TEMA 1

INTRODUCCIÓN A LAS BASES DE DATOS

1.1.− De los sistemas tradicionales de ficheros a las bases de datos.

1.2.− Definición de base de datos.

1.3.− Elementos de una base de datos.

1.4.− Dato operativo.

1.5.− Ventajas de las bases de datos frente a los ficheros clásicos.

1.6.− Independencia de datos.

1.7.− Tipos de bases de datos.

1.1.− De los sistemas tradicionales de ficheros a las bases de datos.

Una de las primeras empresas en desarrollar un lenguaje de programación orientado a bases de datos fue CODASYL que sobre los años 60 desarrolló el COBOL. Los principales motivos para el paso de los sistemas tradicionales al uso de las bases de datos fueron los siguientes:

- Rapidez al acceso de la información.
- Facilidad de trabajo, etc.

1.2.− Definición de base de datos.

Una base de datos es un sistema de captación y mantenimiento de registros de forma computerizada. En este sistema se van a poder realizar las siguientes operaciones: Inserción, borrado y modificación de un dato. También se puede hacer modificaciones, borrados e inserciones de información en la estructura de la base de datos.

1.3.− Elementos de una base de datos.

En una base de datos se tienen 4 elementos:

- **Datos.−** Deben cumplir 2 condiciones:
  - Deben ser **integrados**, es decir, recogen toda la información intentando que la redundancia sea mínima.
  - Deben ser **compartidos** a nivel de aplicación.
- **Hardware.−** Es el soporte físico que permite almacenar la información de la base de datos. Cuando la base de datos está formada por varios sistemas se llama base de datos distribuida. El manejo de las bases de datos distribuidas se complica ya que se va a necesitar comunicación entre los sistemas.
- **Software.−** Permite trabajar y gestionar la base de datos de la forma más eficiente. El SGBD (Sistema gestor de bases de datos) es el encargado de gestionar la base de datos, y debe ofrecer facilidades para la inserción, borrado y modificación de la información. Por lo tanto, todas las operaciones que se realicen

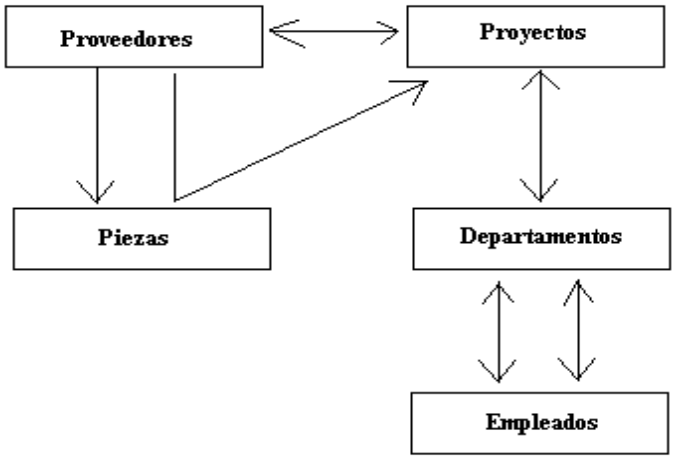
sobre las mismas han de pasar por el SGBD.

- **Usuarios.−** Hay tres tipos de usuarios.

    - **Programadores de aplicaciones.−** Se encargan de diseñar y programar las aplicaciones necesarias para la utilización de la bases de datos, realizando las peticiones pertinentes al SGBD.
    - **Usuario final.−** Es la persona que se dedica a trabajar sobre los datos almacenados en la base de datos. Hay usuarios finales avanzados que por medio del lenguaje de programación SQL pueden acceder a los datos.
    - **Administrador de base de datos.−** Es el usuario más importante de los tres, ya que es el que se encarga de diseñar y modificar la estructura de la base de datos.

1.4.− Dato operativo*.

Es toda la información que necesita una empresa para su funcionamiento. Son las entidades con sus atributos más la conexión que hay entre ellas. La integración de todo lo anterior es el **diseño lógico de la base de datos.**



Ejemplo:

1.5.− Ventajas de las bases de datos frente a los ficheros clásicos.

Las principales ventajas de las bases de datos sobre los ficheros clásicos son las siguientes:

- Compacidad.
- Rapidez de acceso a la información.
- Facilidad de trabajo.
- Actualización.
- Control centralizado, ostentado por el administrador de la base de datos.
- Reducción de redundancias.
- Eliminar inconsistencias.
- Los datos pueden compartirse.
- Los estándares se mantienen.
- Mayor seguridad.
- Mayor facilidad en el chequeo de errores.
- Equilibrado de requerimientos opuestos.

1.6.− Independencia de datos*.

La independencia de los datos es la impunidad de las aplicaciones existentes a cambios en la forma de almacenamiento y acceso de la base de datos. Se dice que una aplicación es dependiente de los datos si es imposible alterar la estructura de almacenamiento o la técnica de acceso sin afectar a la aplicación. En un sistema de bases de datos no es recomendable tener aplicaciones dependientes de los datos, por dos razones:

- Cada aplicación puede requerir una vista diferente de los mismos datos. Una aplicación puede requerir los datos en formato decimal y otra puede requerirlos en binario.
- El administrador de la base de datos ha de tener libertad para modificar la estructura de almacenamiento y las técnicas de acceso para adaptarlos al cambio de los requerimientos sin tener que modificar las aplicaciones ya existentes. Algunas de las modificaciones que podrían ser necesarias sería la adición de datos de otro tipo a la base de datos, la aparición de nuevas normas (€), o un cambio de prioridades.

Se va a buscar la independencia de datos a tres niveles:

- **Nivel de campo almacenado.−** Mínima cantidad de información que se almacena reconocible con un nombre.
- **Nivel de registro almacenado.−** Es un conjunto de campos almacenados relacionados entre sí, que cuenta con su propio nombre. Una ocurrencia de registro almacenado es el valor de todos los campos de un registro (Ej: Color = Azul, Talla = 10, Artículo = Tornillo)
- **Nivel de fichero almacenado.−** Es el conjunto de todas las ocurrencias de un tipo de registro almacenado reconocible con un nombre..

Un registro lógico es el registro que ve el usuario, y un registro físico es un registro tal y como se almacena en la base de datos.

El campo lógico puede ser igual o no al campo almacenado. Por tanto se puede buscar la independencia de datos basándose en este concepto, denominado **materialización**, que puede ser de dos formas:

- **Directa.−** El campo lógico es igual al campo almacenado.
- **Virtual.−** El campo lógico se corresponde con parte o más del campo almacenado.

Ejemplo:

A nivel de fichero almacenado debe preverse el medio físico en el que se va almacenar porque una base de datos es dinámica.

Aspectos de una base de datos susceptibles de modificación.− Hay que tener mucho cuidado a la hora de considerar las siguientes cuestiones

- Representación de datos numéricos (binario, decimal...)
- Representación de caracteres (ASCII, EBCDIC...)
- Unidades para datos numéricos (Pta., €, ¥, £, $, DM)
- Codificación de los datos.

La independencia es de los datos es fundamental porque las bases de datos son dinámicas.

1.7.− Tipos de bases de datos.

Para la implementación de la base de datos nos vamos a basar en dos estructuras de datos:

- La **tabla** o **array bidimensional,** en el que se basa el modelo relacional.
- El **grafo** en el que se basan el modelo jerárquico (árbol) y el modelo en red (grafo cerrado).

TEMA 2

ARQUITECTURA DE UN SISTEMA DE BASES DE DATOS

2.1.− Niveles generales del sistema.

2.1.1.− Nivel Externo.

2.1.2.− Nivel Conceptual.

2.1.3.− Nivel Interno.

2.1.4.− Correspondencias

2.2.− El Administrador de la Base de Datos.

2.1.− Niveles generales del sistema.

El grupo ANSI / SPARC creó un standard para las arquitecturas de las bases de datos. En esta estandarización se define una arquitectura para los sistemas de bases de datos divida en tres niveles:

Vista 1 Vista 2 ..... Vista n **Nivel externo** (Vistas individuales de los usuarios).

Nivel conceptual (Vista comunitaria de los usuarios).

Nivel interno. (Vista de la forma de almacenamiento).

En bases de datos pequeñas el nivel conceptual y el nivel interno suelen estar unidos.

Ejemplo: En una empresa están trabajando dos programadores, uno en sobre el lenguaje de programación PL/I y el otro en COBOL. En el nivel externo los usuarios van a trabajar sobre la base de datos. Sobre el nivel conceptual e interno únicamente trabaja el administrador de la base de datos.

<table>
<tr><td>Externo (PL/I)

DCL 1 EMP . 2 EMP# CHAR(6)

3 SAL FIXED BIN(31)</td><td>Externo (COBOL)

01 EMPC

. 02 NUMEMP PIC X(6)

. 02 NUMDEP PIC X(4)</td><td>Vistas Externas</td></tr>
<tr><td colspan="2">Conceptual

Empleado

numero _ empleado carácter(6)

numero _ departamento carácter(4)

salario numérico(5)</td><td>V. Conceptuales</td></tr>
<tr><td colspan="2">Interno

EMP_ALMAC longitud = 18</td><td>Vista Interna</td></tr>
</table>

Prefijo tipo = byte(6), desplazamiento = 0 EMP# tipo = byte(4), desplazamiento = 6, Índice =EMP DPTO# tipo = byte(4), desplazamiento = 12 Paga tipo = palabra, desplazamiento = 16	

2.1.1.− Nivel Externo.

El nivel externo está formado por las vistas individuales de cada uno de los usuarios, es decir, cómo percibe el usuario la base de datos. Éste es el nivel en el cual trabaja el usuario individual. Los usuarios pueden ser o bien programadores de aplicaciones o usuarios finales, donde cada usuario dispone de un lenguaje. En el caso de un programador de aplicaciones dicho lenguaje puede ser un lenguaje de alto nivel para manejar la base de datos y si la base de datos no lo permite, se utilizará un lenguaje propio del sistema de bases de datos (como NOMAD ó FOCUS). En el caso de ser un usuario final será o bien un lenguaje de consulta, (como el SQL) o algún lenguaje de aplicación basado en menús.

Los lenguajes de programación deben incluir un **sublenguaje de datos (DSL)**, es decir, un subconjunto del lenguaje total que se ocupe de manera específica de los objetos y operaciones de la base de datos. Se dice que el DSL está embebido dentro del lenguaje anfitrión correspondiente. En principio, cualquier DSL es en realidad una combinación de por lo menos dos lenguajes subordinados:

- **DDL. (Lenguaje de definición de datos).−** Con el que es posible definir o declarar los objetos de la base de datos.
- **DML. (Lenguaje de manipulación de datos).−** Con el que es posible manipular o procesar dichos objetos..

Cuando el DSL es indistinguible del lenguaje anfitrión se dice que está fuertemente acoplado y si se pueden separar con nitidez se dice que están débilmente acoplados. Son preferibles los lenguajes fuertemente acoplados.

A la vista individual de cada usuario se denomina **vista externa**. La vista externa está formada por el conjunto de ocurrencias de los registros externos. Toda vista externa se define mediante un **esquema externo** que es la definición de los tipos de registros externos en esa vista externa. El esquema externo se escribe mediante el DDL.

2.1.2.− Nivel Conceptual.

Está formado por la vista comunitaria de los usuarios, es decir, que al unir todas las vistas externas obtenemos toda la información. Es la representación de toda la información contenida en la base de datos.

La **vista conceptual** se compone de las ocurrencias de los diferentes tipos de registro conceptual. Esta vista se define por medio del esquema conceptual, que está formado por la definición de cada uno de los tipos de registro conceptual.

El **esquema conceptual** se define mediante el **DDL conceptual**. El DDL externo debe ser distinto al DDL conceptual. Si se logra hacer totalmente independiente el DDL conceptual a los datos, el nivel externo también lo será.

En el nivel conceptual no deben aparecer consideraciones sobre el almacenamiento (para tratar de conseguir la independencia de los datos). Sin embargo, las definiciones en el esquema conceptual deben incluir

características tales como las verificaciones de seguridad y de integridad.

2.1.3.− Nivel Interno.

Está formado por las vistas del almacenamiento (la forma en que se almacenan los datos). Es una representación de bajo nivel de toda la base de datos y se compone de las ocurrencias de los diferentes tipos de registro interno. Está a un paso del nivel físico, ya que no gestiona a nivel de páginas o bloques.

La **vista interna** se define mediante el **esquema interno**, el cual no sólo define los diversos tipos de registros almacenados, sino que también especifica que índices hay, la representación de los campos almacenados, la secuencia física de los registros almacenados, etc. El esquema interno se define mediante el DDL interno

2.1.4.− Correspondencias.

Se distinguen dos tipos de correspondencias que se encargan de conectar los tres niveles de una base de datos:

- **Conceptual − Interna.−** Permite enlazar la vista conceptual con la base de datos almacenada (vista interna) y permite representar los registros y campos conceptuales en el nivel interno. Si se modifica la definición de la estructura de almacenamiento esta correspondencia deberá modificarse también de para que no varíe el esquema conceptual. Este tipo de correspondencia permite llevar a cabo el concepto de independencia de los datos. En el caso de modificación del nivel interno es esta correspondencia la que debe verse afectada de manera que los cambios no involucren al nivel conceptual.

- **Externa − Conceptual.−** Enlaza las vistas externas con la vista conceptual y permite relacionar los nombres de los registros y campos externos con los nombres de los registros y campos conceptuales. También se encarga de mantener la correspondencia en el caso de que varios registros o campos conceptuales se correspondan con uno o más registros o campos externos (materialización).

2.2.− El Administrador de la Base de Datos.

El administrador de la base de datos tiene seis misiones fundamentales:

- **Describir el contenido de la información** en la base de datos, es decir, diseñar el esquema conceptual. Para esto, primero se mira la información que la empresa necesita para su funcionamiento y luego se hace el diseño lógico de la base de datos.
- **Decidir sobre la estructura de almacenamiento**, es decir, definir el esquema interno por medio del DDL interno. Va a tener que diseñar la parte física de la base de datos (como se representarán los campos, como se organizarán los registros, la indexación, las formas de acceso, la seguridad física...). También va a tener que diseñar la correspondencia conceptual − interna.
- Se encarga de la **conexión con los usuarios.** Capta la visión externa de cada usuario y luego desarrolla el esquema externo al que está asociado. Además va a ser el encargado de diseñar la correspondencia externo − conceptual. También deberá crear un entorno amigable para el usuario. Al programador de aplicaciones le va a proporcionar ayuda para la implementación de la vista externa (DDL externo), aunque algunos sistemas permiten que el programador diseñe e implemente su propia correspondencia. Le dará al programador un lenguaje para la explotación del esquema externo (DML).
- Tratar los problemas de seguridad e integridad.
- Definir la estrategia de recuperación de fallos.
- Ocuparse de los problemas de rendimiento (afinamiento).

Pregunta de examen:

¿A qué equivalen el registro lógico y el registro almacenado a nivel de arquitectura?

El registro lógico se corresponde con el registro externo porque es lo que percibe el usuario, mientras que el registro almacenado se corresponde con el registro conceptual.

TEMA 3

EL MODELO ENTIDAD − RELACIÓN

3.1.− Conceptos básicos.

3.1.1 Conjuntos de entidades

3.1.2 Conjuntos de relaciones.

3.2.− Cuestiones de diseño.

3.3.− Ligaduras de correspondencias.

3.4.− Claves

3.5.− Diagrama Entidad − Relación.

3.6.− Conjuntos de entidades débiles.

3.7.− Características del modelo Entidad − Relación extendido.

El modelo Entidad − Relación está basado en una descripción del mundo real formado por dos tipos de objetos básicos: las **entidades** y las **relaciones** entre objetos. Este modelo surge por la necesidad de plasmar la información que debe ir en la base de datos y para plasmar la especificación de información que necesita una empresa.

El modelo Entidad − Relación pertenece al grupo de modelado semántico que pretende representar el significado de los datos. El modelo Entidad − Relación sirve para crear el **esquema conceptual**.

3.1 Conceptos básicos

Hay tres conceptos básicos:

- Conjuntos de entidades.
- Atributos.
- Conjuntos de relaciones.

3.1.1 Conjuntos de entidades

Una entidad es un objeto en el mundo real que es distinguible de todos los demás y que posee un conjunto de propiedades (**atributos**). Va a haber un subconjunto de propiedades cuyos valores van a determinar de una manera unívoca a una entidad, como por ejemplo un campo DNI.

Un **conjunto de entidades** es la totalidad de entidades del mismo tipo que compartía las mismas propiedades ó atributos.

Las entidades individuales que constituyen un conjunto se llaman **extensiones**. Los conjuntos de entidades no tienen por qué ser disjuntos.

Una entidad se representa mediante un conjunto de atributos, que permiten describir propiedades de cada miembro del conjunto de entidades. Cada atributo tiene un conjunto de valores permitido. Cada conjunto se llama **dominio.**

Una base de datos es una colección de conjuntos de entidades.

Ejemplo: Conjunto de entidades Cliente y Prestamo_Bancario

Formalmente un atributo de un conjunto de entidades es una función que asigna al conjunto de entidades un dominio. Por tanto, una entidad se puede describir como un conjunto de pares atributo − valor, uno por cada atributo.

Ejemplo:

CLIENTE: {(Nombre, Santos), (DNI, 3569852), (Calle, C/Amargura), (Ciudad, A Coruña)

En el modelo Entidad − Relación existen distintos modelos de atributos:

- Simples y compuestos.
- Univalorados y multivalorados.
- Nulos.
- Derivados.

- **Atributos simples y compuestos.−** Los **atributos simples** son aquellos que no tienen capacidad de ser descompuestos, mientras que los **atributos compuestos** son aquellos que permiten descomponerse en otros atributos conformando lo que se denominan **jerarquías**. Un ejemplo de atributo simple podría ser la edad, mientras que uno compuesto podría ser la dirección de un cliente:

Es conveniente usar atributos compuestos porque facilitan el diseño haciéndolo más claro y sencilo.

- **Atributos univalorados y multivalorados.−** Los **atributos univalorados** son aquellos que sólo pueden tomar un valor (NombreCliente), mientras que los **atributos multivalorados** pueden tomar más de un valor para un mismo atributo (DirecCliente, NumTlf). Hay que marcar siempre un límite inferior y un límite superior.

- **Atributos nulos.−** Un atributo es nulo cuando para una determinada entidad ese atributo no tiene valor. Se entiende *nulo* como *desconocido*. Un atributo nulo es **no conocido** cuando no se ha introducido un valor en el campo correspondiente, y está **perdido** cuando se produce un error en la base de datos y el valor se pierde o se corrompe.

- **Atributos derivados.−** El valor para este atributo se puede derivar de los valores de otros atributos ó entidades. Por ejemplo, si existe un campo fecha_nacimiento, la edad sería un atributo derivado

3.1.2 Conjuntos de relaciones

Relación.− Una relación es una asociación entre diferentes entidades. Un **conjunto de relaciones** es una relación matemática con dos o más conjuntos de entidades. Si ***E1, ..., En*** son conjuntos de entidades, entonces un conjunto de relaciones $R \quad \{e_1,...,e_n / e_1 \quad E_1,...,e_n \quad E_n\}$
donde (***e1,..., en***) es una relación. La asociación entre conjuntos de entidades se denomina **participación.**

La función que desempeña una entidad en una relación se denomina **papel de la entidad**, y es necesario especificarlo cuando el significado en una relación necesita aclaración.

Cuando los conjuntos de entidades que se relacionan son iguales, se denominan **conjuntos de entidades recursivos.** Una relación puede tener asociadas una serie de atributos descriptivos (un nombre).

Al número de conjuntos de entidades que participan en un conjunto de relaciones se le denomina grado del conjunto de relaciones.

3.2 Cuestiones de diseño.

- **Uso de conjuntos de entidades ó atributos.−** La utilización de conjuntos de entidades o atributos depende del desarrollo del mundo real que se esté modelando y de la semántica asociada al atributo considerado.
- **Uso de conjuntos de entidades o conjuntos de relaciones.−** Se realizará de la siguiente manera: se debe designar un conjunto de relaciones para describir una acción entre entidades.

3.3 Ligaduras de correspondencias.

Correspondencia de cardinalidad.− Es aquella que expresa el número de entidades a las que otra entidad puede estar asociada mediante un conjunto de relaciones, permitiendo describir conjuntos de relaciones binarias.

Dado un conjunto de relaciones binarias ***R*** entre los conjuntos de entidades **A** y **B**, la correspondencia de cardinalidad puede ser:

- **De uno a uno**.− Se da cuando una entidad en **A** se asocia con una entidad en **B** o cuando una entidad en **B** se asocia con una entidad en **A**.
- **De uno a muchos**.− Se da cuando una entidad en **A** se asocia con cualquier número de entidades en **B** y una entidad en **B** sólo se puede asociar con una entidad en **A**.
- **De muchos a uno**.− Es el caso inverso al anterior, siendo una entidad en **B** la que se puede asociar con cualquier número de entidades en **A** y una entidad en **A** la que sólo puede asociarse con una entidad en **B.**
- **De muchos a muchos.−** Cualquier entidad de cualquiera de los dos conjuntos puede asociarse con cualquier número de entidades del otro conjunto.

La cardinalidad de una relación puede afectar a la situación de los atributos de la relación. Cuando la cardinalidad es **uno a uno**, el atributo descriptivo puede ir en cualquiera de los dos conjuntos. Si es de **uno a muchos** o de **muchos a uno**, deberá ir en el conjunto de muchos. Y si es de **muchos a muchos**, deberá permanecer en el conjunto de relaciones.

Dependencias de existencia.− Si la existencia de la entidad ***X*** depende de la existencia de la entidad ***Y***, se dice que ***X*** tiene **dependencia de existencia** de ***Y***. La entidad ***Y*** es la entidad **dominante** y ***X*** es la entidad **dominada.**

La participación de un conjunto de entidades ***E*** en un conjunto de relaciones ***R*** es **total** si cada entidad en ***E*** participa en al menos una relación de ***R.*** Si sólo algunas entidades en ***E*** participan en relaciones en ***R***, la participación se denomina **parcial.**

3.4 Claves

Las claves tienen que ser definidas respecto al conjunto de entidades y con respecto al conjunto de relaciones.

- Con respecto al conjunto de entidades se definen:

Superclave.− Conjunto de atributos que permiten identificar de manera unívoca a cada una de las entidades del conjunto de entidades. Un ejemplo de esto sería un campo DNI.

Clave Primaria.− Es la clave candidata que es elegida por el diseñador de la base de datos para identificar a cada una de las entidades de un conjunto de entidades.

- Con respecto al conjunto de relaciones se utiliza para distinguir entre las diferentes relaciones de un conjunto y se define:

Sea **R** un conjunto de relaciones que implica a los conjuntos de entidades **E1, ..., En.** Sea clave primaria (**Ei**) el conjunto de atributos que forma la clave primaria para el conjunto de entidades **Ei:**

Clave Primaria (R) = Clave Primaria (E1) "..." Clave Primaria (En)

La clave primaria del conjunto de relaciones es la unión de las claves primarias de los conjuntos de entidades, sin tener en cuenta los posibles atributos descriptivos del conjunto de relaciones **R**.

Esta clave primaria se verá afectada por la cardinalidad del conjunto de relaciones de la siguiente manera:

- **Muchos a muchos**.− Está formada por la clave primaria de cada uno de los conjuntos de entidades de **R**. (K1 " K2).
- **Uno a muchos** (y viceversa).− Está formada por la clave primaria del conjunto de entidades que establece la cardinalidad de *muchos.* (K1 en uno : muchos y K2 en muchos : 1)
- **Uno a uno.−** La clave primaria de **R** está formada por cualquiera de las claves primarias de los conjunto de entidades. (K1 **ó** K2)

3.5 Diagrama Entidad − Relación

Los diagramas Entidad − Relación representan la estructura lógica de una base de datos de manera gráfica. Los símbolos utilizados son los siguientes:

- **Rectángulo.−** Conjunto de entidades.
- **Elipse.**− Atributos.
- **Rombos.**− Conjunto de relaciones
- **Líneas.**− Unen atributos a conjuntos de entidades; unen atributos a conjuntos de relaciones; y unen conjuntos de entidades con conjuntos de relaciones. Si la flecha tiene punta, en ese sentido está el *uno* y si no la tiene, en ese sitio está el *muchos.* La orientación señala cardinalidad.
- **Elípse doble.**− Se trata de dos elipses concéntricas. Representan atributos multivalorados.
- **Elipse discontinua.**− Atributos derivados.
- **Líneas dobles.**− Indican participación total de un conjunto de entidades en un conjunto de relaciones.
- **Subrayado.**− Subraya los atributos que forman parte de la clave primaria del conjunto de entidades.

Si el conjunto de relaciones tiene atributos asociados, se le unen a la relación. En los diagramas Entidad − Relación se indican los papeles (roles) mediante etiquetas en las líneas que unen los conjuntos de relaciones con conjuntos de entidades.

Los conjuntos de relaciones no binarias se especifican uniendo al conjunto de relaciones tantas entidades como marque la relación. No es recomendable su utilización, prefiriéndose el uso de relaciones binarias.

3.6 Conjuntos de entidades débiles.

¿Tiene sentido un conjunto de entidades débiles para cualquier tipo de cardinalidad en un conjunto de relaciones binario? La respuesta es no. Sólo tiene sentido en el caso *uno a muchos.*

Se denomina **discriminante de un conjunto de entidades,** y se representa con un subrayado discontinuo, al conjunto de entidades que permiten la distinción entre las entidades que dependen de una entidad particular fuerte. El discriminante de un conjunto de entidades débil se llama **clave parcial del conjunto de entidades.**

La clave primaria de un conjunto de entidades débil se forma mediante la clave primaria del conjunto de entidades fuerte más el discriminante del conjunto de entidades débil.

El conjunto de entidades dominante se denomina **propietario** del conjunto de entidades débiles que identifica.

La relación que asocia al conjunto de entidades débil con un propietario se llama relación de identificación y se marca con un doble rombo..

3.7 Diseño de un esquema de bases de datos Entidad − Relación.

Para un diseño de un esquema de base de datos hay cuatro fases:

- **Especificación de requisitos del usuario.−** Consiste en obtener las necesidades de datos de los usuarios de la base de datos, esto es, sonsacarle al usuario toda la información que se desea plasmar en la base de datos. Esta es la fase que se dará en el examen.
- **Diseño conceptual (Entidad − Relación).**
- **Especificación de requisitos funcionales.−** Vamos a definir las operaciones que se harán sobre la base de datos (operaciones permitidas sobre la base de datos)
- **Especificación de requisitos funcionales.**− Primero se procede a realizar el **diseño lógico**, que consiste en adaptar el diseño conceptual al sistema de gestión de la base de datos, y a continuación se realiza el **diseño físico**, que consiste en dar todas las características de almacenamiento de la base de datos.

TEMA 4

EL MODELO RELACIONAL

4.1.− Elementos del modelo.

4.2.− Esquemas de relación.

4.3.− Dependencias funcionales.

4.4.− Dependencias transitivas, dependencias parciales, claves.

4.5.− Cierre de un descriptor respecto de un conjunto de dependencias funcionales.

4.6.− Recubrimientos no redundantes.

4.7.− Algoritmo de determinación de las claves de un esquema.

4.8.− Partición funcional.

4.9.− Algoritmo de Simplificación − Reducción.

4.10.− Algoritmo de Síntesis

4.1.− Elementos del modelo.

Se procederá a definir de una manera formal el método de datos. Todo modelo de datos se compone de tres partes:

- **Parte estructural.−** Se encarga de definir las estructuras de datos que sirven como base para el modelo a realizar.
- **Parte manipulativa.−** Es el conjunto de operadores que se van a tener para manejar la estructura de datos y poder trabajar con el modelo.
- **Parte semántica.−** Viene dada por las restricciones semánticas y las reglas de integridad. Marca el conjunto de restricciones que debe verificar el modelo

Parte estructural.

En el modelo de datos relacional la estructura de datos es la **relación**. La relación es un subconjunto del producto cartesiano de dominio. Se van a definir atributos para cada uno de esos dominios y las filas se denominan ***tuplas***.

A1	A2	...	An
a11	*a12*	...	*a1n*
a21	*a22*	...	*a2n*
...	...	...	...
am1	*am2*	...	*amn*

El subíndice de cada columna marca el grado de la relación, mientras que los subíndices de los elementos de la tabla indican su cardinalidad. En estas tablas *m* marca el número de tuplas, y *n* el de atributos. Como la base de datos es dinámica el valor de *m* no se va a mantener constante. El **dominio** es el rango de valores que puede alcanzar un atributo, o lo que es lo mismo, a cada uso particular de un dominio en una relación se le llama atributo.

(A " Dom(A)), r " Dom(A1) x Dom(A2) x ... x Dom(An).

Los dominios de dos o más atributos diferentes pueden ser coincidentes pero los nombres de su atributo asociado tienen que ser diferentes.

Una de las características más importantes es que se pueden tratar como conjuntos y por ello el orden de los atributos y de las tuplas es indiferente.

Parte manipulativa

Se van a tener los operadores del álgebra relacional que nos darán los operadores necesarios para manejar esta estructura. El SQL es una extensión del álgebra.

Parte semántica

Clave primaria.− Conjunto mínimo de atributos cuyo valor determina el de todos los demás de la relación. Cada relación va a tener una clave primaria y los dominios sobre los que se selecciona esta clave se llaman **dominios primarios**.

Clave externa.− También denominada **externa** o **foránea**. Se utiliza para enlazar relaciones y se define de la siguiente manera: es el atributo o conjunto de atributos que forma parte de la clave primaria de una relación **r**

y que aparece como clave primaria de otra relación **s**.

La parte semántica está formada por dos **reglas de integridad:**

Reglas de integridad (de entidad).− Ningún valor de la clave primaria de una relación puede ser o contener algún componente nulo (desconocido).

Reglas de integridad referencial.− Sea un atributo *A* de una clave primaria compuesta (más de un atributo) de una relación **r**, que está definido sobre un dominio primario. Entonces para cada valor *a* de *A* tiene que existir una relación **s** con clave primaria simple (*b*) tal que *a* ocurre como un valor de *b* en **s**.

Ejemplo.− S# P# CTD S# NomS Estado Ciudad

S1 P1 100 S1 X 10 L

S1 P2 20 a S2 Y 20 M

a S2 P3 30 S3 B 30 P

S2 P4 40 S4 X 40 S

S3 P5 30 S5 Y 10 L

4.2.− Esquemas de relación.

Dos ocurrencias **r1** y **r2** que pertenecen a T (conjunto de atributos) verifican el mismo conjunto de reglas de integridad, que vamos a llamar L y que estará formado por las reglas de integridad entidad y referencial y por todas las restricciones que permiten definir la semántica del problema.

El esquema de relación es el par formado por T y L. A cualquier subconjunto de T se le llama **descriptor.**

4.3.− Dependencias funcionales.

Sea x ! y una **dependencia funcional** (x, y " T). Se dice que y depende de x si x ! y " L., esto quiere decir, que para toda ocurrencia de **r** " R(T, L), siendo x e y subconjuntos de T el valor de x determina unívocamente al valor de y.

Dada una relación R, el atributo Y de R depende funcionalmente del atributo X de R si y sólo si, siempre que dos tuplas de R concuerden en su valor de X, deben por fuerza concordar en su valor de Y.

Ejemplo: Ver figura 4.1.

D.F. de S: S# ! NOMS. **D.F. de P:** P# ! NOMP. **D.F. de SP:** S#, P# ! CTD.

S# ! ESTADO. P# ! COLOR.

S# ! CIUDAD. P# ! PESO.

ESTADO ! CIUDAD. P# ! CIUDAD.

El conjunto L está formado por todas las dependencias funcionales del problema. Entonces dado el conjunto L siempre se pueden deducir otras dependencias funcionales a partir de él, esto es lo que se llama el conjunto L

+ que se denomina **cierre de L.**

L + se calcula en base a los **axiomas de Armstrong:**

- **Reflexividad.−** Para todo descriptor **x**, se deduce que **x** depende de si mismo. ("x, x ! x).
- **Aumentatividad.−** Si **y** depende de **x**, entonces **y** depende de **x'** siendo **x'** un superconjunto de **x**.

(x ! y ! x'! y, x' " x).

- **Proyectividad.−** Si **y** depende de **x**, entonces **y'** depende de **x**, siendo **y'** un subconjunto de **y**.

(x ! y ! x! y', y' " y).

- **Aditividad.−** Si **y** depende de **x** y si **w** depende de **u** entonces **y** unido con **w** depende de **x** unido con **u**. (x ! y, u ! w ! x " u ! y " w).
- **Transitividad.−** x!y, y!z ! x ! z.

4.4.− Dependencias transitivas, dependencias parciales, claves.

Se va a definir una **dependencia transitiva** entre dos descriptores (**x** e **y**) de la siguiente forma: x!y x " y = , x " z = , " z " T y se cumple que **z** depende de **x**: x ! z y z ! y ð z ! y.

Ejemplo.− S# ! Estado. S# ! Ciudad. Existe una dependencia transitiva entre

Estado ! Ciudad. Proveedor (S#) y Ciudad

Se va a decir que la dependencia **x ! y** es parcial si existe un **x'** subconjunto de **x**, tal que **y** depende de **x'**. (x ! y parcial ! " x' " x / x' ! y).

Ejemplo.− AB ! C Es una dependencia parcial, ya que hay un subconjunto de AB B ! C del que depende C.

Se va a decir que la dependencia **x ! y** es **total** si no existe ningún subconjunto de **x** del cual dependa **y**. (x ! y total ! no " x' " x / x' ! y).

Clave de un esquema.− Se dice que un descriptor K, subconjunto propio del conjunto de atributos (K " T) es **clave del esquema** R (T, L) cuando se cumple x T ó (K ! T) " L+, y no hay ningún subconjunto propio de K con la misma propiedad.

Un mismo esquema puede tener más de una clave. Los atributos que pertenecen a la clave se llaman atributos principales. Los atributos que no pertenecen a la clave se llaman atributos no principales.

4.5.− Cierre de un descriptor respecto de un conjunto de dependencias funcionales.

Sea un X un descriptor, siendo X subconjunto propio de T (X " T) y L un conjunto de dependencias funcionales. Cierre de X (que vamos a llamar X+) respecto al conjunto de dependencias funcionales L es un descriptor que cumple (X ! X+) " *L*+ y además X+ es máximo.

Ejemplo.− X ! Y, B

X ! Z Cierre de X (X+) es el conjunto de valores identificados por X.

Y ! B

El algoritmo para calcular el cierre de X es el siguiente:

Determinar el descriptor X(i) con la propiedad (X ! X(i)) " *L*+ y de forma que X(i) es X(i−1) incrementando en los atributos Ak tales que (U!V) " *L*, U " X(i−1) y Au " V, partiendo de X(0) = 0.

Ejemplo:

X ! Y X(0) = X

X ! Z X(1) = X, Y

Y ! B X(2) = X, Y, Z, B

Es un proceso finito ya que T también lo es y X+ es máximo ya que van a estar todos los atributos que dependen de X.

Ejemplo: Calcular el cierre del conjunto de atributos (BD)+ respecto al siguiente conjunto de dependencias funcionales:

AB ! C BE ! C X(0) = BD

C ! A CG ! BD X(1) = BDEG

BC ! D CE ! AG X(2) = BDEGC

ACD !B X(3) = BDEGCA

D ! EG

4.6.− Recubrimientos no redundantes.

Dos conjuntos con dependencias funcionales L y M son equivalentes si y sólo si sus cierres son iguales. Esto va a ocurrir si y sólo si f(x ! y) " L + está en M+ y f(u ! v) " M+ esta en L +. También se dice que L recubre a M y que M recubre a L.

La tarea de calcular el cierre de L es muy complicada, por lo tanto para saber si la dependencia X!Y que pertenece a *L* esta en M+ calculamos el cierre de X (X+) respecto a M ya que si se cumple (X!X+) " M entonces si Y " X+ ! X ! Y " M+.

Condiciones que debe cumplir un conjunto de dependencias funcionales para no ser redundante:

- Que todas las dependencias tengan su segundo miembro simple (formado por un único atributo): (X Ai) (Ai = segundos miembros simples).
- Que no haya dependencias funcionales redundantes. Una dependencia XAi " M es redundante en M cuando su supresión no altera el cierre. (M − XAi)+ = M+.
- No hay atributos extraños. Una atributo Bi " X es extraño en la dependencia funcional X!Ai que pertenece a M cuando llamando Z al descriptor X − {Bi} el cierre de M no se altera al sustituir X!Ai por Z ! Ai. M − {X ! Ai } " {Z ! Ai }+ = M+. Sólo hay atributos extraños en miembros compuestos.

Pregunta de Examen.− ¿Es posible que partiendo de un mismo conjunto de dependencias funcionales exista más de un recubrimiento no redundante?

Si, porque depende del orden en que lo hagamos. Pueden ser encontrados diferentes recubrimientos no redundantes, todos ellos equivalentes.

Dado un conjunto L de dependencias funcionales siempre va a existir un subconjunto equivalente M no redundante que será como mínimo igual a L.

El algoritmo para calcular el conjunto M va a contener tres pasos:

- Para toda dependencia X!Y de *L* se sustituye por X!A1, X!A2, ..., X!Ak, siendo A1, A2,.., Ak atributos de Y. Al conjunto resultante se le llama L *(1).*

Ejemplo: A ! BCD ! A ! B, A ! C, A! D.

- Para toda dependencia X ! Ai de L *(1)* determinamos el cierre de X respecto de L *(1)* − {X ! Ai}. Si Ai pertenece a X+ quiere decir que la dependencia X ! Ak es redundante en L *(1)* y se elimina. (Si Ak " X+ ! X ! Ak es redundante y se elimina). Al conjunto resultante se le llama L *(2).*
- Para toda dependencia X ! Ai de L *(2),* si Bi es un atributo de X (Bi "X) y siendo Z = X −{Bi} se calcula el cierre de Z respecto de L *(2).* Si Ai " Z, esto quiere decir que (Z ! Ai) " L (2)+, de modo que la sustitución de X ! Ai por Z ! Ai conduce a un conjunto equivalente.

El conjunto resultante se va a llamar L (3). Se tiene que L (3) = M.

Ejemplo de cálculo del recubrimiento no redundante de:

L*:*

AB ! C ACD ! B CG ! BD

C ! A D ! EG CE ! AG

BC ! D BE ! C

- Calculo de L *(1):*

AB ! C

C ! A

BC ! D

ACD !B Eliminado el paso d)

D ! E

D ! G

BE ! C

CG ! B

CG ! D Eliminado en el paso i)

CE ! A Eliminado en el paso j)

CE ! G

- Calculo de L *(2).*
- AB ! C , (AB)+ respecto de L *(1)* − {AB ! C} = AB

Como C " AB, AB ! C no es redundante.

- C ! A , (C)+ respecto de L *(1)* − {C ! A} = C

Como A " C, C ! A no es redundante.

- BC ! D , (BC)+ respecto de L *(1)* − {BC ! D} = BCA

Como D " BCA, BC ! D no es redundante.

- ACD ! B , (ACD)+ respecto de L *(1)* − {ACD ! B} = ACDEGB

Como B " ACDEGB, ACD ! B es redundante y se elimina.

- D ! E , (D)+ respecto de L *(1*) − {D ! E} = DG

Como E " DG, D ! E no es redundante.

- D ! G , (D)+ respecto de L *(1)* − {D ! G} = DE

Como G " DE, D ! G no es redundante.

- BE ! C , (BE)+ respecto de L *(1)* − {BE ! C} = BE

Como C " BE, BE ! C no es redundante.

- CG ! B , (CG)+ respecto de L *(1)* − {CG ! B} = CGADE

Como B " CGADE, CG ! B no es redundante.

- CG! D , (CG)+ respecto de L *(1)* − {CG ! D} = CGABD

Como D " CGABD, CG ! D es redundante y se elimina.

- CE ! A , (CE)+ respecto de L *(1)* − {CE ! A} = CEA

Como A " CEA, CE ! A es redundante y se elimina.

- CE ! G , (CE)+ respecto de L *(1)* − {CE ! G} = CEA

Como G " CEA, CE ! G no es redundante.

El conjunto resultado L *(2)* es el siguiente:

AB ! C D ! E CG ! B

C ! A D ! G CE ! G

BC ! D BE ! C

- El único atributo no compuesto subconjunto de algún atributo compuesto es C, se calcula el cierre de C, C+ = CA, como no se puede obtener otro a partir de este el conjunto L *(3)* es el mismo que L *(2)* .

4.7.− Algoritmo de determinación de las claves de un esquema.

Este algoritmo determina **todas** las claves del esquema. Dado un esquema de relación R y un conjunto de atributos y otro de dependencias funcionales donde Si van a ser los implicantes de la dependencia y Si " Sj.

R(T, L) $L = \{S_i \quad X_i\}_{i=1}^{m}$
; Si " Sj , Si " Xj = , "i

Ahora se hace la **matriz de implicación** que representa al conjunto de dependencias funcionales L.

Atributos de T

A1 A2 An

Implicantes S1

de cada una S2

de las S3

dependencias

funcionales Sm

La matriz se rellena de la siguiente forma; cada posición de la matriz va a tomar el valor: 1 si Aj pertenece a (Si " Xj) y tomará el valor 0 si Aj no pertenece a (Si " Xj).

1 si Aj " (Si " Xj)

Es decir: lij :

0 si Aj " (Si " Xj)

Ejemplo:

AB ! C

Si AC ! D Xj

B ! A

Matriz de implicación:

	A	B	C	D
AB	1	0	1	1

AC	1	0	1	1
B	1	1	0	0

Una vez calculada la matriz hay que calcular el cierre de L. Se calcula mediante el algoritmo del cierre transitivo de L.

ALGORITMO PARA EL CALCULO DEL CIERRE DE L**:**

L + = L

DO UNTIL L + no cambie más

" Si " Sj en L +, si " Ak " Sj lin = 1

Copiar todas las entradas `l' de la fila Si

en los lugares homólogos de la fina Si.

END DO

END.

Otro método para el calculo del cierre de L es el siguiente:

L + = $\{S_i \quad Y_i\}_{i=1}^{m}$
, Todos los atributos que están a 1 de L + .

Los 0 son todos aquellos descriptores de la forma T − (Si " Yi) = Y'i.

Ejemplo: (AB)+ = ABCD L + A B C D

(AC)+ = ACD AB 1 1 1 1

(B)+ = ABCD AC 1 0 1 1

B 1 1 1 1

Nomenclatura:

Si ! Implicantes. Y'i ! Ceros Yi ! Unos |x| nº de atributos de x.

ALGORITMO DE CLAVES: (Ver folio 4.1.).

4.8.− Partición funcional *.

La partición funcional permite reducir los tamaños de las matrices en el algoritmo de cálculo de claves, por tanto va a haber un menor consumo de memoria con una ganancia de rapidez, para ello se van a buscar las clases de equivalencia de las dependencias funcionales.

Descompone un esquema de relaciones en subesquemas, y se basa en el concepto de relación de equivalencia.

La **relación de equivalencia** cumple las propiedades reflexiva, simétrica y transitiva, y permite particionar un concurso.

Sea L: $\{S_i \quad X_i\}_{i=1}^{m}$
y Si " Sj, para todo i " j, y Sj ! Xi

Si fi " Si ! Xi fj " Sj ! Xj

Se va a definir una **relación de adyacencia** (R) de forma que:

fi está relacionado con fj si y sólo si (Si " Xi) " (Sj " Xj) " 0

fi no está relacionado con fj si y sólo si (Si " Xi) " (Sj " Xj) = 0

Esta relación es reflexiva y simétrica. A partir de esta relación se va a definir otra relación denominada **relación de conexión** (') de forma que fi " fj si y sólo si fi está relacionado con fj. La relación de conexión también es transitiva. En esta relación, si dos fi son adyacentes se conectan, y si no, habrá que buscar un camino entre ellas.

A partir de esto se definen las clases de equivalencia:

fi R f(Ki) i = 1

fi R fKj R f(Kj) ... R F(Ki).

La matriz de adyacencia va a estar formada por el conjunto de dependencias funcionales. Esta matriz también es llamada matriz de conexión y representa la relación de conexión.

	f1 f2 ... f2
f1 f2 ... fn	$l_{ij} = \begin{matrix} 1 \text{ si } f_i \quad f_j \\ 0 \text{ en otro caso} \end{matrix}$

Ejemplo:

f1: AB ! C

f2: B ! D

f3: E ! F

	f1	f2	f3
f1	1	1	0
f2	1	1	0
f3	0	0	1

Ejercicio:

Dado el esquema R (T, L)

T={A,B,C,D,F,G,H,I,J,K,L,M}

L={f1, f2, f3, f4, f5, f6}

f1: AB ! C

f2: H ! KL

f3: BD ! FG

f4: AE ! C

f5: KM ! L

f6: I ! J

- Se parte de un recubrimiento no redundante.
- Calculo de claves.
    - Partición funcional.

Matriz de adyacencia

	f1	f2	f3	f4	f5	F6
f1	1	0	1	1	0	0
f2	0	1	0	0	1	0
f3	1	0	1	0	0	0
f4	1	0	0	1	0	0
f5	0	1	0	0	1	0
f6	0	0	0	0	0	1

Matriz de conexión.

	f1	f2	f3	f4	f5	f6
f1	1	0	1	1	0	0
f2	0	1	0	0	1	0
f3	1	0	1	1 *	0	0
f4	1	0	1 *	1	0	0
f5	0	1	0	0	1	0
f6	0	0	0	0	0	1

* Celda que cambió de 0 a 1.

Clases de equivalencia.− Hay que encontrar filas de unos y ceros que se repitan y estas forman una clase de equivalencia:

[f1, f3, f4] , [f2, f5] , [f6]

- Algoritmo de claves (Para cada clase de equivalencia).
- 1) Calculo de L +.

L1: AB ! C T1 = {A,B,C,D,E,F,G} (AB)+ = ABC

BD ! FG (BD) + = BDFG

AE ! C (AE) + = AEC

L +	A	B	C	D	E	F	G
AB	1	1	1	0	0	0	0
BD	0	1	0	1	0	1	1
AE	1	0	1	0	1	0	0

- 2) M1 = , M2 = .
- 3) M1 = {ABDEFG, ABCDE, ABDEFG}
- 4) No hay ninguna fila que cumpla que |Yi| " 1, y otra que cumpla |Y'j| " 1.
- 5) No todas las entradas de M1 cumplen |Y'i| " 1.
- 6), 7) a12 = (DEFG) " (ABCDE) = DE

a13 = (DEFG) " (ABDEFG) = DEFG

a21 = (ACE) " (ABDEFG) = AE

a23 = (ACE) " (ABDEFG) = AE

a31 = (BDFG) " (ABDEFG) = BDFG

a32 = (BDFG) " (ABCDE) = BD

- 8) S1 " a12 = (AB) " (DE) = ABDE

S2 " a21 = (BD) " (AE) = ABDE M2 = {ABDE}

S3 " A32 = (AE) " (BD) = ABDE

- 9) 10) a212 = (ACE) " (AB " DE) = AE

a312 = (BDFG) " (AB " DE) = BD

- 11) S2 " a212 = BD " AE = ABDE

S3 " a312 = AE " BD = ABDE

M2 ={ABDE}

- 12) K1 = ABDE

L2: H ! KL T2 = {H,K,L,M}

KM ! L

K2 = HM

L3: I ! J T3 ={I,J}

K3 = I

CLAVE = K1" K2 " K3 = ABDEHIM.

Ejercicio:

f1: CD ! XY T = {A, B, C, D, E, X, Y}

f2: AX! B L = {f1, f2, f3, f4}

f3: BY ! C

f4: C ! A

- Partición funcional.

Matriz de adyacencia

	f1	f2	f3	f4
f1	1	1	1	1
f2	1	1	1	1
f3	1	1	1	1
f4	1	1	1	1

La matriz de conexión es igual a la matriz de adyacencia.

Clases de equivalencia: [f1, f2, f3, f4]

- Algoritmo de claves.
- 1) Calculo de L +.

(CD) += ABCDXY (AX) += ABX (BY) += ABCY

(C) += CA

L +	A	B	C	D	E	X	Y
CD	1	1	1	1	0	1	1
AX	1	1	0	0	0	1	0
BY	1	1	1	0	0	0	1
C	1	0	1	0	0	0	0

- 2) M1 = , M1 = .
- 3) M1 = {CDE, ACDEXY, BDEXY, BCDEXY }
- 4) No hay ninguna fila que cumpla que |Yi| " 1, y otra que cumpla |Y'j| " 1.
- 5) No todas las entradas de M1 cumplen |Y'i| " 1.
- 6) 7) a21 = (CDEY) " (CDE) = CDE

a23 = (CDEY) " (BDEXY) = DEY

a24 = (CDEY) " (BCDEXY) = CDEY

a31 = (DEX) " (CDE) = DE

a32 = (DEX) " (ACDEXY) = DEX

a34 = (DEX) " (CBDEXY) = DEX

a41 = (BDEXY) " (CDE) = DE

a42 = (BDEXY) " (ACDEXY) = DEXY

a43 = (BDEXY) " (BDEXY) = BDEXY

- 8) S2 " a21 = (AX) " (CDE) = ACDEX

S2 " a23 = (AX) " (DEY) = ADEXY

S3 " a31 = (BY) " (DE) = BDEY M2 = {ADEX, BDEY, CDE}

S4 " a41 = (C) " (DE) = CDE

- 9) 10) a323 = (DEX) " (AX " DEY) = DEX

a423 = (BDEXY) " (AX " DEY) = DEXY

a231 = (CDEY) " (BY " DE) = DEY

a431 = (BDEXY) " (BY " DE) = BDEY

a241 = (CDEY) " (C " DE) = CDE

a341 = (DEX) " (C " DE) = DE

- 11) S2 " a231 = AX " DEY = ADEXY

S2 " a241 = AX " CDE = ACDEX

S3 " a341 = BY " BE = BEY

S4 " a423 = C " DEXY = CDEXY

S4 " a431 = C " BDEY = CBDEY

M2 = {ADEX, BDEY, CDE}

- 12) **CLAVE = CDE**

Ejercicio:

f1: X ! D T={A, B, C, D, X, Y}

f2: CY! X L={f1, f2, f3, f4, f5}

f3: DY ! C

f4: AX ! B

f5: AY ! C

- Partición funcional.

Matriz de adyacencia

	f1	f2	f3	f4	f5
f1	1	1	1	1	0
f2	1	1	1	1	1
f3	1	1	1	0	1
f4	1	1	0	1	1
f5	0	1	1	1	1

Matriz de conexión

	f1	f2	f3	f4	f5
f1	1	1	1	1	1*
f2	1	1	1	1	1
f3	1	1	1	1*	1
f4	1	1	1*	1	1
F5	1*	1	1	1	1

* Celda que cambió de 0 a 1.

Clases de equivalencia: [f1, f2, f3, f4, f5]

- Algoritmo de claves.

- 1) Calculo de L +.

(X) += XD (CY) += CYXD (DY) += DYCX

(AX) += AXBD (AY) += AYCXDB

L +	**A**	**B**	**C**	**D**	**X**	**Y**
X	0	0	0	1	1	0
CY	0	0	1	1	1	1
DY	0	0	1	1	1	1
AX	1	1	0	1	1	0
AY	1	1	1	1	1	1

- 2) M1 = , M1 = .
- 3) M1 = {ABCX, ABCY, ABDY, ACXY, AY}
- 4) Hay una fila que cumple |Y'i| " 1, pero no hay ningún otra que cumpla |Yj| " 1.
- 5) No todas las entradas de M1 cumplen |Y'i| " 1.
- 6) 7) a12 = (ABCY) " (CY " AB) = ABCY

a13 = (ABCY) " (DY " AB) = ABY

a14 = (ABCY) " (AX " CY) = ACY

a15 = (ABCY) " (AY ") = AY

a21 = (AB) " (X " ABCY) = AB

a23 = (AB) " (DY " AB) =AB

a24 = (AB) " (AX " CY) = A

a25 = (AB) " (AY ") = A

a31 = (AB) " (X " ABCY) = AB

a32 = (AB) " (CY " AB) = AB

a34 = (AB) " (AX " CY) = A

a35 = (AB) " (AY ") = A

a41 = (CY) " (X " ABCY) = CY

a42 = (CY) " (CY " AB)= CY

a43 = (CY) " (DY " AB) =Y

a45 = (CY) " (AY ") = Y

- 8) S1 " a15 = X " AY = AXY

S2 " a25 = CY " A = ACY M2 = {ACY, ADY, AXY}

S3 " a35 = DY " A = ADY

S4 " a45 =AX " Y = AXY

- 9) 10) a125 = (ABCY) " (CY " A) = ACY

a325 = (AB) " (CY " A) = A

a425 = (CY) " (CY " A) = CY

a135 = (ABCY) " (DY " A) = AY

a235 = (AB) " (DY " A) = A

a435 = (CY) " (DY " A) = Y

a145 = (ABCY) " (AX " Y) = AY

a245 = (CY) " (AX " Y) = Y

a345 = (AB) " (AX "Y) = A

- 11) S1 " a145 = X " AY = AXY

S2 " a235 = CY " A = ACY

S2 " a245 = CY " Y = CY

S3 " a345 = DY " A = ADY

S4 " a435 = AX " Y = AXY

M2 = {ACY, ADY, AXY}

- 12) Se copia en M2 los descriptores Si " Y'i con |Y'i| "1, en este caso AY y se borran superconjuntos.

M2 = {ACY, ADY, AXY, AY}

CLAVE = AY

TEMA 5

ÁLGEBRA RELACIONAL

5.1.− Introducción y definición intuitiva.

5.2.− Sintaxis para el manejo de las expresiones relacionales.

5.3.− Los operadores tradicionales.

5.4.− Los operadores relacionales típicos.

5.1.− Introducción

Hasta ahora se han distinguido dos aspectos de las bases de datos: La estructura y el manejo.

Para manejar las estructuras se siguen dos líneas, que son el **álgebra relacional** y el **cálculo relacional.**

Álgebra relacional.− El álgebra relacional consiste en un conjunto de operadores de alto nivel que operan sobre relaciones. Cada uno de estos operadores toma una o dos relaciones como entrada y produce una nueva relación como salida.

Fue Codd quién en el año 1973 diseñó una serie de operadores que le permitieran trabajar con la estructura relacional que el mismo había definido. Estos operadores son los siguientes:

- **Tradicionales ó conjuntistas:**

- Unión .
- Intersección.
- Diferencia.
- Producto cartesiano.

- **Relacionales ó propios:**

- Selección
- Proyección.
- Reunión.
- División

Los operadores 1, 2, 3, 4, 7 y 8 son **binarios**, es decir, dadas dos relaciones se obtiene una y los operadores 5 y 6 son **monarios**, de una relación obtienen otra.

Definición intuitiva.

- **Unión**: Construye una relación formada por todas las tuplas que aparecen en cualquiera de las dos relaciones especificadas. (UNION)
- **Intersección**: Construye una relación formada por aquellas tuplas que aparezcan en las dos relaciones especificadas, es decir, que tienen los mismos atributos. (INTERSECT)
- **Diferencia**: Construye una relación formada por todas las tuplas de la primera relación que no aparezcan en la segunda de las dos relaciones especificadas. (MINUS)
- **Producto cartesiano**: A partir de dos relaciones especificadas, construye una relación que contiene todas las combinaciones posibles de tuplas, una de cada una de las dos relaciones. (TIMES)

Ejemplo:

X	Y	**Z**	W
a	1	***a***	3
a	1	a	4
b	2	a	3
b	2	a	4

X	Y

a	1
b	2
Z	W
a	3
a	4

- **Selección**: Extrae las tuplas especificadas de una relación dada, o lo que es lo mismo, restringe la relación sólo a las tuplas que satisfagan una condición especificada (Selecciona filas).
- **Proyección**: Extrae los atributos especificados de una relación dada (Selecciona columnas).
- **Reunión**: A partir de dos relaciones especificadas, construye una relación que contiene todas las posibles combinaciones de tuplas, una de cada una de las dos relaciones, tales que las dos tuplas participantes en una combinación dada satisfagan alguna condición especificada. Las tuplas deben tener algún atributo en común. Es por esto que las bases de datos deben estar normalizadas. (JOIN)

Ejemplo:

X	C
a	Rojo
b	Azul
c	Amarillo

X	**Y**	C
a	1	Rojo
b	1	Azul
b	2	Azul
b	3	Azul

X	Y
a	1
b	1
b	2
b	3

- **División**: toma dos relaciones, una binaria y una unaria, y construye una relación formada por todos los valores de un atributo de la relación binaria que concuerdan (en el otro atributo) con **todos** los valores en la relación.

Ejemplo:

	Y
	1
	2
	3

	X
	b

X	Y
a	1

b	1
b	2
b	3

Sólo ha cogido a *b* porque tiene asociados a 1, 2 y 3, todos los valores indicados en la tabla Y

5.2.− Sintaxis para el manejo de las expresiones relacionales.

Para definir una sintaxis para el manejo de las expresiones relacionales vamos a utilizar la gramática BNF. Esta gramática tiene una serie de valores:

- Valores terminales: El nombre de la relación, el nombre del atributo y los predicados. Un predicado es una expresión lógica entre atributos del mismo dominio y que da como resultado *verdadero* o *falso*.
- Va a haber operadores lógicos: (AND, OR, NOT).
- También habrá operadores clásicos (<, <= ,>, >=, =, <>).

Una gramática BNF para el álgebra relacional es la siguiente:

- def_rel ::= DEFINE RELACION nombre_relación [nombre_atributo]
- def_alias ::= DEFINE ALIAS nombre_relación FOR nombre_relación
- expr ::= selección | proyección | expresión infija
- selección ::= primitiva WHERE predicado
- primitiva ::= nombre _ relación | (expr)
- proyección ::= primitiva | primitiva [esp _ atrib]
- esp _ atrib ::= nombre _ atributo | nombre _ relación nombre _ atributo
- expr _ infija ::= proyección op _ infija proyección
- op _ infija ::= UNION, INTERSECT, MINUS, TIMES, JOIN, DIVIDE BY

5.3.− Los operadores tradicionales.

La unión, la intersección y la diferencia entre relaciones deben de cumplir la **relación de compatibilidad**. Esta regla dice que dadas dos relaciones A y B son compatibles para la unión, intersección y diferencia si y solo si verifican las dos siguientes condiciones:

- El grado de A tiene que ser igual al grado de B. Grado(A) = Grado(B)
- Si A tiene atributos a1, a2, ..., an, y B tiene atributos b1, b2, ..., bn, el dominio de cada uno de estos atributos tienen que ser iguales. A[a1, a2, ..., an] y B[b1, b2, ..., bn] Dom (Ai) = Dom (Bi).

El producto cartesiano devuelve una relación que es el resultado de la construcción de dos relaciones. La unión, intersección y diferencia van a consistir en operar dos conjuntos que verifiquen la relación de compatibilidad. Los conjuntos que verifiquen esta relación son iguales estructuralmente

Ejemplo: Dadas dos relaciones A y B.

A A1 A2 B B1 B2

X 1 X 1

X 2 Y 2

Y 1 Y 3

Y 2 X 4

A UNION B A INTERSECT B A MINUS B B MINUS A

X 1 X 1 X 2 X 4

X 2 Y 2 Y 1 Y 3

X 1

Y 2

Y 3

X 4

El producto cartesiano (TIMES) es cerrado, con lo que obtenemos una relación a partir de otras dos. Siempre que se hace el producto cartesiano para dos conjuntos con un atributo en común, se le pone **siempre** delante del nombre del atributo el nombre de la relación. A TIMES B es una relación / " t " A, r " B, (t, r) " A TIMES B. El producto cartesiano es asociativo y conmutativo.

Ejemplo:

SP S# P# CTD S S# Noms Estado

S1 P1 300 S1 Sala 20

S2 P1 300 S2 Jara 10

S4 P5 400 S4 Alda 30

SP TIMES S

SP.S# P# CTD S.S# Noms Estado

S1 P1 300 S1 Sala 20

S1 P1 300 S2 Jara 10

S1 P1 300 S4 Alda 30

S2 P1 300 S1 Sala 20

S2 P1 300 S2 Jara 10

S2 P1 300 S4 Alda 30

S4 P5 400 S1 Sala 20

S4 P5 400 S2 Jara 10

S4 P5 400 S4 Alda 30

El producto cartesiano tiene un problema cuando se define un producto cartesiano del mismo conjunto ya que se repetirían el nombre de los atributos, y estos deben de ser únicos. Esto se soluciona definiendo un alias para la relación.

Ejemplo:

DEFINE ALIAS SP FOR A

SP TIMES A

SP.S# SP.P# SP.CTD A.S# A.P# A.CTD

Siempre que se pide buscar parejas de algo hay que hacer el producto cartesiano de una relación por sí misma.

5.4.− Los operadores relacionales típicos.

WHERE (selección).− Si P es un predicado que se puede construir con los atributos de una relación R, entonces R WHERE P, es la selección según el predicado P, es decir, se queda con las tuplas de la relación que hacen cierto el predicado P. (**S WHERE P**). El predicado P puede ser compuesto mediante los operadores AND, OR, < ,..., y devuelve verdadero o falso.

Proyección.− Si se tiene una relación R con atributos A1, A2,..., Am, se dice que se ha proyectado R sobre el conjunto A de atributos A1, A2,..., Am cuando se eliminan de R todas las columnas que no están en el conjunto A y se eliminan las tuplas repetidas que puedan aparecer.

Ejemplo:

SP S# P# CTD [S#, CTD] S# CTD

S1 P1 300 S1 300

S2 P1 300 S2 300

S1 P2 300 S1 300 * se eliminan tuplas repetidas.

JOIN (Reunión).− Sean A y B dos relaciones y P un predicado que conecta atributos de las dos. Se llama reunión al resultado de (A TIMES B) WHERE P. Esto recibe el nombre de Reunión (JOIN).

Ejemplo:

SP S# P# CTD S# Estado SP JOIN B S# P# CTD Estado

S1 P1 100 S1 10 S1 P1 100 10

S1 P2 200 S1 P2 200 10

S1 P3 100 S1 P3 100 10

S2 P1 100

La reunión elimina campos en comparación de igualdad de atributos.

DIVIDE BY (División).− Sea A una relación de grado m + n y B otra relación de grado n. El operador división produce una nueva relación de grado m, donde el (m + i)−esimo atributo de A y el i−esimo atributo de B (i en el rango de 1 a n) deben estar definidos sobre el mismo dominio.

Ejemplo: Proveedores que suministran todas las piezas.

SP [S#, P#] DIVIDE BY S[P#]

SP[S#, P#] S# P# DIVIDE BY S[P#] P# = P#

S1 P1 P1 P1

S1 P2 P2

S2 P3

Encontrar todas las piezas de color rosa:

SP[S#, P#] DIVIDE BY ((P WHERE Color = Rosa)[P#])

TEMA 6

CÁLCULO RELACIONAL

6.1.− Introducción.

6.2.− El cálculo de predicados en Bases de Datos relacionales.

6.3.− El cálculo relacional orientado a tuplas.

6.1.− Introducción.

El álgebra relacional y el cálculo relacional son dos alternativas para establecer una base formal de la parte manipulativa del modelo relacional. La diferencia entre ellas es la siguiente: mientras que el álgebra ofrece un conjunto de operaciones explícitas (reunión, unión,...) que pueden servir en la práctica para indicar al sistema la forma de construir alguna relación deseada a partir de las relaciones dadas en la base de datos, el cálculo solo ofrece una notación para formular la definición de esa relación deseada en términos de esas relaciones dadas.

El cálculo solo plantea el problema y el álgebra proporciona un procedimiento para resolver ese problema. En realidad el álgebra y el cálculo son totalmente equivalentes. Para cada expresión del álgebra, existe una expresión equivalente en el cálculo; de manera similar para cada expresión del cálculo, existe una expresión equivalente en el álgebra.

El cálculo relacional se fundamenta en una rama de la lógica matemática llamada cálculo de predicados.

6.2.− El cálculo de predicados en Bases de Datos relacionales.

Una herramienta que permite representar el conocimiento del mundo real es el cálculo de predicados.

El cálculo de predicados consta de los siguientes elementos:

- **Dominio del discurso.−** Conjunto de objetos de nuestro mundo con una entidad propia, por ejemplo, personal, coches,...
- **Objetos.−** Son constantes del dominio del discurso, por ejemplo, dentro de personas: Juan, Pepe, dentro de colores: azul, verde,...
- **Variables.−** Representación en un momento dado un objeto del domino. Una variable x tomará algún valor en un determinado momento, si hablamos de colores x podría tomar en algún momento el azul.
- **Funciones.−** Permiten relacionar un dominio con otro, por ejemplo, podemos tener una función padre que va del dominio personal al dominio persona. Padre (Juan): devolverá el valor del padre de Juan.
- **Predicados.−** Son relaciones entre dominios cuya evaluación da como resultado *verdadero* o *falso*, por ejemplo: casado (Juan, María). Mediante los predicados se representan sentencias que se denominan **fórmulas atómicas.**
- **Fórmulas atómicas**.− Van a tener un valor real, es decir, una función que, dados los argumentos apropiados, produce un valor falso o verdadero. Se pueden unir distintas fórmulas mediante juntores:

- AND (')
- OR (')
- NOT (')
- IMPLICA (!)

Ejemplo: F1 " Escribió (Cervantes, El Quijote) ! (V)

F2 " Nació (Cervantes, Monforte) ! (F).

F1 " F2 ! (V).

F2 " F2 ! (F).

Cualquier conjunto de fórmulas atómicas se llaman fórmulas bien formadas (WFF ó FBF)

Ejemplo: Si suspendo B.D. o me pego un tiro o se lo pego al profesor.

Suspendo (yo, BD) ! Pego (yo, yo) " Pego (yo, profesor).

Suspendo (yo, x) ! Pego (yo, x) " Pego (yo, profesor (x)).

- **Cuantificadores.−** Restringen el valor que puede tomar la variable x. Dicen en que forma es cierta una fórmula que implica variables. Hay dos tipos de cuantificadores.

- Cuantificador universal ("x).
- Cuantificador particular ("x).

Cundo una variable esté cuantificada se le llama variable ligada y cuando una variable no esté cuantificada se le llama variable libre. Para la representación de las relaciones en término del cálculo de predicados hay dos formas que son las siguientes:

- Cálculo relacional orientado a tuplas.
- Cálculo relacional orientado a dominios.

6.3.− El cálculo relacional orientado a tuplas.

Las expresiones del cálculo de tuplas se construyen a partir de los elementos siguientes:

- **Variables de tupla** T, U, V, Cada variable de tupla se restringe a variar sobre alguna relación con nombre. Si la variable de tupla T representa a la tupla t, entonces la expresión T.A representa al componente A de T, donde A es un atributo de la relación sobre la cual varía T.

- **Condiciones de la forma x*y** donde * es cualquiera de los símbolos =, <>, <, <=, >, >=, y al menos una de entre x e y es una expresión de la forma T.A y la otra es una expresión semejante o una constante.

- **Fórmulas bien formadas**: Son la unión de fórmulas atómicas.

Variables libres y acotadas.− Cada ocurrencia de una variable de tupla dentro de una formula bien formada es libre ó acotada.

- Dentro de una condición, todas las ocurrencias de las variables de tupla son libres.
- Las ocurrencias de las variables de tupla en las fórmulas bien formadas (f), "(f) son libres/acotadas según sean libres/acotadas en f. Las ocurrencias de las variables de tupla en las fórmulas bien formadas (f " g), (f " g) son libres/acotadas según sean libres/acotadas en f o en g (cualquiera de las dos en donde aparezcan).
- Las ocurrencias de T que sean libres en f son acotadas en las fórmulas bien formadas, " (f), "T(f)

Una expresión del cálculo de tuplas es de la forma T.A, U.B, , V.C [WHERE f] donde T, U, , V son variables de tupla; A, B, , C son atributos de las relaciones asociadas, y f es una formula bien formada que contiene exactamente T, U, , V como variables libres. El valor de esta expresión es una proyección del subconjunto del producto cartesiano T x U x x V para la cual f se evalúa como verdadera.

Sintaxis BNF para cálculo relacional orientado a tuplas.− Ver fotocopia 6.1.

Solución a los ejercicios de álgebra relacional utilizando cálculo relacional.

RANGE OF AX IS ALUMNOS

RANGE OF AY IS ALUMNOS

RANGE OF MX IS MATRICULA

RANGE OF MY IS MATRICULA

RANGE OF SX IS ASIGNATURA

RANGE OF SY IS ASIGNATURA

RANGE OF PX IS PROFESOR

RANGE OF PY IS PROFESOR

- MX.AL# WHERE MX.ASIG#=`BD3'
- MX.AL· WHERE " SX(MX.ASIG# = SX.ASIG# " SX.NOMBRE = `BD')
- PX.PR# WHERE " SX(PX.ASIG# = SX.ASIG# " SX.NOMBRE = `EDI')
- SX.NOMBRE, SX.CURSO WHERE " MX(MX.ASIG# = SX.ASIG# " MX.AL#=`Pepito Perez')
- PX.PR# WHERE " MX(MX.ASIG# = PX.ASIG# " MX.AL# = `Pepito Perez')
- MX. AL# WHERE " PX(PX.ASIG# = MX.ASIG# " PX.PR# = `Armando Guerra Segura')
- MX.AL# WHERE " SX(SX.ASIG# = MX.ASIG# " SX.CURSO = `2º')
- MX.AL# WHERE " SX(SX.ASIG# = MX.ASIG# " SX.CURSO = `2º')

- MX.AL# WHERE " SX(SX.ASIG#=MX.ASIG# " SX.CURSO=`2º') " ("" SX(SX.ASIG# = MX.ASIG# " SX.CURSO=2º))
- AX.AL# WHERE (AX.BECA=`SI' " AX.PROVINCIA<>`Orense')
- AX.AL# WHERE (AX.PROVINCIA<>`Orense') " " MX(MX.AL#=AX.AL#) " " SX(SX.ASIG# = MX.ASIG# " SX.CURSO = `1º')))
- AX.AL# WHERE (AX.EDAD>25) " " MX(MX.AL#=AX.AL#) " " SX(SX.ASIG# = MX.ASIG# " SX.CURSO=`1º')))
- PX.PR#, PY.PR# WHERE (PX.ASIG#=PY.ASIG# " PX.PR#<>PY.PR#))
- SX.NOMBRE WHERE " MX(MX.ASIG# = SX.ASIG#) " " AX(AX.AL#=MX.AL# " AX.PROVINCIA=`Pontevedra')))

TEMA 7

TEORÍA DE DISEÑO DE BASES DE DATOS RELACIONALES

7.1.− Descomposición de esquemas.

7.2.− Descomposición con la propiedad de unión sin pérdida de información.

7.3.− Test de la propiedad LJ.

7.4.− Descomposición con preservación de dependencias.

7.5.− Algoritmo de Test de preservación de dependencias.

7.6.− Formas Normales basadas en dependencias funcionales.

7.6.1.− Formas Normales de Codd.

7.6.1.1.− Primera Forma Normal.

7.6.1.2.− Segunda Forma Normal.

7.6.1.3.− Tercera Forma Normal.

7.6.2.− Forma Normal de Boyce−Codd

7.7.− Algoritmo de descomposición de Forma Normal de Boyce−Codd con la propiedad LJ.

7.8.− Descomposición en Tercera Forma Normal de Codd con preservación de dependencias.

7.9.− Descomposición en 3ª Forma Normal de Codd con preservación y verificación de la propiedad LJ.

7.1.− Descomposición de esquemas.

Dado un esquema R, con un conjunto de atributos T y un conjunto de dependencias funcionales F se plantea el problema de que se puede descomponer una serie de proyecciones f = {R1, R2, ..., Rn} de forma que debe representar la misma información que R.

Esto es interesante para evitar las anomalías que se pueden producir en esquemas grandes. Pueden surgir tres anomalías de Codd:

- **Anomalías de Inserción.−** Se produce una pérdida de información porque no se puede insertar una tupla en una relación al no conocer el valor de los atributos primarios.
- **Anomalías de Borrado.−** Consiste en una pérdida de información como consecuencia del borrado de una tupla porque se pierde toda la información de todos sus atributos.
- **Anomalías de Modificación.−** Necesidad de propagar modificaciones debido a un diseño redundante.

Al proceso de sustitución de un esquema por sus proyecciones se le denomina **normalización.**

7.2.− Descomposición con la propiedad de unión sin pérdida de información.

Sea **f** = {R1,..., R2} una descomposición del esquema R, y sea Ri (Ti, Li) la especificación completa del esquema i−ésimo. Se dice que f es una descomposición con la propiedad de Unión sin pérdida de información respecto de L si para toda ocurrencia **r** del esquema R (r ðR) se verifica la siguiente igualdad:

R = R1 [T1] JOIN ... JOIN RK [TK]

La información origen debe ser la **misma** que la contenida en las extensiones del conjunto Ri de esquemas resultantes. Esta fórmula indica que si se hace la unión de todas las proyecciones debe de dar el esquema original. Esto es la llamada **propiedad LJ.**

7.3.− Test de la propiedad LJ*.

Sea una descomposición de R (T, L) con T={A1, A2,An}, L={x ! y / x " y " T} y f = {R1, ..., R2}. Para verificar el cumplimiento en **f** de la propiedad LJ se va a utilizar el siguiente algoritmo:

Se constituye una tabla de **k** filas y **n** columnas. La columna **j** corresponde al atributo **Aj** y la fila **i** al esquema Ri.

En la intersección de ambas se coloca el símbolo **aj** si el atributo de esa posición pertenece al conjunto de atributos y el símbolo **bij** en el caso contrario. Se considera a continuación cada una de las dependencias funcionales (x!y)ð L. Si se encuentran dos filas que coinciden en las entradas correspondientes a **x** entonces se iguala a **Y** (x = y) de la siguiente manera:

- Si un símbolo es **aj** se hace su homólogo igual a **aj**.
- Si ambos son del tipo **b** se igualan los subíndices de uno cualquiera de ellos a los del otro.

Este proceso se repite hasta que la tabla no varíe. Si finalmente se encuentra al menos una fila de la forma (a1, a2, a3, , an) la descomposición verifica la propiedad LJ y no la verificará en caso contrario.

Ejemplo: Dado el siguiente esquema R (T, L).

T = {A, B, C, D, E}

L = {A ! C, B ! C, C ! D, DE ! C, CE ! A}

F = {AD, AB, BE, CDE, AE}

B! C	A	**B**	**C**	D	E
AD	a1	b12	b13	a4	b15
AB	a1	***a2**	**b13**	b24	b25
BE	b31	***a2**	**b33**	b34	a5

CDE	b41	b12	a3	a4	a5
AE	a1	b52	b13	b54	a5

A!C	**A**	B	**C**	D	E
AD	***a1**	b12	**b13**	a4	b15
AB	***a1**	a2	**b23**	b24	b25
BE	b31	a2	b33	b34	a5
CDE	b41	b12	a3	a4	a5
AE	***a1**	b52	**b53**	b54	a5

DE! C	A	B	**C**	**D**	***E***
AD	a1	b13	b13	a4	b15
AB	a1	b13	b13	a4	b25
BE	b31	b13	**b33**	***a4**	***a5**
CDE	b41	a3	**a3**	***a4**	***a5**
AE	a1	b13	**b13**	***a4**	***a5**

C!D	A	B	**C**	**D**	E
AD	a1	b12	***b13**	**a4**	b15
AB	a1	a2	***b13**	**b24**	b25
BE	b31	a2	***b13**	**b34**	a5
CDE	b41	b42	a3	a4	a5
AE	a1	b52	***b13**	**b54**	a5

	A	B	C	D	E
AD	a1	b12	b13	a4	b15
AB	a1	a2	b13	a4	b25
BE	**a1**	**a2**	**a3**	**a4**	**a5**
CDE	a1	b42	a3	a4	a5
AE	a1	b52	a3	a4	a5

CE!A	**A**	B	**C**	D	**E**
AD	a1	b12	b13	a4	b15
AB	a1	a2	b13	a4	b25
BE	**b31**	a2	***a3**	***a4**	a5
CDE	**b41**	b42	***a3**	***a4**	a5
AE	**a1**	b52	***a3**	***a4**	a5

Al haber una fila que cumple (a1, a2, a3, a4, a5) se verifica la propiedad LJ.

Sea R (T, L). Su descomposición en **exactamente** dos subesquemas verifica la propiedad LJ se cumple que:

(T1 " T2) ! (T1 − T2) ð L ó (T1 " T2) ! (T2 − T1) ð L +

7.4.− Descomposición con preservación de dependencias.

Para poder realizar la descomposición con preservación de dependencias, el conjunto de dependencias funcionales de partida debe ser equivalente al conjunto de dependencias de los esquemas funcionales

resultantes.

Se parte de un esquema R (T, L) y una descomposición f = {R1, R2, Rk}, L ={x ! y / x " y " T}. Para Ri (Ti, Li) es la proyección de L.

Ri (Ti, Li), Li ={Xi ! Yi " L+/ X " Y " Ti}. Se cumple la preservación de dependencias si se verifica que $(\bigcup_{i=1}^{k} L_1)^+ = L^+$