural el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Estos lenguajes acceden a un registro, lo procesan y basándose en los resultados obtenidos, acceden a otro registro, que también deben procesar. Así se va accediendo a registros y se van procesando hasta que se obtienen los datos deseados. Las sentencias de un lenguaje procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual.

5.2. ALGUNOS LENGUAJES IMPERATIVOS

Algunos lenguajes de programación imperativos que se pueden mencionar son:

- BASIC
- C
- C++
- Java

- C#
- PHP
- Perl

6. DIFERENCIA ENTRE LENGUAJES DECLARATIVOS E IMPERATIVOS

En los lenguajes declarativos las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

Los lenguajes imperativos describen paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado un programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar un problema.

7. LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

En la Programación Orientada a Objetos (POO u OOP según siglas en inglés) se definen los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) comportamiento (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de la misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

Dada esta propiedad de conjunto de una clase de objetos, que al contar con una serie de atributos definitorios, requiere de unos métodos para poder tratarlos (lo que hace que ambos conceptos están íntimamente entrelazados), el programador debe pensar indistintamente en ambos términos, ya que no debe nunca separar o dar mayor importancia a los atributos a favor de los métodos, ni viceversa. Hacerlo puede llevar al programador a seguir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen esa información por otro (llegando a una programación estructurada camuflada en un lenguaje de programación orientada a objetos).

Esto difiere de los lenguajes imperativos tradicionales, en los que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. Los programadores de lenguajes imperativos escriben funciones y después les pasan los datos. Los programadores que emplean lenguajes orientados a objetos definen objetos con datos y métodos y después envían mensajes a los objetos diciendo que realicen esos métodos por sí mismos.

Un objeto se puede definir como un grupo de procedimientos que comparten un estado. Se define al conjunto de datos como "estado", y "métodos" como el conjunto de procedimientos que pueden alterar ese estado. Un

programa orientado a objetos es un método de implementación en el que los programas están organizados como colecciones de objetos, donde cada uno es una instancia de alguna clase, y donde todas las clases son miembros de una jerarquía de clases conectadas por relaciones de herencia. Este tipo de lenguajes son muy recientes en comparación a los primeros lenguajes de programación que aparecieron.

7.1. CONCEPTOS BÁSICOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos es una nueva forma de programar que trata de encontrar la solución a problemas de una forma que ofrece muchas ventajas y facilidades que no se tenían anteriormente. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entro ellos destacan los siguientes:

OBJETO: entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad ("métodos"). Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa).

CLASE: definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas. Una clase es una colección de objetos similares o la implementación, declaración o definición de un tipo de objeto. Cada vez que se construye un objeto de una clase se crea una instancia de esa clase. Por ejemplo en Visual Basic, se tiene la clase Form, y se pueden crear instancias de esa clase al tener Form1, Form2, etc. Así se está creando una instancia de la clase Form.

MÉTODO: algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en lar propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

EVENTO: un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente.

MENSAJE: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

PROPIEDAD O ATRIBUTO: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto, y cuyo valor puede ser alterado por la ejecución de algún método.

ESTADO INTERNO: es una propiedad invisible de los objetos, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos).

En comparación con un lenguaje imperativo, una "variable no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

7.2. CARACTERÍSTICAS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Las características más importantes de la programación orientada a objetos son las siguientes:

ABSTRACCIÓN: Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se

implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

ENCAPSULAMIENTO: también llamado "ocultación de la información". Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos. Esta característica o propiedad permite por tanto ejecutar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.

POLIMORFISMO: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Por ejemplo en Visual Basic, el polimorfismo se da al tener diferentes tipos de objetos (Form, Label, etc.)

HERENCIA: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto pertenece a más de una clase se dice que hay herencia múltiple; esta característica no está soportada por algunos lenguajes (como Java). Con esta propiedad, los objetos heredan comportamientos dentro de una jerarquía de clases.

7.3. PRINCIPALES DIFERENCIAS ENTRE LA PROGRAMACIÓN IMPERATIVA Y LA PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos es más moderna, es una evolución de la programación imperativa plasmada en el diseño de una familia de lenguajes conceptos que existían previamente, con algunos nuevos.

La programación orientada a objetos se basa en lenguajes que soportan sintáctica y semánticamente la unión entre los tipos abstractos de datos y sus operaciones (a esta unión se la suele llamar clase).

La programación orientada a objetos incorpora en su entorno de ejecución mecanismos tales como el polimorfismo y el envío de mensajes entre objetos.

7.4. ALGUNOS LENGUAJES ORIENTADOS A OBJETOS

Entre los lenguajes orientados a objetos más importantes que se pueden mencionar, aparecen los siguientes:

- Ada
- C++

- C#
- VB.NET
- Clarion
- Delphi
- Eiffel
- Java
- Lexico (en castellano)
- Objective-C
- Ocaml
- Oz
- PHP
- PowerBuilder
- Pitón
- Ruby
- Smalltalk

CONCLUSIÓN

Los lenguajes de programación no son simplemente un detalle más del amplio mundo de la informática, y por lo tanto deben ser vistos como el fundamento y la base del desarrollo y avance de la computación.

Estudiar los conceptos básicos, clasificación, diferencias, propiedades y funcionamiento de los lenguajes de programación es elemental para cualquier estudiante o profesional dedicado a la computación, pues de esa manera se logra tener una perspectiva global y mucho más amplia que tendrá mucho peso al estar bien documentados y al conocer qué son, cómo se clasifican y de qué manera trabajan los lenguajes de programación. Luego del desarrollo de esta investigación resulta fácil comprender los tipos y la clasificación que se les da a los lenguajes de programación en base a sus funcionalidades y características.

Este estudio también permite desarrollar un sentido crítico de los lenguajes de programación, de forma que el programador no seleccione ni emita un juicio respecto a determinado lenguaje basado simplemente en su limitado conocimiento ni basado en la popularidad de la que goza cierto lenguaje, sino que el programador esté capacitado para dar razones contundentes y certeras del por qué un lenguaje es mejor que otro para determinada tarea, qué ventajas tiene uno respecto del otro, y que así también el programador sea capaz de seleccionar el lenguaje que más le convenga para la resolución de un problema determinado.

Esta investigación servirá para evaluar correctamente los lenguajes de programación, determinando las ventajas y desventajas que cada uno de ellos presenta.

REFERENCIAS BIBLIOGRÁFICAS

- Cejas, C.B; Crespillo, O.G.; Jiménez F., M.J.; Ramírez G., C.; Sánchez G., C.; Sánchez N., C. Tipos de Lenguajes de Programación. Extraído el 29 de agosto, 2006 de
- Wikipedia. La enciclopedia libre. Programación declarativa. Extraído el 29 de agosto, 2006 de http://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa
- Concept&Development. Programación declarativa. Extraído el 29 de agosto, 2006 de http://pviojo.net/posts/programacion-declarativa

- Sanders, Donald H.; Informática Presente y Futuro. Tercera Edición. McGrawHill; Naucalpán de Juárez, México; 1990.
- Wikipedia. La enciclopedia libre. Programación imperativa. Extraído el 29 de agosto, 2006 de http://es.wikipedia.org/wiki/Lenguajes_imperativos
- Universidad Tecnológica Nacional. Facultad Regional Tucuman. Lenguajes de Programación. Extraído el 29 de agosto, 2006 de http://www.frt.utn.edu.ar/sistemas/paradigmas/lenguajes.htm
- Matta G., D.A. Tutorial Introducción al Desarrollo de Aplicaciones con Visual Basic.
- Wikipedia. La enciclopedia libre. Programación orientada a objetos. Extraído el 30 de agosto, 2006 de http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos#La_Programaci.C3.B3n_Orientada_a_Objetos_.28POO.29_como_soluci.C3.B3n
- Introducción a la Programación Lógica. Ingeniería Técnica en Informática de Sistemas. Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga. Extraído el 30 de agosto, 2006 de http://72.14.209.104/search?q=cache:doHkGvx7wQIJ:www.lcc.uma.es/~lopez/apuntes/declasis/apuntes/introduccion/intro_sis4pp.pdf+programacion+declarativa+logica&hl=es&gl=sv&ct=clnk&cd=6
- Rossel, G. Programación lógica. Extraído el 30 de agosto, 2006 de http://www.amzi.com/articles/code07_whitepaper.pdf#search=%22programacion%20%20logica%22
- Wikipedia. La enciclopedia libre. Programación funcional. Extraído el 30 de agosto, 2006 de http://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional
- Wanadoo. El Rincón del Vago. Base de Datos. Extraído el 30 de agosto, 2006 de http://html.rincondelvago.com/concepto-de-base-de-datos.html
- Wikipedia. La enciclopedia libre. Base de datos. Extraído el 30 de agosto, 2006 de http://es.wikipedia.org/wiki/Base_de_datos#Tipos_de_bases_de_datos
- Wikipedia. La enciclopedia libre. Normalización de una base de datos. Extraído el 30 de agosto, 2006 de http://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_una_base_de_datos

Autores:
Edilberto Abdulio Baños Martínez.
Jennifer Esmeralda Chacón Carranza.
José Amilcar Chigüén Chegüén
Glenda Maritza España Canalez.
Jaime Oswaldo Montoya Guzmán