# Unidad Didáctica IX

## Teoría de la normalización

*CASO:*

*El proyecto de la empresa **SINUNEURO** (**S**ervicios **IN**mobiliarios de la **UN**ión **EURO**pea) está definido para la venta y alquiler de locales, solares y viviendas. **Julia** y **María** han realizado un proceso de abstracción con las directrices marcadas por la empresa en cuanto a sus necesidades y han conseguido concretar las tareas de desarrollo. Básicamente podemos resumirlo diciendo que el pilar principal de todo el proyecto es un sistema de bases de datos relacional, y en este momento se encuentran ante la necesidad de definir las tablas y ficheros que debe utilizar dicho sistema.*

***María** explica a **Julia** que para conseguir las tablas, es preciso realizar un estudio detallado mediante mecanismos como el **modelo Entidad-Relación** mediante el cual serán identificados los actores y su forma de actuar en el proyecto. De este modo se consigue facilitar la obtención de los diferentes ficheros y tablas de nuestro sistema relacional de Bases de Datos.*

*Evidentemente **Julia** es la primera vez que escucha estos términos y tiene muchas dudas que **María** le irá resolviendo sobre la marcha, con tranquilidad y ejemplos claros. Uno de los principales conceptos que van a tratar durante el desarrollo de la aplicación, será la **normalización de las tablas**, María explica que esto no es más que conseguir que las tablas cumplan una serie de normas que posteriormente van a facilitar su utilización por parte de todos los usuarios, empezando por el equipo de desarrollo y terminando por cualquier usuario que haga la consulta más sencilla en nuestra base de datos.*

*El proceso de normalización nos va a permitir, principalmente en las tareas de diseño, evitar problemas posteriores y de alguna manera, allanar el camino que nos queda por recorrer. Las tablas que cumplen las formas normales son más sencillas y se adaptan mejor a los modelos matemáticos.*



# Unidad Didáctica IX

## Necesidad de un método formal de diseño relacional

Cuando surge un determinado software o aplicación informática, al principio no es más que una lista de requisitos o puntos definidos como un conjunto de **necesidades** que el cliente plantea al analista. Como hemos visto en las unidades anteriores estos **requerimientos** deben ser establecidos formalmente en términos de objetivos o hitos que se pretenden conseguir con unos costes determinados.

La mayoría de proyectos desembocan en la necesaria utilización de **bases de datos** y eso implica un diseño detallado de los datos a utilizar, definidos en términos específicos sin ambigüedad y mediante un completo diccionario de datos.

Uno de los objetivos principales de las aplicaciones informáticas y más concretamente de los sistemas de gestión de bases de datos (SGDB), es proporcionar a los usuarios una **visión abstracta de los datos**, es decir, el usuario va a utilizar esos datos pero no es necesario que conozca cómo están almacenados

físicamente.

Los modelos de datos son el instrumento principal para obtener estos, y son utilizados para la representación y el tratamiento de los problemas. En la unidad anterior hemos visto cómo debe ser llevado a cabo el **diseño conceptual**, obteniendo el diagrama **E/R** para transformarlo después en un esquema relacional en el que serán definidas cada una de las tablas con sus campos (o columnas) correspondientes, uno de los cuales será la clave principal que garantiza la unicidad de los registros. Esas tablas que definen el esquema relacional, pueden presentar ciertas anomalías, como la existencia de redundancia, incoherencia y problemas para mantener la **integridad** de los datos, por ello las tablas resultantes del esquema relacional deben seguir determinadas pautas que eviten esas anomalías y garanticen un resultado adecuado a las necesidades planteadas inicialmente por el cliente.

Para conseguir esto se aplica la **Normalización**. **La teoría de la Normalización (o simplemente la normalización) consiste en aplicar una serie de reglas para asegurarnos la eliminación de redundancias e inconsistencias en las tablas**. En ocasiones esta Normalización se traduce en la separación de los datos en diferentes tablas asegurándonos siempre de que las tablas resultantes mantienen toda la información original y las distintas dependencias entre los datos almacenados.

**Animación sobre la obtención del modelo relacional tras la normalización**

Podemos resumir diciendo que los principales **objetivos** que busca un diseño normalizado son los siguientes:

- Eliminar anomalías de actualización, inserción y borrado.
- Conservar la información original (descomposición sin pérdida de información).
- Conservar las dependencias funcionales originales (descomposición sin pérdida de dependencias funcionales).
- No crear dependencias nuevas o relaciones inexistentes.
- Facilidad de uso y modificación de tablas creadas.
- Eficiencia.

La teoría de normalización se basa en el concepto de **dependencias**, que son una serie de propiedades asociadas al contenido semántico de los datos. No es posible demostrar una dependencia, pero se puede afirmar con la observación del comportamiento de los datos en el mundo real.

***Para saber más***

***El Modelo Entidad Relación es un mecanismo que nos permite abordar los proyectos software con garantías de éxito al hacer un tratamiento específico de los datos a utilizar. En este enlace puedes leer los conceptos básicos de este mecanismo y tener otro punto de vista respecto al que te hemos dado en los apartados anteriores.***

***Modelo Entidad/Relación***

***http://es.wikipedia.org/wiki/Diagrama_entidad-relaci%C3%B3n*** **[versión en cache]**

***El principal exponente del diseño conceptual de bases de datos es el Modelo Entidad Relación. El siguiente enlace es una especie de curso que te inicia en estos conceptos.***

***Diseño Conceptual de Bases de Datos***

***http://www3.uji.es/~mmarques/f47/apun/node79.html***



# Unidad Didáctica IX

## Dependencias funcionales

Hasta ahora hemos llegado a la conclusión de que una **tabla** (relación) está formada por un conjunto de **atributos**, pero cada uno de los valores que pueden tomar esos atributos, tendrá un significado que va a condicionar su comportamiento en una aplicación, y los resultados que se obtengan. Por ello podemos decir que **una tabla (o relación) se compone de un conjunto de atributos y sus dependencias, R (A, DF).** Los atributos (A) sabemos cómo identificarlos pero para las dependencias (DF), tendremos que ahondar en el valor semántico de esos atributos y no pueden obtenerse de manera automática en una tabla determinada.

Resumiendo, las **dependencias funcionales reflejan enlaces semánticos permanentes entre los datos en un diseño concreto** que podemos concretar en:

- **Restricciones de integridad** establecidas por el usuario que permiten conocer las relaciones que existen entre los atributos del problema planteado.
- **Propiedades inherentes** a la semántica del problema.
- **Y que se han de cumplir para todos los registros de una relación**.

Consideramos un esquema de relación general **R(A)**, con A el conjunto de sus atributos, y consideramos X Y, y Z subconjuntos de A llamados **descriptores**.
Nos encontramos con que podemos definir los siguientes tipos de dependencias:

### Dependencia funcional

Se dice que un conjunto de atributos (**Y**) **depende funcionalmente** de otro conjunto de atributos (**X**), si para cada valor de **X** existe un único valor posible para **Y**. Se representa por **X(Y**. También se dice que **X determina a Yo** que **X implica a Y**.

- **X** se conoce como **determinante** o **implicante** e
- **Y** como **implicado**.

***Ejemplo. Si consideramos la tabla (o relación) EMPLEADO (DNI,NOMBRE, DIRECCION, TELEFONO, DEPARTAMENTO) es evidente que el nombre de una persona depende funcionalmente del DNI, es decir, para un DNI determinado existe sólo un nombre que le corresponda. Sin embargo, si consideramos el ejemplo al revés, DNI no depende funcionalmente de NOMBRE, ya que para un mismo nombre puede haber muchos DNI diferentes.***

***¡Piensa en la cantidad de personas con el mismo nombre y apellidos que hay en este país... y cada uno tiene su DNI, que lo identifica!***

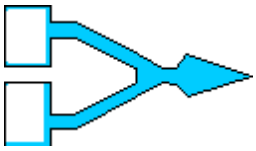
***En este caso, si suponemos que cada persona tiene un único teléfono y una única dirección, y que además trabaja en un único departamento de la empresa, las dependencias funcionales quedan como siguen:***


***No obstante, si en nuestro problema necesitáramos almacenar (o quisiéramos permitir que se almacenara) más de una dirección para una misma persona, o más de un teléfono, o que trabajara en más de un departamento, esos tres atributos no dependerían funcionalmente de DNI. A eso es a lo que nos referimos cuando decimos que las dependencias funcionales vienen definidas por la semántica del problema.***



# Unidad Didáctica IX

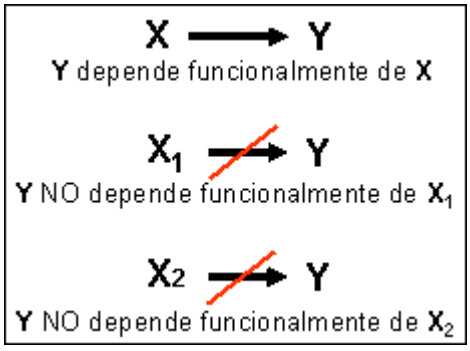
## Dependencia funcional completa



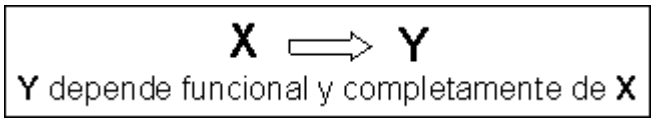
Si el descriptor **X** es compuesto, **X($X_1$, $X_2$),** se dice que **Y tienedependencia funcional completa o plena de X**, si depende funcionalmente de **X**, pero no depende funcionalmente de ningún subconjunto de **X**, es decir:

**Y** depende funcionalmente de **X**, pero **Y** no depende funcionalmente de $X_1$, ni tampoco de $X_2$.

Se representa:




Una dependencia funcional completa se representa como...

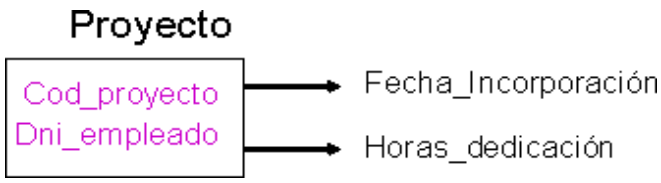



Si consideramos la relación:

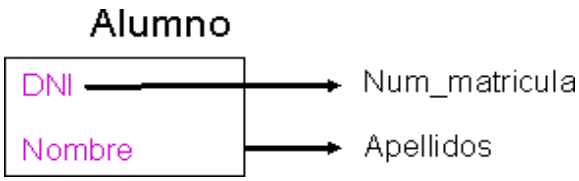
**PROYECTO (COD_PROYECTO, DNI_EMPLEADO, FECHA_INCORPORACION, HORAS_DEDICADAS)**

...está claro que los atributos **FECHA_INCORPORACION** y **HORAS_DEDICADAS** tienen dependencia funcional completa respecto de **COD_PROYECTO** y **DNI_EMPLEADO**. Es evidente que la **FECHA_INCORPORACION** a un proyecto de un empleado depende tanto del proyecto al que se incorpora como del empleado que se incorpora. Un mismo empleado puede pertenecer a varios proyectos a los que se incorpora en diferentes fechas. Igual ocurre con el atributo **HORAS_DEDICADAS.**

La dependencia funcional completa se puede representar mediante un **diagrama de dependenciasfuncionales** como sigue:




En cambio, si consideramos la relación **ALUMNO (DNI, NOMBRE, APELLIDOS, NUM_MATRICULA)**, e conjunto de atributos formado por el **NOMBRE** y el **DNI** producen dependencia funcional sobre el atributo **APELLIDOS**, y si consideramos que el **NUM_MATRICULA** depende únicamente del atributo **DNI** podemos representarlo como sigue:



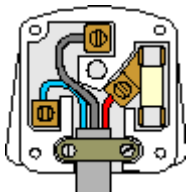

Pero la dependencia de los atributos **DNI** y **NOMBRE** no es completa ya que **DNI** sólo también produce dependencia funcional sobre **APELLIDOS**. De esta manera sólo el atributo **DNI** produce una dependencia funcional completa sobre el campo **APELLIDOS**. Es más, **DNI** produce una dependencia funcional completa sobre el resto de los atributos. Por lo que la dependencia real de la relación **ALUMNO** queda:

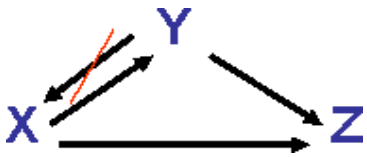



## Unidad Didáctica IX

### Dependencia funcional transitiva

Consideramos la relación **R(X, Y, Z)** con las siguientes dependencias funcionales:





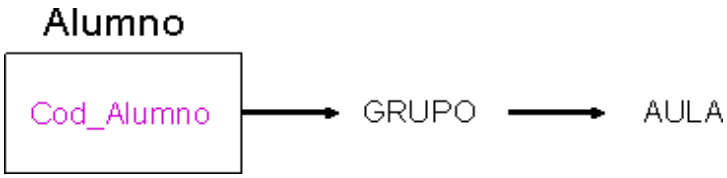

Es decir:

- **Z** depende funcionalmente de **X** y de **Y**,
- **Y** depende funcionalmente de **X**, pero
- **X** no depende funcionalmente de **Y**.

Se nota de la siguiente manera:


En estas condiciones se dice que **Z** tiene una **dependencia funcional transitiva** respecto a **X**, a través de **Y**. Es decir, un descriptor **Z** es transitivamente dependiente de otro **X** si se puede conoce por diferentes vías, una directamente y otra a partir de otro descriptor intermedio **Y**.

Si consideramos la siguiente relación **ALUMNO (COD_ALUMNO, GRUPO, AULA)** claramente se aprecia que los atributos **GRUPO** y **AULA** dependen funcionalmente del atributo **COD_ALUMNO**, y a su vez el atributo **AULA** depende funcionalmente del atributo **GRUPO**, por lo que **AULA** tiene una dependencia funcional transitiva respecto a **COD_ALUMNO** a través de **GRUPO**.






# Unidad Didáctica IX

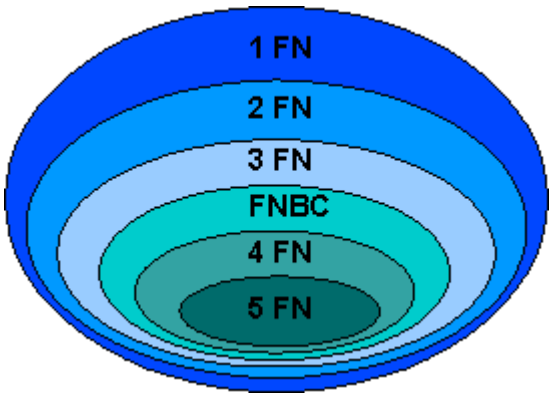
### Teoría formal de Normalización

La **teoría de la normalización** es una técnica bastante difundida en las metodologías de desarrollo de sistemas de Información que suele emplearse tanto a nivel **conceptual** como a nivel **lógico** (en el diseño de BD relacionadas e incluso en algunos lenguajes de programación).

Esta técnica ayuda a los diseñadores a prevenir **problemas de redundancia** y consiste en ir descomponiendo los registros en otros con menos campos, con el fin de satisfacer alguna "forma normal" basadas en los conceptos de dependencias funcionales.

Esta teoría se basa en la aplicación de las **Formas Normales, que son un conjunto de restricciones sobre las tablas que permiten evitar los problemas de redundancia y**

**anomalías en la actualización, inserción y borrado de datos.** De esta manera transformamos, si es necesario, las tablas obtenidas directamente del modelo Entidad Relación, en un conjunto de tablas más simples y fáciles de mantener.




Hasta ahora hemos definido **los conceptos** necesarios (dependencias) para poder entender las distintas formas normales, pero de aquí en adelante vamos a conocer **los mecanismos** que nos van a permitir transformar las relaciones en otras normalizadas y equivalentes.

La normalización hasta la 4FN (**cuarta forma normal**) rara vez se produce, ya que sólo hace falta en sistemas muy concretos e inusuales. En algunos libros se habla incluso de 6FN, 7FN y 8FN, pero dichas formas normales se salen de los objetivos de esta unidad, y del módulo profesional.



## Unidad Didáctica IX

### 1FN (Primera Forma Normal)

> ***Una relación está en primera forma normal (1FN) si y sólo si todos sus atributos son atómicos.***

La siguiente relación no estaría en primera forma normal:


| DNI | DIRECCION | TELEFONO |
|---|---|---|
| 76757473 | Granada, 21 | 950343536 |

| | | |
|---|---|---|
| | | 667667667 |
| | | 950505050 |
| **24252627** | **Islas Vírgenes, 20** | 950343434 |
| | | 696789210 |

El atributo **TELEFONO** no es atómico, ya que para cada algunas tuplas tiene más de un valor. ¿Cómo podríamos solucionar el problema? De manera intuitiva surgen dos posibles soluciones:

- Creando registros **nuevos** de la siguiente manera:

| DNI | DIRECCION | TELEFONO |
|---|---|---|
| **76757473** | **Granada, 21** | **950343536** |
| **76757473** | **Granada, 21** | **667667667** |
| **24252627** | **Islas Vírgenes, 20** | **950505050** |
| **24252627** | **Islas Vírgenes, 20** | **950343434** |
| **24252627** | **Islas Vírgenes, 20** | **696789210** |

- Si sabemos el número exacto de distintos teléfonos que puede tener cada registro de la tabla podemos añadir tantos atributos como teléfonos se necesiten.

| DNI | DIRECCION | TELEFONO_TRABAJO | TELEFONO_MOVIL | TELEFONO_CASA |
|---|---|---|---|---|
| **76757473** | **Granada, 21** | **950343536** | **667667667** | |
| **24252627** | **Islas Vírgenes, 20** | **950343434** | **696789210** | **950505050** |

En ambos casos las tablas resultantes sí están en primera forma normal.



# Unidad Didáctica IX

**2FN (Segunda Forma Normal)**

> ***Una relación está en segunda forma normal (2FN) si y sólo si está en 1FN y todos los atributos no clave dependen por completo de la clave primaria.***

Consideramos la siguiente relación:


**ALUMNO (DNI, COD_ASIGNATURA, NOMBRE, APELLIDOS, NOTA, CURSO, AULA)**

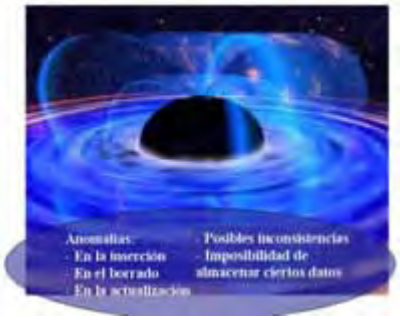
donde se aprecian las siguientes dependencias funcionales:

**DNI ( NOMBRE, APELLIDOS**

**DNI, COD_ASIGNATURA ( NOTA**

**COD_ASIGNATURA ( CURSO, AULA**

Dicha relación **NO se encuentra en 2FN** al estar los atributos **NOMBRE** y **APELLIDOS** determinados sólo por el atributo **DNI** y no por la totalidad de la clave principal. Igual ocurre con los atributos **CURSO** y **AULA** que están determinados sólo por **COD_ASIGNATURA**. Este hecho presenta una serie de inconvenientes tales como:




- **Redundancia de datos**: Con cada alumno de una misma asignatura repetimos toda la información acerca de la asignatura.
- **Anomalías en la inserción de tuplas**: Tenemos anomalías de inserción ya que no se pueden insertar las asignaturas que se imparten en un curso hasta que no insertamos algún alumno matriculado en esas asignaturas. En caso contrario, estaríamos violando la regla de integridad de la clave al existir valores nulos en dicha clave.
- **Anomalías en el borrado de tuplas:** Igualmente aparecen anomalías en el borrado de tuplas, ya que al eliminar todos los alumnos de una determinada asignatura queda eliminada de manera automática dicha asignatura.
- **Anomalías en la actualización de tuplas:** Si se hace un cambio de aula para una asignatura, es necesario modificar todas las tuplas de todos los alumnos matriculados en esa asignatura. Volvemos a ver que existe una gran redundancia de información en la Base de Datos.
- **Posible inconsistencia de datos en las actualizaciones:** No es más que una consecuencia de las anomalías en la actualización de tuplas. Imagina que se te pasa modificar todas las tuplas de todos los alumnos matriculados en una asignatura cuando se producen cambios en la asignatura (por ejemplo, cambia el aula donde se imparte la asignatura). Alumnos distintos podrían tener datos distintos para una misma asignatura, por ejemplo, distintas aulas donde se imparte, cuando realmente se imparte en una única aula. Eso sería una inconsistencia.
- **Imposibilidad de almacenar ciertos datos.** No es más que una consecuencia de las anomalías en la inserción.No nos resultaría posible representar datos sobre asignaturas en las que no haya matriculado ningún alumno, o sobre aulas en las que no se imparte ninguna asignatura, cuando es perfectamente posible que queramos almacenar esa información.

Pero ¿qué se hace cuando una relación **R** no está en 2FN?

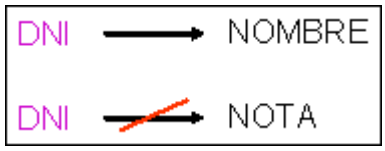
Se transforma para que lo esté, y esta transformación se realiza de la siguiente manera:

> *Dada una Relación **R** con atributos **X, Y, Z** simples o compuestos, con la clave **(X, Y)** si en la relación existe una dependencia funcional incompleta **Y( Z** (**Z** depende de parte de la clave), entonces:*
>
> ***De R se elimina Z.***
>
> ***Se construye una nueva relación R' con:***
>
> ***Z como atributo no clave**.*
>
> ***Y como clave primaria de R'.***

Las relaciones **R** y **R'** resultantes sí están ahora en segunda forma normal (2FN).

Veamos esto con más claridad con ayuda de un **ejemplo**. Consideramos la siguiente relación y comprobamos si está en 2FN:

**ALUMNO (DNI, COD_ASIGNATURA, NOMBRE, APELLIDOS, NOTA, CURSO, AULA)**




Lo primero que tenemos que comprobar es que dicha relación está en primera forma normal. Como sólo tenemos la intención de la relación, damos por supuesto que todos los valores de los atributos son atómicos. Ahora hay que comprobar la segunda parte de la definición de segunda forma normal, es decir que todos los atributos no clave dependen por completo de la clave primaria.

Es decir, se tienen que verificar las siguientes dependencias:

**DNI, COD_ASIGNATURA ( NOMBRE**

**DNI, COD_ASIGNATURA ( APELLIDOS**

**DNI, COD_ASIGNATURA ( NOTA**

**DNI, COD_ASIGNATURA ( CURSO**

**DNI, COD_ASIGNATURA ( AULA**

Pero es evidente que los atributos **NOMBRE** y **APELLIDOS** no dependen por completo de la clave, ya que sólo dependen de **DNI**, y no de **COD_ASIGNATURA**.


***¿Cómo solucionamos esta situación?***

Sencillamente **descomponiendo** la relación dada en otras relaciones que sí estén en segunda forma normal, separando los atributos que no tienen dependencia funcional completa respecto de la clave principal en otra tabla, junto con el/los atributo/s que forman la clave principal con los que sí tienen dependencia funcional completa.

En nuestro caso concreto, la descomposición sería la siguiente:

**ALUMNO** (DNI, NOMBRE, APELLIDOS)

**CALIFICACION** (DNI, COD_ASIGNATURA, NOTA)

**ASIGNATURA** (COD_ASIGNATURA, CURSO, AULA)

Como podemos comprobar, ahora **NOMBRE** y **APELLIDOS** sí tienen dependencia funcional completa respecto de **DNI** en la relación **ALUMNO**, y en el caso de la relación **CALIFICACION**, el atributo **NOTA** depende completamente de la clave principal compuesta por **DNI** y **COD_ASIGNATURA**. Igualmente ocurre en el caso de la relación **ASIGNATURA**, en la que tanto **CURSO** como **AULA** tienen dependencia funcional completa respecto de **COD_ASIGNATURA**.

Por lo tanto, las tres relaciones están en 2FN, habiendo conseguido **conservar las dependencias funcionales originales, la información y solucionado todos los problemas mencionados anteriormente**.

# Unidad Didáctica IX

### 3FN (Tercera Forma Normal)

> ***Una relación está en tercera forma normal (3NF) si y sólo si está en 2NF y todos los atributos no claves dependen de manera no transitiva de la clave primaria.***

Es decir, si sus atributos no clave son **mutuamente independientes** (no existe ningún atributo no clave que dependa funcionalmente de alguna combinación del resto de atributos no clave) y además sean por completo dependientes funcionalmente de la clave primaria.


Consideramos la relación ASIGNATURA del ejemplo anterior:

**ASIGNATURA** (COD_ASIGNATURA, CURSO, AULA)

En la que existen las siguientes dependencias funcionales:

**COD_ASIGNATURA ( CURSO**

**CURSO ( AULA**

Con estas dependencias surgen una serie de inconvenientes en la relación:

- **Anomalías en la inserción de tuplas:** no se puede insertar un curso si no se indica al menos una de las asignaturas de dicho curso. Estaríamos violando la regla de integridad de la clave al insertar una tupla con valores nulos en la clave primaria.
- **Anomalías en el borrado de tuplas:** si se borran las tuplas de todas las asignaturas de un curso concreto, se borra de manera automática dicho curso de la Base de Datos.
- **Anomalías en la actualización de tuplas:** si se realiza un cambio de aula para un curso determinado, tenemos que cambiar todas las tuplas de las asignaturas de dicho curso, lo que indica que existe una gran redundancia de información en la Base de Datos.

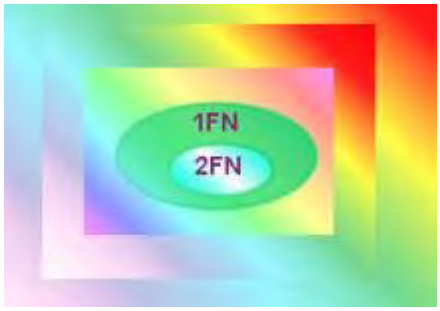
De esta manera volvemos a **cuestionarnos** cómo solucionar que una relación R no esté en 3FN... Al igual que en el caso de la 2FN, la solución pasa por una **descomposición** de la relación dada en otras relaciones que sí estén en 3FN. Esta descomposición se realiza de la siguiente manera:

> *Dada una relación **R** con atributos **X, Y, Z** simples o compuestos, con la clave **X** y atributos no claves **Y, Z** tal que en la relación existen las dependencias funcionales:*
>
> ***X ( Y***
>
> ***Y( Z***
>
> *... es decir, **una dependencia funcional transitiva X( Z**, entonces la relación **R** se descompone como sigue:*
>
> *De **R** se elimina **Z** (el atributo que tiene la dependencia funcional transitiva en la relación).*

*Se construye una nueva relación **R'** con:*
***Z** como atributo, pero que no sea clave principal.*
***Y** como clave principal de **R'**.*
***Y( Z** como dependencia funcional.*

Las relaciones **R** y **R'** resultantes sí están en tercera forma normal. En definitiva, hemos eliminado las dependencias transitivas.

Veamos esta descomposición con el **ejemplo** anterior:




**ASIGNATURA** (COD_ASIGNATURA, CURSO, AULA)

Lo primero que tenemos que hacer es comprobar si la relación está en 2FN, para lo que se tiene verificar que está en 1FN y que los atributos no clave tienen dependencia funcional completa respecto de la clave principal.

Damos por cierto que todos los atributos toman valores atómicos para que la relación esté en 1FN.

Comprobamos ahora que en efecto, tanto **CURSO** como **AULA** tienen dependencia funcional completa respecto de **COD_ASIGNATURA**, por lo que la relación **ASIGNATURA** está en 2FN.

Sólo nos queda comprobar si existen dependencias funcionales transitivas, en cuyo caso la relación dada no estaría en 3FN.

Como podemos ver existen las siguientes dependencias en la relación:

**COD_ASIGNATURA ( CURSO**

**CURSO ( AULA**

... por lo que existe una dependencia funcional transitiva en la relación.

¿Cómo eliminamos la dependencia funcional transitiva?

**Descomponiendo** la relación dada en otras de tal manera que desaparezca dicha dependencia, es decir, separando el/los atributo/s que producen la dependencia funcional transitiva en una nueva relación de la siguiente manera:

**ASIGNATURA** (COD_ASIGNATURA, CURSO)

**CURSO** (CURSO, AULA)



# Unidad Didáctica IX

## FNBC (Forma Normal de Boyce-Codd)

***Una relación está en forma normal Boyce/Codd (BCNF) si y sólo si todo determinante es***

***una clave candidata.***

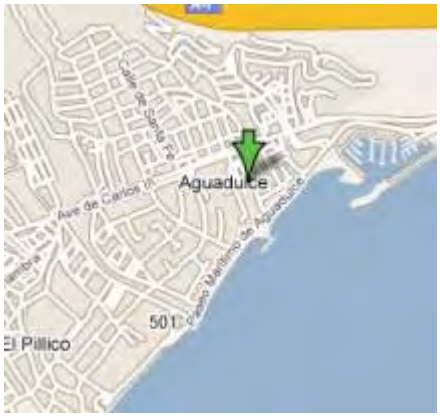
Si consideramos la siguiente relación:

**CALLEJERO** (CALLE, CIUDAD, COD_POSTAL)

con las siguientes dependencias funcionales:

**CALLE, CIUDAD ( COD_POSTAL**

**COD_POSTAL ( CIUDAD**




**COD_POSTAL** es un determinante que no es clave candidata, por lo que la relación no está en **FNBC**.

En esta ocasión vuelven a aparecer anomalías de inserción, actualización y borrado en la Base de Datos, y dichas **anomalías** desaparecen realizando una **descomposición** adecuada de la relación. En ocasiones esta descomposición puede dar lugar a una pérdida de dependencias funcionales, motivo por el que algunos autores no aconsejan pasar a la FNBC y quedarse en la 3FN. Otros sin embargo, prefieren continuar el proceso de normalización hasta sus formas más avanzadas.

En caso de decidir normalizar hasta la FNBC, la descomposición se realiza de forma genérica como sigue:

*Dada una relación **R** con atributos **X, Y, Z** simples o compuestos, con clave **{X, Y}** y tal que en la relación existen las siguientes dependencias funcionales:*

*{X, Y} ( Z*

***Z ( Y***

*es decir, existe un determinante no clave, y por lo tanto la relación no está en forma normal de Boyce/Codd. Dicha relación **R** se descompone como sigue:*

*En **R** se deja la parte de la clave que es independiente y todos los atributos no primarios, es decir, **R ( X, Z)***

*Se crea otra tabla **R'** con la parte de la clave restante y el atributo secundario del que depende, siendo éste último la clave de la nueva tabla. Es decir, **R' (Y, Z).***

Las relaciones **R** y **R'** resultantes sí están en forma normal de Boyce/Codd.

Vemos esta descomposición con la relación **CALLEJERO.**

¿Cómo podemos hacer que dicha relación esté en forma normal Boyce /Codd?

Muy sencillo, siguiendo los pasos indicados más arriba. Tenemos que **separar los atributos** origen del conflicto, para ello creamos las relaciones:

- En **R** dejamos la parte de la clave de **CALLEJERO** que es independiente, es decir, **CALLE**, y todos los atributos no primarios. En este caso sólo tenemos el atributo **COD_POSTAL**. Por lo que **R** queda:

**CALLEJERO'** (CALLE, COD_POSTAL)

- En **R'**(CODIGO_POSTAL) consideramos el resto de la clave primaria de la relación **CALLEJERO**, es decir, **CIUDAD**, y el atributo del que depende, en este caso **COD_POSTAL**, con lo que

**R'** (CODIGO_POSTAL) nos queda:

**CODIGO_POSTAL** (COD_POSTAL, CIUDAD)

Ahora **CALLEJERO'** y **CODIGO_POSTAL** sí están en FNBC.


Como las cosas se aprenden mejor practicándolas y un ejemplo vale más que mil palabras, vamos a ver la normalización "en acción" con dos ejemplos y los pasos a seguir para normalizar.

**Ejemplo 1 de Normalización**

**Ejemplo 2 de Normalización**

***Para saber más***

***Una vez obtenido el Modelo Entidad/Relación es preciso realizar algunas acciones que nos van a permitir conseguir una mejor definición de los datos.***

***Teoría de Normalización***

***http://www.inf.udec.cl/~basedato/apunte/capitulo5/capitulo5.html***

***Si quieres saber algo más acerca del precursor de la teoría de Normalización, pulsa el siguiente enlace donde encontrarás más cosas acerca de sus aportaciones al mundo de la informática:***

***Ronald Fagin***

***http://www.almaden.ibm.com/cs/people/fagin/*** **[versión en cache]**

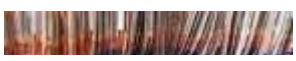


# Unidad Didáctica IX

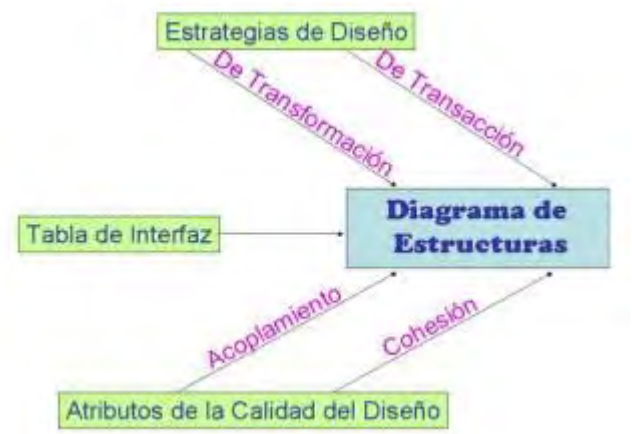
## Diseño estructurado y descomposición modular

*CASO. **SI Andalucía** intenta hacer una estructura para un Videoclub. **José** es el responsable del mismo y tras varias entrevistas con el cliente para conocer su forma de trabajo, ha concretado las especificaciones y ha construido un DFD estructurado en el que refleja el funcionamiento del Videoclub. Tras una visita, **Julia** se interesa en este proyecto y **José** le explica la necesidad de obtener ahora un diagrama en el que se representen todos los módulos de trabajo del equipo de desarrollo hasta conseguir la codificación completa de toda la aplicación. Con el diagrama de estructura obtenido **José** puede repartir el trabajo entre **Carmen** y **Víctor**, al tiempo que puede controlar fácilmente la calidad del software desarrollado, ya que irá recibiendo cada módulo con el tiempo suficiente para estudiarlo y probar su funcionamiento.*

El **objetivo** principal del diseño estructurado es desarrollar la **estructura del programa**, así como las relaciones entre los elementos (módulos) que



componen esa estructura. El diseño orientado al flujo de datos, permite establecer la transición del diagrama de flujo de datos (DFD) a una descripción de la estructura de programa, que se representa mediante un diagrama de estructuras. A continuación veremos estos diagramas y cómo obtenerlos partiendo de un diagrama de flujo de datos expandido.




Para obtener el **Diagrama de Estructuras** el diseño estructurado se ayuda de una serie de **herramientas** y técnicas muy útiles, que el analista debe manejar con soltura. Entre las que podemos encontrar:

- Tabla de Interfaz.
- Estrategias de Diseño:
  - De Transformación.
  - De Transacción.
- Y los atributos de la calidad del diseño:
  - Acoplamiento.
  - Cohesión.

Todas estas técnicas y herramientas van a ser el objetivo de los siguientes apartados de esta unidad.

---



# Unidad Didáctica IX

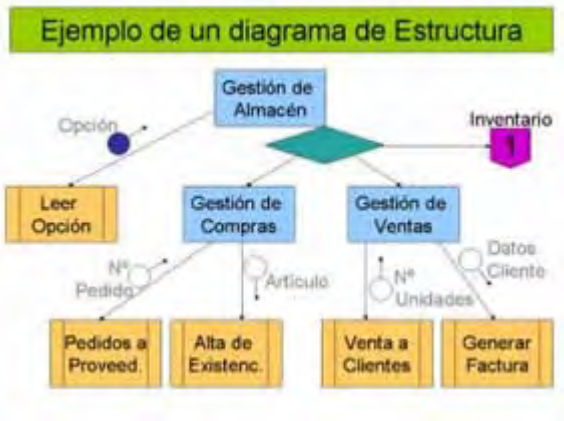
### Diagrama de estructuras

**Un diagrama de estructuras es una representación gráfica de la estructura de los programas, incluidas las relaciones entre los diferentes elementos que la componen.** Los elementos principales de un diagrama de estructura son:

- los módulos,
- las conexiones entre módulos y
- las comunicaciones entre los módulos.

En un diagrama de estructura, es preciso tener en cuenta...

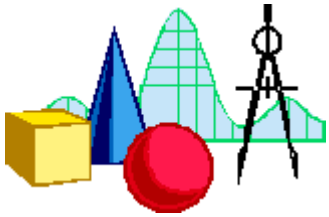
- La comunicación entre módulos se realiza mediante flags.
- Cada módulo se representa como un rectángulo con su nombre en el interior. El nombre que se asigna a cada módulo, debe representar la función que realiza.
- Un módulo "predefinido" es aquél que está disponible en la biblioteca del sistema o de la aplicación La representación de los módulos predefinidos suele ser especial.





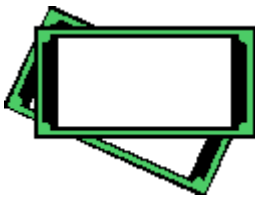

# Unidad Didáctica IX

## Módulo



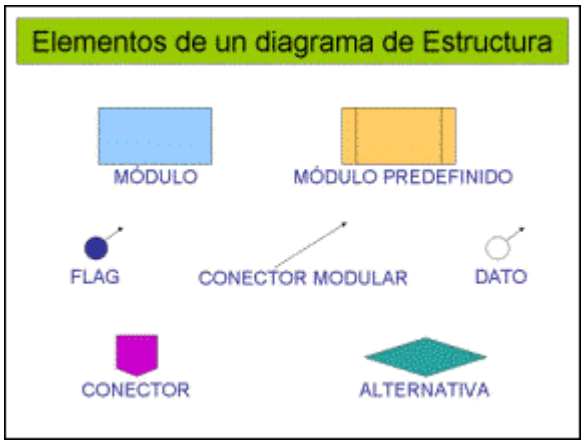
Dado que la arquitectura implica **modularidad**, el software debe dividirse en elementos que llamamos **módulos**. Los módulos se integran entre sí para que al ser ejecutados sean satisfechos los requisitos del sistema. **Un módulo consiste en una unidad claramente definida y manejable, con** interfaces **modulares claramente definidas**. La modularidad mejora la claridad del diseño, facilitando la implementación, la depuración, las pruebas, la documentación y el mantenimiento de un producto software.

Existen muchas definiciones de **módulo**, pero aquí nos vamos a decantar por la que da la teoría del diseño estructurado; "**un módulo es aquella parte de código que puede ser llamada y ejecutada de forma independiente**". Por tanto un módulo será un conjunto de sentencias (normalmente hasta 40 o 50 líneas de código) que realizan una actividad concreta.



En cualquier caso hay que tener en cuenta que pueden existir módulos sumamente **sencillos** (por ejemplo pedir por teclado un valor numérico) con dos o tres líneas de código, pero también módulos **complejos** que en su código incluyen llamadas a otros módulos (por ejemplo un módulo para calcular el descuento en una venta, puede hacer una llamada al módulo sencillo para pedir un valor numérico, entre otros).

La definición de los diferentes módulos del código del software depende exclusivamente del programador, que utilizará como único criterio para esta definición, el que facilite a máximo el trabajo de desarrollo.

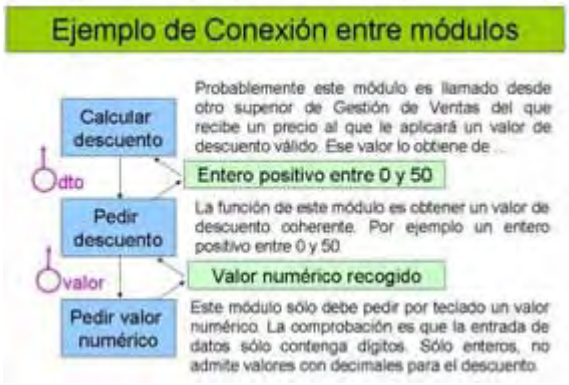





# Unidad Didáctica IX

## Conexión entre módulos

Normalmente un sistema está compuesto por una **organización jerárquica** de módulos que cooperan y se comunican con el fin de realizar una tarea. La llamada de un módulo se representa con una flecha, sobre la que regresa al finalizar la ejecución del módulo. Veamos un **ejemplo** de esto.

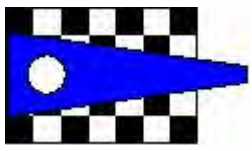



**Esta imagen se puede usar como lanzadera**



# Unidad Didáctica IX

## Comunicación entre módulos



La comunicación entre los módulos se realiza a través de **datos y flags** (banderas o señalizadores).

- Los **datos** son procesados y el resultado de ese procesamiento es utilizado,

en cambio
- los **flags** son valores de condición que se utilizan para comunicar condiciones entre los módulos.

Además los datos suelen estar relacionados con el problema y son importantes en el mundo exterior mientras que los flags sólo tienen importancia durante la comunicación de información entre módulos.

- En el diagrama de estructuras los datos se representan mediante un círculo con una flecha que indica la dirección de envío.
- En el caso de los flags el círculo es negro.

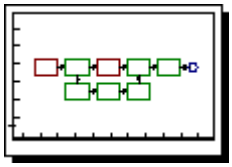
**Ejecución de un diagrama de Estructura**



# Unidad Didáctica IX

### Tabla de interfaz

Los **parámetros** que se pasan entre módulos se pueden representar en la Tabla de Interfaz, que permite una mejor especificación de los parámetros y sirve de apoyo a los diagramas de estructuras, mejorando su claridad cuando el número de parámetros lo hace confuso.



La primera **ventaja** del uso de la tabla de interfaz es verificar que todos los parámetros de entrada sirvan para un propósito determinado. La columna de USO tiene nemotécnicos que indican el modo de uso del parámetro en el módulo. Algunos de esos nemotécnicos pueden ser...

| Nemotécnico | Explicación |
|---|---|
| **P** | El parámetro es PROCESADO. Por ejemplo... a = **b** + 2; **b es procesado.** |
| **M** | El parámetro es MODIFICADO. Por ejemplo... **a** = b + 2; **a es modificado.** |
| **T** | El parámetro es TRANSFERIDO al llamar a otro módulo, sin alterar su valor. |
| **C** | El parámetro es usado como una VARIABLE DE CONTROL, quizás para actuar como conmutador, como un flag o para especificar una función usada al llamar a otro módulo. |
| **I** | El parámetro es TRANSFERIDO a otro módulo y es MODIFICADO en ese segundo módulo. |

***Para saber más***

***Diseño estructurado y descomposición modular. Apuntes muy completos sobre este tema, con apartados correspondientes a las estrategias de diseño, los atributos de calidad y las metodologías de diseño. También permite bajar una versión de estos apuntes en PDF.***

***Apuntes de Diseño estructurado***

***http://www.chaco.gov.ar/UTN/disenodesistemas/apuntes/de/dise%C3%B1o_estructurado.htm***

# Unidad Didáctica IX

## Estrategias de diseño

*CASO. Una de las partes que no comprende **Julia** en el diseño estructurado es cómo es capaz José de construir con tanta rapidez el diagrama de estructuras a partir directamente el DFD. **María** le explica que existen estrategias que facilitan esta tarea y que con cierta experiencia la interpretación de un DFD de no mucha dificultad, se puede conseguir en cuestión de pocos minutos, aunque eso sí, después habrá que depurarlo un poco.*

El **diseño estructurado** ofrece dos estrategias para conseguir una creación rápida de un buen diseño a partir de un DFD (Diagrama de Flujo de Datos):

- el Análisis de Transformación y
- el Análisis de Transacción.

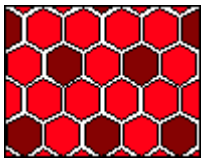
Básicamente el diseño estructurado permite una traducción de las representaciones de la información (DFD) a una descripción de diseño de la estructura del programa, que en función del tipo de flujo de datos de que se trate, elegirá una estrategia u otra.



# Unidad Didáctica IX

## Análisis de transformación

En el Análisis de Transformación, los datos entran en el sistema mediante caminos denominados flujos de llegada, para después ser transformados y finalmente conducidos a la salida, constituyendo el flujo de salida. Cuando un DFD es de este tipo se dice que tiene características de transformación.



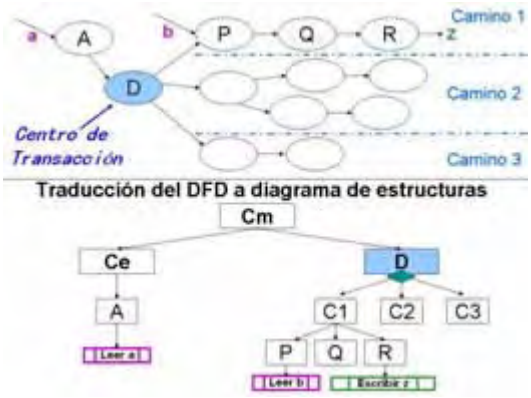



# Unidad Didáctica IX

## Análisis de transacción

Por otro lado, en una aplicación software un dato puede determinar varios caminos

alternativos por los que puede transitar el flujo de información y en cada uno de esos caminos la función sobre los datos varía. Cada uno de estos posibles caminos recibe el nombre de camino de acción. Estos caminos parten de un centro de transacción exclusivo.



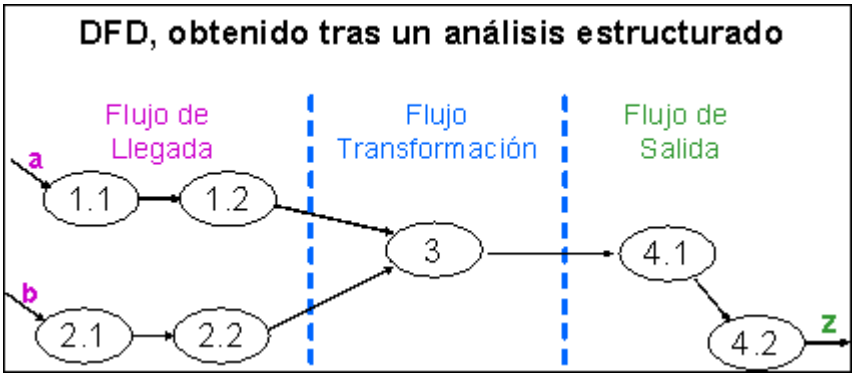



# Unidad Didáctica IX

### Pasos para el análisis estructurado

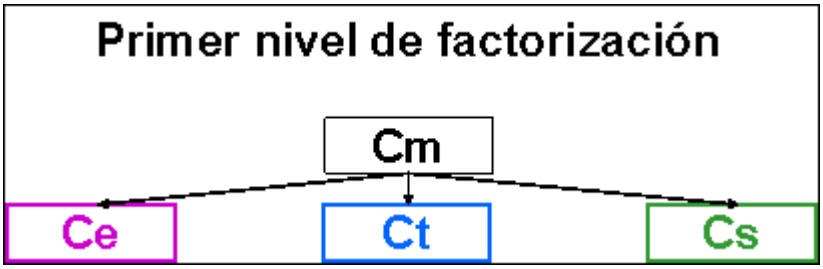
Los pasos del análisis de transformación o de transacción son los siguientes:

1. ***Revisión del modelo fundamental del sistema.*** *Si hemos realizado un análisis estructurado, entonces tendremos los DFD del sistema. En caso contrario es preciso obtenerlos partiendo de las especificaciones del sistema.*
2. ***Determinar si el DFD tiene características de trasformación o de transacción.*** *En general, el flujo de información de un sistema puede representarse siempre como transformación. Sin embargo, conviene elegir para el análisis de transformación sólo aquellos con claras características de transformación. Estas características se aprecian observando la naturaleza que prevalece en el DFD.*
3. ***En caso de análisis de transformación, aislar el centro de transformación, especificando los límites del flujo de llegada y de salida.*** *Estos límites están abiertos a interpretación (dependerán del diseñador), siendo posible la aparición de soluciones de diseño alternativas. En cambio el centro de transformación es la parte del DFD que contiene las funciones esenciales del sistema.*

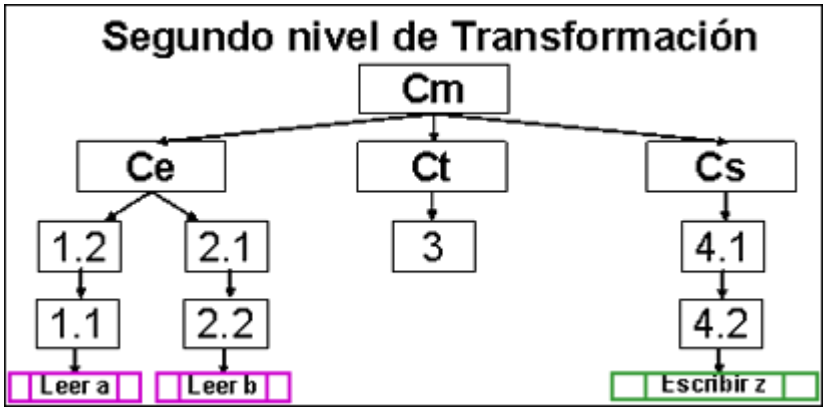



1. ***Realizar el primer corte del análisis de estructuras****. El propósito del análisis de transformación es convertir un DFD en un diagrama de estructuras para este tipo de*

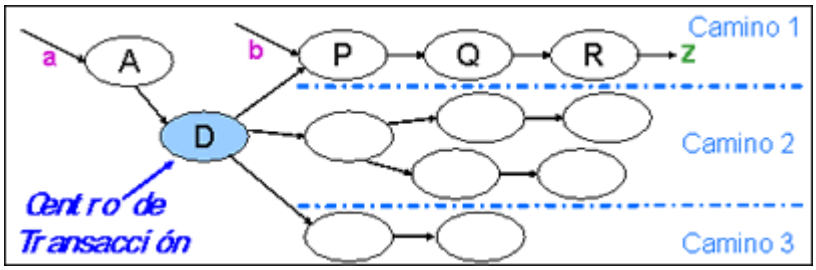
*operación.*



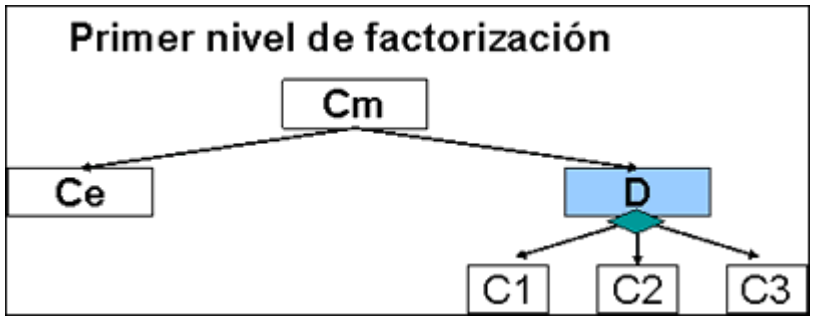

2. ***Ejecución del "segundo nivel de factorización".*** *Se realiza mediante la conversión de los procesos de un DFD en los módulos correspondientes del diagrama de estructuras.*



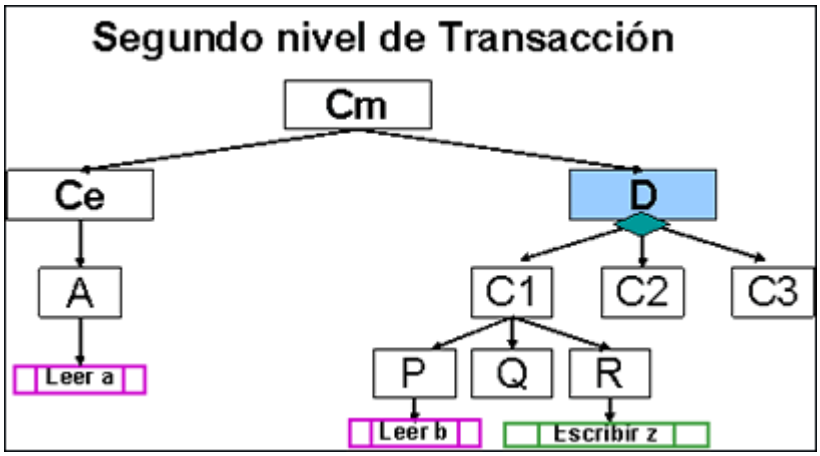

4. ***En caso de análisis de transacción, identificar el centro de transacción y las características del flujo de cada camino de acción.*** *Partiendo del DFD la posición del centro de transacción puede descubrirse inmediatamente, porque será el origen de varios caminos de información.*




1. ***Realizar el primer corte del análisis de estructuras.*** *Hay que convertir el flujo de transacción en una estructura de sistema con una bifurcación de entrada y otra de salida. Para la salida se añade un módulo controlador por cada caminote acción.*




2. ***Ejecución del "segundo nivel de factorización".*** *Se desarrolla cada camino de acción dependiendo de su tipo de flujo.*




5. ***Refinar la estructura del sistema usando medidas y guías de diseño.*** *Se puede aumentar o disminuir el número de módulos para conseguir una factorización lógica, que tenga una buena calidad y una estructura que se implemente sin dificultad.*
6. ***Asegurar la funcionalidad del diseño obtenido.*** *Hay que revisar el diagrama de estructuras observando que el orden de ejecución es el correcto para conseguir los resultados buscados.*

Realmente lo que ocurre es que en un mismo DFD podemos encontrar "**ramas**" o partes que se ajusten a las características para aplicar una estrategia de transformación y otras que lo hagan hacia una estrategia de transacción, por lo que es decisión del analista diseñador adoptar la decisión correcta, si bien su mayor herramienta es la **experiencia** que tenga en este tipo de trabajos. Por **ejemplo**, si construimos el DFD de los pasos que hemos comentado en este apartado, podemos obtener algo que claramente mezcla ambas estrategias, que además no son incompatibles.


Si convertimos ahora este DFD a diagramas de estructuras, podemos observar claramente que tiene ur

flujo de información de entrada y otro de salida, y que precisamente el de transformación presenta dos caminos de acción alternativos.




# Unidad Didáctica IX

## Atributos de la calidad del diseño

*CASO:*
*Pero ¿cómo saber si el diseño es correcto y no hemos cometido ninguna pifia?* ***José*** *conoce bien la teoría del análisis de aplicaciones y por extensión la del diseño estructurado de sistemas. Y eso le permite diseñar los módulos evitando algunos conceptos poco deseables. Para ello intenta siempre que cada módulo obtenido tenga la menor relación posible con el resto de los módulos del sistema, de igual modo consigue que cada módulo haga una única cosa, o en su defecto el menor número posible de cosas.*

En cada proyecto se deben decidir cuáles son los **requisitos** de calidad a cumplir y decidir lo más importantes. Pero para asegurar y evaluar la **calidad del software**, ésta se debe poder medir. Para ello suelen emplearse las medidas o métricas del software. En este apartado nos centraremos en las métricas que miden los aspectos de la calidad estructural en el diseño: el **acoplamiento** y la **cohesión**.

### Acoplamiento

Se define como el grado de **interdependencia** entre los módulos. Para un buen diseño, se debe siempre intentar minimizar el acoplamiento; es decir, hacer los módulos tan independientes unos de otros como sea posible. Para conseguir un acoplamiento mínimo, ningún módulo tiene que preocuparse de los detalles de la construcción interna del resto de los módulos.

### Cohesión

La cohesión estudia la relación que existe entre los elementos de un mismo módulo. La intención básica del concepto de cohesión es **organizar** estos elementos de tal manera que los que tengan una mayor relación a la hora de ejecutar una tarea pertenezcan al mismo módulo, y los

elementos no relacionados estén en módulos separados.

La cohesión se puede definir como la **medida de la relación funcional** de los elementos de un módulo Elemento en este sentido significa cualquier pieza de un módulo que lleva a cabo algún trabajo o define algún dato, es decir, cualquier procedimiento, función, llamada a otros módulos, definición de datos, etc.

En una situación ideal, un módulo coherente debe hacer una única cosa. Esto es importante a la hora de diseño, ya que influye en un menor coste de los programas y en una mayor calidad en el sistema.



# Unidad Didáctica IX

## Metodologías de diseño detallado de programas

*CASO:*
*El paso final en el desarrollo de una aplicación lo constituye la codificación y las pruebas a cada módulo que nos van a indicar que su funcionalidad es correcta.* ***José*** *explica a* ***Julia*** *que lo normal es optar por una de las dos metodologías más utilizadas actualmente para la codificación de cada uno de los módulos obtenidos durante el diseño estructurado. Con estas metodologías va a obtener una perfecta descomposición modular del proyecto que le va a permitir hacer un reparto adecuado del trabajo del equipo de desarrollo. Al finalizar cada uno de los módulos él mismo se encargará de aplicar las pruebas que determinarán si el módulo es correcto o por el contrario, necesita reajustes.*

***Julia*** *comenta que lo tienen todo perfectamente planificado y que de esa forma, ¿cómo es posible que algo salga mal?*

Las metodologías de **programación estructurada** ayudan al diseño tomando como punto de partida una especificación detallada de la entrada, la salida y los algoritmos del programa a construir. Permiten obtener una **estructura jerárquica del programa**, equivalente a un organigrama o diagrama de flujo. El enfoque consiste en el análisis de los datos que deben manejar los programas para generar una estructura lógica de los mismos. Podemos resumir la utilidad de estas metodologías de programación, como un modo de tratar el diseño que permita obtener fácilmente el pseudocódigo de un programa.

Estas metodologías son una consecuencia directa de la **programación estructurada**, cuyo éxito se basaba principalmente en constituir una forma sistemática y elegante de resolver problemas de programación.

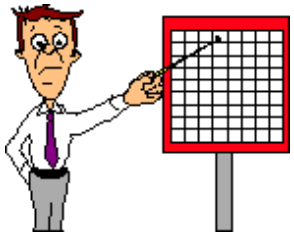
Las metodologías de programación más conocidas son el método **Jackson** y el método **Warnier**, que trataremos en los apartados más inmediatos.



# Unidad Didáctica IX

### Método Jackson

El método Jackson se basa en el principio de que la base inicial del diseño del programa son **los datos del problema** y no los requisitos funcionales exigidos. Esta forma de enfocar el desarrollo de software basada en los **datos** permite una mayor objetividad, (ya que estos son los elementos más objetivos de las especificaciones). Una vez obtenida una **estructura objetiva del programa**, que constituye un reflejo del mundo real con el que trata el programa, resulta más fácil asignar las distintas funciones a realiza en el lugar más apropiado.

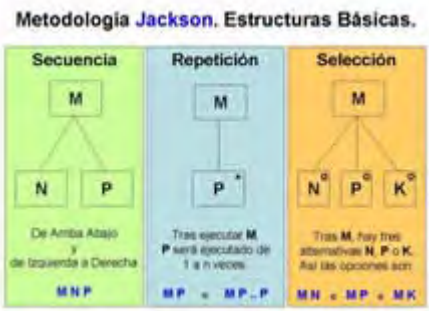


Para aplicar el método Jackson, debemos partir de una buena **especificación del problema** que se quiere resolver; datos de entrada, datos de salida y algoritmos aplicables, y seguir las siguientes fases:

1. *Formar las estructuras de **datos** de salida y de entrada a partir de los datos del problema.*
2. *Determinar las **correspondencias** (o elementos comunes) entre ambas estructuras de datos (entradas y salidas).*
3. *En función de las correspondencias, obtener una **estructura única** para el programa, que puede traducirse fácilmente a un diagrama de flujo de control.*
4. *Asignar a la estructura del programa las **operaciones ejecutables** de programa derivadas de las especificaciones funcionales.*
5. *Traducir el conjunto estructura-operaciones a un formato pseudocódigo, cuya codificación en un lenguaje de programación, resulta bastante más sencilla.*

Tanto las estructuras de datos de entrada y de salida, como la estructura del programa, se documentan mediante los diagramas de estructura de Jackson, que deben ser leídos de arriba abajo y de izquierda a derecha y que se fundamentan en tres estructuras básicas:

- **Secuencia**. Cuando dos o más componentes son colocados juntos en estricto orden secuencial para formar un componente mayor.
- **Repetición**. Cuando un componente o elemento de datos, se repite varias veces.
- **Selección**. Cuando es preciso escoger entre dos o más componentes. La selección puede tener una, dos o más alternativas.




La **notación** en este tipo de diagramas de estructura, puede observarlo en el siguiente ejemplo que representa el cálculo del IVA, cuyos datos de entrada son el precio y el tipo de IVA.


En este **diagrama** entendemos que el primer proceso a ejecutar es la ENTRADA DE DATOS, concretamente el PRECIO de un producto y el TIPO DE IVA. Este último se pedirá mientras sea erróneo, hasta conseguir uno válido (4%, 7% o 16%). A continuación se ejecuta el PROCESO con una única tarea CALCULAR IVA (PRECIO + PRECIO * TIPO DE IVA). Finalmente ejecutamos MOSTRAR RESULTADOS, donde hay que elegir entre PANTALLA O IMPRESORA, sólo una de estas alternativas.

---

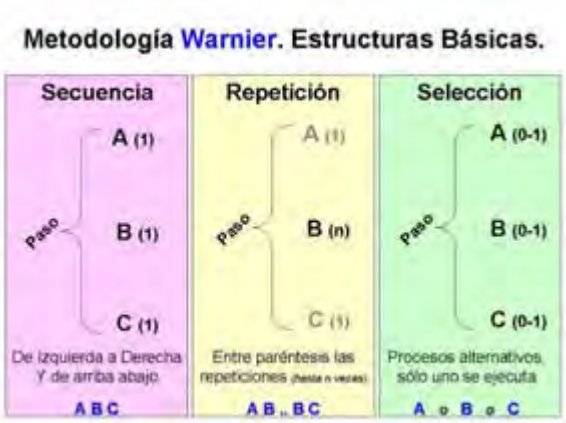


# Unidad Didáctica IX

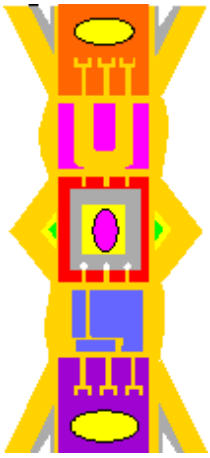
### Método Warnier

La premisa básica de esta metodología es que la **"forma" o estructura de los datos** a procesar determinará la forma o estructura del programa. Es un método muy parecido al Jackson, la única diferencia radica en que la determinación de la estructura del programa no tiene en cuenta las correspondencias entre salida y entrada. Se basa fundamentalmente en la **entrada**, realizando una optimización del número de decisiones en función de las acciones a realizar. La representación de cualquier proceso se puede hacer mediante llaves.

El método se basa en la descomposición por **niveles del problema**. En cada nivel se detallan los tratamientos que permiten la resolución del problema planteado. Las estructuras utilizadas son conceptualmente idénticas a las del Jackson, si bien su representación varía.




Los **pasos** a seguir en esta metodología son:



- Estudio de los datos de salida.
- Estudio de los datos de entrada teniendo en cuenta la organización de los datos en el fichero lógico de salida y analizando las posibles fases de tratamiento.
- Cuadro sinóptico o de descomposición formado a partir de la estructura de los datos de entrada, salida y del tratamiento.
- Diagrama de flujo y lista de instrucciones. El diagrama se obtiene de forma automática del cuadro sinóptico y su uso puede no ser imprescindible. La lista de instrucciones, su orden de ejecución, produce en casi su totalidad la escritura del programa con independencia del lenguaje.

Los **diagramas Warnier** también presentan una estructura **jerárquica**, pero esta vez se extiende horizontalmente de izquierda a derecha, y en un mismo nivel (o llave) de arriba abajo. La notación puede observarla en la siguiente imagen correspondiente a una representación del ejemplo anterior sobre el cálculo del IVA.


***Para saber más***

***Archivo PDF que puede complementar el contenido del tema, ya que se expone en forma esquemática, y que puede ser considerado un resumen de la segunda parte de esta unidad. Explica cada uno de los apartados con ejemplos, de forma clara y concisa. Cabe destacar el tratamiento que hacer de las metodologías Jackson y Warnier.***

***Diseño de Sistemas.***

***http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema07.pdf*** **[versión en cache]**