CICLO FORMATIVO DE GRADO SUPERIOR: ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS

MÓDULO 6: SISTEMAS GESTORES DE BASES DE DATOS

UNIDADES DIDÁCTICAS

UNIDAD DIDÁCTICA Nº 1

INTRODUCCIÓN A LOS SISTEMAS GESTORES DE BASES DE DATOS

1.1. Introducción

Un sistema de manejo de base de datos (DBMS, database management system), consiste en un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a esos datos. El conjunto de datos se conoce comúnmente como base de datos. El objetivo primordial de un DBMS es crear un ambiente en que sea posible guardar y recuperar información de la base de datos en forma conveniente y eficiente.

El manejo de los datos incluye tanto la definición de las estructuras para el almacenamiento de la información como los mecanismos para el manejo de la información, además el sistema de B.D. debe cuidar la seguridad de la información almacenada en la base de datos, y controlar los accesos simultaneos de los usuarios.

1.2. Objetivos de los sistemas de bases de datos

El objetivo principal de un SGBD (sistema gestor de bases de datos), es terminar con todos los problemas que plantean los sistemas de procesamiento de archivos clásico, cuyos principales problemas son los siguientes:

Redundancia e inconsistencia de los datos

Puesto que los archivos y los programas de aplicaciones pueden haber sido creados por distintos programadores, es posible que un mismo dato esté repetido en varios sitios (archivos), cuya redundancia aumenta los costos de almacenamiento y acceso, además de incrementar la posibilidad de que exista inconsistencia en la información.

Dificultad para tener acceso a los datos

Para obtener datos que no estan contemplados en la aplicación actual, o nos tenemos que quedar sin ellos, o el departamento de informática (o el programador) nos tiene que hacer una aplicación para extraer esa información que necesitamos, y que además, puede que mañana sea otra la que necesitemos.

Aislamiento de los datos

Puesto que los datos están repartidos en varios archivos, y éstos pueden tener diferentes formatos, es difícil escribir nuevos programas de aplicaciones para obtener los datos apropiados.

Usuarios múltiples

Como muchos sistemas permiten que varios usuarios actualicen la información de forma simultánea, y puesto que muchos programas de aplicaciones diferentes sin coordinación previa pueden tener acceso a los datos al mismo tiempo, podrían provocar actualizaciones inconsistentes.

Problemas de seguridad

Poder evitar que todos los usuarios del sistema de base de datos puedan tener acceso a toda la información.

Problemas de integridad

Los valores de los datos que se guardan en la base de datos deben satisfacer ciertos tipos de limitantes de consistencia, por ejemplo que el saldo de una cuenta bancaria no debe bajar nunca de un límite fijado (p. ej. 5 pta.). Los sistema de bases de datos si que comtemplan estas posibilidades, y no lo hacen los sistemas de archivos.

1.3. Abstracción de la información

Un sistema de manejo de base de datos es un conjunto de archivos interrelacionados y una serie de programas que permiten a varios usuarios tener acceso a estos archivos y modificarlos. Uno de los objetivos principales de un sistema de base de datos es proporcionar a los usuarios una visión abstracta de la información.

Exiten varios niveles de abstracción:

Nivel físico

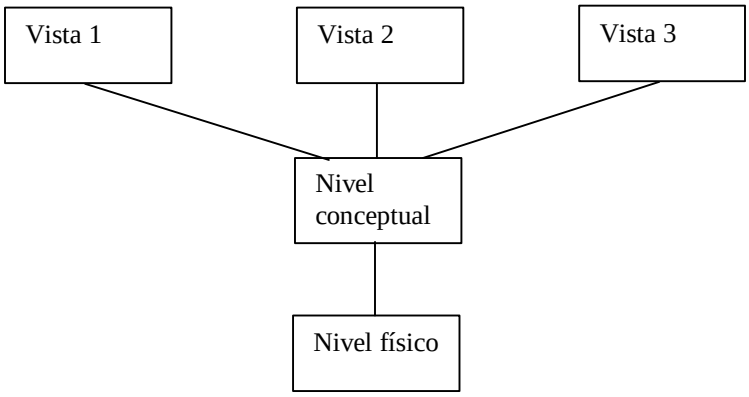
Es el nivel más bajo de abstracción, en el que se describe cómo se almacenan realmente los datos.

Nivel conceptual

Este es el siguiente nivel más alto de abstracción, en el que se describen cuáles son los datos reales que están almacenados en la base de datos y qué relaciones existen entre los datos. Este nivel lo utilizan los administradores de base de datos, quienes deciden qué información se guarda en la base de datos.

Nivel de visión

Este es el nivel más alto, en el cual se describe solamente una parte de la base de datos.



1.4. Modelo de datos

Para describir la estructura de una base de datos es necesario definir el concepto de modelo de datos. Éste es un grupo de herramientas conceptuales para describir los datos, sus relaciones, su semántica y sus limitantes. Se pueden dividir en tres grupos: los modelos lógicos basados en objetos y en registros, y los modelos físicos de datos.

1.4.1. Modelos lógicos basados en objetos

Los modelos lógicos basados en objetos se utilizan para describir los datos en los niveles conceptual y de visión. Se caracterizan por el hecho de que permiten una

estructuración bastante flexible y hacen posible especificar claramente las limitantes de los datos.

Algunos de los más conocidos son:

- El modelo entidad-relación
- El modelo binario
- El modelo semántico de datos
- El modelo infológico

El modelo de datos entidad-relación (E-R) que es el que nosotros utilizaremos, se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos llamados entidades, y de las relaciones entre estos objetos. Una entidad es un objeto que existe y puede distinguirse de otros. La distinción se logra asociando a cada entidad un conjunto de atributos que describen al objeto. También contempla la cardinalidad de mapeo, que expresa el número de entidades con las que puede asociarse otra entidad por medio de un conjunto de relaciones.

La estructura lógica general de una base de datos puede expresarse gráficamente por medio de un diagrama E-R que consta de los siguiente componentes:

- Rectangulos, que representan conjuntos de entidades
- Elipses, que representan atributos
- Rombos, que representan relaciones entre conjuntos de entidades
- Líneas, que conectan los atributos a los conjuntos de entidades y los conjuntos de entidades a las relaciones

1.4.2. Modelos lógicos basados en registros

Los modelos lógicos basados en registros se utilizan para describir los datos en los niveles conceptual y de visión. A diferencia de los modelos de datos basados en objetos, estos modelos sirven para especificar tanto la estructura lógica general de la base de datos como una descripción en un nivel más alto de la implantación. Los tres modelos con más amplia aceptación son:

Modelo relacional

Los datos y las relaciones entre los datos se representan por medio de una serie de tablas, cada una de las cuales tiene varias columnas con nombre únicos

Modelo de red

Los datos en el modelo de red se representan por medio de conjuntos de registros (en el sentido que tiene la palabra en pascal) y las relaciones entre los datos se representan con ligas, que pueden considerarse como apuntadores. Los registros de la base de datos se organizan en forma de conjuntos de gráficas arbitrarias

Modelo jerárquico

Es similar al modelo de red en cuanto a que los datos y las relaciones entre los datos se representan por medio de registros y ligas, respectivamente. El modelo jerárquico difiere del de red en que los registros están organizados como conjuntos de árboles en vez de gráficas arbitrarias.

1.4.3. Modelos físicos de los datos

Los modelos físicos de los datos sirven para describir los datos en el nivel más bajo.

Algunos de estos modelos más conocidos son:

- El modelo unificador
- La memoria de cuadros

1.5. Instancias y esquemas

Las bases de datos cambian con el tiempo al insertarse infomación en la base de datos y eliminarse de ella. El conjunto de información almacenada en la base de datos en cierto momento se denomina una *instancia* de la base de datos. El diseño general de la base de datos se llama *esquema* de la base de datos. Los esquemas se alteran muy raras veces, o nunca.

1.6. Independencia de los datos

Hemos visto que existen tres niveles de abstracción en los que puede verse la base de datos. La capacidad de modificar una definición de esquema en un nivel sin afectar la definición del esquema en el nivel inmediato superior se denomina *independencia de los datos*. Existen dos niveles de tal independencia:

- Independencia física: modificar esquema físico sin escribir de nuevo los programas
- Independencia lógica: modificar esquema conceptual sin obligar a escribir de nuevo los programas

1.7. Lenguaje de definición de datos

Un esquema de base de datos se especifica por medio de una serie de definiciones que se expresan en un lenguaje especial llamado *lenguaje de definición de datos* (DDL: data definition language). El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacenan en un archivo especial llamado *diccionario de datos* (un diccionario de datos es un archivo que contiene metadatos, es decir, datos acerca de los datos, el cual es consultado antes de leer o modificar datos reales en el sistema de base de datos).

Ejemplo de lenguaje de definición de datos (DDL)

CREATE TABLE Pedido (Pedido LONG, Cliente LONG, Vendedor TEXT(6), Fecha DATE, Totcost SINGLE)

1.8. Lenguaje de manejo de datos

Un lenguaje de manejo de datos (DML: data manipulation language) permite a los usuarios manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado. Un lenguaje de manejo de datos incorpora las siguientes tipos de manipulación de datos:

- La recuperación de información almacenada en la base de datos
- La inserción de información nueva en la base de datos
- La eliminación de información de la base de datos

La parte de un DML que implica la recuperación de información se conoce como *lenguaje de consultas*, donde una consulta es una proposición que solicita la recuperación de información.

Ejemplo de lenguaje de manejo de datos (DML) y particularmente lenguaje de consulta

SELECT Ventas.*, Artículos.Descripción, Artículos.Minorista FROM Ventas, Artículos WHERE Ventas.Artículo = Artículos.Artículo

1.9. Manejador de base de datos

Un manejador de base de datos es un módulo de programa que constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas hechas el sistema. El manejador de base de datos es responsable de las siguiente tareas:

- Interacción con el manejador de archivos

Traduce las diferentes proposiciones en DML a comandos de sistema de archivos de bajo nivel.

- Implantación de la integridad

Valores de datos que deben satisfacer ciertos tipos de limitantes de consistencia

- Puesta en práctica de la seguridad

No todos los usuarios tienen acceso a todo el contenido de la base de datos

- Respaldo y recuperación

Recuperación de fallos ante caidas del sistema u otros fallos.

- Control de concurrencia

Cuando varios usuarios actualizan la base de datos concurrentemente

1.10. Administrador de base de datos

Una de las razones principales para contar con sistemas de manejo de base de datos es tener un control centralizado tanto de los datos como de los programas que tienen acceso a ellos. La persona que tiene este control centralizado sobre el sistema es el *administrador de base de datos*. Sus funciones son entre otras, las siguiente:

- Definición de esquema
- Definición de la estructura de almacenamiento y del método de acceso
- Modificación del esquema y de la organización física
- Concesión de autorización para el acceso a los datos
- Especificación de las limitantes de integridad

1.11. Usuarios de la base de datos

Uno de los objetivos primordiales de la base de datos es crear un ambiente para la recuperación de la información y para almacenar información nueva en la base de datos. Existen tres tipos diferentes de usuarios de un sistema de base de datos:

- Programadores de aplicaciones: profesionales de la computación

- Usuarios casuales: usuarios complejos que interactúan con el sistema sin escribir programas.
- Usuarios ingenuos: usuarios poco complejos que interactuan con el sistema llamando alguno de los programas de aplicaciones permanentes escritos previamente.

1.12. Estructura general del sistema

- El manejador de archivos: encargado de asignar espacio en el disco y de las estructuras de datos que se van a emplear para representar la información almacenada en el disco
- El manejador de base de datos: constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programa de aplicaciones y las consultas que se hacen al sistema
- El procesador de consultas: traduce las proposiciones en lenguaje de consulta a instrucciones de bajo nivel que puede entender el manejador de la base de datos. Además , procesador de consultas trata de convertir la solicitud del usuario a una forma equivalente pero más eficiente, encontrando una estrategia adecuada para ejecutar la consulta.
- El precompilador de DML: que convierte las proposiciones en DML incrustadas en un programa de aplicaciones en llamadas normales a procedimientos en el lenguaje huésped. El precompilador debe interactuar con el procesador de consultas para generar el código apropiado
- El compilador de DDL: que convierte las proposiciones DDL en un conjunto de tablas que contienen metadatos. Tales tablas se almacenan después en el diccionario de datos.

Además se requieren varias estructuras de datos como parte de la implantación del sistema físico, que son:

- Archivos de datos
- Diccionario de datos
- Índices

Véase representación gráfica (Figura 1.6. Página 17)

UNIDAD DIDÁCTICA Nº 2

MODELO ENTIDAD-RELACIÓN

2.1. Introducción

Un modelo de datos entidad-relación (E-R) se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos llamados *entidades* y de *relaciones* entre estos objetos. Se desarrolló para facilitar el diseño de bases de datos permitiendo especificar un *esquema empresarial*. Este esquema representa la estructura lógica general de la base de datos.

Dicho modelo fue creado por Chen en los años 1975-76, y pretende ser un modelo muy lógico, muy conceptual, poco físico, poco informático, y cercano al usuario.

2.2. Entidades y conjunto de entidades

Una *entidad* es un objeto que existe y puede distinguirse de otros objetos. Una entidad puede ser concreta, como por ejemplo, una persona o un libro, o abstracta, como un día festivo o un concepto.

Un *conjunto de entidades* es un grupo de entidades del mismo tipo.

Una entidad está representada por un conjunto de *atributos*. Para cada atributo existe un rango de valores permitidos, llamados *dominio* del atributo.

Así, cada entidad se describe por medio de un conjunto de parejas (atributo, valor del dato), una pareja para cada atributo del conjunto de entidades. Por tanto, una entidad persona determinada se describe por medio del conjunto [(nombre, Harris), (dni, 44.897.987), (fecha de nacimiento, 12/01/60)], lo cual quiere decir que la entidad describe a una persona llamada Harris con dni 44.897.987, y que nacio el dia 12/01/60.

Haciendo analogía con los lenguajes de programación, el concepto de conjunto de entidades corresponde a la idea de definición de tipo en un lenguaje de programación. Una variable de un tipo dado tiene un valor determinado en un instante dado. Así, una variable con datos en los lenguajes de programación corresponde al concepto de entidad en el modelo E-R.

2.3. Relaciones y conjunto de relaciones

Una *relación* es una asociación entre varias entidades.

Un *conjunto de relaciones* es un grupo de relaciones del mismo tipo.

La mayor parte de las relaciones en un sistema de base de datos son binarias, pero en ocasiones existen conjuntos de relaciones que incluyen a más de dos conjuntos de entidades (a través de una relación saltamos a otra relación con otra tercera entidad).

La función que tiene una entidad en una relación se denomina *papel*. Los papeles suelen ser implícitos y, por lo general, no se especifican. Sin embargo, son útiles si se necesita aclarar el significado de una relación. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos. Por ejemplo, el conjunto de relaciones *trabaja-con* podría modelarse con parejas ordenadas de entidades *empleado*. El primer empleado de la pareja asume el papel de gerente, mientras que el segundo toma el papel de trabajador.

Una relación también puede tener atributos descriptivos (son muy importantes).

2.4. Limitantes de mapeo

Un esquema E-R empresarial puede definir ciertas limitantes con las que deben cumplir los datos contenidos en la base de datos. Una limitante importante es la de las *cardinalidades de mapeo* que expresan el número de entidades con las que puede asociarse otra entidad mediante una relación.

Las cardinalidades de mapeo son más útiles al describir conjuntos binarios de relaciones, aunque en ocasiones contribuyen a la descripción de conjuntos de relaciones que implican mas de dos conjuntos de entidades

Para un conjunto binario de relaciones R entre los conjuntos de entidades A y B, la cardinalidad de mapeo debe ser una de las siguiente:

- Una a Una. Una entidad en A está asociada únicamente con una entidad en B, y una entidad en B está asociada sólo con una entidad en A
- Una a muchas. Una entidad en A está asociada con cualquier número de entidades en B, pero una entidad en B puede asociarse únicamente con una entidad en A
- Muchas a una. Una entidad en A está vinculada únicamente con una entidad en B, pero una entidad en B está relacionada con cualquier número de entidades en A
- Muchas a muchas. Una entidad en A está asociada con cualquier número de entidades en B, y una entidad en B está vinculada con cualquier número de entidades en A.

Las dependencias de existencia constituyen otra clase importante de limitantes. Específicamente, si la existencia de la entidad *x* depende de la existencia de la entidad *y*, entonces se dice que *x* es *dependiente por existencia* de *y*. Funcionalmente, esto quiere decir que si se elimina *y*, también se eliminará *x*. Se dice que la entidad *y* es una *entidad dominante* y que *x* es una *entidad subordinada*.

2.5. Llaves primarias

Una tarea muy importante dentro de la modelación de bases de datos consiste en especificar cómo se van a distinguir las entidades y las relaciones.

Para hacer estas distinciones, se asigna una *superllave* a cada conjunto de entidades. La superllave es un conjunto de uno o más atributos que, juntos, permiten identificar en forma única a una entidad dentro del conjunto de entidades. Por ejemplo, el atributo *dni* del conjunto de entidades *persona* es suficiente para distinguir a una entidad *persona* de otra. Por tanto, *dni* es una superllave. De manera similar, la combinación de *nombre* y *dni* es una superllave para el conjunto de entidades *persona*. El atributo *nombre* de *persona* no es una superllave, ya que es posible que varias personas tengan el mismo nombre.

El concepto de superllave no es suficiente para el modelado de la base de datos, ya que –como se apreció- una superllave puede incluir atributos ajenos. Si *K* es una superllave, entonces también lo será cualquier superconjunto de *K*. Muchas veces lo que se busca es la superllave más pequeña posible. Es decir, se buscan superllaves para las cuales ningún subconjunto propio es una superllave. Estas superllaves mínimas se denominan

llaves candidato. Por ejemplo, una combinación de *nombre* y *calle* podría ser suficiente para distinguir a cada uno de los miembros del conjunto de entidades *persona*. Así pues, tanto [*dni*] como [*nombre,calle*] son *llaves candidato*.

Se utilizará el término *llave primaria* para referirse a la llave candidato que elija el diseñador de la base de datos como la forma principal de identificar a las entidades dentro de un conjunto de éstas.

Es posible que un conjunto de entidades no tenga suficientes atributos para formar una llave primaria. Por ejemplo, el conjunto de entidades *transacción* tiene tres atributos: *número-transacción*, *fecha* e *importe*. Aunque cada entidad *transacción* es distinta, dos transacciones hechas en cuentas diferentes pueden tener el mismo número de transacción. Así, el conjunto de entidades no cuenta con una llave primaria. Una entidad de un conjunto de ese tipo se denomina *entidad débil*. Una entidad que cuenta con una llave primaria recibe el nombre de *entidad fuerte*.

El concepto de entidades fuertes y débiles está relacionado con el de "dependencia por existencia". Una entidad fuerte es, por definición, dominante, mientras que una entidad débil es subordinada.

Un conjunto de entidades débiles no cuenta con una llave primaria. Sin embargo, es preciso tener una forma de distinguir entre todas esas entidades aquellas que dependen de una entidad fuerte determinada. El *discriminador* de un conjunto de entidades débiles es un conjunto de atributos que permite hacer esta distinción. Por ejemplo, el discriminador del conjunto de entidades débiles *transacción* es el atributo *número-transacción*, ya que para cada cuenta estos números de transacción identifican en forma única cada una de las transacciones.

La llave primaria de un conjunto de entidades débiles está formada por la llave primaria de la entidad fuerte de la que dependen su existencia y su discriminador. En el caso de conjunto de entidades *transacción*, su llave primaria es [*número-cuenta*, *número-transacción*], donde *número-cuenta* identifica a la entidad dominante de una *transacción*, y *número-transacción* distingue a las entidades *transacción* dentro de la misma cuenta.

Los conjuntos de relaciones también tienen llaves primarias. Sus llaves primarias se forman tomando todos los atributos que constituyen las llaves primarias de los conjuntos de entidades que definen al conjunto de relaciones.

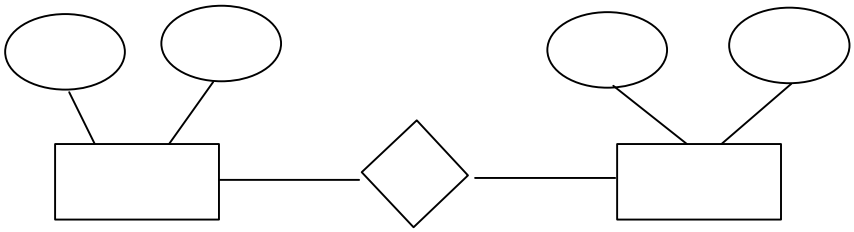
2.6. Diagrama entidad-relación

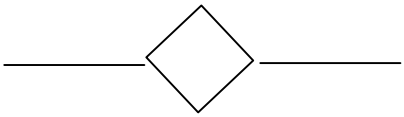
La estructura lógica general de una base de datos puede expresarse en forma gráfica por medio de un *diagrama E-R* que se integra con los siguientes componentes:

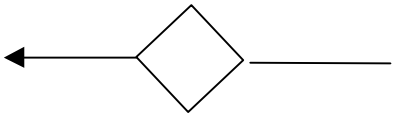
- Rectángulos, que representan conjuntos de entidades
- Elipses, que representan atributos
- Rombos, que representan conjuntos de relaciones
- Líneas, que conectan los atributos a los conjuntos de entidades, y los conjuntos de entidades a los conjuntos de relaciones.

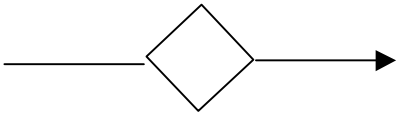
Cada componente se etiqueta con su nombre correspondiente.

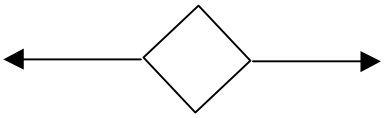
Para distinguir las cardinalidades de mapeo:



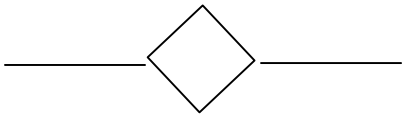
Relación muchas a muchas 

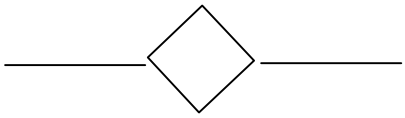
Relación una a muchas 

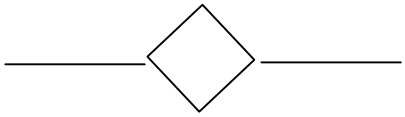
Relación muchas a una 

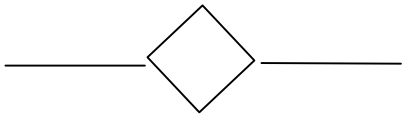
Relación una a una 

O bien utilizar la simbología siguiente:

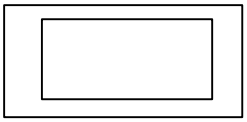
Relación muchas a muchas N  M

Relación una a muchas 1  N

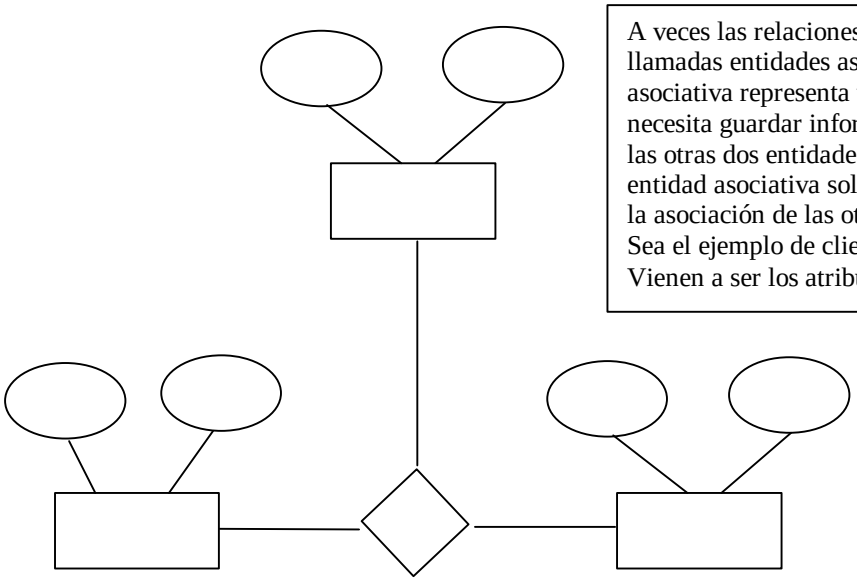
Relación muchas a una N  1

Relación una a una 1  1

Un conjunto de entidades débiles se representan por medio de un rectangulo dentro de otro

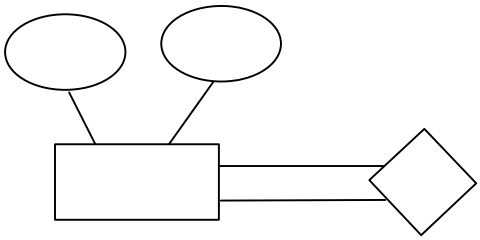


Las relaciones no binarias se representan de la siguiente forma



A veces las relaciones no binarias, provocan las llamadas entidades asociativas. La entidad asociativa representa una relación la cual necesita guardar información propia, y asi como las otras dos entidades existen por si solas, la entidad asociativa solo existe como resultado de la asociación de las otras dos.
Sea el ejemplo de cliente, articulo y compra. Vienen a ser los atributos de relación.

Cuando los conjuntos de entidades de una relación no son distintos



2.7. Interpretación semántica

El modelo E-R es un modelo altamente semántico. No se puede hacer una lectura algorítmica del modelo, sin parar a estudiar su semántica.

Así pues, tendremos que darles nombres a las relaciones, para que describan el objetivo de las mismas, de lo contrario, pueden ser mal interpretadas o ambiguamente interpretadas.

También nos tendremos que plantear si existen algunas relaciones redundantes, que puedan ser eliminadas del modelo y no perder información del modelo de base de datos.

2.8. Generalización y especialización

La generalización y la especialización son relaciones de contención que existen entre un conjunto de entidades de alto nivel y uno o más conjunto de entidades de bajo nivel.

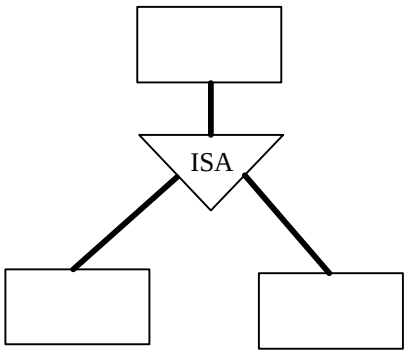
La *generalización* es el resultado de la unión de dos o más conjuntos de entidades (de bajo nivel) para producir un conjunto de entidades de más alto nivel.

La *especialización* es el resultado de tomar un subconjunto de un conjunto de entidades de alto nivel para formar un conjunto de entidades de más bajo nivel.

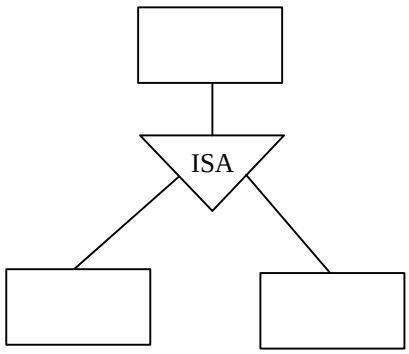
En términos de un diagrama E-R, tanto la generalización como la especialización se representan por medio de un componente *triángulo* marcado "ISA". La etiqueta "ISA" significa "es un (a)" (en inglés: "is a").

La generalización se distingue de la especialización en un diagrama E-R por el mayor grosor de las líneas que conectan el triángulo ISA y cada una de las entidades

La generalización se utiliza para hacer resaltar las semejanzas entre los tipos de entidades de bajo nivel y para ocultar sus diferencias. La especialización es el inverso. Hace resaltar la distinción entro los conjuntos de entidades de alto nivel y de bajo nivel. Los atributos son lo que los distinguen. Esto se realiza mediante la herencia de atributos. Se dice que los conjuntos de entidades de bajo nivel *heredan* los atributos de los conjuntos de entidades de alto nivel



generalización



especialización

2.9. Etapas prácticas para la construcción del modelo

Identificar las entidades

Identificar las relaciones

Identificar atributos principales de las entidades y relaciones

Identificar entidades asociativas (atributos de relación)

Identificar relaciones con entidades asociativas

Refinar el modelo (nuevos atributos, cardinalidad, …)

Definir llaves de las entidades (no de las relaciones, ni las entidades asociativas)

UNIDAD DIDÁCTICA Nº 3

MODELO RELACIONAL

3.1. Introducción

El modelo de datos relacional representa la base de datos como un conjunto de tablas. Aunque las tablas son un concepto simple e intuitivo, existe una correspondencia directa entre el concepto de una tabla y el concepto matemático de una relación.

3.2. Estructura de las bases de datos relacionales

Una base de datos relacional consiste en un conjunto de *tablas*, que tienen asignado un nombre único.

Una columna (visualmente una fila) de una tabla representa una *relación* entre un conjunto de valores. Puesto que una tabla es un conjunto de estas relaciones, existe una correspondencia entre el concepto de *tabla* y el concepto matemático de *relación*, del cual recibe su nombre el modelo de datos relacional.

Por ejemplo sea la tabla *depósito*, la formada por los atributos *nombre-sucursal*, *número-cuenta*, *nombre-cuentahabiente*, *saldo*.

La tabla *depósito* tiene cuatro atributos, y para cada atributo, existe un conjunto de valores permitidos, llamado *dominio* de ese atributo. Por ejemplo, para el atributo *nombre-sucursal*, el dominio sería el conjunto de todos los nombres de las sucursales. Sea D_1 este conjunto y sea D_2 el conjunto de todos los números de cuenta, D_3 el conjunto de todos los nombres de los cuentahabientes y D_4 el conjunto de todos los saldos. Cada una de las columnas (visualmente filas) de *depósito* debe componerse de una tupla de 4 (V_1, V_2, V_3, V_4) donde V_1 es un nombre de sucursal (es decir, V_1 está en el dominio D_1); V_2 es un número de cuenta (es decir, V_2 está en el dominio D_2); V_3 es un nombre de cuentahabiente (es decir, V_3 está en el dominio D_3), y V_4 es un saldo (es decir, V_4 está en el dominio D_4). En general, *depósito* va a contener únicamente un subconjunto del conjunto de todas las columnas posibles. Por tanto, *depósito* es un subconjunto de:

En general, una tabla de *n* columnas debe ser un subconjunto de:

Los matemáticos definen a una *relación* como un subconjunto de un producto cartesiano de un listado de dominios. Esto corresponde casi exactamente a la definición de tabla dada aquí.

Debido a que las tablas son básicamente relaciones, se utilizan los términos matemáticos *relación* y *tupla* en vez de los términos *tabla* y *columna* (fila visualmente).

Cuando se habla de una base de datos, debe diferenciarse entre el *esquema de la base de datos*; es decir, el diseño lógico de la base de datos, y una *instancia de la base de datos*, que se constituye con la información contenida, en la base de datos en cierto momento.

Se adopta la convención de usar en minúsculas para las relaciones y la primera letra mayúsculas para los esquemas de relaciones.

En general, el esquema de una relación es una lista de atributos y sus correspondiente dominios.

Esquema-depósito = (nombre-sucursal, número-cuenta, nombre-cuentahabiente,saldo)

En general no se necesita una definición precisa del dominio de cada atributo, sin embargo se puede seguir la notación siguiente:

(nombre-sucursal: cadena, número-cuenta: entero, nombre-cuentahabiente:cadena, saldo:entero)

3.3. Resumen

3.3.1. Terminología

Figura representativa de terminología formal, con terminología informal

Termino relacional formal	**Equivalente informal**
Relación	Tabla, fichero
Tupla	Fila, registro
Cardinalidad	Número de filas
Grado	Número de columnas
Llave primaria	Identificador
Dominio	Conjunto de valores válidos de una columna
Atributo	Columna, campo

La terminología utilizada para definir modelos de datos, es la siguiente:

Dominios.

Un dominio es un conjunto de *valores escalares*, todos del mismo tipo. Un valor escalar es la menor unidad semántica de información. Es atómico, porque si se descompone pierde su significado.

Un dominio evita las redundancias en la definición de atributos

Relaciones

Una relación se compone de dos partes: una *cabecera* y un *cuerpo*

La *cabecera* está formada por un conjunto fijado de parejas atributo-dominio

El número de estas parejas se denomina *grado* de la relación

El *cuerpo* está formado por un conjunto de tuplas, el cual varia con el tiempo. Cada *tupla* está formada por un conjunto de parejas atributo-valor.

Llaves o identificadores

Superllave: uno o más atributos que juntos permiten identificar de forma única una entidad.

Llave candidato: la superllave mínima

Llave primaria: la llave candidato elegida

Llave alternativa: las restantes candidatas que no son la primaria

Llave extranjera: conjunto de atributos de una relación que son a la vez llave primaria de otra relación (cuando una relación del modelo E-R se convierte en relación del modelo relacional, coge las llaves primarias de las entidades que relaciona)

3.3.2. Propiedades de las relaciones

- No existen tuplas repetidas. El cuerpo de una relación es un conjunto matemático, y en matemáticas los conjuntos por definición no incluyen elementos duplicados.
- Las tuplas no están ordenadas (de arriba a bajo). El cuerpo de una relación es un conjunto matemático, y en matemáticas los conjuntos por definición no están ordenados. El orden de las tuplas no es relevante. No se puede hablar de la tupla 5, ni de la tupla 6, ni de la tupla siguiente. No existe el concepto de dirección por posición.
- Los atributos no están ordenados (de izquierda a derecha). La cabecera de una relación se define también como un conjunto matemático. El orden de los atributos no es relevante. No existe un primer atributo, ni el segundo, ni el tercero.
- Todos los valores de los atributos simples son atómicos. Es una consecuencia del hecho de que todos los dominios contienen valores atómicos. Si existe algun atributo compuesto, este se puede descomponer como concatenación de atributos simples. Una relación que cumple esta condición se dice que está normalizada (1FN). La normalización de las relaciones nos llevará a modelos más simples desde el punto de vista matemático, y por lo tanto será más sencillo operar sobre ellos.

UNIDAD DIDÁCTICA Nº 4

TRADUCCIÓN DEL MODELO E-R AL MODELO RELACIONAL

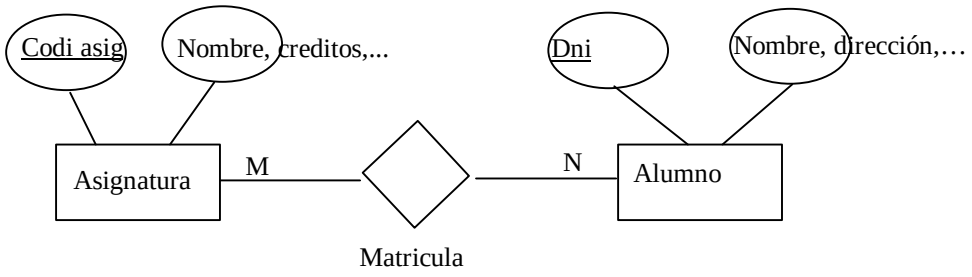
4.1. Introducción

Esquema de equivalencias conceptuales entre el modelo E-R y el modelo relacional

Modelo E-R	**Modelo Relacional**
Entidad	Relación
Relación	Relación
Entidad asociativa	Relación
Atributo	Atributo

4.2. Caso de una relación M:N simple

Supongamos que tan solo nos interesa saber un alumno que asignaturas hace



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (codi asignatura, nombre, créditos)

Alumno = (dni, nombre, dirección, fecha de nacimiento)

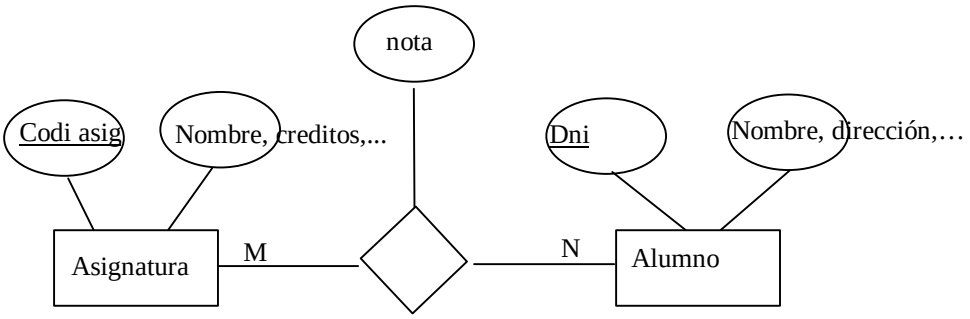
La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Matricula = (codi asignatura, dni)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las dos llaves heredadas.

4.3. Caso de una relación M:N con atributos descriptivos o entidades asociativas o atributos de relación

Supongamos que además de saber un alumno que asignaturas hace, necesitamos también saber la nota que ha sacado en cada asignatura



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (codi asignatura, nombre, créditos)

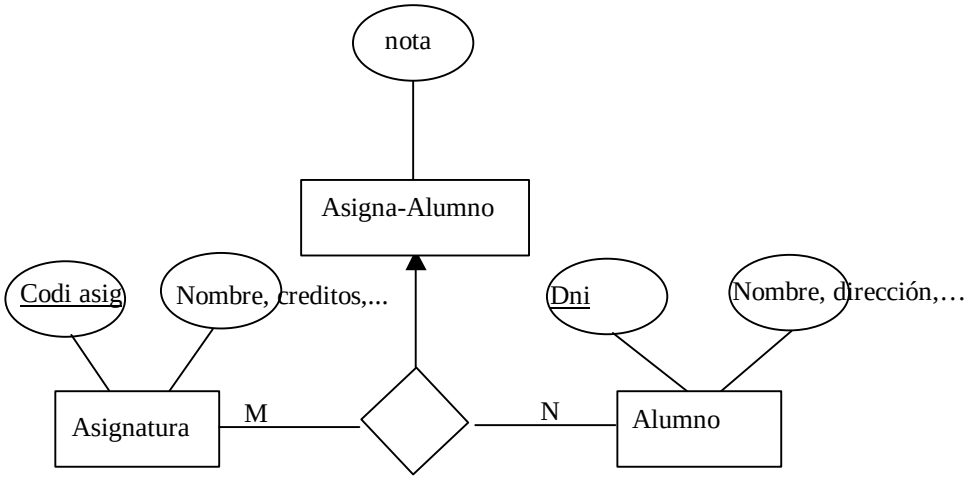
Alumno = (dni, nombre, dirección, fecha de nacimiento)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona, asi como los atributos descriptivos de la propia relación (en este caso el atributo nota)

Matricula = (codi asignatura, dni, nota)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las dos llaves heredadas.

NOTA: el modelo E-R anterior, podría representarse de la forma siguiente, donde Asigna-Alumno viene a representar una entidad asociativa (no puede existir por ella sola, tiene que estar ligada a al menos a otras dos entidades por medio de una relación)



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (codi asignatura, nombre, créditos)

Alumno = (dni, nombre, dirección, fecha de nacimiento)

Asigna-Alumno = (nota, ?)

(La relación del modelo E-R **no** se convierte en relación del modelo relacional).

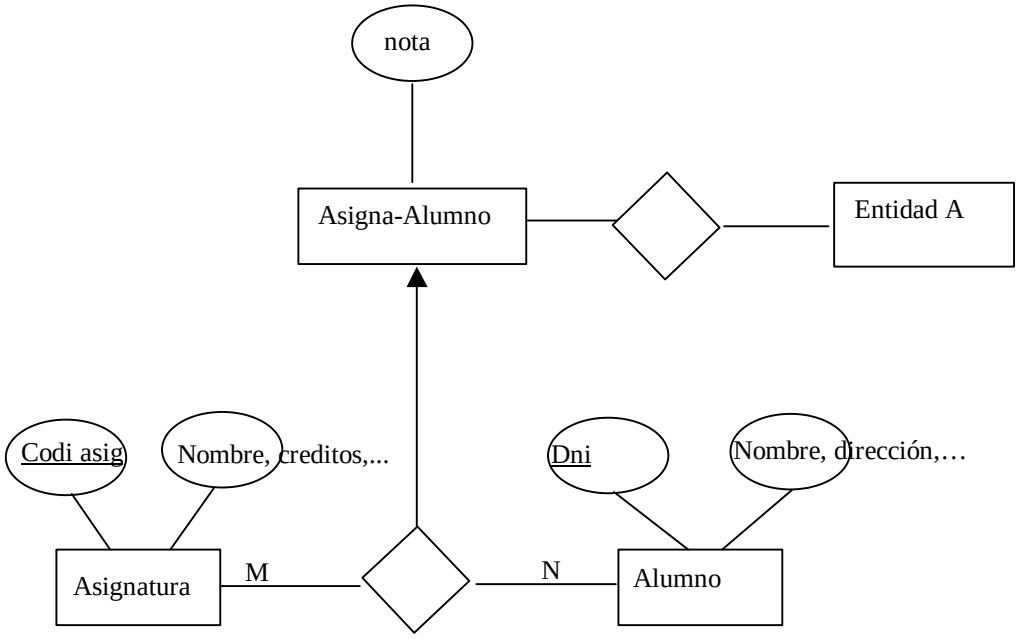
La entidad asociativa (ahora una relación) hereda como atributos las llaves primarias de las entidades que relaciona.

Asigna-Alumno = (codi asignatura, dni, nota)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las dos llaves heredadas.

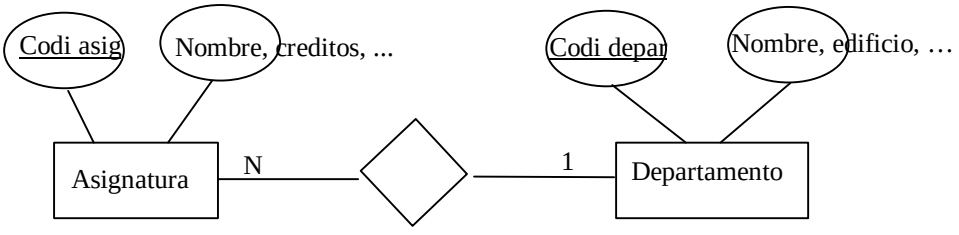
Asigna-Alumno = (<u>codi asignatura, dni</u>, nota)

Este tipo de representación presenta la ventaja, de que si la entidad asociativa va a relacionarse con otras entidades, visualmente es más claro. Véase a continuación.



4.4. Caso de una relación 1:N

Supongamos que tan solo nos interesa saber una asignatura a que departamento pertenece



Asigna-Depar

Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (<u>codi asignatura</u>, nombre, créditos)

Departamento = (<u>codi depar</u>, nombre, edificio, nombre del jefe)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Asigna-Depar = (codi asignatura, codi depar)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la llave heredada de la relación N

Asigna-Depar = (codi asignatura, codi depar)

Optimización habitual

Normalmente no se crea la relación. Es decir, la relación del modelo E-R **no** se convierte en relación del modelo relacional.

Lo que se hace es añadir la llave primaria de la relación 1 como llave extrajera en la relación N.

El resultado es el siguiente:

Asignatura = (codi asignatura, nombre, créditos, codi depar)

Departamento = (codi depar, nombre, edificio, nombre del jefe)

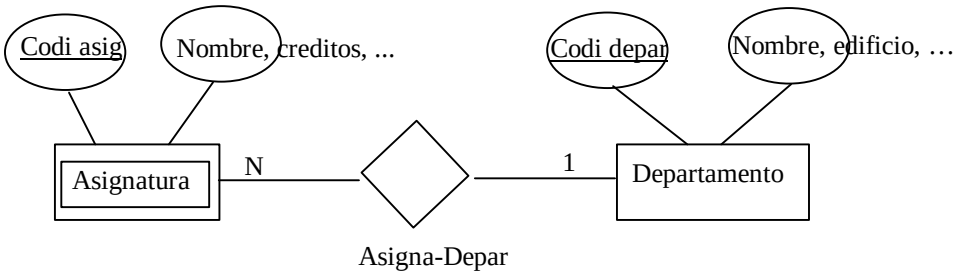
Problemas

En el caso de que la relación sea 0,1:N (no hay dependencia de existencia), provoca valores NULL en la llave extranjera

En el caso de que la relación 1:N del modelo E-R, tenga atributos descriptivos (atributos de relación o lo que es lo mismo entidades asociativas) también han de pasar a formar parte de la relación N. (No es normal que una relación 1:N del modelo E-R tenga atributos, pero puede pasar).

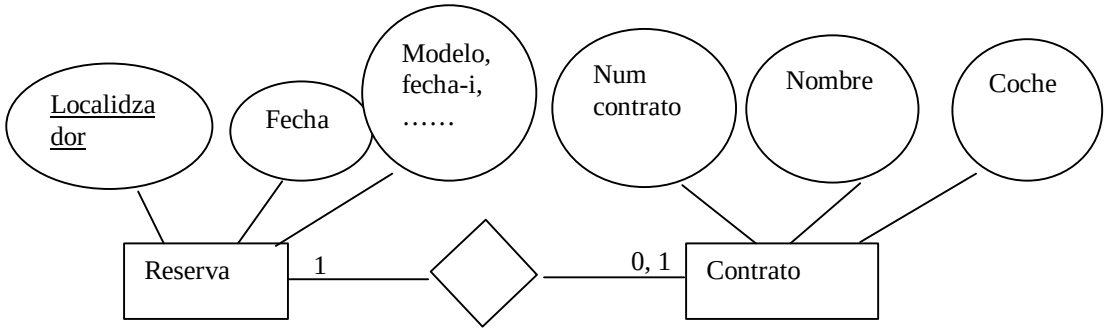
Dependencia existencial

En el caso de que sea 1:N, como ha sido el caso anterior, casi siempre tendremos una dependencia existencial (no puede existir una asignatura que no sea de un departamento, aunque si pueden existir departamentos sin asignaturas, y si desaparece un departamento, también desaparecen sus asignaturas). En estos casos, deberíamos marcar la existencia de tal dependencia existencial remarcando con un subcuadro la entidad debil.



4.5. Caso de una relación 1:1

Sea el caso en que podemos recibir una reserva de un coche, el cual puede tener un contrato o no en función del cliente. Pero además para todo contrato si que habrá una reserva.



Reserv-Contr

Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Reserva = (localizador, fecha, modelo, fecha-i, fecha-f)

Contrato = (num contrato, nombre, coche)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Reserv-Contr = (localizador, num contrato)

Para encontrar una llave primaria de la nueva relación, se usa normalmente cualquiera de las llaves heredadas

Reserv-Contr = (localizador, num contrato)

Optimización habitual

Normalmente no se crea la relación. Es decir, la relación del modelo E-R **no** se convierte en relación del modelo relacional.

Lo que se hace es añadir la llave primaria de **una** de las dos relaciones a la otra relación.

Reserva = (localizador, fecha, modelo, fecha-i, fecha-f)

Contrato = (num contrato, nombre, coche, localizador)

Además se tendrá que establecer la restricción de que "localizador" sea indexado sin duplicados (UNIQUE), para garantizar que la relación sea 1:1, ya que sino estaríamos ante una 1:N.

Problemas

En el caso de que la relación sea 0,1:1, mejor poner la llave primaria de la relación "1" sobre la "0, 1", para evitar valores NULL (en la relación "1").

4.6. Caso de una relación entre múltiples entidades

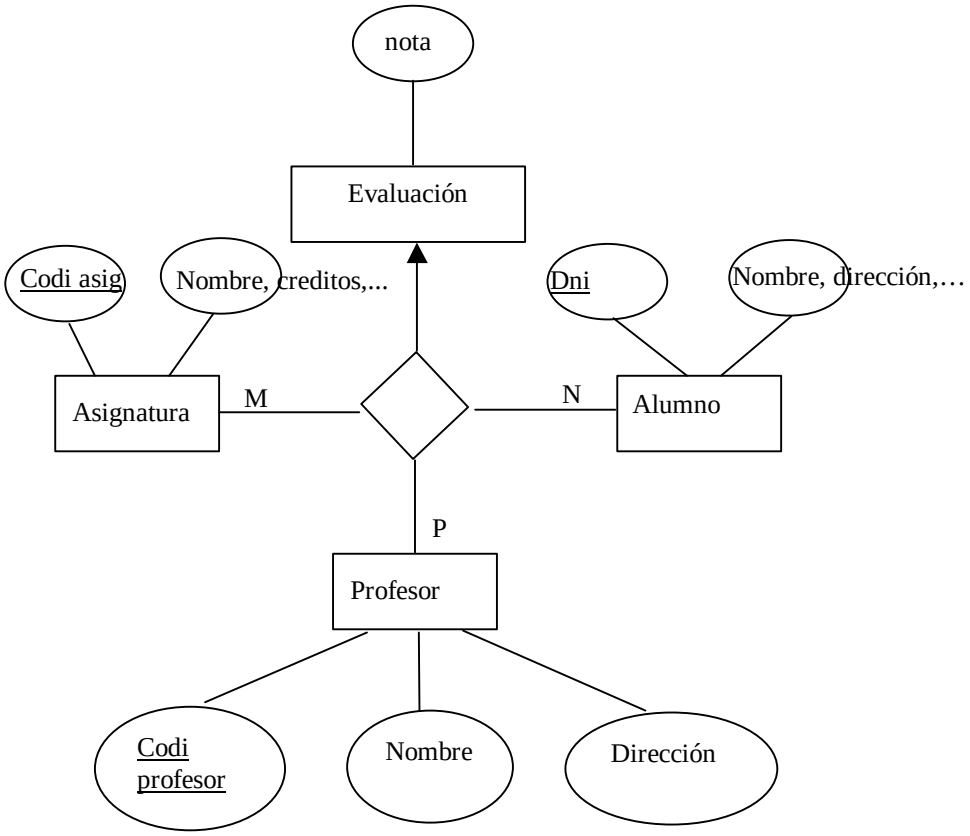
En este ejemplo se ha utilizado en el modelo E-R la simbología de entidades asociativas.

En este ejemplo de relación entre múltiples entidades, la relación y todas las entidades conectadas se han de interpretar como una unidad. La relación se tiene que estudiar desde la perspectiva de cada uno de los objetos participantes.

Es el conjunto de todas estas perspectivas la que describe completamente la relación.

En el caso del ejemplo, es el siguiente:

- El alumno tiene una nota para cada asignatura y profesor.
- El profesor pone una nota a cada alumno para cada asignatura
- De una asignatura se tiene una nota para cada alumno y profesor



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (codi asignatura, nombre, créditos)

Alumno = (dni, nombre, dirección, fecha de nacimiento)

Profesor = (codi profesor, nombre, dirección)

Evaluación = (nota, ?)

(La relación del modelo E-R **no** se convierte en relación del modelo relacional).

La entidad asociativa (ahora una relación) hereda como atributos las llaves primarias de las entidades que relaciona.

Evaluación = (nota, codi asignatura, dni, codi profesor)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de todas las llaves heredadas.

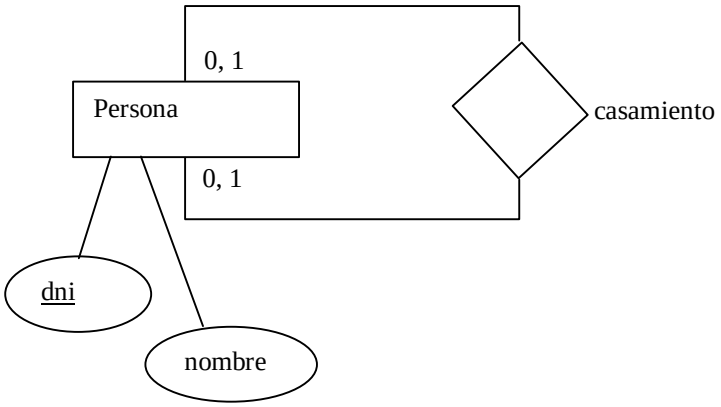
Evaluación = (codi asignatura, dni, codi profesor, nota)

4.7. Caso de una relación reflexiva

Se han divido las relaciones reflexivas en tres tipos de casos:

Caso 1:1 Caso 1:N Caso N:M

Caso 1:1



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Persona = (dni, nombre)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Casamiento = (dni esposo, dni esposa)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las llaves heredadas

Casamiento = (dni esposo, dni esposa)

Posible optimización

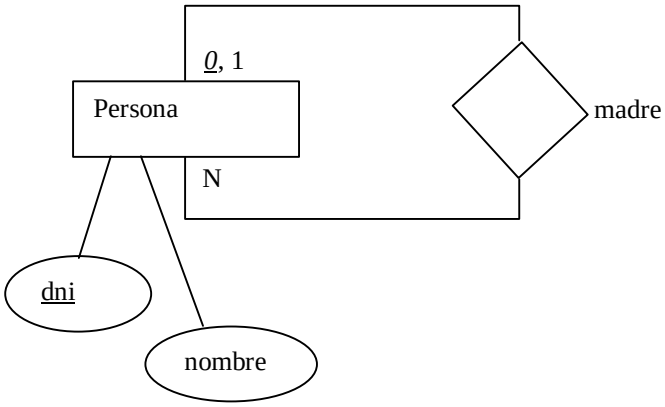
Lo que se hace es añadir un atributo a la entidad que se convirtió en relación, y la relación del modelo E-R desaparece, quedando únicamente (viene a ser lo mismo que se hacia con las relaciones 1:1 no reflexivas: *añadir la llave primaria de **una** de las dos relaciones a la otra relación*)

Persona = (dni, nombre, dni cónyuge)

Problemas

En el caso de que la relación sea 0,1:0,1 podríamdos tener valores NULL, cuando se diese el caso de que una persona no tenga cónyuge. Si la relación fuese 1:1, no habría problemas porque significaría que toda persona tiene un cónyuge.

Caso 1:N



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Persona = (dni, nombre)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Madre = (dni madre, dni hijo)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las llaves heredadas

Madre = (dni madre, dni hijo)

Posible optimización

Lo que se hace es añadir un atributo a la entidad que se convirtió en relación, y la relación del modelo E-R desaparece, quedando únicamente (viene a ser lo mismo que se hacia con las relaciones 1:1 no reflexivas: *añadir la llave primaria de **una** de las dos relaciones a la otra relación*)

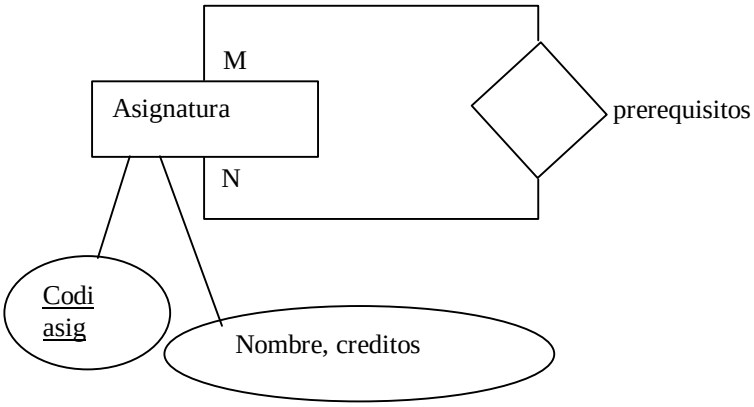
Persona = (dni, nombre, dni madre)

Problemas

En el caso de que la relación sea 0,1:N podríamdos tener valores NULL, cuando se diese el caso de que una persona no tenga madre (aunque conceptualmente es imposible, tendriamos que interpretarlo como una persona que la tenemos dada de alta, pero no sabemos quien es su madre o su madre no esta dada de alta en la relación *Persona*).

Sin embargo en la mayoría de casos es factible la optimización planteada.

Caso N:M



Toda entidad del modelo E-R se convierte en relación del modelo relacional.

Asignatura = (codi asig, nombre, creditos)

La relación del modelo E-R se convierte en relación del modelo relacional, heredando como atributos las llaves primarias de las entidades que relaciona.

Prerequisitos = (codi asig, codi asig pre)

Para encontrar una llave primaria de la nueva relación, se usa normalmente la unión de las llaves heredadas

Prerequisitos = (codi asig, codi asig pre)

UNIDAD DIDÁCTICA Nº 5

INTEGRIDAD REFERENCIAL

5.1. Introducción

Se ha comentado que el modelo relacional incluye tres aspectos fundamentales:

- La estructura
- La integridad
- La manipulación

Todo sistema relacional debe cumplir dos reglas generales de integridad, que son

- La regla de integridad de las entidades
- La regla de integridad referencial

Además de estas dos reglas generales, también existen reglas de integridad que son definidas por el usuario.

5.2. Regla de integridad de entidades

Los atributos que forman la llave primaria de una relación no pueden tomar valores nulos

Es decir, en una base de datos relacional nunca se registrará información sobre una tupla que no se pueda identificar

5.3. Regla de integridad referencial

La base de datos no puede contener valores en una llave extrajera que sean inconsistentes

Es decir, que no podemos tener valores en una llave extranjera que después resulte que no existe la tupla correspondiente con ese valor en llave primaria (a no ser que sea el valor NULL).

De esta forma, los valores validos para una llave extranjera son valores existentes en llave primaria de referencia y el valor NULL.

5.4. Reglas para las llaves extranjera

Para cada una de las llaves extranjera tenemos que plantearnos las siguientes cuestiones

- Si puede contener valores NULL
- Que hacer si se intenta eliminar la llave primaria referenciada por una llave extranjera

    No permitirlo

    Eliminar en cascada

    Asignarle un valor NULL

- Que hacer si se intenta modificar la llave primaria referenciada por una llave extranjera

  No permitirlo

  Modificar en cascada

  Asignarle un valor NULL

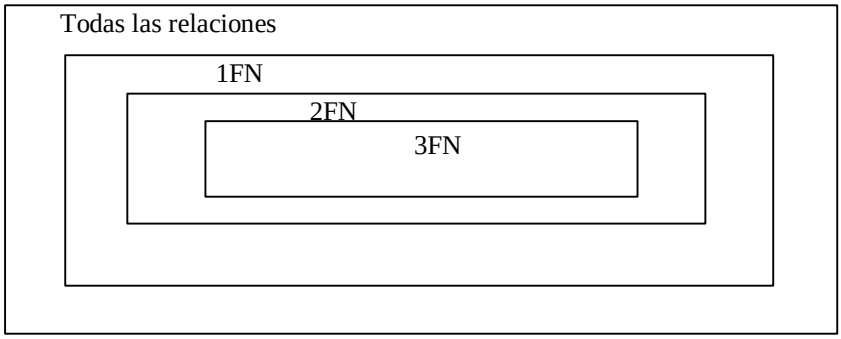
Todas estas características o reglas que definen las llaves extranjeras se establecen con el lenguaje para la definición de datos correspondiente, en el momento en que se define dicha llave.

UNIDAD DIDÁCTICA Nº 6

FORMAS NORMALES SOBRE BASES DE DATOS RELACIONALES

6.1. Introducción

Se dice que una relación está en una derterminada forma normal si satisface un cierto conjunto de restricciones



La figura tiene que interpretarse de la siguiente forma:

De todas las relaciones las hay que están en 1FN. De estas las hay que además están en 2FN. De estas las hay que están en 3FN. Así pues si una relación esta en en 3FN también está en 2FN y 3FN.

Boyce Codd definió la 1FN, 2FN y 3FN. Luego surgieron al 4FN, 5FN, etc.

6.2. El concepto de dependencia funcional

Sea R una relación, y X e Y dos atributos de R.

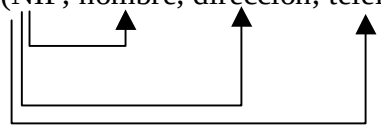
Se dice que X determina funcionalmente a Y o Y depende funcionalmente de X si y solo si:

- Para cada valor de X hay una y solo una imagen de Y (a X se le llama determinante)

Nota: X e Y pueden ser compuestos

Ejemplos

CLIENTE (NIF, nombre, dirección, teléfono)

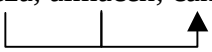


“nombre” depende funcionalmente de “NIF”

“dirección” depende funcionalmente de “NIF”

“teléfono” depende funcionalmente de “NIF”

ESTOC (pieza, almacen, cantidad)



“cantidad” depende funcionalmente de “pieza almacen”

6.3. El concepto de dependencia funcional plena

Sea R una relación, y X e Y dos atributos de R.

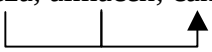
Se dice que X determina funcionalmente a Y de forma plena o Y depende funcionalmente de X de forma plena si y solo si:

- Y depende funcionalmente de X, y
- Y no depende funcionalmente de ninguno de los subconjunto propios de X

Nota: solo es aplicable si X es compuesto

Ejemplo

ESTOC (pieza, almacen, cantidad)



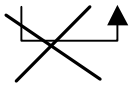
Tomando a X como “pieza almacen” y a Y como “cantidad”

Diremos que “pieza almacen” determina funcionalmente a “cantidad” de forma plena si y solo si:

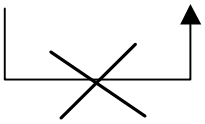
- “cantidad” depende funcionalmente de “pieza almacen”
- “cantidad” no depende funcionalmente de ninguno de los subconjuntos propios de “pieza almacen”

Como son ciertos ambos criterios, podemos decir que “pieza almacen” determina a “cantidad” de forma plena, o lo que es lo mismo, que “cantidad” depende funcionalmente de “pieza almacen” de forma plena.

ESTOC (pieza, almacen, cantidad)



ESTOC (pieza, almacen, cantidad)



6.4. Primera formal normal (1FN)

Una relación está en 1FN si y solo si:

- Todos los dominios sobre los que están definidos los atributos son simples o atómicos

Ejemplo

CLIENTE (NIF, nombre, dirección, teléfono, lista_de_facturas)

No está en 1FN porque *lista_de_facturas* no es un atributo atómico, sino que es un atributo compuesto por una iteración de *num_factura, fecha, importe*.

Esta relación, por lo tanto se descompondría en otras dos relaciones que si estaría en 1FN, que serían:

CLIENTE (NIF, nombre, dirección, teléfono)

FACTURA (num_factura, fecha, importe, NIF)

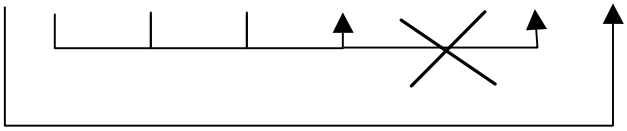
6.5. Segunda formal normal (2FN)

Una relación está en 2FN si y solo si:

- Está en 1FN
- Todo atributo que no forma parte de llave primaria depende funcionalmente de **toda** la llave primaria. Dependencia funcional plena.

Ejemplo

COMPRA (proveedor, artículo, fecha, cantidad, ciudad_proveedor)



No está en 2FN porque "ciudad_proveedor" no depende funcionalmente de toda la llave primaria, sino solamente de un subconjunto de ella.

La descomposición resultante en 2FN es la siguiente:

PROVEEDOR (proveedor, ciudad)

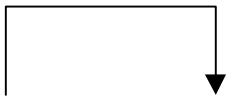
COMPRA (proveedor, artículo, fecha, cantidad)

6.6. Tercera formal normal (3FN)

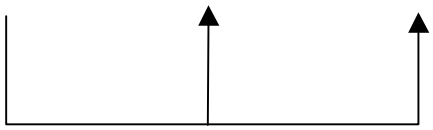
Una relación está en 3FN si y solo si:

- Está en 2FN
- Todo atributo que no forma parte de llave primaria no depende funcionalmente de ningun otro atributo que no sea llave (cada atributo no llave depende solamente de la llave)

Ejemplo



EMPLEADO (empleado, departamento, ubicación_departamento)



No está en 3FN porque "ubicación_departamento" también depende funcionalmente de "departamento", que es un atributo no llave.

La descomposición resultante en 3FN es la siguiente:

EMPLEADO (empleado, departamento)

DEPARTAMENTO (departamento, ubicación)

UNIDAD DIDÁCTICA Nº 7

TRADUCCIÓN DEL MODELO RELACIONAL DE DATOS AL MODELO FÍSICO DE DATOS

7.1. Reglas de traducción de relaciones a tablas

Regla 1ª

Toda relación del modelo relacional se convierte en una tabla del modelo físico

Regla 2ª

Cada atributo de una relación del modelo relacional se convierte en un campo de una tabla del modelo físico

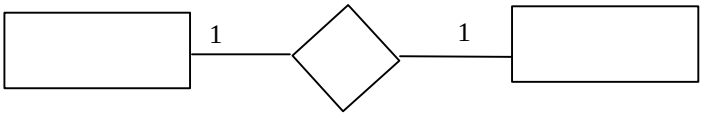
Regla 3ª

Cada dominio de los atributos de una relación del modelo relacional, definen el tipo de dato del campo de una tabla

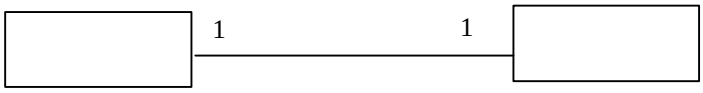
7.2. Reglas de cardinalidad

Regla 4ª: Relaciones 1:1

Si una relación 1:1 del modelo E-R se convierte en una relación del modelo relacional, se redefine la cardinalidad como 1:1 y 1:1

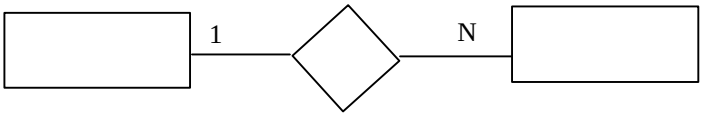


Si una relación 1:1 del modelo E-R no se convierte en una relación del modelo relacional, se redefine la cardinalidad como 1:1

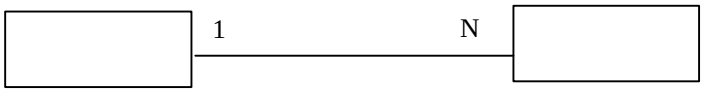


Regla 5ª: Relaciones 1:N

Si una relación 1:N del modelo E-R se convierte en una relación del modelo relacional, se redefine la cardinalidad como 1:N y 1:1

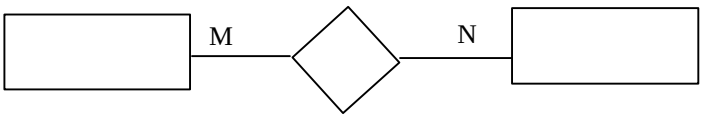


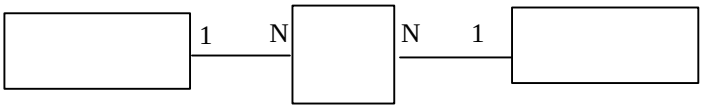
Si una relación 1:N del modelo E-R no se convierte en una relación del modelo relacional, se redefine la cardinalidad como 1:N



Regla 6ª: Relaciones M:N

Si una relación M:N del modelo E-R se convierte en una relación del modelo relacional, se redefine la cardinalidad como 1:N y N:1





UNIDAD DIDÁCTICA Nº 8

SQL ESTANDAR

8.1. Introducción

SQL (lenguaje de consulta estructurado), es un conjunto de comandos de programación especializados que habilitan al programador o usuario final para poder llevar a cabo las tareas siguientes:

- Recuperar datos de una o más tablas, de una o más bases de datos
- Manejar datos de las tablas, insertando, eliminando o actualizando registros
- Obtener resumenes de los datos
- Crear, modificar o eliminar tablas
- Crear o eliminar índice de una tabla

8.2. Partes de una instrucción SQL

- Declaración de parámetros
- Instrucción manipulante
- Declaración de las opciones

Y se organizan de la forma siguiente

[Declaración de parámetros] Instrucción manipulante [opciones]

Instrucción manipulante	Función
DELETE FROM	Elimina un registro de una tabla
INSERT TO	Agrega un grupo de registros a una tabla
SELECT	Recupera un grupo de registros
TRANSFORM	Genera una tabla resumen
UPDATE	Especifica los valores de los campos de una tabla

8.3. Instrucciones de consulta SELECT. (DML: Lenguaje de Manejo de Datos)

Seleccionar todos los campos de una tabla

SELECT * FROM Ventas

Seleccionar campos individuales de una tabla

SELECT Artículo, Cantidad FROM Ventas

Seleccionar campos individuales con nombre con espacios en blanco de una tabla

SELECT [Código artículo], Cantidad FROM Ventas

Seleccionar campos de múltiples tablas

SELECT Ventas.*, Artículos.Descripción, Artículos.Minorista
FROM Ventas, Artículos
WHERE Ventas.Artículo = Artículos.Artículo

Creación de campos calculados

Calcular el precio total para los artículos

SELECT Artículos.Minorista * Ventas.Cantidad FROM Artículos, Ventas
WHERE Ventas.Artículo = Artículos.Artículo

Crear un nombre de campo encadenando los campos Apellido y Nombre

SELECT Apellido & ‘, ‘ & Nombre FROM Clientes

Hallar la raíz cuadrada de un número

SELECT Cantidad, SQR(Cantidad) FROM Ventas

Utilizar tablas de otras bases de datos

Utilización de la base de datos CODIGOS que sigue abierta, y que tiene la tabla “Código Postal” con los campos “Ciudad”, “Provincia” y “Código Postal”

SELECT Clientes.Apellido, Clientes.Nombre, [Código Postal].Ciudad, [Código Postal].Provincia FROM Clientes, [Código Postal] IN CODIGOS
WHERE Clientes.CódigoPostal = [Código Postal].CódigoPostal

Base de datos Microsoft Jet. JetData.mdb es el nombre de la base de datos Jet que contiene la tabla Clientes

SELECT IdCliente FROM Clientes IN 'C:\Mis Documentos\JetData.mdb'
WHERE IdCliente Like 'A*'

Asignación de nombres alias a las tablas

SELECT CS.Apellido,CS.Nombre, CP.Ciudad, CP.Provincia

FROM Clientes AS CS, [Código Postal] IN CODIGOS AS CP

WHERE CS.CódigoPostal = CP.CódigoPostal

Uso de predicados ALL, DISTINCT y DISTINCTROW

Seleccionar todos los registros que cumplan los criterios especificos

SELECT ALL * FROM Clientes

Es equivalente a SELECT * FROM Clientes

Omitir los registros que contienen datos duplicados en los campos seleccionados

SELECT DISTINCT Apellidos FROM Empleados

Si la cláusula SELECT contiene más de un campo, la combinación de valores de todos los campos debe ser única para un registro concreto que se va a incluir en el resultado.

Omite los datos basados en registros duplicados completos, no sólo campos duplicados.

SELECT DISTINCTROW NombreCompañía FROM Clientes INNER JOIN Pedidos

ON Clientes. IdCliente = Pedidos.IdCliente ORDER BY NombreCompañía

Por ejemplo, la consulta anterior combina las tablas Clientes y Pedidos por el campo IdCliente. La tabla Clientes no contiene ningún campo IdCliente duplicado, pero la tabla Pedidos sí los tiene porque cada cliente puede tener varios pedidos. La instrucción SQL anterior muestra cómo puede utilizar el predicado DISTINCTROW para crear una lista de compañías que tengan al menos un pedido pero sin obtener detalles acerca de los mismos

Relaciones entre tablas

Relaciones entre tablas utilizando WHERE

```
SELECT Ventas.*, Artículos.Descripción, Artículos.Minorista
    FROM Ventas, Artículos
    WHERE Ventas.Artículo = Artículos.Artículo
```

NOTA: se genera un conjunto de registros de solo lectura. Para crear un conjunto de registros modificable, es preciso utilizar la cláusula JOIN

Relaciones entre tablas utilizando JOIN

INNER JOIN

```
SELECT CS.Apellido, CS.Nombre, VN.Apellido, VN.Nombre
    FROM Clientes AS CS, Vendedores AS VN, CS INNER JOIN VN
    ON CS.Vendedor = VN.Vendedor
```

LEFT JOIN

```
SELECT CS.Apellido, CS.Nombre, VN.Apellido, VN.Nombre
    FROM Clientes AS CS, Vendedores AS VN, CS LEFT JOIN VN
    ON CS.Vendedor = VN.Vendedor
```

RIGHT JOIN

```
SELECT CS.Apellido, CS.Nombre, VN.Apellido, VN.Nombre
    FROM Clientes AS CS, Vendedores AS VN, CS RIGHT JOIN VN
    ON CS.Vendedor = VN.Vendedor
```

Cuadro resumen de los resultados entre diferentes JOIN

Tipo de JOIN	Registros de la tabla izquierda	Registros de la tabla derecha
INNER	Solo registros con correspondencia en la tabla derecha	Solo registros con correspondencia en la tabla izquierda
LEFT	Todos los registros	Solo registros con correspondencia en la tabla izquierda
RIGHT	Solo registros con correspondencia en la tabla derecha	Todos los registros

Establecer criterios de filtrado

Filtrar registros con un valor concreto

```
SELECT * FROM Clientes WHERE Apellido = 'Perez'
```

Filtrar registros que cumplan el modelo del predicado LIKE

SELECT * FROM Clientes WHERE Apellido LIKE 'P*'

Filtrar registros que pertenecen a un conjunto de valores determinados

SELECT * FROM Clientes WHERE Provincia IN ('TO', 'CO', 'SE')

Filtrar registros cuyos valores se encuentren dentro de un intervalo

SELECT * FROM Ventas WHERE Fecha BETWEEN #08/01/94# AND #31/01/94#

Combinación de varios criterios de filtrado

```
SELECT * FROM Clientes WHERE Apellido = 'Martinez' AND Provincia
      IN ('TO', 'CO', 'SE')
```

Especificar condiciones de ordenación

SELECT * FROM Clientes ORDER BY Apellido DESC, Nombre

Utilizar funciones de agregado

```
SELECT    Min (VN.Cantidad * AR.Articulos) AS Minvns,
          Max (VN.Cantidad * AR.Articulos) AS Maxvns,
          Avg (VN.Cantidad * AR.Articulos) AS Medvns,
          Sum (VN.Cantidad * AR.Articulos) AS Totvns,
          Sum (VN.Cantidad) AS Totvol
          FROM Ventas AS VN, Articulos AS AR
          WHERE VN.Articulo = AR.Articulo
```

Crear grupos de registros

```
SELECT    VN.Vendedor, Min (VN.Cantidad * AR.Articulos) AS Minvns,
          Max (VN.Cantidad * AR.Articulos) AS Maxvns,
          Avg (VN.Cantidad * AR.Articulos) AS Medvns,
          Sum (VN.Cantidad * AR.Articulos) AS Totvns,
          Sum (VN.Cantidad) AS Totvol
          FROM Ventas AS VN, Articulos AS AR
          WHERE VN.Articulo = AR.Articulo GROUP BY VN.Vendedor
```

Creando un grupo de registros y determinando que registros seleccionados hay que visualizar, poniendo condiciones solo a los valores de campos de los registros devueltos

```
SELECT    VN.Vendedor, Min (VN.Cantidad * AR.Articulos) AS Minvns,
          Max (VN.Cantidad * AR.Articulos) AS Maxvns,
          Avg (VN.Cantidad * AR.Articulos) AS Medvns,
          Sum (VN.Cantidad * AR.Articulos) AS Totvns,
          Sum (VN.Cantidad) AS Totvol
          FROM Ventas AS VN, Articulos AS AR
          WHERE VN.Articulo = AR.Articulo GROUP BY VN.Vendedor
          HAVING Sum(VN.Cantidad * AR.Articulos) > 10.000
```

Crear una tabla a partir del resultado de una consulta con SELECT

Hasta ahora todos los resultados (conjuntos de registros) de un SELECT salian en una dynaset o snapshot que son puramente temporales. Si queremos almacenar la información del conjunto de registros de forma permanente para el uso posterior, utilizaremos el INTO.

```
SELECT Empleados.Nombre, Apellido INTO [Representantesde ventas]
    FROM Empleados
    WHERE Cargo = 'Representante de ventas'
```

```
SELECT Empleados.* INTO Empleados IN Copia.mdb FROM Empleados
```

```
SELECT Empleados.*, Salario INTO Prácticas FROM Empleados
    INNER JOIN Nómina ON Empleados.IdEmpleado = Nómina.IdEmpleado
    WHERE Cargo = 'Prácticas'
```

NOTA: el nombre de la tabla ha de ser nueva. Si ya existe se rescribe.

8.4. Consultas de acción. (DML: Lenguaje de Manejo de Datos)

Eliminar registros específicos de una tabla

```
DELETE FROM Clientes WHERE Provincia = ‘SE’
```

Agregar un grupo de registros a una tabla

Cuando no se designan columnas individuales, los nombres de columna de la tabla SELECT deben coincidir exactamente con los de la tabla INSERT INTO.

```
INSERT INTO Clientes SELECT * FROM ClientesNuevos
```

En el ejemplo siguiente se crea un registro nuevo en la tabla Empleados.

```
INSERT INTO Empleados (Nombre,Apellido, Cargo) VALUES ('Carlos', 'Sesma',
'Prácticas')
```

Modificar valores de campos específicos de una tabla

```
UPDATE Clientes SET Vendedor = ‘AGUTIERREZ’ WHERE Vendedor = ‘JPEREZ’
```

```
UPDATE Artículos SET Minorista = Minorista * 1.05
```

Consultas de Unión

Consultas de Intersección

Consultas de Diferencia

8.5. Instrucciones para la definición de datos. (DDL: Lenguaje de Definición de Datos)

Estas instrucciones permiten crear, modificar y eliminar tablas e índices de una base de datos con una sola instrucción.

Crear una tabla

```
CREATE TABLE Pedidos (Pedido LONG, Cliente LONG, Vendedor TEXT(6),
    Fecha DATE, Totcost SINGLE)
```

Modificación de una tabla

Añadir un campo a una tabla

```
ALTER TABLE Pedidos ADD COLUMN Entrega SINGLE
```

Borrar un campo de una tabla

```
ALTER TABLE Pedidos DROP COLUMN Entrega
```

Eliminación de una tabla

DROP TABLE Pedidos

Definición de índices

Crear índices

CREATE INDEX Cliente ON Clientes (Cliente) WITH PRIMARY

CREATE INDEX Nombre2 ON Clientes (Apellido ASC, Nombre DESC)

Eliminar índices

DROP INDEX Cliente ON Clientes

CICLO FORMATIVO DE GRADO SUPERIOR: ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS

MÓDULO 6: SISTEMAS GESTORES DE BASES DE DATOS

UNIDADES DE TRABAJO

UNIDAD DE TRABAJO Nº 1

GESTOR DE BASES DE DATOS MICROSOFT ACCESS

1. INTRODUCCIÓN

1.1. Qué es una Base de Datos
1.2. Qué es una Base de Datos relacional
1.3. Qué es Microsoft Access
  1.3.1. Tablas
  1.3.2. Consultas
  1.3.3. Formularios
  1.3.4. Macros
  1.3.5. Módulos

2. EMPEZANDO A TRABAJAR CON ACCESS

2.1. Antes de empezar
  2.1.1. Preparación del disco de trabajo
  2.1.2. Planificando la Base de Datos
2.2. Creando la Base de Datos
  2.2.1. Creación de tablas
  2.2.2. Definiendo los campos
  2.2.3. La clave principal
  2.2.4. Establecer la clave principal
  2.2.5. Los índices
  2.2.6. Guardar una tabla
  2.2.7. Cerrar una tabla
2.3. Mantenimiento de la Base de Datos
  2.3.1. Modificar la estructura de una tabla
  2.3.2. Botones Presentación Diseño y Hoja de datos
  2.3.3. Cambiar de nombre una tabla
  2.3.4. Eliminando una tabla
2.4. Compactando una Base de Datos

3. INTRODUCIENDO Y EDITANDO DATOS

3.1. Introducir datos en una tabla
  3.1.1. Añadir registros a una tabla
3.2. Modificando registros
3.3. Desplazamientos en la tabla
  3.3.1. Ir a
3.4. Copiar, mover y eliminar datos
  3.4.1. Seleccionar datos
  3.4.2. Moviendo datos entre registros
  3.4.3. Copiar, mover o eliminar registros
  3.4.4. Copiar y mover datos entre tablas distintas
3.5. El comando deshacer
3.6. Modificar la apariencia de la hoja de datos
  3.6.1. Anchura y Altura de las filas
  3.6.2. Cambiar el orden de las columnas
  3.6.3. Inmovilizar, liberar y ocultar columnas
  3.6.4. Cambiando el tipo y el tamaño de letra
3.7. Buscar y reemplazar datos

3.7.1. Opciones
3.7.2. Caracteres comodín
3.7.3. Buscar y reemplazar

4. RELACIONES
4.1. Tipos de relaciones
4.1.1. Relación de uno a uno
4.1.2. Relación de muchos a muchos
4.1.3. Relación de uno a muchos
4.2. Las relaciones en Access
4.2.1. Creando relaciones
4.2.2. Ver las relaciones
4.2.3. Editando relaciones
4.2.4. Eliminar una relación
4.3. La integridad referencial

5. IMPRESIÓN

6. EXPRESIONES Y FUNCIONES
6.1. Expresiones
6.1.1. Operadores
6.1.2. Identificadores
6.1.3. Literales
6.1.4. Constantes
6.2. Funciones
6.2.1. Funciones estadísticas
6.2.2. Funciones matemáticas
6.2.3. Funciones financieras
6.2.4. Funciones varias
6.3. El generador de expresiones

7. CONSULTAS
7.1. Qué son las consultas
7.2. Creando la consulta
7.2.1. Planificar una consulta
7.2.2. Identificar la tabla que necesitamos
7.2.3. Elementos de la ventana de diseño
7.2.4. Añadiendo, eliminando campos a la consulta
7.2.5. Ejecutar la consulta
7.2.6. La hoja de datos de la consulta
7.2.7. Guardar una consulta
7.3. Introduciendo criterios. Expresiones
7.3.1. El operador Y, O
7.4. Modificar los campos
7.4.1. Cambiar los nombres de los campos
7.4.2. Ordenar los campos de una consulta
7.5. Ocultar campos, la fila mostrar
7.6. Clasificar una consulta
7.7. Los campos calculados

8. CONSULTAS AVANZADAS

- 8.1. Consultas con múltiples tablas
  - 8.1.1. Las uniones de tablas en las consultas
  - 8.1.2. Las uniones y relaciones
  - 8.1.3. Uniones externas y auto-uniones
  - 8.1.4. Incluir condiciones en la consulta
- 8.2. Consultas de agrupación y totales
- 8.3. Consultas de acción
  - 8.3.1. Creación de nuevas tablas
  - 8.3.2. Borrar registros
  - 8.3.3. Añadir registros
  - 8.3.4. Actualizar tablas
- 8.4. Las consultas de parámetros
- 8.5. Consulta de buscar duplicados
- 8.6. Consultas de archivación
- 8.7. Propiedades de la consulta

9. FORMULARIOS (I)

- 9.1. Qué es un Formulario
- 9.2. Formularios que se pueden realizar con Access
- 9.3. El Asistente de formularios
  - 9.3.1. Creación de un formulario de Columna simple
  - 9.3.2. Creación de un Formulario Tabular
  - 9.3.3. Creación de un Formulario Principal/Subformulario
  - 9.3.4. Creación de un Formulario Gráfico
- 9.4. Uso de un Formulario
- 9.5. Los filtros en los Formularios
  - 9.5.1. Utilizar consultas como filtros
  - 9.5.2. Mostrar todos los registros de nuevo

10. FORMULARIOS (II)

- 10.1. Creación de un Formulario en blanco
- 10.2. Ventana de diseño de Formularios
  - 10.2.1. Las secciones del formulario
  - 10.2.2. Las reglas
  - 10.2.3. La caja de herramientas
  - 10.2.4. Los controles
  - 10.2.5. Clases de controles
- 10.3. Operaciones básicas con controles
  - 10.3.1. Seleccionar un control
  - 10.3.2. Mover un control
  - 10.3.3. Cambiar el tamaño de un control
  - 10.3.4. Cortar, copiar y pegar
- 10.4. Añadir y eliminar controles. Bloquear la herramienta
- 10.5. Las propiedades de los controles
  - 10.5.1. Herencia de propiedades de campo
  - 10.5.2. Las propiedades por omisión de los tipos de controles
- 10.6. Clases de controles
  - 10.6.1. Los cuadros de lista
    - 10.6.1.1. Valores fijos

10.6.1.2. Extraer la lista de valores de un campo
10.6.1.3. Los cuadros combinados
10.6.1.4. Botones de opción
10.6.1.5. Cuadros de vitrificación
10.6.1.6. Botones de alternar
10.6.1.7. Grupos de opciones
10.6.1.8. Los botones de comandos
10.7. Propiedades del formulario
10.7.1. Cambiar la tabla o consulta en la que está basado
10.7.2. Presentación predeterminada
10.7.3. Definir la cuadrícula
10.8. Modificar el orden de selección de la tecla Tab

11. ILUSTRACIONES, GRÁFICOS Y OTROS OBJETOS
11.1. En qué consiste incrustrar o enlazar un objeto
11.1.1. Qué es un objeto
11.1.2. Las posibilidades de OLE: Incrustar y enlazar
11.2. Incrustar objetos en Access
11.2.1. Objetos dependientes y objetos independientes
11.2.2. Incrustar objetos dependientes en tablas
11.2.3. Editar objetos dependientes
11.2.4. Mostrar objetos dependientes en formularios o informes
11.2.5. Cambiar un objeto dependiente para que no pueda editarse
11.3. Incrustar objetos independientes
11.4. Las propiedades de los marcos de objetos

12. INFORMES (I)
12.1. Qué es un Informe
12.2. Informes que se pueden realizar con Access
12.3. El asistente para informes
12.3.1. Creación de un Informe de Columna simple
12.3.2. Creación de un Informe de Grupos/Totales
12.3.3. Creación de un Informe de Resumen
12.3.4. Creación de un Informe Tabular
12.3.5. Creación de un Informe Automático

13. INFORMES (II)
13.1. Crear un informe desde un formulario
13.2. Creación de un Informe en blanco
13.3. La ventana de presentación
13.4. La ventana de diseño
13.4.1. Propiedades especiales de los cuadros de texto en los Informes
13.4.2. Las secciones y saltos de página
13.4.3. Agrupaciones y clasificación de datos
13.4.4. Las propiedades del Informe
13.5. Subinformes
13.5.1. Insertar y enlazar un subinforme en un Informe
13.6. Impresión del Informe

UNIDAD DE TRABAJO Nº 2

ODBC (Open DataBase Connectivity)

2.1. Introducción

ODBC es un protocolo estándar para el acceso a la información de servidores de bases de datos SQL, como por ejemplo Microsoft SQL Server. Se pueden instalar controladores de ODBC que permitan que Microsoft Access se conecte a estos servidores de bases de datos SQL y tenga acceso a los datos de las bases de datos SQL.

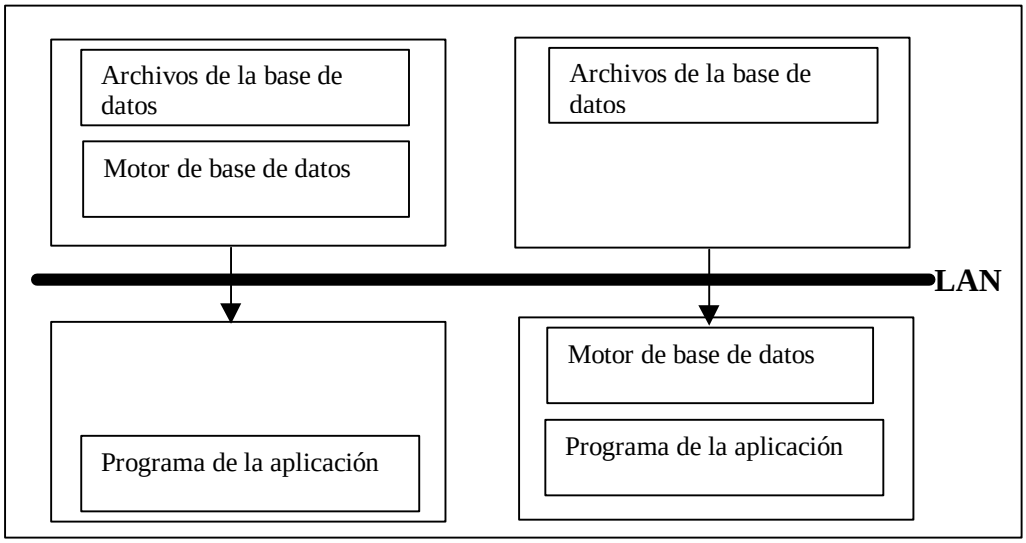
2.2. Aplicaciones con base de datos remota frente a aplicaciones cliente/servidor

Bases de datos remotas

Las aplicaciones con bases de datos remotas tienen los archivos de la base de datos en un servidor de archivos compartido, y se accede desde los equipos terminales a dicha base de datos compartida utilizando la aplicación y el motor de base de datos local.

Aplicaciones cliente/servidor

En aplicaciones cliente/servidor se accede a la información haciendo que la aplicación cliente envíe una consulta al servidor. La aplicación cliente suele enviar la consulta a través de un protocolo de red, el servidor recupera los datos y los envia de nuevo a la aplicación cliente.



2.3. Operaciones con ODBC

Instalar o tener instalado el controlador ODBC correspondiente

Para ver los controladores instalados, véase “Controladores ODBC” del Administrador de orígenes de datos ODBC del panel de control.

Para crear un nuevo origen de datos de usuario (**D**ata**S**ource**N**ame Usuario)

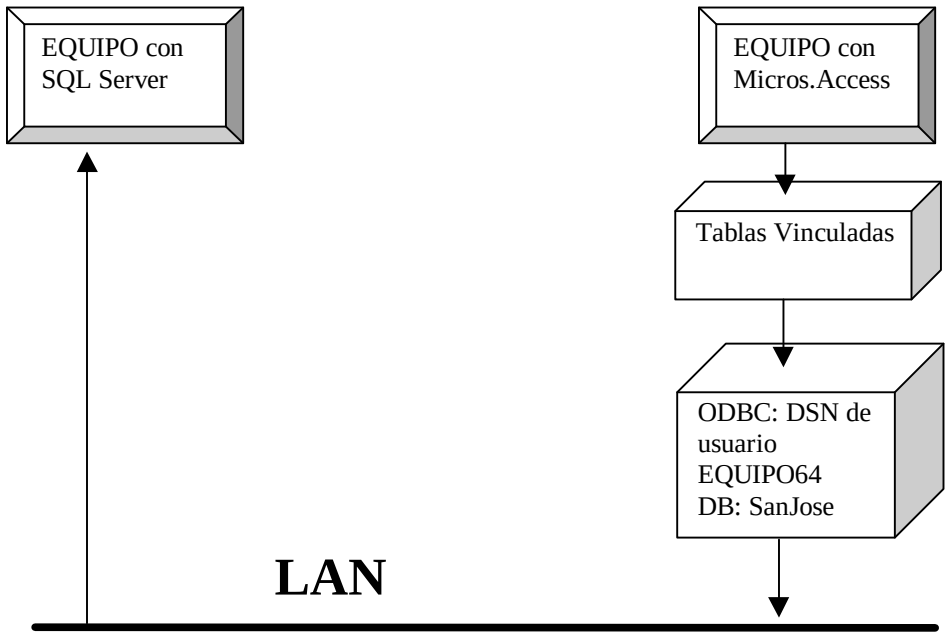
Las operaciones se realizan desde el Administrador de orígenes de datos ODBC del panel de control, a partir de aquí, activar la ficha rotulada como “DSN de usuario”

A partit de aquí realizar los siguientes pasos:

- Pulsar sobre botón de agregar
- Elegir el controlador sobre el cual vamos a montar el DSN de usuario
- Pulsar sobre "Opciones" para ver todas las posibilidades
- Rellenar los siguiente apartados
- Data Source Name: El nombre que queremos darle al DSN de usuario. Es libre, podemos poner lo que queramos
- Descripción: una descripción del DNS. Libre contenido.
- Server: el nombre del servidor que da el servicio de base de datos, por ejemplo EQUIPO64. (Es el nombre del equipo de la red que da el servicio de SQL Server)
- Activar la casilla de verificación "Use Trusted Connection"
- Database Name: escribir el nombre la base de datos del servidor SQL a la que queremos conectarnos con este DSN de usuario. Recordemos que en un servidor de SQL Server puede haber muchas bases de datos. Así por ejemplo pondremos, por ejemplo "SanJose".

2.4. Conexiones entre un servidor SQL Server y Access

Vinculando tablas

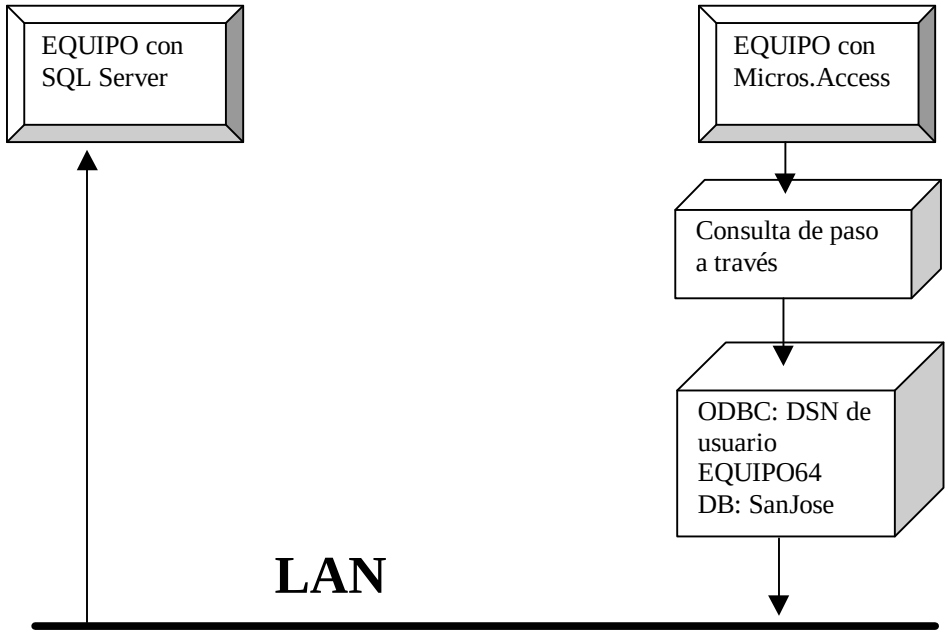


Mediante tablas vinculadas, estamos ante un sistema de aplicaciones de bases de datos remotas, ya que mediante la vinculación, el propio motor Jet de Access local, tiene que realizar gran parte del trabajo, compilando la instrucción, procesandola y enviándola por partes al servidor. También estamos sujetos a las funciones e instrucciones del motor Jet y no a las mayores posibilidades que puede ofrecer el servidor.

Para vincular tablas realizar en Access los siguientes pasos:

- Archivo|Obtener datos externos|Vincular Tablas
- Tipo de archivo: Bases de datos ODBC
- Origen de datos de equipo: elegir el DSN correspondiente
- Elegir tablas y campos

Consultas de paso a través



Este tipo de consulta envía comandos directamente a las bases de datos ODBC, como las de Microsoft SQL Server, utilizando comandos aceptados por el servidor. Por ejemplo, puede emplear una consulta de paso a través para recuperar registros o modificar datos. Este tipo de consultas pertenecen a aplicaciones de tipo cliente/servidor.

Algunas ventajas son:

- La reducción del ancho de banda utilizado de la red
- Se puede acceder a la funcionalidad específica del servidor
- Soporta uniones de multiples tablas de multiples bases de datos
- Las consultas de acción son mucho más rápidas

Para crear una consulta de paso a través

- Crear la consulta en vista de diseño sin agregar ninguna tabla
- Consulta|Específica de SQL|Paso a través

- Visualizar las propiedades de la consulta
- En cadena de conexión ODBC, pulsar sobre le generador, y elegir el DSN correspondiente, entonces se escribira una cadena de conexión del tipo:
- ODBC;DSN=Acceso desde Access;UID=juanra;DATABASE=SanJose
- Decir si la cadena de conexión devuelve registro o no. (un comando DDL no devuelve registros, un DML si)
- Cerrar las propiedades de la consulta
- Escribir la instrucción SQL

2.5. Conexiones entre un servidor SQL Server y Access mediante código de visual basic

Qué es ODBCDirect

ODBCDirect es una tecnología que permite trabajar con servidores de bases de datos ODBC sin necesidad de cargar el motor de base de datos Microsoft Jet. ODBCDirect se basa en el modelo de objetos Microsoft DAO 3.5, con lo que puede modificar fácilmente el código DAO existente y obtener ventaja de ODBCDirect. Microsoft DAO 3.5 incluye nuevos objetos, métodos, y propiedades compatibles con ODBCDirect.

Ventajas del acceso a datos ODBC con ODBCDirect

ODBCDirect ofrece las siguientes ventajas para las operaciones ODBC:

- ODBCDirect puede hacer el código más rápido y más eficiente al ofrecer acceso directo a orígenes de datos ODBC. Puesto que no requiere que se cargue el motor de base de datos Microsoft Jet, ODBCDirect consume menos recursos en el lado del cliente. El servidor ODBC es el responsable de todo el proceso de consulta.
- ODBCDirect ofrece un acceso mejorado a las características específicas del servidor que no están disponibles usando ODBC a través de Microsoft Jet. Por ejemplo, para los servidores compatibles con la especificación del cursor, ODBCDirect permite especificar dónde está ubicado el cursor, si en el servidor o localmente. Además, para interactuar con los procedimientos almacenados al nivel del servidor, puede especificar valores de entrada y comprobar los valores de retorno, lo que no puede hacer cuando usa Microsoft Jet.
- ODBCDirect también admite consultas asíncronas. Cuando ejecuta una consulta, no tiene que esperar a que la consulta finalice la ejecución antes de empezar otra operación. Puede hacer el seguimiento de la consulta comprobando la propiedad StillExecuting.
- ODBCDirect admite actualización por lotes, lo que permite almacenar localmente en el caché los cambios del objeto Recordset y luego enviar estos cambios al servidor en un solo lote.
- Con ODBCDirect, puede crear conjuntos de resultados sin cursor simples o cursores más complejos. También puede ejecutar consultas que devuelven cualquier número de conjuntos de resultados. Puede limitar el número de filas devueltas y controlar todos los mensajes y errores generados por el origen de datos remoto sin que esto afecte al rendimiento de la consulta en ejecución.

A continuación se lista el contenido de un módulo de Access con código de visual basic que utilizando el objeto Connection, permite pasar a través de ODBC realizar consultas tipo cliente/servidor sobre una base de datos SQL Server.

```
Sub OpenRecordsetX()
        Dim wrkJet As Workspace
        Dim wrkODBC As Workspace
        Dim dbsNeptuno As Database
        Dim conEditores As Connection
        Dim rstTemp As Recordset
        Dim rstTemp2 As Recordset
   ' Abre Microsoft Jet y el espacio de trabajo ODBCDirect, la base de datos
   ' Microsoft Jet y  la conexión ODBCDirect.
   Set wrkJet = CreateWorkspace("", "admin", "", dbUseJet)
   Set wrkODBC = CreateWorkspace("", "admin", "", dbUseODBC)
   Set dbsNeptuno = wrkJet.OpenDatabase("c:\juanra\experimentos2.mdb")
   Set conEditores = wrkODBC.OpenConnection("", , , _
      "ODBC;DATABASE=SanJose;UID=juanra;DSN=Acceso desde Access")
' ODBC;DSN=Acceso desde Access;UID=juanra;DATABASE=SanJose
   ' Abre cinco objetos Recordset diferentes y muestra el contenido de cada uno.

Debug.Print "Abriendo el recordset de tipo Forward-only" & _
      "donde el origen es un objeto QueryDef..."
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "Amigos", dbOpenForwardOnly)
   SalidaAbrirRecordset rstTemp

   Debug.Print " Abriendo el recordset de sólo lectura de tipo Dynaset " & _
      "donde el origen es una instrucción SQL..."
   Set rstTemp = dbsNeptuno.OpenRecordset( _
      "SELECT * FROM Amigos", dbOpenDynaset, dbReadOnly)
   SalidaAbrirRecordset rstTemp
```

```
    ' Utiliza la propiedad Filter para recuperar sólo ciertos
    ' registros con la siguiente llamada OpenRecordset.
    Debug.Print "Abriendo el recordset de un objeto" & _
        "Recordset existente para filtrar..."
    rstTemp.Filter = "Nombre >= 'ana'"
    Set rstTemp2 = rstTemp.OpenRecordset()
SalidaAbrirRecordset rstTemp2

    Debug.Print " Abriendo el recordset de tipo Dynamic de" & _
        "una conexión ODBC..."
    Set rstTemp = conEditores.OpenRecordset( _
        "SELECT * FROM profesores", dbOpenDynamic)
    SalidaAbrirRecordset rstTemp

    ' Utiliza la propiedad StillExecuting para determinar cuando está
    ' el Recordset preparado para manipular.
    Debug.Print " Abriendo el recordset de tipo Snapshot basado" & _
        "en una consulta asíncrona a una conexión ODBC..."
Set rstTemp = conEditores.OpenRecordset("alumnos", _
        dbOpenSnapshot, dbRunAsync)
    Do While rstTemp.StillExecuting
        Debug.Print "    [todavía en ejecución...]"
    Loop
    SalidaAbrirRecordset rstTemp

    rstTemp.Close
    dbsNeptuno.Close
    conEditores.Close
    wrkJet.Close
    wrkODBC.Close

End Sub
```

```
Sub SalidaAbrirRecordset(rstOutput As Recordset)

   ' Enumera el objeto Recordset especificado.
   With rstOutput
      Do While Not .EOF
         Debug.Print , .Fields(0), .Fields(1)
.MoveNext
         Loop
   End With

End Sub
```

UNIDAD DE TRABAJO Nº 3

UTILIZACIÓN DE BASES DE DATOS SQL CON ACCESS

3.1. Explotar todas las posibilidades del entorno Access, pero utilizando el motor de SQLServer en todas las consultas y acciones contra datos

Es tan sencillo como:

- Crear un conexión ODBC en el equipo que tiene la aplicación Access.
- Crear una consulta de paso a través para cada una de las tablas que residen en base de datos SQLServer (cada consulta es idéntica a la tabla de la base de datos de SQLServer, tienen exactamente los mismos campos).
- A partir de este momento utilizar como tablas para trabajar con el Access las consultas creadas en el apartado anterior, creando por lo tanto formularios contra estas consultas, informes, y a su vez consultas sobre dichas consultas. El único problema son las relaciones entre las tablas que en toda consulta nueva tendrán que ser repintadas. También podrán crearse formularios con subformularios.

3.2. Realizar un formulario en Access que lleve todo el control de desplazamiento de registro de un recordset que es producto de una conexión "Connection" contra un ODBC que interactual contra SQLServer

```
Option Compare Database
Option Explicit
Dim NewDb As Database, NewWs As Workspace, NewTbl As Recordset
Dim NewDb As Connection
```

```
Private Sub Adelante_Click()

NewTbl.MoveNext
Forms!MostrarPersonas.Controls!Nombre.Value = NewTbl("Nombre")
Forms!MostrarPersonas.Controls!Peso.Value = NewTbl("Peso")

End Sub
```

```
Private Sub Atras_Click()

NewTbl.MovePrevious
Forms!MostrarPersonas.Controls!Nombre.Value = NewTbl("Nombre")
Forms!MostrarPersonas.Controls!Peso.Value = NewTbl("Peso")

End Sub
```

```
Private Sub Form_Close()

NewTbl.Close
NewDb.Close
NewWs.Close

End Sub
```

```
Private Sub Form_Open(Cancel As Integer)

Set NewWs = DBEngine.Workspaces(0)
Set NewWs = CreateWorkspace("", "admin", "", dbUseODBC)

Set NewDb = NewWs.OpenDatabase("c:\mis documentos\juanra verano 98\san jose verano 98\m6 - sistemas gestores de bases de datos\solodatos.mdb")
Set NewDb =NewWs .OpenConnection("", , , "ODBC;DATABASE=SanJose;
UID=juanra;DSN=Acceso desde Access")

Set NewTbl = NewDb.OpenRecordset("Personas", dbOpenDynaset)
Set NewTbl = NewDb.OpenRecordset("Personas", dbOpenDynaset)

NewTbl.MoveFirst
Forms!MostrarPersonas.Controls!Nombre.Value = NewTbl("Nombre")
Forms!MostrarPersonas.Controls!Peso.Value = NewTbl("Peso")

End Sub
```

```
Private Sub ConsultaSQL_Click()

Dim SQLConsulta As String

NewTbl.Close

SQLConsulta = "Select * "
SQLConsulta = SQLConsulta & "from personas where Peso>10"

Set NewTbl = NewDb.OpenRecordset(SQLConsulta, dbOpenDynaset)

NewTbl.MoveFirst
Forms!MostrarPersonas.Controls!Nombre.Value = NewTbl("Nombre")
Forms!MostrarPersonas.Controls!Peso.Value = NewTbl("Peso")

End Sub
```

```
Private Sub Todos_Click()

NewTbl.Close

Set NewTbl = NewDb.OpenRecordset("Personas", dbOpenDynaset)

NewTbl.MoveFirst
Forms!MostrarPersonas.Controls!Nombre.Value = NewTbl("Nombre")
Forms!MostrarPersonas.Controls!Peso.Value = NewTbl("Peso")

End Sub
```

3.3. Uso de métodos para recuperar un conjunto de registros definidos mediante una instrucción SQL.

Se pueden definir instrucciones SQL, y que el recordset obtenido sea producto de una consulta SQL y no de una tabla completa de la base de datos. Para ello podrá definirse la instrucción SQL en una variable, y al crear el recordset con el método "OpenRecordset" sobre la base de datos o connection, utilizar dicha variable que contiene la instrucción SQL. Veámoslo en un ejemplo:

```
‘Declarar la variable
Dim SQLconsulta1 as String

'Definir la instrucción SQL y almacenarla en una variable
SQLconsulta1 = "SELECT Articulos.Nombre, Articulos.Precio"
SQLconsulta1 = SQLconsulta1 & "FROM Articulos"
```

```
SQLconsulta1 = SQLconsulta1 & "WHERE Articulos.Stock >20"


'Crear el recordset de tipo dynaset, que sea resultado de la ejecución de la consulta
'anterior
Set NewTbl = NewDb.OpenRecordset(SQLconsutla1, dbOpenDynaset)
```

UNIDAD DE TRABAJO Nº 4

TRANSACT-SQL

4.1. Ejemplo de Trigger que se dispara para actualizar el stock de un artículo, cuando un cliente compra un cierto número de unidades.

Tablas

Clientes = (NumeroC, Nombre)

Artículos = (NumeroA, Descripción, Stock)

Compras = (NumeroC, NumeroA, Unidades)

NOTA: solo permite que un cliente compre una sola vez un artículo, aunque compre varias unidades de ese mismo artículo en la compra.

TRIGGER asociado a la tabla Compras

```
CREATE TRIGGER ActualizarStock ON dbo.Compras
FOR INSERT
AS

-declaración de variables
declare @articulocomprado int, @unidadescompradas int

--cojemos la información del Numero del ultimo articulo vendido y cuantas unidades
--se han vendido
select @articulocomprado=NumeroA, @unidadescompradas=Unidades from Compras

-actualizar el stock de articulo
--decrementamos el stock del articulo, en el numero de unidades que se hayan vendido
update Articulos set Stock=(Stock-@unidadescompradas) where NumeroA=@articulocomprado

GO
```

4.2. Ejemplo de Trigger que se dispara para actualizar la descripción de un producto cuando un registro sufre una modificación (update) en alguno de sus campos.

Tablas

Artículos = (Numero, Nombre, Importe, Descripción)

TRIGGER asociado a la tabla Articulos

```
CREATE TRIGGER Descripcion ON dbo.Articulos
FOR UPDATE
AS
update Articulos set descripcion =
        case
                when importe>100 then 'Caro'
                when importe<=100 then 'Barato'
                else 'Imposible'
        end
GO
```

4.3. Ejemplo de Trigger que se dispara para actualizar el stock de un artículo, cuando un cliente compra un cierto número de unidades y que actualiza si estamos por debajo del stock mínimo.

Tablas

Clientes = (NumeroC, Nombre)

Artículos = (NumeroA, Descripción, Stock, BajoStock)

Compras = (NumeroC, NumeroA, FechaHora Unidades)

Si el stock está por debajo de 10, lo clasifica como artículo por debajo del stock mínimo.

TRIGGER asociado a la tabla Compras

```
CREATE TRIGGER ActualizarStock ON dbo.Compras
FOR INSERT
AS

--declaración de variables
declare @articulocomprado int, @unidadescompradas int

--cojemos la información del Numero del último artículo vendido y cuantas unidades
--se han vendido
select @articulocomprado=NumeroA, @unidadescompradas=Unidades from Compras

--actualizar el stock de artículo
--decrementamos el stock del artículo, en el número de unidades que se hayan vendido
update Articulos set Stock=(Stock-@unidadescompradas) where NumeroA=@articulocomprado
GO
```

TRIGGER asociado a la tabla Articulos

```
CREATE TRIGGER ControlBajoStock ON dbo.Articulos
FOR UPDATE
AS
update Articulos set BajoStock=
        case
                when Stock<10 then 'Sí'
                else 'No'
        end
GO
```

4.4. Ejemplo de cómo ejecutar un procedimiento almacenado en el SQL Server, desde el Access.

Procedimiento Almacenado en el servidor SQL Server.

```
CREATE PROCEDURE Datos_del_Articulo @Numero int AS
Select Descripcion from Articulos where NumeroA = @Numero
GO
```

Código asociado a un botón de un formulario de **Access** al hacer "click"

```
Private Sub Comando0_Click()
Dim wrkODBC As Workspace
Dim articulos As Connection
Dim registro As Recordset
Set wrkODBC = CreateWorkspace("","Admin", "", dbUseODBC)
Set articulos = wrkODBC.OpenConnection("", , ,
"ODBC;DATABASE=SanJose;UID=Juanra;DSN=AccesoSQL")
Set registro = articulos.OpenRecordset("execute Datos_del_Articulo 1", dbOpenDynamic)
With registro
        Do While Not .EOF
                Debug.Print, .Fields(0)
                .MoveNext
        Loop
End With
End Sub
```

4.5. Ejemplo de venta de artículo y correspondiente decremento del stock, pero con vuelta atrás si el nuevo stock se vuelve inferior al stock mínimo.

Tablas

Véase base de datos **Tienda**, en SQL Server.

TRIGGER asociado a la tabla Ventas

```
CREATE TRIGGER ActualizarStock ON dbo.Ventas
FOR INSERT
AS

-- En un pedido, se produce la venta de "x" unidades de un artículo, lo que provoca
-- que decrementemos en "x" unidades el stock del artículo. Si ahora el stock es más
-- pequeño que el stock mínimo, deshacemos la transacción y eliminamos la venta en ese
-- pedido. Sino, aceptamos la transacción.

declare @pedido char (4), @articulo char (4), @unidades int, @nuevostock int, @stockminimo
int

select @pedido=NPedido, @articulo=CArticulo, @unidades=Unidades from Ventas

begin transaction

update Articulos set @nuevostock=(Stock-@unidades), @stockminimo=stockminimo,
Stock=(Stock-@unidades) where CArticulo=@articulo

if (@nuevostock < @stockminimo)
        begin
                rollback transaction
                delete from ventas where (NPedido = @pedido) and (CArticulo=@articulo)
        end
        else
        begin
                commit transaction
        end
GO
```

UNIDAD DE TRABAJO Nº 5

UTILIZACIÓN DE SENTENCIAS DE CONTROL DE FLUJO CON TRANSACT-SQL

4.1. Introducción.

Transact-SQL contiene varias sentencias que se utilizan para modificar el orden de ejecución de las sentencias dentro de un conjunto de sentencias tales como un procedimiento almacenado.

4.2. Utilización de IF...ELSE

La sintaxis de una sentencia IF...ELSE es la siguiente:

```
IF expresión
        Sentencia
        [ELSE]
                [IF expresión]
                        [sentencia]
```

Ejemplo:

```
If exist  (select * from trabajadores where Placa = 1234)
        Print 'Empleado presente'
                Else
                        Print 'empleado no encontrado'
```

4.3. Utilización de BEGIN...END

La sintaxis de una sentencia BEGIN...END es la siguiente:

```
BEGIN
        Sentencias
END
```

Ejemplo:

```
If exist (select * from empleados where placa = 1234)
        Begin
                Print 'entrada disponible'
                Select nombre, departamento, from empleados where placa = 1234
        End
        Else
                Print 'No hay información sobre el empleado'
```

4.4. Utilización de WHILE

La sentencia WHILE se utiliza para definir una condición que ejecute una o más sentencias de Transact-SQL.

La sintaxis de una sentencia WHILE es la siguiente:

```
WHILE <expresión>
        <sentencias sql>
```

Ejemplo:

```
Declare @x int
Select @x = 1
While @x<5
        Begin
                Print 'x sigue siendo menor que 5'
                Select @x = @x + 1
        End
```

4.5. Utilización de BREAK

La sentencia BREAK se utiliza dentro de un bloque de sentencias de Transact-SQL que se encuentre dentro de una sentencia condicional WHILE para finalizar la ejecución de las sentencias. La ejecución de la sentencia BREAK da lugar a que la primera sentencia que siga al fin del bloque comience a ejecutarse.

La sintaxis de una sentencia BREAK es la siguiente:

```
WHILE
        <expresión booleana>
        <sentencias sql>
BREAK
        <sentencias sql>
```

Ejemplo:

```
Declare @x int
Select @x = 1
While @x<5
        Begin
                Print 'x sigue siendo menor que 5'
                Select @x = @x + 1
                Break
        End
```

4.6. Utilización de CONTINUE

La sentencia CONTINUE se utiliza dentro de un bloque de sentencias de Transact-SQL que se encuentre dentro de una sentencia condicional WHILE para continuar explicitamente el conjunto de sentencias que estaban contenidas dentro de la sentencia condicional.

La sintaxis de una sentencia CONTINUE es la siguiente:

```
WHILE <expresión booleana>
        <sentencia>
CONTINUE
```

Ejemplo:

```
Declare @x int
Select @x = 1
While @x<5
        Begin
                Print 'x sigue siendo menor que 5'
                Select @x = @x + 1
                Continue
                Print 'esta sentencia no se va a ejecutar'
```

End

4.7. Definición y utilización de variables

Las variables pueden ser locales o globales. Las variables locales se definen utilizando una sentencia DECLARE y asignándoles un tipo de datos. A las variables locales se les asigna un valor utilizando la sentencia SELECT.

Las variables locales se declaran, asignan un valor y se utilizan siempre dentro del mismo lote o procedimiento almacenado, de aquí el nombre de locales.

Para declarar una variable local de pone el "@" delante de lo que será la variable.

La sintaxis de las VARIABLES es la siguiente:

DECLARE @nombre_variable tipo_de_datos [,@nombre_variable tipo_de_datos....]
SELECT @nombre_variable = expresión | sentencia select [,@nombre_variable = expresión | sentencia select.....]

Ejemplo:

```
Declare @minum int
Select @minum = count(*) from trabajadores
Declare @michar char(2)
Select @michar = convert(char(2), @minum)
Declare @menss = 'Hay ' + @michar + 'filas en la tabla trabajadores'
Print @menss
```

NOTA: cada sentencia select proporciona un mensaje de recuento dentro del ejemplo anterior. Si se desea suprimir el mensaje de recuento, primero es preciso ejecutar la sentencia SET NOCOUNT.

Las variable globales están definidas por SQL Server. Las variables globales van precedidas de los símbolos "@@".

Ejemplo:

```
Print @@VERSION
```

4.8. Utilización de PRINT con variables

La sentencia PRINT se utiliza para mostrar textos en ASCII o variables de hasta 255 caracteres. Solo puede utilizarse para sacar datos que sean de tipo CHAR o VARCHAR. Tampoco es posible concatenar cadenas de datos en una sentencia PRINT de forma directa, es preciso concatenar el texto o las variables en una única variable y sacar los resultados mediante la sentencia PRINT.

Sintaxis:

PRINT 'texto' | @variable_local | @@variable_global

4.9. Utilización de GOTO

La sentencia GOTO permite saltos incondicionales a una etiqueta.

Sintaxis:

```
Rótulo:
GOTO Rótulo
```

Si se quiere emplear el GOTO para permitir saltos condicionales se tiene que combinar con el WHILE, como muestra el ejemplo siguiente:

Ejemplo:

```
Declare @contador smallint
Select @contador = 1
Reiniciar: print 'sí'
Select @contador = @contador + 1
If @contador <=4 Goto reiniciar
```

4.10. Utilización de RAISERROR

La sentencia RAISERROR se utiliza para proporcionar un mensaje especificado por el usuario.

Sintaxis:

RAISERROR (<expresión_entera> | <'texto de mensaje'>, [gravedad] [,estado [,argumento1] [,argumento2]) [WITH LOG]

4.11. Utilización de WAITFOR

La sentencia WAITFOR se utiliza para especificar una hora, un intervalo de tiempo, o un suceso para ejecutar una sentencia, un bloque de sentencias, un procedimiento almacenado o una transacción.

Sintaxis:

WAITFOR (DELAY <'hora'> | TIME <'hora'> | ERROREXIT | PROCESSEXIT | MIRROREXIT)

Ejemplos:

```
Waitfor delay '00:00:40'
Select * from empleados


Waitfor time '15:10:51'
Select * from empleados
```

4.12. Utilización de expresiones CASE

La sentencia CASE se utiliza para llevar a cabo una decisión de ejecución basada en múltiples opciones.

Sintaxis:

```
CASE [expresión]
WHEN expresión sencilla1 | expresión booleana1 THEN expresión1
[WHEN expresión sencilla2 | expresión booleana2 THEN expresión2]...
        [ELSE expresiónN]
END
```

Ejemplo:

```
Select nombre,división=
        Case departamento
                When ‘Ventas’ then ‘Ventas y Marketing’
                When ‘Separaciones’ then ‘Grupo de Apoyo’
```

```
                When ‘Lógistica’ then ‘Piezas’
                     Else ‘Otro departamento’
End,
placa from compañía
```

UNIDAD DE TRABAJO Nº 6

EMPEZAR CON ORACLE

5.1. Crear un alias de base de datos.

Cuando se instala el oracle, se crea una base de datos llamada ORCL, pero para poder trabajar con ella, tenemos que crear un alias (un segundo nombre para la base de datos), que será el que utilizaremos para conectarnos y crear tablas, etc. sobre ella.

En nuestro ejemplo, hemos llamado al alias ORCL, o sea, que le hemos dado el mismo nombre que la base de datos original que se crea durante la instalación.

Realizar los siguientes pasos, para cubrir las necesidades explicadas en los párrafos anteriores:

- Poner en marcha el programa SQL Net Easy Configuration.
- Añadir un alias de base de datos.
- Darle como nombre de alias, por ejemplo ORCL.
- Elegir el protocolo TCP/IP.
- Poner la dirección IP del Servidor de Windows NT.

5.2. Como conectarse a ORACLE.

Cuando queramos emplear cualquier aplicación de ORACLE, deberemos de realizar una conexión ORACLE, para lo cual podremos emplear las siguientes cuentas:

Usuario	SYSTEM	SYS
Clave	MANAGER	CHANGE_ON_INSTALL
Cadena de conexión	ORCL (es el nombre del alias)	ORCL

Estas cuentas nos serán útiles, para conectarnos por ejemplo a SQL PLUS, SCHEMA MANAGER, SQL WORKSHEET, etc.

5.3. Crear un espacio para trabajar.

Antes de ponernos a trabajar, debemos crear un espacio lógico y físico, en donde se almacenen las tablas que los usuarios (administradores de la base de datos) van a crear para utilizar en las aplicaciones.

Para ello deberemos poner en marcha en primer lugar la aplicación “Storage Manager”, y realizar los siguientes pasos:

- Primero crearemos un directorio nuestro particular, que será donde almacenaremos los archivos físicos que guardarán los datos de la base de datos (tablas, usuarios, etc.). Este directorio particular debe estar dentro del directorio siguiente: F:\ORANT\DATABASE, así pues, nosotros podemos crear el directorio siguiente: F:\ORANT\DATABASE\SANJOSE.
- Crear un nuevo TABLESPACE llamado por ejemplo “SANJOSE”.

- Al crear este nuevo TABLESPACE, tendremos que detallar los siguientes datos: en DATAFILES tendremos que poner el nombre del fichero (o ficheros) que va ha utilizar ese TABLESPACE, el nombre del fichero puede ser cualquiera, pero su extensión debe ser .DBF, y por supuesto, lo almacenaremos dentro del directorio que hemos creado en el apartado anterior. Es decir, que el nombre del DATAFILE será por ejemplo: F:\ORANT\DATABASE\SANJOSE\SANJOSE.DBF (el nombre del archivo es SANJOSE.DBF). A continuación dentro de la ficha "general" detallaremos el espacio inicial, por ejemplo 10 Megas, y dentro de la ficha "Auto Extend" detallaremos auto incremento que irá sufriendo el archivo si este quedase lleno.
- Pulsaremos sobre el botón crear, y se creará el TABLESPACE.

5.4. Crear un usuario para que pueda trabajar sobre un TABLESPACE.

El propósito es crear un usuario y asociarlo a un TABLESPACE, para que todas sus tablas (de las cuales el será el propietario) utilicen el TABLESPACE antes creado, por ejemplo el TABLESPACE "SANJOSE", el cual a su vez guardará las tablas y demás objetos que dicho usuario cree, sobre el archivo físico SANJOSE.DBF, del directorio correspondiente.

Para todo ello hay que poner en marcha el programa SECURITY MANAGER, y realizar los siguientes pasos dentro del proceso de crear un usuario:

- Darle un nombre al nuevo usuario (por ejemplo ALUMNOS)
- Introducir la contraseña del usuario (por ejemplo ALUMNOS)
- Introducir el TABLESPACE que va a utilizar, por ejemplo el TABLESPACE "SANJOSE" creado anteriormente.
- En cuanto a ROLES, se detallan a partir de las características de la cuenta de usuario.
- En OBJECTS PRIVILEGES se detalla, sobre que otros objetos de otros usuario va a tener permiso.
- Finalmente se crea el usuario

5.5. Crear tabla para la base de datos.

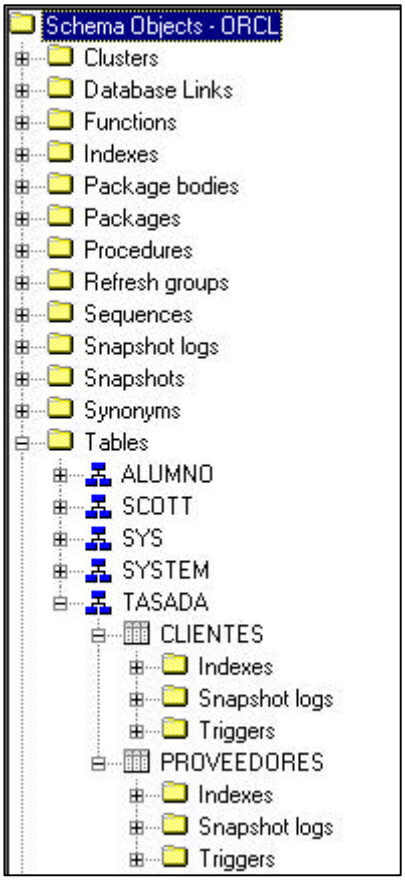
Para crear tablas en ORACLE, hay que pensar que estas siempre van asociadas a una cuenta de usuario y a un TABLESPACE.

La herramienta que permite crear tablas de una forma visual es el programa SCHEMA MANAGER. Cuando pongamos en marcha este programa, entraremos con la cuenta del usuario (que quiere ser la propietaria de los objetos que se van a crear), por ejemplo ALUMNOS y con la cadena de conexión ORCL.

Podremos apreciar que el SCHEMA MANAGER está formado por un árbol de objetos (todos los objetos de ORACLE), y dentro de cada objeto tenemos la CUENTAS DE USUARIO (más bien objetos SCHEMAS que se crean la primera vez que un usuario entra y que llevan el mismo nombre que la cuenta de usuario con la cual se ha entrado) que tienen alguno de estos objetos. Así pues nosotros si creamos un objeto de tipo TABLES, lo primero que nos pedirá es el SCHEMA (o cuenta de USUARIO si es la

primera vez)(por ejemplo ALUMNO) y la TABLESPACE (por ejemplo SANJOSE) al que tiene que ir asociado el objeto o objetos que queremos crear.

Es decir, para cada tipo de objeto de la colección de objetos de ORACLE, encontraremos en su interior un SCHEMA (que coincidirá con el nombre de un usuario), a partir del cual "colgarán" los objetos de ese mismo tipo que se vayan creando para dicho SCHEMA.



NOTA: El SCHEMA "Tasada" no se crea hasta que el usuario "Tasada" entra en el sistema y crea además un objeto de tipo TABLES.

5.5. Aplicaciones que se pueden utilizar para realizar consultas SQL sobre el servidor de ORACLE

SQL PLUS

Sirve para escribir sentencias de SQL. El proceso es particular de este interprete de SQL y funciona de la siguiente forma:

- Primero ponemos en marcha la aplicación, para la cual tendremos que identificarnos como uno de los usuarios, por ejemplo ALUMNO.

- A continuación nos aparece el interprete de SQL.

En este interprete podemos escribir una sentencia de SQL, por ejemplo

```
SQL> INSERT INTO Clientes (Nombre, Apellidos) VALUES ('Ana', 'Campos')
  2
```

- Esta sentencia no se ejecuta, para que se ejecute tenemos que escribir RUN, véase en la siguiente linea

```
SQL> run
  1* INSERT INTO Clientes (Nombre, Apellidos) VALUES ('Ana', 'Campos')
1 fila creada.
```

- Si queremos modificar la última línea escrita en el editor, y guardar las modificaciones que hagamos, deberemos utilizar el editor ED, escribiendo el comando que lleva su mismo nombre, ED. Véase la siguiente línea

```
SQL> ed
Escrito file afiedt.buf
1* INSERT INTO Clientes (Nombre, Apellidos) VALUES ('Ana', 'Campos')
```

ahora la podríamos modificar si lo deseásemos, guardar las modificaciones, salir del editor, y hacer un RUN (de cómo ha quedado la instrucción SQL modificada).

- También podríamos escribir en el interprete nuevas instrucciones

```
SQL> select * from Clientes
  2
SQL> run
1* select * from Clientes
NOMBRE     APELLIDOS
---------- ----------
Juan       Amengual
Jose       Rossello
MIGUEL     ROMBERT
Ana        Campos
SQL>
```

OBSERVACIÓN: Si ALUMNO quiere hacer un SELECT de una tabla para la que tiene permiso, pero no se encuentra dentro de su SCHEMA, deberá de hacer indicando en que SCHEMA se encuentra la TABLA, eso es por ejemplo:

```
SELECT * FROM TASADA.PROVEEDORES
```

Donde TASADA es el SCHEMA, y PROVEEDORES la TABLE.

SQL WORKSHEET

También sirve para realizar consultas de SQL, solo que en la parte inferior podemos escribir las consultas SQL, y arriba ver el resultado de las consultas.

MS-QUERY

También puede crearse un ODBC para ORACLE, y utilizarlo para conectarse desde el MS-QUERY. Los puntos importantes que nos pide son simplemente, el SQL NET CONNECT STRING que es ORCL, y el USER ID que puede ser por ejemplo ALUMNO.