# TEMA 55

# DISEÑO FÍSICO DE DATOS Y FUNCIONES. CRITERIOS DE DISEÑO. DOCUMENTACIÓN

**ÍNDICE**

## 1. INTRODUCCIÓN.

Durante la fase de diseño físico se produce la creación efectiva del software mediante la traducción de las especificaciones de las fases de análisis a una forma que pueda ser entendida y ejecutada por el ordenador.
Para ello utilizando uno o más lenguajes de programación, se crea código fuente que más tarde será traducido, de una forma u otra, a código máquina. Asimismo, el Modelo Lógico de Datos va a dar paso a la implementación física de la estructura de datos en un sistema computacional concreto.

## 2. DISEÑO FÍSICO DE FUNCIONES Y SU DOCUMENTACIÓN.

Una vez efectuada una planificación de la fase de codificación, que se llevará a cabo teniendo en cuenta, la complejidad de los problemas a resolver, la potencia de las herramientas disponibles y la cualificación del personal encargado, se procederá a la elaboración del código fuente.

### 2.1. Módulo y estructura.

Es necesario indicar que los adjetivos modular y estructurado se suelen aplicar a los conocidos como lenguajes de tercera y cuarta generación y al código fuente creado por ellos.

Así pues, incluimos aquí la codificación en lenguajes tales como:

COBOL estructurado
PASCAL
C
ADA
4GL de INFORMIX
ORACLE……

#### *2.1.1 Módulo*

Un módulo es un componente de un programa con significado propio y que a menudo puede compilarse por separado, es decir, un subprograma.
Dependiendo del lenguaje de programación, un subprograma puede denominarse subrutina, procedimiento, función u otro nombre técnico.

Un subprograma suele tener una serie de características genéricas;

Una **sección de especificación** que incluye el nombre del subprograma y la descripción de la interfaz de éste con el resto de la aplicación

Una **sección de implementación** que incluye los datos (atención a su ámbito de validez y visibilidad) y las estructuras de control.

Un **mecanismo de activación** que permite que el subprograma sea llamado desde cualquier otro punto del programa, se ejecute y devuelva el control (normalmente a la instrucción siguiente a la de la llamada ). Este mecanismo suele consistir en invocar el nombre del módulo acompañado de los parámetros a utilizar por el mismo.

Utilizando pues una construcción a base de módulos, un programa se convierte en una superestructura modular compuesta por una serie de subprogramas que se van llamando unos a otros.

#### *2.1.2. Estructuras de control.*

Un programa estructurado es aquel que evita código cruzado y utiliza exclusivamente las estructuras de control básocas: secuencial, condicional y repetitiva en cualquiera de sus variantes.

Existen otras estructuras de control, como la **recursividad**, que permite que un subprograma se active a sí mimos como parte de su propia ejecución.

La **concurrencia** que permite la ejecucuón paralela (por tiempo compartido, o por multiproceso) de múltiples tareas, su sincronización y comunicación entre sí.

El **manejo de excepciones** que permite atrapar los errores del sistema o ciertas condiciones definidas previamente, pasando el control a subprogramas manejadores de excepciones ….

## 2.2. EL DISEÑO ARRIBA-ABAJO (TOP-DOWN) Y LA CODIFICACIÓN ABAJO-ARRIBA (BOTTOM-UP).

### *2.2.1 El diseño TOP-DOWN.*

En la fase de análisis de un programa se suelen aplicar las técnicas de arriba-abajo (top-down) consistentes en ir estudiando el problema en sucesivos niveles de menor a mayor grado de detalle.

En el nivel de mínimo detalle toda la aplicación se presenta como una **caja negra** con unas entradas y unas salidas, de la cual sabemos que hace algo pero sin analizar el cómo. .

A continuación , se baja un escalón hacia un mayor nivel de detalle, cambia el punto de vista y, lo que antes aparecía como una caja negra ( proceso único en un diagrama de contexto) se convierte en un proceso de nivel mayor que 0 en un DFD, en una primitiva funcional y, más tarde, en una unidad de programación o en un subprograma que a su vez puede contener otros subprogramas que se presentan como cajas negras y que dejarán de serlo en futuros descensos hacia mayores niveles de detalle.

El diseño Top-down también se conoce como técnica de refinamiento sucesivo y como hmeos comentado se utiliza en la fase de diseño del software.

### *2.2.2 Codificación BOTTOM.UP*

Una vez que se ha efectuado el estudio de una aplicación, es necesario pasar a su codificación. Para ello el programador irá traduciendo las especificaciones del análisis a código fuente. La cuestión es por donde empezar.

Si la aplicación comienza a codificarse a parir del menor nivel de detalle, no será posible efectuar pruebas hasta su completa terminación, puesto que mientras que los módulos de máximo detalle no estén terminados no podrá ejecutarse.

## 2.3. EL ESTILO DE CODIFICACIÓN DEL SOFTWARE.

Una vez que se ha generado el código fuente, lo que el módulo realiza ha de poder ser comprendido sin necesidad de consultar la documentación del análisis.

### *2.3.1. Documentación dentro del código.*

La documentación dentro del código es de importancia capital pues es la mejor garantía de unos programas legibles.

Hay que cuidar la elección de los nombres de los identificadores de forma que sean suficientemente explicativos sin alcanzar tamaños enormes que dificulten su utilización.

Otro elemento de documentación es la posibilidad de incluir comentarios en lenguaje natural como parte del código fuente..

Podemos distinguir entre comentarios de prólogo y los comentarios de descriptivos . Los primeros se deben incluir en el inicio de cada subprograma y entre otros dato s pueden incluir:

el nombre del subprograma
su propósito
una descripción del interfaz que incluya
un comentario sobre su funcionamiento, limitaciones, variables principales utilizadas etc.
Un resumen de la historia de su desarrollo que incluya el autor, o autores y los modificadores posteriores junto con la fecha en que cada uno efectuó su trabajo.

Un último elemento que afecta a la documentación interna de un programa es la forma en que aparece el listado del mismo. La sangría del código puede realzar las estructuras de control utilizadas por el programador ayudando a su mejor comprensión.

## 2.4. LAS PRUEBAS . Y LA DEPURACIÓN DE ERRORES.

### *2.4.1. La depuración de errores en la codificación.*

El proceso de compilación constituye un primer filtro que va a detectar todos aquellos errores sintácticos que se hayan introducido durante la codificación de un módulo.

Cuanto más avanzado es un compilador, mejor es su capacidad de detectar errores de sintaxis y más explicativos son los mensajes de aviso que en estos casos se producen.

Además de errores es muy posible que surjan anomalías en tiempo de ejecución. Para depurarlas son muy útiles herramientas tales como trazadores y analizadores de ejecución, que permiten ejecutar el programa paso a paso mientras se va viendo el código fuente visualizar y cambiar valores a las variables, conocer el estatus de los canales de comunicación abiertos .. situar puntos de interrupción fijos y condicionales.. etc, En resumen, suponen una gran ayuda en el proceso de depuración de anomalías del código fuente.

### *2.4.2. Las pruebas del software.*

La prueba del software es un elemento crítico para obtener la garantía de que las aplicaciones cumplen con los niveles de calidad requeridos.

Esta fase supone una última oportunidad de verificar las especificaciones del diseño y las codificaciones de un programa.

## 3. DISEÑO FISICO DE DATOS Y SU DOCUMENTACIÓN.

### 3.1. Dependencia del producto comercial específico.

La última etapa del proceso de diseño de datos es la que atañe a la organización física de los mismos, en la cual, teniendo presentes requisitos de los procesos del SGBD, del Sistema operativo y del hardware, se pretenden entre otros los siguientes objetivos:

Disminuir los tiempos de respuesta.
Minimizar el espacio de almacenamiento
Evitar las reorganizaciones costosas en tiempo y recursos.
Proporcionar la máxima seguridad

En definitiva, cumplir los objetivos del sistema y conseguir la optimización de la ratio coste/beneficio.

Desafortunadamente no existe un modelo formal para el diseño físico, si bien se están llevando a cabo en este terreno algunos trabajos de investigación. El diseño físico resulta, por tanto y hasta ahora, muy dependiente del producto comercial concreto que se escoja para la implementación de la aplicación.

### 3.2. Consideraciones generales sobre el diseño físico de datos.

El problema del diseño físico para el administrador de la Base de Datos consiste en proveer un conjunto eficiente de estructuras de acceso de modo que el optimizador pueda tomar las mejores decisiones.

Entre los instrumentos más importantes y generales del diseño físico se encuentra la selección de los índices secundarios, que es uno de los problemas clave en la instrumentación física de una base de datos.

Una vez diseñadas las aplicaciones se conocerá cuales son las consultas más frecuentes y/o prioritarias a la base de datos, por lo que será conveniente crear un índice secundario que ayude a localizar las filas seleccionadas en dichas consultas y reducir así los accesos a disco.

Existen otros elementos importantes en el diseño físico, aunque no todos los sistemas comerciales disponen de ellos. Algunos de estos elementos son:
Características de los registros físicos.

- Asignación de dirección calculada (hashing)
- Agrupamiento de registros
- Bloqueo y comprensión de datos.
- Asignación de memoria intermedias (buffering)
- Asignación de conjuntos de datos a particiones y a disposiciones físicos..

En general, existen tres estrategias de los fabricantes en cuanto al diseño:
El SGB impone una estructura interna, dejándole al diseñador muy poca flexibilidad, lo que suele aumentar la independencia físico/lógica, pero disminuye la eficiencia.

El administrador diseña la estructura interna, lo que en general, supone una importante carga para el administrador y puede influir negativamente en la independencia, aunque puede mejorar la eficiencia.
El SGBD proporciona una estructura interna opcional que el diseñador puede cambiar a fin de optimizar el rendimiento de la base de datos.

## 4. CONCLUSIÓN.

La fase de diseño constituye la última fase previa a la implementación física de la aplicación en un sistema computacional concreto.

Recoge pues todas las orientaciones y trabajos previos de las fases de análisis y diseño lógico, suponiendo la confirmación o refutación de su validez y dividiéndose también en diseño físico funcional y diseño físico de la estructura externa de datos.

## 5 BIBLIOGRAFIA

Alcalde ; Ormaechea: “Arquitectura de ordenadores” ed. McGraw-Hill 1991.
Nelson V.P.”Análisis y Diseño de Circuitos Lógicos Digitales “ Prentice Hall 1996