Módulo : BASES DE DATOS

Curso : Vision General de Bases de Datos

ALFREDO GOÑI
EDUARDO MENA

Departamento de lenguajes
y Sistemas Informáticos
Facultad de Informática de Sn.Sn.
U.P.V. / E.H.U.

INDICE

1. Nociones Generales sobre Bases de Datos

ABSTRACCION DE DATOS

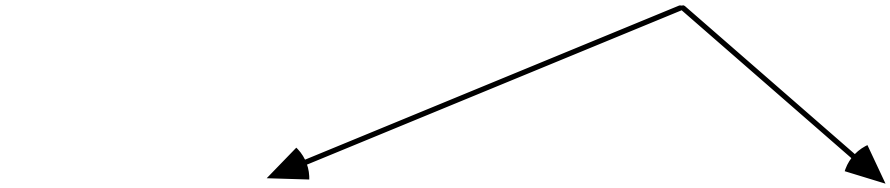
Tres mundos:

MUNDO REAL existen cosas y se producen acontecimientos
cosas: libros, personas ...
acontecimientos: nace una persona ...

MUNDO CONCEPCIONES de las personas acerca del mundo real. Cada persona percibe el Universo de Discurso.
Cada observador conceptualiza ciertas cosas como entidades (que tienen atributos).

MUNDO REPRESENTACIONES de las concepciones de la realidad.
Permiten almacenar,transmitir (toda o parte de una concepción) así como hacer operacionescon ella.

Toda representación tiene:



Aspecto lógico
Lenguaje

Aspecto físico
Soporte de datos

DATOS: Representación de hechos, conceptos o instrucciones, hecho de una manera formalizada, apta para su comunicación, interpretación o proceso, bien por seres humanos, bien por medios automáticos.

Manipulados por programas y representados en diferentes formas.

INFORMACION: Significado que se atribuye a los datos a partir de las reglas convencionales utilizadas para su representación.

Se facilita información para una futura interpretación y uso por parte del usuario. Con el fin de facilitar información a los usuarios, hay que realizar diversas tareas con los datos.

MANEJO DE INFORMACION EN LA EMPRESA

1. Tamaño de la empresa: Pequeño

Toda la información al alcance de las manos o en la cabeza del empresario.

2. Empresa crece. (Varios departamentos)

Cada director crea sus propios ficheros y libros de contabilidad (no compartían datos)

- Surgen problemas porque existían múltiples copias de un mismo dato, entre las que se creaban inconsistencias

Empresario recibe informes inconsistentes

Posible solución

EMPRESA CREA UN DEPARTAMENTO DE PROCESO DE DATOS

El ordenador procesaba grandes volúmenes de datos más rápidamente y con más fiabilidad (Conjunto de ficheros)

Sin embargo el empresario recibe informes inconsistentes

Dificultad en modificar el sistema para permitir el manejo de nuevos datos

3. **Solución ideal:** Comprar un Sistema de Gestión de Bases de Datos

   Se podían añadir nuevos datos para la obtención de nuevos informes sin modificar los programas ya existentes

   Sin embargo los informes siguen siendo inconsistentes

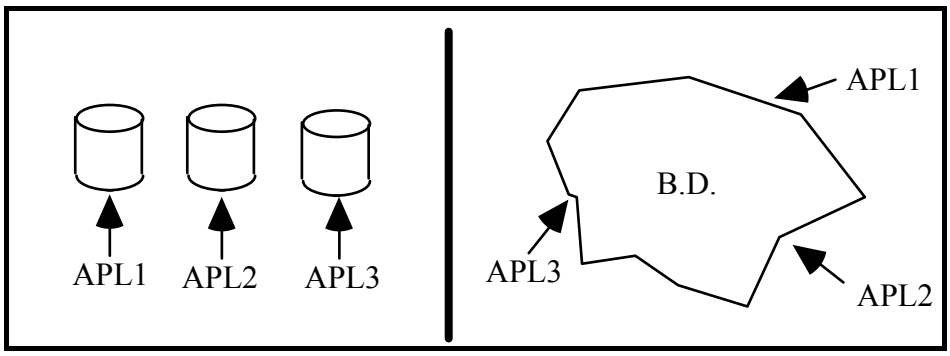
EL USO DE UN S.G.B.D. no significa que automáticamente se crea una base de datos

Esencial: Tener una filosofía de datos compartidos y de eliminación, o al menos, control, de los datos duplicados que beneficie a toda la empresa.

Los jefes de departamento deben comprender la necesidad de compartir los datos. Los datos que recogen y utilizan pertenecen a la compañìa en su conjunto.

Hay que establecer la función de Administrador de la Base de Datos para que actúe como fuerza de reorganización.

DE LOS SISTEMAS TRADICIONALES DE FICHEROS A LOS SISTEMAS DE BASES DE DATOS



SISTEMAS ORIENTADOS HACIA EL PROCESO

- Dependencia entre la estructura lógica y física
- Redundancia de la Información
- Descripción de la estructura de los ficheros en los programas

SISTEMAS ORIENTADOS HACIA LOS DATOS

- Independencia entre la estructura lógica y física

BASE DE DATOS

Conjunto estructurado de datos, abierto a los universos de discurso de todas las aplicaciones y de todos los usuarios

Programador de aplicaciones
Usuario final
Administrador de la B.D.

A veces se añaden a la definición expresiones que indican propiedades deseadas, tales como:

- Con la mínima redundancia
- Con una responsabilidad definida sobre el uso de cada esquema de datos

BANCO DE DATOS:

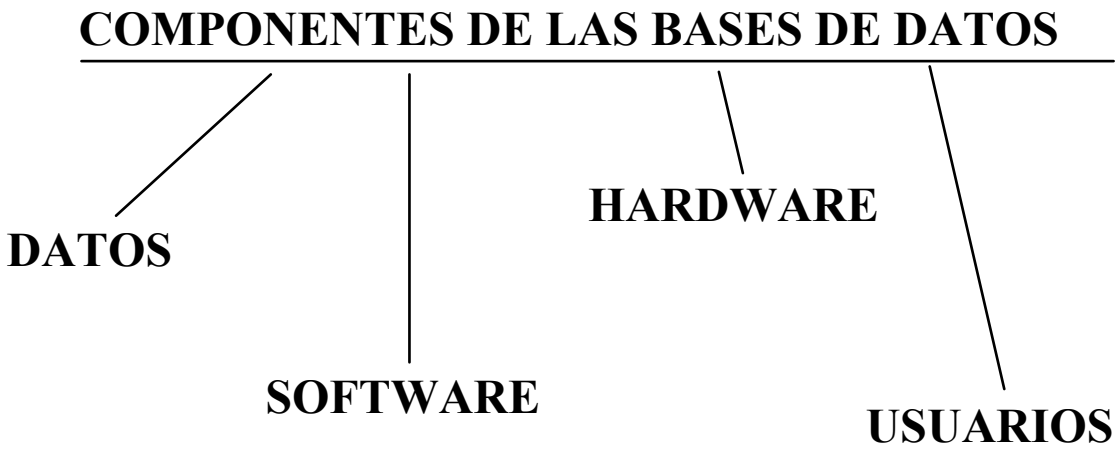
El usuario únicamente puede consultar la información, nunca actualizarla.

DIFERENCIA ENTRE FICHEROS Y BASES DE DATOS

FICHERO - Propio de un usuario o una aplicación
- Diseñado en función de las necesidades del usuario o de la aplicación
- Representación de una sola concepción del mundo real, de un único Universo de Discurso
- Los registros del fichero corresponden a una clase del Universo de Discurso

(Tanto si el fichero está informatizado, como si es un conjunto de fichas de cartulina)

BASE DE DATOS - Múltiples usuarios
- Cada usuario tiene su concepción de su Universo de Discurso
- Todas las concepciones tienen que estar representadas en la B.D.
- Cada usuario debe poder utilizar la B.D. independientemente de los demás
- Una B.D. se diseña integrando todos los puntos de vista, comprendiendo todos los Universos de Discurso (actuales o futuros)



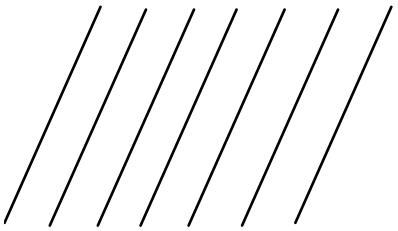
DATOS Integrados y Compartidos

Integrados BD puede considerarse como unificación de diferentes ficheros de datos en los que se ha eliminado la redundancia.

Compartidos Datos de una BD pueden ser compartidos por diferentes usuarios (Distintos usuarios ven una misma base de datos de diferentes formas).
A un mismo conjunto de datos pueden acceder al mismo tiempo diferentes usuarios.

SOFTWARE

Base de Datos Física



Sistema de Gestión de Bases de Datos

Usuarios

Diccionario de Datos Almacena información sobre los datos de la B.D. (Definiciones y descripciones)

HARDWARE

Memoria secundaria

Máquinas Bases de Datos

Hardware diseñado específicamente para soportar sistemas de B.D.

USUARIOS

Informáticos Escriben programas de aplicación que utilizan la B.D.

Programas operan con los datos:
- Obteniendo información
- Creando nueva información
- Borrando nueva información

Finales Acceden a la B.D. a través de un programa escrito por informáticos o mediante un interfaz proporcionado por el sistema.

ADMINISTRADOR DE LA BASE DE DATOS

Persona (o grupo de personas)
responsable del control global
de sistema de B.D.

FUNCIONES DEL ADMINISTRADOR DE LA B.D.

- Decidir el contenido de la B.D.
  (seleccionar entidades, propiedades de las entidades y definir el esquema conceptual)

- Decidir el tipo de estructuras de almacenamiento y estrategias de acceso a utilizar
  (diseño del esquema interno)

- Conectar con los usuarios
  (definir o ayudar a definir los esquemas externos)

- Definir los controles de integridad y seguridad

- Diseñar las estrategias para la reconstrucción de la B.D. ante fallos que se produzcan
  (copias de seguridad, etc.)

- Control sobre el rendimiento de la B.D. y dar respuesta a cambios de requerimientos
  (rutinas estáticas, rutinas de reorganización... etc)

OBJETIVOS DE LAS BASES DE DATOS

1. CONTROL CENTRALIZADO DE LA EMPRESA SOBRE SUS DATOS

- Disminución de la redundancia
- Eliminación de Inconsistencias
- Los Datos pueden ser Compartidos
- Mejor y más Normalizada Documentación de la Información
- Mayor valor informativo
- Restricciones de Seguridad
- Mantenimiento de la Integridad

2. **INDEPENDENCIA DE LOS DATOS**

Inmunidad de las Aplicaciones a los Cambios en:

Estructura de Almacenamiento y en los Requerimientos

EVOLUCION HISTORICA

PARALELA AL DESARROLLO DE LA TECNOLOGIA DE LOS ORDENADORES

Ficheros

. Soporte Papel...
. ..1945 aparecen Cintas Magnéticas
. Finales 50 discos..
. Principios 60 "Sistema de Gestión de Información" ISD (65). General Electric. (Crear grandes ficheros integrados que podían ser compartidos por cierto número de aplicaciones)
. Mediados 60 ficheros invertidos.

Bases de Datos

. Finales 60, principios 70 aparecen los S.G.B.D.
IMS/1 (71)
. Especificaciones CODASYL (71) y (78)

Arquitectura 2 niveles
Arquitectura 3 niveles

IDS/2..
. Especificaciones ANSI (72)
. Sistemas relacionales. CODD 1970
INGRES 75, ORACLE 81
. Futuro ???

INTEGRIDAD Asegurar que los datos almacenados en la B.D. son correctos.

Datos incorrectos pueden surgir por
- errores en la entrada de los datos
- error del programa de aplicación
- falsificación deliverada

RESTRICCIONES DE INTEGRIDAD

Aseguran que modificaciones realizadas por usuarios autorizados no den lugar a una pérdida de consistencia.

Conjunto de predicados (reglas escritas en lenguaje de alto nivel) que deben ser verificados por los datos de la base de datos.

(Compiladas y almacenadas en el diccionario)

Sistemas comerciales : manejan un nº limitado
"usuario" encargado de su definición y manejo

Problema : COSTO en tiempo

Solución "triggers"

TIPOS DE RESTRICCIONES DE INTEGRIDAD

a) Referentes a un solo dato

   a.1 Restricciones sobre el dominio

   a.2 Restricciones sobre el rango de posibles valores

   a.3 Restricciones sobre la clave

b) Referentes a varios datos

   b.1 Dependencias funcionales

   b.2 Depencias multievaluadas

   b.3 Restricciones aritméticas que deben verificar ciertos datos

   b.4 Valores invariantes

HIPOTETICO LENGUAJE PARA DEFINIR RESTRICCIONES DE INTEGRIDAD

CREATE CONSTRAINT
(en general debe especificar)

- Nombre de la restricción
- La restricción en sí misma
- La respuesta en caso de violación

ejemplos:

- El valor de "estado" debe ser siempre positivo

```
CREATE CONSTRAINT    C1
        CHECK ESTADO>0
        ELSE     rechazar operación
```

- Los valores de "estado" nunca pueden decrecer

```
CREATE CONSTRAINT    C2
        BEFORE UPDATE   Estado FROM Nuevo-Estado
        CHECK    Nuevo-Estado > Estado
```

- El valor medio de "estado debe ser mayor 25

```
CREATE CONSTRAINT   C3
        CHECK IF EXISTE S THEN AVG S. ESTADO >25
```

SEGURIDAD

La Base de Datos debe estar protegida frente a accesos no autorizados.

Para proteger la B.D. se pueden definir medidas de seguridad a distintos niveles:

- **Físico** Controles en la sala donde se encuentra el ordenador.

- **Humano** Decidir a qué personas se conceden las autorizaciones.

- **Sistema Operativo** Ya que la mayoría de los sistemas de B.D. permiten el acceso a través de redes, resulta interesante la seguridad a nivel de software junto con el Sistema Operativo.

- **Sistema de Base de Datos**.

  - Personas autorizadas a acceder a la B.D.
  - Personas autorizadas a acceder a partes de la B.D. (Vistas)
  - Personas autorizadas a acceder a valores de unos datos.
    Personas autorizadas a realizar operaciones sobre los datos.
  - Personas autorizadas a realizar consultas estadísticas.

Ejemplo de definición de medidas de seguridad

GRANT <privilegios> ON <relación o vista> TO <lista usuarios>

GRANT Update ON depositos to V_1, V_2,V_3

Consideraciones adicionales

- Niveles de autorización

  Un acceso será autorizado si y solo si el nivel de la persona que accede es igual o superior al nivel de la persona que transfiere la actualización.

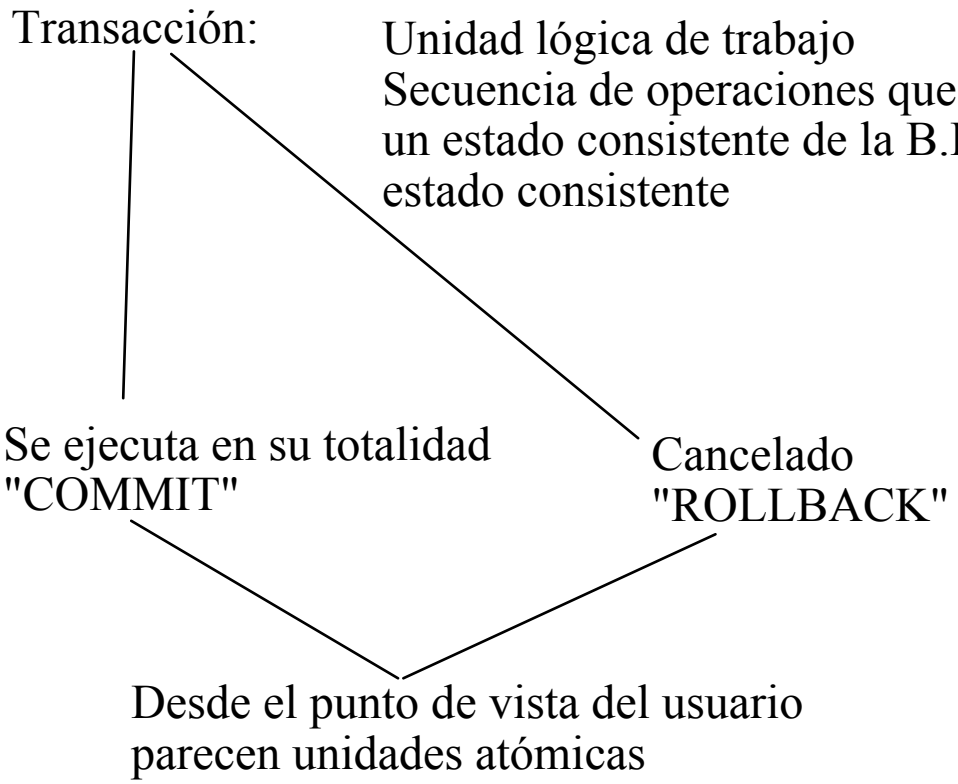
- Nunca los sistemas de seguridad son perfectos
  Interesante realizar "auditorías"

  fichero especial que almacena información sobre las operaciones realizadas por todos los usuarios.

- Criptografiado de datos.

- En algunos sistemas el mecanismo global de seguridad puede ser opcional.

CONCURRENCIA Técnicas necesarias para asegurar que transacciones concurrentes (se ejecutan al mismo tiempo) no interfieran en sus respectivas operaciones.



PROBLEMAS DE CONCURRENCIA

1. Pérdida de actualizaciones

Transacción A	tiempo	Transacción B
_____		_____
_____		_____
lee E	t1	_____
_____		_____
_____	t2	lee E
_____		_____
modifica E + 100	t3	_____
_____		_____
_____	t4	modifica E + 200

Resultado final E + 200, Resultado correcto E + 300

2. Dependencia con "UNCOMMIT" transacción

Transacción A	tiempo	Transacción B
_____		
_____		
	t1	Modifica A + 100
Modifica A + 100	t2	
	t3	ROLLBACK

Resultado final A + 200, Resultado correcto A

3. Análisis de Inconsistencias

Transacción A	tiempo	Transacción B
lee C_1, sum = 40	t_1	
lee C_2, sum = 40	t_2	
	t_3	lee C_3
	t_4	Modifica C_3 - 10
	t_5	Modifica C_1 + 10
	t_6	COMMIT
lee C_3, sum 110	t_7	

Transacción A: Suma los balances de las cuentas
Transacción B: Transfiere 10 de C_3 a C_1

Resultado final sum = 110
Resultado correcto sum = 120

Se observa en este caso que la transacción B termina correctamente.

OBJETIVO: Caracterización de las ejecuciones sin conflicto.

Dado un conjunto de transacciones T_1, T_2, ... T_n, se denomina **HORARIO** a toda intercalación de las acciones (comando indivisible ejecutado por una transacción) de las mismas que respete el orden relativo de las acciones dentro de cada transacción.

Ej.: T_1 a_{11} a_{12} a_{13}
T_2 a_{21} a_{22}

Posible horario a_{21} a_{11} a_{12} a_{13} a_{22}

Un horario, para un cierto conjunto de transacciones, produce un resultado válido si dicho resultado coincide con el resultado de algún horario serial. Se dice que el horario es **SERIALIZABLE.**

SERIAL $\neq$ SERIALIZABLE

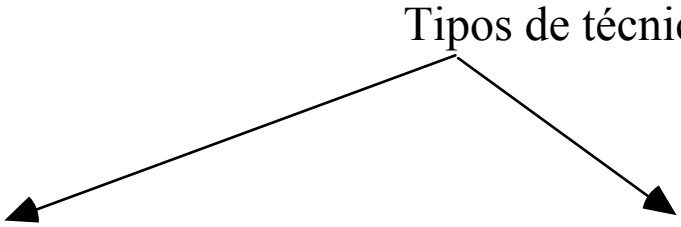
↓

las transacciones se siguen ejecutando concurrentemente pero se utiliza algún mecanismo de concurrencia cuando sea necesario

TECNICAS PARA EL CONTROL DE INTERFERENCIAS

Determinar si un horario es serializable es NP-completo.

Por lo tanto, en la práctica se imponen restricciones a la libre intercalación de las acciones de las distintas transacciones.



Pesimistas

Imponen a priori ciertas restricciones al solapamiento de transacciones que impiden horarios sospechosos.

- TIME-STAMPING
- RESERVAS

Optimistas

No ponen trabas al solapamiento pero a posteriori se comprueba si ha habido interferencias.

- KUNG

TIME - STAMPING La idea fundamental es conseguir la serializabilidad a base de que dos transacciones T_1 y T_2 que se interfieren, T_1 (por ejemplo) accede a todos los gránulos comunes antes que T_2. Esto se consigue asignando una hora (TIME - STAMP) distinta a cada transacción y al leer (grabar) un gránulo comprobar si ha sido (leído o grabado) ya por otra más joven.

RESERVA Si una transacción T_1 quiere hacer determinadas acciones sobre un gránulo G, tiene que pedir mediante una instrucción (**LOCK**) y conseguir una reserva de determinado tipo.
T_1 solo puede obtener la reserva si no existe ninguna otra transacción T_2 que haya obtenido antes una reserva todavía vigente sobre G que sea incompatible con el tipo de reserva que pide T_1.

Problemas: **DEADLOCK, UNLOCK.**..

Diferencia

Time - Stamping : Sincroniza de tal forma que sea equivalente a una ejecución en serie específica.

Reserva: Sincroniza de tal forma que sea equivalente a alguna ejecución en serie de esas transacciones.

BASES DE DATOS ESTADISTICAS

Solo admiten estudios estadísticos pero no permiten obtener información sobre individuos.

Problemas para salvaguardar la confidencialidad de los datos individuales.

PROHIBIR PREGUNTAS REFERENTES A UNA SOLA OCURRENCIA

1. Promedio de salarios de los habitantes de
   ----------que cumplan la condición----------
   - SI SOLO CUMPLE 1 INDIVIDUO
   - SI SOLO CUMPLEN 2 Y EL QUE PREGUNTA ES UNO DE LOS DOS

Limitar tamaño del conjunto

2. a) Promedio de salarios ---------- tales que ----------
   tamaño m
   b) Promedio de salarios ---------- tales que ----------
   tamaño m+1
   Todos los de a están en b

Limitar el tamaño de las intersecciones entre consultas.

TECNICAS DE DISEÑO

Objetivo: SATISFACER LAS NECESIDADES DE LOS USUARIOS EN TERMINOS DE - Completitud, Integridad, Rendimiento.

Teory and Fry (1982) presentan una metodología de diseño.

- ANALISIS DE REQUERIMIENTOS
  Identificar y describir los datos necesarios para la organización

- DISEÑO CONCEPTUAL
  Sintetizar las diferentes vistas de usuario y los requerimientos de información en un diseño global de B.D.

- SELECCION DEL S.G.B.D.
  Traducir el esquema conceptual en un esquema que pueda ser procesado por un S.G.B.D. concreto

- DISEÑO FISICO
  Diseño de los formatos de los registros almacenados, selección de acceso...

ANALISIS DE REQUERIMIENTOS

IDENTIFICAR Y DESCRIBIR LOS DATOS QUE NECESITAN LOS USUARIOS DE LA B.D. PARA SATISFACER NECESIDADES PRESENTES Y FUTURAS.

- ¿Qué vistas de usuario se necesitan?
- ¿Qué elementos se necesitan en las vistas?
- ¿Cuales son las claves que identifican las entidades?
- ¿Cuales son las relaciones entre los elementos?
- ¿Cuales son los requerimientos operacionales en cuanto a integridad, seguridad y tiempo de respuesta?

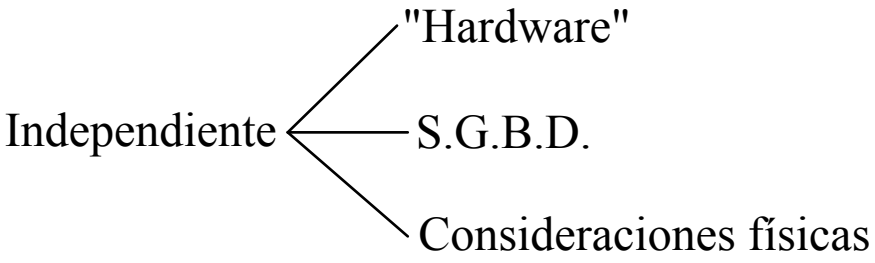
¿QUE DATOS SE UTILIZAN?

¿COMO SE UTILIZAN?

DISEÑO CONCEPTUAL

DESARROLLO DETALLADO DE UN "PLAN ARQUITECTONICO".

EL ESQUEMA CONCEPTUAL REPRESENTA LAS ENTIDADES DE LA ORGANIZACION, LOS ATRIBUTOS DE LAS ENTIDADES Y LAS RELACIONES ENTRE LAS ENTIDADES.



ALTERNATIVAS A ELEGIR:

- Covertura de la Base de Datos (1 única, varias)
- Tipos de Diseño: Top-down, Botton-up
- ¿Dependencia del SGBD?

MODELO ENTIDAD RELACION

Diseñado por Chen

Estructura de Datos: Entidades, Atributos, Relaciones

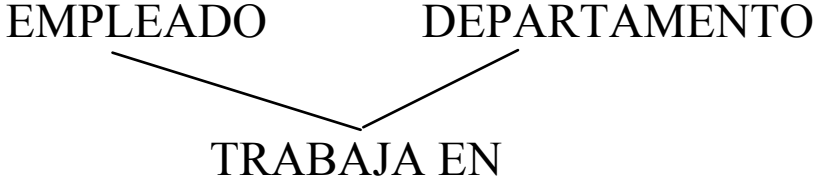
ENTIDAD: ALGO QUE LA PERSONA CONCEPTUALIZA COMO DISTINTO DE TODO LO DEMAS, Y ACERCA DE LO CUAL LE INTERESAN LAS PROPIEDADES.

EMPLEADO

ATRIBUTO: PROPIEDAD DE LA ENTIDAD
TOMAN SUS VALORES EN UN DOMINIO

EDAD

RELACION: ASOCIACION NO TRIVIAL ENTRE LOS COMPONENTES DE UN MODELO, CUYA NO INCLUSION EN EL MODELO SUPONE UNA PERDIDA DE INFORMACION.



SELECCION DEL S.G.B.D.

TRADUCCION DEL ESQUEMA CONCEPTUAL A UN ESQUEMA PROCESABLE POR UN S.G.B.D. CONCRETO.

Etapas a realizar:

- Traducción a un modelo Lógico (correspondiente a un S.G.B.D. concreto)

  JERARQUICO, EN RED, RELACIONAL

  Modos de realizarlo
  - * Manualmente
  - * Automáticamente

- Diseño de subesquemas
- Diseño de Programas

DISEÑO FISICO

DESARROLLAR UNA ESTRUCTURA DE BASE DE DATOS FISICA EFICIENTE. PROPORCIONAR UN BALANCE OPTIMO ENTRE RENDIMIENTO Y COSTO.

MODO EN EL QUE LOS DATOS ESTAN ALMACENADOS Y NO COMO SE PRESENTAN A LOS USUARIOS.

Etapas a realizar:

- Diseño de registros alamacenados
  - longitud
  - apuntadores
  - técnicas de compresión
- Agrupación de registros
  - interregistro
  - intraregistro
- Selección de métodos de acceso
  - indexación
  - aleatorización

Empresa constituida por dos secciones importantes:

- Almacén: se lleva el control de **productos** que hay en el almacén, los **proveedores** que sirven dichos productos y los **pedidos** que se realizan a los proveedores de los productos.

- Proyectos: se lleva el control de los **proyectos** que la empresa realiza y de los **trabajadores** asociados a dichos proyectos.

APLICACIONES

Almacen

- Informe alfabético de proveedores.
- Informe de proveedores por código.
- Informe de cumplimiento de los plazos de entrega de los proveedores.
- Informe de los productos por código.
- Informe de los productos por familias.
- Informe de los productos ofertados por proveedores.
- Informe de los descuentos para un producto de todos los proveedores.
- Informe de los productos de un pedido comprados a un precio superior al standard.
- Informe de los proveedores de los productos de una familia.
- Informe de pedidos.

Proyectos

- Informe alfabético de proyectos.
- Informe de proyectos en curso.
- Informe de proyectos finalizados.
- Informe del costo estimado de un proyecto.
- Informe del costo real de un proyecto.
- Informe de desviaciones de costos globales de un proyecto.
- Informe de personas que trabajan en proyectos.
- Informe de desviaciones de precios de los productos en los proyectos.
- Informe de Clientes que solicitaron un proyecto.
- Informe de precios standard de productos en un proyecto. Almacén
- Informe alfabético de proveedores.

2. Aspectos Lógicos y Físicos

ARQUITECTURAS DE TRES NIVELES

NIVEL INTERNO: El más cercano al almacenamiento físico.
Se encarga de cómo se almacenan los datos.

NIVEL EXTERNO: El más cercano a los usuarios.
Se encarga de cómo va a ver cada usuario los datos.

NIVEL CONCEPTUAL: Nivel intermedio.
Se encarga de dar una vista general de la BD a todos los usuarios. Es en este nivel donde se definen las reglas de integridad y seguridad.

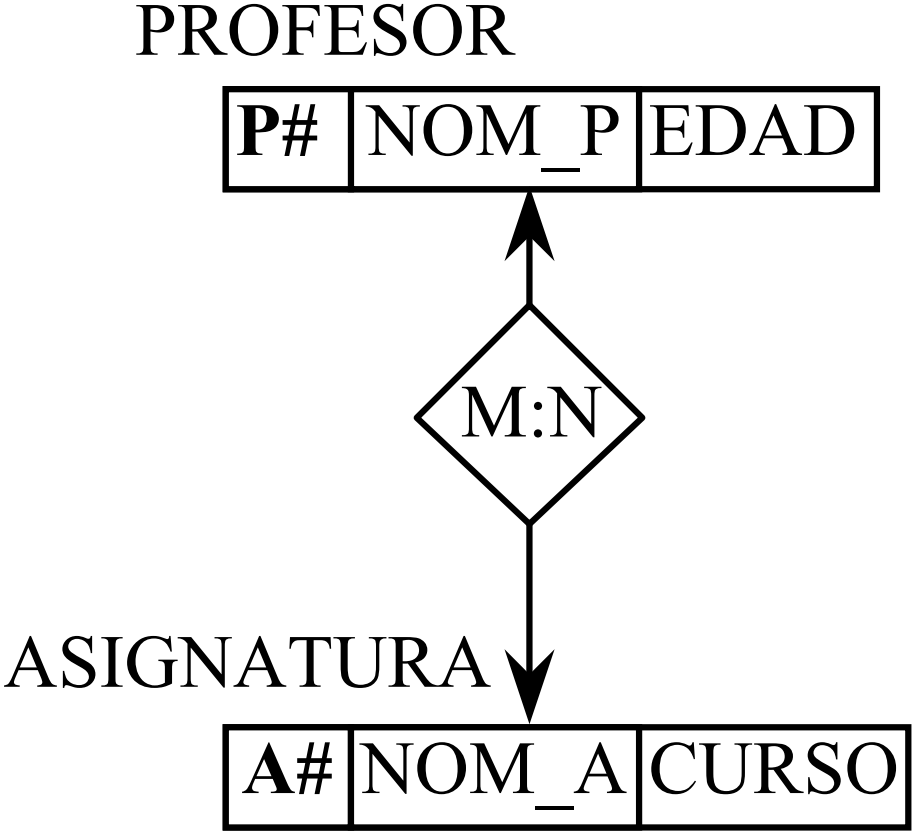
ANSI	NIVEL EXTERNO	NIVEL CONCEPTUAL	NIVEL INTERNO
CODASYL	SUBESQUEMA	ESQUEMA	ESQUEMA ALMACENADO
	ESTRUCTURA LOGICA	ESTRUCTURA LOGICA GLOBAL	ESTRUCTURA FISICA

OBJETIVOS

1. Obtener independencia lógica
   - Usar sólo entidades relevantes para una aplicación.
   - Redefinir un atributo existente usando un nuevo tipo de datos.
   - Renombrar una entidad o atributo.

2. Obtener independencia física
   - Modificar el modo en que están conectados los registros (puntero/contiguos).
   - Modificar el orden de las ocurrencias de registros (ascendente/descendente).
   - Modificar el nodo de acceso secuenciales: aleatorio/indexado.

EJEMPLO



NIVEL INTERNO

ASIGNATURA-DISK

A#	TYPE = BYTE(5), OFFSET=0, INDEX=AX
NOM-A	TYPE = BYTE(50), OFFSET=5,
CURSO	TYPE = FULLWORD,OFFSET=55

PROFESOR-DISK

P#	TYPE=BYTE(6), OFFSET=0, INDEX=PX
NOM-P	TYPE=BYTE(20), OFFSET=6,
EDAD	TYPE=FULLWORD, OFFSET=26

NIVEL CONCEPTUAL

ASIGNATURA

A#	CHARACTER(5)
NOM-ASIG	CHARACTER(50)
CUR-ASIG	NUMERIC(1)

PROFESOR

P#	CHARACTER(6)
NOM-PROF	CHARACTER (20)
EDAD	NUMERIC(2)

NIVEL EXTERNO

LEJONA (CONTRATOS)

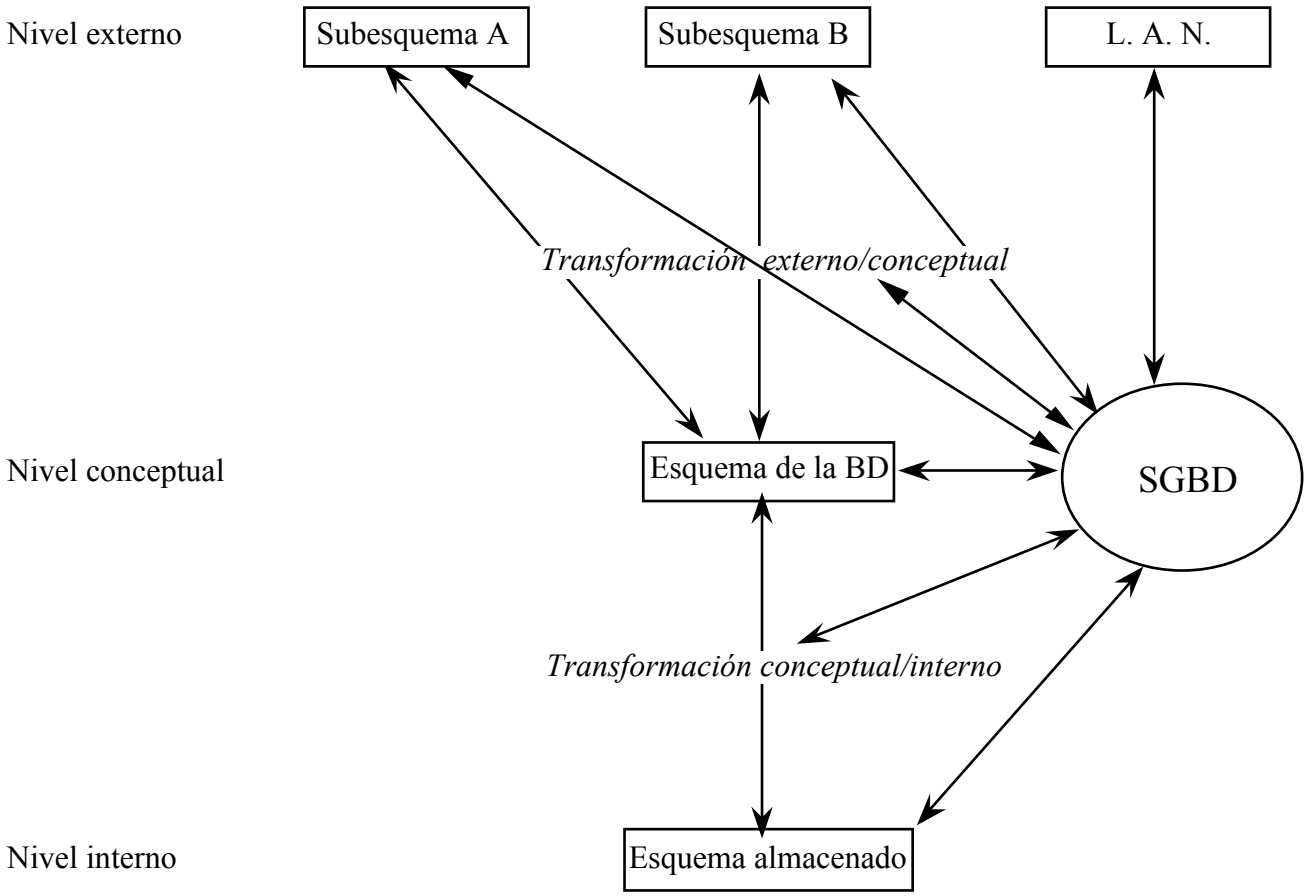
Obtener todos los profesores que están trabajando en la facultad.

SECRETARIA

Obtener las asignaturas que se imparten en 5º en el curso 90/91

LEJONA	SECRETARIA
PROF	ASIG
COD-PROF --> CHAR(6)	COD-ASIGNATURA --> character(6)
NOMBRE ---> CHAR(20)	NOM-ASIGNATURA --> character(50)
EDAD ---> Binario con coma fija	CURSO ---> character(1)

El SGBD (DBMS) se encarga de definir cada uno de los diferentes niveles, así como de atender a los requerimientos de los Lenguajes de alto nivel.



El SGBD debe asegurar el paso de los datos del formato correspondiente a un nivel al formato correspondiente a otro nivel. Mediante transformaciones o Mappings.

SGBD: - Software que se encarga de todos los accesos a la BD.
- Conjunto de Lenguajes, Procedimientos, etc., que suministra, tanto a los analistas, programadores o al ABD, los medios para describir, recuperar y manipular los datos integrados en la BD, asegurando las medidas de Integridad y confidencialidad.

FUNCIONES DE LOS SGBD

DESCRIPCION: LDD: Lenguaje de Definición de Datos.
Permite al ABD especificar
los elementos de datos que la integran,
su estructura y relaciones que existen entre ellos.
Reglas de Integridad, y
Reglas de seguridad.

MANIPULACION: LMD
Añadir,
buscar,
suprimir, o
modificar datos.
Siempre de acuerdo con las especificaciones dictadas por el ABD.

UTILIZACION: Utilidades para que los diferentes usuarios puedan comunicarse con la BD.

- Estadísticas
- Copias de seguridad
- LOG de transacciones (dietario, auditorías)

AUTOSUFICIENTES

LDD

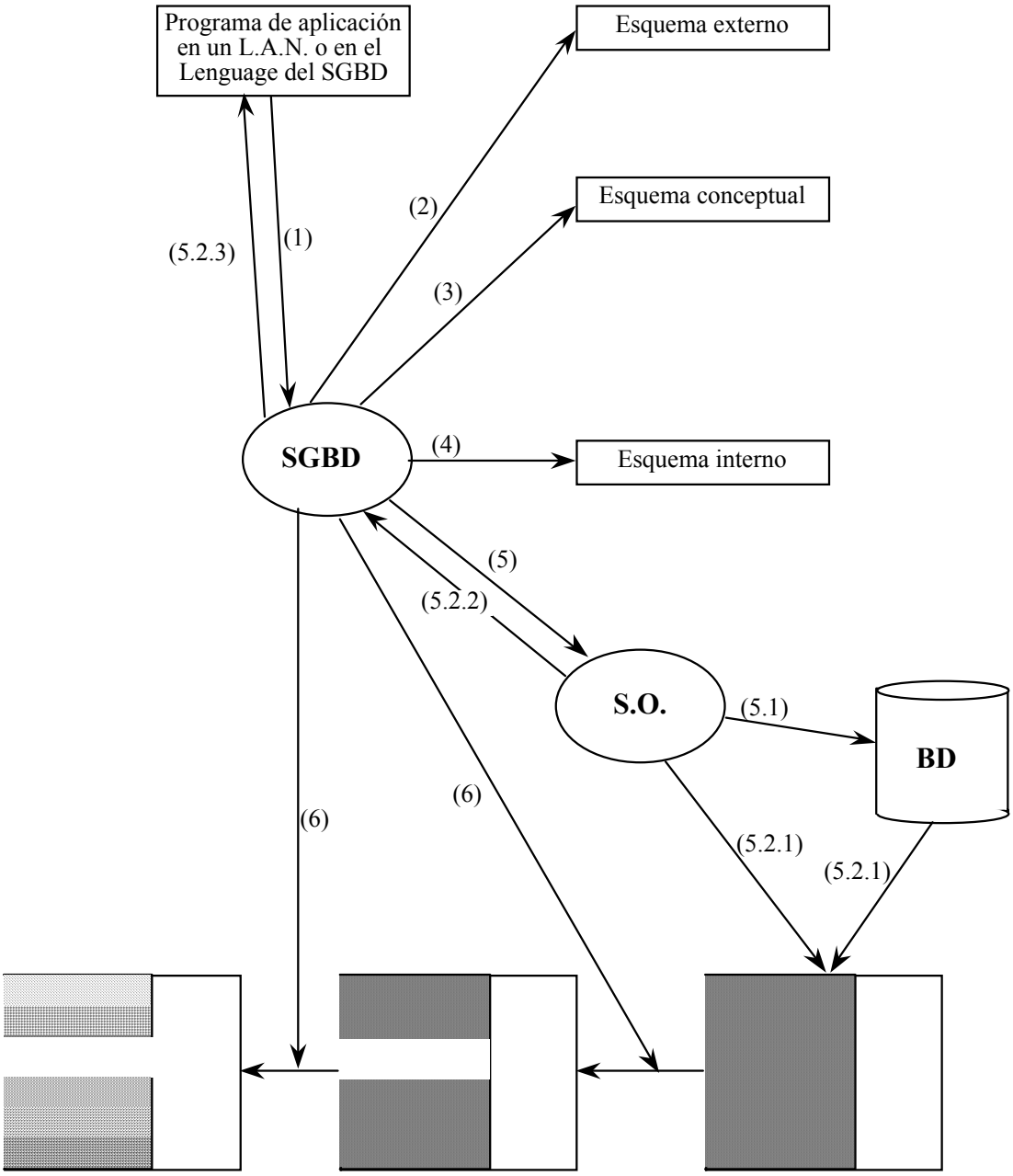
TIPO HUESPED

A) DE SUBESQUEMAS: Y ESQUEMAS
   - Definir tipos de entidades y campos
   - Especificar orden de los campos
   - Definir dominios (integer,…) y rangos (0..100) de campos
   - Por clave de acceso

B) ESQUEMAS ALMACENADOS
   - Asignar ficheros
   - Especificar estructuras de almacenamiento

	AUTOSUFICIENTES	PROCEDURALES
LMD		
	HUESPED	NO PROCEDURALES O ASERCIONALES

- ABRIR
- CERRAR
- INSERTAR
- BORRAR
- MODIFICAR
- BUSCAR
- OBTENER


Programa de aplicación en un L.A.N. o en el Lenguage del SGBD
Esquema externo
Esquema conceptual
Esquema interno
SGBD
S.O.
BD
(1)
(2)
(3)
(4)
(5)
(5.1)
(5.2.1)
(5.2.2)
(5.2.3)
(6)

LECTURA SIN CONFLICTOS POR CONCURRENCIA

1. Aplicación envía petición al SGBD.
2. Consulta esquema externo para verificar posibilidad de acceso a los datos y ver que tipo tienen esos datos.
3. Consulta esquema conceptual para deducir el tipo lógico de los datos.
4. Consulta esquema interno para deducir la unidad física a leer.
5. Petición de lectura al S.O.
   (5.1) S.O. Busca en la unidad física solicitada
   (5.2.2) S.O. Devuelve error.
   (5.2.3) SGBD trata el error y/o devuelve el error a la aplicación.
   (5.2.1) Devuelve el S.O. Los datos en la Memoria de trabajo.
6. SGBD selecciona los datos referidos por la aplicación y realiza las transformaciones necesarias de los datos hasta que se queden en el formato especificado por el esquema externo.

ORGANIZACION INTERNA DE LOS DATOS

Almacenar una BD físicamente significa almacenar datos en ficheros compuestos por ocurrencias registros de idéntico formato (registro).

Formato: lista de nombres de campos, donde cada campo utiliza un nº fijo de bytes y un tipo de datos fijo.

FORMATO:

NOMBRE	APELLIDO1	APELLIDO2	EDAD

Una **ocurrencia de registro**: valores asociados a cada uno de los campos.

OCURRENCIA DE REGISTRO:

JOAQUIN	OCHOA	IBAÑEZ	62

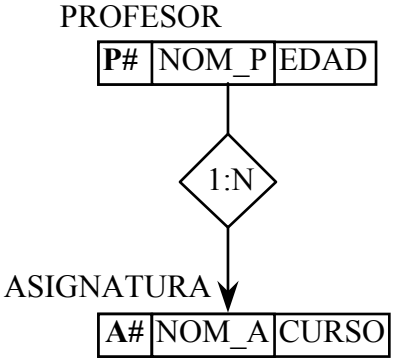
PROBLEMA

¿Cómo relacionar los datos que están almacenados?

Existen 2 posibles soluciones:

A) ESTRUCTURA INTRAREGISTRO:

- •Los datos relacionados se graban de forma contigua.
- •1 acceso a disco para encontrar ocurrencias de registros relacionados con una determinada ocurrencia.



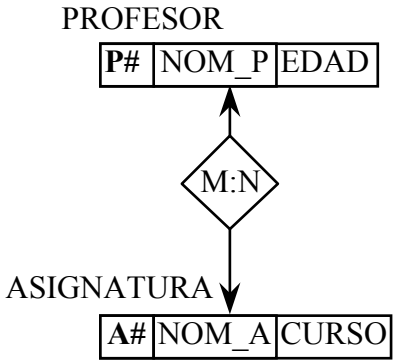
P1	A1	A4	P3	A2	A1	A3	A4	A5	A8	A7	P2	A2	A3	A5	A6	A9

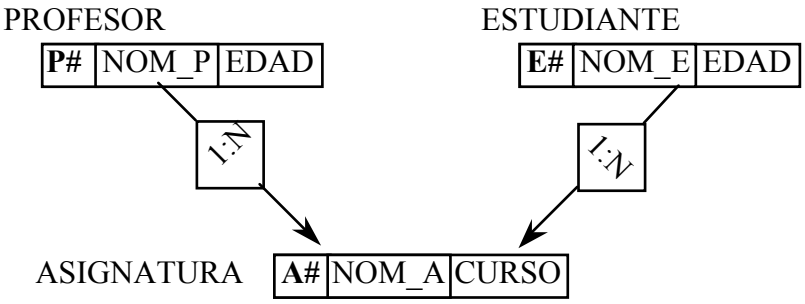
INCONVENIENTES

Este tipo de estructuras sólo admiten jerarquías. ¡NO se admiten grafos!

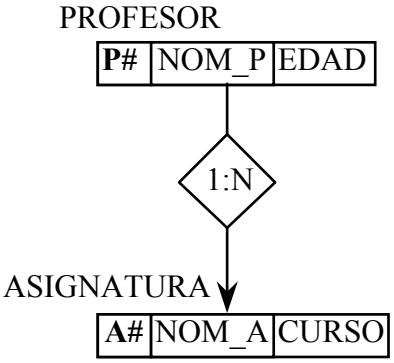
Hay un Dato principal, y luego están los datos que dependen de éste y así sucesivamente.

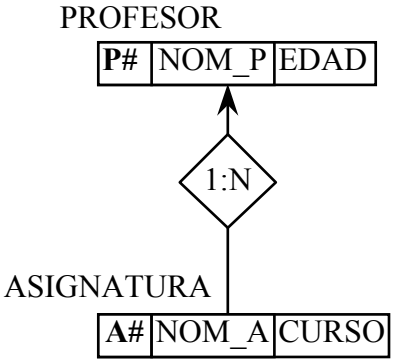
Así no se pueden representar de forma satisfactoria relaciones de la forma:





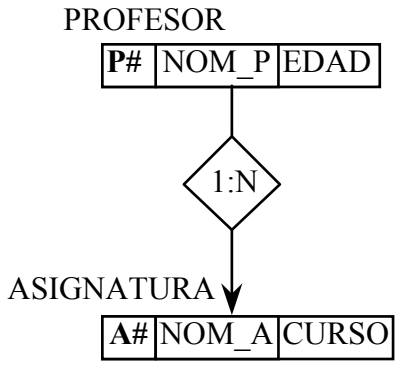
P1	A1	A4						
E3	A2	A1	A3	A4	A5			
E2	A2	A3	A5	A6	A9			


PROFESOR
P#
NOM_P
EDAD
1:N
ASIGNATURA
A#
NOM_A
CURSO


PROFESOR
P#
NOM_P
EDAD
1:N
ASIGNATURA
A#
NOM_A
CURSO

A.1) Registros de longitud fija:

Hay que reservar espacio para la mayor ocurrencia de registro.



Se supone que no pueden existir más de 7 Asignaturas relacionadas con cada Profesor.

P1	A1	A4					
P3	A2	A1	A3	A4	A5	A8	A7
P2	A2	A3	A5	A6	A9		

A.2) Registros de longitud variable

- Mayor complejidad a la hora de insertar y borrar en la BD.
- Problemas para aprovechar el espacio (tras un borrado): El espacio ocupado por un registro borrado sólo puede ser reemplazado por otro registro del mismo tipo que el primero.

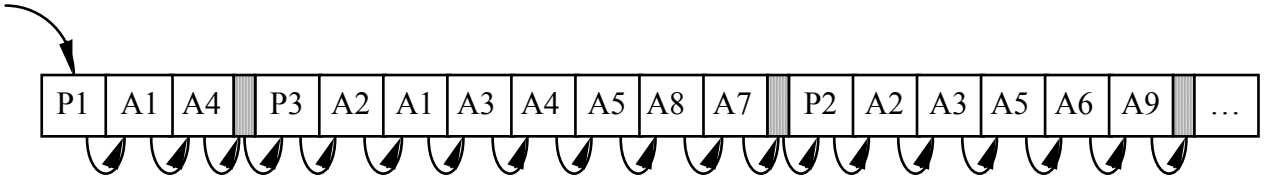
Problema:

¿dónde termina una ocurrencia de registro y comienza la siguiente?

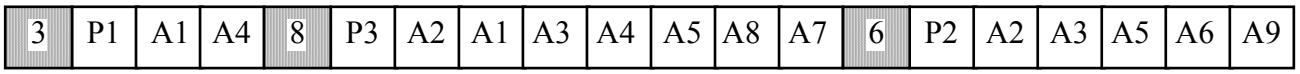
A2.1) Marcar con un carácter especial al final de cada registro.

P1	A1	A4		P3	A2	A1	A3	A4	A5	A8	A7		P2	A2	A3	A5	A6	A9		…

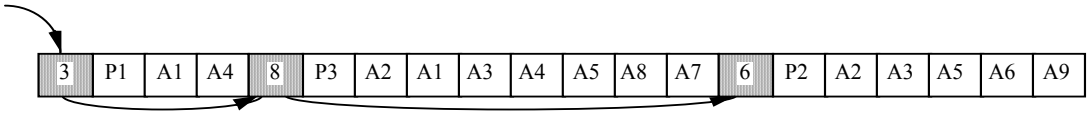
Acceso secuencial a los datos, ya que no sabemos donde puede estar el carácter que nos indique el final del registro.



A2.2) Tener un campo que dé la longitud de cada registro.



- Ventaja sobre el anterior: Acceso aleatorio a los datos menos secuencial; ya que al leer en la cabecera los datos, sabemos que nº de bytes nos podemos saltar en el fichero en el anterior no se sabe cuando se acaba un registro, luego hay que leer todo el fichero hasta llegar al lugar deseado.



B) ESTRUCTURA INTERREGISTRO

Para establecer las relaciones se utilizan los apuntadores (directos o indirectos).
Varios accesos a disco
Se puede representar combinaciones de árboles elementales.

B.1) Apuntadores directos del padre a cada hijo:

B1.1) Matriz de apuntadores

Dato

P3	•	•	•	•	•	•	•

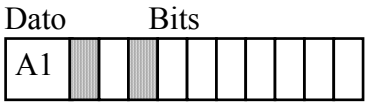
Apuntadores

Se puede indicar el orden de consulta a datos.
2 posibilidades:
1 - apuntadores a B, apuntadores a C etc.
2 - apuntadores en general + información además del puntero.

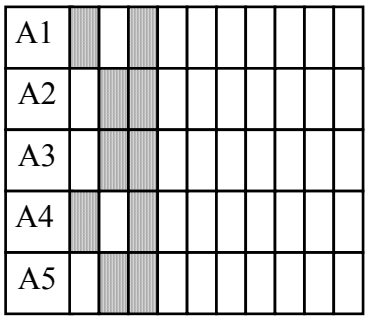
P1	•	•	•	•	•	•	•
P2	•	•	•	•	•	•	•
P3	•	•	•	•	•	•	•

A1	•	•	•	•	•
A2	•	•	•	•	•
A3	•	•	•	•	•
A4	•	•	•	•	•
A5	•	•	•	•	•
A6	•	•	•	•	•
A7	•	•	•	•	•
A8	•	•	•	•	•
A9	•	•	•	•	•

B1.2) Matriz de bits

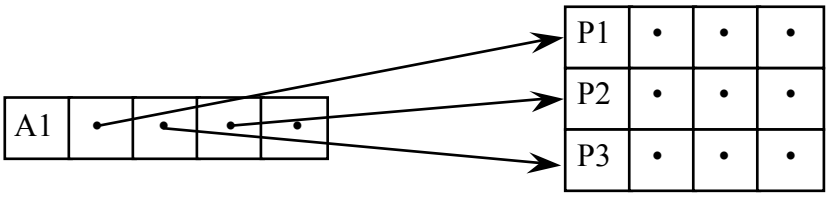


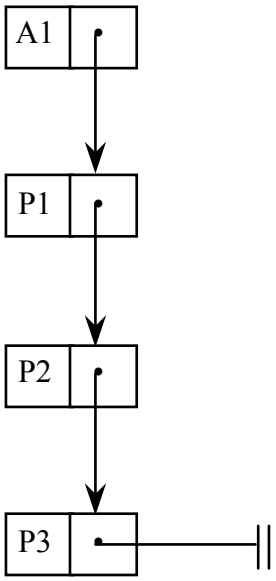
El orden de consulta viene dado por el orden en el fichero.



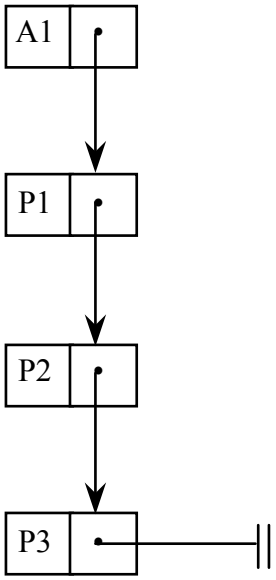
B.2) Del padre al primer hijo y los hermanos entre sí

- Se pierde eficiencia en cuanto al tiempo ya que no están tan accesibles los datos como antes.

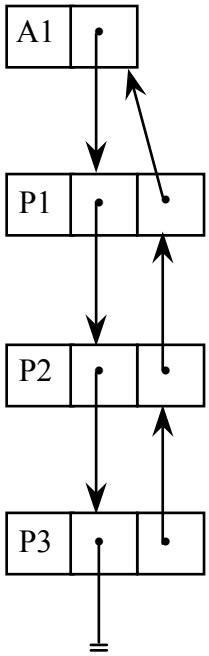




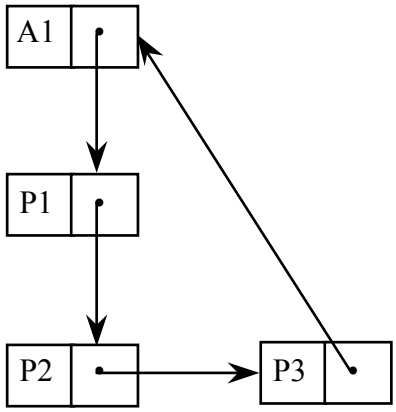
B2.1) Cadenas de 1 sentido



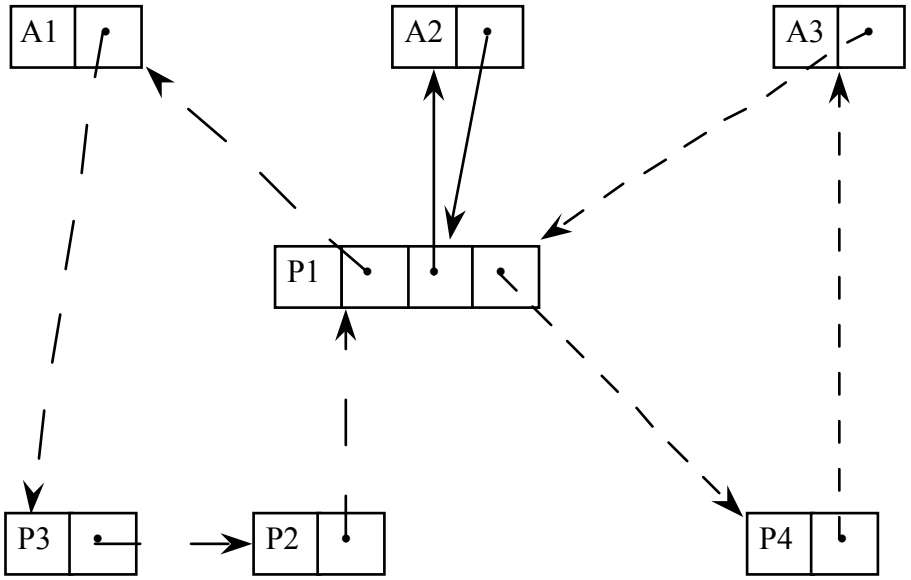
B2.2) Cadenas de doble sentido



B2.3) Anillos simples



B2.4) Anillos múltiples



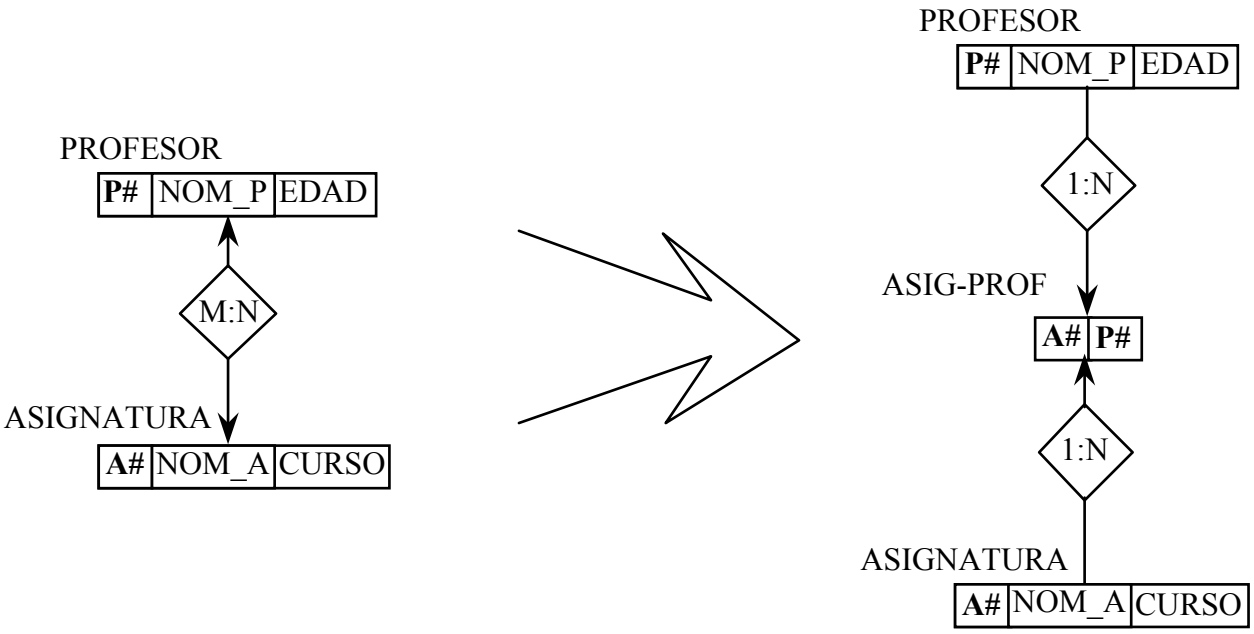
Cuando aparecen estructuras como son los anillos múltiples, es porque existen relaciones de un mismo registro con varias ocurrencias de entidad.

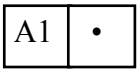
-Hay ocurrencias de registro que necesitan más apuntadores que otras.

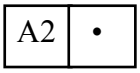
-nº variable de apuntadores ---> difícil de implementar

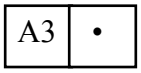
Ai y P2,P3,P4: 1 apuntador;
P1: 3 apuntadores.

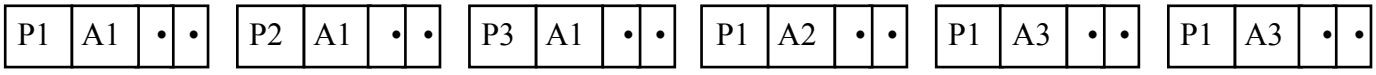
Solución:

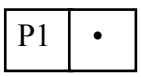


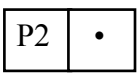


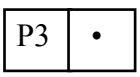


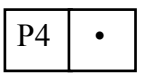










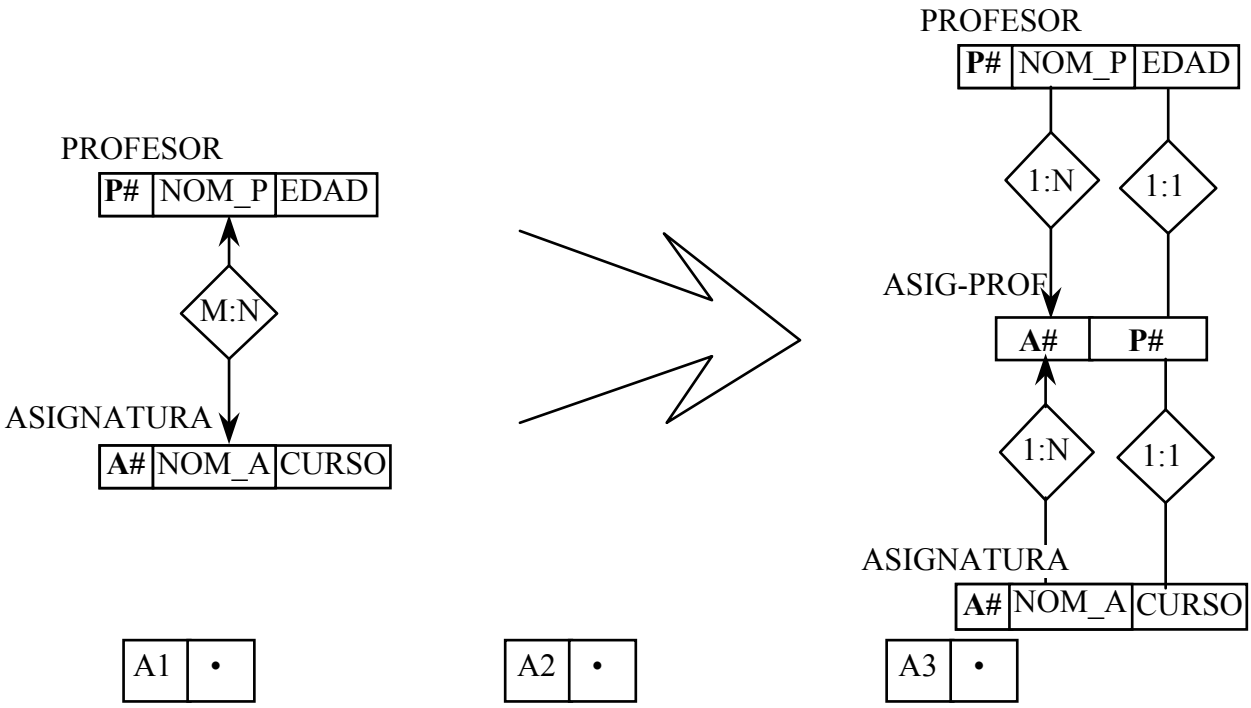


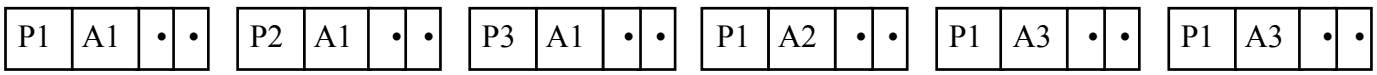
Ai, Bi: 1 apuntador

AiBi: 2 apuntadores.

ÁRBOLES FANTASMAS

- Apuntador desde cada hijo al padre.
- Mejora tiempos de respuesta.
- Complica las estructuras (añade 1 puntero más por ocurrencia de registros).
- Se puede añadir un árbol fantasma por relación





Ai y Pi: 1 apuntador;

AiPi: 4 apuntadores.

TECNICAS DE ACCESO

INDEXACION

SON 2:

ALEATORIZACION

ASPECTOS RELEVANTES:

1) **Tiempo de Acceso**: tiempo para localizar un dato.

2) **Tiempo de inserción**: tiempo para introducir 1 dato;
   - encontrar el lugar
   - modificar el fichero índice (indexación)

3) **Tiempo de Borrado**: tiempo para borrar un dato.
   - encontrar el lugar
   - modificar el fichero índice

4) En indexación: **espacio adicional necesario** para el fichero índice.

TECNICAS DE INDEXACION

- Se basan en un fichero auxiliar llamado índice.

CIUDAD
Almería
Donostia
Madrid
Zaragoza

Fichero Indice

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	Luis	Donostia
40	Ana	Almería
17	María	Zaragoza

Fichero indexado

Fichero índice: Fichero auxiliar para acceder a registros de otros ficheros. Cada elemento del fichero índice se llama entrada y tiene el valor de un DATO correspondiente al valor de un campo del fichero indexado y un apuntador el fichero indexado.

USO DE LOS INDICES

- ACCESO DIRECTO: Permite acceder a registros individuales en el fichero indexado.

- ACCESO SECUENCIAL: Recorre el fichero indexado en el orden del fichero índice.

- PREGUNTAS TIPO TEST: Para responder acudiendo al fichero índice.

VENTAJAS

- Menor E/S puesto que el fichero índice siempre será menor que el indexado
- Rapidez en la búsqueda
- Se sabe cuando parar una búsqueda (el fichero índice está indexado)

DESVENTAJAS

- Hay un fichero más:
  - insertar y borrar en 2 ficheros
  - se necesita más espacio de almacenamiento

FICHERO CON INDICES: Fichero indexado con varios índices sin que llegue a tener un índice para cada campo.

FICHERO INVERTIDO: Fichero indexado con un índice para cada campo.

EDAD
17
18
25
40

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	Luis	Donostia
40	Ana	Almería
17	María	Zaragoza

CIUDAD
Almería
Donostia
Madrid
Zaragoza

NOMBRE	Ana	Juan	Luis	María

FICHERO INDICE DENSO: Una entrada para cada ocurrencia del registro del fichero indexado. (el fichero indexado no necesita ordenación)

FICHERO NO DENSO: No existe una entrada para cada valor.
El fichero tiene que estar ordenado por el mismo campo que el fichero índice.

CIUDAD
Almería
Donostia
Madrid
Madrid
Zaragoza

EDAD	NOMBRE	CIUDAD
40	Ana	Almería
18	Luis	Donostia
17	Pedro	Madrid
25	Juan	Madrid
17	María	Zaragoza

CIUDAD
Almería
Madrid
Zaragoza

no denso

Otros:

EDAD
17
18
25
40

Denso

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	Luis	Donostia
40	Ana	Almería
17	María	Zaragoza
17	Pedro	Madrid

CIUDAD
Almería
Donostia
Madrid
Zaragoza

Denso

INDEXACION POR COMBINACION DE CAMPO

El fichero índice tiene varios campos en lugar de uno sólo.

NOMBRE	CIUDAD
Juan	Madrid
Luis	Donostia
Ana	Almería
María	Zaragoza

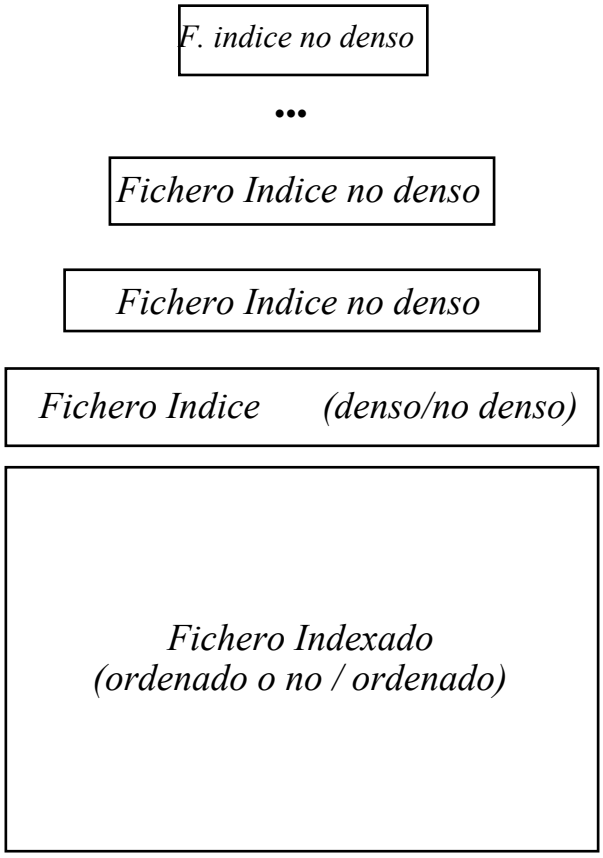
EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	Luis	Donostia
40	Ana	Almería
17	María	Zaragoza

Un índice de combinación de N campos C_1... C_n puede hacer veces de un índice.

- del campo C_1
- del campo C_1 C_2
- ...
- del campo $C_1, C_2, ..C_n$

JERARQUIAS DE INDICES

Si un fichero índice es muy grande, se pierde eficiencia al recorrerlo. Una solución: crear un fichero sobre un índice. Así estamos definiendo una jerarquía de índices.



- Número fijo de niveles

  Existen mejoras de las jerarquías de índices como son los

  1) B-trees que hacen variable el nº de niveles y ahorra algo de espacio en los diferentes niveles del B-tree.

  2) B^+-trees es una mejora del B-tree que soluciona el recorrido secuencial del fichero indexado (fallo del B-tree).

  3)B*-trees mejora del B^+-tree: ahorra más espacio que el B^+-tree.

TECNICAS DE ALEATORIZACION

Se basan en un algoritmo (algoritmo de aleatorización) que determina la posición de un registro en función del contenido de un campo o conjunto de campos (clave de aleatorización).

DIFERENCIAS FRENTE A INDEXACION

- Un fichero puede tener definido cualquier número de índices pero sólo podrá tener definida una clave de aleatorización.

- Más rápido (se evita tener que acceder al fichero índice).

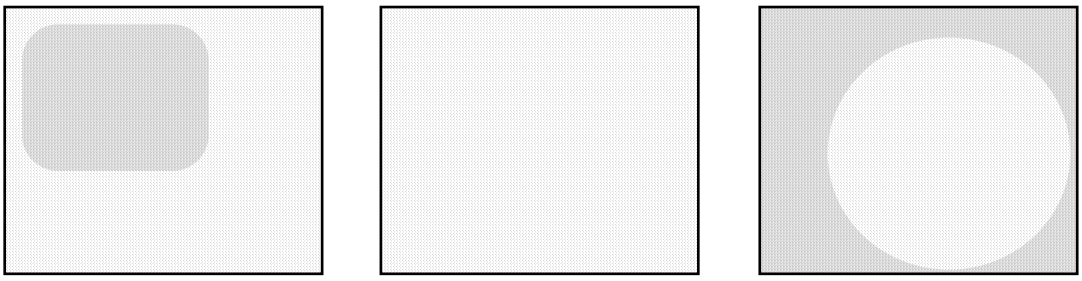
Idea inicial: trabajar el algoritmo de aleatorización = función identidad.

---> Perdida de espacio de almacenamiento: ---> no válido
• DNI

Sinónimos: valores de la clave a los cuales el algoritmo hace corresponder la misma posición.

CARACTERISTICAS DEL ALGORITMO DE ALEATORIZACION

1. **Distribución uniforme** de los valores de la clave.



2. Ejecución rápida
   Al leer o grabar hay que calcular la posición.

3. Convertir valores no numéricos a numéricos
   Char ---> posición
   Char + lógico + fecha ---> posición

CUBO (BUCKET):

Posición que devuelve al algoritmo de la aleatorización con cabina para más de una ocurrencia de registro.

- Disminuye el nº de sinónimos

MANEJO DE SINONIMOS

¿Dónde ponerlos?
¿Cómo relacionarlos?

¿DONDE PONERLOS?

I) Todos los cubos son del mismo tipo. (cubos primarios)
II) Hay cubos especiales para sinónimos.

IA) **Sinónimos errantes**

Sacar hacia otro cubo uno de los sinónimos y poner el nuevo registro en la posición que ha quedado libre. (sólo si existe un sinónimo que no sea sinónimo del que vamos a meter).

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	José	Donostia
40	Jeremías	Almería
17	Felipe	Zaragoza

Se inserta Jesús, Alava,32

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	José	Donostia
40	Jeremías	Almería
32	Jesús	Alava
…	…	…
17	Felipe	Zaragoza

IB) Respetar derechos adquiridos.

Al registro a insertar se considera como sinónimo del que se encuentra en el cubo.

EDAD	NOMBRE	CIUDAD
25	Juan	Madrid
18	José	Donostia
40	Jeremías	Almería
17	Felipe	Zaragoza
…	…	…
32	Jesús	Alava

IIa) Cubos de sinónimos distribuidos entre los cubos primarios:

P	S	P	S	P	S	P	S	P	S	P	S	P	S	P	S

IIb) Zona especial para los cubos de sinónimos:

P	P	P	P	P	P	P	S	S	S	S	S	S	S	S	S

PROBLEMAS DE LA ALEATORIZACION CONSTANTE (tamaño tipo de la B.D.)

- No permite aumentar el tamaño de la BD sin reorganizar toda la BD.
- Sólo existe una única clave de aleatorización.
- La distribución no se corresponde con los sucesivos valores de la clave.

ALEATORIZACION VARIABLE

Surge para solucionar el primer problema. Se consigue añadiendo un cubo cuando se llena otro y redistribuyendo los registros de uno o más cubos ya existentes junto con el nuevo cubo.

Se distinguen 2 tipos:

a) Aleatorización variable con directorio

Cuando se produce un sinónimo se desdobla el cubo en cuestión, que debe ser cualquiera, por lo que se necesita un directorio que nos señala qué cubos están desdoblados y cuál es la dirección física de los resultados del desdoblamiento. (Dentro de este tipo hay una técnica que se ve con más profundidad que es la ALEATORIZACION EXTENSIBLE)

b) Aleatorización variable sin directorio /*los cubos se desdoblan en un cierto orden*/

A cada sinónimo que se produce, se desdobla un cubo según un orden establecido. No hace falta un directorio, porque basta con un campo que indique el último cubo desdoblado, para saber qué cubo están desdoblados y cuales no. Pero se necesita espacio para los sinónimos, que serán temporales, porque irán desapareciendo en sucesivos desdoblamientos.

ALEATORIZACION EXTENSIBLE

Características:

Admite cambios en el tamaño de la BD desdoblando los cubos a medida que se llenan.

Inicialmente, el directorio consta de una sola entrada que aporta al único cubo que hay.

Cuando se produce el primer sinónimo se añade un nuevo cubo y los 2 se reparten los registros de la siguiente forma: Al primer cubo van aquellos registros a cuyos valores de la clave corresponden series de bits que empiecen con 0 mientras que los demás (empiezan con 1) van al otro CUBO.

El directorio pasa a tener 2 entradas, que apuntan a cada uno de los cubos.

Cuando se vuelva a llenar uno de los cubos, se repartirán los registros con un nuevo cubo, basándose en el segundo bit de la serie respectiva. El directorio pasará a tener 4 entradas que apuntarán respectivamente a los cubos donde están los registros cuyas series de bits empiecen por 00, 01, 10 y 11.

Desdoblamiento:

Problema: El paso de los registros de un cubo a otro en los desdoblamientos.

Ventaja: Se aprovecha muy bien el espacio

Entrada:

CLAVE	EDAD	NOMBRE	CIUDAD
0010	25	Juan	Madrid
1011	40	Ana	Almería
0011	18	Luis	Donostia
0111	17	María	Zaragoza

Proceso:

CLAVE	EDAD	NOMBRE	CIUDAD

CLAVE	EDAD	NOMBRE	CIUDAD
0010	25	Juan	Madrid

CLAVE	EDAD	NOMBRE	CIUDAD
0010	25	Juan	Madrid
1011	40	Ana	Almería

0	1

CLAVE	EDAD	NOMBRE	CIUDAD
0010	25	Juan	Madrid
0011	18	Luis	Donostia
1011	40	Ana	Almería

00	01	10=11	11=10

CLAVE	EDAD	NOMBRE	CIUDAD
0010	25	Juan	Madrid
0011	18	Luis	Donostia
0111	17	María	Zaragoza
1011	40	Ana	Almería

¿COMO RELACIONAR LOS SINONIMOS?

A) MEDIANTE APUNTADORES ENCADENADOS

Se crea una cadena entre los sinónimos. Se pueden borrar registros (en este caso se modifican los apuntadores).

CLAVE	EDAD	NOMBRE	CIUDAD	SINONIMO
0010	25	Juan	Madrid	•
1011	40	Ana	Almería	•
0011	18	Luis	Donostia	•
0111	17	María	Zaragoza	•

B) MEDIANTE DIRECCIONAMIENTO ABIERTO

Cuando el cubo C1 asignado por el algoritmo está lleno, se intenta colocar el sinónimo en otro cubo C2 fijado de antemano. Si C2 está lleno se va a otro cubo C3, y así sucesivamente.

Para que el algoritmo funcione, no se pueden borrar los registros porque se perderá la pista a los registros almacenados a continuación.

ALEATORIZACION POR UNA CLAVE COMPUESTA POR VARIOS CAMPOS

2 situaciones posibles.

A) TODOS LOS CAMPOS FORMAN LA CLAVE CONJUNTA

Clave = C1...Cn

C1	C2	C3	…		Cn

Esto significa que si se desconoce el valor de algún Ci, sólo se podría localizar el registro de forma secuencial.

B) ALEATORIZACION PARTICIONAL:

Idea similar a un fichero indexado por varios campos.

Ejemplo

C1 (=NOMBRE)---> permite obtener 3 bits de la dirección de un cubo

C2 (=CIUDAD)---> permite obtener 5 bits de la dirección de un cubo

APLICACION 1

"DADO UN NOMBRE, OBTENER TODAS SUS CIUDADES"

(CLAVE) ---> X X X _ _ _ _ _
NOMBRE

habrá que recorrer 2^5 valores distintos para contestar a la pregunta.

APLICACION 2

"DADO UNA CIUDAD, OBTENER TODOS LOS NOMBRES"

_ _ _ X X X X X
CIUDAD

Habrá que recorrer 2^3 valores distintos para contestar a la pregunta.

APLICACION 3

"OBTENER LA INFORMACION PARA UN NOMBRE Y UNA CIUDAD"

X X X X X X X X

Se puede acceder directamente al cubo si no hay sinónimos.

3. Modelos de Datos

MODELO: Instrumento teórico que permite representar los datos de un organismo.

COMPONENTES DE UN MODELO DE DATOS

1. Estructuras de datos
2. Operaciones sobre estas estructuras
3. Reglas o restricciones de integridad

1. Estructuras de datos

   Las ocurrencias de los datos no son independientes las unas de las otras. Forman estructuras. Ej: árboles, redes, tablas,...

2. Operaciones

   Se necesitan operaciones para manejar dichas estructuras: para consultar, borrar, actualizar datos.

3. Reglas de integridad

   Permiten especificar qué estados de la base de datos son consistentes.

FAMILIAS DE MODELOS

Han ido apareciendo, a lo largo del tiempo, varios modelos que se pueden clasificar en familias.

a) Familias Pre-Relacionales
   - Modelos invertidos
   - Modelos jerárquicos
   - Modelos en red

b) Familia Relacional

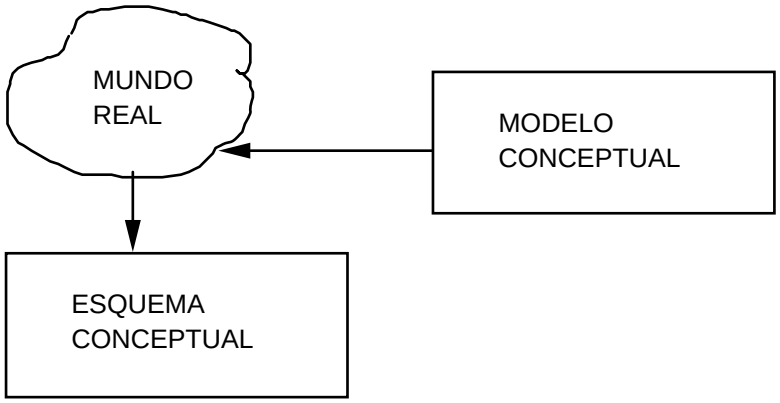
c) Familias Post-Relacionales
   - Modelo entidad-relación
   - Modelos semánticos
   - Modelos funcionales
   - Modelos orientados a objetos

MODELO CONCEPTUAL

a) Mundo Real. En este mundo existen cosas (Ej: personas, animales, coches...) y se producen acontecimientos (Ej: Las personas nacen, mueren, trabajan, compran coches, tienen animales, ...)

b) Mundo de las concepciones. Cada observador percibe el mundo real y lo conceptualiza a su manera. Distingue ciertas cosas como entidades, que tienen atributos y que se interrelacionan entre sí.

c) Mundo de las representaciones.
Toda representación tiene un aspecto lógico: palabras, códigos, ... y un aspecto físico: el soporte de los datos.



OCURRENCIA DE ENTIDAD: Algo que la persona conceptualiza como distinto de todo los demás, y acerca de lo cual le interesan propiedades

ENTIDAD: Conjunto de ocurrencias acerca de las cuales le interesan los mismos atributos, las mismas propiedades.

ATRIBUTO: Propiedad de una entidad. Los atributos toman sus valores en un dominio.

Ejemplo: ENTIDAD : PERSONA
ATRIBUTOS : NOMBRE DOMINIO : CONJUNTO DE STRINGS
EDAD DOMINIO : NUMEROS NATURALES
SEXO DOMINIO : { H, V }
OCURRENCIA DE ENTIDAD : < AMAIA, 22, H >

DATO ELEMENTAL: Es el valor que toma un atributo de una determinada ocurrencia de entidad.
Hay que conocer cómo se representan y cómo se interpretan estos valores.

Ejemplo: "SER MUJER" se representa como H
"SER HOMBRE" se representa como V

CLAVE: Atributo tal que a cada ocurrencia de entidad le hace corresponder un valor diferente. Sirve para distinguir las ocurrencias de una entidad.

Si existen varios atributos clave se escoge uno como clave primaria y los otros quedan como claves secundarias o alternativas.

INTER-RELACIONES

Una inter-relación entre las entidades E_1, E_2, ..., E_N, es un subconjunto del producto cartesiano E_1 x E_2 x...x E_N (Es un conjunto relación).

Ejemplo:

PERSONAS(NOMBRE,...)	COCHES(NOMBRE,...)	PER-COC
< PEIO,... >	< R-5,... >	<PEIO,... > < R-5...>
< AMAIA,... >	< GOLF,... >	< AMAIA,... > < R-5,... >
< ANA,... >		< AMAIA,... > < GOLF,... >

Cada tupla (e_1, e_2, ..., e_N) del conjunto inter-relación expresa que existe una relación entre las ocurrencias e_1 □ E_1, e_2 □ E_2, y e_N □ E_N.

Una relación es una asociación no trivial entre los componentes de un modelo, es decir una asociación cuya no inclusión en el modelo supone una pérdida de información.

asociación trivial: LE-GUSTARIA-QUE-LE-REGALARAN es PERSONAS x COCHES

GRADO DE LAS INTER-RELACIONES

La inter-relación E entre E_1, y E_2 es de grado

1:1 Si a una tupla de E_1 le corresponde a lo sumo una de E_2, y a una de E_2 le corresponde a los sumo una de E_1,

1:N Si a una tupla de E_1 le pueden corresponder varias de E_2
A una de E_2 le corresponde a los sumo una de E_1

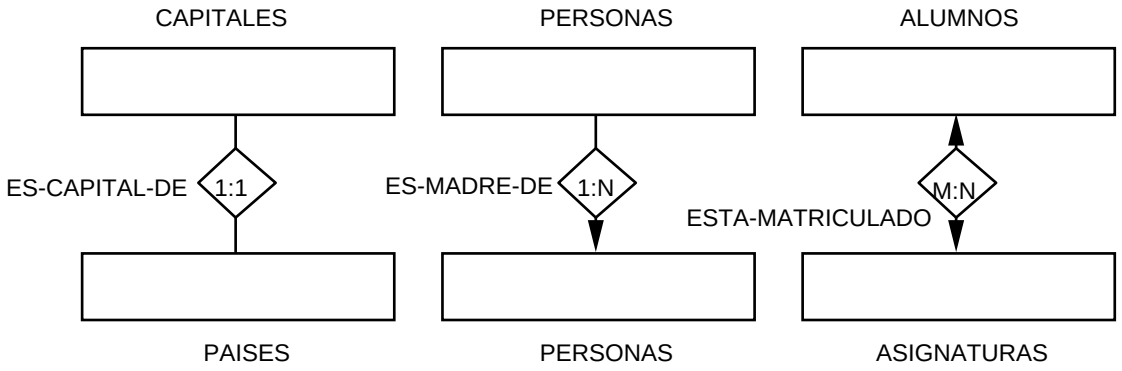
M:N Si a una tupla de E_1 le pueden corresponder varias de E_2
Si a una tupla de E_2 le pueden corresponder varias de E_1

Ejemplo:

1:1 Inter-relación es-capital-de entre capitales y países.

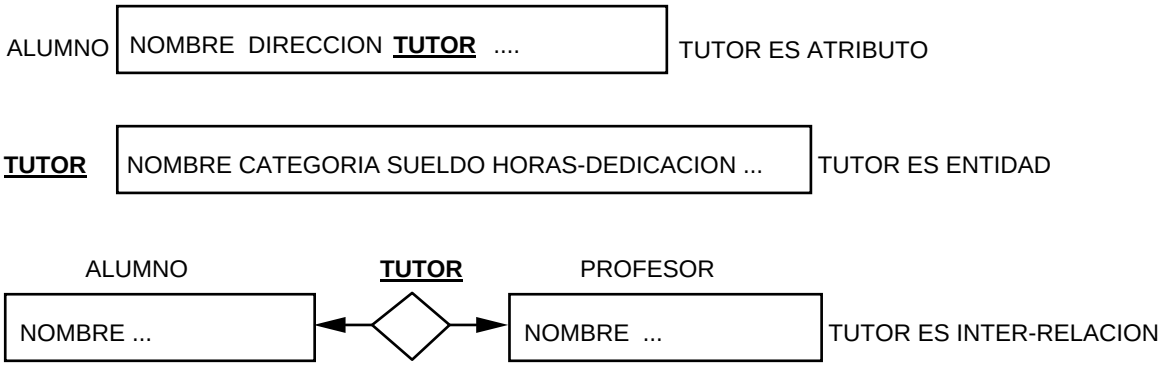
1:N Inter-relación es-madre-de entre personas y personas.

M:N Inter-relación está-matriculado entre alumnos y asignaturas.



EXISTEN PROBLEMAS EN LA DISTINCION DE ATRIBUTOS, ENTIDADES, INTER-RELACIONES

Cada observador tendrá que decidir si representa su información en forma de entidades, de atributos o de inter-relaciones.

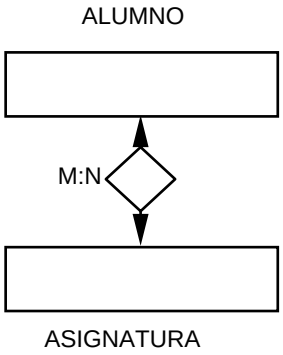


MODELO JERARQUICO

1. ESTRUCTURAS DE DATOS

Conjunto ordenado de ocurrencias de tipo árbol.

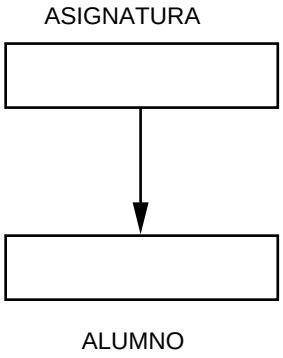
Ej: Dadas las dos entidades y la inter-relación siguientes;



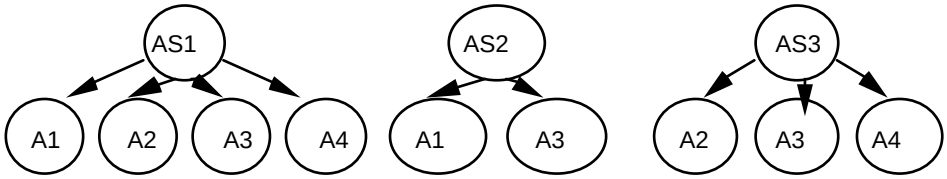
En el modelo jerárquico hay que ver este modelo conceptual como si fuera un árbol.

Hay dos posibilidades:

1)

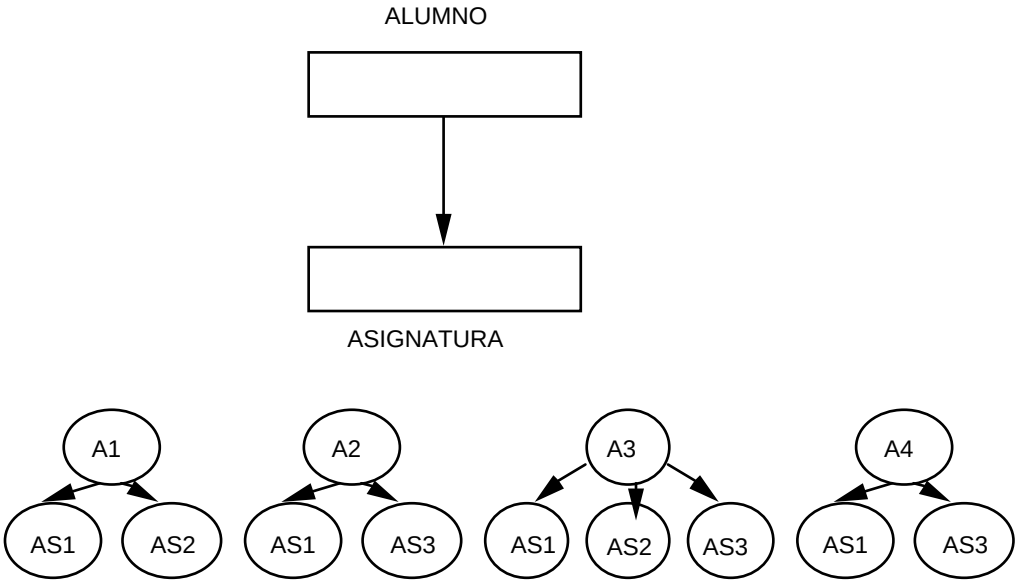


Una extensión posible podría ser esta:



Existe redundancia de alumnos.

2)



Ahora existe redundancia de asignaturas.
Hay que reducir la redundancia, según el Nº de alumnos, el Nº de asignaturas y el tamaño de cada registro.

2. OPERACIONES

Los lenguajes de manipulación de datos en este modelo proporcionan un conjunto de operadores para el proceso de datos, que están representados en forma de árboles.

OPERADORES: Localizar un árbol concreto en la B.D.
Obtener el árbol siguiente al que estamos.

Ej: En el caso 1 anterior.

a) Obtener los alumnos de la asignatura AS3.

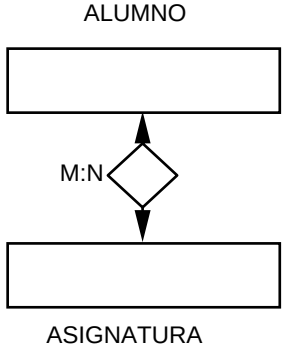
Localizar asignatura AS3
Obtener alumnos

b) Obtener las asignaturas del alumno A3.

Localizar primera asignatura
Mientras existan asignaturas
    Localizar primer alumno
    Mientras existan alumnos y alumno≠A3
        Localizar siguiente alumno
    Localizar siguiente asignatura

Para contestar a la pregunta b) hay que recorrer toda la base de datos. Para la a) no hace falta. Es mucho más ineficiente contestar a la pregunta b).

Además, las preguntas son simétricas según el esquema entidad-relación.



Pero las formulaciones son totalmente distintas.

Si quisiéramos responder a estas respuestas con el esquema jerárquico 2, la situación es la contraria.

Para pasar del esquema entidad-relación al jerárquico hay que colocar más cerca de la raíz aquellas entidades por las que se pregunta más frecuentemente y/o aquellas que necesiten un tiempo de respuesta corto. Aunque también existe el problema de la redundancia.

3. REGLAS DE INTEGRIDAD

No se pueden expresar reglas que detecten si el estado de la base de datos es consistente o no. Por ejemplo, que los valores de algún atributo estén en un determinado rango, que si el atributo de una ocurrencia de entidad toma un determinado valor exista otra ocurrencia que tenga como clave dicho valor (integridad referencial), etc....

Lo único que existe es un soporte automático de ciertas formas de integridad referencial, se asegura que ningún hijo podrá existir en la base de datos sin su correspondiente padre.

PROBLEMAS EN EL MODELO JERARQUICO

1. Existen excesivas restricciones técnicas que impiden el acceso a la B.D. de un número elevado de usuarios.

   Para hacer las preguntas hay que dar el algoritmo que obtenga lo que queremos teniendo en cuanta cuál es el esquema jerárquico.

2. Poca adaptación a las estructuras del mundo real.
   Las relaciones M:N son muy frecuentes en el mundo real.

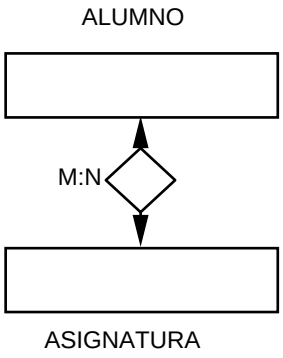
3. Falta de simetría. Los algoritmos correspondientes a preguntas simétricas tienen distinta complejidad.

MODELO EN RED

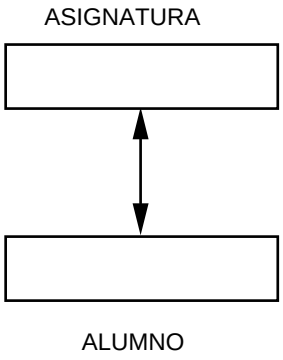
1. ESTRUCTURAS DE DATOS

Extensión de las estructuras del modelo jerárquico de tal manera que ahora un hijo puede tener varios padres.

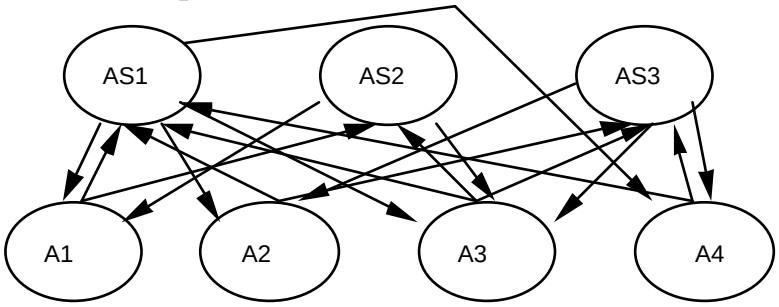
Ej: Dadas las dos entidades y la inter-relación siguiente:



En el modelo en red se puede representar fácilmente.

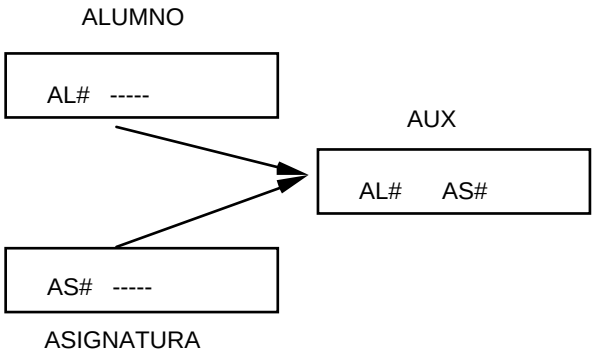


Donde una extensión podría ser:

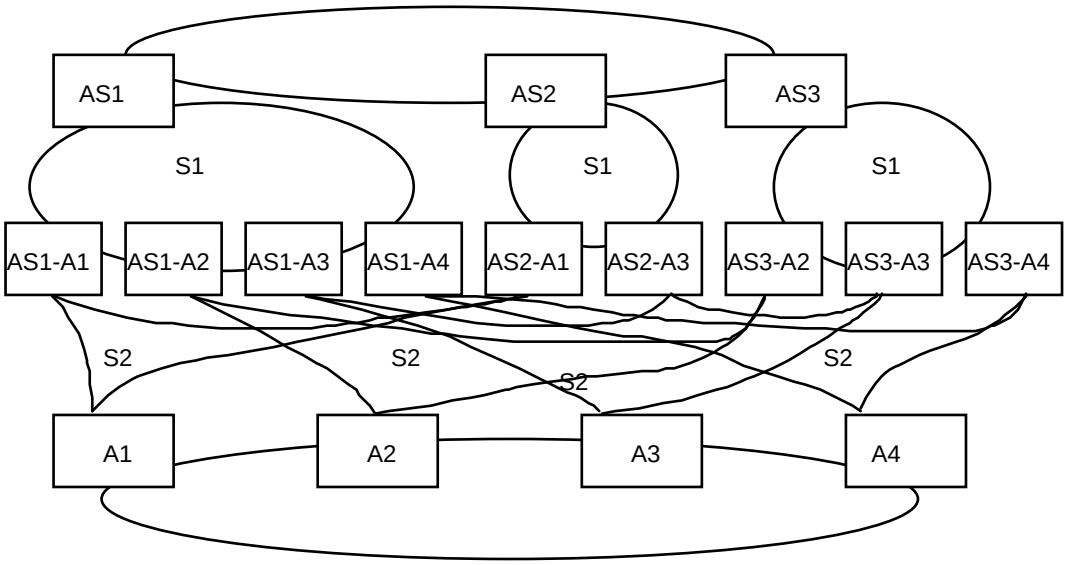


Cada ocurrencia de entidad tiene tantos apuntadores como relaciones tiene con las ocurrencias de la otra entidad. Esto es un problema de implementación ya que no se sabe cuántos apuntadores debe tener cada registro correspondiente a dicha entidad.

Restricción CODASYL: Una relación M:N se transforma en dos relaciones 1:N de la siguiente forma:



La extensión quedaría ahora de la siguiente forma:



Las ocurrencias del mismo tipo tienen igual número de apuntadores.

2. OPERACIONES

El lenguaje de manipulación de datos consta de un conjunto de operadores que permiten procesar datos representados en forma de registros y enlaces.

OPERADORES: Localizar un registro concreto (el primer registro, según el valor de algún campo,...)
Moverse de un hijo al siguiente dentro de algún enlace

a) Obtener los alumnos de la asignatura AS3.

```
Localizar asignatura AS3
Localizar primer aux en S1
mientras existan aux
    "Localizar padre de aux en S2"
    Localizar siguiente aux en S1

donde "Localizar padre de aux en S2"
si existe un apuntador "directo" del aux a su padre se utiliza
si no    Localizar siguiente aux en S2
         mientras sea de tipo aux
                Localizar siguiente aux en S2
         "Padre localizado"
         Localizar siguiente aux en S2
         mientras aux en S1 ≠ aux en S2
                Localizar siguiente aux en S2
```

b) Obtener las asignaturas del alumno A3.

```
Localizar alumno A3
Localizar primer aux en S2
mientras existan aux
    "Localizar padre de aux en S1"
    Localizar siguiente aux en S2

donde "Localizar padre de aux en S1"
si existe un apuntador "directo" del aux a su padre se utiliza
si no    Localizar siguiente aux en S1
         mientras sea de tipo aux
                Localizar siguiente aux en S1
         "Padre localizado"
         Localizar siguiente aux en S1
         mientras aux en S2 ≠ aux en S1
                Localizar siguiente aux en S1
```

3. REGLAS DE INTEGRIDAD

No se pueden expresar reglas que detecten si el estado de la base de datos es consistente o no.

Incluye ciertas formas de integridad referenciada en virtud de sus estructuras de datos primarias: los enlaces.

PROBLEMAS EN EL MODELO EN RED

1. Existen excesivas restricciones técnicas que impiden el acceso a la B.D. de un número elevado de usuarios.

   Para hacer las preguntas hay que dar el algoritmo que obtenga lo que queremos teniendo en cuenta cuál es el esquema en red.

   Ya no existe el problema 2 anterior ya que las relaciones M:N se pueden representar de manera más o menos sencilla.

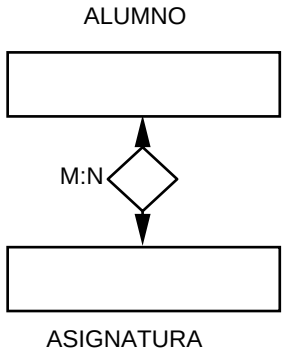
   No existe el problema 1 anterior, ya que para preguntas simétricas los algoritmos son simétricos y tienen la misma complejidad.

MODELO RELACIONAL

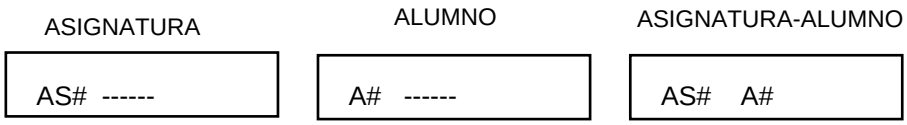
1. ESTRUCTURAS DE DATOS

Las estructuras de datos son relaciones (tablas).

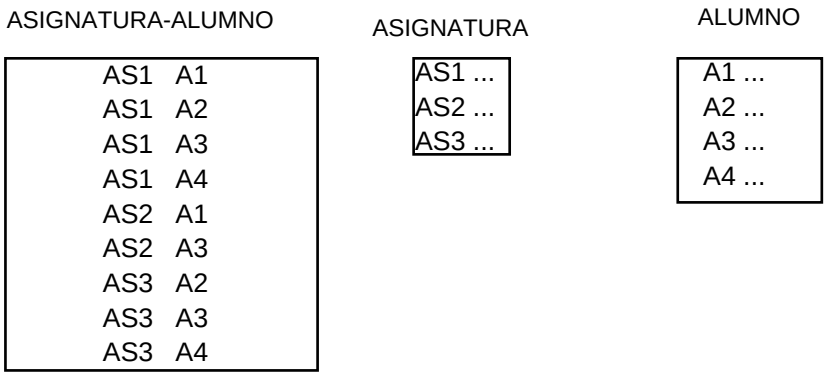
Ej: Dadas las dos entidades y la inter-relación siguientes;



En el modelo relacional se puede representar fácilmente con tres tablas.



Donde una extensión podría ser:



2. OPERACIONES

El lenguaje de manipulación de datos permite operar sobre filas y sobre tablas

OPERADORES:

a) Sobre filas: Añadir o eliminar filas

b) Sobre tablas: Unión, intersección, diferencia, proyección, selección, producto cartesiano, ...

a) Obtener los alumnos de la asignatura AS3.

```
Seleccionar "atributos-alumno"
de alumno, asignatura-alumno
donde alumno.A#=asignatura-alumno.A#
        y asignatura-alumno.AS#=AS3
```

b) Obtener las asignaturas del alumno A3.

```
Seleccionar "atributos-asignatura"
de asignatura, asignatura-alumno
donde asignatura.AS#=asignatura-alumno.AS#
        y asignatura-alumno.A#=A3
```

3. REGLAS DE INTEGRIDAD

El modelo relacional incluye la integridad de entidad (no existen tuplas sin clave), integridad referencial (un atributo toma un valor que es clave en otra tupla) y asegura que no hay tuplas con la misma clave.

PROBLEMAS QUE EXISTIAN EN LOS MODELOS ANTERIORES.

No existen excesivas restricciones técnicas que impiden el acceso a la B.D. de un número elevado de usuarios.

Tampoco existe el problema 2 anterior ya que las relaciones M:N se pueden representar de manera sencilla.

Tampoco existe el problema 1 anterior, ya que para preguntas simétricas los algoritmos son simétricos y tienen la misma complejidad.

MODELOS POST-RELACIONALES

MODELOS SEMANTICOS DE DATOS

Pretenden incorporar más aspectos semánticos, más significado.

MODELO ENTIDAD / RELACION

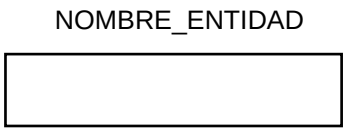
Diseñado por CHEN.

En realidad sólo ofrece estructuras de datos y no operaciones ni reglas de integridad.

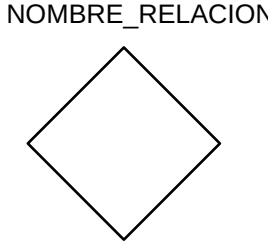
Estas estructuras de datos son: entidades, atributos, valores e inter-relaciones.

Introduce una técnica de representación: los diagramas de entidad/relación, que son muy utilizados al hacer para hacer diseños de bases de datos (para diseñar el esquema conceptual).

Las entidades se representan como



Las relaciones

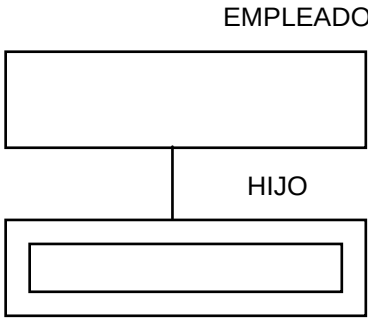


Tanto las entidades como las relaciones pueden llevar atributos

Clasificación de las entidades:

Entidades débiles. Son entidades cuya existencia depende de otra entidad en el sentido de que no podrá existir si no existe aquélla.

Ejemplo: La entidad HIJO en la base de datos de una empresa puede ser una entidad débil, dependiente de la entidad EMPLEADO.



Entidades regulares. Las restantes.

Las relaciones también se clasifican en regulares y débiles.

MODELO RM/T

Definido por Codd.

La diferencia con el modelo entidad/relación es que

- No distingue entre entidades y relaciones. Las relaciones se consideran tipos especiales de entidades. Las estructuras de datos son entidades y propiedades.

- Los aspectos estructurales y de integridad son más amplios.

- Incluye operadores distintos a los del modelo relacional.

Clasificación de las entidades:

Entidades nucleares: Aquéllas que tienen una existencia independiente.

Entidades características: Aquéllas cuya única función es describir o caracterizar a otra entidad.

Entidades asociativas: Aquéllas cuya función es representar una relación entre dos o más entidades.

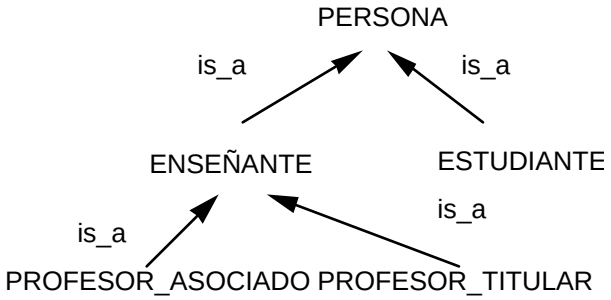
El objetivo de esta clasificación de entidades consiste en estructurar de algún modo la información.

Ejemplo. En un almacén se pueden distinguir
Entidades nucleares: los *productos*, los *vendedores* y los *clientes*.
Entidades asociativas: los *pedidos*, que relacionan a vendedores, productos y clientes.
Entidades características: los *envíos*, entidades características de pedidos.

Las entidades se agrupan también en conjuntos de entidades. Es equivalente a que las ocurrencias de entidad se asocien a entidades.

Cada ocurrencia se identifica entre todas las ocurrencias de una entidad por el valor de un subrogado (que es generado por el sistema) En el modelo relacional las ocurrencias se identifican por el valor de la clave.

Un tipo de entidades e1 es subtipo de otro tipo e2 si todas las entidades pertenecientes a e1 pertenecen también a e2. Además, un subtipo puede serlo de más de un tipo a la vez.
Ejemplo:



Propiedades de las entidades.

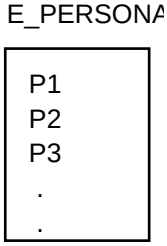
a) Propiedades inmediatas. Sólo pueden tener un valor como máximo para cada ocurrencia de entidad y no pueden tener a su vez propiedades.
Ejemplo: nombre, fecha de nacimiento son propiedades inmediatas de personas.

b) Entidades características. Son atributos que a su vez son entidades y como tales pueden tener a su vez propiedades inmediatas y características. Pueden tener valores múltiples, a una misma entidad le pueden corresponder varias entidades características del mismo tipo.
Ejemplo: envío es una entidad característica de pedido con propiedades: tipo, número

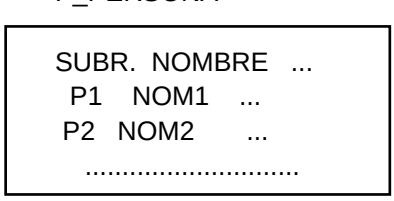
Las propiedades inmediatas y entidades características de un tipo lo son también de sus subtipos, es decir, que las propiedades se heredan.
Ejemplo: la propiedad nombre de persona lo es también de estudiante, enseñante,...

Representación de las entidades.

Todas las entidades, sean del tipo que sean, se representan por E_relaciones o por P_relaciones. Hay una E_relación por cada tipo de entidad. Es una relación que contiene los subrogados de todas las entidades pertenecientes al tipo entidad. Por lo tanto, contiene información de qué ocurrencias de entidad son de dicha entidad.



En las P_relaciones se representan las propiedades de una entidad.



Reglas de integridad.

- Integridad de entidades en RM/T. Una E_relación acepta inserciones y borrados pero no modificaciones.

- Integridad de las propiedades. Si una tupla t aparece en una P_relación entonces el subrogado de la tupla t debe aparecer en la E_relación correspondiente.

- Integridad característica. Una entidad característica no puede existir en la BD a no ser que la entidad que describa esté en la BD.

- Integridad de asociación. Una entidad asociativa no puede existir en la BD a no ser que las entidades participantes en la asociación también estén en la BD.

- Integridad sub_tipo. Siempre que un subrogado aparezca en una E_relación correspondiente a una entidad e1 deberá aparecer también en la E_relación de una entidad e2 para la cual e1 es un subtipo.

Operaciones.

Especialización. Consiste en pasar de un objeto colectivo a un objeto individual o bien a otro objeto colectivo que es parte del primero.

- Instanciación: se escoge una ocurrencia concreta de una entidad.
  ej: entidad: enseñante
  instancia: Koldo Agirre

- Subtipo: se pasa de un tipo de entidades a uno de sus subtipos.
  ej: entidad: enseñante
  subtipo: ayudante

Generalización. Consiste en pasar de un objeto individual a uno colectivo o de un objeto colectivo a otro colectivo que lo incluye.
ej: persona generaliza a estudiante

Agregación. Por ella, una relación entre objetos es considerada a su vez un objeto.

- Agregación de propiedades para dar lugar a un tipo de entidad que puede ser característica, asociativa o nuclear y que tendrán como propiedades inmediatas a dichos atributos.

Ejemplo: Las propiedades día, mes y año de una entidad pueden pasar a ser en sí mismas una entidad que sea fecha.

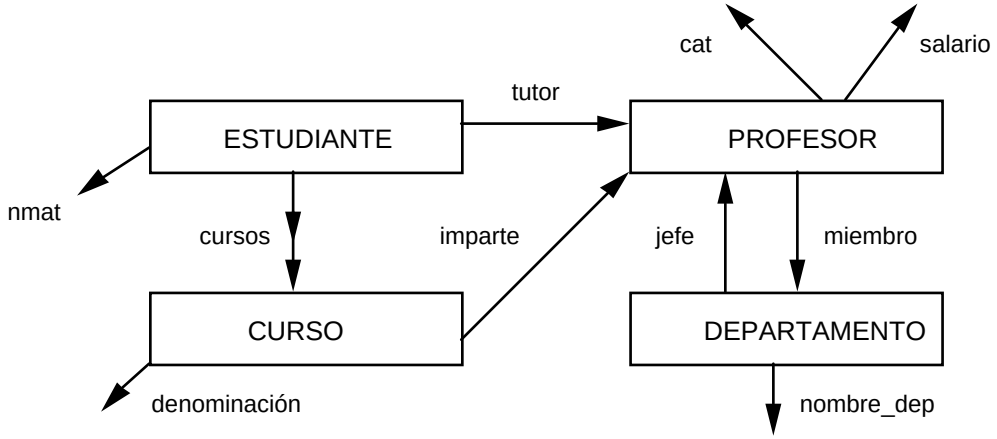
- Agregación de entidades características produce un conjunto de entidades características, nucleares o asociativas que tendrán como atributos dichas entidades características.

MODELO FUNCIONAL DE DATOS

Definido por Skipman (1979)

Se basa en la noción de red semántica de inteligencia artificial. Una red semántica es una estructura que se utiliza para representar asociaciones entre objetos.

Para cada objeto de un tipo dado, existe una colección de funciones que son aplicables al mismo. Algunas de estas funciones proporcionan valores simples. El resultado de otras se encuentra siguiendo arcos en la red que conectan objetos con otros objetos de varios tipos.



Tipos de items.

El modelo funcional trabaja con dos tipos de items: entidades y escalares.

Escalares. Son valores atómicos simples. Ej: enteros, reales, ...

Entidad. Identifica un objeto único en la BD. (No se puede imprimir, sólo se pueden imprimir las funciones escalares definidas sobre él).
Ejemplo: Si una entidad representa a una persona, se podrían aplicar las funciones nombre, edad, sexo e imprimir los valores que devolviera.

Tipos de funciones.

Valor escalar. ej: Nombre(estudiante) devuelve un string
Salario(profesor) devuelve un entero

Valor simple. ej: jefe(departamento) devuelve un único profesor, una única instancia de la entidad profesor

Valor múltiple. ej: cursos(estudiante) devuelve varios cursos, un conjunto de instancias de la entidad curso

Composición de funciones.

Existen dos situaciones diferentes

a) la función composición es valor escalar o simple
f(g(x))=[y tales que y=f(z) y z pertenece a g(x)]

b) la función composición es de valor múltiple
f(g(x))=[y tales que y pertenece a f(z) y z pertenece a g(x)]

Lenguaje de manipulación

Daplex.
Pregunta: Obtener el nombre de los profesores de los cursos que admiten a estudiantes de más de 21 años.

```
for each estudiante
such that edad(estudiante)>21
for each imparte(cursos(estudiante))
        print nombre(profesor)
                edad(estudiante)
```

Soporta la noción de subtipo.
Suponemos que toda persona tiene edad y nombre. Además, profesor y estudiante son subtipos de persona, por lo que heredan dichos atributos.

4. Modelo Relacional

1.- INTRODUCCION
2.- DEFINICION DEL MODELO RELACIONAL
3.- TECNICAS DE DISEÑO
4.- LENGUAJES RELACIONALES
5.- TECNICAS DE OPTIMIZACION

1. INTRODUCCION

Ideado por Edgar F. Codd en 1970, cuando trabajaba en IBM.

Los objetivos del modelo relacional son:

1.- Mejorar la independencia física y lógica.

Independencia física:

Capacidad para modificar las estructuras de datos sin tener que cambiar los programas de aplicación.

Modo en que están conectados los registros.
Orden en las secuencias de registros.
Reorganizar áreas de overflow.
Cambiar un fichero indexado por uno hash.

Independencia lógica:

Capacidad para modificar las estructuras lógicas sin tener que cambiar los programas de aplicación.

Renombrar alguna entidad o atributo en el esquema conceptual.
Redefinir el tipo de algún atributo.

Se consigue gracias a las VISTAS.

2.- Quitar las excesivas restricciones técnicas que impiden el acceso a la B.D. a un amplio número de usuarios.

Los usuarios de los modelos en red y jerárquico tenían que saber bastante para hacer preguntas a la B.D.

3.- Proporcionar a los usuarios lenguajes asercionales, en los que sólo hay que indicar qué es lo que se quiere obtener y no cómo se tiene que obtener. Los modelos en red y jerárquico ofrecen lenguajes navegacionales. En este tipo de lenguajes hay que decir qué es lo que se quiere obtener y la manera (el algoritmo) de obtenerlo.

4.- Mejorar la integridad y confidencialidad.

Permiten expresar reglas de integridad.

5.- Optimizar los accesos a la B.D.

Como las preguntas son asercionales, se puede buscar la manera más eficiente de obtener aquello que ha pedido el usuario. En los modelos jerárquico y en red no se puede ya que las preguntas son navegacionales.

6.- Aumentar el número de aplicaciones que se pueden realizar sobre una B.D., ya que hacer una aplicación con el modelo relacional no supone tanto esfuerzo como hacerla con los modelos pre-relacionales.

7.- Obtener una metodología que nos ayude a diseñar bases de datos.

Es posible ya que existe un fundamento matemático fuerte en el modelo relacional.

2. DEFINICION DEL MODELO RELACIONAL

MODELO DE DATOS: Instrumento teórico que permite representar los datos de un organismo.

COMPONENTES DE UN MODELO DE DATOS:

1. Estructuras de datos

Las ocurrencias de los datos no son independientes las unas de las otras. Forman estructuras. Ej: árboles, redes, tablas,...

2. Operaciones

Se necesitan operaciones para manejar dichas estructuras: para consultar, borrar, actualizar datos.

3. Reglas de integridad

Permiten especificar qué estados de la base de datos son consistentes.

2.1- ESTRUCTURAS DE DATOS

Las estructuras de datos son la relaciones (tablas).

Dominio: Conjunto de valores identificados por un nombre.

Ejemplos: entero, real, string, días-semana, meses, ...

Relación:

Sean D_1, D_2, ... D_n dominios. Una relación R es un subconjunto del producto cartesiano $D_1 \times D_2 \ldots \times D_n$

Ejemplo: Relación PERSONAS subconjunto de string x entero

PERSONAS = { <PEPE, 15>, <KOLDO, 32>, <ANA, 32> }

Atributo: Nombre que se le da a cada dominio que participa en la relación, y qué significa.

Ejemplo: En la relación anterior existen dos atributos
NOMBRE que toma valores en el dominio string
y EDAD que toma valores en el dominio entero.

Tupla: Cada elemento de la relación.
Ejemplo: <PEPE, 15> es una tupla de la relación PERSONAS

Cardinalidad de la relación: Número de tuplas de la relación.
En el ejemplo anterior: 3

Grado de la relación: Número de atributos de la relación.
En el ejemplo anterior: 2

Clave de la relación: Atributo o conjunto de atributos mínimo que identifica de manera unívoca a cada tupla.

Como una relación es un conjunto de tuplas entonces no pueden existir dos tuplas iguales. Esta es una restricción del modelo relacional. Por lo tanto, existirá un conjunto de atributos que identifique a cada tupla. En el peor de los casos serán todos los atributos.

Ejemplo: PERSONAS:

NOMBRE	EDAD
PEPE	15
KOLDO	32
ANA	32

NOMBRE es clave, ya que conociendo el NOMBRE de una persona sabemos a qué tupla nos referimos.
si NOMBRE es PEPE la tupla es <PEPE, 15>

EDAD no es clave
si EDAD = 32 no sabemos si es la tupla <KOLDO, 32> o <ANA, 32>

PERO si la tupla <PEPE, 16> estuviera en la relación PERSONAS entonces

NOMBRE no sería clave
si NOMBRE es PEPE no sabemos si es la tupla <PEPE, 15> o <PEPE, 16>

La clave sería NOMBRE y EDAD.
si NOMBRE es PEPE y EDAD es 16 sabemos que la tupla es <PEPE, 16>

Si existen varias claves posibles hay que escoger una clave (la clave primaria) y las demás quedan como alternativas o secundarias.

Ejemplo:

RELACION: EMPLEADOS

ATRIBUTOS:	CODIGO	APELLIDO	EDAD	CIUDAD
	1	GARCIA	17	LONDRES
	2	PEREZ	17	NEW YORK
TUPLAS:	3	MARTIN	20	SEVILLA
	4	RODRIGUEZ	25	SEVILLA
	5	AGIRRE	23	PARIS

CARDINALIDAD: 5 ya que hay 5 tuplas.
GRADO: 4 ya que hay 4 atributos.
CLAVES POSIBLES: CODIGO y APELLIDO

Si existieran apellidos repetidos APELLIDO no podría ser clave. Aunque en este momento (en esta extensión de la relación) no haya apellidos repetidos parece razonable que alguna vez existan empleados con igual apellido.
Al escoger una clave primaria se está diciendo que nunca va a haber tuplas con igual valor en la clave primaria. Por eso es mejor escoger CODIGO en vez de APELLIDO.

Esquema de la relación: El nombre de la relación y el de sus atributos con sus dominios.
Ej: EMPLEADOS(CODIGO : ENTERO,APELLIDO : STRING,
EDAD : ENTERO, CIUDAD : STRING)

Intensión de la relación: Propiedades que cumplen las tuplas y que no varían en el tiempo.
- El esquema forma parte de la intensión.
- También se pueden dar propiedades como la siguiente: "la edad será siempre mayor que 18 y menor que 70"

Extensión de la relación: Es el conjunto de tuplas que una relación tiene en un determinado momento de tiempo.

Valores nulos: Hay atributos que pueden tomar valores nulos.
Un atributo para una tupla toma un valor nulo (llamado NULL).
1) Si **se desconoce** dicho valor.

PERSONA (NOMBRE, EDAD)

JUAN	---
KEPA	17

2) o si **no es aplicable,** si no tiene sentido.

PERSONA (NOMBRE, TRABAJA, SUELDO)

JUAN	SI	100.000
ANA	SI	150.000
LUIS	NO	---

NOTA: Un atributo clave no puede tomar valores NULL.

Clave extranjera: Atributo o conjunto de atributos en una relación que toma valores que deben coincidir con los valores que toma la clave primaria de otra relación.
Ej: PROFESOR (NOM-PROF,EDAD,DIRECCION,...)
ALUMNO (NOM-ALUM,EDAD,..., TUTOR)
TUTOR es una clave extranjera.
Puede tomar como valores o bien los tomados por NOM-PROF o bien NULL.

2.2 OPERACIONES EN EL ALGEBRA RELACIONAL

Las operaciones que manejan las estructuras de datos del modelo relacional (las tablas) son cerradas, es decir, toman como argumentos tablas y devuelven como resultado tablas.

a) SELECCION

$\sigma_Q(r)$ (selección de la tabla r según el predicado Q)
es una tabla.

con los mismos atributos que la tabla r
cuya extensión es el conjunto de tuplas de r que cumplen la condición Q.

Ejemplo: **PERSONA**

NOMBRE	**EDAD**	**CIUDAD**
KOLDO	30	IRUÑEA
KEPA	25	BILBO
NEREA	26	DONOSTIA
MAITE	17	GAZTEIZ
MIKEL	16	IRUÑEA

Ejemplo: $\sigma_{CIUDAD='IRUÑEA'}(PERSONA)$

NOMBRE	**EDAD**	**CIUDAD**
KOLDO	30	IRUÑEA
MIKEL	16	IRUÑEA

Ejemplo: $\sigma_{CIUDAD \neq 'IRUÑEA' \wedge EDAD >= 25}(PERSONA)$

NOMBRE	**EDAD**	**CIUDAD**
KEPA	25	BILBO
NEREA	26	DONOSTIA

b) PROYECCION

$\Pi_{A1,...,Ap}(r)$

(proyección de la tabla r sobre los atributos A1, ..., Ap) es una tabla:

con los atributos A1, ..., Ap (que son también de r)

cuya extensión son las tuplas de r teniendo en cuenta sólo los valores de los atributos

A1, ..., Ap y en la que se quitan los duplicados que haya.

Ejemplo: Π_{EDAD} (PERSONA)

EDAD
30
25
26
17
16

Π_{CIUDAD} (PERSONA)

CIUDAD
IRUÑEA
BILBO
DONOSTIA
GAZTEIZ

Π_{CIUDAD} ($\sigma_{CIUDAD = 'IRUÑEA'}$ (PERSONA))

CIUDAD
IRUÑEA

C) PRODUCTO CARTESIANO

r x t (producto cartesiano de las tablas r y t) es una tabla

con todos los atributos de r y de t

cuya extensión es el conjunto formado por todas las tuplas que resultan al concatenar cada tupla de r con cada tupla de t

Ejemplo:

Sean las tablas PERSONA (ya definida) y COCHE

COCHE

NOM_COCHE	**PRECIO**
GOLF	25
PANDA	20

PERSONA x COCHE

NOMBRE	**EDAD**	**CIUDAD**	**NOM_COCHE**	**PRECIO**
KOLDO	30	IRUÑEA	GOLF	25
KOLDO	30	IRUÑEA	PANDA	20
KEPA	25	BILBO	GOLF	25
KEPA	25	BILBO	PANDA	20
NEREA	26	DONOSTIA	GOLF	25
NEREA	26	DONOSTIA	PANDA	20
MAITE	17	GAZTEIZ	GOLF	25
MAITE	17	GAZTEIZ	PANDA	20
MIKEL	16	IRUÑEA	GOLF	25
MIKEL	16	IRUÑEA	PANDA	20

D) JOIN

r $x_{A=B}$ t (join entre las tablas r y t donde los atributos A de r y B de t tienen el mismo dominio)

No es una operación básica ya que se puede expresar utilizando las operaciones anteriores

r $x_{A=B}$ t es igual a $s_{A=B}$(r x t)

Ejemplo:

Dadas las tablas PERSONA, COCHE y POSEE

POSEE

NOMBRE_PER	NOMBRE_COC
KOLDO	GOLF
NEREA	GOLF
KOLDO	PANDA

PERSONA $X_{NOMBRE=NOMBRE-PER}$ POSEE

NOMBRE	EDAD	CIUDAD	NOMBRE_COC
KOLDO	30	IRUÑEA	GOLF
NEREA	26	DONOSTIA	GOLF
KOLDO	30	IRUÑEA	PANDA

PERSONA $X_{NOMBRE=NOMBRE-PER}$POSEE $X_{NOMBRE-COC=NOM-COCHE}$ COCHE

NOMBRE	EDAD	CIUDAD	NOM_COCHE	PRECIO
KOLDO	30	IRUÑEA	GOLF	25
NEREA	26	DONOSTIA	GOLF	25
KOLDO	30	IRUÑEA	PANDA	20

E) UNION

r U t (unión entre las tablas r y t, con atributos definidos sobre los mismos dominios)

EST_INFORMATICA

NOMBRE	**EDAD**
ANA	25
KOLDO	19
JUAN	20

EST_DERECHO

NOMBRE	**EDAD**
MIREN	22
KOLDO	19
ROSA	23

EST_INFORMATICA : EST_DERECHO

NOMBRE	**EDAD**
ANA	25
KOLDO	19
JUAN	20
MIREN	22
ROSA	23

F) DIFERENCIA

r - t (diferencia entre las tablas r y t, con atributos definidos sobre los mismos dominios)

EST_INFORMATICA - EST_DERECHO

NOMBRE	**EDAD**
ANA	25
JUAN	20

G) <u>INTERSECCION</u>

r ∩ t (intersección entre las tablas r y t, con atributos definidos sobre los mismos dominios)

No es operación básica ya que r Ñ t es igual a r - (r - t)

EST_INFORMATICA Ñ EST_DERECHO

NOMBRE	**EDAD**
KOLDO	19

H) <u>DIVISION</u>

r / t (división entre las tablas r y t, donde todos los atributos de t están en r)

r es la relación **dividendo**

t es la relación **divisor**

para cada atributo de t hay uno en r definido sobre el mismo dominio

r / t es la relación **cociente** formada por:

- todos los atributos de r que no corresponden a ninguno de t
- una tupla pertenece a **(r / t)** si concatenándola con **todas las tuplas de t** nos dan **tuplas que pertenecen a r**.

No es operación básica ya que

r / t es igual a $\pi_{(r-t)} - \pi_{(r-t)}((\pi_{(r-t)} \times t) - r)$

donde (r-t) son los atributos de r no correspondientes a t

ESTUDIA ASIGNATURA ESTUDIA/ASIGNATURA

NOMBRE	**ASIGNAT**
KOLDO	MATE
KOLDO	LATIN
PEDRO	MATE
ANA	LATIN

ASIGNAT
MATE
LATIN

NOMBRE
KOLDO

Otro componente del modelo relacional: la vista.

Las operaciones del álgebra relacional son cerradas, es decir, toman como argumentos relaciones y devuelven como resultado también relaciones.

Una vista es una expresión del álgebra relacional, por lo tanto una relación, a la que se le asigna un nombre.

Se distinguen ahora entre las relaciones básicas (tablas de base), aquéllas que existen físicamente en la base de datos.

Ejemplo: PERSONA, POSEE, COCHE anteriormente definidas.

Y las vistas, definidas mediante los operadores del álgebra relacional sobre las tablas de base y/o vistas.

La vista DONOSTIARRA definida como $\sigma_{CIUDAD='DONOSTIA'}(PERSONA)$

La vista DONOST_CON_COCHE definida como $\sigma_{CIUDAD='DONOSTIA'}(PERSONA \times_{NOMBRE=NOMBRE\_PER} POSEE)$

2.3 REGLAS DE INTEGRIDAD

En una base de datos es importante que los datos sean consistentes, es decir, que se pueda asegurar ciertas cosas sobre ellos sin tener que mirar en la base de datos, ya que sabemos que cumplen unas restricciones de integridad.

Estas restricciones pueden ser:

1) estructurales: Son las que garantiza el modelo relacional:
unicidad de clave, integridad de entidad e integridad referencial.

2) de comportamiento: Características de cada aplicación y definidas por el usuario.

- Un atributo toma valores comprendidos en un determinado rango.

Ejemplo de una restricción de integridad de comportamiento: En la base de datos anterior podríamos tener la restricción de integridad: $16 \leq EDAD \leq 70$

Si a esta base de datos se le preguntara por el nombre de aquellas personas que tienen 15 años y que poseen un GOLF, es decir,

$\pi_{NOMBRE}(\sigma_{EDAD=15 \text{ ñ } NOMBRE\text{-}COC='GOLF'}(PERSONA \times_{NOMBRE=NOMBRE\text{-}PER} POSEE))$

nos devolvería una tabla vacía.

NOMBRE

El sistema podría deducirlo sin tener que mirar todos los datos de la base de datos, ya sabe que no existe ninguna persona con 15 años y con un GOLF ya que no existen en esa base de datos personas con menos de 16 años (si permitiera declarar este tipo de restricciones y realizara optimización semántica).

Restricciones de integridad estructurales.

a) Unicidad de clave.

La clave no puede tomar dos valores iguales.

Existe para cada relación un conjunto mínimo de atributos, la clave, que identifica de manera unívoca a cada tupla. Si existen varias claves posibles, se escoge una como clave primaria y el resto quedan como claves alternativas o secundarias.

b) Integridad de entidad.

Ninguno de los atributos que forma parte de la clave primaria puede tomar valores nulos (NULL).

Como clave primaria es el conjunto de atributos que identifica unívocamente a una tupla (según el valor que tomen), parece lógico que una tupla no tome un valor desconocido en su clave primaria.

c) Integridad referencial

Las claves extranjeras deben tomar valores que coinciden con el valor de la clave primaria correspondiente en alguna tupla, o si no el valor NULL.

Ej: A (A#, NOMBRE)
B (B#, NOMBRE, QUE-A#)

La integridad referencial garantiza que toda tupla de B toma en el atributo QUE-A# o bien el valor NULL o bien un valor **a** tal que existe la tupla <**a**, ...> en A.

Generalmente todos los sistemas garantizan las restricciones de unicidad de clave y la integridad de entidad.Cada vez que se quiere introducir una tupla en una relación hay que introducir valores para todos los atributos de la clave primaria (para garantizar la integridad de entidad) y una vez introducidos se comprueba que no existe ninguna tupla en esa relación con la misma clave ya que en ese caso no se puede introducir dicha tupla (para garantizar la unicidad de clave).
Sin embargo garantizar la integridad referencial es bastante más costoso.
Dadas las relaciones

A (A#, NOMBRE)
B (B#, NOMBRE, QUE-A#)

correspondientes al esquema E/R.
Suponiendo que la clave extranjera cumple la integridad referencial en un momento dado.
1) ¿Qué sucede si se quita una tupla de A?
¿Qué se hace con aquellas tuplas de B que tienen como valor en el atributo QUE-A# el valor de la clave de la tupla borrada?
No se pueden dejar igual ya que la integridad referencial dejaría de cumplirse.
Hay varias posibilidades:

a) Se puede poner a NULL el valor del atributo QUE-A# para dichas tuplas de B. (Opción ANULADO)
b) Se puede borrar todas esas tuplas de B. (Opción CASCADA)
c) No permitir el borrado de la tupla de A. (Opción RESTRINGIDO)

2) ¿Qué sucede si se cambia el valor de la clave primaria de una tupla de A?
¿Qué se hace con aquellas tuplas de B que tienen como valor en el atributo QUE-A# el valor anterior de la clave primaria de la tupla actualizada?
Otra vez se deja de cumplir la integridad referencial. Las posibilidades para solucionar el problema son las mismas: ANULADO, CASCADA, RESTRINGIDO. **Qué posibilidad escoger depende en cada caso de las entidades relacionadas.**

Ej: ALUMNO (A#, ...)
NOTA (AS#, 𝔸#, NUM)

Si borramos la información de un alumno, ¿queremos para algo sus notas? Supongamos que no. Entonces al declarar a A# como una clave extranjera tendremos que decir que al borrar hay que utilizar la opción CASCADA.

Ej: PERSONA (P#, ...)
MULTA (M#, P#, CANTIDAD)

Si borramos la información de una persona, ¿borramos sus multas? Seguramente lo mejor será no permitir borrar la información de dicha persona. Esta es la opción RESTRINGIDO.

Ej: PERSONA (P#, ...)
DESPACHO (D#, ..., ℙ#)

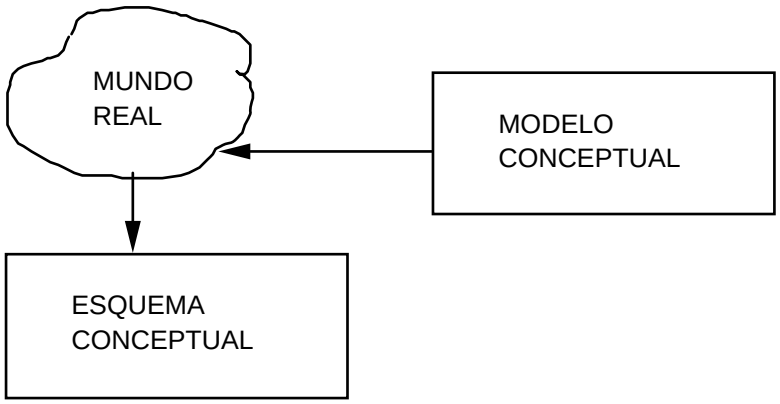
Si borramos la información de una persona, ¿borramos el despacho que ocupa? No, habrá que dejar a NULL el valor del campo P# indicando que ese despacho no está ocupado por nadie. Es la opción ANULADO.

3. TECNICAS DE DISEÑO

1) OBTENCION DEL ESQUEMA CONCEPTUAL
2) PASO DEL ESQUEMA CONCEPTUAL EN EL MODELO ENTIDAD/RELACION AL RELACIONAL.
3) NORMALIZACION DE LAS TABLAS RELACIONALES

3.1. OBTENCION DEL ESQUEMA CONCEPTUAL

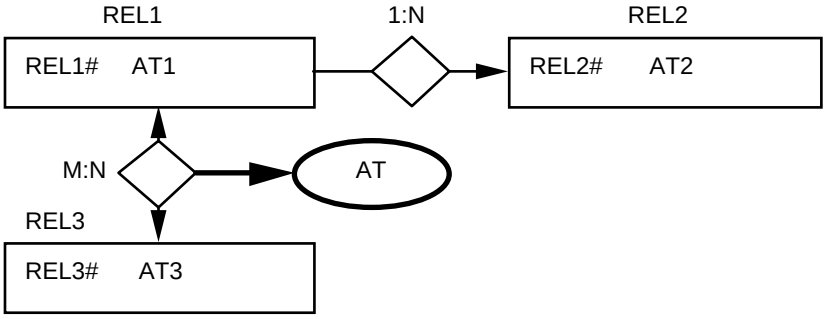
a) Mundo Real. En este mundo existen cosas (Ej: personas, animales, coches...) y se producen acontecimientos (Ej: Las personas nacen, mueren, trabajan, compran coches, tienen animales, ...)
b) Mundo de las concepciones. Cada observador percibe el mundo real y lo conceptualiza a su manera. Distingue ciertas cosas como entidades, que tienen atributos y que se interrelacionan entre sí.
c) Mundo de las representaciones.
Toda representación tiene un aspecto lógico: palabras, códigos, ... y un aspecto físico: el soporte de los datos.



EL ESQUEMA CONCEPTUAL se suele representar utilizando el modelo semántico de datos ENTIDAD / RELACION (E/R)

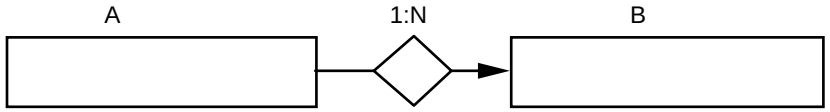
3.2 PASO DEL MODELO E/R AL MODELO RELACIONAL

Dado el esquema E/R



Se obtienen las tablas
REL1 (REL1#, AT1)
REL2 (REL2#, AT2, QUE-REL1)
REL3 (REL3#, AT3)
REL1-REL3 (REL1#, REL3#, AT)

a) Las entidades del modelo E/R son tablas en el modelo relacional.
b) Los atributos del modelo E/R son atributos en el modelo relacional.
Para representar las relaciones del modelo E/R.
c)



Se añade un atributo a la tabla correspondiente a la entidad B. Cada tupla de B tendrá como valor en dicho atributo una referencia lógica (no un apuntador físico) a la tupla de A con la que está relacionada.

Nótese que para cada tupla de B habrá a lo sumo una tupla de A relacionada (según nos dice el grado 1:N de la relación).

Ejemplo:

Sea la entidad A con atributos A#, NOMBRE
Ocurrencias de A: <A1 XX> <A2 YY> <A3 ZZ>
Sea la entidad B con atributos B#, NOMBRE
Ocurrencias de B: <B1 AA> <B2 BB> <B3 CC>
<B4 DD> <B5 EE>

Además existe una relación entre las entidades A y B tal que a cada tupla de A le pueden corresponder varias de B, pero a cada tupla de B como máximo una de A.

En concreto <A1 XX> relacionada con <B1 AA > y < B2 BB>
<A2 YY> con <B3 CC> y <B4 DD>

En el modelo relacional tendríamos las tablas.

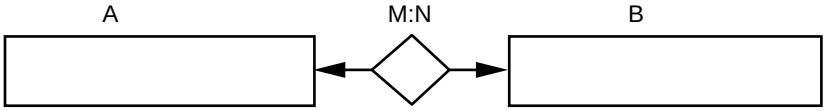
A

A#	NOMBRE
A1	XX
A2	YY
A3	ZZ

B

B#	NOMBRE	QUE_A#
B1	AA	A1
B2	BB	A1
B3	CC	A2
B4	DD	A2
B5	EE	NULL

d)



No se puede hacer como antes añadiendo un atributo a cada tabla ya que para una tupla determinada debería tomar como valor todas aquellas referencias a las tuplas de la otra con las que está relacionada. Pero esto no puede ser ya que los atributos en el modelo relacional no son multievaluados. Por lo tanto es necesario utilizar otra tabla con los atributos para referenciar lógicamente a una tupla de A y a otra de B (aquéllas que están relacionadas). Estos atributos son la clave de A y la de B.

Ejemplo: Si en el ejemplo anterior tenemos que

<A1 XX> está relacionada con <B1 AA>, <B2 BB> y <B3 CC>
<A2 YY> con <B1 AA>, <B3 CC> y <B4 DD>
<A3 ZZ> con <B1 AA>, <B2 BB> y <B3 CC>

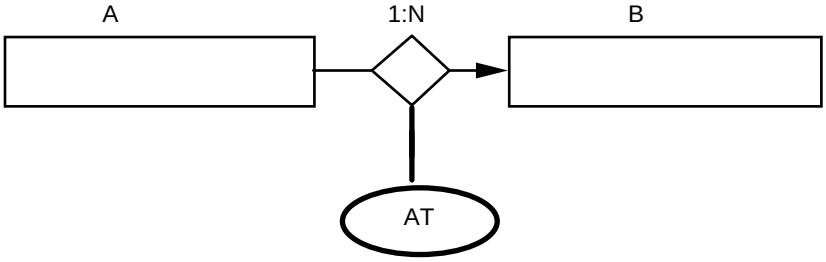
Las tablas del modelo relacional serían:

A#	NOMBRE
A1	XX
A2	YY
A3	ZZ

B#	NOMBRE
B1	AA
B2	BB
B3	CC
B4	DD
B5	EE

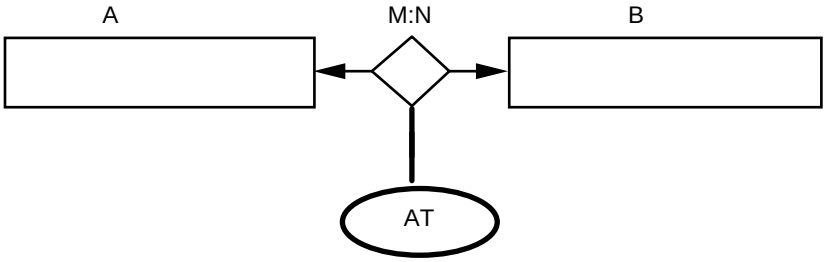
A#	B#
A1	B1
A1	B2
A1	B3
A2	B1
A2	B3
A2	B4
A3	B1
A3	B2
A3	B3

e)



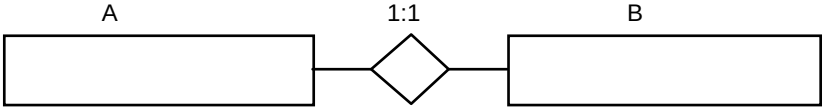
RA (A#, ...)
RB (B#, ..., A#, AT)

f)



RA (A#, ...)
RB (R#, ...)
RAB (A#, B#, AT)

g)



RA (A#, ..., B#)
RB (B#, ..., A#)

3.3 NORMALIZACION.

Las tablas obtenidas a partir del modelo E/R pueden tener una serie de características que creen problemas al insertar nuevas tuplas y/o al actualizar o borrar otras. Además puede haber redundancia de información.

Ejemplo: Supongamos que un inexperto en bases de datos tiene que construir una BD con información sobre una biblioteca. Tras preguntar a los bibliotecarios éstos le dicen que necesitan datos sobre los lectores (su número, su nombre, la ciudad en la que viven, el tipo de libros que le gustan y el número de habitantes de dicha ciudad, para hacer estadísticas sobre las ciudades con más lectores), sobre los libros que tiene (su código, su título, y el tipo de libro que es), y lo más importante, saber qué libro está prestado y a quién.
Tras mucho pensar decide que lo mejor es tener un esquema E/R sencillo con una única entidad.

PRESTAMO

LE	LI	NOMBR	CIUDAI	NºHAB	T_LIBRO	TITULO	TIPO

La tabla relacional correspondiente a dicha entidad sería:
PRESTAMO(LE#,LI#,NOMBRE,CIUDAD,NHAB,TIPO_LIBROS,TITULO,TIPO)
donde LE# y LI# forman la clave.

Una posible extensión podría ser:

LE#	LI#	NOMBRE	CIUDAD	NºHAB	TIPO_LIBRO	TITULO	TIPO
1	1	PEREZ	SEVILLA	500000	AVENTURA ROMANTICO	ENAMORADOS HASTA MEDULA	ROMANTICO
1	3	PEREZ	SEVILLA	500000	AVENTURA ROMANTICO	TIJUANA JONES	AVENTURA
2	4	SMITH	SEVILLA	500000	RELIGIOSO CC	LA BIBLIA	RELIGIOSO
2	5	SMITH	SEVILLA	500000	RELIGIOSO CC	EL CORAN	RELIGIOSO
3	1	GONZALEZ	BARCELONA	3000000	ROMANTICO	ENAMORADOS HASTA MEDULA	ROMANTICO

En esta tabla se producen varias anomalías.

1.- El atributo TIPO_LIBROS es multievaluado ya que puede tomar más de un valor. Las tablas en el modelo relacional no admiten atributos multievaluados.

2.- Supongamos que viene un nuevo lector a la biblioteca que quiere hacerse socio pero que todavía no desea coger en préstamo un libro. No se puede introducir su información ya que para hacerlo hay que conocer los valores para la clave.

LE# es el código de lector

LI# es el código del libro que se coge en préstamo. NO SE SABE.

Esta es una anomalía de inserción.

3.- Si González devuelve su libro se borra la única tupla que contiene información sobre el número de habitantes de Barcelona, y sobre el tipo de libros que le gustan, su ciudad, etc... Si Smith devuelve la biblia se pierde la información de que existe dicho libro en la biblioteca.

Estas son anomalías de borrado.

4.- Si se quiere modificar el número de habitantes de Sevilla porque ha crecido, hay que hacerlo en todas las tuplas en las que aparece Sevilla.

Esta es una anomalía de actualización.

Además existe gran redundancia de información y con gran riesgo de que existan inconsistencias en las tuplas (por ejemplo distinto NHAB para la misma CIUDAD).

En resumen, existen unas características en las relaciones que producen unos problemas de inserción, actualización o borrado (anomalías). Estas características se pueden eliminar si se descomponen las relaciones en otras de estructura más sencilla. También existe redundancia de información.

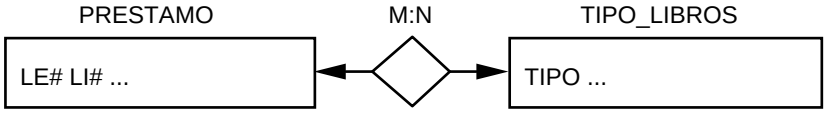
El proceso de normalización descompone las relaciones para eliminar las anomalías manteniendo el contenido de la base de datos.

PRIMERA FORMA NORMAL (1FN)

Una relación está en 1FN si no tiene atributos multievaluados, es decir, si una tupla no toma como valor para un atributo un conjunto de elementos.

En el ejemplo anterior TIPO_LIBROS es una atributo multievaluado ya que por ejemplo
- la tupla de clave LE#=1 y LI#=1 toma los valores AVENTURA y ROMANTICO.
- la tupla de clave LE#=1 y LI#=3 toma los valores AVENTURA y ROMANTICO.

¿Cómo se consigue que no haya atributos multievaluados? En vez de considerar que existe un atributo multievaluado, se considera que existe una relación entre la entidad PRESTAMO y otra nueva TIPO_LIBROS.



El grado es M:N ya que a una tupla de PRESTAMO le corresponde más de un TIPO_LIBROS y un TIPO_LIBRO le puede corresponder a más de un PRESTAMO. (En el ejemplo, el tipo AVENTURA le corresponde al préstamo de clave LE#=1, LI#=1 y al préstamo de clave LE#=1, LI#=3)

Si pasamos este esquema de E/R a tablas del modelo relacional tenemos:

PRESTAMO(LE#,LI#,NOMBRE,CIUDAD,NHAB,TITULO,TIPO)
TIPO_LIBROS(TIPO)
PRESTAMO_TIPO_LIBROS(LE#,LI#,TIPO)
donde las claves están subrayadas.

Las extensiones serían:

PRESTAMO

LE#	LI#	NOMBRE	CIUDAD	NºHAB	TITULO	TIPO
1	1	PEREZ	SEVILLA	500000	ENAMORADOS HASTA LA MED	ROMANTICO
1	3	PEREZ	SEVILLA	500000	TIJUANA JONES	AVENTURA
2	4	SMITH	SEVILLA	500000	LA BIBLIA	RELIGIOSO
2	5	SMITH	SEVILLA	500000	EL CORAN	RELIGIOSO
3	1	GONZALEZ	BARCELONA	3000000	ENAMORADOS HASTA LA MED	ROMANTICO

TIPO_LIBROS

TIPO_LIBRO
ROMANTICO
AVENTURA
RELIGIOSO
COMICO

PRESTAMO_TIPO_LIBROS

LE#	LI#	TIPO_LIBRO
1	1	AVENTURA
1	1	ROMANTICO
1	3	AVENTURA
1	3	ROMANTICO
2	4	RELIGIOSO
2	4	COMICO
2	5	RELIGIOSO
2	5	COMICO
3	1	ROMANTICO

No existen atributos multievaluados. Se elimina el problema 1 anterior.

Los problemas 2,3 y 4 anteriores se siguen produciendo.

SEGUNDA FORMA NORMAL (2FN)

Una relación está en 2FN si está en 1FN y todo atributo que no pertenece a la clave primaria depende funcionalmente de toda la clave, y no de parte de ella.

Dependencia funcional:

Sea la relación R y X e Y subconjuntos de atributos de R. Existe una dependencia funcional entre X e Y (se representa X 0 Y) o bien Y depende funcionalmente de X si $P_Y(S_{X='x'}(R))$ tiene como máximo una tupla. Es decir, todas las tuplas en las que X toma el mismo valor, Y toma el mismo valor.

Ejemplo: En las tablas anteriores hay varias dependencias funcionales:

LE# 0 NOMBRE

ya que $P_{NOMBRE}(S_{LE\#=1}(PRESTAMO))$ tiene como única tupla a <'PEREZ'>,
$P_{NOMBRE}(S_{LE\#=2}(PRESTAMO))$ tiene como única tupla a <'SMITH'>
$P_{NOMBRE}(S_{LE\#=3}(PRESTAMO))$ tiene como única tupla a <'GONZALEZ'>

NOMBRE 0 CIUDAD
CIUDAD 0 NHAB
y varias más.

Volviendo al ejemplo anterior, ¿la relación PRESTAMO está en 2FN?, o lo que es lo mismo, ¿todo atributo que no pertenece a la clave primaria depende funcionalmente de toda la clave y no de parte de ella?

Son atributos no primarios NOMBRE, CIUDAD, NHAB, TITULO y TIPO. La clave es LE# LI#.

Existen las dependencias
LE# 0 NOMBRE
LE# 0 CIUDAD
LE# 0 NHAB
LI# 0 TITULO
LI# 0 TIPO

por lo que todos los atributos no primarios dependen parcialmente de la clave. Por lo tanto, no está en 2FN.

Descomposición.

Una relación R que no está en 2FN se descompone en las relaciones formadas por
- la clave y los atributos totalmente dependientes de ella.
- cada subconjunto de la clave y los atributos que son totalmente dependientes de dicho subconjunto.

Sea R(X,Y,Z,P) donde

XY es clave

Z son los atributos totalmente dependientes de la clave XY 0 Z

P son los atributos parcialmente dependientes de la clave X 0 P

se descompone en R(X,Y,Z) y en R(X,P)

Ejemplo: PRESTAMO(LE#,LI#,NOMBRE,CIUDAD,NHAB,TITULO,TIPO) con

LE# 0 NOMBRE, LE# 0 CIUDAD, LE# 0 NHAB, LI# 0 TITULO, LI# 0 TIPO

se descompone en

PRESTAMO1(LE#,LI#) No hay atributos totalmente dependientes de la clave

PRESTAMO2(LE#,NOMBRE,CIUDAD,NHAB)

PRESTAMO3(LI#,TITULO,TIPO)

donde

PRESTAMO1

LE#	LI#
1	1
1	3
2	4
2	5
3	1

PRESTAMO2

LE#	NOMBRE	CIUDAD	NºHAB
1	PEREZ	SEVILLA	500000
2	SMITH	SEVILLA	500000
3	GONZALEZ	BARCELONA	3000000

PRESTAMO3

LI#	TITULO	TIPO
1	ENAMORADOS HASTA MEDULA	ROMANTICO
3	TIJUANA JONES	AVENTURA
4	LA BIBLIA	RELIGIOSO
5	EL CORAN	RELIGIOSO

- No existe la anomalía 2 ni parte de la 3.
- Queda la anomalía 4 y la 3 queda así:

Anomalía 3': Si González devuelve su libro, se pierde información sobre los tipos de libros que le gustan. (Ver tabla PRESTAMO_TIPO_LIBROS).

- Se cumple que

PRESTAMO = PRESTAMO2 $X_{LE\#}$ PRESTAMO1 $X_{LI\#}$ PRESTAMO

Es importante que al descomponer una relación R(X,Y,Z) en dos relaciones S(X,Y) y T(X,Z) se cumpla que R=S X_X T para que no se añada ni se elimine información.

Ej: Si tenemos R(X,Y,Z)

R

X	Y	Z
A	B	D
A	C	E

y la descomponemos en S(X,Y) y T(X,Z)

X	Y
A	B
A	C

X	Z
A	D
A	E

S X_X T

X	Y	Z
A	B	D
A	B	E
A	C	D
A	C	E

No se cumple que R(X,Y,Z)=R[X,Y] X_X R[X,Z]

Teorema:
Sea R(X,Y,Z) se cumple que
si X 0 Y o X 0 Z entonces R(X,Y,Z)=R[X,Y] Xx R[X,Z]

Tal y como se ha descompuesto la relación R(X,Y,Z,P) en R(X,Y,Z) y en R(X,P) para que estuvieran en 2FN, se puede ver que
R(X,Y,Z,P)=R[X,Y,Z] Xx R[X,P] ya que X 0 P
Por lo tanto, la descomposición es válida.

TERCERA FORMA NORMAL (3FN)

Una relación está en 3FN si está en 2FN y no hay dependencias transitivas con claves candidatas. Una clave candidata es una clave primaria o una alternativa que proporciona un conjunto mínimo de atributos.

Dependencia transitiva:
Sea la relación R con X,Y y Z subconjuntos de atributos. Se dice que Z depende transitivamente de X en la relación R si se cumple
X 0 Y, Y]0 X, Y 0 Z
X 0 Z, Z]0 X
Para que exista dependencia transitiva el grado de la relación ha de ser mayor o igual que 3.
Ejemplo: DNI(NUMERO,NOMBRE,APELLIDOS,POBLACION,PROVINCIA,...)
NUMERO 0 POBLACION, POBLACION]0 NUMERO, POBLACION 0 PROVINCIA (Suposición)
NUMERO 0 PROVINCIA, PROVINCIA]0 NUMERO
POR LO TANTO, PROVINCIA DEPENDE TRANSITIVAMENTE DE NUMERO.

En el ejemplo anterior existe una dependencia transitiva en la tabla PRESTAMO2.

LE#	NOMBRE	CIUDAD	NºHAB
1	PEREZ	SEVILLA	500000
2	SMITH	SEVILLA	500000
3	GONZALEZ	BARCELONA	3000000

NHAB depende transitivamente de LE# ya que
LE# 0 CIUDAD, CIUDAD]0 LE#, CIUDAD 0 NHAB
LE# 0 NHAB, NHAB]0 LE#
Como existe un atributo NHAB que depende transitivamente de la clave primaria, entonces la relación PRESTAMO2 no está en 3FN.

Descomposición:

Una relación que no está en 3FN se descompone en
- una relación formada por la clave y los atributos que sólo dependen funcionalmente de ella.
- para cada determinante no contenido en la clave (si tenemos A 0 B a A se le llama determinante) se crea una relación formada por dicho determinante y los atributos que dependen de él.

Sea R(X,Y,Z) donde
X es clave y X 0 Y, Y]0 X, Y 0 Z, X 0 Z, Z]0 X
Se descompone en
R(X,Y) X es clave, Y sólo depende funcionalmente de X.
R(Y,Z) correspondiente al determinante Y 0 Z

Por lo tanto, la relación PRESTAMO2 hay que descomponerla en

PRESTAMO21 (en realidad LECTOR)

LE#	NOMBRE	CIUDAD
1	PEREZ	SEVILLA
2	SMITH	SEVILLA
3	GONZALEZ	BARCELONA

PRESTAMO22 (en realidad CIUDAD)

CIUDAD	NºHAB
SEVILLA	500000
BARCELONA	3000000

- Ya no existe la anomalía 4.
- La anomalía 3´ continúa.
- Se cumple PRESTAMO2 = PRESTAMO21 X_{CIUDAD} PRESTAMO22 ya que CIUDAD 0 NHAB

FORMA NORMAL DE BOYCE-CODD (BCFN)

Una relación está en BCFN si y sólo si todo determinante es una clave candidata, es decir, si en una relación R(X,Y,Z) existe una dependencia Y 0 Z entonces Y debe ser clave candidata, tienen que existir las dependencias funcionales con todos los demás atributos de la relación, en este caso Y 0 X.

Otra definición del concepto de clave: Si tenemos R(A1,A2,...,An) y se cumple .i(Aj 0 Ai) entonces Aj es una clave candidata. Esta es
Ejemplo: Dada la relación R(CIUDAD,DIRECCION,COD_POSTAL)
Otra forma de encontrar la clave es la siguiente:
¿CIUDAD es clave?
¿CIUDAD 0 COD_POSTAL? No ¿CIUDAD 0 DIRECCION? No
¿DIRECCION es clave?
¿DIRECCION 0 CIUDAD? No ¿DIRECCION 0 COD_POSTAL? No
¿COD_POSTAL es clave?
¿COD_POSTAL 0 CIUDAD? Sí ¿COD_POSTAL 0 DIRECCION? No
¿CIUDAD DIRECCION es clave?
¿CIUDAD DIRECCION 0 COD_POSTAL? Sí (Es clave)
¿CIUDAD COD_POSTAL es clave?
¿CIUDAD COD_POSTAL 0 DIRECCION? No
¿DIRECCION COD_POSTAL es clave?
¿DIRECCION COD_POSTAL 0 CIUDAD? Sí (Es clave)
Existen dos claves candidatas: CIUDAD DIRECCION y DIRECCION COD_POSTAL
Se escoge una de ellas como clave primaria, por ejemplo CIUDAD DIRECCION.

R

CIUDAD	DIRECCION	COD_POSTAL
X	AQUI	31011
X	ALLI	31013
Y	ALLA	28010
Y	AQUI	28010
Z	LEJOS	26001

Esta relación está en 3FN ya que no existe más que un atributo que no pertenece a la clave primaria y se necesitan dos por lo menos.

Sin embargo, se producen anomalías:
1.- No se puede añadir la información de que el código 26002 es de la ciudad Z si no conocemos la dirección.

CIUDAD=Z
DIRECCION=? ¡Pero DIRECCION es parte de la clave!
COD_POSTAL=26002

Esta es una anomalía de inserción.

2.- Si se borra la dirección AQUI de la ciudad X se borra la información de que el código 31011 es de la ciudad X. Esta es una anomalía de borrado.

3.- Si queremos cambiar el código postal 28010 por 28014 tenemos que hacerla en todas las tuplas en los que aparezca. Esta es una anomalía de actualización.

Todo esto sucede porque en la relación R hay un determinante COD_POSTAL que no es clave candidata. Es determinante ya que COD_POSTAL 0 CIUDAD.

Descomposición:

La relación en 3FN que no está en BCFN se descompone separando los atributos que dependen de determinantes que no son claves candidatas. Se obtiene
- para cada determinante que no es clave candidata una relación con dicho determinante y los atributos que dependen de él.
- una relación con los determinantes y el resto de atributos.

Sea R(X,Y,Z) donde
Y 0 Z, Y]0 X, Y es determinante pero no clave candidata
se descompone en
R(Y,Z) correspondiente a Y 0 Z
R(Y,X) determinante Y y resto de atributos no dep. funcionalmente

Ejemplo: R(CIUDAD,DIRECCION,COD_POSTAL) se descompone en
R1(COD_POSTAL,DIRECCION) y R2(COD_POSTAL,CIUDAD)
donde COD_POSTAL]0 DIRECCION y COD_POSTAL 0 CIUDAD

R1

COD_POSTAL	DIRECCION
31011	AQUI
31013	ALLI
28010	ALLA
28010	AQUI
26001	LEJOS

R2

COD_POSTAL	CIUDAD
31011	X
31013	X
28010	Y
26001	Z

- Se cumple que R = R1 X_{COD_POSTAL} R2 ya que COD_POSTAL 0 CIUDAD

Y ya no existen las anomalías anteriores:

- Se puede añadir la información de que el código 26002 es de la ciudad Z en R2.
- Se puede borrar la dirección AQUI de la ciudad X sin borrar la información de que el código 31011 es de la ciudad X. Sólo hay que borrar la tupla <31011,AQUI> de R1.
- Si se quiere cambiar el código 28010 por 28014 se puede hacer en R2. Si se mantiene la restricción de integridad referencial el sistema cambiará automáticamente en R1, 28010 por 28014.

Es posible que no nos importen demasiado estas anomalías ya que una vez que tengamos todas las direcciones, ciudades y códigos postales correctamente en la tabla R no modifiquemos esta información en mucho tiempo. Aunque en otros casos estas anomalías pueden ser muy molestas.

Sin embargo, lo que sí producen estas dependencias es una gran redundancia de información.

Cálculo de la redundancia de información:

Supongamos que hay x tuplas en la tabla R correspondientes a x direcciones completas.

cada código postal ocupa 5 caracteres
cada nombre de ciudad ocupa 15 caracteres
cada dirección ocupa 30 caracteres
cada dirección completa en R ocupa 5+15+30=50 caracteres

En la tabla R hay entonces 50x caracteres.

En la tabla R1 habrá x tuplas, y cada tupla tendrá 5+30=35 caracteres. Por lo tanto, habrá 35x caracteres.

Suponemos que en R2 hay y tuplas, correspondientes a y códigos postales distintos. Cada tupla tendrá 5+15=20 caracteres. Por lo tanto, en R2 habrá 20y caracteres.

número de caracteres redundantes=50x - 35x - 20y=15x - 20y

15x - 20y >= 0 = x >= (4/3)y =
= núm.direcciones completas >= (4/3) núm.códigos postales

Volviendo al ejemplo inicial de la biblioteca, todavía no se ha conseguido resolver la anomalía 3´. Si González devuelve su libro, se pierde información sobre los tipos de libros que le gustan.

PRESTAMO_TIPO_LIBROS

LE#	LI#	TIPO_LIBRO
1	1	AVENTURA
1	1	ROMANTICO
1	3	AVENTURA
1	3	ROMANTICO
2	4	RELIGIOSO
2	4	COMICO
2	5	RELIGIOSO
2	5	COMICO
3	1	ROMANTICO

Se borraría la tupla <3,1,ROMANTICO> que indica que al lector de código LE#=3 (González) le gusta el TIPO_LIBRO=ROMANTICO, independientemente del libro que haya cogido en préstamo (en este caso el de código LI#=1).

CUARTA FORMA NORMAL (4FN)

Una relación está en 4FN si y sólo si para toda dependencia multievaluada X 0⇌ 0 Y en R se cumple que el resto de atributos de R son funcionalmente dependientes de X.

Dependencia multievaluada:

Existe en una relación R(X,Y,Z) una dependencia multievaluada entre los atributos X e Y (X 0⇌ 0 Y) si se verifica que
si las tuplas (x,y1,z1) y (x,y2,z2) están en R también lo están (x,y1,z2) y (x,y2,z1)
ejemplo: LE# 0⇌ 0 TIPO_LIBRO
LE# 0⇌ 0 LI#

En el ejemplo anterior, la tabla PRESTAMO_TIPO_LIBROS no está en 4FN ya que LE# 0⇌ 0 TIPO_LIBRO y sin embargo LE#]0 LI#

Descomposición:

Una relación R que no está en 4FN porque existe una dependencia multievaluada X 0⇌ 0 Y y X]0 Z se descompone en R(X,Y) y en R(X,Z)
Ejemplo: La tabla PRESTAMO_TIPO_LIBROS se descompone en
PRESTAMO_TIPO_LIBROS1 PRESTAMO_TIPO_LIBROS2

LE#	TIPO_LIBRO
1	AVENTURA
1	ROMANTICO
2	RELIGIOSO
2	COMICO
3	ROMANTICO

LE#	LI#
1	1
1	3
2	4
2	5
3	1

- PRESTAMO_TIPO_LIBROS1 contiene información de qué libros le gustan a cada lector.
- PRESTAMO_TIPO_LIBROS2 indica qué libros están prestados a qué lectores. Es igual a la tabla PRESTAMO2 obtenida anteriormente.
- Se cumple que

PRESTAMO_TIPO_LIBROS = PRESTAMO_TIPO_LIBROS1 XLE# PRESTAMO_TIPO_LIBROS2
y sin embargo LE#]0 TIPO_LIBRO, LE#]0 LI#

Teorema: (Asegura que la descomposición de la tabla es correcta)

Sea R(X,Y,Z) con X 0≠≠ 0 Y

entonces R(X,Y,Z) = R[X,Y] Xx R[X,Z]

La anomalía 3´ ya no existe. Si González devuelve su libro entonces se borra la tupla <1,3>

Tras el proceso de normalización hemos descompuesto la tabla inicial

PRESTAMO(LE#,LI#,NOMBRE,CIUDAD,NHAB,TIPO_LIBROS,TITULO,TIPO)

y hemos obtenido las tablas:

TIPO_LIBROS

TIPO_LIBRO
ROMANTICO
AVENTURA
RELIGIOSO
COMICO

PRESTAMO

LE#	LI#
1	1
1	3
2	4
2	5
3	1

LIBRO

LI#	TITULO	TIPO
1	ENAMORADOS HAST MEDULA	ROMANTICO
3	TIJUANA JONES	AVENTURA
4	LA BIBLIA	RELIGIOSO
5	EL CORAN	RELIGIOSO

LECTOR

LE#	NOMBRE	CIUDAD
1	PEREZ	SEVILLA
2	SMITH	SEVILLA
3	GONZALEZ	BARCELONA

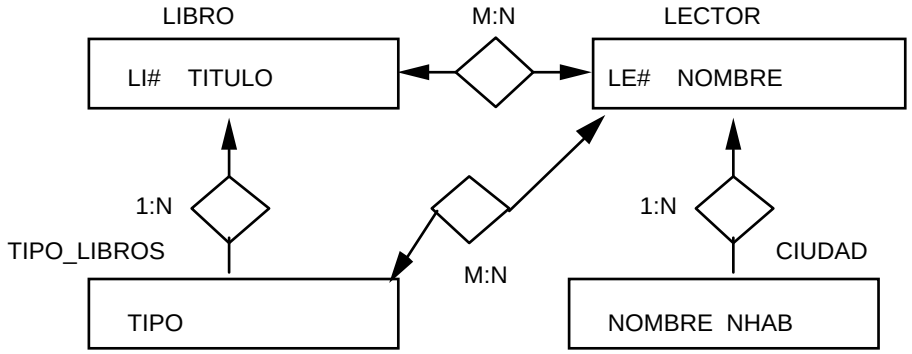
CIUDAD

CIUDAD	NºHAB
SEVILLA	500000
BARCELONA	3000000

LECTOR_TIPO_LIBROS

LE#	TIPO_LIBRO
1	AVENTURA
1	ROMANTICO
2	RELIGIOSO
2	COMICO
3	ROMANTICO

Estas son las tablas que habrían salido si el esquema E/R hubiera sido de la siguiente forma:



QUINTA FORMA NORMAL (5FN)

Una relación está en 5FN si no presenta dependencias de join no triviales (no basadas en claves candidatas).

Dependencia de join:

Existe una dependencia de join en una relación R si ésta se puede expresar como una combinación de los operadores de proyección y de join entre sus atributos.

Dada una relación R con atributos X,Y,Z existe una dependencia de join si se cumple que $R(X,Y,Z) = R[X,Y] X_Y R[Y,Z] X_{Z,X} R[Z,X]$

Una dependencia de join trivial sería: $R(\underline{X},Y,Z) = R[X,Y] X_X R[X,Z]$ porque está basada en la clave candidata X.

Ejemplo de una relación que no está en 5FN:

La siguiente tabla contiene información sobre los productos representados por distintas compañías vendidos por los agentes.

AG-PR-CO

AGENTE#	PRODUCTO#	COMPAÑIA#
A1	P1	C2
A1	P2	C1
A2	P1	C1
A1	P1	C1

Esta tabla no está en 4FN ya que <A1 P1 C2> y <A1 P2 C1>
están en AG-PR-CO y sin embargo <A1 P2 C2> no lo está.

Sin embargo, supongamos que se cumple la siguiente regla simétrica: *"Si un agente A vende un producto P, y el agente A representa a la compañía C y la compañía C representa al producto P, entonces el agente A vende el producto P de la compañía C"*. En realidad, esta es una manera de expresar lo siguiente:

AG-PR-CO(A#,P#,C#)=AG-PR-CO[A#,P#] X AG-PR-CO[A#,C#] X AG-PR-CO[P#,C#]

Esto quiere decir que existe una dependencia de join no trivial y que la tabla AG-PR-CO se puede descomponer en esas tres proyecciones.

La 5FN es la última forma normal basada al hacer la descomposición en operaciones de proyección y join (por definición de dependencia de join).

Hasta ahora, para obtener las formas normales había que ver si existían dependencias funcionales, transitivas y/o multievaluadas. Pero para ver si está en 5FN no hay una técnica directa de este estilo, hay que ver si existen restricciones de simetría como la anterior, es decir, si existen dependencias de join.

AG#	**PR#**
A1	P1
A1	P2
A2	P1

AG#	**CO#**
A1	C2
A1	C1
A2	C1

PR#	**CO#**
P1	C2
P2	C1
P1	C1

En resumen, gracias a este proceso de normalización se garantiza que no se produzcan anomalías de inserción, borrado y/o modificación, y además se evita redundancia. En realidad la normalización indirectamente nos ha ayudado en el diseño del esquema conceptual. Si se hubiera diseñado este esquema E/R desde el principio, no habría habido problemas de anomalías ni de redundancia.

Sin embargo, como resultado de la normalización se obtienen más tablas, y por lo tanto las preguntas son más difíciles de expresar ya que tienen más joins y generalmente más caras de responder.

Por ejemplo si se quieren *obtener los tipos de libros que le gustan a los sevillanos/as*.

1) Teniendo la entidad no normalizada inicial, PRESTAMO, la pregunta expresada con las operaciones del álgebra relacional sería:

$\pi_{\text{TIPO\_LIBRO}} (\text{S}_{\text{CIUDAD = 'SEVILLA'}} (\text{PRESTAMO}))$

2) Pero con las tablas obtenidas tras la normalización, la pregunta sería:

$\pi_{\text{TIPO\_LIBRO}} (\text{S}_{\text{CIUDAD = 'SEVILLA'}} (\text{LECTOR\_TIPO\_LIBROS} \ \text{X}_{\text{LE\#}} \ \text{LECTOR}))$

Una posible solución para que la pregunta no sea tan complicada podría ser ofrecer una vista PRESTAMO sobre las tablas normalizadas, aunque el problema de que se ejecute el join, que es muy caro, sigue existiendo.

Por esta razón, si no importa mucho que exista redundancia o que se produzcan anomalías de cualquier tipo ya que no van a ser frecuentes las actualizaciones, borrados o inserciones, a veces las tablas no interesa que estén totalmente normalizadas

3.4 ¿ES INTERESANTE NORMALIZAR SIEMPRE?

Sea la tabla siguiente:
PERSONA(NOMBRE,CALLE,COD_POSTAL,CIUDAD)
que no está en 3FN ya que existe una dependencia transitiva entre los atributos NOMBRE y CIUDAD

NOMBRE 0 COD_POSTAL NOMBRE 0 CIUDAD
COD_POSTAL]0 NOMBRE + CIUDAD]0 NOMBRE
COD_POSTAL 0 CIUDAD

Para que la relación esté en 3FN hay que descomponerla así:

PERSONA(NOMBRE,CALLE,COD_POSTAL)
CODIGOS(COD_POSTAL,CIUDAD)

VENTAJAS:

1.- **No habrá tanta redundancia.**
Que el código postal X es de la ciudad Y sólo aparecerá una vez en una tupla de la tabla CODIGOS y no en tantas tuplas como aparece X en la tabla PERSONA.

2.- **No se producen anomalías de borrado.**
Si se borra la última persona que vive en un mismo código postal no se pierde la información de la ciudad a la que pertenece.

3.- **No se producen anomalías de actualización.**
Si se cambia la ciudad correspondiente a un determinado código postal sólo hay que hacerlo en una tupla y no en tantas como aparece en la tabla persona.
Si se cambia el código postal correspondiente a una ciudad sólo hay que hacerlo en una tupla, siempre que el sistema de gestión de base de datos relacional garantice la integridad referencial, es decir, que sea el propio sistema el que se encargue de cambiar un código postal por el otro en la tabla persona que es la que contiene la clave extranjera cod_postal. Los SGBDR comerciales no garantizan la integridad referencial precisamente por ser muy cara.

4.- **No se producen anomalías de inserción.**
Se puede insertar la información de que un determinado código postal pertenece a una ciudad sin necesidad de que exista una determinada persona en la base de datos que viva en dicho código postal.

DESVENTAJAS:

1.- **Las preguntas son más difíciles de expresar** ya que en algunos casos hay que utilizar joins y por lo tanto pueden ser **más caras de responder** ya que esta operación es la más costosa.
Ejemplo: *¿En qué ciudad vive Koldo Mitxelena?*
a) Utilizando la tabla sin normalizar PERSONA.

π_{CIUDAD} ($\sigma_{NOMBRE = 'Koldo\ Mitxelena'}$ (PERSONA))

b) Utilizando las tablas normalizadas PERSONA y CODIGOS.

π_{CIUDAD} ($\sigma_{NOMBRE = 'Koldo\ Mitxelena'}$ (PERSONA $\bowtie_{COD\_POSTAL}$ CODIGOS))

Si suponemos que:

Las anomalías de borrado y de inserción no son importantes ya que esta base de datos ante todo contiene información de personas.

Las anomalías de actualización no son importantes ya que los códigos postales y los nombres de ciudades no cambian.

La redundancia no es muy importante ya que hay espacio suficiente.

Entonces es mejor no normalizar la tabla persona ya que la ventaja al hacer las preguntas sin joins es bastante grande.

Otra posibilidad es mantener las dos tablas normalizadas y ofrecer la tabla anterior persona como una vista de ellas dos.

Así las preguntas se hacen sin joins, pero sí que se tienen que ejecutar los joins, por lo que la respuesta a las preguntas sigue siendo cara. Otro aspecto a estudiar es si la vista es actualizable o no.

3.5 EJEMPLO DE NORMALIZACION

A partir del esquema ENTIDAD/RELACION correspondiente al control de proyectos y de almacén, se obtienen las siguientes relaciones:

1.- PEDIDO(PED_NUM,PED_FECHA,PED_FECHA_LIM,PED_FECHA_ENT)

2.-PROVEEDOR(PROV_COD,PROV_NOM,PROV_DIR,
PROV_TEL,PROV_CIF_CNI,PROV_FAX,PROV_PER)

3.- PRODUCTO(PROD_COD,PROD_NOM,PRO_DISP,
PROD_STOCK_MIN,PRO_PRE_STAN,PROD_TIPO)

4.- PED_PROV_PROD(PED_NUM,PROV_COD,PROD_COD,CANT_PED,PRECIO_PED)

5.- PROV_PROD(PROV_COD,PROD_COD,CANT_MIN,
CANT_MAX,PRECIO,FECHA,DESCUENTO)

6.- PROYECTO(PROY_COD,PROY_NOM,PROY_JEFE,PROY_FI,PROY_FF,PROY_FREAL,
PROY_ESTADO,PROY_COSTO_EST,PROY_COSTO_REAL,CLI_COD)

7. PROD_PROY(PROD_COD,PROY_COD,CANT_ESTIM,PRECIO_ESTIM,PRECIO_REAL)

8.- CLIENTE(CLI_COD,CLI_CIF_DNI,CLI_NOM,CLI_DIR,CLI_TEL)

9.- PERSONAS(PER_COD,PER_DNI,PER_NOM,PER_PREC_HORA)

10.- PROY_PER(PROY_COD,PER_COD,HORAS_EST,HORAS_REAL)

El proceso de normalización de estas relaciones es:

1FN: No hay atributos multievaluados en las relaciones. Están en 1FN.

2FN: Hay que comprobar si existen dependencias parciales de algún atributo con respecto a la clave. Las relaciones 1,2, 3, 6, 8 y 9 están en 2FN. La clave está formada por un único atributo.

Relación 4:
PED_PROV_PROD(PED_NUM,PROV_COD,PROD_COD,CANT_PED,PRECIO_PED)
PED_NUM]0 CANT_PED, PED_NUM]0 PRECIO_PED
PROD_COD]0 CANT_PED, PROD_COD]0 PRECIO_PED

Pero PED_NUM 0 PROV_COD, el número de pedido determina el proveedor ya que un pedido se hace a un único proveedor. Se descompone la relación en dos:

PED_PROV(PED_NUM,PROV_COD)
PED_PROV(PED_NUM,PROD_COD,CANT_PED,PRECIO_PED)
que están en 2FN.

Relación 5:
PROV_PROD(PROV_COD,PROD_COD,CANT_MIN,
CANT_MAX,PRECIO,FECHA,DESCUENTO)

Se cumple que PROV_COD PROD_COD FECHA 0 PRECIO

El precio al que un proveedor nos proporciona un producto en una determinada fecha es independiente de la cantidad comprada. La cantidad (máxima y mínima) determina el descuento pero no el precio del producto.

Se descompone la relación en estas dos:

PROV_PROD_PRECIO(PROV_COD,PROD_COD,FECHA,PRECIO)
PROV_PROD_DESC(PROV_COD,PROD_COD,FECHA,
CANT_MIN,CANT_MAX,DESCUENTO)

Relación 7:
PROD_PROY(PROD_COD,PROY_COD,CANT_ESTIM,PRECIO_ESTIM,PRECIO_REA)
PROD_COD]0 CANT_ESTIM, PROD_COD]0 PRECIO_ESTIM,
PROD_COD]0 PRECIO_REAL
PROY_COD]0 CANT_ESTIM, PROY_COD]0 PRECIO_ESTIM,
PROY_COD]0 PRECIO_REAL

No hay dependencias parciales con la clave.

Relación 10:
PROY_PER(PROY_COD,PER_COD,HORAS_EST,HORAS_REAL)
PROY_COD]0 HORAS_EST, PROY_COD]0 HORAS_REAL,
PER_COD]0 HORAS_EST, PER_COD]0 HORAS_REAL,

No hay dependencias parciales con la clave.

3FN: No hay dependencias transitivas en ninguna relación.

BCFN: Todos los determinantes son claves candidatas.

4FN: No hay dependencias multievaluadas.

5FN: no presenta dependencias de join no triviales (no basadas en claves candidatas)

4. LENGUAJES RELACIONALES

Los lenguajes relacionales deben incluir:

Lenguajes de definición de datos

Para definir tablas de base, vistas, índices, privilegios de acceso, restricciones de integridad, ...

Lenguajes de manipulación de datos

Para recuperación de datos de las tablas (lectura). Estos lenguajes deben poseer la potencia del álgebra relacional.

Para inserción y actualización de datos (escritura).

Hay varios tipos de lenguajes relacionales, que aunque se distinguen entre sí principalmente por cómo es el lenguaje de manipulación, también incluyen lenguajes de definición de datos.

lenguajes algebraicos: SQL

lenguajes de cálculo relacional de tuplas: QUEL

lenguajes de cálculo relacional de dominios: QBE

*El **SQL** (STRUCTURED QUERY LANGUAGE) es el lenguaje relacional que más éxito ha tenido y que ha sido adoptado como un estándar. Prácticamente todos los Sistemas de Gestión de Bases de Datos lo proporcionan aunque con distintas características. En realidad, el SQL no es el que más se ajusta al modelo relacional.*

A partir de ahora vamos a centrarnos sólo en el SQL.

4.1. LENGUAJE DE MANIPULACION DE DATOS

La relación es vista como un conjunto de tuplas sobre las que se pueden realizar las operaciones típicas de conjuntos: UNION, INTERSECCION, DIFERENCIA, PRODUCTO CARTESIANO y otras típicamente relacionales como son la SELECCION, PROYECCION, JOIN y DIVISION. Todas estas operaciones son cerradas.

Suponemos que tenemos las tablas siguientes:
S (S#,NOMS,ESTADO,CIUDAD)
P (P#,NOMP,COLOR,PESO,CIUDAD)
SP (S#, P#,CTD)

1) PROYECCION.

SELECT [UNIQUE] <lista_atributos>
FROM <nombre_relaciones>

Por ejemplo: SELECT P# FROM SP
obtiene los códigos P# de las tuplas de la relación SP.

No elimina elementos duplicados. Para que lo haga hay que poner UNIQUE.

```
SELECT *
FROM SP
```

El * hace que se proyecten todos los atributos de la relación, por lo tanto es como si se hubiera hecho:

```
SELECT S#, P#, CTD
FROM SP
```

También se permite hacer:

```
SELECT 'Código', P#, 'Color', COLOR, 'Peso más 100', PESO+100
FROM P
```

2) SELECCION.

SELECT [UNIQUE] <lista_atributos>
FROM <nombre_relaciones>
WHERE <condición>

<condición> admite operadores de comparación (=,<,>,...) y operadores booleanos (and,or,not)

Obtener los suministradores que viven en Londres.

```
SELECT *
FROM S
WHERE CIUDAD = 'Londres'
```

Obtener los suministradores de Londres con estado > 5.

```
SELECT *
FROM S
WHERE ESTADO > 5 AND CIUDAD = 'Londres'
```

Obtener las piezas que son o de color rojo o con peso < 10.

```
SELECT NOMP
FROM P
WHERE COLOR = 'Rojo' OR PESO <10
```

3) JOIN.

Obtener los nombres de los suministradores y los códigos de las piezas que suministran.

```
SELECT NOMS,P#
FROM S,SP
WHERE S.S#=SP.S#
```

Obtener los nombres de los suministradores y los nombres de las piezas que suministran.

```
SELECT NOMS,NOMP
FROM S,P,SP
WHERE S.S#=SP.S# AND P.P#=SP.P#
```

Obtener los suministradores que viven en la misma ciudad. Para ello hay que hacer un join de la tabla S con ella misma y comprobar si son de la misma ciudad.

```
SELECT NOMS
FROM S,S
WHERE S.CIUDAD = S.CIUDAD
```

pero así no se puede distinguir si S.CIUDAD es de la primera relación o de la segunda. Se necesita utilizar variables rango.

```
SELECT UNO.S#,DOS.S#
FROM S UNO, S DOS
WHERE UNO.CIUDAD = DOS.CIUDAD AND
        UNO.S# <> DOS.S#
```

Devuelve pares de códigos de suministradores distintos que viven en la misma ciudad.

4) DIVISION

Dada una relación R con atributos A1,A2,...,An y otra relación S con atributos Ap+1,...,An. La relación división T entre R y S es una relación con atributos A1,...,Ap y cuyas tuplas son todas aquellas que concatenadas a cada una de las tuplas de S da siempre una tupla de R.

La tupla <t1,t2,...,tp> es de T si para toda tupla de S se cumple que
<t1,t2,...,tp> . <sp+1,...,sn> es de R.

Ejemplo: Obtener el código de los suministradores que facilitan la pieza P2.

En este caso R=SP[S#,P#]
S= REL1(P#) con una única tupla <'P2'>
T=REL2(S#)

Las tuplas de T cumplirán que <S> . <'P2'> es una tupla de R.

```
SELECT S#
FROM S
WHERE EXISTS (SELECT *
              FROM SP
              WHERE SP.S#=S.S# AND SP.P#='P2')
```

Aunque también se podía haber hecho

```
SELECT UNIQUE S#
FROM SP
WHERE P#='P2'
```

En SQL la función EXISTS(SELECT * FROM ...) devuelve true si existe alguna tupla en la tabla argumento o false en caso contrario. Esta es la forma de expresar en SQL que para una cierta tupla todas las tuplas de otra tabla relacionadas o no con esta cumplen una cierta condición.

En vez de encontrar cuál es el conjunto división es mejor expresar la pregunta de forma lógica y luego pasarla a SQL.

Ejemplo: Obtener el nombre de los suministradores que facilitan todas las piezas.

s.nombre tal que . p □ P (SP(s,p))
s.nombre tal que □]' p □ P (□]SP(s,p))

```
SELECT NOMS
FROM S
WHERE NOT EXISTS(
                    SELECT *
                    FROM P
                    WHERE NOT EXISTS(
                                        SELECT *
                                        FROM SP
                                        WHERE SP.S#=S.S# AND
                                              SP.P#=P.P#))
```

Ejemplo: Obtener el nombre de los suministradores que faciliten al menos todas las piezas facilitadas por el suministrador S2.

s.nombre tal que .p □ P (SP(S2,p) 0 SP(s,p))
s.nombre tal que]' p □ P](SP(S2,p) 0 SP(s,p))
s.nombre tal que]' p □ P]( ]SP(S2,p) o SP(s,p))
s.nombre tal que]' p □ P (SP(S2,p) y]SP(s,p))

```
SELECT NOMS FROM S
WHERE NOT EXISTS( SELECT * FROM P
                        WHERE EXISTS(SELECT * FROM SP
                                        WHERE S#= ´S2´ AND
                                              P#=P.P#)
                        AND NOT EXISTS( SELECT * FROM SP
                                        WHERE S#=S.S#
                                    AND P#=P.P#))
```

5) UNION

<relación> UNION [ALL] <relación>
[UNION [ALL] <relación> ...]

Las relaciones que participan en la unión han de tener el mismo grado y los atributos que se emparejan han de ser del mismo tipo (los i-ésimos atributos tienen que ser compatibles).

Si se pone ALL entonces al hacer la unión no se eliminan los duplicados. El sistema responde más de prisa ya que se evita tener que recorrer la tabla resultante para eliminar los elementos repetidos. Cuando se sepa que en la unión no van a aparecer duplicados es mejor poner ALL en la pregunta.

Ejemplo: Obtener el código de pieza para aquellas piezas que pesen más de 16 kilos o que sean facilitadas por el suministrador S2.

```
SELECT P#
FROM P
WHERE PESO > 16
        UNION
SELECT P#
FROM SP
WHERE S#='S2'
```

Ejemplo: Obtener el nombre de suministrador junto con los códigos de pieza que suministra. Si no suministra ninguna pieza entonces aparecerán blancos en el campo P#.

```
(SELECT NOMS,P#
FROM S,SP
WHERE S.S#=SP.S#)
        UNION ALL
(SELECT NOMS,'     '
FROM S
WHERE NOT EXISTS( SELECT * FROM SP
                          WHERE SP.S#=S.S#))
```

Se supone que el campo P# de la tabla SP sea CHAR(5)
Se pone ALL ya que se sabe seguro que no habrá duplicados.

6) INTERSECCION

Sean dos relaciones REL1 y REL2. Si se quieren obtener aquellos elementos que están en REL1 y en REL2 suponiendo que un elemento está en REL1 y en REL2 si existe una tupla en REL1 que toma el mismo valor para un atributo C# que otra tupla de REL2.

```
SELECT REL1.C#
FROM REL1
WHERE EXISTS(SELECT *
             FROM REL2
             WHERE REL1.C#=REL2.C#)
```

Ejemplo: Obtener los suministradores que suministran algún producto.

```
SELECT NOMS
FROM S
WHERE EXISTS(      SELECT *
             FROM SP
             WHERE SP.S#=S.S#)
```

Esta pregunta se podía haber expresado en forma lógica primero

s.nombre tal que ' p □ SP (SP(s,p))

y después expresarla en SQL.

7) DIFERENCIA

Sean dos relaciones REL1 y REL2. Si se quieren obtener aquellos elementos que están en REL1 pero no en REL2 suponiendo que un elemento está en REL1 y no en REL2 si no existe una tupla en REL2 que tome el mismo valor que el que toma la tupla de REL1 en el atributo C#.

```
SELECT REL1.C#
FROM REL1
WHERE NOT EXISTS( SELECT *
                  FROM REL2
                  WHERE REL1.C#=REL2.C#)
```

Ejemplo: Obtener los suministradores que no suministran ningún producto.

```
SELECT NOMS
FROM S
WHERE NOT EXISTS( SELECT *
                  FROM SP
                  WHERE SP.S#=S.S#)
```

Esta pregunta se podía haber expresado en forma lógica primero

s.nombre tal que]' p □ SP (SP(s,p))

y después haberla expresado en SQL.

8) FUNCIONES AGREGADAS

Se pueden utilizar una serie de funciones especiales, llamadas agregadas, para aumentar el poder de expresión del SQL y para que una serie de preguntas puedan realizarse de manera sencilla: ¿Cuántos proveedores hay? ¿Cuántos proveedores hay en Londres?

Estas funciones son:

COUNT: número de elementos de una columna
SUM: suma de los valores de una columna
AVG: promedio de los valores de una columna
MAX: máximo valor en una columna
MIN: mínimo valor en una columna

Ejemplo: Obtener el número de suministradores.

```
SELECT COUNT(S#)
FROM S
```

Ejemplo: Obtener el número de suministradores que suministran algún producto.

```
SELECT COUNT( UNIQUE S#)
FROM SP
```

Ejemplo: Obtener la cantidad total suministrada del producto P2

```
SELECT SUM(CTD)
FROM SP
WHERE P#='P2'
```

Ejemplo: Obtener los suministradores con estado menor que el máximo valor de estado en la tabla S.

```
SELECT S#
FROM S
WHERE ESTADO < (SELECT MAX(ESTADO) FROM S)
```

GROUP BY: Agrupa las tuplas de una tabla por el atributo que se le indica, y realiza las operaciones según dicho atributo.

Sea la tabla SP:

S#	P#	CTD
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

```
SELECT P#,SUM(CTD)
FROM S
GROUP BY P#
```

Devuelve la tabla

P#	---
P1	600
P2	1000
P3	400
P4	500
P5	500
P6	100

ORDER BY: La tabla resultante se ordena por el campo que se especifica.

Ejemplo: Obtener todos los productos por orden de peso.

```
SELECT P#
FROM P
ORDER BY PESO
```

9) OPERACIONES DE ACTUALIZACION

Son operaciones para insertar, modificar y borrar tuplas.

INSERCION

```
INSERT
INTO <tabla> [ ( <campo> [ , <campo> ] ... ) ]
VALUES ( <constante> [ , <constante> ] ... )
```

Ejemplo:

```
INSERT INTO P
VALUES ('P7' ,'ARANDELA','GRIS',2,'ATENAS')
```

Inserta la tupla <'P7','ARANDELA','GRIS',2,'ATENAS'> en la tabla P.

```
INSERT INTO P (P#,PESO,NOMP)
VALUES ('P7',2,'ARANDELA')
```

Inserta la tupla <'P7','ARANDELA',NULL,2,NULL> en la tabla P, si se supone que se aceptan valores nulos para los atributos COLOR y CIUDAD.

Ejemplo: Queremos obtener para cada pieza, el código y el total suministrado. Además, se quiere guardar el resultado en una tabla nueva llamada NUEVA.

- Hay que crear una tabla nueva. En el modelo relacional esto se puede realizar en cualquier momento.
- Se insertan las tuplas en dicha tabla.

```
CREATE TABLE NUEVA
    (P# CHAR(6) NOT NULL
    TOTAL INTEGER);

INSERT
INTO NUEVA (P#,TOTAL)
    SELECT P#,SUM(CTD)
    FROM SP
    GROUP BY P#
```

En el caso anterior poníamos VALUES y se insertaba tupla a tupla, y en este caso se insertan todas las tuplas que devuelve el SELECT.

ACTUALIZACION

```
UPDATE <tabla>
SET <campo> = <exp escalar>
        [ , <campo> = <exp escalar> ... ]
[WHERE <predicado> ]
```

Ejemplo: Cambiar el color de la pieza P2 a amarillo, incrementar su peso en 5, y poner el valor de la ciudad a desconocido.

```
UPDATE P
SET COLOR='AMARILLO', PESO=PESO+5, CIUDAD=NULL
WHERE P#='P2'
```

Ejemplo: Poner a cero la cantidad de todos los suministradores de Londres.

```
UPDATE SP
SET CANTIDAD=0
WHERE 'LONDRES' = (SELECT CIUDAD
                   FROM S
                   WHERE S.S#=SP.S#)
```

BORRADO

```
DELETE
FROM <tabla>
[WHERE <predicado> ]
```

Ejemplo: Borrar todos los suministradores que residen en Madrid.

```
DELETE
FROM S
WHERE CIUDAD = 'MADRID'
```

Ejemplo: Borrar todas las tuplas de SP donde el suministrador sea londinense.

```
DELETE
FROM SP
WHERE 'LONDRES' = (SELECT CIUDAD
                   FROM S
                   WHERE S.S#=SP.S#)
```

Ejemplo: Borrar todos los suministradores.

```
DELETE
FROM S
```

Al hacer DELETE se vacía una relación pero no se quita su información del catálogo. Sin embargo, al hacer DROP sí.

4.2 LENGUAJE DE DEFINICION DE DATOS

Las relaciones obtenidas hay que crearlas utilizando un lenguaje de definición de datos.

TABLA DE BASE: Es una tabla que tiene un identificador y que está soportada físicamente, a diferencia de las vistas.

Operadores de definición de tablas de base:

1) Creación:

CREATE TABLE <tabla_base>
(<definición_columna> [, <definición_columna>] ...)
[otros parámetros] ;

<definición_columna> ::= <nombre_columna> <tipo_dato>
[NOT NULL [WITH DEFAULT]]

Ejemplo: CREATE TABLE PROVEEDOR (
P# CHAR(5) NOT NULL,
PNOMBRE CHAR(20) NOT NULL WITH DEFAULT,
PCOD_POS SMALLINT,
CIUDAD CHAR(15) NOT NULL WITH DEFAULT);

WITH DEFAULT rellena el campo a blancos si el tipo es char y si es int lo pone a cero.

Al crear una tabla de base, el sistema:
- Crea una relación vacía.
- Introduce su definición en el catálogo.

Se pueden crear tablas de base en cualquier momento del tiempo de vida de una aplicación. En la definición de una tabla no aparecen detalles de cómo está implementada, sus características físicas.

2) Modificación:

ALTER TABLE <tabla_base>
ADD <definición_columna>

Ejemplo: ALTER TABLE PROVEEDOR
ADD TFNO SMALLINT NOT NULL;

3) Borrado:

DROP TABLE <tabla_base>

Ejemplo: DROP TABLE PROVEEDOR

Borra la tabla PROVEEDOR del catálogo junto con los índices y vistas definidos sobre ella.

CREACION DE INDICES:

CREATE [UNIQUE] INDEX <nombre_índice>
ON <tabla_base>
(columna [orden] [, columna [orden]] ...)
[otros parámetros] ;

Ejemplo:

CREATE INDEX IND_PROV
ON PROVEEDOR
(CIUDAD ASC P# DES);

Se define un índice sobre la tabla PROVEEDOR de la siguiente forma:
Por ciudad en orden ascendente, y para los de igual ciudad por P# en orden descendente.

VISTAS

Las tablas de base son tablas que tienen un identificador y un soporte físico.
Las vistas no son tablas de base almacenadas físicamente, sino que su definición se hace en función de otras tablas.

El usuario, en principio, no tiene por qué conocer si la tabla que está utilizando es una tabla de base o es una vista.

Definición de una vista

CREATE VIEW <vista> [<columna> [, <columna>] ...]
AS <sub_pregunta>

Ejemplo:

```
CREATE VIEW S_LONDRES
AS    SELECT S#,NOMS,ESTADO
      FROM S
      WHERE CIUDAD = 'LONDRES'
```

Se define una tabla S_LONDRES en función de la tabla S, y se introduce en el catálogo dicha definición.
Cada vez que el usuario haga una pregunta sobre la tabla S_LONDRES se traducirá a la correspondiente pregunta sobre S, pero eso será transparente para él. Incluso podría ser que S hubiera sido a su vez una vista.

Ejemplo:

```
CREATE VIEW PARES_CIUDADES (PRIM,SEG)
AS    SELECT UNO.S#, DOS.S#
      FROM S UNO, S DOS
      WHERE UNO.S# <> DOS.S# AND
            UNO.CIUDAD = DOS.CIUDAD
```

Se les da los nombres PRIM y SEG a los atributos de la vista. Si no se les da nombre entonces toma los mismos que los que se proyectan en la subpregunta.

Borrado de una vista

DROP VIEW <vista>

Borra la definición de la vista del catálogo, y la de aquellas vistas definidas sobre esta vista.

Operaciones de recuperación sobre vistas:

El usuario maneja las vistas y las tablas de base de la misma manera. No tiene por qué saber si una tabla es vista o es tabla de base.

Ejemplo: Obtener los suministradores de S_LONDRES con estado mayor que 10.

```
SELECT *
FROM S_LONDRES
WHERE ESTADO > 10
```

Es el sistema el que tiene traducir dicha pregunta a la correspondiente pregunta sobre la tabla de base S, pero esa información la puede conseguir del catálogo.

```
SELECT S#,NOMS,ESTADO
FROM S
WHERE CIUDAD = 'LONDRES'  AND ESTADO > 10
```

Operaciones de actualización sobre vistas:

Cuando se quiere insertar una tupla en una vista, que es una tabla que no está directamente soportada por una tabla de base, ¿dónde se inserta? Parece que la inserción se debería hacer en la tabla que soporta directamente a la vista.

Sin embargo esto no es siempre posible:

Si tenemos la vista DEFINE VIEW V1 AS
SELECT UNIQUE COLOR,CIUDAD
FROM P

Si se quiere hacer INSERT INTO V1 VALUES ('ROJO','LONDRES') habría que introducir dicha tupla en la tabla de base P pero ¿cuál es el valor de la clave P# para dicha tupla? No se puede insertar sobre V1.

Si tenemos la vista DEFINE VIEW V2 AS
SELECT P#,CIUDAD
FROM SP,S
WHERE SP.S#=S.S#

Si se quiere insertar en V2 la tupla <'P8','MADRID'> habría que insertar una nueva tupla en SP de la forma <---,'P8',---> pero no se puede hacer sin conocer el valor de S# que es parte de la clave. No se puede insertar sobre V2.

Si tenemos la vista DEFINE VIEW V3 (P#,SUMACTD) AS
SELECT P#,SUM(CTD)
FROM SP
GROUP BY P#

Si se quiere insertar en V3 la tupla <'P8',1000> habría que insertar varias tuplas en SP sin conocer los valores de la clave S# y sin conocer los valores para CTD, pero sabiendo que estos valores desconocidos han de sumar 1000. No se puede insertar sobre V3.

El sistema DB2 sólo va a permitir que las operaciones de actualización sobre una vista repercutan en las tablas subyacentes si las vistas son actualizables. Una vista es actualizable si cumple lo siguiente:

- se deriva de una única tabla de base.
- se deriva a partir del conjunto de columnas y filas de la tabla subyacente. Las únicas operaciones para obtener esa vista son la selección y la proyección.

Ventajas de las vistas.

Las **vistas consiguen la independencia lógica**, que es la capacidad para modificar las estructuras lógicas sin tener que cambiar los programas de aplicación.

Se puede cambiar el esquema conceptual por varias razones:

-Crecimiento de la BD. Puede que haya que añadir un atributo nuevo a una tabla.

-Reestructuración del esquema conceptual.

Ejemplo:

La tabla de base S(S#,NOMS,ESTADO,CIUDAD) se podría querer dividir en S1(S#,ESTADO) y en S2(S#,NOMBRE,CIUDAD) puesto que las aplicaciones acceden casi siempre a ESTADO y el ABD ha decidido que quiere dejar S1 en un disco de acceso muy rápido. Lo que puede hacer es crear S1 y S2 como tablas base y luego definir la vista S a partir de S1 y de S2. Las aplicaciones de usuario que antes accedían a S (que era una tabla de base) ahora accederán a S igual (siendo S una vista). No tienen por qué cambiar.

```
CREATE TABLE S1
        (S# CHAR(5) NOT NULL
        NOMBRE CHAR(20) NOT NULL WITH DEFAULT,
        CIUDAD CHAR(15) NOT NULL WITH DEFAULT);
CREATE TABLE S2
        (S# CHAR(5) NOT NULL
        ESTADO SMALLINT);
CREATE UNIQUE INDEX IS1 ON S1(S#);
CREATE UNIQUE INDEX IS2 ON S2(S#);
INSERT INTO S1
        SELECT S#,NOMBRE,CIUDAD
        FROM S
INSERT INTO S2
        SELECT S#,ESTADO
        FROM S
DROP TABLE S
```

Una vez borrada del catálogo la tabla de base S se puede definir como una vista.

```
CREATE VIEW S (S#,NOMBRE,ESTADO,CIUDAD)
AS
        SELECT S1.S#,S2.NOMBRE,S1.ESTADO,S2.CIUDAD
        FROM S1,S2
        WHERE S1.S#=S2.S#
```

No es exactamente cierto que las aplicaciones no tengan que cambiar, ya que esta vista no es actualizable, por lo que las operaciones de inserción, de borrado y de modificación no se van a poder realizar sobre S. El usuario tendrá que actualizar sobre S1 y/o S2. Sin embargo, las operaciones de consulta sí se podrán realizar.

En DB2 sólo se consigue independencia lógica total si la vista es actualizable. En otro caso, la independencia lógica sólo lo es con respecto a la recuperación y los demás programas que actualizan habrán de ser cambiados.

En resumen:

- Las vistas facilitan hasta cierto grado la **independencia lógica.**

- Permite que los **datos sean vistos de distinta manera** por diferentes usuarios.

- Para aquellos usuarios finales no expertos, se les puede proporcionar una vista sencilla de los datos que sea el join de todas las tablas, para que pregunten sólo utilizando selecciones y proyecciones. No podrán actualizar tablas, pero eso será incluso positivo, para que no estropeen mucho.

- **Facilitan la seguridad**. Cuando se crea una tabla, el ABD puede decir qué usuarios pueden acceder a ella. Al definir las vistas esta seguridad aumenta ya que puede conseguir que un usuario acceda a ciertos atributos de una tabla y no a otros.

5. TECNICAS DE OPTIMIZACION

5.1 INTRODUCCION

En los sistemas de bases de datos relacionales las **preguntas** que se formulan **son asercionales**. El usuario formula sus preguntas expresando qué desea obtener y no cómo debe ser obtenido.

En cambio, en los sistemas en red y jerárquico las preguntas son navegacionales o procedurales. El usuario formula la pregunta expresando cómo deben ser obtenidos los datos.

Ejemplo: *Obtener los nombres de las personas con más de 23 años*

En SQL:
```
SELECT NOMBRE
FROM PERSONA
WHERE EDAD > 23
```

En los modelos jerárquico y en red:
```
Obtener primera PERSONA
mientras existan personas en la B.D.
        si EDAD > 23 entonces obtener NOMBRE
        localizar siguiente PERSONA
```

El subsistema encargado del procesamiento de las preguntas en el Sistema de Gestión de Bases de Datos Relacional debe **seleccionar los planes óptimos** que permitan obtener las respuestas a las preguntas.

En el modelo relacional **es posible realizar la optimización** ya que las preguntas son asercionales y no hacen referencia a características físicas como índices.

Además **la optimización es necesaria** ya que si no se hiciera las preguntas se podrían responder de manera muy ineficiente.

5.2 EJEMPLO DE OPTIMIZACION

Dadas las tablas

S(S#,NOMS,ESTADO,CIUDAD) **(100 suministradores)**

P(P#,NOMP,COLOR,PESO,CIUDAD)

SP(S#,P#,CTD) **(10.000 tuplas donde 50 suministran 'P2')**

y la pregunta *Obtener los nombres de los suministradores de la pieza P2.*

```
SELECT NOMS
FROM S, SP
WHERE S.S# = SP.S# AND P# = 'P2'
```

Existen varias estrategias para responderla:

ESTRATEGIA 1:

1) Realizar el **producto cartesiano** S X P

leer 100 tuplas de S

leer 10.000 tuplas de P

escribir 100 * 10.000 = 1.000.000 tuplas de S X P

2) **Seleccionar** de S X P aquellas tuplas con P# = 'P2'

leer 1.000.000 tuplas de S X P

escribir 50 tuplas de la selección $S_{P\# = 'P2'}(S \times P)$

3) Realizar la **proyección** sobre el atributo NOMS

leer y escribir 50 tuplas

ESTRATEGIA 2:

1) **Seleccionar** de SP las tuplas con P# = 'P2'
   leer 10.000 tuplas
   escribir 50 tuplas de $S_{P\# = 'P2'}(SP)$
2) Hacer el **join** entre la tabla anterior y S
   leer 50 tuplas de $S_{P\# = 'P2'}(SP)$
   leer 100 tuplas de S
   escribir 50 tuplas del join
3) **Proyectar** en esta tabla el atributo NOMS
   leer y escribir 50 tuplas

Como se puede ver esta estrategia es muchísimo mejor que la anterior.
Además no se han tenido en cuenta la existencia de **índices**.
Si hay un índice sobre SP.P# entonces

Seleccionar de SP las tuplas con P# = 'P2'
leer 50 tuplas (¡ y no 10.000 !)

Tampoco se han tenido en cuenta los **tamaños de los resultados intermedios**, ya que si son pequeños se pueden mantener en memoria principal y no en memoria secundaria. Hay que tener en cuenta que las lecturas y escrituras en memoria secundaria son muchísimo más lentas.

5.3 OBJETIVO DEL PROCESO DE OPTIMIZACION

Minimizar el tiempo necesario para responder a una pregunta. Para ello hay que minimizar:

EL COSTO DE ACCESO A MEMORIA SECUNDARIA
EL COSTO DE COMPUTACION
EL COSTO DE ALMACENAMIENTO
EL COSTO DE LAS COMUNICACIONES.

5.4 VENTAJAS DEL PROCESO DE OPTIMIZACION:

El **usuario no se preocupa de formular la pregunta de una manera más eficiente**. Incluso puede desconocer si hay índices definidos para ciertos atributos o no.
El **optimizador puede seleccionar un plan mejor** que el que hubiera podido diseñar el usuario.
- porque tiene acceso a información que igual el usuario desconoce: número de tuplas de cada relación, índices definidos para las tablas, etc...
- porque puede evaluar un número mayor de alternativas.

5.5 ETAPAS DEL PROCESO DE OPTIMIZACION

A) ESTANDARIZACION DE LA PREGUNTA
B) SIMPLIFICACION DE LA PREGUNTA
C) GENERACION DE PLANES DE ACCESO
D) EVALUACION DE COSTES Y SELECCION DEL PLAN OPTIMO

A) ESTANDARIZACION.

Transformación de la pregunta a una representación estándar con el fin de obtener un punto de partida uniforme a partir del cual se puedan realizar los diferentes tipos de optimización.

Esta representación estándar puede ser: *cálculo relacional, álgebra relacional, grafos de preguntas o notaciones de tablas.*

B) SIMPLIFICACION

Restringir cuanto sea posible el volumen de datos al que se va a acceder en la Base de Datos para responder a la pregunta, eliminando objetos innecesarios o condiciones y reduciendo el número de operaciones.

Para ello se utilizan dos tipos de reglas de optimización:

REGLAS DE OPTIMIZACION SINTACTICA

REGLAS DE OPTIMIZACION SEMANTICA

B.1) OPTIMIZACION SINTACTICA

Utiliza propiedades específicas del lenguaje de interrogación utilizado para transformar la pregunta.

- Aplicar reglas de idempotencia:
  A and A = A, A or A = A, A and false = false, etc...
- Propagar constantes:
  (A = 5 and A > C) es igual a (A = 5 and 5 > C)
- Agrupar proyecciones en una sola

  $\pi_A \pi_B (R)$ es igual a $\pi_{A,B} (R)$
- Agrupar selecciones en una sola

  $\sigma_A \sigma_B (R)$ es igual a $\sigma_{A\ y\ B} (R)$
- Realizar antes selecciones que productos

  $\sigma_{R.A} (R \times S)$ es igual a $(\sigma_A (R) \times S)$
- Localizar sub-expresiones comunes para no ser reejecutadas.

B.2) OPTIMIZACION SEMANTICA

Utiliza propiedades específicas de la base de datos, utilizando conocimiento semántico.

Este conocimiento semántico pueden ser restricciones de integridad definidas por el usuario, o conocimiento sobre claves extranjeras, etc...

Ejemplo1:

Sea la tabla EMPLEADO(E#,CATEGORIA,SALARIO)

Y la regla de integridad EMPLEADO.SALARIO < 1000.000

La pregunta S SALARIO > 2000000 (EMPLEADO) siempre devolverá una tabla vacía. NO HACE FALTA BUSCAR EN LA B.D. !!!

Ejemplo 2:

Sea la tabla EMPLEADO(E#,CATEGORIA,SALARIO)

Y la regla de integridad

EMPLEADO.SALARIO > 100.000 0 EMPLEADO.CATEGORIA = 1

Suponiendo que existe un indice para EMPLEADO.CATEGORIA

La pregunta S SALARIO > 140000 (EMPLEADO) se transformará en

S SALARIO > 140000 y CATEGORIA = 1(EMPLEADO) que se responderá de manera más eficiente ya que CATEGORIA tiene un índice.

C) GENERACION DE PLANES DE ACCESO

Existen varias transformaciones sintácticas y semánticas. Para cada una de ellas hay que:

Seleccionar el orden en el que se deben realizar las operaciones.
Asociar un algoritmo a cada operador.

D) EVALUACION DE COSTES Y SELECCION DEL PLAN OPTIMO

Obtener la respuesta con el mínimo costo tiempo / espacio
- Número de accesos a memoria secundaria.
- Tamaño de la B.D. y de las relaciones intermedias.

5. BIBLIOGRAFIA

CERI S., PELAGATTI ***Distributed Databases. Principles and Systems***
Mac Graw-Hill 1984

DATE C.J. ***An Introduction to Database Systems***
Addison-Wesley Vol 1, Ed. 4 1986.

GARDARIN G., VALDURIEZ P. ***Relational Databases and Knowledge Bases.***
Addison-Wesley 1989

INFORMIX-4GL ***Rapid Development Systems.SQL BASED Application Development Language. Reference Manual***
Informix Software Inc. Vol 1, 2 1987

KIM W., LOCHOVSKY F. ***Object-Oriented Concepts, Databases, and Applications***
ACM PRESS 1989

KORTH H., SILBERSCHATZ A. ***Database System Concepts***
Mc. Graw-Hill, 1986

MYLOPOULOS J., BRODIE M. ***Reading in Artificial Intelligence and Databases***
Morgan Kaufmann Publishers Inc. 1989

STONEBRAKER M. ***Readings in Database Systems***
Morgan Kaufmann Publishers Inc. 1988

TEOREY T., FRY J.P ***Design of Database Structures***
Prentice-Hall 1982

ULLMAN J.D. ***Principles of Database and Knowledge-Base Systems***
Computer Science Press Vol 1 U.S.A. 1988

ZDONIK S., MAIER D. ***Reading in Object- Oriented Databases***
Morgan Kaufmann Publishers Inc. 1989

6. Anexo: Informix

INFORMIX - 4GL

CARACTERISTICAS

- Lenguaje de 4ª Generación --> sentencias no procedurales.
- Diseñado para el manejo de BD's.
- Se indica lo que se desea, sin explicar cómo se ha de hacer para conseguirlo

POSIBILIDADES

- Ventanas
- Crear menus
- Tomar datos de pantallas predefinidas
- SQL de INFORMIX (RDSQL) que incluye ANSI SQL
- Uso de pantallas de ayuda
- Hacer informes mediante un lenguaje especial (ACE)
- Realizar preguntas a la BD con el formato QBE
- Definir el uso de las teclas de función y secuencias de control: (F_1, F_2,..., CTRL+A, CTRL+B,…)
- Modificar atributos de pantalla
- Acceso a herramientas de Debug (Aparte)
- Llamar a funciones de librería de 4GL o de C

INFORMIX-4GL

- Lenguaje de 4ª Generación:
  Diseñado para un tipo concreto de aplicaciones: Manejo de BD's.

- Proporciona todas las herramientas para un SGBD relacional:
  1) Lenguaje de interrogación a la BD: RDSQL, extensión del lenguaje SQL de IBM por parte de la empresa creadora de INFORMIX.

     SQL Structured Query Language.

  2) Lenguaje de Programación:
     Hay sentencias básicas para realizar:
     a) Asignación (**LET**, **INITIALIZE**)
     b) Bucles (**WHILE**, **FOR**, **FOREACH**)
     c) Condiciones (**IF**, **CASE**)
     d) Manejo de subrutinas (**FUNCTION**, **CALL**, **RETURN**)

  3) Utilidad para crear formatos de pantallas
     OPEN FORM
     CLOSE FORM

  4) Utilidad para crear Menus:
     Sentencia **MENU**

  5) Utilidad para escribir informes que posee un lenguaje particular (**ACE**)

  6) Gestor de ventanas

ESTRUCTURA DE UN PROGRAMA ESCRITO EN INFORMIX-4GL

MAIN
sentencia
...
END MAIN

donde *sentencia* es cualquier sentencia de INFORMIX-4GL excepto **MAIN**.

Cada Programa INFORMIX-4GL debe tener 1 sentencia MAIN y puede tener definidas cualquier número de funciones o reports.

COMO CREAR UNA BD

CREATE DATABASE *nombre-BD*

- Crea una nueva BD llamada *nombre-BD*
- Físicamente crea un nuevo directorio de nombre *nombre-BD*.dbs que contendrá los ficheros necesarios para implementar el diccionario de datos de esa BD.
- Los ficheros de datos junto con sus ficheros índices (si los tuvieran) referidos a esa BD también se van a colocar en ese subdirectorio.

CREATE TABLE *nombre-tabla*
(*columna tipo* [**NOT NULL**]
[,...])

- sólo son significativos los 10 primeros caracteres del nombre de la tabla.
- NOT NULL indica que esa columna no admite valores nulos al efectuar inserciones o modificaciones de tuplas en esa columna.

TIPOS DE COLUMNAS

TIPO	Nº bytes	Comentario
CHAR(*n*)	*n*	string de longitud *n* ($1 \leq n \leq 32.767$)
SMALLINT	2	enteros en el intervalo [-32.767, +32.767]
INTEGER	4	enteros en el intervalo [-2.147.483.647, +2.147.483.647]
DECIMAL[(*m*[,*n]*)]	1+*m*/2	nº real con *m* dígitos significativos ($m \leq 32$) [precisión] y con *n* dígitos decimales ($n \leq m$) [escala]
SMALLFLOAT	4	equivalente a float de C o a DECIMAL (8)
FLOAT	8	equivalente a double de C o a DECIMAL (16)
MONEY [(*m*[,*n*])]	1+*m*/2	DECIMAL con un $ delante
SERIAL (*n*)	4	nº entero asignado de forma secuencial a partir de *n*
DATE	4	string de la forma definida en DBDATE

NOTAS

- MONEY (m) = DECIMAL (m, 2)
- MONEY = DECIMAL (16, 2)
- SERIAL por defecto *n*=1.Sólo se permite una columna serial en cada tabla.
- DATE se almacena como el nº de días transcurridos a partir del día 31-12-1899

CREAR INDICES

CREATE [**UNIQUE** | **DISTINC** | **CLUSTER**] **INDEX** *nombre-índice*
ON *tabla* (*columna* [**ASC** | **DESC**] [,...])

•**UNIQUE = DISTINCT**: Evita la aparición de valores duplicados en aquellas columnas a las que atañe el índice.

•**CLUSTER**: Hace que la tabla sobre la que se define el índice se ordene físicamente de la misma manera que lo hace el índice.
NOTA: No se pueden crear 2 índices CLUSTER sobre una misma tabla.

•**ASC**: Indica que el orden de almacenamiento de valores en el fichero índice es ascendente.

•**DESC**: Indica que el orden de almacenamiento de valores en el fichero índice es descendente.

IMPLEMENTACION DEL CONCEPTO DE CLAVE DE BD RELACIONALES EN INFORMIX

- Definiendo los atributos que forman parte de la clave como NOT NULL.

- Definiendo un índice único sobre todos los atributos que forman parte de la clave.

Ejemplo Profesor - Asignaturas

```
MAIN

    CREATE DATABASE ejemplo

    CREATE TABLE profesor (P char(6) NOT NULL,
                           NOM-PROF char (20),
                           EDAD smallint).

    CREATE TABLE asignatura   (A char (5) NOT NULL,
                               NOM-ASIG char (50),
                               CUR-ASIG smallint).

    CREATE UNIQUE INDEX xprofesor ON profesor (P ASC)
    CREATE DISTINCT INDEX xasignatura ON asignatura (A DESC)

END MAIN
```

MANIPULACION DE LA BD

1. Para trabajar con una BD antes hay que seleccionarla como BD actual. Esto se consigue mediante el comando:

   DATABASE *nombre-BD*

NOTA: Si un programa no referencia a una BD en particular, la sentencia es innecesaria.

2. VARIABLES

Informix permite definir variables simples de dos maneras:

a) indicando directamente el tipo de una variable

```
DEFINE var_a INTEGER
```

b) indicando que el tipo de la variable es del mismo tipo que el de un atributo de una tabla concreta.

```
DEFINE var_pedido LIKE PEDIDO.PDNUM
```

La definición de variables estructuradas es de forma similar:

a_1)
```
DEFINE var_pedido RECORD
    PDNUM integer,
    PDFECHA date,
    PDFECHALIM date,
    PDFECHAENT date
END RECORD
```

a_2)
```
DEFINE var_pedido RECORD
    PDNUM integer,
    PDFECHA LIKE PD.PDFECHA
    PDFECHALIM LIKE PD.PDFECHALIM,
    PDFECHAENT date
END RECORD
```

b)
```
DEFINE var_pedido RECORD LIKE pd.*
```

NOTA:

Al usar **LIKE** se debe haber usado sentencia **DATABASE** previamente.

Hay veces que el nombre de un atributo de una tabla es igual al de una variable.

Informix toma por defecto la referencia a variables. Para evitar esta referencia, se añade delante del nombre de la columna una @.

3) ASIGNACION

LET *variable = expresión*.

o

LET *variable-record*.* = *variable-record*.*.

A) OPERACIONES NUMERICAS

Operador	Significado
**	exponenciación
*	producto
/	división
mod	modulo
+	suma
-	resta

B) OPERACIONES CONSTRINGS

Operador	Significado
,	concatenación
[m, n]	substring desde la posición m a la n
USING "formato"	formatea el string
CLIPPED	quita los espacios al final de un string

c) OPERACIONES CON EXPRESIONES BOOLEANAS

Operador	Significado
=	igual
!= ó <>	distinto
>	mayor
>=	mayor o igual
<	menor
<=	menor o igual

AND	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

OR	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

NOT	
T	F
F	T
?	?

c) OTRAS OPERACIONES

string-expr [**NOT**] **LIKE** string-expr
Formato "X#9"

string-expr [**NOT**] **MATCHES** string-expr

* = 0 ó más caracteres [] cualquier carácter de los que aparecen entre corchetes. los rangos : a-ZA-Z

? = cualquier carácter [^] cualquier carácter excepto los que aparecen entre corchetes.

expr [**NOT**] **BETWEEN** $expr_1$ **AND** $expr_2$

expr [**NOT**] **IN** ($item_1$,$item_2$,...)

expr [**NOT**] **IN** (**SELECT**...)

expr rel-op {**ALL** | **ANY** | **SOME**} (**SELECT**...)

EXISTS (**SELECT**...)'

INTERACCION CON EL USUARIO

A) ENTRADA DE DATOS

- Hay 2 formas de obtener datos por parte del usuario,

  a) Utilizando una pantalla destinada a tal fin

     FORMS

  b) Mediante una sentencia de entrada 4GL:

     PROMPT *lista-salida* **FOR** [**CHAR**] *variable*.

     Se escribe en la pantalla el string generado por los valores de lista-salida y toma un valor que le programa el usuario en la variable.
     CHAR: Espera un carácter alfanumérico.

B) SALIDA DE DATOS

También existen 2 maneras de proporcionar datos a los usuarios.

  a) Utilizando la facilidad para crear informes:

     REPORT...

  b) Efectuando la salida a través de una sentencia 4GL:

     DISPLAY *lista-salida* [**AT** *fila*, *columna*]

     La *lista-salida* está compuesta por una serie de expresiones.
     Se puede utilizar el comando **USING** teniendo los siguientes efectos.

OPERACIONES BASICAS SOBRE LA BASE DE DATOS

1) INSERTAR FILAS

INSERT INTO *tabla* [(*lista-columnas*)] **VALUES** (*lista-valores*)

Con esta instrucción se introducen los nuevos valores para una fila de una tabla en la BD.
Si se omite la enumeración de columnas, hay que proporcionar un valor para cada columna de la tabla en cuestión.
Si no se omite tal enumeración se introduce el valor NULL para aquellas columnas en las que no se introduce valor.

- Si se desea introducir valores de tipo SERIAL, se coloca en su lugar el valor 0.

2) OBTENER FILAS

SELECT {*lista-columnas* | *****}
INTO *lista-variables*
FROM *lista-tablas*
WHERE *condiciones*

lista-columnas: lista-de-columnas y/o expresiones separadas por comas.
lista-variables: variables donde se va a almacenar cada fila de la tabla resultante.
lista-tablas: lista de tablas separadas por comas.
condiciones: condición(es) que deben cumplir las filas seleccionadas.

3) MODIFICAR FILAS

UPDATE *tabla* **SET** {*columna* = *expr* [,...]
| {(*lista-columnas*) | *[tabla*.] *} = {(*lista-exps*) | *registro*.*}}
[**WHERE** *condiciones*]

No se puede modificar ningún valor de tipo SERIAL.

4) BORRAR FILAS

DELETE FROM *Tabla* [**WHERE** *condiciones*]

Borra 1 o más filas.
Para borrar sólo parte de una fila hay que usar UPDATE.

FUNCIONES AGREGADAS QUE SE PUEDEN USAR EN LA INSTRUCCION SELECT

CONT (*) : Nº de filas que satisfacen la condición WHERE si la hay
SUM (*columna***)** : Suma los valores que están en la columna indicada y se obtienen
AVG (*columna***)** : Calcula la media de los valores que están en la columna indicada y se obtienen
MAX (*columna***)** : Calcula el máximo de los valores que se obtienen de la columna indicada
MIN (*columna***)** : Calcula el mínimo de los valores que se obtienen de la columna indicada

TRABAJAR CON MAS DE UNA FILA DE LA BD

Hace falta definirse un **CURSOR**.

CURSOR: Puntero a una fila obtenida mediante una sentencia SELECT.

Hay dos tipos:

A) con SCROLL: se pueden manejar las filas en el orden en el que desee el programador. No es aconsejable definirlos a menos que se necesite acceder a los datos de forma aleatoria, ya que este sistema necesita mucha más información a procesar.

B) sin SCROLL: se recorre secuencialmente las filas obtenidas por una sentencia SELECT.

DECLARE *cursor* [**SCROLL**] **CURSOR FOR**
{*sentencia select* [**FOR UPDATE** [**OF** *lista-colmnas*]]
| *sentencia-insert*}

Para mover el cursor de una fila a otra fila seleccionada por una misma instrucción SELECT utilizaremos la sentencia siguiente :

FETCH o **FETCH NEXT** (sólo con un cursor sin scroll)
FETCH [**NEXT** |
PREVIOUS | **PRIOR**
FIRST |
LAST |
CURRENT |
RELATIVE *m* |
ABSOLUTE *n* } *cursor* **INTO** *Lista-var* (en cursores con scroll)

INSTRUCCIONES DE CONTROL DE FLUJO DE PROGRAMA

1) **CALL** *función* ([*lista-argumentos*]) [**RETURNING** *lista-var*]
   Llama a una función que devuelve 0 o más valores.
   Sólo se puede usar una función en una expresión si devuelve un único valor.

2) **FOR** *var-entera* = *exp-entera* **TO** *exp-entera* [**STEP** *exp-entera*]
   sentencia
   ...
   [**CONTINUE FOR**]
   ...
   [**EXIT FOR**]
   ...
   END FOR

3) **FOREACH** *cursor* [**INTO** *lista-var*]
   sentencia
   ...
   [**CONTINUE FOREACH**]
   ...
   [**EXIT FOREACH**]
   ...
   END FOREACH

   Se ejecutan las sentencias para cada una de las filas afectadas por una query a la BD.

4) **WHILE** *exp-boleana*
   sentencia
   ...
   [**CONTINUE WHILE**]
   ...
   [**EXIT WHILE**]
   ...
   END WHILE

5) **IF** *exp-bool*
   THEN
      sentencia
      ...
   ELSE
      sentencia
      ...
   END IF

6) **CASE** [(*expr*)]
   WHEN [*expr* | *exp-booleana*}
      sentencia
      ...
      [**EXIT CASE**]
      ...
   WHEN [*expr* | *exp-booleana*}
      sentencia
      ...
      [**EXIT CASE**]
      ...
   ...
   [**OTHERWISE**
      sentencia
      ...
      [**EXIT CASE**]
      ...
   END CASE

   expr: expresión que se evalúa a un INTERGER, SMALLINT, DECIMAL, CHAR (1).

7) **SLEEP** *exp-entera*.
   Se suspende la ejecución del programa durante exp-entera segundos.

8) **RUN** "*nombre-programa*"
   Ejecuta una instrucción a nivel de S.O. y devuelve el control a informix.

FUNCIONES

FUNCTION *función* ([*lista-argumentos*])
sentencia
...
[**RETURN** [*lista-expr*]]
...
END FUNCTION

NOTA: Los argumentos de una función se deben definir como variables en las primeras instrucciones del código de la función (sentencia **DEFINE**).

DEFINICION DE MENUS

MENU "*nombre-menu*"
COMMAND {**KEY** (*lista-teclas*) |
[**KEY** (*lista-teclas*)] "*opción-menú*" ["*lista-ayuda*"] [**HELP** *nº-help*]}
sentencia
...
[**CONTINUE MENU**]
...
[**EXIT MENU**]
...
[**NEXT OPTION** "*opción-menú*"]
...
...
END MENU

Crea un menú del tipo LOTUS 123™.en el que se cambia de opción mediante las teclas de los cursores o el espacio y se selecciona una opción apretando la tecla de ENTER o RETURN.

CREACION DE PANTALLAS

- ficheros *.per

Hay 5 secciones en una pantalla:

1) Database

DATABASE *nombre-BD*

NOTA: se puede crear una pantalla sin que esté relacionada con un BD. para ello poner de nombre-BD "formonly".

2) Screen

SCREEN
{
No más de 20 líneas. sino trunca. Cualquier carácter.
Los campos de entrada son de la forma: *[etiqueta]*
}
[**END**]

3) Tables

TABLES
tabla
...
[**END**]

No es necesaria esta sección si se especificó *formonly* en la sección Database.

4) Atributes

ATRIBUTES
etiqueta = [*tabla*.] *columna* [, *lista-atributos*];
...
END

En pantallas del tipo *formonly*:

ATRIBUTES
etiqueta = **FORMONLY**.*nombre-campo*
[**TYPE** [*tipo-datos* | **LIKE** *tabla.columna*]] [**NOT NULL**] [*, lista-atributos*];
...
END

Cuando hay un campo que pertenece a varias tablas:
etiqueta = tabla1.columna = tabla2.columna [, *lista-atributos*];

ATRIBUTOS POSIBLES

Son los valores que pueden existir en *lista-atributos*:

- AUTO NEXT
- COMMENTS = "mensaje";
- DEFAULT = valor por defecto
- DISPLAY LIKE = x.y enseñar el campo según el tipo de x.y
- DOWNSHIFT = a minúsculas
- FORMAT = "formato": ver formatos de display
- INCLUDE = (lista valores): valores válidos para ese campo
- NOENTRY; en este campo no se puede realizar entrada de datos. es solo de salida
- PICTURE = "____" A=letras; # = digitos; X cualquier carácter o letra: patrón de como se deben meter los datos
- REQUIRED: necesario darle un valor.
- REVERSE = saca en reverse
- UPSHIPT
- VALIDATE LIKE = comprueba con el tipo de x.y
- VERIFY = pide confirmación

5) Instructions

INSTRUCTIONS

Instrucción

...

[**END**]

INFORMES

1. ESQUEMA GENERAL DE LLAMADA A UN INFORME

START REPORT *report* [**TO** {**PRINTER** | **PIPE** *programa* | *fichero*}
Comienzo de bucle de cualquier tipo
...
OUTPUT TO REPORT *report* (*valores a imprimir*)
...
fin-bucle
FINISH REPORT *report1*

2. ESTRUCTURA DE UN INFORME

REPORT *report* (*lista-argumentos*)
[**DEFINE** ...]
[**OUTPUT** ...]
[**ORDER BY** ...]
FORMAT ...
END REPORT

2.1 DEFINE

Se definen todos los argumentos que se pasan como argumento así como otras variables que se necesiten.

2.2 OUTPUT

OUTPUT
[**REPORT TO** {"*fichero*" | **PIPE** *programa* | **PRINTER**}]
[**LEFT MARGIN** *entero*]
[**RIGHT MARGIN** *entero*]
[**TOP MARGIN** *entero*]
[**BOTTOM MARGIN** *entero*]
[**PAGE LENGTH** *entero*]

2.3 ORDER.BY

ORDER [**EXTERNAL**] **BY** *lista-argumento*

2.4 FORMAT

FORMAT
EVERY ROW
END REPORT

FORMAT
[**PAGE HEADER**]
[**PAGE TRAILER**]
[**FIRST PAGE HEADER**]
[**ON EVERY ROW**]
[**BEFORE GROUP OF**
...]
[**AFTER GROUP OF**
..]
END REPORT

Las sentencias válidas son:
•NEED *expresión-entera* LINES: si no quedan *expresión-entera* líneas en la página actual, se pasa a la siguiente página.

•**PAUSE** ["*string*"]
muestra el string y espera hasta que se pulse <RETURN>

• **PRINT** [*lista-exps*] [;]
escribe las expresiones indicadas
; suprime el carácter de fin de línea

•**PRINT FILE** "*fichero*"
incluye el contenido de un fichero en el informe

•**SKIP** {*entero* **LINE**[**S**] | **TO TOP OF PAGE**}
salta *entero* líneas o pasa de página