INDICE

ASIGNATURA : BASES DE DATOS

EXPECTATIVAS DE LOGRO

- **Comprender la necesidad de sistematizar la información, conforme con las características de las organizaciones actuales.**
- **Conocer la organización y la estructura de las Bases de Datos.**
- **Diseñar y administrar una base de datos mediante un lenguaje adecuado (SQL)**
- **Administrar Base de Datos Relacionales.**
- **Diseñar un adecuado Diagrama de Entidad Relación (DER).**
- **Administrar relaciones simples y complejas.**

PROGRAMA ANALITICO

Unidad I: Introducción.

Sistema de bases de datos.
Definición de Bases de datos.
¿Por qué utilizar una base de datos?.
Administración de datos y administración de base de datos.
Ventajas del enfoque de base de datos.
Independencia de datos.

Unidad II: Arquitectura.

Los tres niveles de la arquitectura.
El nivel externo.
El nivel conceptual.
El nivel interno.
Correspondencia.
El administrador de bases de datos.
El Sistema de Administracion de Bases de Datos (DBMS)

Unidad III: El nivel interno.

Introducción.
Acceso a bases de datos.
Manejador de disco.
Manejador de archivo.
Agrupamiento.
Indización.

Unidad IV: Sistemas Relacionales.

Bases de datos relacionales.
El lenguaje SQL.

Unidad V: Definición de datos.

Tablas base.
Create Table, Alter Table, Drop Table.
Indices.

Unidad VI: Manipulación de datos.

Proposiciones DML.
Consultas simples.
Consultas de reunión.
Funciones de agregados.
Características avanzadas.
Operaciones de actualización.
Álgebra Relacional.

Unidad VII: Vistas.

Definición de vistas.
Operaciones de DML sobre vistas.
Independencia lógica de los datos.
Ventajas de las vistas.

Unidad VIII: El Modelo Relacional.

Estructura de Datos Relacional
Dominios.
Relaciones.
Base de datos relacional.

Unidad IX: Reglas de Integridad Relacional.

Claves primarias.
La regla de integridad de las entidades.
Claves foráneas.
La regla de integridad referencial.
Reglas para claves ajenas.

Unidad X: Normalización.

Formas Normales.
Dependencia funcional.
Primera, segunda y tercera formas normales.
Buenas y malas descomposiciones.
Forma normal Boyce/Codd.

Unidad XI: Modelado Semántico.

Enfoque General.
El modelo Entidades/Interrelaciones.
Diagramas de Entidades/Interrelacionales (DER).
Diseño de Bases de Datos con el Modelo de Entidades/Interrelaciones.
Metodología de Trabajo.
Generación de informes.

Unidad XII: Recuperación, Seguridad e integridad.

Recuperación de transacciones.
Recuperación del sistema y de los medios de almacenamiento.
Seguridad e Integridad: Introducción.
Seguridad: Consideraciones Generales .
Seguridad en SQL.
Otros Aspectos de Seguridad.
Integridad: Consideraciones Generales.

Unidad XIII: Ejemplos de aplicación.

Narrativa: Compra De Mercadería
Consultas SQL.
Proyeccion.
Selección (Restriccion).
Distinct.
Recuperación con ordenamiento.
Funciones de Agregados.
Recuperación de datos con Like(como).
Uso de In y Not In,
Empleo de Group By (agrupar por).
Empleo de Having (con):
Utilización de dos tablas.
Consultas de Union.
Utilización de Tres (o mas) Tablas.
Anidamientos In.

Contenidos Procedimentales :

- Investigar sobre ciclo de vida de las Bases de Datos.
- Generar capacidades para identificar problemas y plantear soluciones.
- Implementar soluciones para el procesamiento adecuado de datos e información.
- Desarrollar competencias relacionadas con la gestión y diseño de proyectos.
- Analizar la pertinencia de los proyectos en cuanto al contexto de la organización.
- Realizar adecuadamente la presentación de informes y documentación, reflejando una lectura profunda de la información.
- Usar y adecuar el funcionamiento de los recursos en función de la optimización del procesamiento de la información.
- Manejar herramientas para la construcción de modelos de base de datos a través de la metodología propuesta.
- Implementar los pasos que propone la metodología para la construcción de Base de Datos .

Contenidos Actitudinales :

- Valorar la importancia de organizar los datos a través de Bases de Datos.
- Desarrollar la capacidad crítica para tomar decisiones pertinentes en la resolución de problemas.
- Desarrollar actitudes de colaboración en el trabajo grupal, como así también competencias para escuchar, reflexionar y consensuar criterios.
- Desarrollar la capacidad de acompañar y adaptarse al desarrollo tecnológico del área de Informática.
- Valorar a la informática como una herramienta eficiente en las organizaciones de la sociedad moderna.

Bibliografía Consultada :

- Introducción a los Sistemas de Bases de datos.
  Volumen I.
  C.J. Date.

- Manual de Sql Server
  Database Developer' s Companion
  Microsoft SQL Server.

- Programación de bases de Datos con Visual Basic 6.
  Evangelos Petrouteos.
  ANAYA.

- Manuales Compumagazine.
  Microsoft Visual Basic 6.0
  Baltazar y Mariano Birnios.

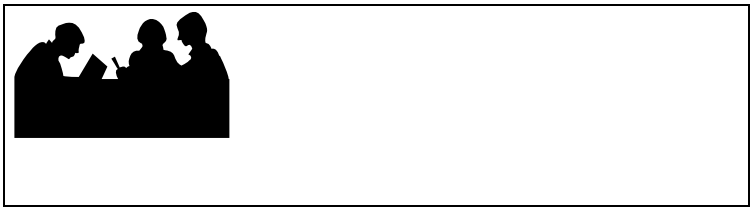
- Sistemas de base de datos, relacionales, orientadas a objetos y distribuidas.
  Cartilla realiza en la Universidad Nacional de Tucumán.

- Bases de datos.
  Cartilla elaborada por la cátedra base de datos de la U.N.Sa.

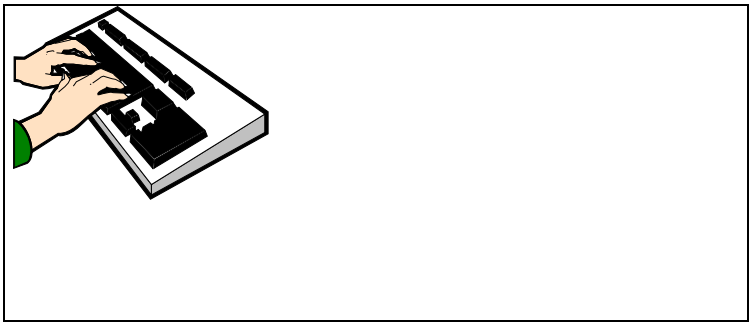
El Manual contiene una serie de indicaciones gráficas que pretendemos te ayuden a identificar con facilidad sus distintos componentes y a organizar las actividades de estudio :

Señala la presencia de actividades que te ayudarán a transferir los conceptos teóricos a la realidad.

Propone la realización de actividades de evaluación de tus aprendizajes.



Indica la presencia de actividades grupales, éstas te ayudarán a desarrollar habilidades para el trabajo en equipo.



Sugiere la realización de prácticas en computadora, éstas te ayudarán a desarrollar habilidades procedimentales para el dominio de los programas estudiados.

UNIDAD I: INTRODUCCIÓN

Cuando una empresa alcanza un cierto nivel de desarrollo, se percata de que el volumen de información que debe manejar es grande y esto dificulta su tratamiento. Considere el simple caso de un almacenero que anota en las libretas el fiado a los clientes. Si dicho almacenero atiende a diez o veinte clientes por hora no hay problema, pero si el flujo aumenta a cien o doscientos por hora, entonces pondrá empleados para atender al público y el se dedicará al llenado y cobro de las libretas. Pero surgirán muchos inconvenientes nuevos:

- Por ejemplo, con los clientes morosos, hay que ver en cada una de las mil o dos libretas si el cliente no entró en mora y proceder al cobro a través de cuotas.
- Cada vez que un cliente quiere abonar, hay que sacarle la deuda que posee en la libreta. Imagine lo que pasaría en los días de pago en donde trescientas o cuatrocientas personas llegarían el mismo día para abonar.

Todos estos casos harían que al almacenero no le alcancen las horas del día para administrar tal cantidad de información.

Pero aún falta lo más preocupante, suponga que el almacenero va a cobrar a domicilio, según el día de pago de cada persona, (por ejemplo si el cliente es empleado provincial irá a cobrarle el día que el noticiero anuncie el pago) entonces tendría que RELACIONAR cada cliente con el trabajo que realiza.

Prontamente se dará cuenta de la necesidad de una base de datos informatizada.

Una base de datos puede considerarse como una especie de archivero electrónico; en donde se almacena un conjunto de archivos de datos computarizados.

Pero, siguiendo este concepto, se podría abrir una hoja de cálculo y almacenar los datos, lo cual funcionaría seguramente para el caso del almacenero.

Para empresas de cierta envergadura, tanto los datos que administran como las relaciones entre ellos hacen necesario contar con un Sistemas de Base de datos.

SISTEMA DE BASES DE DATOS.

Un sistema de bases de datos es básicamente un sistema para **archivar información en una base de datos inserta en un computador**. El propósito es **administrar la información**, logrando que esté disponible, en tiempo y forma, cuando se la solicite.

Los términos “datos” e “información” se manejarán como sinónimos. Algunos autores prefieren hacer una distinción entre ellos, empleando “datos” cuando se refieren a los valores almacenados en realidad en la base de datos e “información” cuando se refieren al significado de esos valores desde el punto de vista del usuario.

Los cuatros componentes principales de un sistema de bases de datos son:

- **Información:** La información en la base de datos deberá estar *integrada* y además será *compartida*.
  - **"Integrada"** significa que la base de datos puede considerarse como una unificación de varios archivos de datos, y que elimina del todo o en parte cualquier redundancia entre ellos. Es decir la información esta diseminada en varios archivos, pero perfectamente relacionados (si tengo dos archivos, uno con los datos de los empleados y otro con los datos de sus hijos, tengo que tener perfectamente indicado qué hijos se coresponden con tales empleados).
  - **"Compartida"** significa que los elementos individuales de información en la base de datos pueden compartirse entre varios usuarios distintos.
- **Equipo:** Los componentes de equipo del sistema son:
  - Los volúmenes de almacenamiento secundario – por lo regular discos magnéticos de cabeza móvil – donde se conservan los datos almacenados, junto con los dispositivos de E/S (entrada y salida) asociados.
  - El procesador o procesadores y la memoria principal asociada que hacen posible la ejecución de los programas del sistema de base de datos.
- **Programas:** Son los software disponibles para poder administrar los datos almacenados en la base de datos. Entre la base de datos física misma y los usuarios del sistema existe un nivel de programas, el *manejador de base de datos* o, el *sistema de administración de base de datos (DBMS)*. **El DBMS maneja todas las solicitudes de acceso a la base de datos formuladas por los usuarios.** Así, una de las funciones generales del DBMS es *distanciar a los usuarios de la base de datos de detalles al nivel del equipo*.El DBMS es definitivamente el componente de software mas importante de todo el sistema, pero no es el único. Entre los demás pueden mencionarse las utilerías, las herramientas para desarrollar aplicaciones, las ayudas para el diseño, los generadores de informes, etcétera.

Investigue acerca de los distintos dbms (access, visual fox, informix, postgres, sql server, mysql, oracle, etc), precios, prestaciones etc.

- **Usuarios:** Se toman en cuenta tres clases de usuarios:
  - En primer término, está el ***programador de aplicaciones***. Esos Programas operan sobre los datos en todas las formas acostumbradas: recuperación de información, inserción de información nueva, eliminación o modificación de datos ya existentes. Pos supuesto, todas estas funciones se llevan a cabo dirigiendo las solicitudes apropiadas al DBMS.

- La segunda clase de usuario es el ***Usuario final***, quien interactúa con el sistema desde una terminal en linea. Un usuario final puede tener acceso a la base de datos a través de una de las aplicaciones en linea o puede utilizar una interfaz incluida como parte integral de los programas del sistema de base de datos. Esas aplicaciones vienen integradas, y no las escriben los usuarios. Casi todos los sistemas incluyen por lo menos una aplicación integrada de ese tipo, a saber, un *procesador de lenguaje de consulta* interactivo, mediante el cual el usuario puede formular mandatos o proposiciones de alto nivel (como SELECT, INSERT, etc) al DBMS.
- La tercera clase de usuario es el *administrador de base de datos*, **DBA (*database administrator*). Es el encargado de ver que datos se almacenarán en la base de datos y cómo serán tratados los mismos.**

DEFINICIÓN DE BASES DE DATOS.

Definición I: Bases de Datos Es un conjunto de información almacenada en memoria auxiliar que permite acceso directo y un conjunto de programas que manipulan esos datos.

Definición II: Base de Datos es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquinas accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no siempre predecible.

Una base de datos contiene:

a. **Datos persistentes.**
b. **Entidades.**
c. **Interrelaciones.**
d. **Propiedades.**

a. Datos persistentes:

Conviene llamar "persistentes" a los datos de una base de datos. Difieren de los datos de entrada y de salida, los resultados intermedios y, cualquier información cuya naturaleza sea hasta cierto punto transitoria.

Una *base de datos* esta constituida por cierto conjunto de datos persistentes. Por Ejemplo en una universidad los datos de los estudiantes son "Datos persistentes".

- "datos de entrada" se refiere a la información que entra al sistema por primera vez (casi siempre desde el teclado de una terminal). Podría dar pie a una modificación de los datos persistentes, pero en principio no forma parte de la base de datos propiamente dicha.
- "datos de salida" se refiere a mensajes y resultados que emanan del sistema.

b. Entidades:

Entidad es : "cualquier objeto acerca del cual deseamos registrar información"

Consideremos el caso de una compañía manufacturera, en donde se desea registrar información referente a los *proyectos* que esta manejando; las partes utilizadas en esos proyectos; los *proveedores* que suministran esas partes; las bodegas donde se almacenan; los *empleados* asignados a los proyectos, etcétera. Los proyectos, partes, proveedores y demás constituyen así las *entidades* básicas acerca de las cuales la empresa necesita registrar información (en el campo de las bases de datos se utiliza ampliamente el término "entidad" para referirse a cualquier objeto distinguible que ha de representarse en la base de datos).

c. Interrelaciones:

Es importante comprender que, ademas de las entidades básicas mismas, existirán también ***interrelaciones*** **que vinculen dichas entidades.** Por ejemplo: cada proveedor suministra ciertas *clases* de partes, y cada *clase* de partes es suministrada por ciertos proveedores. Estas interrelaciones son *bidireccionales*.

Una interrelacion puede considerarse como una entidad por si misma**.** Por ejemplo si tengo la entidad mercadería y la entidad clientes, entonces existe una relación, la cual es: clientes **adquieren** mercaderías (la interrelación "adquieren" relaciona dos entidades: clientes y mercaderías). La información respecto a la fecha en que el cliente XX adquiere la mercadería YY tiene que ser almacenada en la interrelación "adquieren", por lo que dicha interrelación es también una entidad en sí misma.

d. Propiedades:

Las entidades tienen *propiedades*. Por ejemplo, los proveedores tienen *localidades*; las partes tienen *pesos*, dichas propiedades deben estar representadas tambien en la base de datos.

Nota**:** una propiedad podria ser por naturaleza muy sencilla, o bien poseer una estructura interna de complejidad arbitraria. Por ejemplo una bodega podría tener una propiedad de "Planta", que podria ser en extremo compleja, incluyendo quizás todo un plano arquitectónico junto con textos descriptivos del mismo. Se supondrá en general que todas las propiedades son "simples". Como ejemplos pueden mencionarse los números, cadenas, fechas, horas, etcétera.

Elabore una definición de su propia autoría sobre el concepto : Base de Datos.

¿POR QUÉ UTILIZAR UNA BASE DE DATOS?

Las ventajas de un sistema de base de datos respecto de un sistema manual son:

- Es compacto: no hacen falta archivos de papeles.
- Es rápido: la máquina puede obtener y modificar datos con mucha mayor velocidad que un ser humano.
- Es menos laborioso: se elimina gran parte del tedio de mantener archivos a mano.
- Es actual: Se dispone en cualquier momento de información precisa y al dia.

Existe una ventaja adicional apabullante: ***el sistema de base de datos ofrece a la empresa un control centralizado de su información*****.** En una empresa, sin un sistema de base de datos, los datos estarán dispersos y con seguridad dificiles de controlar en cualquier forma sistemática.

ADMINISTRACIÓN DE DATOS Y ADMINISTRACIÓN DE BASE DE DATOS.

En una empresa con un sistema de base de datos, existe una persona identificable con esta responsabilidad central sobre los datos. Ese individuo es el ***administrador de datos*** (abreviado a veces DA, data administrator). La labor del administrador es deducir cuales datos deben almacenarse en la base de datos, para mantener y manejar los datos una vez almacenados. El administrador de datos es una persona de la empresa que comprende la naturaleza de los negocios de la misma. Generalmente es un gerente no un técnico.

El técnico responsable de poner en práctica las decisiones del administrador de datos es el administrador de *base* de datos (DBA). La tarea del DBA es crear la base de datos en si y poner en vigor los controles técnicos necesarios para apoyar las políticas dictadas por el administrador de datos. El DBA se encargará también de garantizar el funcionamiento adecuado del sistema y de proporcionar otros servicios de índole técnico relacionados.

VENTAJAS DEL ENFOQUE DE BASE DE DATOS.

- Es posible **disminuir la redundancia**: En los sistemas sin bases de datos cada aplicación tiene sus propios archivos privados.
- Es posible **evitar inconsistencia** (hasta cierto punto): por ejemplo el hecho de que el empleado E3 trabaja en el departamento D8, está representado por dos entradas distintas en la base de datos almacenada. (tendríamos la entrada E3,D8 y la entrada D8,E3). Supongamos tambien que el DBMS no esta consciente de esta duplicación (es decir, la redundancia no esta controlada).
- Es posible **compartir datos**.
- Es posible **hacer cumplir las normas**: Al tener un control centralizado de la base de datos, el DBA (siguiendo las indicaciones del

administrador de datos) puede garantizar las observancia de todas las normas aplicables para la representacion de los datos.

- Es posible **aplicar restricciones de seguridad**: Al tener jurisdicción completa sobre la base de datos, el DBA puede asegurar que el acceso a la base de datos sea solo a través de los canales apropiados y, por tanto, puede definir las verificaciones de seguridad por realizar cuando se intente acceder a información delicada.
- Es posible **mantener la integridad**: El problema de la integridad radica en asegurar que la información de la base de datos sea correcta.
- Es posible **equilibrar requerimientos opuestos**: Al conocer los requerimientos generales de la empresa –en contraste con los requerimientos de cualquier usuario individual– el DBA (como siempre bajo la dirección del administrador de datos) puede estructurar el sistema con miras a proporcionar un servicio general "óptimo para la empresa".

Es menester agregar a la lista un punto mas "***independencia de los datos***".

INDEPENDENCIA DE DATOS.

La independencia de datos,puede definirse como la inmunidad de las aplicaciones ante los cambios en la estructura de almacenamiento y en la técnica de acceso, lo cual implica que las aplicaciones en cuestión no dependen de una estructura de almacenamiento o una técnica de acceso específicas.

En un sistema de bases de datos no es nada recomendable tener aplicaciones dependientes de los datos, al menos por las razones siguientes :

- Cada aplicación requiere una vista diferente de los mismos datos.
- El DBA debe tener libertad para modificar la estructura de almacenamiento o la técnica de acceso (o las dos cosas) para adaptarlas a cambios en los requerimientos, sin tener que modificar las aplicaciones ya existentes.

Se dice que una aplicación es *dependiente de los datos* cuando es imposible alterar la estructura de almacenamiento (la organización física de los datos) o la técnica de acceso (la forma de acceder a ellos) sin afectar, quizás gravemente, a la aplicación.

a. Campo, registro y archivo almacenado:

Comenzaremos por definir tres términos : campo almacenado, registro almacenado y archivo almacenado

- Un ***campo almacenado*** es la unidad más pequeña de información almacenada que recibe un nombre.
- Un **registro *almacenado*** es un conjunto de campos almacenados, relacionados entre si, que cuenta con su propio nombre.

- Por último, un ***archivo almacenado*** es el conjunto (con nombre) de todas las ocurrencias de un tipo de registro almacenado.

Por ejemplo, los datos de un alumnos pueden estar compuestos por los atributos: Nombre, Dirección, DNI. Cada atributo es un campo, el conjunto de campos forma un registro. Por ejemplo el nombre ”pepe” es un campo, y el conjunto ”pepe”,”Sarmiento 585”,”13456234” es un registro, representa un alumno. Varios de estos registros, o sea, varios alumnos formarán un archivo.

b. Sistemas Relacionales:

Casi todos los productos de bases de datos desarrollados en años recientes se basan en lo que se conoce como el *enfoque relacional.*

En pocas palabras, un sistema relacional es aquel en el cual:

1) El usuario percibe los datos en forma de tablas (y solo como tablas).
2) Los operadores al alcance del usuario generan tablas nuevas a partir de las existentes. Asi, existirá un operador para extraer un subconjunto de filas de una tabla determinada, y otro para extraer un subconjunto de las columnas (y, por supuesto, tanto un subconjunto de filas como un subconjunto de columnas de una tabla pueden considerarse a su vez como tablas ellas mismas).

Nota: En el desarrollo del modelo relacional se especificarán claramente estos conceptos.

ACTIVIDAD Nº 1:

- **Responda el siguiente cuestionario:**

1) ¿Qué es un Sistema de Base de Datos?.
2) Defina los cuatros componentes principales de un sistema de bases de datos.
3) ¿Qué significa que la información sea integrada y compartida.?
4) ¿Cuál es el componente de software mas importante de todo el sistema.?
5) Defina Base de Datos.
6) Defina Datos persistentes.
7) Defina Entidades.
8) Defina Interrelaciones.
9) Defina Propiedades.
10) ¿Cuál es la ventaja de utilizar una base de datos?.
11) ¿Cuál es la diferencia entre un DBA y un DA?.
12) ¿Qué significa independencia de datos?

Unidad II: Arquitectura

LOS TRES NIVELES DE LA ARQUITECTURA.

La arquitectura ANSI/SPARC se divide en tres *niveles*, denominados **niveles internos, conceptual y externo**. En términos generales:

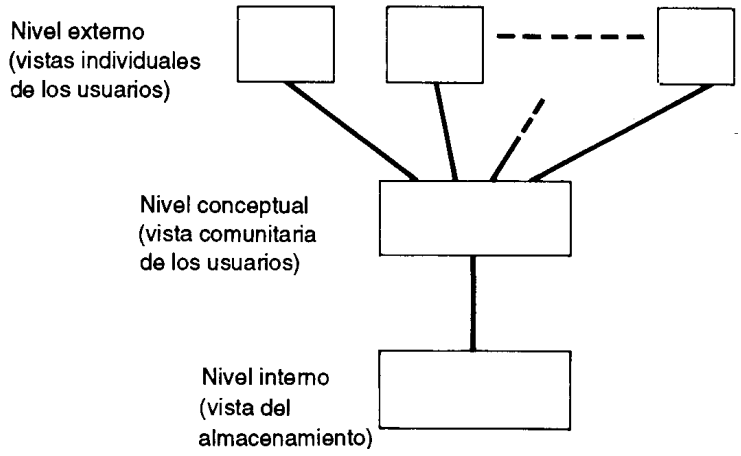


Figura 2.1 Los tres niveles de la Arquitectura

- El nivel *interno* es el más cercano al almacenamiento físico, es decir, es el que se ocupa de la forma como se almacenan físicamente los datos.
- El nivel *externo* es el más cercano a los usuarios, es decir, es el que se ocupa de la forma como los usuarios individuales perciben los datos.
- El nivel *conceptual* es un "nivel de mediación" entre los otros dos.

Pueden existir muchas "vistas externas" distintas, pero una sóla "vista conceptual".

A la mayoría de los usuarios no les interesará toda la base de datos, sino sólo una porción limitada de ella.

Hay sólo una "vista interna", la cual representará a toda la base de datos tal como está almacenada físicamente.

Puede ser útil indicar en forma concisa cómo se percibirían normalmente los tres niveles de la arquitectura en un sistema relacional:

- El nivel conceptual *será* relacional, en el sentido de que los objetos visibles en ese nivel serán tablas relacionales.
- Una vista externa determinada casi siempre será relacional también.
- Es casi seguro que el nivel interno "no será relacional", porque los objetos en ese nivel no serán por lo regular sólo tablas relacionales (almacenadas), sino que serán objetos similares a los encontrados en el nivel interno de otros tipos de sistemas (a saber, registros almacenados, apuntadores, índices, dispersiones, etc.). De hecho, la teoría relacional

como tal nada tiene que decir acerca del nivel interno: se ocupa de la forma como el *usuario* percibe la base de datos.

EL NIVEL EXTERNO

El nivel externo es el del usuario individual. Los usuarios pueden ser o bien programadores de aplicaciones o usuarios de terminales en línea. (El DBA deberá interesarse también en los niveles conceptual e interno).

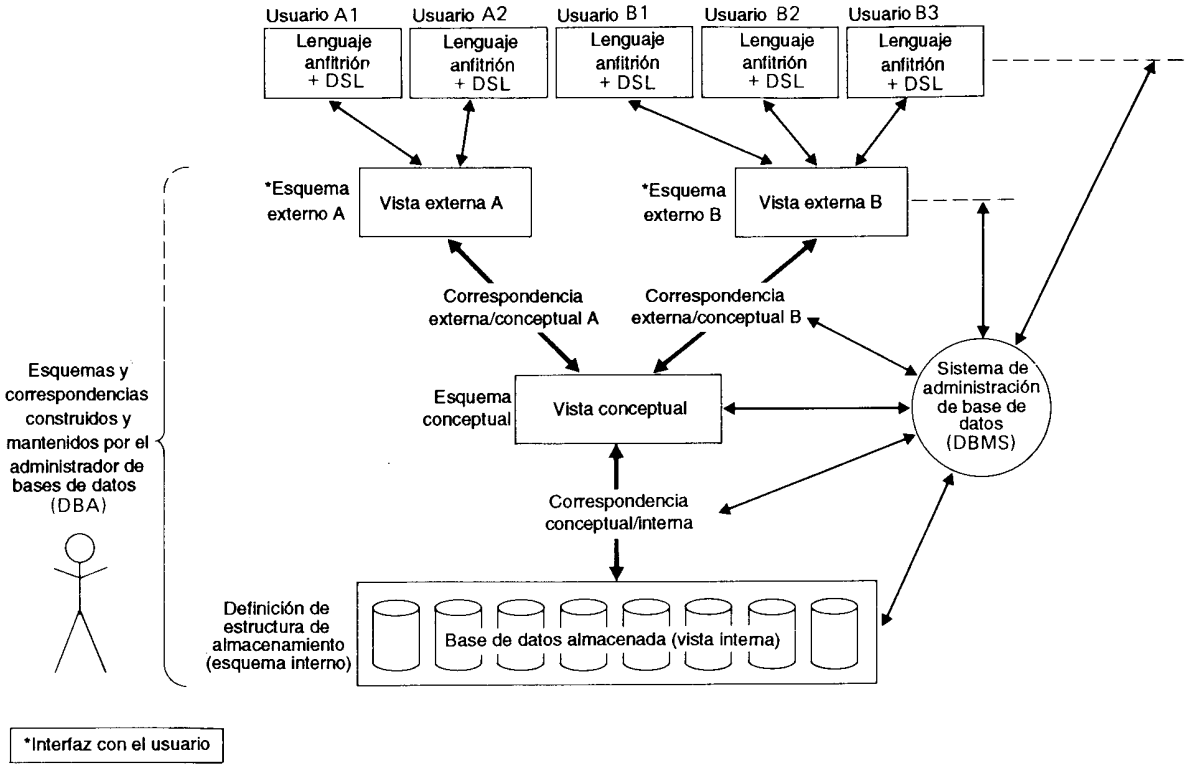


Fig. 2.3 Arquitectura detallada del sistema

Investigue acerca de los lenguajes existentes para cada tipo de usuario.

Cada usuario dispone de un *lenguaje*:

- En el caso del programador de aplicaciones, dicho lenguaje será o bien uno de los lenguajes de programación convencionales, o bien un lenguaje propio (“de cuarta generación”) específico para el sistema en cuestión.

- Para el usuario final, será o bien un lenguaje de consulta, o algún lenguaje de aplicación especial.

Todos esos lenguajes deben incluir un *sublenguaje de datos* que se ocupe de manera específica de los objetos y operaciones de la base de datos. **El sublenguaje de datos (abreviado DSL**, *data sublenguaje*, en la figura 2.3) **está *embebido* (o *inmerso*)** dentro del *lenguaje anfitrión* correspondiente.

Nota: Un sublenguaje de datos es el lenguaje SQL. En casi todos estos sistemas, el lenguaje SQL puede utilizarse de manera interactiva (como lenguaje de consulta independiente) o embebido en otros lenguajes (Vbasic, por ej.).

Aunque es conveniente, al estudiar la arquitectura, distinguir entre el sublenguaje de datos y el lenguaje anfitrión que lo contiene, en realidad los dos pueden considerarse *idénticos* por lo que toca al usuario. Se dice que ambos están *fuertemente acoplados*. Si se pueden separar con nitidez y facilidad, se dicen que están *débilmente acoplados*.

En principio, cualquier sublenguaje de datos es en realidad una combinación de por lo menos dos lenguajes subordinados: un ***lenguaje de definición de datos* (DDL**, *data definition language*). Con el cual es posible **definir o declarar los objetos de la base de datos**, y un ***lenguaje de manipulación de datos* (DML**, *data manipulation language*) con el que **es posible manipular o procesar dichos objetos**.

Al usuario individual sólo le interesará una porción de la base de datos total. El término que utiliza ANSI/SPARC para la vista individual de un usuario es *vista externa*. Así, **una vista externa es el contenido de la base de datos tal como lo percibe algún usuario determinado**, para ese usuario la vista externa es la base de datos.

Por ejemplo si tenemos una base de datos del “Colegio Del Milagro”, las entidades posibles (archivos) serán: Alumnos, Materia, Profesores, Personal Administrativo, Personal Directivos. Además de las respectivas relaciones entre ellas. Si el usuario final es un alumno, a éste no le interesarán las entidades Personal Administrativo ni Personal Directivo. Para el la Base de Datos estará compuesta por las otras entidades con sus respectivas relaciones.

En general, una vista externa se compone de varias ocurrencias de varios tipos de registros externos. Un registro externo no es por fuerza idéntico a un registro almacenado. Por ejemplo, siguiendo con la base de datos “Colegio del Milagro”, el usuario final alumno solamente percibirá de la entidad Profesores la propiedad “Nombre”, aunque en realidad dicha entidad también posee la propiedad “Direccion”, “Teléfono” etc.

El sublenguaje de datos del usuario se define en términos de registros externos, por ejemplo, una operación de “consulta” en DML extraerá una ocurrencia de un registro externo (o quizá varias) pero no una ocurrencia de un registro almacenado.

Debe haber una definición de la correspondencia entre el esquema externo y el esquema conceptual subyacente.

EL NIVEL CONCEPTUAL

La vista conceptual es una representación de toda la información contenida en la base de datos, puede ser muy diferente de la forma en como percibe los datos cualquier usuario individual. **La vista conceptual debe ser un panorama de los datos "tal como son".**

La vista conceptual se compone de varias ocurrencias de varios tipos de registro conceptual. Un registro conceptual no es por necesidad idéntico a un registro externo, por un lado, ni a un registro almacenado, por el otro.

La vista conceptual se define mediante un *esquema conceptual*, se escribe utilizando otro lenguaje de definición de datos, el *DDL conceptual.* Si ha de lograrse la independencia de los datos, esas definiciones en DDL conceptual no deberán implicar consideraciones de estructura de almacenamiento o de técnica de acceso. Deben ser *sólo* definiciones de contenido de información. Por tanto, en el esquema conceptual no debe aludirse a representaciones de campos almacenados, indización o cualquier otro detalle de almacenamiento y acceso. Si el esquema conceptual se hace en verdad independiente de los datos, entonces los esquemas externos, definidos en términos del esquema conceptual, serán por fuerza también independientes de los datos.

La vista conceptual es una vista del contenido total de la base de datos, y el esquema conceptual es una definición de esa vista. En casi todos los sistemas existentes el "esquema conceptual" no es mucho más que una simple unión de todos los esquemas externos individuales, con la posible adición de algunas verificaciones sencillas de integridad y seguridad. Con todo, parece evidente que los sistemas del futuro llegarán a mantener niveles conceptuales mucho más complejos.

EL NIVEL INTERNO

La vista interna **es una representación de bajo nivel de toda la base de datos**. Se compone de varias ocurrencias de varios tipos de *registro interno*, llamado registro *almacenado*. La vista interna, por tanto, todavía está a un paso del nivel físico, ya que no maneja registros *físicos* (llamados también *páginas* o *bloques*), ni otras consideraciones específicas de los dispositivos como son los tamaños de cilindros o de pistas.

La vista interna se define mediante el *esquema interno*, el cual no sólo define los diversos tipos de registros almacenados sino también especifica qué índices hay, cómo se representan los campos almacenados, en qué secuencia física se encuentran los registros almacenados, etc. El esquema interno se escribe con otro lenguaje más de definición de datos, el *DDL interno*.

CORRESPONDENCIA

La correspondencia *conceptual/interna* es la que existe entre la vista

conceptual y la base de datos almacenada. Especifica cómo se representan los registros y campos conceptuales en el nivel interno. Los efectos de las alteraciones deberán aislarse por debajo del nivel conceptual, a fin de conservar la independencia de los datos.

La correspondencia *externa/conceptual* es la que existe entre una determinada vista externa y la vista conceptual. Las diferencias que pueden existir entre estos dos niveles son similares a las que pueden existir entre la vista conceptual y la base de datos almacenada. Por ejemplo, los campos pueden tener distintos tipos de datos, los nombres de los campos y los registros pueden diferir, pueden combinarse varios campos conceptuales para formar un sólo campo externo (virtual), etcétera. Puede existir cualquier cantidad de vistas externas; cualquier número de usuarios pueden compartir una determinada vista externa; puede haber traslados entre vistas externas distintas.

Algunos sistemas permiten expresar la definición de una vista externa en términos de otras (correspondencia *externa/externa*). Los sistemas relacionales en particular casi siempre permiten hacer esto.

EL ADMINISTRADOR DE BASES DE DATOS

El administrador de datos (DA) toma las decisiones estratégicas y de política con respecto a la información de la empresa, y el administrador de *bases* de datos (DBA) es quien proporciona el apoyo técnico necesario. En general, **las funciones del DBA** serán las siguientes:

- **Definir el esquema conceptual**:
  Cuando el administrador de datos decide el contenido de la base de datos en un nivel abstracto, el DBA crea a continuación el esquema conceptual correspondiente, empleando el DDL conceptual. El DBMS utilizará la versión objeto (compilada) de ese esquema para responder a las solicitudes de acceso. La versión fuente (sin compilar) servirá como documento de referencia para los usuarios del sistema.
- **Definir el esquema interno:**
  El DBA debe decidir también cómo se representará la información en la base de datos almacenada. A este proceso suele llamarse diseño *físico* de la base de datos. El DBA deberá crear la definición de estructura de almacenamiento correspondiente (es decir, el esquema interno) valiéndose del DDL interno. Además, deberá definir la correspondencia pertinente entre los esquemas interno y conceptual. Las dos funciones (crear el esquema, definir la correspondencia) deberán poder separarse con nitidez. Al igual que el esquema conceptual, el esquema interno y la correspondencia asociada, existirán tanto en la versión fuente como en la versión objeto.
- **Vincularse con los usuarios:**
  El DBA debe encargarse de la comunicación con los usuarios, garantizar la disponibilidad de los datos que requieren y escribir —o ayudar a los usuarios a escribir— los esquemas externos necesarios, empleando el DDL externo aplicable. Además, será preciso definir la

correspondencia entre cualquier esquema externo y el esquema conceptual. El esquema y la correspondencia deberán poder separarse con claridad. Cada esquema externo y la correspondencia asociada existirán en ambas versiones, fuente y objeto.

- **Definir las verificaciones de seguridad e integridad:**
  Como ya se explicó, las verificaciones de seguridad y de integridad pueden considerarse parte del esquema conceptual. El DDL conceptual incluirá (o debería incluir) los medios para especificar dichas verificaciones.
- **Definir procedimientos de respaldo y recuperación:**
  Resulta esencial poder parar los datos implicados con un mínimo de retraso y afectando lo menos posible al resto del sistema. El DBA debe definir y poner en práctica un plan de recuperación adecuado que incluya, por ejemplo, una descarga o "vaciado" periódico de la base de datos en un medio de almacenamiento de respaldo y procedimientos para cargar otra vez la base de datos a partir del vaciado más reciente cuando sea necesario.
- **Supervisar el desempeño y responder a cambios en los requerimientos:**
  Es responsabilidad del DBA organizar el sistema de modo que se obtenga el desempeño que sea "mejor para la empresa" y realizar los ajustes apropiados cuando cambien los requerimientos. Cualquier modificación del nivel de almacenamiento físico (interno) del sistema debe ir acompañado por el cambio respectivo en la definición de la correspondencia con el nivel conceptual, pues sólo así podrá permanecer constante el esquema conceptual.

EL SISTEMA DE ADMINISTRACION DE BASES DE DATOS (DBMS)

El *sistema de administración de la base de datos* (DBMS) es por supuesto el **conjunto de programas que maneja todo acceso a la base de datos**. Conceptualmente, lo que sucede es lo siguiente:

1) Un usuario solicita acceso, empleando algún sublenguaje de datos determinado (por ejemplo, SQL).
2) El DBMS interpreta esa solicitud y la analiza.
3) El DBMS inspecciona, en orden, el esquema externo de ese usuario, la correspondencia externa/conceptual asociada, el esquema conceptual, la correspondencia conceptual/interna, y la definición de la estructura de almacenamiento.
4) El DBMS ejecuta las operaciones necesarias sobre la base de datos almacenada.

Como ejemplo, considere lo que se necesita para extraer una cierta ocurrencia de registro externo. En general, se requerirán campos de varias ocurrencias de registro conceptual. Cada ocurrencia de registro conceptual, a su vez, puede requerir campos de varias ocurrencias de registro almacenado. Así, en teoría al menos, el DBMS debe obtener primero todas las ocurrencias de registro almacenado requeridas, construir enseguida las ocurrencias de registro conceptual necesarias, y después construir la ocurrencia de registro

externo requerida. En cada etapa, quizá se requieran conversiones de tipos de datos u otras.

Examinemos ahora las **funciones del DBMS** con un poco más de detalle. Dichas funciones incluirán por lo menos las siguientes.

- **Definición de datos:**

  El DBMS debe ser capaz de aceptar definiciones de datos (esquemas externos, el esquema conceptual, el esquema interno, y todas las correspondencias asociadas) en versión fuente y convertirlas en la versión objeto apropiada. El DBMS debe incluir componentes procesadores de lenguajes para cada uno de los diversos lenguajes de definición de datos (DDL).

- **Manipulación de datos:**

  El DBMS debe ser capaz de atender las solicitudes del usuario para extraer, y quizás poner al día, datos que ya existen en la base de datos, o para agregar en ella datos nuevos.

  En general, **las solicitudes en DML pueden ser "planeadas" o "no planeadas":**

  - Una solicitud planeada es aquella cuya necesidad se previó mucho tiempo antes de que tuviera que ejecutarse por primera vez. El DBA habrá afinado con toda probabilidad el diseño físico de la base de datos a fin de garantizar un buen desempeño para estas solicitudes.
  - Una solicitud no planeada, en cambio, es una consulta *ad hoc, es* decir, una solicitud cuya necesidad no se previó, sino que surgió de improviso.

  Las solicitudes planeadas son características de las aplicaciones "operacionales" o "de producción"; las no planeadas son representativas de las aplicaciones de "apoyo a decisiones". Las solicitudes planeadas casi siempre se originan en programas de aplicación previamente escritos, en tanto que las solicitudes no planeadas, por definición, se emitirán de manera interactiva.

- **Seguridad e integridad de los datos:**

  El DBMS debe supervisar las solicitudes de los usuarios y rechazar los intentos de violar las medidas de seguridad e integridad definidas por el DBA.

- **Recuperación y concurrencia de los datos:**

  El DBMS debe cuidar del cumplimiento de ciertos controles de recuperación y concurrencia.

- **Diccionario de datos:**

  El DBMS debe incluir una función de *diccionario de datos*. Una base de datos del sistema, no del usuario. El contenido del diccionario puede considerarse como **"datos acerca de los datos"**, reciben en ocasiones el nombre de "metadatos". Se almacenarán físicamente todos los diversos esquemas y correspondencias (externos, conceptuales, etc.) tanto en sus versiones fuente como en las versiones objeto. Un diccionario completo incluirá también referencias cruzadas para indicar, por ejemplo, cuáles programas usan cuáles partes de la base de datos, cuáles usuarios requieren cuáles

informes, qué terminales están conectadas al sistema y cosas por el estilo. Deberá ser posible consultar el diccionario igual que cualquier otra base de datos de modo que se pueda saber, por ejemplo, cuáles programas o usuarios podrían verse afectados por alguna modificación propuesta para el sistema.

- **Desempeño:**

  El DBMS deberá ejecutar todas las funciones recién identificadas en la forma más eficiente posible.

Como conclusión, El DBMS constituye la *Interfaz* entre el *usuario* y el sistema de bases de datos. La interfaz del usuario puede definirse como una frontera del sistema, más allá de la cual todo resulta invisible para el usuario. Por definición, entonces, la interfaz del usuario está en el nivel *externo*.

Los DBMS modernos suelen ofrecer al Administrador de la Base de Datos un interfaz visual, de modo que pueda definir el esquema conceptual y externo, de un modo sencillo y gráfico. Luego el DBMS se encarga de convertir los requerimientos del usuario en las sentencias correspondientes del DDL conceptual e interno correpondiente.

ACTIVIDAD Nº 2:

- **Responda el siguiente cuestionario:**

1) Busque en internet el significado de la arquitectura ANSI/SPARC.
2) ¿Cúales son los tres niveles de la arquitectura ANSI/SPARC?.
3) ¿Qué es el Nivel Externo?.
4) ¿Qué significa que El sublenguaje de datos está *embebido* (o *inmerso*) dentro del *lenguaje anfitrión* correspondiente?.
5) ¿Qué significa y para que sirve el DDL y el DML?
6) ¿Qué es el Nivel Conceptual?.
7) ¿Qué es el Nivel Interno?.
8) ¿Qué tipo de Correspondencia tienen los tres niveles de la arquitectura ANSI/SPARC?.
9) ¿Cuáles son las funciones del DBA?
10) ¿Quién administra el conjunto de programas que maneja todo acceso a la base de datos?. ¿Conceptualmente, cómo lo hace?.
11) ¿Cúales son las funciones del DBMS?.
12) ¿Dónde se utilizan las solicitudes en DML “planeadas” y las “no planeadas”?.

Unidad III: El nivel interno

INTRODUCCIÓN

El nivel interno de una base de datos es el que se ocupa de la forma como están almacenados en realidad los datos. Las bases de datos casi siempre se almacenan en medios de accesos directo. Utilizaremos el término “disco” para referirnos en forma genérica a todos esos medios. También supondremos, que la base de datos completa puede almacenarse en solo un disco.

Los tiempos de acceso representativos van desde cerca de 400 milisegundos o más para un disco flexible en un microcomputador hasta unos 30 milisegundo o menos para un disco “rápido” grande en un microcomputador de gran tamaño; el acceso a la memoria principal será con toda probabilidad por lo menos cuatro o cinco órdenes de magnitud más rápido que el acceso a disco en un sistema dado. Por tanto, **un objeto prioritario de desempeño en sistemas de bases de datos es *reducir al mínimo e número de accesos a disco*** (E/S a disco). Esta unidad se ocupa de las técnicas para lograr ese objetivo, es decir, técnicas para organizar los datos almacenados en el disco de manera tal que un elemento de información requerido, digamos un registro almacenado solicitado, se puede localizar con un mínimo de E/S.

Cualquier organización de los datos en el disco se denomina *estructura de almacenamiento*. No existe una sola estructura óptima para todas las aplicaciones. Un buen sistema deberá poder utilizar varias estructuras distintas, y deberá ser posible cambiar la estructura de almacenamiento de una porción determinada cuando varíen o se comprendan mejor los requerimientos de desempeño.

Elegir una representación de almacenamiento apropiada para una cierta base de datos se conoce como ***diseño físico de bases de datos***. Conviene subrayar que, al menos en el caso de una base de datos grande y compleja, el diseño físico puede ser una tarea nada trivial.

Los ejemplos que veremos se refieren a la base de datos de la figura 3.1, en ella se presenta una vista relacional. La tabla S, representa a los proveedores, la tabla P, representa a las partes, y la tabla SP, representa a los envíos de partes hechos por proveedores. Cada proveedor *tiene un número de proveedor* único (P#). Además, supondremos que no pueden existir, al mismo tiempo, dos envíos del mismo proveedor y de la misma parte, de modo que cada envío tiene una combinación única de número de proveedor/número de parte. En la terminología relacional, S#, P#, y el campo compuesto (S#, P#) (en la tabla SP) se denominan ***claves primarias***. El término Tabla, hace referencia a los archivos, en la unidad referida al modelo relacional explicaremos detalladamente este concepto.

ACCESO A BASE DE DATOS.

los principios de acceso a BD pueden explicarse a grandes rasgos de la siguiente manera:

1) En primer término, el DBMS decide cual registro almacenado se necesita, y pide al manejador de archivos que extraiga ese registro. En la práctica puede ser necesario extraer un conjunto de varios registros y examinarlos dentro de la memoria principal para encontrar el que se busca. En principio, esto solo significa que la secuencia de paso 1 a 3 deberá repetirse para cada uno de los registros almacenados de ese conjunto.
2) A su vez, el manejador de archivo decide cuál página contiene el registro deseado, y pide al manejador de disco que lea esa página. La página es la unidad de E/S, es decir; la cantidad de datos transferido entre el disco y la memoria principal en un solo acceso a disco. Los tamaños usuales de la pagina son de 1k, 2k o 4k bytes (k=1024).
3) Por útimo, el manejador de disco determina la localización física de la página deseada en el disco, y realiza la operación de E/S necesaria. Habrá ocasiones en que la página requerida esté ya en un área de almacenamiento temporal en la memoria principal como resultado de una lectura anterior, en cuyo caso resulta obvio que no será necesario leerla otra vez.

Tabla S

S#	SNOMBRE	SITUACIÓN	CIUDAD
S1	Salazar	20	Londres

Tabla P

P#	SNOMBRE	COLOR	PESO	CIUDAD
P1	Tuerca	Rojo	12	Londres

Tabla SP

S#	P#	CANT
S1	P1	300

FIGURA Nº 3.1 La base de datos de proveedores y partes

En términos generales, el DBMS percibe la base de datos como un conjunto de registros almacenados, y el manejador de archivos apoya esta percepción; el manejador de archivos a su vez, percibe la base de datos como un conjunto de páginas, y el manejador de disco apoya esa percepción; el manejador de disco percibe el disco “como es en realidad”.

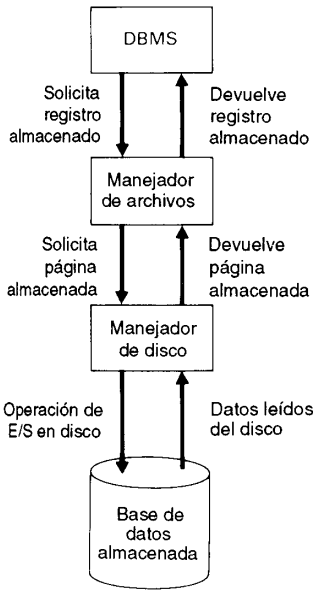


Fig 3.2 El DBMS, el manejador de archivos y el manejador de disco

MANEJADOR DE DISCO.

Es un componente del sistema operativo. Es el **encargado de todas las operaciones físicas de E/S**, es evidente que necesita conocer las direcciones físicas en el disco.

Para el manejador de archivos, el disco es solamente una conexión lógica de conjunto de páginas, cada uno de los cuales se compone de un grupo de páginas de tamaño fijo.

Cada conjunto de páginas se identifica mediante un identificador de conjuntos de páginas único.

Cada página, a su vez, se identifica mediante un número de página que es único dentro del disco; los diferentes conjuntos de páginas no se trasladan.

El manejador de disco entiende y mantiene la correspondencia entre números de página y direcciones físicas en el disco. La ventaja principal de este arreglo es que todas las instrucciones codificadas especificas para un dispositivo pueden aislarse dentro de un solo componente del sistema, el manejador de disco y todos los componentes de nivel mas alto, sobre todo el manejador de archivo, pueden ser así independientes de los dispositivos.

Entre las opciones que puede realizar el manejador de disco con los conjuntos de páginas están las siguientes:

- Leer la página *p* del conjunto de páginas *c*.
- Reemplazar la página *p* dentro del conjunto de páginas *c.*

- Añadir una página nueva al conjunto de páginas *c* (es decir obtener una página vacía del conjunto de páginas del espacio libre y devolver el nuevo número de página *p*).
- Eliminar la página *p* del conjunto de páginas *c* (es decir, volver la página *p* al conjunto de páginas del espacio libre).

MANEJADOR DE ARCHIVOS

El manejador de archivos utiliza los recursos recién descriptos del manejador de disco de manera tal que su usuario (el DBMS) pueden percibir el disco como un conjunto de archivos almacenados, un archivo almacenado es el conjunto de todas las ocurrencias de un tipo de registro almacenado. Cada conjunto de páginas contendrá uno o más archivos almacenados. Nota: El DBMS si necesita saber que existen conjunto de páginas, aunque no se encarga de manejarlos en detalles.

Cada archivo almacenado se identifica mediante un nombre de archivo o identificador de archivo, único por lo menos dentro del conjunto de páginas que lo contiene, y cada registro almacenado a su vez, se identifica mediante un número de registro o identificador de registro único al menos dentro del archivo almacenado que lo contiene.

En algunos sistemas el manejador de archivos es un componente del sistema operativo subyacente, en otros esta empacado con el DBMS. Muchas veces el manejador de archivos de uso general ofrecido por el sistema operativo no resulta ideal para los requerimientos de la "aplicación" de uso especial que es el DBMS.

Entre las operaciones que puede realizar el manejador de archivo con los archivos almacenados están las siguientes:

- Leer el registro almacenado r del archivo almacenado a.
- Reemplazar el archivo almacenado r dentro del archivo almacenado a.
- Añadir al archivo almacenado a un nuevo registro y devolver el nuevo identificador de registro r.
- Eliminar el registro almacenado r del archivo almacenado a.
- Crear un nuevo archivo almacenado a.
- Destruir el archivo almacenado a.

AGRUPAMIENTO

La idea básica del agrupamiento es **procurar almacenar juntos físicamente los registros que tienen una relación lógica entre sí** (y que por eso se utilizan con frecuencia al mismo tiempo). Vamos a suponer que el registro almacenado de acceso más reciente es r1, y que a continuación se requerirá el registro almacenado r2. Supongamos también que r1 esta almacenado en la página p1 y r2 en la página p2. Entonces:

1) Si p1 y p2 son una misma, entonces el acceso a r2 no requerirá E/S física alguna, porque la página deseada p2 ya estará en almacenamiento temporal dentro de la memoria principal.
2) Si p1 y p2 son distintas pero cercanas físicamente, entonces el acceso a r2 requerirá una E/S física, pero el tiempo de búsqueda implicado en esa E/S será pequeño, porque las cabezas de lectura/grabación ya estarán cerca de la posición deseada.

Desde luego, un archivo (o conjunto de archivos) determinado puede agruparse físicamente en una y solo una forma a la vez.

El DBMS puede hacer posible el agrupamiento, tanto dentro de un archivo como entre varios archivos, si almacena los registros que tienen una relación lógica entre sí en la misma página cuando sea posible y en páginas adyacentes cuando no lo sea. (por esta razón el DBMS debe saber acerca de las páginas y no solo de los archivos almacenados). El manejador de archivos, a su vez, hará lo posible por lograr que dos páginas adyacentes lógicamente estén contiguas físicamente en el disco. El DBMS solo puede saber cual agrupamiento se requiere si el administrador de la base de datos es capaz de indicárselo. Un buen DBMS deberá permitir al DBA especificar diferentes clases de agrupamiento para distintos archivos. También deberá permitir la modificación del agrupamiento de un archivo determinado (o de un grupo de archivos) en caso de cambiar los requerimientos de desempeño. Desde luego, si ha de lograrse la independencia de datos, ninguno de estos cambios en el agrupamiento físico deberá requerir cambios correspondientes en los programas de aplicación.

El manejador de disco casi siempre asigna páginas a los conjuntos (y las libera de ellos), no una por una como se sugiere en el ejemplo, sino más bien en grupos físicamente contiguos o “extensiones” de (digamos) 64 páginas a la vez.

La “Pagina cero”, contiene por lo regular una lista de los conjuntos de páginas que existen de momento en el disco, junto con un apuntador a la primera página de cada conjunto.

CONJUNTO DE PAGINAS Y ARCHIVOS

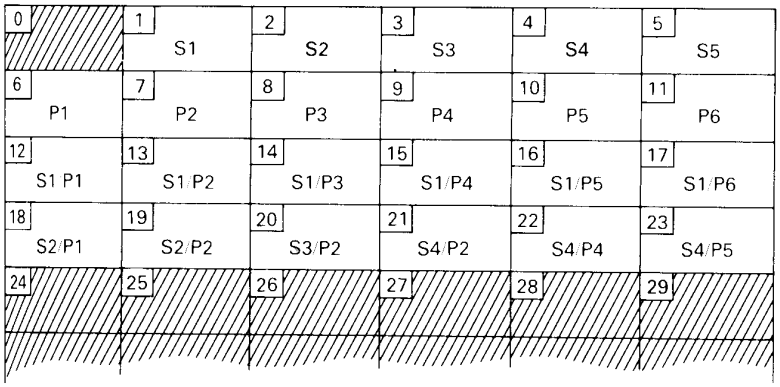


Fig. 3.3 Disposición del disco después de la creación y carga inicial de la base de datos de proveedores y partes de la figura 3.1

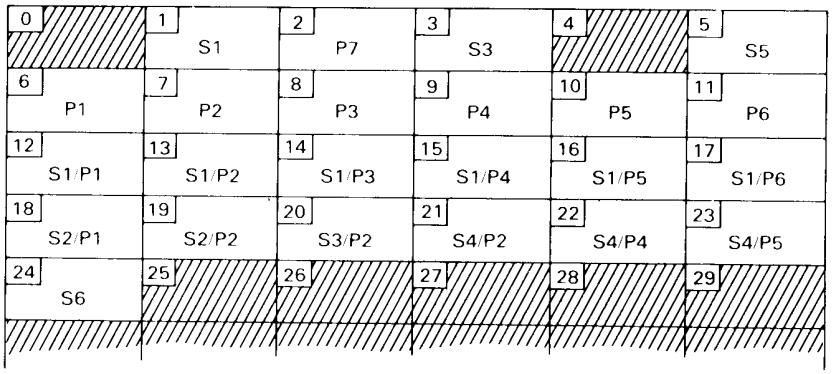


Fig. 3.4 Disposición del disco después de insertar el proveedor S6, eliminar el proveedor S2, insertar la parte P7 y eliminar el proveedor S4

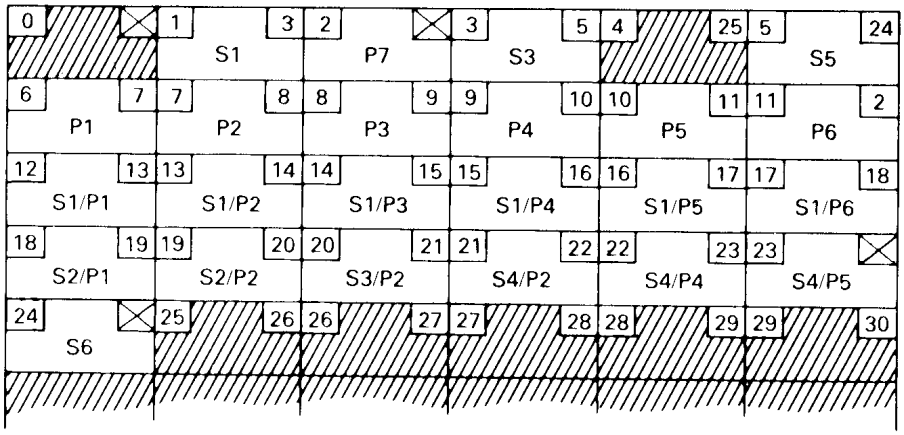


Fig. 3.5 Modificación de la figura 3.4 para ilustrar los apuntadores a la "siguiente página" (esquina superior izquierda de cada página)

Conjunto de páginas	Dirección de la primera página
Espacio libre	4
Proveedores	1
Partes	6
Envíos	12

Fig 3.6 El directorio del disco ("página cero").

El manejador de archivos. Permite al DBMS olvidarse de los detalles de E/S de páginas y pensar casi por completo en términos de archivos y registros almacenados. Esta función del manejador de archivos se conoce como *administración de registros almacenados.*

Los registros almacenados se identifican internamente por su "identificador de registro", o RID.

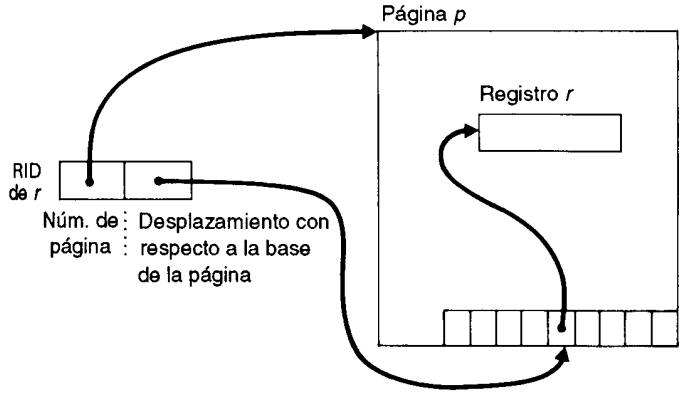


Fig 3.7: Forma de manejar los identificadores de registro almacenado (RID)

INDIZACION

Vamos a suponer que la consulta "encontrar todos los proveedores en la ciudad *C*" (donde *C* es un parámetro) es importante, es decir que se ejecuta con frecuencia y por ende debe tener buen desempeño. Ver figura 3.8. En esa representación hay dos archivos almacenados, uno de proveedores y otro de ciudades (con toda probabilidad en distintos conjuntos de páginas); el archivos de ciudades, almacenado en orden por ciudad, incluye apuntadores al archivo de proveedores. Si necesita encontrar todos los proveedores de Londres, el DBMS puede adoptar ahora dos posibles estrategias:

1) Examinar el archivo de proveedores completo, buscando todos los registros en los cuales el valor del campo de ciudad sea Londres.
2) Buscar, en el archivo de ciudades, todas las entradas de Londres, y para cada una de ellas seguir el apuntador al registro correspondiente en el archivo de proveedores.

En este ejemplo, se dice que **el archivo de ciudades es un *índice* del archivo de proveedores. Un índice es un tipo especial de archivo almacenado**. Es un archivo en el cual **cada entrada (registro) se compone de solo dos valores, un dato y un apuntador (RID)**, el apuntador identifica un registro de ese archivo que tiene ese valor en ese campo. El campo en cuestión del archivo indizado se llama *campo indizado*, o a veces *clave de indización*. Un índice según un campo de clave primaria se denomina índice *primario*. Un índice según cualquier otro campo – digamos el campo CIUDAD en el ejemplo – se denomina índice *secundario*.

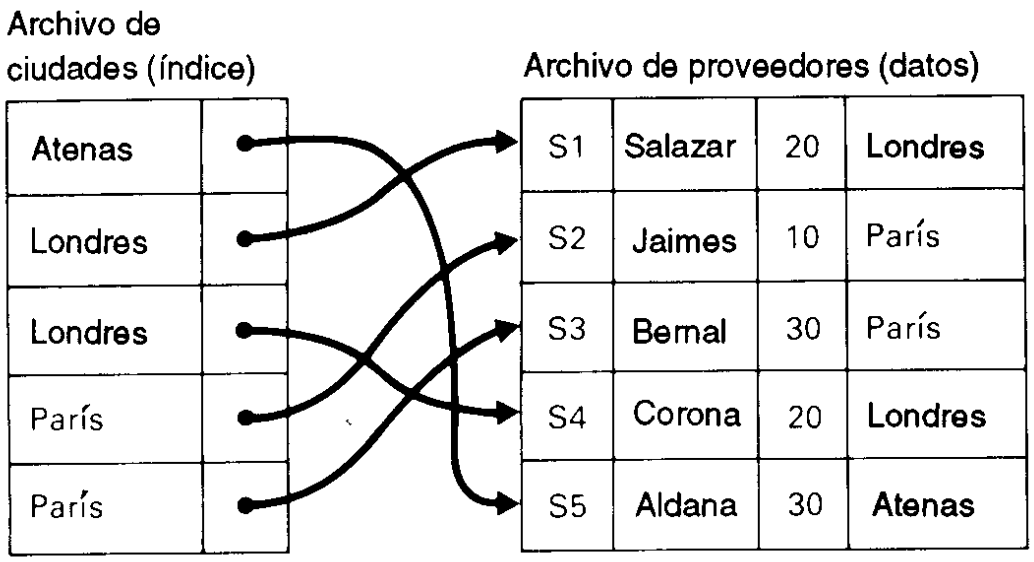


Fig. 3.8 Indización del archivo de proveedores según CIUDAD

En este ejemplo, se dice que el archivo de ciudades es un *índice* del archivo de proveedores. Un índice es un tipo especial de archivo almacenado. Es un archivo en el cual cada entrada (registro) se compone de solo dos valores, un dato y un apuntador (RID), el apuntador identifica un registro de ese archivo que tiene ese valor en ese campo. El campo en cuestión del archivo indizado se llama *campo indizado*, o a veces *clave de indización*. Un índice según un campo de clave primaria se denomina índice *primario*. Un índice según cualquier otro campo – digamos el campo CIUDAD en el ejemplo – se denomina índice *secundario*.

Cómo se utilizan los índices:

La ventaja primordial de los índices es la agilización de la obtención de datos. Existe una desventaja correspondiente, a saber, hacen más lenta la actualización de los archivos.

Cuando se añade un registro almacenado nuevo al archivo indizado, es preciso agregar también una entrada al índice.

Un archivo almacenado puede tener cualquier cantidad de índices.

Más terminología: algunos llaman a los índices *listas invertidas*. y cuando un archivo tiene un índice para cada campo se dice a veces que está *totalmente invertido.*

También es posible construir un índice con base en los valores combinados de dos o más campos.

Los árboles B ofrecen, al parecer, el mejor desempeño general.

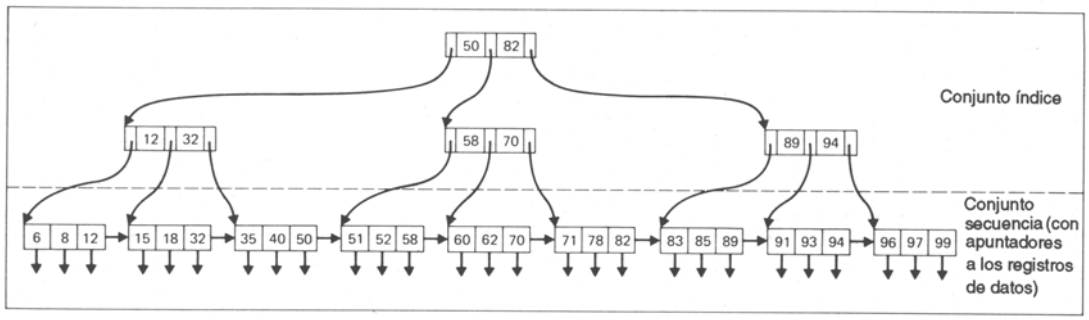


Fig 3.9 Parte de un árbol B sencillo (variación de Knuth)

Investigue: ¿qué es búsqueda binaria?, ¿qué tipos de árboles existen? (binarios, b etc.), vsam.

ACTIVIDAD Nº 3:

- **Responda el siguiente cuestionario**:

  1) ¿Por qué un objeto prioritario de desempeño en sistemas de bases de datos es reducir al mínimo e número de accesos a disco?.

  2) ¿Qué es el diseño físico de bases de datos?

  3) ¿Cómo pueden explicarse los principios de acceso a BD?.

  4) ¿Qué es el manejador de disco?

  5) ¿Qué es el manejador de archivos?.

  6) ¿Cuál es la idea básica del agrupamiento?.

  7) ¿Qué es un índice y cómo se utilizan?.

Unidad IV: Sistemas Relacionales

BASES DE DATOS RELACIONALES

Una base de datos relacional es aquella cuyos usuarios la perciben como un conjunto de tablas.

Tabla S

S#	SNOMBRE	SITUACIÓN	CIUDAD
S1	Salazar	20	Londres
S2	Jaimes	10	París
S3	Bernal	30	París
S4	Corona	20	Londres
S5	Aldana	30	Atenas

Tabla P

P#	SNOMBRE	COLOR	PESO	CIUDAD_
P1	Tuerca	Rojo	12	Londres
P2	Perno	Verde	17	París
P3	Birlo	Azul	17	Roma
P4	Birlo	Rojo	14	Londres
P5	Leva	Azul	12	París
P6	Engrane	Rojo	19	Londres

Tabla SP

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

FIGURA Nº 4.1 La base de datos de proveedores y partes

Como puede verse en la figura, la base de datos se compones de tres tablas, cuyos nombres son S, P y SP.

- La tabla S representa proveedores, supondremos que cada proveedor está situado en una sola ciudad.

- La tabla P representa partes; contiene una localidad donde se almacenan las partes (CIUDAD), suponemos que cada tipo de parte tiene un solo color y se almacena en una bodega de una sola ciudad.
- La tabla SP representa envíos. Sirve en cierto sentido para conectar entre sí las otras dos tablas. Así, cada envío tiene un número de proveedor (S#), un número de parte (P#), y una cantidad (CANT). Supondremos que en un momento dado sólo puede haber un envio para un proveedor dado y una parte dada; para un embarque específico, la combinación de valor S# y valor de P# es única con respecto al conjunto de envíos que aparece en ese momento en la tabla SP.

Los proveedores y las partes pueden considerarse *entidades*, y un envio puede considerarse una *interrelación* entre un determinado proveedor y una parte específica, pero, lo mejor es considerar las interrelaciones como un caso especial de las entidades.

Las entidades, se representan mediante tablas.

Si bien se eligieron nombres para las tablas (S, P y SP); es más recomendable utilizar nombres descriptivos, por ejemplo Proveedores en vez de tabla S.

La clave primaria de una tabla es un identificador único para los registros de esa tabla. Son el campo S# de ta tabla S, el campo P# de la tabla P, y el campo compuesto(S#, P#) de la tabla SP.

Dos aspectos:

1) **Todos los valores de los datos son atómicos**. Es decir, en cada intersección de fila y columna de cada tabla hay siempre un solo valor, nunca un conjunto de valores. Así, por ejemplo, en la tabla SP tenemos:

S#	P#
.	.
S2	P1
S2	P2
.	.
S4	P2
S4	P4
S4	P5
.	.
.	.

y no tenemos

S#	P#
.	.
S2	(P1, P2)
.	.
S4	(P2, P4, P5)
.	.

(versión 2 de P#)

. .

Una columna como la de P#, en la segunda versión de esta tabla, representa lo que a veces se denomina un "grupo repetitivo", que contiene conjuntos de valores. ***Las bases de datos relacionales no pueden tener grupos repetitivos.***

2) En segundo término, adviértase que todo el contenido de información de la base de datos se representa como ***valores explícitos de los datos.***, no existen "ligas" o apuntadores que conecten una tabla con otra. Por ejemplo, existe una interrelación entre la fila P1; pero esa interrelación se representa no mediante apuntadores, sino mediante la existencia de una fila en la tabla SP en la cual el valor de S# es S1 y el valor de P# es P1.

Cuando decimos que no se permiten grupos repetitivos ni apuntadores, no queremos decir que tales construcciones no puedan existir en el nivel físico. Al llamar relacional a un sistema dado quiere decirse que éste maneja tablas relacionales *en los niveles externos y conceptua*l; en el nivel *interno*, el sistema está en libertad de utilizar cualquier estructura que desee, con la única condición de que sea capaz de presentar esas estructuras al usuario (en el nivel externo o conceptual) en la forma tabular simple.

Así, **las tablas relacionales son *abstracciones* de la forma como se almacenan físicamente los datos**.

"Relación" no es más que un término matemático para referirse a una tabla.

EL LENGUAJE SQL

SQL ("Structured Query Lenguaje, lenguaje de consulta estructurada), con este lenguaje se **formulan operaciones relacionales** (o sea, operaciones que definen y manipulan datos en forma relacional).

La figura 4.1, es una representación de esa base de datos tal como podría verse en un instante específico, es una instantánea de la base de datos. La Figura 4.2, muestra en cambio, la estructura de la base de datos.

```
CREATE  TABLE S
   ( S#            CHAR (5)     NOT NULL,
   SNOMBRE     CHAR (20)    NOT NULL,
   SITUACION   SMALLINT   NOT NULL,
   CIUDAD      CHAR (15)    NOT NULL,
   PRIMARY KEY   ( S#     )    ) ;
```

```
CREATE TABLE  P
   ( P#            CHAR (6)     NOT NULL,
   PNOMBRE     CHAR (20)    NOT NULL,
   COLOR         CHAR (6)     NOT NULL,
   PESO           SMALLINT   NOT NULL,
   CIUDAD       CHAR(15)     NOT NULL,
   PRIMARY KEY   ( P#      )    ) ;

CREATE TABLE  SP
   ( S#            CHAR (5)     NOT NULL,
     P#            CHAR (6)     NOT NULL,
   CANT          SMALLINT   NOT NULL,
   PRIMARY KEY   ( S#   ,  P#      ) ,
   FOREIGN KEY   ( S#      )   REFERENCES S,
   FOREIGN KEY   ( P#      )   REFERENCES P   ) ;
```

Fig. 4.2 La base de datos de proveedores y partes (definición de los datos).

La definición incluye una proposición CREATE TABLE (crear tabla) para cada una de las tres tablas, especifica el nombre de la tabla que se va a crear, los nombres y tipos de datos de sus columnas (campos o atributos), y la clave primaria de la tabla. En el caso de la tabla SP, especifica que S# y P# son claves ajenas (FOREIGN KEY), las cuales hacen referencia a las tablas S y P repectivamente; por ahora, nos limitaremos a decir que **una clave ajena es una columna de una tabla cuyos valores están restringidos a los valores de alguna otra clave primaria**. Por ejemplo, todo valor de la columna S# en la tabla SP debe aparecer en la columna S# de la tabla S.

CREATE TABLE es una *proposición ejecutable.* Al principio esas tablas estarían vacías; es decir, sólo incluirían la fila de cabeceras de las columnas, podríamos insertar enseguida filas de datos, por ejemplo con la proposición INSERT, y así podríamos tener a nuestra disposición una base de datos.

Podemos comenzar a trabajar con ellas utilizando las proposiciones de *manipulación de datos* de SQL. En la Figura 4.3 muestra un ejemplo de recuperación de datos.

SQL puede utilizarse a través de dos interfaces diferentes a saber, una interfaz interactiva (sql interactivo) y una interfaz para programas de aplicaciones (sql embebido). La Figura 4.3 muestra un ejemplo de la interfaz interactiva.

SQL es al mismo tiempo un lenguaje de consulta interactiva y un lenguaje de programación de bases de datos.

El SQL interactivo y el SQL embebido difieren en ciertos detalles.

```
SELECT      CIUDAD
    FROM        S
WHERE     S# = 'S4';
```

Resultado

CIUDAD
Londres

Fig. 4.3 Ejemplo de obtención de datos

Observe la Figura 4.4, en condiciones normales habrán muchos usuarios, trabajando con los mismos datos al mismo tiempo, ya sea que utilicen el sql interactivo o el sql embebido en algún lenguaje de aplicación. El motor de base de datos (DBMS) aplicará en forma automática los controles necesarios.

Las tablas también son de dos clases a saber, *tablas base y vistas*:

- Una tabla base es una tabla "real"; es decir una tabla con existencia física.
- Una vista es una tabla "virtual" es decir, una tabla que no está almacenada físicamente. Las vistas pueden concebirse como formas diferentes de ver las tablas "reales".

Una tabla base dada puede tener como máximo un índice de agrupamiento (para controlar el agrupamiento dentro del archivo) y cualquier cantidad de índices adicionales que no sean de agrupamiento, en casi todos los sistemas relaciónales actuales es posible manejar árboles B.

Las vistas se pueden crear en cualquier momento. Lo mismo puede decirse de los índices. La proposición CREATE VIEW permite crear vistas o tablas "virtuales", y la preposición CREATE INDEX es análoga para crear índices. Las tablas bases, vistas e índices pueden desecharse en cualquier momento mediante las proposiciones DROP TABLE (desechar tabla), DROP VIEW (desechar vista) o DROP INDEX (desechar índice). En el caso de los índices debe tenerse muy presente que aunque el usuario es el responsable de su creación y eliminación, no tiene injerencia en su utilización. Los índices nunca se mencionan en proposiciones de SQL para manipulación de datos, como SELECT. La decisión de utilizar o no un índice determinado para responder a una consulta, corresponde al sistema, no al usuario.

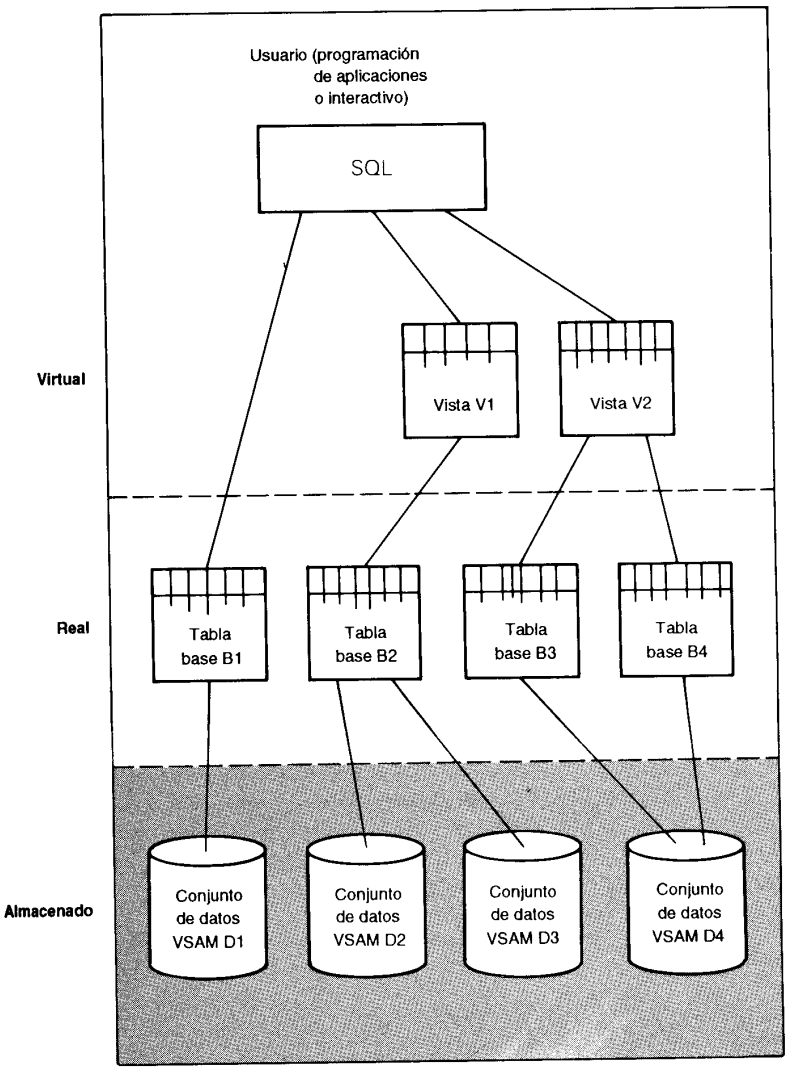


Fig 4.4 Cómo percibe el usuario el sistema

Las principales funciones para la definición de datos son las siguientes:

CREATE TABLE
CREATE VIEW
CREATE INDEX

DROP TABLE
DROP VIEW
DROP INDEX

Las principales funciones para manipulación de datos son:
SELECT
UPDATE

DELETE
INSERT

Las proposiciones de SQL, para manipulación de datos operan por lo regular sobre conjuntos enteros de registros, y no sobre un solo registro a la vez, la proposición SELECT (Fig. 4.5(a)) devuelve un conjunto de cuatro valores, no un solo valor, y la proposición UPDATE (Fig. 4.5 (b)) modifica dos registros, no solo uno. Dicho de otro modo **SQL, es un lenguaje a nivel de conjuntos.**

Los usuarios especifican qué y no cómo. El proceso de "navegación" dentro de la base de datos física para localizar los datos deseados lo realiza de manera automática el sistema. Por eso los sistemas Relacionales se denominan a veces sistemas de "navegación automática". En los sistemas no relacionales esta navegación casi siempre es responsabilidad del usuario.

A)

```
SELECT      S#
FROM        SP
WHERE       P# = '2';
```

Resultado

S#
S1
S2
S3
S4

B)

```
UPDATE      S
SET         SITUACION = 2 * SITUACION
WHERE       CIUDAD = 'Londres';
```

Resultado : Se duplica la situación de los proveedores en Londres (o sea S1 y S4)

Figura 4.5 Ejemplos de manipulación de datos en SQL.

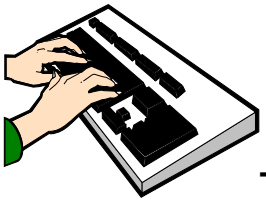
Los usuarios no pueden en absoluto hacer referencia directa a un índice en una solicitud de acceso en SQL. Por ello los índices se pueden crear o destruir en cualquier momento sin afectar en lo más mínimo a los usuarios (excepto en el desempeño, desde luego).

SQL es el sublenguaje de datos del sistema. Como tal, incluye un componente de **lenguaje de definición de datos (DDL)** y también un componente de **lenguaje de manipulación de datos (DML)**.

El DDL puede utilizarse para definir objetos en el nivel externo (vistas), en el nivel conceptual (tablas base), e incluso en el nivel interno (índices). Además, SQL ofrece también ciertos recursos para el "control de datos",

recursos que no pertenecen de manera clara ni al DDL ni al DML. Un ejemplo de estos recursos es la proposición GRANT, con lo cual un usuario pude conceder ciertos privilegios de acceso a otro.

Los programas de aplicación tienen acceso a la base de datos desde un lenguaje anfitrión como puede ser VBASIC, mediante proposiciones de SQL embebido, representa un "acoplamiento débil" entre SQL y el lenguaje anfitrión. En general, cualquier proposición que se puede utilizar en SQL interactivo se puede emplear también en SQL embebido. Además, se incluyen ciertas proposiciones especiales para usarse sólo en el ambiente de lenguaje embebido.



TRABAJO EN LABORATORIO...

- Utilizando el DBMS ACCESS, cree las tablas S, P SP en forma visual, luego cree S1, P1, SP1, utilizando el correspondiente DDL.
- ¿Cuando conviene utilizar el ddl y cuando el entorno gráfico?

ACTIVIDAD Nº 4:

- **Responda el siguiente cuestionario:**

1) ¿Qué es la clave primaria de una tabla?.

2) ¿Qué significa datos atómicos, ejemplifique?.

3) ¿Qué significa valores explícitos de los datos, ejemplifique?.

4) ¿Qué son las tablas base y vistas?.

5) ¿Por qué los sistemas Relacionales se denominan a veces sistemas de “navegación automática” ?.

6) ¿Cuáles son las principales funciones para la definición de datos?.

7) ¿Cuáles son las principales funciones para la manipulación de datos?.

Unidad V: Definición de datos

INTRODUCCIÓN

Las principales proposiciones de DDL son:

CREATE TABLE	CREATE VIEW	CREATE INDEX
ALTER TABLE		
DROP TABLE	DROP VIEW	DROP INDEX

TABLAS BASE

Una tabla es un sistema relacional y se compone de una fila de cabeceras de columna, junto con cero o más filas de valores de datos.

La fila de cabeceras de columna específica una o más columnas (dando entre otras cosas, un tipo de datos para cada una).

Cada fila de datos contiene un solo valor para cada una de las columnas especificadas en la fila de cabeceras de columna (valores atómicos). Todos los valores de una columna dada tienen el mismo tipo de datos.

Debemos tener en cuenta que:

- No se menciona ningún ordenamiento de filas. Se considera que las filas de una tabla relacional están desordenadas. Tal ordenamiento se debe considerar sólo como algo conveniente para el usuario.
- Las columnas de una tabla se consideran ordenadas, de izquierda a derecha.

*Digresión: P*or supuesto, las filas y columnas sí tienen un ordenamiento físico en la tabla almacenada en el disco, pero son transparentes para el usuario.

CREATE TABLE, ALTER TABLE, DROP TABLE

CREATE TABLE (sintaxis):

CREATE TABLE tabla-base
(definición-de-columna [, definición-de-columna] ...
[, definición-de-clave-primaria]
[, definición-de-clave-ajena [, definición-de-clave-ajena] ....]);

donde una “definición-de columna”, a su vez, tiene la forma:

columna tipo-de-datos [NOT NULL]

Los corchetes se utilizarán en todas las definiciones sintácticas para indicar que lo encerrado por ellos es opcional. Los puntos suspensivos (...) significan que la unidad sintáctica inmediata anterior puede repetirse de manera opcional una o más veces. Lo que va en mayúscula debe escribirse tal como se muestra; lo que va en minúsculas debe reemplazarse con valores elegidos por el usuario.

CREATE TABLE (ejemplo):

```
CREATE TABLE  S
(  S#              CHAR (5)        NOT NULL,
   SNOMBRE         CHAR (20)       NOT NULL,
   SITUACION       SMALLINT        NOT NULL,
   CIUDAD          CHAR (15)       NOT NULL,
   PRIMARY  KEY    ( S#     )  ) ;
```

Tipos de datos (para sql estándar):

- ***Datos numéricos***

INTEGER	entero binario de palabra completa.
SMALLINT	entero binario de media palabra.
DECIMAL (*p, q)*	número decimal empacado, *p* dígitos y signo, con *q* dígitos a la derecha del punto decimal.
FLOAT *(p*)	número de punto flotante, con precisión de *p* dígitos binarios.

- ***Datos de cadena***

CARÁCTER (*n*)	cadena de longitud fija con exactamente *n* caracteres de 8 bits.
VARCHAR (*n*)	cadena de longitud variable con hasta *n* caracteres de 8 bits.
GRAPHIC (*n*)	cadena de longitud fija con exactamente *n* caracteres de 16 bits.
VARGRAPHIC (*n*)	cadena de longitud variable con hasta *n* caracteres de 16 bits.

- ***Datos de fecha /hora***

DATE	fecha (*aaaammdd).*
TIME	hora *(hhmmss).*
TIMESTAMP	"marca de tiempo" (combinación de fecha y hora, con una precisión de microsegundos).

Abreviaturas (por ejemplo, CHAR en vez de CARÁCTER, DEC en vez de DECIMAL, DEC en vez de DECIMAL, INT en vez de INTEGER).

Con la ayuda de Access observe los tipos de datos existentes para sql y la estructura que posee la proposición Create Table.

INFORMACIÓN FALTANTE:

La información faltante tales como "fecha de nacimiento desconocida", se suele representar mediante indicadores especiales llamados nulos (nulls). Es importante señalar que no es lo mismo un nulo que (por ejemplo) un espacio en blanco o un cero. Cualquier campo puede contener nulos a menos que la definición de ese campo especifique NOT NULL (no nulo) de manera explícita.

Digresión: Una columna capaz de aceptar nulos se representa físicamente en la base de datos almacenada mediante dos columnas, la columna de datos propiamente dicha y una columna indicadora oculta, con ancho de un carácter. Si la columna indicadora contiene unos binarios, se hará caso omiso del valor correspondiente en la columna de datos; si la columna indicadora contiene ceros binarios, el valor correspondiente en la columna de datos se tomará como válido.

Las expresiones escalares de cálculo en las cuales uno de los operandos es nulo dan nulo como resultado.

Las expresiones escalares de comparación en las cuales uno de los comparandos es nulo dan como resultado el valor lógico *desconocido.* Los nulos provocan mucho más problemas de los que resuelven y conviene evitarlos.

ALTER TABLE:

Podemos *alterar* una tabla base ya existente agregando una columna nueva a la derecha, mediante la proposición ALTER TABLE (alterar tabla).

```
ALTER TABLE  tabla-base ADD columna tipo-de-datos;
```

Por ejemplo:

```
ALTER TABLE S ADD DESCUENTO SMALLINT;
```

No se permite la especificación NOT NULL en ALTER TABLE.

La próxima vez que se lea del disco, luego de un alter table, el motor anexará el nulo en los registros que contienen el nuevo campo adicionado antes de ponerlos a disposición del usuario. En algunos sistemas son posibles otros tipos de alteraciones de las tablas.

DROP TABLE:

También es posible eliminar en cualquier momento una tabla base existente:

```
DROP TABLE tabla-base;
```

ÍNDICES

CREATE INDEX (sintaxis):

```
CREATE [UNIQUE] INDEX indices
ON table-base  (columna [orden][, columna[ orden]]     …….)
[CLUSTER];
```

La decisión de utilizar o no un índice determinado para responder a una cierta consulta en SQL no la toma el usuario, sino el motor.

Cada especificación de “orden” es ASC (Ascendente) o bien DESC (descendente).

CREATE INDEX (ejemplo):

```
CREATE INDEX X  ON T  ( Q DESC )   CLUSTER  ;
```

Esta proposición crea un índice de agrupamiento llamado X para la tabla T en el cual las entradas se ordenan en forma descendente según el valor de Q.

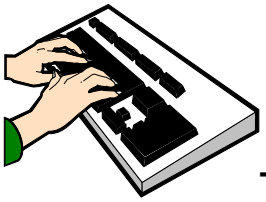
Con la opción UNIQUE (único) en CREATE INDEX, no se permitirá que los registros de la tabla base indizada tengan al mismo tiempo el mismo valor en el campo o combinación de campos de indización.

```
CREATE UNIQUE INDEX XS  ON  S      ( S#     ) ;
CREATE UNIQUE INDEX XP   ON  P     ( P#     )  ;
CREATE UNIQUE INDEX XSP ON SP( S#  ,   P#    ) ;
```

Ahora se rechazará cualquier intento de introducir mediante una operación INSERT (insertar) o UPDATE (actualizar), un valor repetido en el campo S#, o en el campo P#, o en el campo (compuesto) S#P#, de las tabla S,P y SP repectivamente.

La proposición para desechar un índice es:

```
DROP  INDEX  índice;
```



TRABAJO EN LABORATORIO...

- Utilizando el DBMS ACCESS, mediante el DDL, cree las tablas MEDICOS (matrícula, nombre, especialidad) y TURNOS (matrícula, nombre_paciente, fecha, hora).
- En MEDICOS: matrículas es clave primaria,
- En TURNOS: nombre_paciente, fecha y hora forman la clave principal. Matrícula es la clave foranea que relaciona la tabla TURNOS con MEDICOS.

ACTIVIDAD Nº 5:

- **Responda el siguiente cuestionario:**
  1) ¿Cómo se compone una tabla en un sistema relacional ?
  2) ¿Cómo se comportan los campos NULLS en los cálculos y las comparaciones?
  3) ¿Cómo es la sintaxis de proposición CREATE TABLE ?.
  4) ¿Cómo es la sintaxis de proposición ALTER TABLE ?.
  5) ¿Cómo es la sintaxis de proposición DROP TABLE ?.
  6) ¿Cómo es la sintaxis de proposición CREATE INDEX ?.
  7) ¿Para qué sirve la opción UNIQUE?.

Unidad VI: Manipulación de datos.

PROPOSICIONES DML

El sql ofrece cuatro proposiciones de DML:

- SELECT (seleccionar)
- UPDATE (actualizar).
- DELETE (borrar).
- INSERT (insertar).

La que más utilizaremos será la proposición SELECT, el cual tiene la siguiente sintaxis:

```
SELECT  [DISTINCT]    elemento(s)
FROM          tabla (s)
   [ WHERE    condición   ]
   [GROUP    BY  campo(s)   ]
   [ ORDER    BY   campo(s)   ]
   [HAVING   condición  ]
```

CONSULTAS SIMPLES.

Ejemplo: "Obtener el número y la situación de todos los proveedores de París"

```
SELECT    S#, SITUACIÓN
FROM        S
WHERE     CIUDAD =  "Paris";
```

Resultado:

S#	SITUACION
S2	10
S3	30

"SELECT (seleccionar) los campos especificados FROM (de) la tabla especificada WHERE (donde) se cumpla la condición especificada". *El resultado de la consulta es otra tabla*.

Por cierto, podríamos haber formulado la consulta empleando nombres de *campo calificado* en todos los casos:

```
SELECT     S.S#,  S.SITUACIÓN
FROM       S
WHERE      S.CIUDAD   =  "Paris"  ;
```

Un nombre de campo calificado se compone de un nombre de tabla y un nombre de campo, en ese orden separados mediante un punto.

DISTINCT:

SQL no elimina filas repetidas, si el usuario no se lo pide de manera explícita mediante la palabra clave DISTINCT (distinto), como en:

```
SELECT DISTINCT  P#
FROM     SP ;
```

Resultado:

P#
P1
P2
P3
P4
P5
P6

DISTINTCT implica “eliminar *filas* repetidas “

VALORES CALCULADOS:

```
SELECT  P#,   “Peso en gramos  =“,  PESO *  454 AS PESOS
FROM    P;
```

Resultado:

P#		PESOS
P1	peso en gramos =	5448
P2	peso en gramos =	7718
P3	peso en gramos =	7718
P4	peso en gramos =	6356
P5	peso en gramos =	5448
P6	peso en gramos =	8626

USO DEL “ * ”:

```
SELECT  *
FROM     S ;
```

Resultado: una copia de toda la tabla S. El asterisco representa una lista de todos los nombre de campo de la tabla o tablas nombradas en la cláusula FROM.

La notación de asterisco es cómoda, puede ser peligrosa en SQL embebido. Se utilizará “SELECT * “ solo en contextos donde no pueda provocar problemas.

RECUPERACIÓN CALIFICADA:

Obtener los números de los proveedores radicados en Paris cuya situación sea mayor que 20

```
SELECT  S#
FROM    S
WHERE   CIUDAD  =  "Paris"
AND     SITUACIÓN >20;
```

```
Resultado:   S#
             - -
             S3
```

La condición que sigue a WHERE (donde) puede incluir los operadores de comparación = , <> (diferente de), >, >=, <, y <=; los operadores boléanos AND, OR y NOT; y paréntesis para indicar un orden de evaluación deseado.

RECUPERACIÓN CON ORDENAMIENTO:

Obtener números de proveedor y situación de los proveedores radicados en París, en orden descendente por situación.

```
SELECT   S#, SITUACIÓN
FROM     S
WHERE    CIUDAD  =  "París "
ORDER BY SITUACIÓN DESC;
```

```
Resultado:    - - -      - - - - - - - -
              S#         SITUACIÓN
              - - -      - - - - - - - - - -
              S3               30
              S2               10
```

"Orden" puede ser ASC ó DESC, y ASC es el orden por omisión.

También es posible identificar columnas en la cláusula ORDER BY (ordenar por) empleando *números* de columna en vez de nombre.

```
SELECT P.P# ,   ' Peso en gramos  = ' , P.PESO * 454
FROM    P
ORDER   BY  3,  P# ;                /* o bien  "ORDER BY  3, 1 ;"   */
```

El "3" se refiere a la tercera columna de la tabla de resultado.

```
Resultado:    - - -   - - - - - - - - - - - - - -   - - - -
              P
              - - -   - - - - - - - - - - - - - -   - - - -
              P1      Peso en gramos  =   5448
              P5      Peso en gramos  =   5448
              P4      Peso en gramos  =   6356
```

P2 Peso en gramos = 7718
P3 Peso en gramos = 7718
P6 Peso en gramos = 8626

CONSULTAS DE REUNION

La posibilidad de "reunir" dos ó más tablas en una de las características más poderosas de los sistemas relaciones.

Una reunión es una *consulta en la cual se obtiene información de más de una tabla*.

EQUIRREUNIÓN SIMPLE:

Obtener todas las combinaciones de información de proveedores y partes tales que el proveedor y la parte en cuestión estén situados en la misma ciudad (cosituados).

```
SELECT S.*, P.*
FROM     S, P
WHERE S.CIUDAD = P.CIUDAD ;
```

Resultado:

S #	SNOMBRE	SITUACIÓN	CIUDAD	P #	PNOMBRE	COLOR	PESO	P.CIUDAD
S1	Salazar	20	Londres	P1	Tuerca	Rojo	12	Londres
S1	Salazar	20	Londres	P4	Birlo	Rojo	14	Londres
S1	Salazar	20	Londres	P6	Engrane	Rojo	19	Londres
S2	Jaimes	10	París	P2	Perno	Verde	17	París
S2	Jaimes	10	París	P5	Leva	Azul	12	París
S3	Bernal	30	París	P2	Perno	Verde	17	París
S3	Bernal	30	París	P5	Leva	Azul	12	París
S4	Corona	20	Londres	P1	Tuerca	Rojo	12	Londres
S4	Corona	20	Londres	P4	Birlo	Rojo	14	Londres
S4	Corona	20	Londres	P6	Engrane	Rojo	19	Londres

La forma de operar es la siguiente:

1. Se forma el *producto cartesiano* de las tablas incluidas en la cláusula FROM (de). Por ejemplo si la tabla P tiene (P1, P2) y la tabla S tiene (S1,S2), entonces el producto cartesiano tendrá cuatro pares (P1 S1, P1 S2, P2 S1, P2 S2). La cantidad de registros será el producto de los registro de P por los registros de S es decir 2x2=4(registros).
2. Se Eliminan ahora de este producto cartesiano todas las filas que no satisfagan la condición de reunión.

REUNIÓN MAYOR – QUE:

Obtener todas las combinaciones de información de proveedor y parte donde la ciudad del proveedor siga a la ciudad de la parte en el orden alfabético.

```
SELECT  S.*,  P.*
FROM    S, P
WHERE  S.CIUDAD > P.CIUDAD ;
```

Resultado:

S #	SNOMBRE	SITUACIÓN	CIUDAD	P #	PNOMBRE	COLOR	PESO	P.CIUDAD
S2	Jaimes	10	París	P1	Tuerca	Rojo	12	Londres
S2	Jaimes	10	París	P4	Birlo	Rojo	14	Londres
S2	Jaimes	10	París	P6	Engrane	Rojo	19	Londres
S3	Bernal	30	París	P1	Tuerca	Rojo	12	Londres
S3	Bernal	30	París	P4	Birlo	Rojo	14	Londres
S3	Bernal	30	París	P6	Engrane	Rojo	19	Londres

CONSULTA DE REUNIÓN CON UNA CONDICIÓN ADICIONAL:

Obtener todas las combinaciones de información de proveedor y parte donde el proveedor y la parte en cuestión estén cosituados, pero omitiendo a los proveedores cuya situación sea 20.

```
SELECT  S.*,  P.*
FROM    S,  P
WHERE  S.CIUDAD = P.CIUDAD
AND      S.SITUACION <>  20  ;
```

Resultado:

S #	SNOMBRE	SITUACIÓN	S.CIUDAD	P #	PNOMBRE	COLOR	PESO	P.CIUDAD
S2	Jaimes	10	París	P2	Perno	Verde	17	París
S2	Jaimes	10	París	P5	Leva	Azul	12	París
S3	Bernal	30	París	P2	Perno	Verde	17	París
S3	Bernal	30	París	P5	Leva	Azul	12	París

RECUPERACIÓN DE CAMPOS ESPECÍFICOS DE UNA REUNIÓN:

Obtener todas las combinaciones de (número de proveedor / número de parte) tales que el proveedor y la parte en cuestión estén cosituados.

```
SELECT  S.S # ,  P.P ·#
FROM    S,  P
WHERE  S.CIUDAD = P.CIUDAD ;
```

Resultado:

S #	P#
S1	P1
S1	P4
S1	P6
S2	P2
S2	P5
S3	P2
S3	P5
S4	P1
S4	P4
S4	P6

REUNIÓN DE TRES TABLAS:

Obtener todas las parejas de nombres de ciudad tales que un proveedor situado en la primera ciudad suministre una parte almacenada en la segunda ciudad. Por ejemplo, el proveedor S1 suministra la parte P1; el proveedor S1 está situado en Londres, y la parte P1 se almacena en Londres; por tanto, (Londres, Londres) es una pareja de ciudades en el resultado.

```
SELECT  DISTINCT  S.CIUDAD,  P.CIUDAD
   FROM     S,  SP,  P
   WHERE   S.S #  =  SP.S #
   AND       SP. P #  =  P.P #   ;
```

Resultado:

S.CIUDAD	P.CIUDAD
Londres	Londres
Londres	París
Londres	Roma
París	Londres
París	París

REUNIÓN DE UNA TABLA CONSIGO MISMA:

Obtener todas las parejas de números de proveedor tales que los dos proveedores en cuestión estén cosituados.

```
SELECT  PRIMERA.S# , SEGUNDA.S#
   FROM      S PRIMERA,  S SEGUNDA
   WHERE    PRIMERA.CIUDAD  =  SEGUNDA.CIUDAD
   AND        PRIMERA.S#  <  SEGUNDA.S# ;
```

Resultado:

S#	S
S1	S4
S2	S3

Esta consulta implica una reunión de la tabla S consigo misma (con base en igualdad de ciudades). Suponga por un momento que tenemos dos copias separadas de la tabla S, la "primera" y la "segunda". Entonces, la lógica de la consulta será la siguiente: necesitamos poder analizar todas las parejas posibles de fila de proveedores, una de la primera copia de S y otra de la segunda, y extraer los dos números de proveedor de una de esas parejas de filas si y sólo si los valores de ciudad son iguales. Por tanto necesitamos poder hacer referencia a dos filas de proveedor al mismo tiempo. Para distinguir entre las dos referencias, introducimos las variables de recorrido arbitrarias PRIMERA y SEGUNDA, cada una de las cuales "recorre" la tabla S. En todo momento, PRIMERA representa alguna fila de la "primera" copia. El resultado de la consulta se obtiene examinando todas las posibles parejas de valores PRIMERA/SEGUNDA y probando la condición WHERE en cada caso.

Nota: El propósito de la condición PRIMERA.S# < SEGUNDA.S# es doble:

a) elimina parejas de números de proveedor de la forma (x,x);
b) garantiza la aparición de sólo una de las parejas (x,y) e (y,x).

Éste es el primer ejemplo que hemos visto en el cual ha sido necesario el empleo de variables de recorrido. Sin embargo, nunca es erróneo introducir tales variables, aun cuando no sea indispensable su empleo, y en ocasiones pueden hacer más clara la proposición.

FUNCIONES DE AGREGADOS.

SQL ofrece una serie de funciones de agregados especiales para ampliar su capacidad básica de recuperación de información. Esta funciones son:

- COUNT (cuenta).
- SUM (suma).
- AVG (promedio).
- MAX (máximo).
- MIN (mínimo).

Cada una de estas funciones trabaja sobre el total de valores en una columna de alguna tabla y produce un solo valor como resultado según estas definiciones.

En el caso se SUM y AVG la columna debe contener valores numéricos. En general, el argumento de la función puede ir precedido de manera opcional por la palabra clave DISTINCT para indicar que los valores repetidos deben

eliminarse antes de que se aplique la función. En el caso de MAX y MIN, empero, DISTINCT no tiene efecto alguno. En el caso de COUNT, es necesario especificar DISTINCT; se incluye la función especial COUNT(*) – no se permite DISTINCT- para contar todas las filas de una tabla sin eliminación de duplicados.

Si hay nulos en la columna del argumento, se eliminarán antes de aplicarse la función.

FUNCIÓN DE AGREGADOS EN LA CLÁUSULA SELECT:

Obtener el número total de proveedores.

```
SELECT  COUNT(*)
FROM    S;
```

Resultado:

```
---
5
```

Observe que el resultado es una tabla, con una fila y una columna (sin nombre), y contiene un solo valor escalar, 5.

FUNCIÓN DE AGREGADOS EN LA CLÁUSULA SELECT, CON DISTINCT:

Obtener el número total de proveedores que suministran partes en la actualidad.

```
SELECT  COUNT(DISTINT S#)
FROM    SP;
```

Resultado:

```
---
4
```

FUNCIÓN DE AGREGADOS EN LA CLÁUSULA SELECT, CON UNA CONDICIÓN:

Obtener el número de envíos de la parte P2.

```
SELECT  COUNT(*)
FROM    SP
WHERE  P# = 'P2' ;
```

Resultado:

```
---
4
```

Obtener la cantidad total suministrada de la parte P2.

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

Resultado:

```
--------
1000
```

EMPLEO DE GROUP BY (AGRUPAR POR):

Vamos a suponer que deseamos calcular la cantidad total suministrada por cada parte: es decir, para cada parte suministrada, obtener el número de parte y la cantidad total enviada de esa parte.

```
SELECT  P#, SUM  (CANT), COUNT(*)
FROM    SP
GROUP BY P# ;
```

Resultado:

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

El operador GROUP BY (agrupar por) reorganiza en el sentido lógico la tabla representada por la cláusula FROM (de) formando particiones o grupos, de manera que dentro de un grupo dado todas las filas tengan el mismo valor en el campo de GROUP BY. En el ejemplo la tabla SP se agrupa de manera tal que un grupo contiene todas las filas de la parte P1, otro contiene todas las filas de la parte P2, etcétera. En seguida se aplica la cláusula SELECT a cada grupo de la tabla dividida (y no a cada fila de la tabla original). Cada expresión en la cláusula SELECT debe producir un solo valor por grupo.

Adviértase que GROUP BY no implica ORDER BY (ordenar por); deberemos especificar también la cláusula ORDER BY P# (después de la cláusula GROUP BY).

EMPLEO DE HAVING (CON):

Obtener los números de todas las partes suministradas por más de un proveedor.

```
SELECT  P#
FROM    SP
GROUP BY P#
HAVING COUNT(*) > 1 ;
```

```
--------
Resultado:  P#
--------
            P1
            P2
            P4
            P5
```

HAVING es a los grupos lo que WHERE es a las filas (si se especifica HAVING, deberá haberse especificado también GROUP BY). En otras palabras, HAVING (con) sirve para eliminar grupos de la misma manera como WHERE (donde) sirve para eliminar filas. Las expresiones en una cláusula HAVING deben producir un solo valor por grupo.

CARACTERISTICAS AVANZADAS

RECUPERACIÓN DE DATOS CON LIKE(COMO):

Obtener todas las partes cuyos nombres comiencen con la letra B.

```
SELECT  P.*
FROM    P
WHERE P.NOMBRE LIKE 'B%';
```

```
------  ----------------  ----------  ----------  ------------
Resultado:  P#  PNOMBRE   COLOR       PESO        CIUDAD
------  ----------------  ----------  ----------  ------------
            P3  Birlo      Azul        17          Roma
            P4  Birlo      Rojo        14          Londres
```

En general, una condición LIKE(como) adopta la forma:

Columna LIKE literal-de-columna

Columna debe designar una columna de tipo de cadena (CHAR o VARCHAR, o GRAPHIC o VARGRAPHIC). Los caracteres dentro de "literal" se interpretan de esta manera:

- El carácter "_" (subrayado) representa cualquier carácter individual.
- El carácter "%" (por ciento) representa cualquier secuencia de n caracteres (donde n puede ser cero).
- Todos los demás caracteres se representan a sí mismos.

Ejemplos:

DOMICILIO LIKE '%Lima'

-Se cumplirá si DOMICILIO contiene la cadena "Lima" en cualquier posición

S# LIKE 'S__'

-Se cumplirá si S# tiene una longitud de tres caracteres y el primero es una "S".

PNOMBRE LIKE '%c___'
-Se cumplirá si PNOMBRE tiene una longitud de cuatro o más caracteres y el tercero antes del último es una "c".
CIUDAD NOT LIKE '%E%'
-Se cumplirá si CIUDAD no contiene una "E".

Nota: Depende del SQL que esté utilizando, cambiarán estos caracteres por ejemplo por $ y *.

RECUPERACIÓN DE DATOS CON SUBCONSULTAS:

Obtener los nombres de los proveedores que suministran la parte P2.

```
SELECT  SNOMBRE
FROM    S
WHERE S# IN
   (SELECT S#
   FROM SP
   WHERE P# = 'P2' );
```

Resultado:

```
 ----------------
 SNOMBRE
----------------
Salazar
Jaimes
Bernal
Corona
```

Explicación: Una subconsulta es (en términos informales) una expresión SELECT ... FROM ... WHERE .... GROUP BY .... HAVING anidada dentro de otra expresión del mismo tipo. Las subconsultas se utilizan por lo regular para representar el conjunto de valores en el cual se realizará una búsqueda mediante una "condición IN (en)". El sistema evaluará primero la subconsulta anidada. Esa subconsulta produce como resultado el conjunto de números de proveedor de los proveedores que suministran la parte P2, o sea el conjunto (S1,S2,S3,S4). Así, la subconsulta original equivale a esta otra consulta más sencilla.

```
SELECT  SNOMBRE
FROM    S
WHERE S# IN ('S1','S2','S3','S4');
```

Adviértase, por cierto, que el problema original "obtener los nombres de los proveedores que suministran la parte P2" se puede expresar también como una consulta de reunión así:

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

Los dos formularios de la consulta original son igualmente correctas. La elección de una u otra será cuestión sólo del gusto del usuario.

SUBCONSULTA CON VARIOS NIVELES DE ANIDAMIENTO:

Obtener los nombres de los proveedores que suministran por lo menos una parte roja.

```
SELECT  SNOMBRE
FROM    S
WHERE S# IN
            (SELECT S#
            FROM SP
            WHERE P# IN
                        (SELECT P#
                        FROM P
                        WHERE COLOR = 'Rojo' ) );
```

```
----------
SNOMBRE
----------
Salazar
Jaimes
Corona
```

Resultado:

Las subconsultas pueden estar anidadas a cualquier profundidad.

Veamos los resultados parciales.

La sentencia:

```
SELECT P#
FROM P
WHERE COLOR = 'Rojo':
```

Retorna: P1,P4,P6

La sentencia:

```
SELECT S#
FROM SP
WHERE P# IN ('P1','P4','P6'):
```

Retorna: S1,S2,S1,S4,S1

La sentencia:

```
SELECT  DISTINCT(SNOMBRE)
FROM    S
WHERE S# IN ('S1','S2','S1','S4','S1'):
```

Retorna: SALAZAR, JAIME, CORONA

IMPORTANTE: Debido a que la información final que se muestra, es solicitada solamente de la tabla (S), entonces es posible reemplazar esta consulta anidada por la siguiente consulta:

```
SELECT DISTINCT(SNOMBRE)
FROM S,SP,P
WHERE S.S# = SP.S# and SP.P#= P.P#  and  P.COLOR = 'Rojo';
```

SUBCONSULTA CON OPERADOR DE COMPARACIÓN DISTINTO DE IN:

Obtener los números de los proveedores situados en la misma ciudad que el proveedor S1.

```
SELECT  S#
FROM    S
WHERE CIUDAD =
                (SELECT CIUDAD
                FROM S
                WHERE S# = 'S1' );
```

```
-----
S#
-----
S1
S4
```

Resultado:

FUNCIÓN DE AGREGADOS EN UNA SUBCONSULTA:

Obtener los números de los proveedores cuya situación sea menor que el valor máximo actual de situación en la tabla S.

```
SELECT  S#
FROM    S
WHERE SITUACION <
                (SELECT MAX (SITUACION)
                FROM S);
```

```
------
S#
------
S1
S2
S4
```

Resultado:

CONSULTA CON UNION:

Obtener los números de las partes que pesen más de 16 libras,o sean suministradas por el proveedorr S2 (o las dos cosas).

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

Resultado:

P1
P2
P3
P6

OPERACIONES DE ACTUALIZACION.

El lenguaje de manipulación de datos de SQL incluye tres operaciones de actualización:

- UPDATE (actualizar en el sentido de alterar o modificar).
- DELETE(eliminar).
- INSERT(insertar).

❖ UPDATE (ACTUALIZAR)

Formato general:

```
UPDATE tabla
SET  campo = expresión –escalar
     [, campo = expresión –escalar ]  .....
[WHERE condición] ;
```

Todos los registros de "tabla" que satisfagan "condición" serán modificados de acuerdo con las asignaciones ("campo = expresión escalar") de la cláusula SET (establecer).

MODIFICACIÓN DE UN SOLO REGISTRO:

Cambiar a amarillo el color de la parte P2, aumentar su peso en 5 e indicar que su ciudad es "desconocida" (NULL).

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

MODIFICACIÓN DE VARIOS REGISTROS:

Duplicar la situación de todos los proveedores situados en londres.

```
UPDATE   S
SET  SITUACION  = 2 * SITUACION
WHERE CIUDAD = 'Londres' ;
```

MODIFICACIÓN CON SUBCONSULTAS.

Poner en ceros la cantidad enviada por todos los proveedores de Londres.

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

- **DELETE (ELIMINAR):**

Formato general:

```
DELETE
FROM tabla
[WHERE condición] ;
```

Se eliminarán todos los registros de "tabla" que satisfagan "condición".

ELIMINACIÓN DE UN SOLO REGISTRO:

Eliminar el proveedor S5.

```
DELETE
FROM S
WHERE S# = 'S5' ;
```

ELIMINACIÓN DE VARIOS REGISTROS:

Eliminar todos los envíos cuya cantidad sea mayor que 300.

```
DELETE
FROM SP
WHERE CANT > 300 ;
```

ELIMINACIÓN DE VARIOS REGISTROS:

Eliminar todos los embarques.

```
DELETE
FROM SP ;
```

ELIMINACIÓN CON SUBCONSULTAS

Eliminar todos los envíos de los proveedores situados en Londres.

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

- **INSERT (INSERTAR)**

Formato general:

```
INSERT
INTO tabla  [ ( campo [ , campo ] .... ) ]
VALUES ( literal [ , literal ] ... );
```

O bien

```
INSERT
INTO tabla  [ ( campo [ , campo ] .... ) ]
subconsulta;
```

INSERCIÓN DE UN SOLO REGISTRO:

Añadir la parte P7 (ciudad, Atenas, peso, 24, nombre y color desconocidos por ahora) a la tabla P.

```
INSERT
INTO P ( P#, CIUDAD, PESO)
VALUES ( 'P7', 'Atenas', 24);
```

Se crea un nuevo registro de parte, con el número de parte, la ciudad y el peso especificados y con nulos en las posiciones de nombre y color.

INSERCIÓN DE UN SOLO REGISTRO, OMITIENDO NOMBRES DE CAMPOS:

Añadir la parte P8(nombre, Cadena, color, rosa, peso, 14, ciudad, Niza) a la tabla P.

```
INSERT
INTO P
VALUES ( 'P8', ''Cadena', 'Rosa', 14, 'Niza');
```

Omitir la lista de campos equivale a especificar una lista con todos los campos de la tabla .

INSERCIÓN DE UN SOLO REGISTRO:

Insertar un nuevo envío con número de proveedor S20, número de parte P20 y cantidad 1000.

```
INSERT
INTO SP ( S#, P#, CANT)
VALUES ( 'S20', 'P20', 1000 ) ;
```

INSERCIÓN DE VARIOS REGISTROS:

Para cada parte suministrada,obtener el número de parte y la cantidad total suministrada, y guardar el resultado en la base de datos.

```
CREATE TABLE TEMP
   ( P#        CHAR(6) NOT NULL,
   CANTTOTAL INTEGER NOT NULL,
   PRIMARY KEY (P#);

CREATE UNIQUE INDEX XT ON TEMP (P#);

INSERT  INTO TEMP (P# , CANTTOTAL)
SELECT P# , SUM(CANT)
FROM SP
GROUP BY P;
```

Se ejecuta la proposición SELECT, igual que en una selección ordinaria, pero el resultado, en vez de presentarse al usuario, se copia en la tabla TEMP.

Cuando ya no se necesite, la tabla TEMP podrá desecharse:

```
DROP TABLE TEMP.
```

ÁLGEBRA RELACIONAL

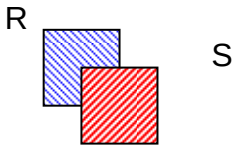
OPERACIONES ENTRE ESTRUCTURAS:

Se basan en el **'Álgebra relacional'** considerando las tablas (las relaciones) como conjuntos.
Inicialmente se definen 8 operaciones de **Álgebra relacional**:

Sean las relaciones **R** y **S**:

1) UNIÓN: **R ∪ S**

R y S deben de ser del mismo grado. El resultado es una relación con todas las filas de R y S con una única concurrencia si hay repeticiones.



R

A	B	C
D	A	T
C	B	D

S

B	G	A
D	A	T

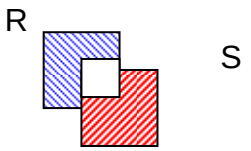
R ∪ S

A	B	C
D	A	T
C	B	D
B	G	A

En una consulta de unión se ve el resultado pero no se almacena en ninguna tabla al contrario que en una consulta de datos añadidos que si que se guardan los cambios en otra tabla.

2) DIFERENCIA: **R - S**

R y S deben de ser de igual grado. El resultado incluye todas las filas de R que no están en S.



R

A	B	C
D	A	T
C	B	D

S

B	G	A
D	A	T

R - S

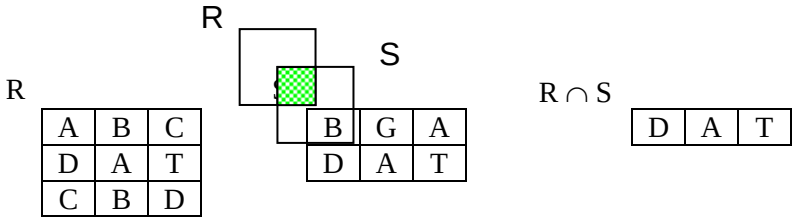
A	B	C
C	B	D

En SQL no existe una palabra específica para esta relación, se tiene que montar con:

NOT EXIST
NOT IN

3) INTERSECCIÓN: R ∩ S

El resultado son las filas que están en R y S.



A	B	C
D	A	T
C	B	D

B	G	A
D	A	T

D	A	T

En SQL no existe una palabra específica para esta relación, se tiene que montar con:

EXIST
IN

4) PRODUCTO CARTESIANO: R ∗ S

Devuelve el resultado de concatenar todas las filas de R con cada una de las filas de S.

R

A	B
C	D
E	F

S

B	X
D	Y

R ∗ S

A	B	B	X
A	B	D	Y
C	D	B	X
C	D	D	Y
E	F	B	X
E	F	D	Y

Col = ColR + ColS
Fil = FilR x FilS

En SQL:

SELECT *
FROM R,S

5) PROYECCIÓN:

Devuelve todas las filas de R con las columnas seleccionadas.

R R_1 R_2

A	B
C	D
E	F

R p (R_1)

A
C
E

En SQL:

SELECT R_1
FROM R

6) SELECCIÓN (restricción):

Devuelve todas las filas que cumplan una condición.

R R_1 R_2

A	B
C	D
E	F
G	B

R_S (R_2 = "B")

A	B
G	B

Se aplican a los valores de una o más columnas.

En SQL:

SELECT
FROM R
WHERE $R_2 = R_1$

7) DIVISIÓN: (R binaria, S unitaria)

Devuelve una relación formada por todos los valores de una columna de R que concuerdan con todos los valores de S.

R

A	X
A	Y
A	Z
B	X
C	Y

S

X
Z

R / S

A

8) REUNIÓN NATURAL (join, yunion):

Devuelve un subconjunto sobre el producto cartesiano R x S en que los valores de una columna de R son iguales a los de una columna de S. Estas columnas se llaman columnas de composición.

R

A	B
C	D
E	F

S

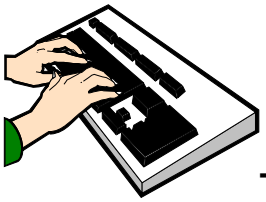
B	X
D	Z

$R \cup S$

A	B	X
C	D	Z

SINTAXIS:

R	∪	S	Unión
R	∩	S	Intersección
R	X	S	Producto Cartesiano
R	DIVISIÓN	S	DIVISIÓN Entre R Y S
R	-	S	Diferencia



TRABAJO EN LABORATORIO...

1) Pruebe en access los ejemplos dados.
2) Cree las tablas ALUMNOS, MATERIA y EXAMENES(en forma visual)
3) inserte registros en las tablas creadas, modifique los datos y elimine registros. utilice para ello las proposicones DML. (lenguaje de manipulación de datos).

TRABAJO EN GRUPO .

- En la unidad XIII: “Ejemplos de Aplicación”. Interprete y compruebe las consultas sql, que allí se ejemplifican.

ACTIVIDAD Nº 6:

- Realizar los siguientes trabajos prácticos
- Desarrollarlos en forma manual y comprobarlos en la computadora.

TRABAJO PRÁCTICO Nº 6.1

dni	nombre	direccion	ciudad_ natal	cp	telefono	Tit_secu	carr_ actual	est_ civil
16883997	RODRIGUEZ L.	ZABALA 32	SALTA	4400	4245368	TECNICO	SISTEMAS	C
14234567	ROMERO ARIEL	ZAPORTA 342	CATAMARC	4200	31254	BACHILLER	SISTEMAS	S
13234653	BALMON LUIS	CATAMARCA 32	JUJUY	4700	4251638	BACHILLER	PSICOPE DAGOGIA	C
18765678	PEREZ JOSE	LAVALLE 32	SALTA	4200	4215689	COMERCIAL	MARKETI NG	S
23456098	GIMENEZ JULIO	BUENOS AIRES 43	SALTA	4400	4312587	TECNICO	PSICOPE DAGOGIA	S
13452345	TORO MARIO	DEAN FUNES 123	JUJUY	4700	4258745	COMERCIAL	MARKETI NG	S
25432009	DORIAN JOSE	PUEYRREDON 62	SALTA	4400	4312564	BACHILLER	SISTEMAS	C
22345213	DORADO PEDRO	RIVADAVIA 451	SALTA	4400	4215784	TECNICO	SISTEMAS	S

ALUMNOS

DESARROLLAR:

1-SELECT ANIDADO: Obtener los nombres de los alumnos que viven en la misma ciudad que el alumno RODRIGUEZ L.

```
SELECT nombre
FROM ALUMNOS
WHERE ciudad_natal  IN
        (SELECT ciudad_natal
         FROM ALUMNOS
         WHERE nombre="RODRIGUEZ L") ;
```

Nombre
RODRIGUEZ L
GIMENEZ JULIO
PEREZ JOSE
DORIAN JOSE
DORADO PEDRO

2-COUNT: Obtener la cantidad de alumnos que son solteros.
3-MAX: Mostrar el alumno que tenga el máximo dni y que viva en SALTA.
4-SELECT ANIDADO: Mostrar el dni y nombre de los alumnos que cursen la misma carrera que el alumno RODRIGUEZ L y que sean casados.
5-MIN: Mostrar el menor nombre (según el orden alfabético) del alumno que cursa la carrera de analista y viva en SALTA.
6-Mostrar el dni de los alumnos cuyo nombre comienza con la letra D y que cursen la carrera de SISTEMAS.
7-Mostrar dni, nombre, dirección y teléfono de los alumnos cuyo c_postal sea mayor que 4300 y el dni menor que 22000000.
8-Mostrar el dni y el nombre de los alumnos que sean solteros o que cursen la carrera de PSICOPEDAGOGIA.

9-Mostrar el estado civil de los alumnos cuyo teléfono sea mayor que 4213456 y su nombre contenga la frase PE (ej. PEREZ, PEDRO y YUSEPE contiene la frase PE).
10-Mostrar el dni, nombre y dirección cuyo código postal sea distinto de 4400 y el título secundario sea TECNICO.
11-Mostrar el nombre de los alumnos cuya dirección contenga la letra I.
12-Mostrar el dni, nombre y código postal de los alumnos. En forma descendente por dni y ascendente por código postal.
13-GROUP BY: Mostrar la cantidad de alumnos que asisten al colegio agrupados por ciudad natal.
14-GROUP BY: Mostrar la cantidad de alumnos que asisten al colegio agrupados por estado civil.
15-GROUP BY: Mostrar el mínimo dni de los alumnos agrupados por su estado civil.
16-HAVING: Mostrar la cantidad de alumnos que estudian la misma carrera, siempre que tengan como mínimo tres alumnos.
17-HAVING: Mostrar la cantidad de alumnos agrupados por estado civil, siempre que estén casado.

```
SELECT count(*), est_civil
FROM ALUMNOS
GROUP BY est_civil
HAVING est_civil="C";
```

TRABAJO PRÁCTICO Nº 6.2

FACTURAS				
NUM_FACT@	**FECHA**	**TOTAL**	**DNI**	**DNI_VEND**
1568	23/01/2005	$460	12356897	15235487
1569	23/01/2005	$85	14231564	14254368
1570	25/01/2005	$170	12356897	14254368

Dada la tabla factura genere:
10 preguntas que incluyan select (funciones de agregado, distinct, like, in, etc)
2 preguntas que incluyan order by
2 preguntas que incluyan group by.
1 pregunta que incluya having.

TRABAJO PRÁCTICO Nº 6.3

CLIENTES					
DNI@	**NOMBRE**	**DIRECCIÓN**	**TELÉFONO**	**DIREC_ENVIO**	**CIUDAD_ENVIO**
12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	SALTA
10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	SALTA
18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	JUJUY
19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	CATAMARCA
14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	JUJUY

FACTURAS				
NUM_FACT@	**FECHA**	**TOTAL**	**DNI**	**DNI_VEND**
1568	23/01/2005	$460	12356897	15235487
1569	23/01/2005	$85	14231564	14254368
1570	25/01/2005	$170	12356897	14254368

DETALLE		
NUM_FACT@	**COD_MERCAD@**	**CANTIDAD**
1568	023	20
1568	053	50
1568	013	10
1568	014	10
1569	009	5
1569	053	15
1570	008	20
1570	023	8
1570	010	30

Dada las tablas CLIENTES, FACTURAS y DETALLE; las cuales se encuentran relacionadas, responder:

DOS TABLAS

1-Obtener: El nombre de los clientes que realizaron alguna compra.
2-Obtener: La dirección de los clientes que realizaron alguna compra, el 23/01/2005.
3-Obtener: El código de la mercadería comprada por el cliente con dni=12356897.

TRES TABLAS (USAR IN CUANDO SEA POSIBLE)

4-Mostrar el nombre del cliente que adquirió la mercadería 008.
5-Indicar la cantidad total de la mercadería 023 adquirida por el cliente GOMEZ, LUIS.

6-Generar 2 preguntas que involucren a dos tablas.
7-Generar 2 preguntas que involucren a tres tablas.

Unidad VII: Vistas

DEFINICIÓN DE VISTAS

En el nivel externo de la arquitectura ANSI/SPARC, la base de datos se percibe como una "vista externa", definida por el esquema externo. Diferentes usuarios tienen diferentes vistas. El término "vistas" se reserva en SQL para referirse a una **tabla virtual derivada con nombre**.

Una vista en una tabla virtual, es decir una tabla que no existe en sí pero ante el usuario parece existir. He aquí un ejemplo:

```
CREATE VIEW BUENOS_PROVEEDORES
    AS SELECT S#,  SITUACIÓN,  CIUDAD
    FROM  S
    WHERE  SITUACIÓN  >  15 ;
```

De hecho, BUENOS_PROVEEDORES es una "ventana" a través de la cual se ve la tabla real S. Además, esa ventana es dinámica, lo que implica que las modificaciones hechas a S serán visibles automática e instantáneamente a través de esa ventana. Las modificaciones hechas a BUENOS_PROVEEDORES se aplicarán en forma automática e instantánea a la tabla real S.

El usuario podrá o no darse cuenta de que BUENOS_PROVEEDORES es en realidad una vista, algunos usuarios pueden estar conscientes de ese hecho y comprender que existe una tabla real S debajo de ella, otros pueden creer que BUENOS_PROVEEDORES es una tabla real en sí misma. Lo fundamental es que los usuarios puedan trabajar con BUENOS_PROVEEDORES como si fuera una tabla real. He aquí un ejemplo de operación de recuperación de datos:

```
SELECT  *
FROM    BUENOS_PROVEEDORES
WHERE   CIUDAD <> ´Londres´  ;
```

Resultado:

S#	SITUACIÓN	CIUDAD
S3	30	París
S5	30	Atenas

Esta proposición SELECT tiene todo el aspecto de una selección normal realizada sobre una tabla base normal. El sistema maneja este tipo de operaciones convirtiéndolas en una operación equivalente realizada sobre la taba base subyacente. En el ejemplo, la operación equivalente es:

BUENOS_PROVEEDORES

S#	SNOMBRES	SITUACIÓN	CIUDAD
S1	Salazar	20	Londres
S2	Jaimes	10	París
S3	Bernal	30	París
S4	Corona	20	Londres
S5	Aldana	30	Atenas

Fig. 7.1 BUENOS_PROVEEDORES como una vista de la tabla base S (partes no sombreadas)

```
SELECT S#, SITUACIÓN, CIUDAD
FROM   S
WHERE  CIUDAD <> ´Londres´
AND    SITUACIÓN  > 15   ;
```

Las operaciones de actualización se manejan de manera similar. Por ejemplo, la operación:

```
UPDATE BUENOS_PROVEEDORES
SET    SITUACIÓN  =  SITUACIÓN  +  10
WHERE  CIUDAD  =  ´París´
```

Será convertida por el ligador en

```
UPDATE  S
SET    SITUACIÓN  =  SITUACIÓN  +  10
WHERE  CIUDAD  =  ´París´
AND    SITUACIÓN  > 15   ;
```

Las operaciones de inserción y eliminación se manejan en forma análoga.

VISTAS (sintaxis):

La sintaxis general de la operación CREATE VIEW (crear vista) en SQL es

```
CREATE VIEW vista
```

En principio, cualquier tabla derivable –es decir, cualquier tabla que puede obtenerse mediante una proposición SELECT– puede en teoría definirse como una vista. El motor no permite incluir el operador UNION en una definición de vista.

Ejemplos de CREATE VIEW (crear vista):

```
CREATE VIEW PARTESROJAS (P#, PNOMBRE, PESO, CD)
    AS SELECT P#, PNOMBRE, PESO, CIUDAD
    FROM P
    WHERE COLOR = 'Rojo';

CREATE VIEW PC (P#, CANTTOT)
    AS SELECT P#, SUM(CANT)
    FROM SP
    GROUP BY P#;
```

En este ejemplo no existe un nombre que pueda heredar la segunda columna, pues ésta se deriva de una función.

```
CREATE VIEW PAREJAS_DE_CIUDADES (SCIUDAD, PCIUDAD)
    AS SELECT DISTINTIC S.CIUDAD, P.CIUDAD
    FROM S, SP,P
    WHERE S.S# = SP.S#
    AND SP.P# = P.P# ;
```

El significado de esta vista en particular es que aparecerá una pareja de nombres de ciudad (x,y) en la vista si un proveedor en la ciudad *x* suministra una parte almacenada en la ciudad *y*. Por ejemplo, el proveedor S1 suministra la parte P1; ese proveedor está situado en Londres, y esa parte se almacena en Londres; Por tanto, esa pareja (Londres, Londres) aparecerá en la vista. Nótese que la definición de esta vista implica una reunión, de manera que éste en un ejemplo de vista derivada de varias tablas subyacentes. Adviértase que los nombres de columna deben especificarse de manera explícita, pues de lo contrario las dos columnas de la vista tendrían el mismo nombre, CIUDAD.

La sintaxis de DROP VIEW (desechar vista) es

```
DROP VIEW vista;
```

Por ejemplo:

```
DROP VIEW PARTESROJAS;
```

Si se desecha una tabla (tabla base o vista), se desecharán también en forma automática todas las vistas definidas en término de esa tabla.

No existe una proposición ALTER VIEW (alterar vista). La alteración de una tabla base (mediante ALTER TABLE) no afecta a las vistas ya existentes.

OPERACIONES DE DML SOBRE VISTAS.

No todas las vistas se pueden actualizar. Como ilustración, consideraremos las siguientes dos vistas, S#_CIUDAD y SITUACIÓN_CIUDAD, siendo ambas vistas de la misma tabla base S:

```
CREATE VIEW S#_CIUDAD
        AS SELECT  S#, CIUDAD
        FROM S;

CREATE VIEW SITUACION_CIUDAD
        AS SELECT  SITUACION, CIUDAD
        FROM S;
```

De estas dos vistas, S#_CIUDAD se puede actualizar (en teoría), en tanto que SITUACIÓN_CIUDAD no se pueden (también en teoría). Es instructivo examinar las razones de ello.

En el caso de S#_CIUDAD:

- Se puede insertar un registro nuevo en la vista, digamos el registro (S6,Roma), mediante la inserción real del registro correspondiente (S6,NULL,NULL,Roma) en la tabla base subyacente.
- Se puede eliminar un registro ya existente de la vista, digamos el registro (S1,Londres), eliminando en realidad el registro correspondiente (S1,Salazar,20,Londres) de la tabla base subyacente.
- Se puede modificar un campo ya existente en la vista, digamos cambiar la ciudad del proveedor S1 de Londres a Roma, haciendo en realidad la misma modificación sobre el campo correspondiente en la tabla base subyacente.

En cambio, en el caso de SITUACIÓN_CIUDAD:

- Si tratamos de insertar un registro nuevo en la vista, digamos el registro (40,Roma), el sistema tendrá que tratar de insertar el registro correspondiente (NULL, NULL,40,Roma) en la tabla base subyacente, y esa operación fracasará, pues los números de proveedor se han definido como no nulos (NOT NULL).
- Si tratamos de eliminar algún registro ya existente de la vista, digamos el registro (20,Londres), el sistema tendrá que intentar eliminar algún registro correspondiente de la tabla base subyacente; pero ¿cuál?. El sistema no tiene forma de saberlo, porque no se ha especificado el número de proveedor (y no puede especificarse, porque el campo S# no es parte de vista).
- Si tratamos de modificar algún registro ya existente en la vista, digamos cambiar el registro (20,Londres) a (20,Roma), el sistema tendrá que tratar de modificar algún registro correspondiente en la tabla base subyacente en la tabla base subyacente; pero, de nuevo, ¿cuál?

Si inspeccionamos las vistas S#_CIUDAD y SITUACIÓN_CIUDAD veremos que la diferencia crítica entre ellas es que S#_CIUDAD incluye la clave primaria (es decir, S#) de la tabla subyacente, y SITUACIÓN_CIUDAD no.

En el caso de S#_CIUDAD, es precisamente la presencia del campo S# lo que hace posible identificar el registro correspondiente que se va a modificar en la tabla subyacente. Y a la inversa, en la tabla SITUACIÓN_CIUDAD, es precisamente la ausencia de ese campo lo que imposibilita la identificación de un registro correspondiente.

Las dos vistas, S#_CIUDAD y SITUACIÓN_CIUDAD se pueden caracterizar como vistas de "subconjunto de columnas": cada una está formada por un subconjunto de las columnas de una sola tabla subyacente. Como se demuestra en el ejemplo, **una vista de "subconjunto de columnas" teóricamente se puede actualizar si conserva la clave primaria de la tabla subyacente** (suponiendo, desde luego, que es posible poner al día primero la tabla subyacente en sí; podría resultar ser ella misma una vista no actualizable).

En realidad existen diferentes casos en que no se pueden actualizar las vistas.

INDEPENDENCIA LÓGICA DE LOS DATOS

Todavía no hemos explicado en realidad para que sirven las vistas. **Uno de sus propósitos es lograr lo que se ha dado en llamar *independencia lógica de los datos.***

Se dice que un sistema DBMS proporciona independencia física de los datos porque los usuarios y sus programas no dependen de la estructura física de la base de datos almacenada.

Se dice que un sistema ofrece independencia *lógica* de los datos si los usuarios y sus programas son asimismo independientes de la estructura lógica de la base de datos. Este segundo tipo de independencia presenta dos aspectos, a saber, *crecimiento y reestructuración.*

CRECIMIENTO:

Conforme crezca la base de datos para incorporar nuevos tipos de información, así también deberá crecer la definición de la base de datos. Se pueden presentar dos posibles tipos de crecimiento:

1) La expansión de una tabla base ya existente para incluir un campo nuevo.
2) La inclusión de una nueva tabla base.

Ninguno de estos dos tipos de modificaciones deberá afectar en absoluto a los usuarios ya existentes.

REESTRUCTURACIÓN:

Vamos a suponer, que resulta necesario sustituir la tabla S por estas dos tablas base:

SX (S#, SNOMBRE, CIUDAD)

SY (S#, SITUACION)

La antigua tabla S es la reunión de las dos nuevas tablas SX y SY.
Creamos una vista que sea exactamente esa reunión, y la llamamos S:

```
CREATE VIEW S (S#, SNOMBRE, SITUACION, CIUDAD)
    AS SELECT SX.S#, SX.SNOMBRE, SY.SITUACION, SX.CIUDAD
    FROM SX, SY
    WHERE SX.S# = SY.S# ;
```

Cualquier programa que hiciera referencia antes a la tabla base S hará referencia ahora a la vista S. Las operaciones SELECT seguirán funcionando tal como lo hacían antes (aunque requerirán un análisis adicional durante el proceso de ligado). Sin embargo, las operaciones de actualización ya no funcionarán, pues no se permiten actualizaciones de vistas definidas como reuniones.

VENTAJAS DE LAS VISTAS

- Ofrecen un cierto grado de independencia lógica de los datos en casos de reestructuración de la base de datos.
- Permiten a diferentes usuarios ver los mismos datos de distintas maneras.
- Se simplifica la percepción del usuario.
- Se cuenta con seguridad automática para datos ocultos.

"Datos ocultos" se refiere a información no visible a través de una vista dada.

ACTIVIDAD Nº 7:

- **Investigar: ¿ Para qué sirven las Vistas ?. Hacer un informe ejemplificando casos de aplicación.**
-

Unidad VIII: El Modelo Relacional.

ESTRUCTURA DE DATOS RELACIONAL

El modelo relacional se divide en tres partes, las cuales se ocupan de:

- La estructura.
- La integridad.
- La manipulación.

En relación con la estructura de los datos, los propios términos especiales se muestran en la figura 8.1, utilizando la relación S de la base de datos de proveedores y partes como ilustración.

Los términos en cuestión son ***relación*****,** ***tupla, cardinalidad, atributo, grado, dominio y clave primaria.***

- Una ***relación*** corresponde a lo que hasta ahora hemos llamado en general tabla.
- Una ***tupla*** corresponde a una fila de esa tabla y un *atributo* a una columna. El número de tuplas se denomina ***cardinalidad*** y el número de atributos se llama **grado**.
- La *clave primaria* es un identificador único para la tabla; es decir, una columna o combinación de columnas con la siguiente propiedad: nunca existen dos filas de la tabla con el mismo valor en esa columna o combinación de columnas.
- Un ***dominio*** es una colección de valores, de los cuales uno o más atributos (columnas) obtienen sus valores reales.

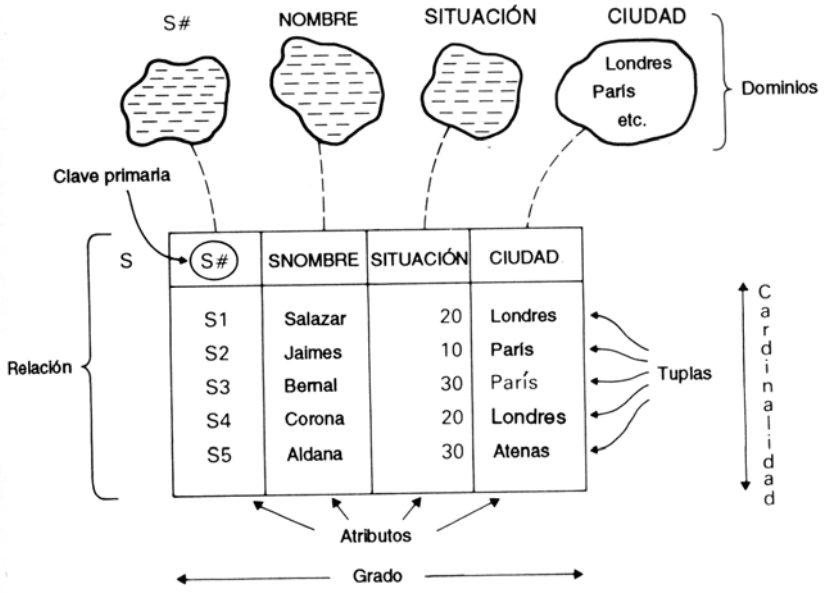


Fig. 8.1 La relación de proveedores S

Estos términos se resumen en la tabla de la figura 8.2.

Las equivalencias mostradas son solo aproximadas, los términos situados a la izquierda tienen definiciones precisas, pero los "equivalentes" informales de la derecha sólo poseen definiciones aproximadas, aunque prácticas.

Término relacional formal	Equivalentes informales
relación	tabla
tupla	fila o registro
cardinalidad	número de filas
atributo	columna o campo
grado	número de columnas
clave primaria	identificador único
dominio	fondos de valores legales

Fig 8.2: Terminología de la estructura de los datos

DOMINIOS

La ***menor unidad semántica de información*** es el valor de un dato individual (como el número de un proveedor individual). Llamaremos a estos valores **"escalares"**. Los valores escalares representan "la menor unidad semántica de información", en el sentido de que son *atómicos,* es decir no poseen estructura interna, no se pueden descomponer.

Por ejemplo, un nombre de ciudad sin duda tiene una estructura interna, sin embargo, si descomponemos un nombre de ciudad en sus letras componentes, perdemos significado. El significado sólo se hace evidente si las letras aparecen todas juntas y en la secuencia correcta.

Definimos **un *dominio* como un conjunto de valores escalares**, **todos del mismo tipo**. Por ejemplo, el dominio de números de proveedor es el conjunto de todos los números de proveedor posibles.

Así, **los dominios son *fondos de valores*, de los cuales se extraen los valores reales** que aparecen en los atributos. Cada valor del atributo P.P# debe ser un valor del dominio de números de parte, y lo mismo sucede con el atributo SP.P#.

Será común encontrar en un dominio, determinados valores que no aparecen de momento en ninguno de los atributos correspondientes a ese dominio. Por ejemplo, si el valor P8 es un número de parte legal, aparecerá en el dominio de número de parte, aunque no aparezca de hecho en la relación P de nuestra base de datos de proveedores y partes (Fig. 4.1); es decir, la parte P8 no existe en realidad en este momento.

Siempre se da por hecho que los dominios son simples si no se indica de manera explícita lo contrario.

Los dominios restringen las comparaciones.

```
SELECT   P.*, SP.*
FROM      P, SP
WHERE   P.P# = SP.P# ;
```

```
SELECT   P.*, SP.*
FROM     P, SP
WHERE    P. PESO  =  SP.CANT;
```

De estas dos consultas, la primera con toda probabilidad es razonable, pero la segunda probablemente no lo es. La consulta primera incluye una comparación entre dos atributos, P.P# y SP.P# definidos sobre el mismo dominio, en tanto que la segunda incluye una comparación entre dos atributos, P.PESO y SP.CANT definidos (al parecer) sobre dominios distintos.

Una ventaja del manejo de dominios por parte del sistema es que lo hace capaz de impedir errores tontos de los usuarios.

Los dominios *compuestos,* en lo esencial, no es mas que una combinación de dominios simples.

```
CREATE DOMAIN DIA       CHAR(2) ;
CREATE DOMAIN MES       CHAR(2) ;
CREATE DOMAIN AÑO       CHAR(4) ;
CREATE DOMAIN FECHA    (DÍA DOMAIN ( DÍA ) ,
                           MES DOMAIN ( MES ) ,
                           AÑO DOMAIN ( AÑO)  ) ;
```

El dominio compuesto FECHA tiene tres componentes llamados DÍA, MES y AÑO. Si el conjunto de valores de DÍA es de 1 a 31, el conjunto de valores de MES es de 1 a 12, y el conjunto de valores de AÑO es de 0 a 1999, el conjunto de valores de FECHA será:

```
 1          1          0
 2          1          0
 3          1          0
 .          .          .
31          1          0
 1          2          0
 .          .          . .
 .          .          ...
31          12       9999
```

Un total de 12*31*10000=3.720.000 valores (no todos esos 3.720.000 valores son fechas válidas).

Los atributos a su vez son simples o compuestos dependiendo de la naturaleza del dominio sobre el cual se definen.

```
CREATE TABLE EMPLEADO
        ( NUMEMP            DOMAIN  ( NUMEMP ) ,
            . . . . .  ,
          FECHA_ INGRESO DOMAIN  ( FECHA )  ,
            . . . . .  ,
          PRIMARY KEY   ( NUMEMP )  )  ;
```

Aquí, FECHA_INGRESO es un atributo compuesto. Las operaciones relacionales deberán ser capaces de hacer referencia a ese atributo en su totalidad y a sus componentes DÍA(FECHA_INGRESO) en forma individual, empleando los nombres de componentes DÍA, MES y AÑO como *funciones selectoras de componentes*. Por ejemplo:

```
SELECT . . .
FROM     EMPLEADO
WHERE    EMPLEADO.FECHA_INGRESO = '01011984' ;

SELECT . . .
FROM     EMPLEADO
WHERE    AÑO  ( EMPLEADO.FECHA_INGRESO  )  <  '1984'  ;
```

Por desgracia muy pocos de los sistemas actuales, aun aquellos que si admiten dominios, manejan en la práctica tales atributos compuestos.

Un dominio no es ni más ni menos que un *tipo de datos*. Por ejemplo en el lenguaje Pascal es legal lo siguiente:

```
TYPE  MES  =  (  1. . 12  );
    VAR  MES_ INGRESO  :  MES;
```

SQL (por ejemplo) maneja dominios en un sentido muy primitivo, en tanto incluye ciertos tipos de datos primitivos integrados (INTEGER, FLOAT, etc).

Cuando hablamos de manejo de dominios en el contexto de los sistemas relacionales queremos referirnos a algo mas que ese nivel primitivo, queremos decir que el sistema debe ofrecer un recurso mediante el cual los usuarios puedan definir sus propios tipo de datos más complejos como por ejemplo "números de proveedor", "números de parte", etc, con todo lo que implica un recurso así.

RELACIONES

- Entidades.- Objetos sobre los cuales queremos guardar información y la característica es que tengan existencia por si mismos.
- Relaciones.- Asociaciones entre entidades.

Una ***relación*** **(digamos R) se compone de dos partes, una *cabecera* y un *cuerpo:***

- La *cabecera* esta formada por un conjunto fijo de *atributos.* Donde cada atributo se corresponde a uno y solo uno de los dominios. La cabecera es la parte estática de la relación
- El *cuerpo* esta formado por un conjunto de *tuplas* el cual varia con el tiempo.

El número de atributos se llama grado y el número de tuplas se llama cardinalidad de la relación R. La cardinalidad varia con el tiempo, pero el grado no.

Tomaremos como ejemplo la tabla S de la figura 8.1 (a propósito no la llamaremos relación por el momento) a fin de comprobar cuánto se ajusta a esta definición.

Esa tabla tiene cuatro dominios subyacentes:

- El dominio de números de proveedor (S#).
- El dominio de nombres (NOMBRE).
- El dominio de valores de situación (SITUACIÓN)
- El dominio de nombres de ciudad (CIUDAD).

Cuando dibujamos una relación en forma de tabla en un papel, por lo regular no nos molestamos en representar los dominios subyacentes, pero debemos comprender que, al menos en lo conceptual, siempre están allí.

La tabla tiene sin duda dos partes; tiene una fila de cabeceras de columna y un conjunto de filas de datos. Examinaremos primero la fila de cabecera de columna:

(S#, SNOMBRE, SITUACIÓN, CIUDAD)

En cuanto al resto de la tabla, ciertamente esta formado por un conjunto, a saber, un conjunto de filas. Además, ese conjunto sin duda varia con el tiempo. Se insertan nuevas filas en el conjunto y se modifican ó se eliminan otras filas ya existentes. La figura 8.1 es solo una instantánea de los valores que aparecen en un momento dado en la base de datos.

Si observamos una de las filas, digamos la fila

(S1, SALAZAR, 20, LONDRES)

Vemos que cada fila representa una tupla, en el sentido de la definición.

La tabla S de la figura 8.1 sí puede considerarse efectivamente como una representación de una relación en el sentido de la definición.

Así, ahora podemos dar cuenta de que una tabla y una relación no son en realidad la misma cosa, **una relación es una especie bastante abstracta de objeto**; **y una tabla es una representación concreta de tal objeto abstracto.** En contextos informales, es usual, y del todo aceptable, decir que son las mismas cosas, es decir que **"una relación es una tabla"**.

Terminología: el valor n (el número de atributos) se llama grado de la relación. Una relación de grado uno se llama *unaria;* una relación de grado dos, *binaria*; una relación de grado tres, *ternaria;*....., y una relación de grado n, n-aria. Así el modelo relacional se ocupa, en general, de las relaciones n-arias, para un entero arbitrario *n* no negativo. En la base de datos de proveedores

y partes, los grados de las relaciones S, P y SP son 4, 5 y 3, respectivamente.

El valor m (el número de tuplas en una relación) se llama *cardinalidad* de la relación. La cardinalidad cambia con el tiempo, pero el grado no.

Los dominios son estáticos, las relaciones son dinámicas. El contenido de una relación cambia con el tiempo, pero el de un dominio no.

Los dominios adyacentes de una relación no son "necesariamente todos distintos".

Ej. una relación que incluya Ciudad de la tabla P y Ciudad de la tabla S.

- **PROPIEDADES DE LAS RELACIONES:**

1) No existen tuplas repetidas.
2) Las tuplas no están ordenadas (de arriba hacia abajo).
3) Los atributos no están ordenados (de izquierda a derecha).
4) Todos los valores de los atributos son atómicos.

No existen tuplas repetidas:

La relación es un conjunto matemático (es decir, un conjunto de tuplas), **y en matemáticas los conjuntos por definición no incluyen elementos repetidos**.

ESTRUCTURA_DE_PARTES

NUMP_PRINCIPAL	NUMP_SECUNDARIA	CANT
P1	P2	2
P1	P3	4
P2	P3	1
P2	P4	3
P3	P5	9
P4	P5	8
P5	P6	3

Fig 8.3: la relación ESTRUCTURA_DE_PARTES

Esta primera propiedad nos sirve de inmediato para ilustrar la diferencia entre una relación y una tabla, porque **una tabla (en general) podría contener filas repetidas** -al faltar una disciplina que evite tal situación- pero **una relación no puede contener tuplas repetidas** (pues si una "relación" contiene tuplas duplicadas, no es una relación, por definición). Pero desafortunadamente **la mayor parte de los DBMS relacionales disponibles en el mercado si permiten filas repetidas en las tablas**.

Un corolario importante de la ausencia de tuplas repetidas es que siempre existe una clave primaria, esto implica que posee la propiedad de unicidad, de modo que al menos la combinación de todos los atributos puede servir (en caso necesario) como clave primaria. En la práctica casi nunca resulta necesario implicar a todos los atributos; así, por ejemplo, la combinación (S#, CIUDAD), aunque "única", no es la clave primaria de la relación S, porque es posible desechar el atributo CIUDAD sin destruir la propiedad de unicidad

.

Las tuplas no están ordenadas (de arriba hacia abajo):

El cuerpo de una relación es un conjunto matemático. Los conjuntos matemáticos no son ordenados. En la figura 8.1, las tuplas de la relación S bien podrían haberse presentado en la secuencia inversa.

Las filas de una tabla tienen un orden obvio de arriba hacia abajo en tanto que las tuplas de una relación carecen de tal orden.

Los atributos no están ordenados (de izquierda a derecha):

La cabecera de una relación se define también como conjunto. En la figura 8.1, por ejemplo, los atributos de la relación S podrían haberse presentado en el orden (digamos) SNOMBRE, CIUDAD, SITUACIÓN, S#; de todos modos sería la misma relación, al menos en lo concerniente al modelo relacional. Así pues, no puede hablarse de “el primer atributo” o “el segundo atributo” (etc.). A los atributos siempre se hacen referencia por su nombre no por su posición. Esta situación contrasta con la que se presenta en muchos sistemas de programación, donde a menudo es posible aprovechar la adyacencia física del elemento discreto del punto de vista lógico.

Las columnas de una tabla tienen un orden evidente de izquierda a derecha, pero los atributos de una relación carecen de tal orden.

Todos los valores de los atributos son atómicos:

Así **en cada posición de fila y columna** dentro de la tabla, siempre **existe un solo valor, nunca una lista de valores**. O en otra forma equivalente: **las relaciones no contienen grupos repetitivos**. **Si una relación satisface esta condición, se dice que esta normalizada.**

Todas las relaciones están normalizadas, en lo concerniente al modelo relacional, una relación matemática no necesita por fuerza estar normalizada.

Una base de datos relacional no permite relaciones que contengan grupos repetitivos, la misma debe ser sustituida por una relación equivalente en lo semántico, pero normalizada a través del proceso de normalización.

Una relación normalizada es una estructura más simple, desde el punto de vista matemático, que una no normalizada.

La normalización, conduce a la sencillez; es decir, sencillez en la estructura de los datos, lo cual conduce a su vez a simplificaciones correspondientes a todos los aspectos del sistema.

ANTES

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

DESPUÉS

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

Fig 8.4 Un ejemplo de normalización

- **TIPOS DE RELACIONES:**

1) **Relaciones base** (llamadas también relaciones reales): son aquellas cuya importancia es tal que el diseñador de la base de datos ha decidido darles un nombre y hacerlas parte directa de la base de datos en sí, a diferencia de otras relaciones cuya naturaleza es más efímera, como por ejemplo, el resultado de una consulta.
2) **Vistas** (llamadas también relaciones virtuales): una vista es una relación derivada con nombre, representada dentro del sistema exclusivamente mediante su definición en términos de otras relaciones con nombres; no posee datos almacenados propios, separados y distinguibles (a diferencia de las relaciones bases).
3) **Instantánea**: una instantánea (snapshot) es también una relación derivada, con nombre, como una vista. Pero a diferencia de las vistas, las instantáneas son reales, no virtuales; es decir, están presentadas no sólo por su definición en términos de otras relaciones con nombres, sino también por sus propios datos almacenados. Una instantánea se almacena en la base de datos con el nombre especificado como una relación de solo lectura en forma periódica la instantánea se renueva, es decir, se desecha su valor actual, se ejecuta de nuevo la consulta y el resultado de esa nueva ejecución se convierte en el nuevo valor de la instantánea.
4) **Resultados de consultas**: puede o no tener nombre. No tienen existencia persistente dentro de la base de datos.
5) **Resultados intermedios**: un resultado intermedio es una relación (casi siempre sin nombre) resultante de alguna expresión relacional anidada dentro de alguna otra expresión relacional mas grande. Por ejemplo:

```
SELECT      DISTINCT  S. CIUDAD
FROM        S
WHERE       S.S#   IN
            ( SELECT   SP.S#
              FROM     SP
              WHERE    SP.P#  =  'P2' );
```

A semejanza de los resultados finales de las consultas, no tiene existencia permanente dentro de la base de datos.

6) **Relaciones temporales**: Una relación temporal es una relación con nombre, similar a una relación base o vista o instantánea, pero se destruye en forma automática en algún momento apropiado. Las relaciones base, vistas o instantáneas, en cambio, son más permanentes, en cuanto a que solo se destruyen como resultado de alguna acción explicita del usuario.

BASES DE DATOS RELACIONALES

Una base de datos relacional es una base de datos percibida por el usuario como una **colección de relaciones normalizadas** de diversos grados que varia con el tiempo.

La frase "percibida por el usuario" es importante: las ideas del modelo relacional se aplican en los niveles externo y conceptual del sistema, no en el nivel interno.

Podemos decir que en términos tradicionales una relación se asemeja a un archivo disciplinado, en donde:

- Cada "archivo" contiene solo un tipo de registros.
- Los campos no tienen un orden específico, de izquierda a derecha.
- Los registros no tienen un orden específico, de arriba hacia abajo.
- Cada campo tiene un solo valor.
- Los registros poseen un campo identificador único (o combinación de campos) llamado clave primaria.

ACTIVIDAD Nº 8

- **Responda el siguiente cuestionario:**

1) Grafique y ejemplifique los siguientes términos: relación, tupla, cardinalidad, atributo, grado, dominio y clave primaria. Muestre las equivalencias informales de los mismos.

2) ¿Que son los valores escalares y los dominios?.

3) ¿Qué es una Entidad?.

4) ¿Qué es y como se compone una relación?.

5) ¿Es lo mismo relación que tabla?. Justifique la respuesta.

6) ¿Cuáles son las propiedades de una relación?.

7) ¿Cuándo se dice que las relaciones están normalizadas?.

8) ¿Explique Tipos de Relaciones?.

9) ¿Cómo percibe el usuario una Base de Datos Relacional?

Unidad IX: Reglas de Integridad Relacional

INTRODUCCIÓN

Cualquier base de datos "refleja la realidad" (es un modelo o representación de algún fragmento del mundo real).

La mayor parte de las bases de datos estarán sujetas a un gran número de tales reglas de integridad. Por ejemplo:

- Los valores de situación de los proveedores deben quedar en el intervalo de 1 a 100;
- Los pesos de las partes deben ser mayores que cero;

El modelo relacional Incluye dos reglas generales de integridad que se aplican a todas las bases de datos se refieren, respectivamente, a las claves primarias y las claves ajenas.

CLAVES PRIMARIAS

En términos informales, la clave primaria de una relación (o tabla) **es sólo un identificador único para esa relación (o tabla)**. O sea que si miro la tabla, por ejemplo Alumnos, esta tendrá seguramente un atributo (campo) llamado DNI y la ocurrencia de un DNI determinado aparecerá solamente una vez, es decir, por ejemplo, el DNI 12.123.123 aparecerá en una sola tupla o registro.

Recuerde que previamente se definió que: *En contextos informales, es usual decir que **"una relación es una tabla"***.

La clave primaria puede ser compuesta.

Es posible tener una relación con más de un identificador único. Como ejemplo, supongamos que cada proveedor tiene siempre un "número" de proveedor único y un "nombre" de proveedor único. En un caso así, diríamos que la relación tiene varias claves candidatas; nosotros escogeríamos entonces una de esas claves candidatas como clave primarias, y a las demás las llamaríamos entonces claves alternativas.

El atributo K (posiblemente compuesto) de la relación R es una clave candidata de R si y sólo si satisface las siguientes dos propiedades, independientes del tiempo:

1) **Unicidad:** En cualquier momento dado, no existen dos tuplas (o registros) en R con el mismo valor de K.
2) **Minimalidad:** Si K es compuesto, no será posible eliminar ningún componente de K sin destruir la propiedad de unicidad.

Toda relación (o tabla) tiene por lo menos una clave candidata, porque las relaciones no contienen tuplas repetidas.

Surgen los siguientes puntos:

1) Puesto que toda relación tiene por lo menos una clave candidata, toda relación tendrá por fuerza una clave primaria. ¡Sin excepción!

2) En términos estrictos sería más exacto decir que, por ejemplo, la clave primaria de la relación SP es el conjunto (S#,P#); de manera similar, la clave primaria de las relación S es el conjunto (S#), y así sucesivamente.
3) En ningún momento se implica la necesidad de la existencia de un índice (o cualquier otra ruta de acceso especial) según la clave primaria. En la práctica, desde luego, si habrá con toda probabilidad una ruta de acceso especial, pero el que exista o no queda (una vez más) fuera del alcance del modelo relacional en sí.

El único modo, garantizado por el sistema, de localizar alguna tupla específica es por el valor de su clave primaria. Por ejemplo si hacemos:

```
SELECT  *
FROM        P
WHERE  P#  = 'P3';
```

Obtendremos (a lo más) una tupla.
En cambio, la solicitud

```
SELECT  *
FROM        P
WHERE  CIUDAD  = 'París';
```

Obtendrá en general un número impredecible de tuplas.

LA REGLA DE INTEGRIDAD DE LAS ENTIDADES

Ningún componente de la clave primaria de una relación base puede aceptar nulos. Con "nulos" queremos decir aquí información faltante por alguna razón.

La justificación de la regla de integridad de las entidades es la siguiente:

- Las relaciones base corresponden a entidades en el mundo real. Por ejemplo, la relación base S corresponde a un conjunto de proveedores del mundo real.
- Por definición, las entidades en el mundo real son distinguibles.
- Los representantes de entidades dentro de la base de datos deben ser distinguibles.
- Las claves primarias realizan esta función de identificación única en el modelo relacional.
- Supongamos (por ejemplo) que la relación base S incluyera una tupla en la cual el valor de S fuera nulo. Eso equivaldría a decir que en el mundo real existe un proveedor sin identidad.
- Si una entidad es lo bastante importante en el mundo real como para requerir una representación explícita en la base de datos, tal entidad deberá ser definitivamente susceptible de identificación definida y sin ambigüedad, la regla de integridad de las entidades se expresa a veces así:

  En una base de datos relacional, nunca registraremos información acerca de algo que no podamos identificar.

  Surgen los siguientes puntos:

  1) En el caso de las claves primarias compuestas, la regla de integridad de entidades dice que cada valor individual de la

clave primaria debe ser no nulo en su totalidad (no sólo en parte). Por ejemplo, en la clave primaria compuesta SP(S#,P#), ni S# ni P# podrán aceptar nulos.

2) La regla de integridad de las entidades se aplica a las relaciones base.
3) La regla de integridad de las entidades se aplica sólo a las claves primarias. En las claves alternativas se podrán permitir o no nulos.
4) Hacemos notar la idea frecuente pero errónea de que la regla de integridad de las entidades dice algo así como "los valores de la clave primaria deben ser únicos". No es así. Ese requerimiento de unicidad es parte de la definición del concepto de clave primarias en sí. Repetimos: **la regla de identidad de las entidades dice que las claves primarias de las relaciones base no deben contener nulos.**

CLAVES FORANEAS (O AJENAS)

Por ejemplo, sería absurdo que la relación (o tabla) SP incluyera un envío del proveedor S9 (por ejemplo) si no existiera un proveedor S9 en la relación (o tabla) S.

Los atributos S# y P# de la relación SP son ejemplos de lo que se conoce como claves ajenas. **Una clave ajena es un atributo (quizá compuesto) de una relación R2 cuyos valores deben concordar con los de la clave primaria de alguna relación R1 (donde R1 y R2 no necesariamente son distintos).** Por ejemplo, Si tengo la tabla ALUMNOS:

DNI_1@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503

.....

y la tabla EXAMEN:

DNI_2@	MATERIA@	FECHA @
12.123.123	Base de Datos	22/02/2005

.....

Aquí claramente se expresa que el alumno Gutierrez, José va a rendir la materia Base de Datos. **DNI_2 es la clave foranea**, la cual concuerda (contienen la misma información) con la clave primaria DNI_1.

La clave primaria de ALUMNOS es DNI_1 .

La clave primaria de EXAMEN es una clave compuesta (DNI_2, MATERIA, FECHA).

Digresión: Una clave ajena dada podría contener un valor que no aparezca de momento como valor de esa clave ajena. Por ejemplo el número de proveedor S5 aparece en la relación S pero no en la relación SP.

Terminología: Un valor de clave ajena representa una referencia a la tupla donde se encuentra el valor correspondiente de la clave primaria. El problema de garantizar que la base de datos no incluya valores no válidos de una clave ajena se conoce como el problema de la **integridad referencial.**

Podemos representar la situación de las tabla S,P y SP con el siguiente diagrama.

S <======= SP =======> P

Cada flecha significa que existe una clave ajena en la relación de la cual sale la flecha, la cual hace referencia a la clave primaria de la relación a la cual apunta la flecha. Es conveniente a veces rotular cada flecha con el nombre de la clave ajena pertinente. Por ejemplo:

S# P#
S <======= SP =======> P

LA REGLA DE INTEGRIDAD REFERENCIAL

La base de datos no debe contener valores de clave ajena sin concordancia.

Queremos decir aquí un valor no nulo de clave ajena para el cual no existe un valor concordante de la clave primaria en la relación objetivo pertinente.

Si B hace referencia a A, entonces A debe existir.

La integridad referencial exige concordancia de las claves ajenas muy específicamente, con claves primarias, no con claves alternativas.

Si vemos el ejemplo anterior:

tabla ALUMNOS:

DNI_1@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503

.....

tabla EXAMEN:

DNI_2@	MATERIA@	FECHA @
12.123.123	Base de Datos	22/02/2005
14.124.234	Matemática	22/02/2005

.....

En el ejemplo, el valor DNI_2 = 14.124.234 de la tabla EXAMEN hace que dicha tabla o relación no cumpla con la regla de Integridad Referencial, pues el alumno con el DNI_1 = 14.124.234 no se encuentra en la tabla ALUMNOS.

REGLAS PARA CLAVES FORANEAS O AJENAS.

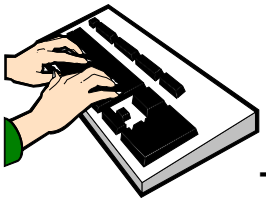
Una posibilidad, es que el sistema rechazara cualquier operación, ya que en caso de ejecutarse, produciría un estado ilegal. Una alternativa sería que el sistema aceptara la operación pero realizará ciertas operaciones de compensación con objeto de garantizar un estado legal como resultado final.

Por ejemplo, si el usuario solicita eliminar al proveedor S1 de la relación S, debería ser posible hacer que el sistema elimine también de la relación SP los envíos del proveedor S1.

En consecuencia, para cualquier base de datos, el usuario (en este contexto, con toda probabilidad el diseñador de base de datos) deberá poder especificar qué operaciones han de rechazarse y cuáles han de aceptarse, y en el caso de estas últimas, cuáles operaciones de compensación (si acaso) deberá realizar el sistema. Presentamos un breve análisis de esta posibilidad. Para cada clave ajena, es necesario responder tres preguntas:

1) ¿Puede aceptar nulos esa clave ajena?
2) ¿Qué deberá suceder si hay un intento de eliminar el objetivo de una referencia de clave ajena?; por ejemplo, un intento de eliminar un proveedor del cual existe por lo menos un envío. Existe por lo menos tres posibilidades:
   - **RESTRINGIDA** – La operación de eliminación ésta "restringida" al caso en el cual no existen tales envíos (se rechazará en caso contrario)
   - **SE PROPAGA** – La operación de eliminación "se propaga" ("en cascada") eliminando también los envíos correspondientes.
   - **ANULA** – Se asigna nulos a la clave ajena en todos los envíos correspondientes y en seguida se elimina el proveedor (por supuesto, este caso no sería aplicable si en principio la clave ajena no puede aceptar nulos)
3) ¿Qué deberá suceder si hay un intento de modificar la clave primaria del objetivo de una referencia de clave ajena, por ejemplo un intento de modificar el número de un proveedor del cual existe por lo menos un envío?. Existen tres posibilidades:
   - **RESTRINGIDA** – La operación de modificación está "restringida" al caso en el cual no existen tales envíos (se rechazará en caso contrario).
   - **SE PROPAGA** – La operación de modificación "se propaga" ("en cascada") modificando también la clave ajena en los envíos correspondientes.
   - **ANULA** – Se asignan nulos a la clave ajena en todos los envíos correspondientes y enseguida se modifica el proveedor (claro que este caso tampoco sería aplicable si en principio la clave ajena no puede aceptar nulos)

Por lo tanto, para cada clave ajena incluida en el diseño, el diseñador de la base de datos deberá especificar, no sólo el atributo o comunicación de atributos que constituye esa clave ajena y la relación objetivo a la cual hace referencia esa clave ajena, sino también las respuestas a las tres preguntas anteriores.



TRABAJO EN LABORATORIO...

- En access genere las tablas ALUMNOS y EXAMENES, haga las relaciones debidas y utilice la propiedad de eliminación y actualización en cascadas.
- Realice pruebas de verificación, insertando y eliminando registros**.**

ACTIVIDAD Nº 9

- **Responda el siguiente cuestionario:**
  - **1)** ¿Qué es una clave primaria?.
  - **2)** ¿Qué son las propiedades de Unicidad y Minimalidad?.
  - **3)** ¿Por qué ningún componente de la clave primaria, de una relación base, puede aceptar nulos?. justifique esta regla de integridad de las entidades.
  - **4)** ¿Qué es una clave ajena o foránea?.
  - **5)** ¿En qué consiste la regla de integridad referencial?
  - **6)** ¿Cuáles son las reglas para claves foráneas o ajenas?

Unidad X: Normalización

FORMAS NORMALES

El diseño de bases de datos es en realidad el diseño de esquemas. Por definición, cuando diseñamos una base de datos, nos interesan las propiedades de los datos que siempre se cumplen, no propiedades que por casualidad son ciertas en algún instante específico. Por ejemplo, la propiedad "toda parte tiene un peso" siempre es cierta; en cambio, la propiedad "todas las partes rojas están almacenadas en Londres" es sólo una coincidencia. El propósito del esquema es precisamente capturar aquellas propiedades que siempre son verdaderas.

Las relaciones en una base de datos relacional siempre están normalizadas, en el sentido de que los dominios simples subyacentes contienen sólo valores atómicos.

La teoría de la normalización, dice que el punto fundamental es que una relación dada, aún cuando pudiera estar normalizada, podría poseer todavía ciertas propiedades indeseables. Por ejemplo, en cuenta de las tablas ALUMNOS y EXAMENES :

tabla ALUMNOS:

DNI_1@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503
14.124.234	GERMAN, MARIO	MENDOZA 23

.....

tabla EXAMENES:

DNI_2@	COD_MAT@	FECHA @	MATERIA	NOTA
12.123.123	1	22/02/2005	Base de Datos	SEIS
12.123.123	2	23/02/2005	Matemática	SIETE
12.123.123	3	24/02/2005	Programación I	CUATRO
14.124.234	2	23/02/2005	Matemática	DIEZ

.....

Los campos claves están indicados por el carácter @.
COD_MAT significa código de la materia. Así la materia Matemática posee el código 2.

Podríamos tener toda la información en una sola tabla ALUMNOS-EXAMENES:

tabla ALUMNOS-EXAMENES

DNI@	NOMBRE	DIRECCION	COD_MAT@	MATERIA	FECHA@	NOTA
12.123.123	GUTIERREZ, J.	LA MADRID 503	1	Base de Datos	22/02/2005	SEIS
12.123.123	GUTIERREZ, J.	LA MADRID 503	2	Matemática	23/02/2005	SIETE
12.123.123	GUTIERREZ, J.	LA MADRID 503	3	Programación I	24/02/2005	CUATRO
14.124.234	GERMAN, M.	MENDOZA 23	2	Matemática	23/02/2005	DIEZ

Como podrá apreciar, la tabla ALUMNOS-EXAMENES, posee una serie de falencias, como ser la existencia de datos duplicados, por ejemplo, el nombre del alumno y su dirección aparecen reiteradamente, lo que involucra

un problema de almacenamiento y de actualización (si cambia el domicilio, tendrá que modificar varios registros en este caso).

La teoría de la normalización nos permite reconocer esos casos y nos muestra como podemos convertir esas relaciones a una forma más deseable.

La teoría de la normalización tiene como fundamento el concepto de formas normales. Se dice que una relación está en una determinada forma normal si satisface un cierto conjunto de restricciones. Por ejemplo, se dice que una relación está en primera forma normal (abreviada 1NF) si y solo si satisface la restricción de que sus dominios simple subyacentes contengan sólo valores atómicos (los atributos, no tienen datos compuestos).

Se han definido un gran número de formas normales. Originalmente Codd definió la primera, segunda y tercera formas normales.

El llamado procedimiento de normalización, dice que una relación en cierta forma normal digamos 2NF, se puede convertir en un conjunto de relaciones en una forma más deseable, digamos 3NF.

Podemos caracterizar ese procedimiento como la reducción sucesiva de un conjunto dado de relaciones a una forma más deseable.

La definición original de 3NF dada por Codd. resultó ser inadecuada en ciertos aspectos.

Hoy día la nueva 3NF se conoce por lo regular como “forma normal Boyce/Codd” (BCNF) para distinguirla de la forma antigua.

La idea general de normalización es que el diseñador de base de datos deberá poner su mira en relaciones en forma normal “final” (5NF). Sin embargo, esta recomendación no deberá tomarse como una ley. En ocasiones hay buenas razones para descartar los principios de normalización. No es nuestra intención sugerir que el diseño debe basarse por fuerza sólo en los principios de normalización.

DEPENDENCIA FUNCIONAL

Dada una relación R, **el atributo Y de R depende funcionalmente del atributo X de R**. (En símbolos **R.X---->R.Y** Léase**”R.X determina funcionalmente R.Y”**) **si y solo si un solo valor Y en R está asociado a cada valor X en R** (en cualquier momento dado). Los atributos X y Y pueden ser compuestos.

En la base de datos de proveedores y partes, por ejemplo, los atributos SNOMBRE, SITUACIÓN y CIUDAD de la relación S son todos funcionalmente dependientes del atributo S# de la relación S, **porque, dado un valor especifico de S.S#, existe sólo un valor correspondiente de S.SNOMBRE, de S.SITUACIÓN y de S.CIUDAD.**

En símbolos:

S.S# --------> S.SNOMBRE
S.S# --------> S.SITUACIÓN
S.S# --------> S.CIUDAD

O, en forma más concisa,

S.S# --------> S.(SNOMBRE,SITUACIÓN,CIUDAD)

Adviértase que **si el atributo X es una clave primaria de la relación R, entonces todos los atributos Y de la relación R deben por fuerza depender funcionalmente de X.**

Adviértase que la definición de dependencia funcional (DF) no requiere que X sea una clave candidata de R.

También definimos el concepto de dependencia funcional completa. **Se dice que el atributo Y de la relación R es por completo dependiente funcionalmente del atributo X de la relación R si depende funcionalmente de X y no depende funcionalmente de ningún subconjunto propio de X .**

Por ejemplo, si tenemos:

tabla ALUMNOS-EXAMENES

DNI@	NOMBRE	DIRECCION	COD_MAT@	MATERIA	FECHA@	NOTA
12.123.123	GUTIERREZ, J.	LAMADRID 50	1	Base de Datos	22/02/2005	SEIS
12.123.123	GUTIERREZ, J.	LAMADRID 50	2	Matemática	23/02/2005	SIETE
12.123.123	GUTIERREZ, J.	LAMADRID 50	3	Programación I	24/02/2005	CUATRO
14.124.234	GERMAN, M.	MENDOZA 23	2	Matemática	23/02/2005	DIEZ

En el ejemplo la clave primaria está compuesta por los atributos DNI, COD_MAT y FECHA.

El atributo o campo **NOTA** depende funcionalmente de la totalidad de la clave primaria, es decir que **NOTA** solo cobra sentido si la relacionamos con el dni de un alumnos, el código de la materia que rindió y la fecha de dicho examen. Esto es lo que se denomina una DEPENDENCIA FUNCIONAL COMPLETA, de forma tal que si algún atributo, que compone la clave principal, no existiera, entonces Nota pierde su significado (¿de cual alumno es la nota si no poseemos su DNI?, ¿A qué matería correponde la Nota? y ¿Cuándo rindió? Recuerde que un alumno puede rendir la misma materia varias veces hasta aprobar).

En cambio el atributo **Nombre_materia**, depende solamente de Código_materia nada más. Si no conocemos el Dni, por ejemplo, el atributo Nombre_materia no pierde su significado. Aquí vemos que la dependencia funcional de Nombre_materia no es completa.

Obsérvese que si Y depende funcionalmente de X, pero no por completo, X debe ser compuesto.

De aquí en adelante **supondremos que "dependencia funcional" se refiere a una dependencia funcional completa**.

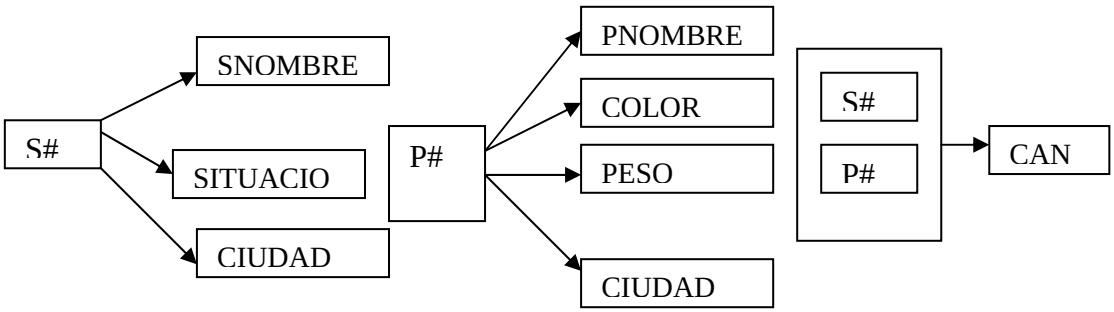


FIG 10.1 Dependencias funcionales en la relaciones S,P,SP

Cada flecha que se ve en los diagramas de la Fig. 10.1 es una flecha que sale de una clave candidata (de hecho la clave primaria) de la relación pertinente.

El procedimiento de normalización puede caracterizarse en términos muy informales, como un procedimiento para eliminar flechas que no salgan de claves candidatas.

Debe entenderse que **la dependencia funcional es un concepto semántico. Reconocer las dependencias funcionales es parte del proceso de entender qué significan los datos.** El hecho de que CIUDAD dependa funcionalmente de S#, por ejemplo significa que cada proveedor esta situado en una sola ciudad.

Adviértase que las dependencias funcionales representan interrelaciones de muchos a uno. Por ejemplo puede leerse como “en la misma ciudad pueden estar situados muchos proveedores” (pero un proveedor esta situado en una sola ciudad, por supuesto).

PRIMERA, SEGUNDA Y TERCERA FORMAS NORMALES

Un atributo no clave es cualquier atributo que no participa en la clave primaria de la relación en cuestión.

Dos o más atributos son mutuamente independientes si ninguno de ellos depende funcionalmente de cualquier combinación de los otros. Tal independencia implica que cualquiera de esos atributos puede actualizarse sin tomar en cuenta a los demás.

Por ejemplo, en la relación P (de partes) los atributos PNOMBRE, COLOR, PESO y CIUDAD son sin duda independientes entre sí (es posible cambiar, por ejemplo, el color de una parte sin tener que modificar al mismo tiempo su peso), y desde luego son dependientes por completo de P#, la clave primaria (las dependencias --funcionales-- deben ser completas, porque P# no es compuesta).

- **PRIMERA FORMA NORMAL:**

"Una relación está en primera forma normal (1FN) si y sólo si todos los dominios simples subyacentes contienen sólo valores atómicos".

Es decir que todos los atributos no contengan datos compuestos. Por ejemplo un datos compuesto sería: "Datos_personales", pues estaría compuesta por dni, nombre dirección, etc. En cuenta de esto hay que crear los atributos o campos: "DNI", "NOMBRE", "DIRECCION", ETC. De forma tal que cada atributo contenga valores simples o sea valores atómicos.

Una relación que solo está en primera forma normal, tiene una estructura indeseable por varias razones. Supongamos que la información acerca de proveedores y envíos, en vez de estar dividida en dos relaciones (S y SP), está amontonada en una sola relación como sigue:

PRIMERA (S#, SITUACION, CIUDAD, P#, CANT)

Introducimos una restricción adicional, a saber, SITUACION depende funcionalmente de CIUDAD; el significado de esta restricción es que la situación de un proveedor está determinada por la ciudad en la cual está situado (por ejemplo, todos los proveedores de Londres deben tener una situación de 20).

La clave primaria de PRIMERA es la combinación (S#, P#).

Adviértase que este diagrama DF (dependencia funcional) es "más complejo", hay flechas que salen de la clave primaria junto con ciertas flechas adicionales, y son esas flechas adicionales las que causan todos los problemas. Los atributos no clave no son todos independientes entre sí, porque SITUACION depende de CIUDAD (una flecha adicional), y no todas dependen por completo de la clave primaria, porque tanto SITUACION como CIUDAD dependen sólo de S# (otras dos flechas adicionales).

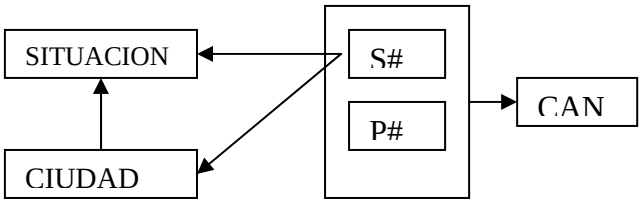


Fig. 10.2 Dependencias funcionales en la relación PRIMERA

Por ejemplo, si tenemos:

tabla ALUMNOS-EXAMENES

DNI@	NOMBRE	DIRECCION	COD_MAT@	MATERIA	FECHA@	NOTA
12.123.123	GUTIERREZ, J.	LAMADRID 50	1	Base de Datos	22/02/2005	SEIS
12.123.123	GUTIERREZ, J.	LAMADRID 50	2	Matemática	23/02/2005	SIETE
12.123.123	GUTIERREZ, J.	LAMADRID 50	3	Programación I	24/02/2005	CUATRO
14.124.234	GERMAN, M.	MENDOZA 23	2	Matemática	23/02/2005	DIEZ

La tabla exámenes esta en primera forma normal, pero como ya dijimos tenemos duplicación de información y problemas de mantenimiento entre otros.

Las redundancias en la relación PRIMERA provocan diversos casos de lo que suele llamarse “anomalías de actualización”; es decir, problemas con tres operaciones de actualización INSERT(inserte), DELETE(eliminar) y UPDATE(actualizar).

- **SEGUNDA FORMA NORMAL:**

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

Por ejemplo la tabla:

PRIMERA (S#, SITUACION, CIUDAD, P#, CANT).

Está en primera forma normal pero no en segunda forma normal, puesto que SITUACIÓN depende sólo de S# y no de la clave completa S#,P# (fig. 10.2).

La solución es dividir la relación o tabla PRIMERA en las tablas SEGUNDA y SP respectivamente:

SEGUNDA (S#, SITUACION, CIUDAD).
SP (S#, P#, CANT).

Las relaciones SEGUNDA y SP están en 2FN (las claves primarias son S# y la combinación (S#, P#), respectivamente).

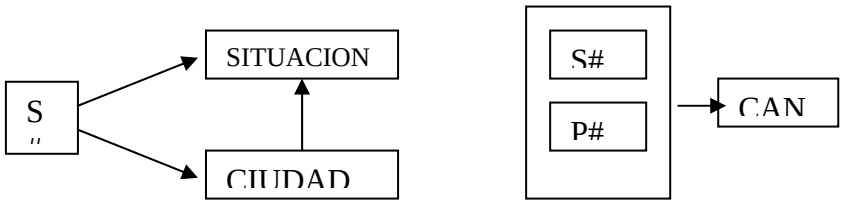


Fig. 10.3 Dependencias funcionales en las relacións SEGUNDA y SP

El proceso de reducción consiste en sustituir la relación en 1FN por proyecciones apropiadas; el conjunto de proyecciones así obtenido es equivalente a la relación original, en el sentido de que la relación original siempre se podrá recuperar efectuando la reunión (natural) de esas proyecciones, de manera que no se pierde información con la reducción. En otras palabras, el proceso es reversible. De modo tal que partiendo de la tabla PRIMERA obtenemos las tablas SEGUNDA y SP y viceversa.

De igual forma la relación SEGUNDA, todavía sufre de una falta de independencia mutua entre sus atributos no clave. En términos específicos,

la dependencia de SITUACION con respecto a S#, aunque es funcional, y de hecho completa, es transitiva (a través de CIUDAD): cada valor S# determina un valor de CIUDAD, y ese valor de CIUDAD a su vez determina el valor de SITUACION.

Las dependencias transitivas producen una vez más, anomalías de actualización.

De igual forma al sustituir la tabla ALUMNOS-EXAMENES (la misma está en 1fn, pero no en 2fn):

ALUMNOS-EXAMENES

DNI@	NOMBRE	DIRECCION	COD_MAT@	MATERIA	FECHA@	NOTA
12.123.123	GUTIERREZ, J.	LAMADRID 50	1	Base de Datos	22/02/2005	SEIS
12.123.123	GUTIERREZ, J.	LAMADRID 50	2	Matemática	23/02/2005	SIETE
12.123.123	GUTIERREZ, J.	LAMADRID 50	3	Programación I	24/02/2005	CUATRO
14.124.234	GERMAN, M.	MENDOZA 23	2	Matemática	23/02/2005	DIEZ

por las tablas:

ALUMNOS:

DNI_1@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503
14.124.234	GERMAN, MARIO	MENDOZA 23

.....

EXAMENES:

DNI_2@	COD_MAT@	FECHA @	MATERIA	NOTA
12.123.123	1	22/02/2005	Base de Datos	SEIS
12.123.123	2	23/02/2005	Matemática	SIETE
12.123.123	3	24/02/2005	Programación I	CUATRO
14.124.234	2	23/02/2005	Matemática	DIEZ

Obsevamos que:

1) La tabla ALUMNOS está en 1fn y en 2fn.

2) La tabla EXAMENES no está en 2fn puesto que el atributo MATERIA depende solamente de COD_MAT y no de DNI_2 ni de FECHA. La solución para convertir a 2fn es:

tabla ALUMNOS:

DNI@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503
14.124.234	GERMAN, MARIO	MENDOZA 23

.....

tabla EXAMENES:

DNI@	COD_MAT@	FECHA @	NOTA
12.123.123	1	22/02/2005	SEIS
12.123.123	2	23/02/2005	SIETE
12.123.123	3	24/02/2005	CUATRO
14.124.234	2	23/02/2005	DIEZ

tabla MATERIAS:

COD_MAT@	MATERIA
1	Base de Datos
2	Matemática
3	Programación I

Ahora suponga que colocamos un nuevo atributo en la tabla Materias, para indicar si pertenecen al area de ciencias exactas(1), naturales (2), informática (3), etc tendríamos:

tabla MATERIAS:

COD_MAT@	MATERIA	AREA
1	Base de Datos	3
2	Matemática	1
3	Programación I	3

Está tabla está en 1fn (**todos los dominios simples subyacentes contienen sólo valores atómicos, o sea que los campos no poseen estructuras**) y en 2fn (**todos los atributos no clave dependen por completo de la clave primaria**), como la clave no es compuesta, entonces se cumple la dependencia exigida para estar la relación en 2fn.
El inconveniente es que el atributo MATERIA depende del atributo AREA.

- **TERCERA FORMA NORMAL:**

Una relación está en tercera forma normal (3NF) si y sólo si está en 2FN y todos los atributos no clave no tengan dependencia funcional entre ellos.

Una vez más, la solución a los problemas es sustituir la relación original (SEGUNDA, en este caso) por dos proyecciones, a saber :

SC (S#, CIUDAD)
y
CS (CIUDAD, SITUACION)

Los diagramas DF para estas dos relaciones se muestran en la figura 10.4

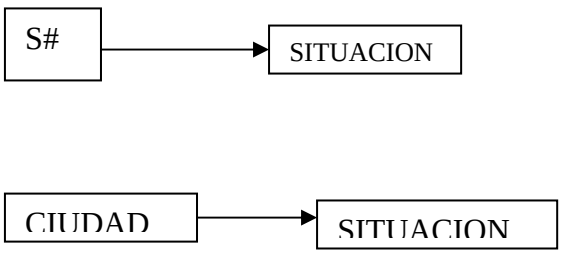


Fig. 10.4 Dependencias Funcionales en las relaciones SC y CS.

No es posible observar una relación dada en un instante determinado y decir, sin más información, si esa relación está en 3FN (por ejemplo); es necesario además conocer el significado de los datos, es decir, las dependencias, antes de emitir un juicio semejante.

En particular, el DBMS no puede garantizar el mantenimiento de una relación en 3FN, si no le indican todas las dependencias pertinentes.

De igual forma la tabla:

tabla MATERIAS:

COD_MAT@	MATERIA	AREA
1	Base de Datos	3
2	Matemática	1
3	Programación I	3

Para estar en 3fn quedaría:

tabla MATERIAS:

COD_MAT@	MATERIA
1	Base de Datos
2	Matemática
3	Programación I

tabla AREAS:

COD_MAT@	AREA
1	3
2	1
3	3

Podemos deducir que a partir de la tabla original:

tabla ALUMNOS-EXAMENES

DNI@	NOMBRE	DIRECCION	COD_MAT@	MATERIA	FECHA@	NOTA
12.123.123	GUTIERREZ, J.	LAMADRID 50	1	Base de Datos	22/02/2005	SEIS
12.123.123	GUTIERREZ, J.	LAMADRID 50	2	Matemática	23/02/2005	SIETE
12.123.123	GUTIERREZ, J.	LAMADRID 50	3	Programación I	24/02/2005	CUATRO
14.124.234	GERMAN, M.	MENDOZA 23	2	Matemática	23/02/2005	DIEZ

Que no cumplía con las fomas normales 2fn y 3fn, lo que traía aparejado diversos problemas, por lo que obtuvimos las siguientes tablas:

tabla ALUMNOS:

DNI_1@	NOMBRE	DIRECCION
12.123.123	GUTIERREZ, JOSE	LA MADRID 503
14.124.234	GERMAN, MARIO	MENDOZA 23

.....

tabla EXAMENES:

DNI_2@	COD_MAT@	FECHA @	MATERIA	NOTA
12.123.123	1	22/02/2005	Base de Datos	SEIS
12.123.123	2	23/02/2005	Matemática	SIETE
12.123.123	3	24/02/2005	Programación I	CUATRO
14.124.234	2	23/02/2005	Matemática	DIEZ

tabla MATERIAS:

COD_MAT@	MATERIA
1	Base de Datos
2	Matemática
3	Programación I

tabla AREAS:

COD_MAT@	AREA
1	3
2	1
3	3

Las cuales se hayan en 1fn, 2fn y 3fn. Si bien es mas trabajo concebir la información de esta manera, pero a la hora de querer lograr exactitud, integridad, fidelidad, etc., en la información, este método de normalidades para bases de datos relacionales es el adecuado.

BUENA Y MALAS DESCOMPOSICIONES

Las anomalías de actualización encontradas con SEGUNDA podían resolverse si la sustituíamos por su descomposición en dos nuevas proyecciones 3FN

- Descomposición **"A"**:

  SC(S#,CIUDAD) y CS(CIUDAD, SITUACION).

- Descomposición **"B"** :

  SC(S#,CIUDAD) y SS(S#, SITUACION).

La descomposición **B** tampoco provoca pérdidas, y las dos proyecciones están en 3FN. Pero la descomposición **B** es menos satisfactoria que la descomposición **A.** Por ejemplo, sigue siendo imposible (en **B**) insertar la información de que una ciudad determinada tiene una cierta situación si no hay un proveedor situado en esa ciudad.

En cambio, en el ejemplo de MATERIAS y AREAS, no ocurre lo mismo, puesto que al incluír una nueva materia (su descripción o nombre) tengo que incluir el código de la misma.

FORMA NORMAL DE BOYCE/CODD

La definición original de Codd de 3FN tenía ciertas deficiencias. En términos más precisos, no manejaba de manera satisfactoria el caso de una relación en la cual

a) Hay varias claves candidatas.
b) Esas claves candidatas son compuestas, y
c) Las claves candidatas se traslapan (es decir, tienen por lo menos un atributo en común).

Así pues, la definición original de 3FN se sustituyó después por una definición más sólida. La combinación de las condiciones (a), (b) y (c) podría presentarse muy raras veces en la práctica. En el caso de una relación en la cual no son aplicables (a), (b) y (c), BNCF se reduce a la antigua 3FN.

Para definir BNCF, es conveniente introducir primero un término nuevo.:

"Definimos un determinante como un atributo del cual depende funcionalmente (por completo) algún otro atributo."

Ahora definimos BNCF de la siguiente manera:

"Una relación está en forma normal Boyce/Cood (BCNF) si y sólo si todo determinante es una clave candidata."

CONCLUSIONES:

Los objetivos generales del proceso de normalización son los siguientes:

- Eliminar ciertos tipos de redundancias.
- Evitar ciertas anomalías de actualización.
- Producir un diseño que sea una "buena" representación del mundo real, que sea fácil de entender intuitivamente y constituya una buena base para un crecimiento futuro.
- Simplificar la imposición de ciertas restricciones de integridad.

Habrá razones válidas para no normalizar por completo.

Las ideas de la normalización son útiles en el diseño de bases de datos, pero no son una panacea.

Podríamos mencionar también que en todo caso las buenas metodologías de diseño descendente tienden a generar diseños por completo normalizados.

TRABAJO EN GRUPO:

1) Construya tablas en 1fn solamente (pero no que esté en 2fn). Compruebe si existen dificultades de actualización.

2) Construya tablas en 2fn solamente (pero no que esté en 3fn). Compruebe si existen dificultades de actualización.

3) Busque tablas a partir de un ejemplo de la realidad y normalice dichas tablas.

ACTIVIDAD Nº 10

- **Responda el siguiente cuestionario:**

¿Qué significa que una relación esté en una determinada forma normal?.

¿Cuántas formas normales definió Codd originalmente?.

Defina y ejemplifique Dependencia Funcional.

Defina Dependencia Funcional Completa.

¿Qué significa que la dependencia funcional es un concepto semántico?.

¿Cuándo los atributos son mutuamente independientes?.

Defina primera forma normal.

¿A qué se llama anomalías de actualización?.

Defina segunda forma normal.

Defina tercera forma normal.

¿Qué es un determinante?.

Defina forma normal Boyce/Cood (BCNF).

Unidad XI: Modelado Semántico

ENFOQUE GENERAL

Los sistemas de bases de datos en general –relacionales o no- sólo tienen en realidad una comprensión limitada del significado de la información contenida en la base de datos; por lo regular "entienden" ciertos valores atómicos sencillos de los datos y quizás ciertas interrelaciones de muchos a uno entre los valores, pero casi nada más (toda interpretación más allá de eso se deja al usuario).

El modelo semántico se conoce también con otros nombres, entre los términos que se escuchan con frecuencia podemos mencionar el modelado de datos, el modelado de entidades/interrelaciones (**DER = "Diagrama de Entidad Relación"**), y el modelado de entidades.

DEFINICIONES

CONCEPTO	DEFINICIÓN INFORMAL	EJEMPLOS
ENTIDAD	Un objeto distinguible **(sustantivo)**	Proveedor. Empleado. Departamento.
PROPIEDAD	Un elemento de información que describe a una identidad	Número de proveedor. Altura de persona. Código de identific. de departamento
INTERRELACIÓN	Una entidad que sirve para conectar entre sí a otras dos o más entidades **(verbo)**	Envío (proveedor-parte) Asignación (empleado-departamento)
SUBTIPO	El tipo de entidad Y es un subtipo de tipo entidad X si y solo si todo Y es por fuerza un X	Empleado es un subtipo de persona

FIG: 11.1 Algunos conceptos semánticos útiles.

En la figura 11.1 los ejemplos se eligieron con la intención de ilustrar el punto de que un objeto dado del mundo real bien podría ser considerado como una entidad por algunas personas, como una propiedad por otras, y como una interrelación por otras más. (Por cierto, esto demuestra por qué es imposible dar una definición precisa de términos como "entidad".)

EL MODELO ENTIDADES/INTERRELACIONES.

Abordamos el problema de diseñar bases de datos.

Uno de los enfoques más conocidos es el llamado enfoque de Entidades/Interrelacionales (E/R, entity / relationship), basado en el "modelo de Entidades/Interrelacionales" presentado por Chen.

En la figura 11.2 el ejemplo representa los datos de operación de una compañía manufacturera sencilla.

DEFINICIONES:

- **Entidad:**

  Chen comienza por definir las entidades como "**cosas que se pueden identificar claramente**". A continuación las clasifica como entidades regulares y entidades débiles.

  Una entidad débil es aquella cuya existencia depende de otra entidad, en el sentido de que no puede existir sino existe también esa otra entidad. Por ejemplo las transacciones no existirían si no existen los clientes. Una entidad regular es una entidad que no es débil.

- **Propiedad:**

  Las entidades (y las interrelaciones, como puede verse) tienen propiedades (también conocidas como atributos). Por ejemplo todos los empleados tienen un número de empleado, un nombre, un salario, etcétera. Cada tipo de propiedad toma sus valores de un conjunto de valores correspondientes (o sea, dominio, en términos relacionales).

  Además, las propiedades pueden ser:

  - ➔ Simples o Compuestas
  - ➔ Clave
  - ➔ Univaluadas o multivaluadas (es decir, se permiten grupos repetitivos)
  - ➔ Faltantes (o sea, "desconocidas" o "no aplicables")
  - ➔ Base o derivadas o calculada(por ejemplo, "cantidad total"")

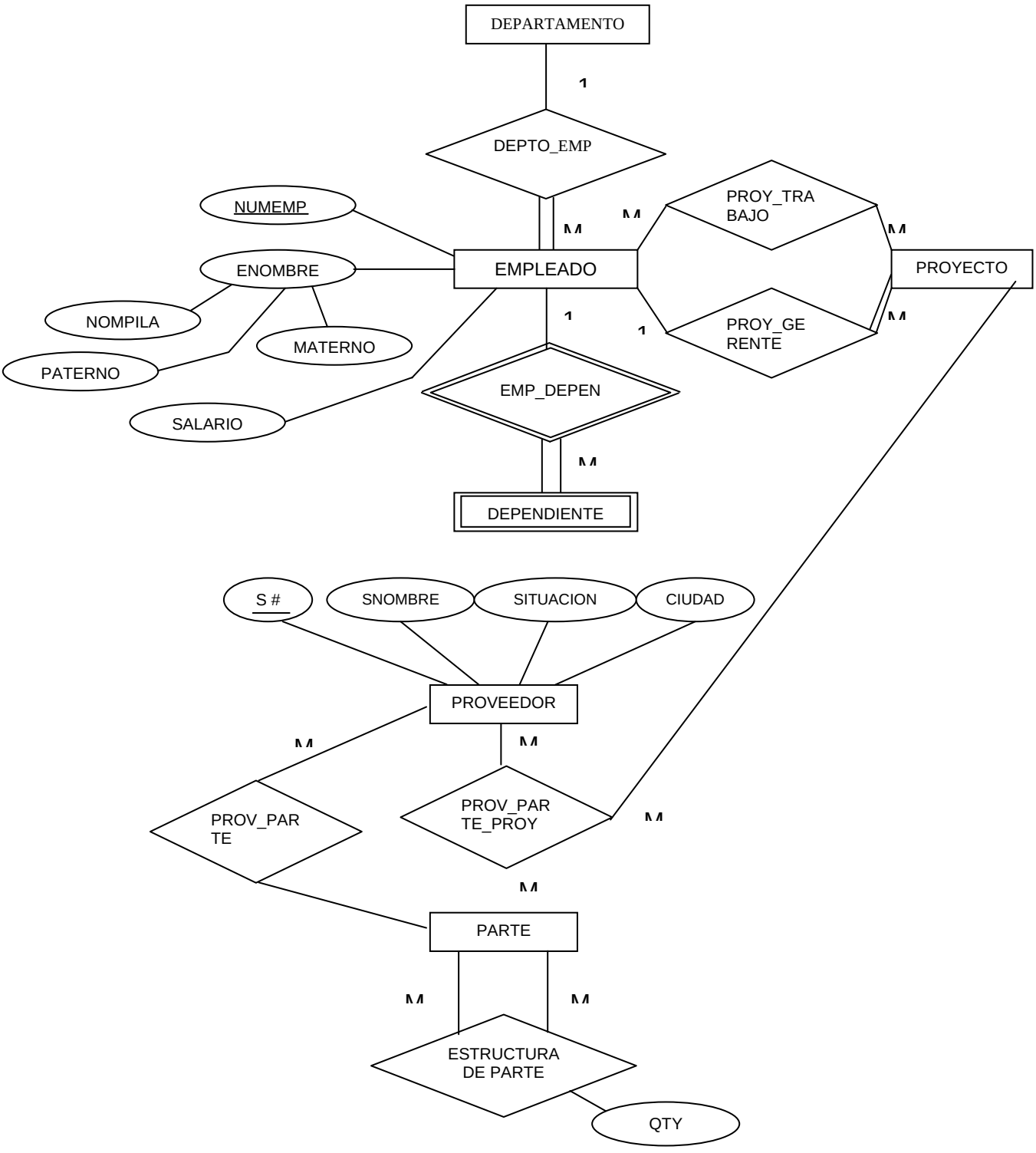


Fig. 11.2 Diagrama de entidades / interrelacionales (ejemplo)

- **Interrelación:**

  Chen define una interrelación como "**una vinculación entre entidades**". Por ejemplo, existe una interrelación (DEPTO_EMP) entre DEPARTAMENTO y EMPLEADO, la cual representa el hecho de que un cierto departamento ocupa a un conjunto dado de empleados. Es necesario distinguir entre los tipos de interrelaciones.

  Las entidades implicadas en una interrelación dada son los participantes de esa interrelación. El número de participantes en una interrelación se conoce como el grado de esa interrelación.

  Adviértase que una interrelación E/R puede ser de uno a uno, de uno a muchos o de muchos a muchos.

- **Subtipo:**

  Toda entidad pertenece por lo menos a un tipo de entidad, pero una entidad puede ser de varios tipos al mismo tiempo. Por ejemplo, si algunos empleados son programadores (y todos los programadores son empleados), podríamos decir que PROGRAMADOR es un subtipo del supertipo EMPLEADO. Toda las propiedades de los empleados se aplican de manera automática a los programadores, pero lo contratio no se cumple. De manera similar, los programadores participan de manera automática en todas las interrelaciones en las cuales participan los empleados.

  Adviértase además que algunos programadores podrían ser programadores de aplicaciones; y otros podrían ser programadores de sistemas. En la figura 11.3 puede verse un ejemplo.

  Las jerarquías de tipos se conocen con varios nombres distintos, entre ellos:

  - Jerarquías de Generalización (por razón de que, por ejemplo, un empleado es una generalización de un programador).
  - Jerarquías de especialización (por razón de que, por ejemplo, un programador es una especialización de un empleado).
  - Jerarquías ES UN (ISA) (por razón de que, por ejemplo, todo programador "es un" empleado (en ingles, "is a"))

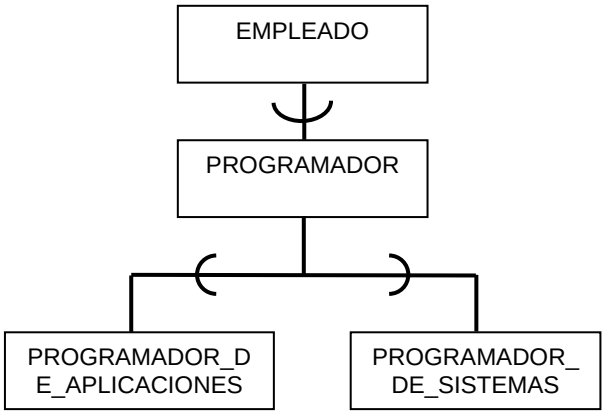


Fig.11.3 Ejemplo de Jerarquía de Tipos.

DIAGRAMAS DE ENTIDADES/INTERRELACIONALES. (DER)

Los diagramas E/R son una técnica para representar gráficamente la estructura lógica de una base de datos. ("una imagen vale mas que mil palabras").

Describiremos las reglas para construir un diagrama E/R en términos de los ejemplos dados ya en las figuras 11.2 y 11.3.

La técnica de diagramación E/R ha evolucionado un tanto con el tiempo. La versión descrita en esta sección difiere en ciertos aspectos importantes de la descrita originalmente por Chen.

- **Entidades:**
  Cada tipo de entidad se indica con un rectángulo, rotulado con el nombre del tipo de entidad en cuestión. En el caso de los tipos de entidad débiles, las aristas de los rectángulos son dobles.
  Ejemplo de Entidades: DEPARTAMENTO, EMPLEADO, PROVEEDOR, PARTE.
  Ejemplo de Entidades Débil: DEPENDIENTE.

- **Propiedades:**
  Se indican con óvalos, rotulados con el nombre de la propiedad en cuestión y conectados a la entidad (o interrelación) pertinente con una línea recta.
  El óvalo esta punteado si la propiedad es derivada y su contorno es doble si la propiedad es multivaluada.
  Si la propiedad es compuesta, sus propiedades componentes se indican con óvalos adicionales, conectados al óvalo de la propiedad compuesta en cuestión con una línea recta (otra vez).
  Las propiedades clave van subrayadas.
  Ejemplo: De EMPLEADO: NUMEMP (clave), ENOMBRE (compuesto, formado por NOMPILA, PATERNO, MMATERNO)

- **Interrelaciones:**

  Cada tipo de interrelación se indica con un rombo, rotulado con el nombre del tipo de la interrelación en cuestión.

  El rombo tiene aristas dobles si dicha interrelación es la que hay entre un tipo de entidad débil y el tipo de entidad del cual depende su existencia.

  Los participantes de cada interrelación se conectan a la interrelación pertinente con líneas rectas, y todas estas líneas se rotulan con "uno" o "muchos" para indicar si la interrelación es de uno a uno, de uno muchos o de muchos a muchos.

  Por Ejemplo:

  DEPTO_EMP (interrelación de uno a muchos entre DEPARTAMENTO y EMPLEADO)

  EMP_DEPEN (interrelación de uno a muchos entre EMPLEADO y DEPENDIENTE, un tipo de entidad débil)

  PROY_TRABAJO y PROY_GERENTE (las dos interrelaciones entre EMPLEADO y PROYECTO, la primera de muchos a muchos, la segunda de uno a muchos)

  ESTRUCTURA_DE_PARTE es un ejemplo de lo que a veces se denomina interrelación recursiva.

- **Subtipo y supertipo**

  Sea Y un subtipo de X. Entonces, trazamos una línea recta de Y a X, marcada con un gancho para representar el operador matemático "subconjunto de" (porque el conjunto de todas las Y es un subconjunto del conjunto de todas las X).

  Ejemplos (fig. 11.3)

  -PROGRAMADOR es un subtipo de EMPLEADO

  -PROGRAMADOR_DE_APLICACIONES y

  -PROGRAMADOR_DE_SISTEMAS son subtipo de PROGRAMADOR.

DISEÑO DE BASES DE DATOS CON EL MODELO DE ENTIDADES / INTERRELACIONES.

Cada tipo de entidad regular corresponde a una relación base.

Interrelaciones de muchos a muchos: Las interrelaciones de muchos a muchos mostradas en el ejemplo son las siguientes:

PROY_TRABAJO (asocia empleados y proyectos)
PROV_PARTE (asocia proveedores y partes)
PROV_PARTE_PROY (asocia proveedores, partes y proyectos)
ESTRUCTURA_DE_PARTE (asocia partes y partes)

Interrelaciones de muchos a uno: En el ejemplo hay tres Interrelaciones de muchos a uno:

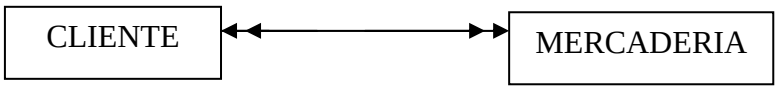
PROY_GERENTE (los proyectos designan a los gerentes)

DEPTO_EMP (los empleados designan a los departamentos)
EMP_DEPEN (los dependientes designan a los empleados)

interrelaciones uno a uno: Se manejan exactamente del mismo modo que las interrelaciones de muchos a uno. (Las cuales en cualquier caso no son muy frecuentes en la práctica)

Luego de sucesivos refinamientos, lo ideal es que el DER definitivos se vean relaciones de uno a muchos (y quizás alguna de uno a uno). Por lo que las relaciones de mucho a mucho se deben modificar y convertirse en dos relaciones de uno a mucho.

Por ejemplo si tengo dos entidades: CLIENTE y MERCADERIA y la relación COMPRA (CLIENTE compra MERCADERIA), la relación es de mucho a mucho, pues un cliente puede comprar muchas mercaderías y un tipo de mercadería (por ejemplo arroz) puede ser adquerida por muchos clientes. Entonces, se debe crear la entidad COMPRA y la relación entre CLIENTE y COMPRA es de uno a mucho y la relación entre COMPRA y MERCADERÍA, también es de uno a mucho.



En el ejemplo, hemos obviado el rombo con la etiqueta COMPRA, por cuestiones de rapidez, pero es conveniente ponerlo. Hemos incluido una doble fecha para indicar “mucho” en vez de M, o sea la relación es de mucho a mucho. El refinamiento de este ejemplo quedaría de la siguiente forma:

La relación mucho a mucho se redujo a dos relaciones uno a mucho.

La tabla COMPRA posee como clave principal por lo menos las claves principales de CLIENTE y MERCADERIA, es decir si la clave de CLIENTE es DNI y la clave de MERCADERIA es COD_MERC, entonces la clave principal de COMPRA será por lo menos DNI,COD_MERC, pudieron agregarse otro atributo, si es que la combinación DNI,COD_MERC no cumple con lo requisitos de unicidad de la clave principal.

Si la Relación tiene atributos en sí, entonces esta será una tabla. Supongamos el caso ALUMNO rinde MATERIA. Donde rinde es la Relación existente entre ALUMNO y MATERIA, pero el atributo o propiedad “Fecha de Examen”

no puede pertenecer a ALUMNO ni a MATERIA, sino que pertenece a la Relación RINDE, por lo que RINDE se conviente en una tabla en sí.

METODOLOGÍA DE TRABAJO

Una forma metodológica de encarar la construcción de un DER es de la siguiente:

Dado una Narrativa, es decir una descripción del sistema desde el punto de vista de los datos o del flujo de información, se identifican los SUSTANTIVOS, los VERBOS y las PROPIEDADES.

Los SUSTANTIVOS podrán ser TABLAS, si tienen cardinalidad sobre todo.

Los VERBOS podrán ser relaciones entre las TABLAS.

De este modo genero un DER inicial, al cual pongo el grado de cada relación (uno a mucho, etc.)

Luego vamos refinando, tratando de completar las tablas y de lograr relaciones del tipo uno a mucho (o uno a uno a lo sumo).

Simultáneamente conviene ir NORMALIZANDO las tablas, sobre todo en la 2fn, como mínimo.

Siempre se debe ir analizando la narrativa y colocando ejemplos concretos para analizar si las tablas diseñadas, contemplan adecuadamente todas las necesidades de información.

EJEMPLO:

Encaremos un pequeño ejemplo, en el realizaremos el DER de un Consultorio de Médicos. En el mismo vamos a obviar varios detalles, de modo que el DER no se complique en demasía y se pierda el hilo del ejemplo. En la realidad los DER se suelen complicar en demasía, y las personas que no manejan adecuadamente los conceptos sobre base de datos relacionales, incurren en errores constantes.

NARRATIVA:

Sea el Consultorio Médico “Buena Vida”, en el mismo atienden cinco (5) Profesionales y dos Secretarias.

Los Profesionales, son médicos con distintas especialidades.

Los Pacientes solicitan turnos para ser atendidos por los Médicos. Las secretarias otorgan los turnos correspondientes, volcando dicha información en la planilla de turnos.

Cuando un Paciente le toca ser atendido según el turno previamente cedido, el mismo abona a la Secretaría el valor de la consulta, dependiendo lo que cobre cada médico. (En este consultorio médico no se aceptan Obras

Sociales).

Al momento de ser atendido un Paciente, la secretaria entrega al Médico la ficha de cada Paciente, en donde figura la historia médica del Paciente con el Médico que lo está atendiendo en ese momento. En la ficha figura la fecha de cada consulta, el diagnóstico y el medicamento recetado.

Luego de la atención el médico llena la ficha y la entrega a la secretaria. Luego el Paciente vuelve a solicitar otro turno si es necesario.

PLANILLA DE TURNOS Nombre del Médico: Dr. GUTIERREZ Fecha: 8/2/2005	
Turno Hs.	**Paciente**
9.0	Juan Perez
10.0	Luis Medina
....	.....

FICHA MEDICA
Nombre del Médico: Dr. GUTIERREZ
Nombre del Paciente: Juan Perez
Documento: 15.325.897
Fecha de nacimiento: 26/11/1960
Dirección: Lavalle 342
Teléfono: 4245368
Fecha de Atención : 3/07/2004
Diagnóstico: Bronquitis
Medicación: Antibiótico ....., Expectorante:....., Antifebril:....
Fecha de Atención:....
Diagnóstico:.....
Medicación:.....

Primeramente listamos los sustantivos insertos en la narrativa:

- Consultorio Médico.
- Profesionales.
- Secretarias.
- Médicos.0
- Especialidades.
- Pacientes.
- Turnos.
- Planilla de turnos.

- Consulta.
- Obras Sociales.
- Ficha
- Historia médica.
- Fecha.
- Diagnóstico.
- Medicamento.

Luego analizamos si los sustantivos, pueden ser ENTIDADES. Vemos si es necesario mantener información de la supuesta Entidad. Análisis Semántico:

- **Consultorio Médico**: Es un solo consultorio, por lo que no tiene cardinalidad mayor que uno. No es necesario generar una Entidad que sólo tendrá un solo dato. (**NO ES ENTIDAD**)
- **Profesionales**: Es lo mismo que Médicos. (**NO ES ENTIDAD**)
- **Secretarias**. Son dos, pero pueden incrementarse o cambiar, tienen cardinalidad y es necesario registrarlas, para saber quién otorgó un turno. (**ES ENTIDAD**)
- **Médicos**: Tienen cardinalidad y es necesario registrar información sobre los mismos. (**ES ENTIDAD**)
- **Especialidades**: No es necesario registrar las especialidades de los médicos, según lo indica la narrativa. (**NO ES ENTIDAD**)
- **Pacientes**: (**ES ENTIDAD**).
- **Turnos**:Es necesario registrar los turnos. (**ES ENTIDAD**).
- **Planilla de turnos**: Idem a turnos. (**NO ES ENTIDAD**):
- **Consulta**: Se debe registrar lo que cobra cada Médico. (**ES ENTIDAD**).
- **Obras Sociales**: En el contexto de la narrativa, no se registra dicha información. (**NO ES ENTIDAD**).
- **Ficha**: De cada paciente. (**ES ENTIDAD**).
- **Historia médica**: Contenida en la ficha, se la toma como sinónimo de ficha en este contexto. (**NO ES ENTIDAD**).
- **Fecha**: Es un atributo o propiedad de Ficha, (**NO ES ENTIDAD**).
- **Diagnóstico**: Es un atributo o propiedad de Ficha, (**NO ES ENTIDAD**).
- **Medicamento**: Es un atributo o propiedad de Ficha, (**NO ES ENTIDAD**).

En consecuencia tenemos las siguientes entidades:

MEDICOS			
@Matricula	**D.N.I_M**	**Nombre**	**Especialidad**
15000	12.135.235	GUTIERREZ, Juan	Clínico
12000	10.235.689	LOPEZ, Carlos	Cardiólogo

PACIENTES				
@D.N.I_P	**Nombre**	**Dirección**	**Fecha_Nacim**	**Teléfono**
15.325.897	PEREZ, Juan	26/11/1960	26/11/1960	4245368
17.265.358	MEDINA, Luis	Zuviria 1234	21/05/1966	4231568

SECRETARIAS		
@D.N.I_S	**Nombre**	**Teléfono**
14.231.568	GAMBOA, Liliana	4235689
12.254.365	LENCINA, Maria	4245864

TURNOS				
@Matricula	**@Fecha_aten**	**@Hora**	**D.N.I_P**	**D.N.I_S**
15000	08/02/2005	09	15.325.897	14.231.568
15000	08/02/2005	10	17.265.358	14.231.568
12000	08/02/2005	10	19.235.357	17.265.358

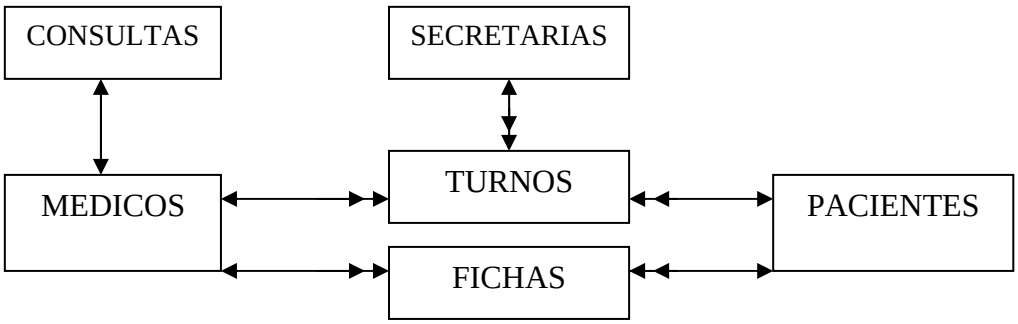
CONSULTAS	
@Matricula	**Valor_consulta**
15000	$20.
12000	$30.

FICHAS						
Matricula	**Nombre_Pac**	**Fecha_nac**	**Dirección**	**Fecha_aten**	**Diagnóstico**	**Medicamentos**
15000	PEREZ, Juan	26/11/1960	26/11/1960	08/02/2005	Bronquitis	Antibiótico:... Expectorante.. Antifebril.....
15000	PEREZ, Juan	26/11/1960	26/11/1960	08/08/2005	Control- Ok	.....

Fijese que las Entidades poseen el caracter "@" delante de los atributos que constituyen la clave primaria de cada entidad.

En la Entidad TURNOS en vez de colocar Nombre del Médico o Nombre del Paciente, se colocó directamente Matricula y DNI, para hacer alusión a la clave primaria de MEDICOS y PACIENTES respectivamente.

A continuación elaboramos un primer DER:



El DER se interpreta de la siguiente forma:

Un Médico (una flecha) atiende a muchos TURNOS (dos flechas). En realidad se podría interpretar, como que un MEDICO atiende a muchos PACIENTES

y un PACIENTE se puede hacer atender por muchos MEDICOS. En este caso la relación es de muchos a muchos, y como vimos se tiene que crear una nueva tabla, en este caso TURNOS para tener dos relaciones de uno a muchos como se ve entre MEDICOS -TURNOS y TURNOS – PACIENTES.

El mismo análisis es para la relación MEDIOCS – FICHAS – PACIENTES.

De igual forma las secretarias otorgan turnos. Una SECRETARIA otorga muchos turnos, pero un TURNO es otorgado por solo una SECRETARIA.

Por último vemos que un MEDICO tiene un solo valor de CONSULTA y una CONSULTA corresponde a solo un MEDICO. La relación es uno a uno. En este caso conviene analizar si los atributos de CONSULTA, no son en realidad atributos de MEDICOS, en este caso si lo es, pues basta colocar el atributo Valor_consulta de la entidad CONSULTA en la entidad MEDICOS, de forma tal que la entidad CONSULTAS ya no existiría.

Si analizamos las Entidades, observamos que todas están en 1 y 2 Forma Normal, excepto la tabla FICHAS. En primera medida podemos notar que el atributo Medicamentos no es atómico, pues está compuesto por el nombre del medicamento y la dosis que se aplica en cada caso. Aparte del hecho que se pueden requerir varios medicamentos para un solo paciente. Por lo expuesto creamos una nueva Entidad llamada MEDICAMENTOS:

MEDICAMENTOS		
Código_Medic@	**Nombre_Medic**	**Presentación**
000		
001	Aspirineta	Tabletas
002	Mildungen	Jarabe
003	Mildungen	Cápsulas
010	Antibiótico Forte	Cápsulas x 20
011	Exportorante Ultra	Cápsulas x 20
012	Antefebril Duo	Cápsulas x 80

Además observe que los datos referidos al nombre, dirección y fecha de nacimiento de los pacientes, se repiten innecesariamente en la entidad FICHAS, esos datos ya estan en la entidad PACIENTES. La solución es reemplazar dichos atributos por DNI_P que es la clave primaria de PACIENTES.

La Entidad Ficha quedaría de la siguiente forma:

FICHAS					
Matricula@	**D.N.I_P@**	**Fecha_aten@**	**Diagnóstico**	**Código_Medic@**	**Dosis**
15000	15.325.897	08/02/2005	Bronquitis	010 011 012	1 Caps... ....
15000	15.325.897	08/08/2005	Control- Ok	000	

Aún no está en 1FN Código_Medic no es atómico. Debemos crear una nueva

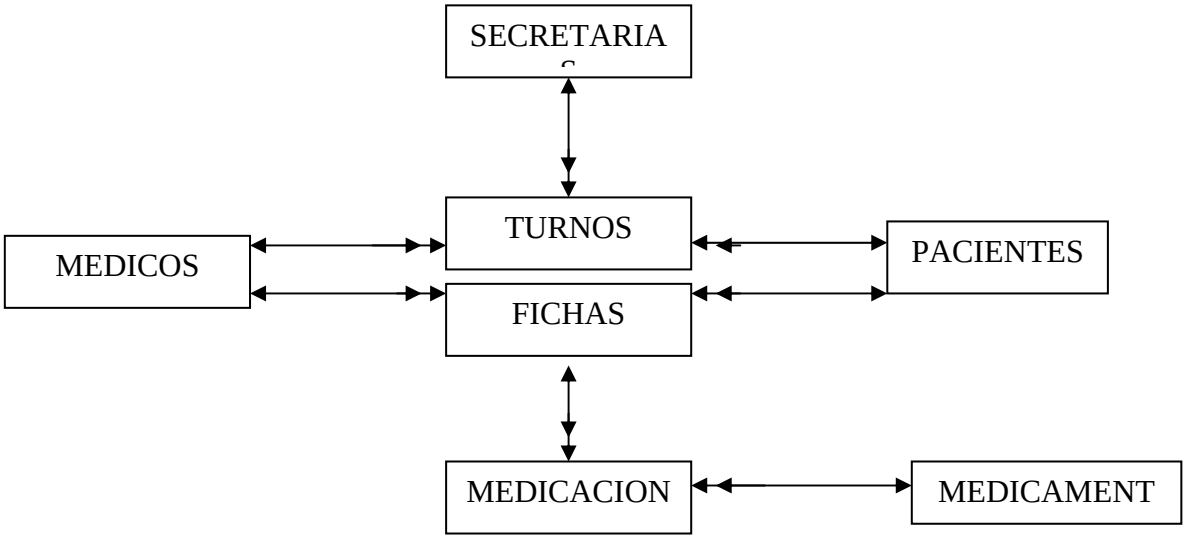
tabla:

MEDICACION				
Matricula@	**D.N.I_P@**	**Fecha_aten@**	**Código_Medic@**	**Dosis**
15000	15.325.897	08/02/2005	010	1 Caps c/12 Hs
15000	15.325.897	08/02/2005	011	1 Caps c/8 Hs
15000	15.325.897	08/02/2005	012	1 Caps c/6 Hs
15000	15.325.897	08/08/2005	000	

FICHAS quedaría de la siguiente forma:

FICHAS			
Matricula@	**D.N.I_P@**	**Fecha_aten@**	**Diagnóstico**
15000	15.325.897	08/02/2005	Bronquitis
15000	15.325.897	08/08/2005	Control- Ok

A continuación elaboramos un segundo DER:



Parecería que este segundo DER, es suficiente (faltaría ver si esta en 3FN y si se desea generar un nuevo DER).

Analice si las siguientes tablas están en 3fn.

Las tablas o ENTIDADES serían las siguientes:

MEDICOS				
@Matricula	**D.N.I_M**	**Nombre**	**Especialidad**	**Valor_consulta**
15000	12.135.235	GUTIERREZ, Juan	Clínico	$20.
12000	10.235.689	LOPEZ, Carlos	Cardiólogo	$30.

PACIENTES				
@D.N.I_P	**Nombre**	**Dirección**	**Fecha_Nacim**	**Teléfono**
15325897	PEREZ, Juan	Lavalle 44	26/11/1960	4245368
17265358	MEDINA, Luis	Zuviria 1234	21/05/1966	4231568
10999888	LORENZO, Ruben	Catamarca 30	20/08/1954	4245857

SECRETARIAS		
@D.N.I_S	**Nombre**	**Teléfono**
14.231.568	GAMBOA, Liliana	4235689
12.254.365	LENCINA, Maria	4245864

TURNOS				
@Matricula	**@Fecha_aten**	**@Hora**	**D.N.I_P**	**D.N.I_S**
15000	08/02/2005	09	15325897	14.231.568
15000	08/02/2005	10	10999888	14.231.568
12000	08/02/2005	10	17265358	17.265.358

FICHAS			
@Matricula	**@D.N.I_P**	**@Fecha_aten**	**Diagnóstico**
15000	15.325.897	08/02/2005	Bronquitis
15000	15.325.897	08/08/2005	Control- Ok

MEDICACION				
@Matricula	**@D.N.I_P**	**@Fecha_aten**	**@Código_Medic**	**Dosis**
15000	15.325.897	08/02/2005	010	1 caps c/12Hs.
15000	15.325.897	08/02/2005	011	1 caps c/8Hs.
15000	15.325.897	08/02/2005	012	1 caps c/6Hs.

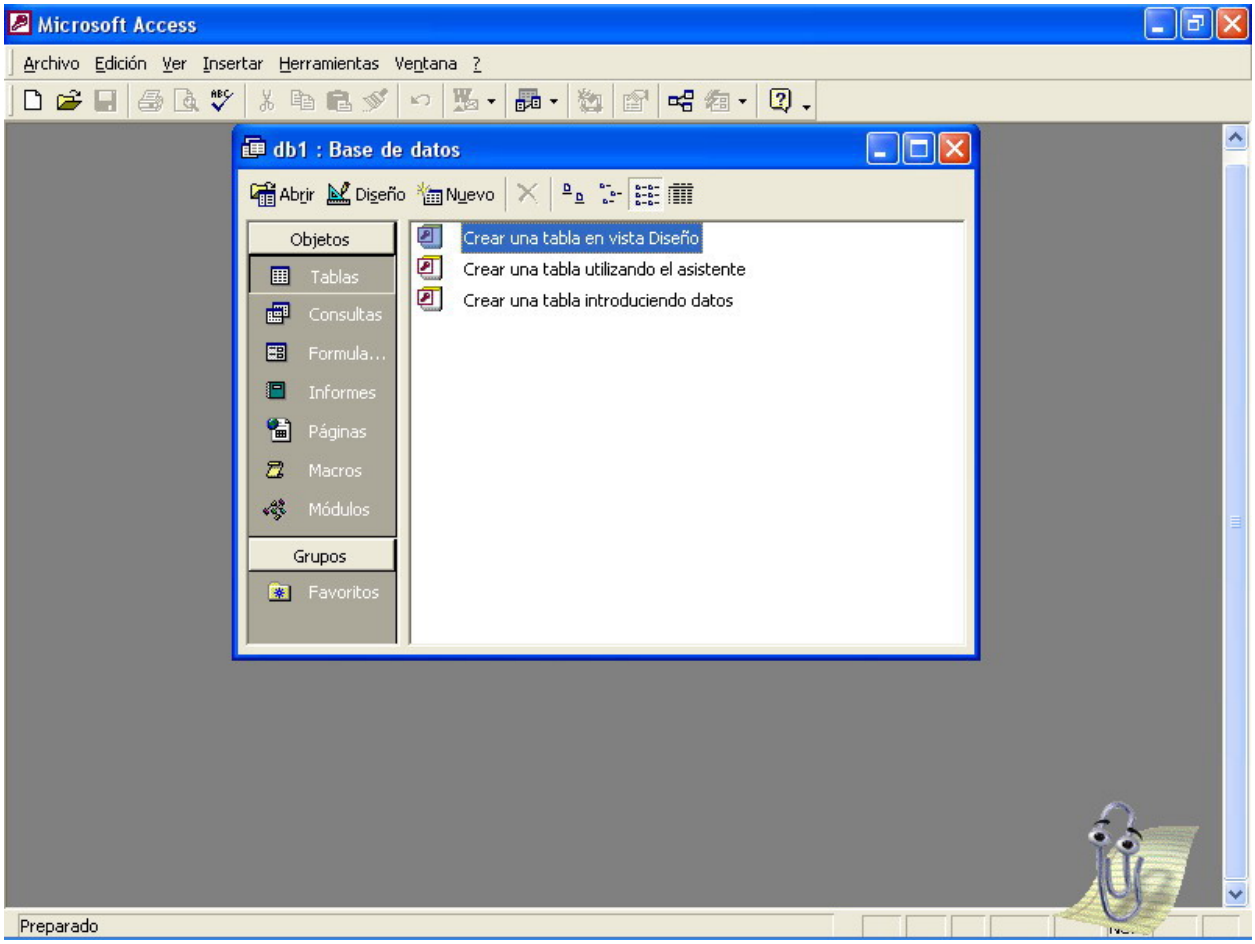
MEDICAMENTOS		
@Código_Medic	**Nombre_Medic**	**Presentación**
000		
001	Aspirineta	Tabletas
002	Mildungen	Jarabe
003	Mildungen	Cápsulas
010	Antibiótico Forte	Cápsulas x 20
011	Exportorante Ultra	Cápsulas x 20
012	Antefebril Duo	Cápsulas x 80

Recuerde que en una relación de uno a muchos, en la entidad que tiene la relación muchos, tiene que existir la clave principal de la entidad que tiene la relación uno. Por ejemplo entre SECRETARIAS y TURNO. La entidad TURNO (muchos) posee como atributo D.N.I_S que es la **clave principal** de la entidad SECRETARIAS.

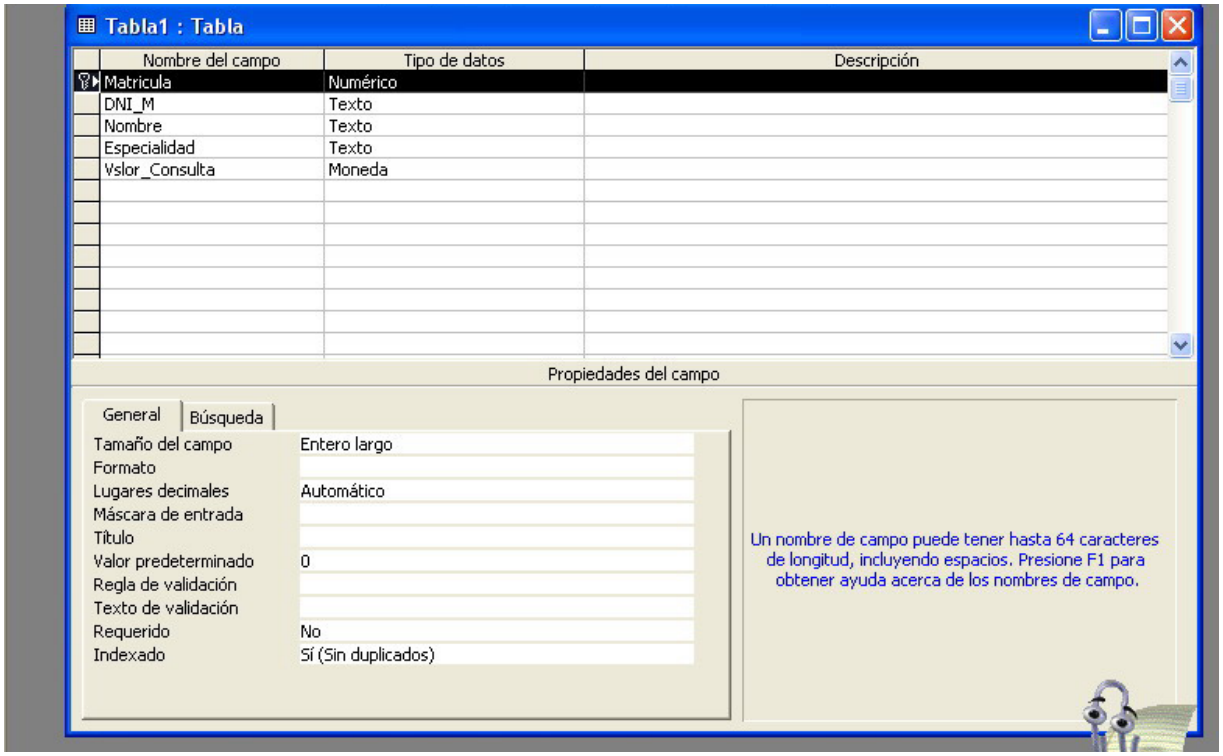
Finalmente para comprobar las Relaciones generadas, se pueden insertar datos de la realidad, para comprobar que las entidades y sus relaciones son correctas.

El siguiente pasó consistirá en utilizar un DBMS (Sistema Administrador de Bases de Datos). En este caso usaremos ACCESS de Microsoft, el mismo es un potente motor cuyas herramientas visuales son excelentes. Cabe acotar que el DBMS Access está pensado para Base de Datos pequeñas y medianas, debiendo usarse motores más potentes para Base de Datos de gran envergadura (SQL-SERVER, POSTGRESS, INFORMIX, MYSQL, ETC.).

A continuación se muestran las pantallas principales que se utilizan para generar el DER del Consultorio Médico en ACCESS.

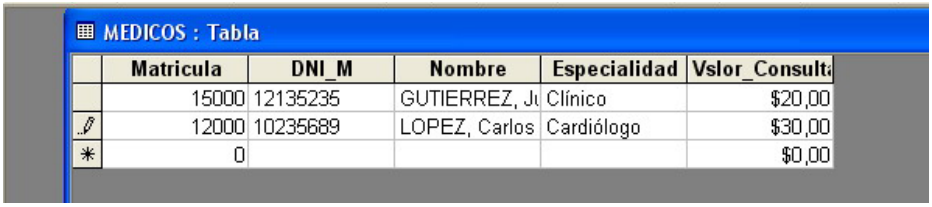


En access elegimos crear una base de datos en blanco, la llamamos CONSULTORIO (en este caso se llama db1). Luego elegimos Crear una tabla en vista diseño y comenzamos el diseño.



Empezamos a diseñar la tabla MEDICOS, colocamos el nombre de los campos (atributos) y el tipo de datos. Luego elegimos la clave principal, en este caso Matricula (haga clik en la llave luego de seleccionar el campo Matricula).

Guardamos la tabla y pasamos a modo Vista Hoja de Datos, para agregar nuevas tuplas o registros.



Luego cerramos la ventana y procedemos a crear las demás tablas.

Luego Hacemos click en el botón relaciones, y empezamos a dibujar el DER, nos pedirá que elijamos las tablas que conformarán dicho DER, en este caso son todas. Luego la clave principal de una tabla (por ejemplo DNI_S de la tabla SECRETARIA) y la arrastramos hasta la tabla en que queremos relacionarla (en este caso hasta la tabla TURNOS campo DNI_S).

Aquí nos indica que se está formando una relación entre la tabla SECRETARIAS (campo DNI_S) y la tabla TURNOS (campo DNI_S).

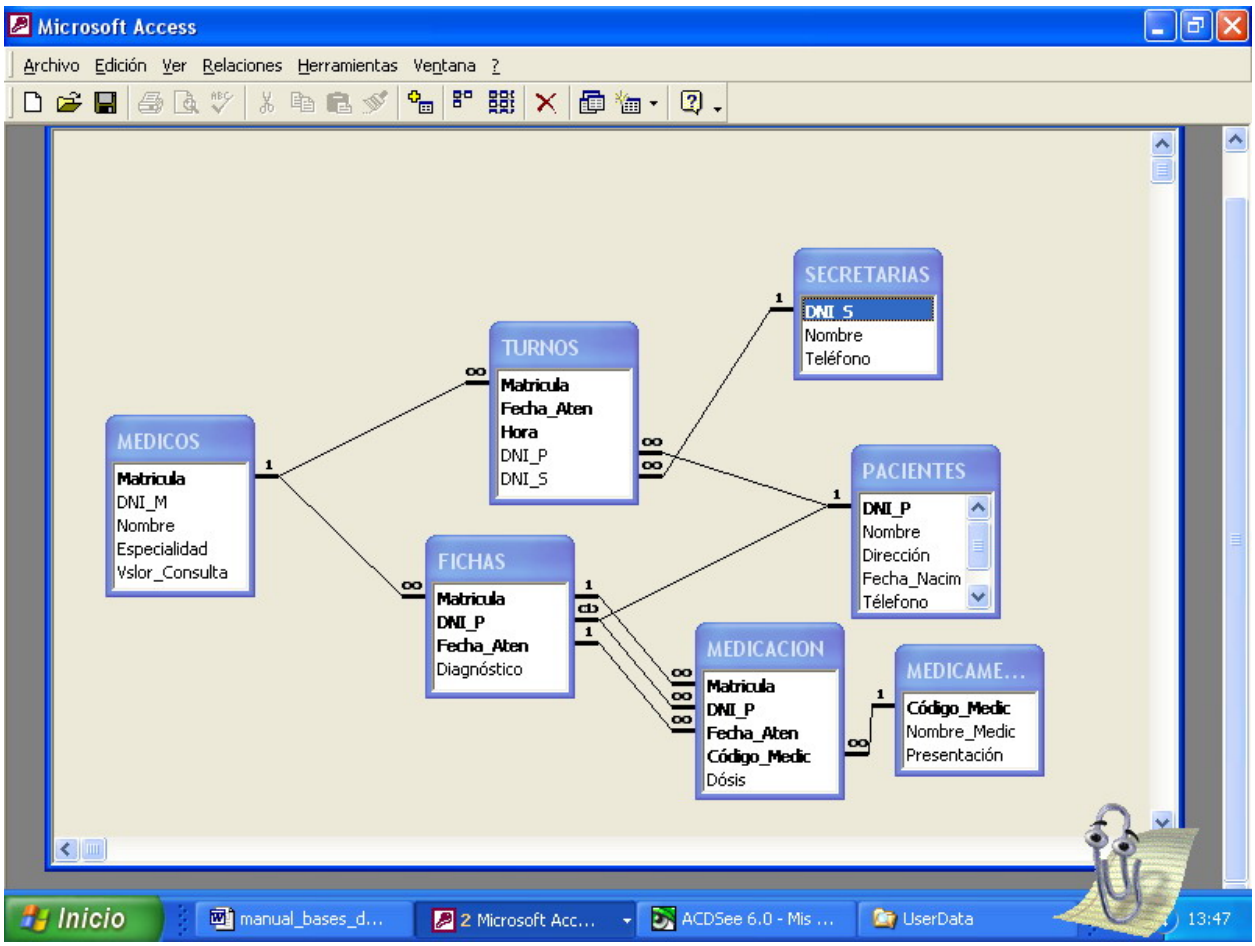
Nos dice que el tipo de relación es de **Uno A Varios.**

También les indicamos que se exija la integridad referencial, es decir si la relación es de uno a muchos, entonces el valor colocado en DNI_S de la tabla TURNOS tiene que existir previamente en DNI_S de la tabla SECRETARIAS. Puede existir en la tabla SECRETARIAS y no en la tabla TURNOS pero no al contrario.

Al tildar **Actualizar en cascada los campos relacionados**, le decimos que si modificamos un campo del lado de uno (por ejemplo cambiamos el dni del campo DNI_S de la tabla SECRETARIAS) entonces se modifica utomáticamente del lado de mucho o varios (es decir todos los valores del campo DNI_S de la tabla TURNOS,que concuerde con el campo modificado en SECRETARIAS, se modificarán).

Fíjese que no se tildó **Eliminar en cascada los campos relacionados**, eso se debe a que si estaría tildado, entonces al borrar un campo del lado de uno se borran todos (automáticamente) del lado de muchos, lo cual es muy peligroso.

Finalmente quedará de la siguiente forma:



El motor de base de datos ahora controlará automáticamente todo lo que se le indicó. Si se intenta ingresar en el lado de muchos un valor inexistente del lado de uno, entonces no me permititá realizar la acción y me mostrará un cartel indicatorio del caso.

En este caso intentamos agregar un registro en la tabla TURNOS insertando un DNI_S inexistente en la tabla SECRETARIAS.

Realice inserción, eliminación y modificación de los registros de las tablas a fin de comprobar el funcionamiento del DBMS.

Una vez finalizado el DER podemos utilizar la base de datos a través de aplicaciones. Es decir usamos un lenguaje de programación (Visual Basic, Delphi, Java, etc.) y el sql embebido, es decir el que viene incluido en el lenguaje.

Averigüe qué es el odbc para windows, observe el panel de control de windows.

GENERACIÓN DE INFORMES

Para generar un informe en word, se procede a abrir un documento en blanco, luego escribimos el texto deseado, si queremos insertar una pantalla (por ejemplo de access), con las teclas alt –tab me posesiono en la pantalla y la capturo con las teclas alt –Impr_ pant (Para capturar la ventana activa) ó con Impr_pant (Para capturar la pantalla entera), La imagen se guarda automáticamente en el portapapeles.

Si desea, en el word bastará con elegir Edición → Pegar (ctrl –v), y se insertará en el documento la imagen previamente copiada al portapapeles. El problema es que se pega en formato bmp, lo cual hace que el tamaño del documento sea muy grandes.

Para solucionar el problema, lo mejor es usar algún programa para convertir los formatos, por ejemplo puede usar el ACDSee (es un shareware) para convertir la imagen a formato jpeg. Luego debe guardar la imagen en mis_imágenes.

Para insertar la imagen en el documento word, debe elegir Insertar → Imagen → Desde_Archivos. Aparece una ventana, buscar la imagen convertida e instarla.

Investigue:

a. ¿Cuáles son los programas más conocidos que convierten los formatos?
b. ¿Qué es un programa shareware?
c. ¿Qué tipo de formatos gráficos existen, cual es el más conveniente para internet?

TRABAJO EN GRUPO :

En base a las siguientes frases, redacte una pequeña narrativa y

realice el correspondiente DER:

- Alumno rinde Materia.
- Cliente alquila Video.

ACTIVIDAD Nº 11

- **Responda el siguiente cuestionario:**

1) ¿Cómo se conoce también el modelo semántico?.

2) Defina: Entidad, Propiedad, Interrelación, Subtipo,.

3) ¿Qué son los DER?.

4) ¿Cómo se indican las entidades, propiedades, interrelaciones, subtipo y supertipo en el DER propuesto?.

5) Si posee dos entidades cuya interrelación es de mucho a mucho, ¿Cómo hago para que las relaciones sean de uno a mucho?.

Unidad XII: Recuperación, Seguridad e Integridad

INTRODUCCION

Los problemas de recuperación y concurrencia en un sistema de base de datos están muy vinculados a la noción de *procesamiento de transacciones* que involucran a las operaciones COMMIT (comprometer) y ROLLBACK (retroceder) de SQL.

Nota: suponemos en la mayor parte de esta unidad que estamos en un ambiente "grande" o de macrocomputador.

RECUPERACION DE TRANSACCIONES

Una transacción es una unidad lógica de trabajo. Ej. Supongamos, para efecto de la ilustración, que la tabla P, la de partes, contiene un campo adicional CANTTOTAL que presenta la cantidad total enviada de la parte en cuestión; en otras palabras, el valor de CANTTOTAL para una parte dada es igual a la suma de todos los valores SP.CANT de todos los registros SP correspondientes a esa parte. Consideremos ahora la siguiente secuencia de operaciones, cuya intención es añadir un nuevo envío (S5,P1,1000) a la base de datos.

```
EXEC SQL WHENEVER SQLERROR GO TO ANULAR;
EXEC SQL INSERT
        INTO SP   ( S#, P#, CANT )
        VALUES  ( 'S5', 'P1', 1000 ) ;
EXEC SQL UPDATE P
        SET     CANTTOTAL = CANTTOTAL + 1000
        WHERE         P# = 'P1' ;
EXEC SQL COMMIT;
        GO TO TERMINAR;
ANULAR :
        EXEC SQL ROLLBACK;
TERMINAR : RETURN;
```

La proposición INSERT (insertar) agrega el nuevo envío a la tabla SP, la proposición UPDATE (actualizar) pone al día en forma apropiada el campo CANTTOTAL de la parte P1; viola en forma temporal el requerimiento según el cual el valor de CANTTOTAL para la parte P1 debe ser igual a la suma de todos los valores SP.CANT correspondiente a esa parte. Así pues, **una unidad lógica de trabajo (o sea, una transacción)** no es por fuerza una sola operación en la base de datos; Mas bien, **es en general una *secuencia* de varias de esas operaciones mediante la cual un estado consistente de la base de datos se transforma en otro estado consistente,** sin conservar por fuerza la consistencia en todos los puntos intermedios.

Siempre existe la posibilidad de una falla. Por ejemplo, el sistema podría caerse justo después de la primera modificación, si el sistema maneja *el procesamiento de transacciones,* en términos específicos, garantizará que si la transacción ejecuta algunas modificaciones y después se presenta una falla (por cualquier razón) antes de que llegue el término normal de la transacción, *se anularán esas modificaciones.* Así, o bien la transacción se lleva a cabo en su totalidad, o se cancela su totalidad.

El componente del sistema encargado de lograr esta atomicidad (o apariencia de automaticidad) se conoce como ***manejador de transacciones*** y las operaciones **COMMIT** (comprometer) y **ROLLBACK** (retroceder) son la clave de su funcionamiento:

La operación COMMIT señala el termino exitoso de la transacción: le dice al manejador de transacciones que ha finalizado con éxito una unidad lógica de trabajo, y que se pueden ”comprometer”, o hacer permanentes, todas las modificaciones efectuadas por esa unidad de trabajo.

La operación ROLLBACK, en cambio, señala el término no exitoso de la transacción: dice al manejador de transacciones que algo salió mal, que la base de datos podría estar en un estado inconsistente, y que todas las modificaciones, efectuadas hasta el momento por la unidad lógica de trabajo, deben ”retroceder “ o anularse.

El sistema emitirá automáticamente una instrucción COMMIT cuando termine en forma normal cualquier programa, y emitirá también automáticamente una instrucción ROLLBACK cuando cualquier programa no termine en forma normal. En el ejemplo por tanto, podríamos haber omitido la instrucción COMMIT explícita, pero no la instrucción ROLLBACK explícita.

El sistema mantiene una *bitácora o diario* en cinta o (mas comúnmente) en disco, en los cuales se registran los detalles de todas las operaciones de actualización, en particular , los valores inicial y final del objeto modificado, esto permite hacer un ROLLBACK.

PUNTOS DE SINCRONIZACIÓN

La ejecución de una operación COMMIT (comprometer) o ROLLBACK (retroceder) establece lo que se conoce como un *punto de sincronización.* **Representa el límite entre dos transacciones consecutivas**. Las únicas operaciones que establecen un punto de sincronización son COMMIT, ROLLBACK y el inicio de un programa. Cuando se establece un punto de sincronización:

- se comprometen (COMMIT) o anula (ROLLBACK) todas las modificaciones realizadas por el programa desde el punto de sincronización anterior;
- se cierran todos los cursores abiertos y se pierde todo posicionamiento en la base de datos (al menos, esto sucede en la mayor parte de los sistemas, aunque no en todos).
- Se liberan todos los registros bloqueados.

Las transacciones no solo son la unidad de trabajo sino también la unidad de *recuperación.* Es muy posible, por ejemplo, una caída del sistema después de haberse realizado la instrucción COMMIT. El sistema instalará de todas

maneras esas modificaciones en la base de datos. Para ello, la bitácora se deberá haber grabado físicamente antes de poderse completar el procesamiento de una instrucción COMMIT, Esta importante regla se conoce como *protocolo de bitácora e escritura adelantada.)* Así el procedimiento de reinicio recuperará todas las unidades de trabajo (transacciones) completadas con éxito pero cuyas modificaciones no lograron grabarse físicamente antes de la caída; por tanto, y como se dijo antes, las transacciones son en realidad la unidad de recuperación.

RECUPERACION DEL SISTEMA Y DE LOS MEDIOS DE ALMACENAMIENTO.

Existen dos categorías de fallas en un sistema:

- *Fallas del sistema* (por ejemplo, interrupción del suministro de electricidad), las cuales afectan a todas las transacciones que se están realizando pero no dañan físicamente a la base de datos. Las fallas del sistema se conocen como *caídas suaves.*
- *Fallas de los medios de almacenamiento* (por ejemplo, un aterrizaje de cabezas en el disco), las cuales si causan daños a la base de datos, o una porción de ella, y afectan al menos a las transacciones que están utilizando esa porción. Las fallas de los medios de almacenamiento se denominan a veces *caídas duras.*

FALLAS DEL SISTEMA

El punto critico es que *se pierde el contenido de la memoria principal.* Por tanto, ya no se conocerá el estado preciso de la transacción en que se estuviera realizando en el momento de la falla; esa transacción jamás se podrá completar con éxito, por lo cual será preciso anular (retroceder) cuando se reinicie el sistema.

Cada cierto intervalo previamente establecido el sistema “establece un punto de revisión” de manera automática.

El establecimiento de un punto de revisión implica:

a) grabar físicamente el contenido de los buffers de datos en la base de datos física .
b) grabar físicamente un *registro de punto de revisión* especial en la bitácora física. El registro de punto de revisión incluye una lista de todas las transacciones que se estaban realizando en el momento de establecerse el punto de revisión.

Para comprender la forma como se utiliza esta información, examínese la figura 16.1, la cual deberá leerse de la siguiente manera:

> Al reiniciarse el sistema deberá anularse las transacciones de los tipos T3 y T5, y deberán realizarse de nuevo las transacciones de los tipos T2 y T4. Las transacciones T1 no entran en absoluto en el proceso de reinicio.

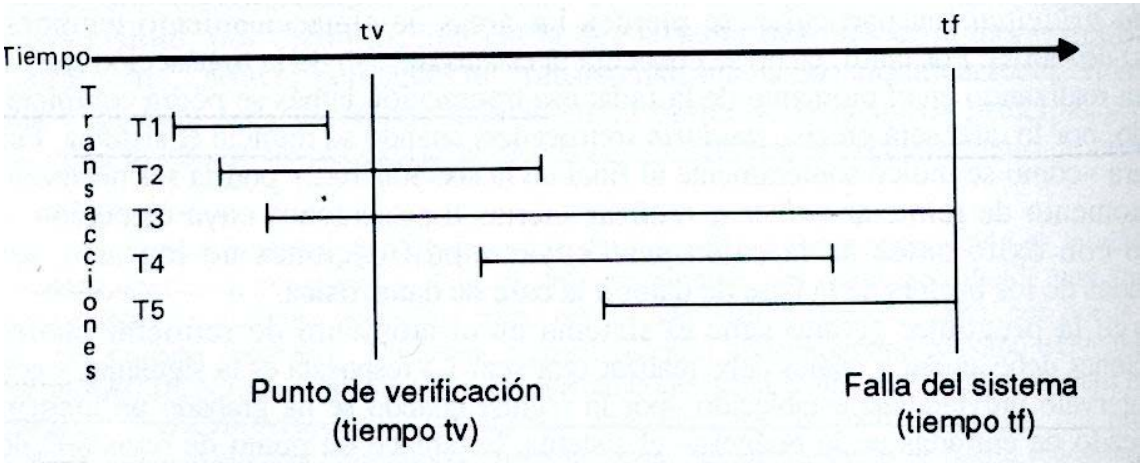


Fig. 12.1 Cinco categorías de transacciones.

FALLAS DE LOS MEDIOS DE ALMACENAMIENTO

La recuperación de una falla semejante implica en esencia cargar de nuevo (o restaurar) la base de datos a partir de una copia de respaldo (o vaciado) y utilizar después la bitácora para realizar de nuevo todas las transacciones terminadas desde que se hizo esa copia de respaldo.

SEGURIDAD E INTEGRIDAD: INTRODUCCIÓN.

Seguridad se refiere a la protección de los datos contra una revelación, alteración o destrucción no autorizada.

Integridad se refiere a la exactitud o validez de los datos.

La seguridad implica asegurar que los usuarios están autorizados para llevar a cabo lo que tratan de hacer.

La integridad implica asegurar que lo que tratan de hacer es corecto.

El sistema necesita estar al tanto de ciertas restricciones que no deben ser violadas por los usuarios; esas *restricciones* se deben especificar en algún lenguaje apropiado, y se deben mantener en el catálogo o diccionario del sistema .

SEGURIDAD: CONSIDERACIONES GENERALES

El problema de la seguridad tiene muchos aspectos, entre ellos los siguientes:

- Aspectos legales, sociales y éticos.
- Controles físicos.
- Cuestiones de política interna.
- Problemas de operación.
- Controles de equipo.

- Seguridad del sistema operativo.
- Materias de relevancia específica para el sistema mismo de base de datos (por ejemplo, ¿tiene el sistema de base de datos un concepto de propiedad de los datos?).

Nos limitaremos en general a considerar los aspectos incluidos sólo en esta última categoría.

Un usuario dado tendrá por lo regular diferentes derechos de acceso o *autorización* sobre diferentes objetos de información (por ejemplo, autorización para usar SELECT(seleccionar) solo con una tabla, autorización para usar SELECT Y UPDATE (modificar) con otra, etcétera),.

El usuario A podría tener autorización(solo) SELECT con alguna tabla dada, el usuario B podría tener al mismo tiempo autorización para usar tanto SELECT como UPDATE con esa misma tabla.

En el caso especifico de SQL, el sistema cuenta con dos características diferentes mas o menos independientes implicadas en el mantenimiento de la seguridad:

1) El mecanismo de vistas, el cual puede servir para ocultar datos confidenciales a usuarios no autorizados.
2) El subsistema de autorización, mediante el cual usuarios con derechos específicos pueden conceder de manera selectiva y dinámica eso derechos a otros usuarios, y después revocar esos derechos, si lo desean.

Por supuesto todas las decisiones sobre qué derechos deben concederse a ciertos usuarios, son decisiones de política no técnica. Lo único que puede hacer el DBMS es obligar al cumplimiento de esas decisiones una vez tomadas.

Para que el DBMS pueda llevar a cabo esas funciones en forma apropiada reuiere que:

a) Los resultados de esas decisiones deben darse a conocer al sistema (en SQL esto se hace con las **proporciones GRANT (conceder) y REVOKE (revocar)**), y el sistema debe ser capaz de recordarlas (esto se hace grabándolas en el catálogo, en forma de **restricciones de autorización**.)
b) Debe existir alguna forma de identificar una solicitud de acceso dada contra las restricciones de autorización aplicables.
c) Para poder decir cuales restricciones son aplicables a una solicitud dada, el sistema debe ser capaz de reconocer el origen de dicha solicitud; es decir, debe ser capaz de reconocer al usuario específico del cual proviene una determinada solicitud. Por esa razón, cuando los usuarios obtienen acceso al sistema, casi siempre se les pide proporcionar no solo su identificador de usuario (decir quiénes son) sino también una contraseña (para demostrar que son quienes dicen ser). En teoría, solo el sistema y los usuarios legítimos del identificador en cuestión conocen la contraseña.

SEGURIDAD EN SQL.

VISTAS DE SEGURIDAD

Por ejemplo, si a un usuario se le permite tener acceso a registros contables de proveedores, pero de los proveedores situados en Paris;

```
CREATE VIEW PROVEEDORES_PARISIANOS
    AS SELECT S#, SNOMBRE, SITUACIÓN, CIUDAD
    FROM      S
    WHERE CIUDAD = 'Paris' ;
```

Los usuarios de esta vista ven un "subconjunto horizontal", o sea un subconjunto de filas o subconjunto dependiente del valor de la tabla base S.

GRANT (conceder) y REVOKE (revocar)

Cuando se instala un DBMS por primera vez, parte del procedimiento de instalación implica la designación de un usuario con privilegios especiales como ***administrador de sistema*** para ese sistema instalado.

Ese usuario privilegiado recibe de manera automática una autorización especial que confiere a quien la posee el derecho de realizar todas y cada una de las operación manejadas por el sistema.

Supongamos ahora que el operador de sistema concede a algún otro usuario *U* el derecho de crear algún objeto -una vista o una tabla base– el usuario *U* obtiene de manera automática todo tipo de derechos sobre ese objeto, incluyendo en particular el derecho de conceder esos derechos a otro usuario.

La concesión de derechos se hace mediante la proposición GRANT (conceder). He aquí algunos ejemplos

```
GRANT SELECT ON TABLE  S  CARLOS;
GRANT ALL ON TABLE  S, P, SP TO FERNANDO, MARIA;
GRANT SELECT ON TABLE   P   TO  PUBLIC;
GRANT INDEX ON TABLE  S  TO FELIPE;
```

PUBLIC es una palabra clave especial que representa a "todos los usuarios del sistema". En general, los derechos aplicables a tablas (tanto tablas bases como vistas) son los siguientes:

- SELECT (SELECCIONAR)
- UPDATE (ACTUALIZAR, SER ESPECIFICO POR COLUMNA)
- DELETE (ELIMINAR)
- INSERT (INSERTAR)

Las dos restantes se aplican sólo a tablas bases.

- ALTER (derecho a ejecutar ALTER TABLE sobre la tabla)
- INDEX (derecho a ejecutar CREATE INDEX sobre la tabla)

Si el usuario *U* concede cierta autorización a otro usuario *U2*, el usuario U puede *revocar* después esa autorización. Esto se hace con la proporción REVOKE

He aquí algunos ejemplos:

```
REVOKE SELECT ON TABLE  S  FROM CARLOS;
```

REVOKE UPDATE ON TABLE S FROM JAIME;

Si el usuario U1 tiene el derecho de conceder cierta autorización a otro usuario U2 (especificando WITH GRANT OPTION en la proporción GRANT). U2 también tiene el derecho a pasarle a U3 la opción GRANT, etcétera.

Por ejemplo:

Usuario *U1:*

GRANT SELECT ON TABLE S TO U2 WITH GRANT OPTION;

Usuario *U2:*

GRANT SELECT ON TABLE S TO U3 WITH GRANT OPTION;

Usuario *U3:*

GRANT SELECT ON TABLE S TO U4 WITH GRANT OPTION;

Y así sucesivamente. Si el usuario *U1* emite ahora :

REVOKE SELECT ON TABLE S FROM U2;

La revocación se propagará de U2 a U3 y la de U3 a U4.

OTROS ASPECTOS DE SEGURIDAD

Es importante no dar por sentado que el sistema de seguridad es perfecto, **se hace indispensable un *seguimiento de auditoría***, Si por ejemplo, las discrepancias en los datos hacen sospechar que los datos se han alterado, el seguimiento de auditoría puede servir para examinar lo sucedido y verificar que las cosas estén bajo control (o, si no están, ayudar a describir al culpable).

Una entrada representativa en el seguimiento de auditoría podría contener la siguiente información:

- Operación (por ejemplo UPDATE).
- Terminal desde la cual se invoco la operación.
- Usuario que invoque la operación.
- Fecha y hora de la operación.
- Base de datos, tabla, registros y campos afectados.
- Valor anterior del campo.
- Valor nuevo del campo.

En algunos casos, el simple hecho de mantener un seguimiento de auditoria es suficiente para desanimar a un posible infiltrador .

Es posible añadir un nivel mas de seguridad mediante el cifrado de los datos. La idea básica aquí es que los datos pueden almacenarse físicamente en el disco, o transmitirse a través de las líneas de comunicación, en forma codificada o cifrada.

INTEGRIDAD: CONSIDERACIONES GENERALES.

El término ‘integridad’ se refiere a la corrección de la información contenida en la base de datos. La verificación de integridad se realiza hoy día mediante código de procedimientos escrito por los usuarios. Obviamente, seria preferible poder especificar restricciones de integridad en una forma más declarativa y hacer así que el sistema se encargara de la verificación

Una restricción de integridad puede considerarse como una condición, que debe ser todos los estados correctos de la base de datos. Un ejemplo sencillo de tal condición podría ser

FORALL SX (SX.SITUACIÓN > 0)

(“para todos los proveedores SX, la situación de SX debe ser positiva”).

Unidad XIII: Ejemplos de aplicación

NARRATIVA: COMPRA DE MERCADERÍA.

Dada la siguiente factura (que se muestra a continuación), producto de la compra de mercaderías, elaborar la base de datos Relacional que pueda contener eficientemente los datos de las facturas efectuadas:

ALMACENES: EL BARATO S.A	
FACTURA Nº	1568
FECHA	23/01/2005
VENDEDOR	MERILES, DARIO
DATOS DEL CLIENTE:	
NOMBRE	GOMEZ, LUIS ALBERTO.
DNI Nº	12356897
DIRECCIÓN	Av. Paraguay 234.
TELEFONO	4234568
DIRECCIÓN DE ENVIO	Zabala 345
CIUDAD DE ENVIO	SALTA

DETALLE DE LA COMPRA:

Datos de la mercadería				
CODIGO MERCADERÍA	**DESCRIPCIÓN**	**PRECIO UNITARIO**	**CANTIDAD**	**TOTAL**
023	LECHE SANCOR LARGAVIDA	$10	20	$200
053	YERBA AMANECER x 1KILO	$5	50	$250
014	AZUCAR LEDESMA x 1 KILO	$1	10	$10
			TOTAL	**$460**

Desarrollo del ejemplo:
Podríamos volcar toda la información en una sola entidad o tabla:

FACTURAS			
NUM_FACT	**FECHA**	**DATOS DEL CLIENTE**	**DETALLE DE LA COMPRA**
1568	23/01/2005	NOMBRE: GOMEZ, LUIS. DNI: 12356897 DIRECCION: Av. Paraguay 234 TELEFONO: 234568 DIREC_ENVIO: Zabala 345 CIUDAD_ENVIO: SALTA	Datos de la mercadería: COD_MERC: 023 DESCRIPCION: Leche Sancor LargaVida PRECIO: $10. CANTIDAD: 20. TOTAL:$200. .....

Como se puede apreciar la relación (tabla) **FACTURAS** no contiene todos sus atributos (campos) atómicos.
Nota: Falta agregar datos del vendedor y el total de la factura.
Por ejemplo **DATOS_DEL_CLIENTES** contiene NOMBRE, DNI, DIRECION, TELEFONO, ETC. Igual situación se aprecia en **DETALLE_DE_LA_COMPRA.**
La manera de solucionarlo, es crear nuevas entidades:

CLIENTES					
DNI	**NOMBRE**	**DIRECCIÓN**	**TELÉFONO**	**DIREC_ENVIO**	**CIUDAD_ENVIO**
12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	SALTA
10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	SALTA
18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	JUJUY
19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	CATAMARCA
14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	JUJUY

DETALLES		
COD_MERCAD	**CANTIDAD**	**TOTAL**
023	20	200
053	50	250
014	10	10

FACTURAS		
NUM_FACT	**FECHA**	**TOTAL**
1568	23/01/2005	$460

VENDEDORES			
DNI_VEND	**NOMBRE**	**DIREC_VEND**	**TELEF_VEND**
15235487	MERILES, DARIO	MENDOZA 600	4235687

MERCADERIAS		
COD_MERCAD	**DESCRIPCION**	**PRECIO**
023	LECHE SANCOR LARGAVIDA	$10
053	YERBA AMANECER x 1KILO	$5
014	AZUCAR LEDESMA x 1 KILO	$1

Todos los atributos (propiedades o campos) son atómicos, puede apreciar que al insertar tuplas en las tablas no hay datos repetidos sin sentido, entre otros cosas.

Ahora nos abocamos al sentido semántico, es decir analizamos las relaciones existentes.

En primer lugar, se vé claramente que la tabla DETALLES se refiere a la FACTURA emitida, por lo que hay una relación de uno a muchos. Es decir, dado un NUM_FACT=1568 (por ejemplo), si vemos la tabla FACTURA el atributo NUM_FACT=1568 aparece una y solo una vez; en cambio en la tabla DETALLES el mismo puede aparecer muchas veces.

Esta situación implica que la tabla DETALLE tiene que poseer un atributo (o más) que sea clave foránea y que se corresponda con la clave primaria de FACTURA (Relación de uno a muchos).

DETALLE quedaría de la siguiente forma:

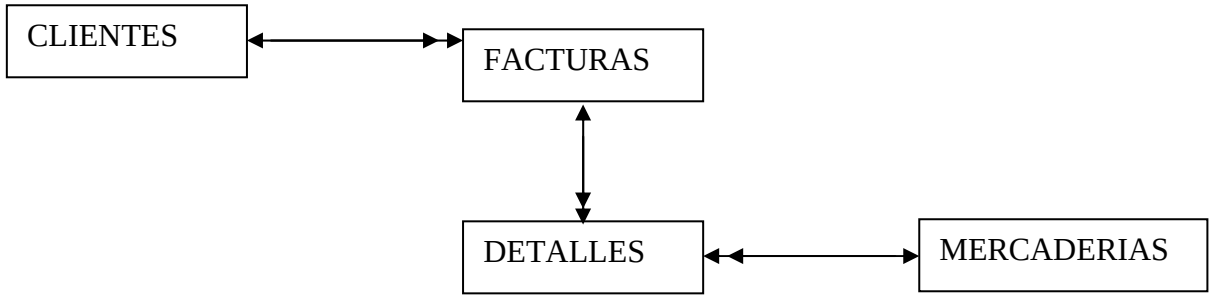
DETALLE			
NUM_FACT	**COD_MERCAD**	**CANTIDAD**	**TOTAL**
1568	023	20	200
1568	053	50	250
1568	014	10	10

Además sabemos que los clientes adquieren mercaderías, esa relación se puede llamar COMPRA:

Como la relación es de muchos a muchos se debe insertar una nueva tabla, la cual contendrá los campos claves de CLIENTES y MERCADERIAS.

COMPRAS						
DNI	**COD_MERCAD**	**NUM_FACT**	**FECHA**	**TOTAL**	**DESCRIPCION**	**PRECIO**

Como podrá ver COMPRAS es una combinación de las tablas FACTURAS y DETALLES. La tabla DETALLES se relaciona con MECADERIAS, puesto que posee como atributos COD_MERCAD.
Por lo que quedaría de la siguiente forma:

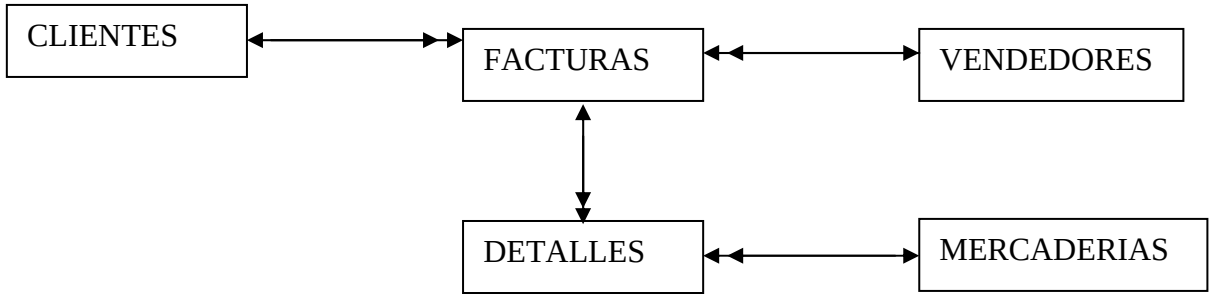


FACTURAS Y DETALLES, reemplazan COMPRAS.

Obviamente, a la tabla FACTURAS, hay que insertarle el campo (o los campos) que conforman el campo clave de CLIENTES.

FACTURAS			
NUM_FACT	**FECHA**	**TOTAL**	**DNI**
1568	23/01/2005	$460	12356897

Finalmente vendedor está relacionado con la compra, es decir con la factura.
El DER definitivo será:



CLIENTES					
DNI@	**NOMBRE**	**DIRECCIÓN**	**TELÉFONO**	**DIREC_ENVIO**	**CIUDAD_ENVIO**
12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	SALTA
10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	SALTA
18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	JUJUY
19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	CATAMARCA
14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	JUJUY

FACTURAS				
NUM_FACT@	**FECHA**	**TOTAL**	**DNI**	**DNI_VEND**
1568	23/01/2005	$460	12356897	15235487
1569	23/01/2005	$85	14231564	14254368
1570	25/01/2005	$170	12356897	14254368

VENDEDORES			
DNI_VEND@	**NOMBRE**	**DIREC_VEND**	**TELEF_VEND**
15235487	MERILES, DARIO	MENDOZA 600	4235687
14254368	GUAYMAS, ROGELIO	BROWN 254	4251235

DETALLE			
NUM_FACT@	**COD_MERCAD@**	**CANTIDAD**	**TOTAL**
1568	023	20	$200
1568	053	50	$250
1568	014	10	$10
1569	009	5	$10
1569	053	15	$75
1570	008	20	$60
1570	023	8	$80
1570	010	30	$30

MERCADERIAS		
COD_MERCAD@	**DESCRIPCION**	**PRECIO**
008	FIDEOS AL HUEVO	$3
009	GASEOSA SS	$2
010	JUGOS RICOS	$1
014	AZUCAR LEDESMA x 1 KILO	$1
023	LECHE SANCOR LARGAVIDA	$10
053	YERBA AMANECER x 1KILO	$5

Finalmente observe el campo TOTAL, aparece en las tablas DETALLES y FACTURAS. En principio este campo es CALCULADO, es decir no necesita ser guardada la información, pués si se la requiere, la misma puede calcularse a partir de la cantidad pedida del producto por el precio unitario del mismo.

Por lo antes expuesto es claro que el campo TOTAL de la tabla DETALLE no tiene que existir

DETALLE		
NUM_FACT@	**COD_MERCAD@**	**CANTIDAD**
1568	023	20
1568	053	50
1568	014	10
1568	014	10
1569	009	5
1569	053	15
1570	008	20
1570	023	8
1570	010	30

En cambio el campo TOTAL de la tabla FACTURAS, es conveniente que quede, puesto que si bien se puede calcular, el esfuerzo para lograrlo es considerable.

CONSULTAS SQL

Ahora empezaremos a utilizar las tablas relacionadas anteriormente, a través del uso de las sentencias SQL. Las mismas se pueden realizar:

a) Por medio de un editor SQL (es lo que emplearemos a continuación).
b) Usando el SQL embebido de algún lenguaje de aplicación (VISUAL BASIC, DELPHI, etc.)

CONSULTAS QUE INVOLUCRAN SOLO UNA TABLA:

CLIENTES					
DNI@	**NOMBRE**	**DIRECCIÓN**	**TELÉFONO**	**DIREC_ENVIO**	**CIUDAD_ENVIO**
12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	SALTA
10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	SALTA
18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	JUJUY
19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	CATAMARCA
14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	JUJUY

PROYECCION

-Mostrar el nombre de todos los clientes.

```
SELECT CLIENTES.nombre
FROM CLIENTES;
```

Nombre
RUIZ, CARLO
GOMEZ, LUIS
MENDEZ, ARIEL
YAPURA, LUIS
SAENZ, DARIO

-Mostrar el nombre y el dni de todos los clientes.

```
SELECT CLIENTES.nombre, CLIENTES.dni
FROM CLIENTES;
```

Nombre	dni
RUIZ, CARLO	10253698
GOMEZ, LUIS	12356897
MENDEZ, ARIEL	14231564
YAPURA, LUIS	18253698
SAENZ, DARIO	19234568

-Mostrar todos los datos de los clientes.

```
SELECT *
FROM CLIENTES;
```

dni	nombre	Direccion	telefono	direc_envio	ciudad_envio
10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	SALTA
12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	SALTA
14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	JUJUY
18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	JUJUY
19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	CATAMARCA

SELECCIÓN (RESTRICCION)

-Mostrar el nombre de los clientes cuya ciudad de envio sea SALTA.

```
SELECT nombre
FROM CLIENTES
WHERE ciudad_envio = "SALTA";
```

Nombre
GOMEZ, LUIS
RUIZ, CARLO

-Mostrar el nombre de los clientes cuya ciudad de envio sea SALTA y cuyo dni sea mayor que doce millones.

```
SELECT nombre
FROM CLIENTES
WHERE ciudad_envio = "SALTA" and dni > "12000000";
```

Nota :“12000000” va entre comillas, porque se declaró como texto

Nombre
GOMEZ, LUIS

DISTINCT

-Mostrar las ciudades a donde se envían los pedido de los clientes (no mostrar repeticiones).

```
SELECT DISTINCT ciudad_envio
FROM CLIENTES;
```

ciudad_envio
CATAMARCA
JUJUY
SALTA

RECUPERACIÓN CON ORDENAMIENTO

-Mostrar el dni y nombre de los clientes, ordenados alfabéticamente en orden descendente.

```
SELECT dni, nombre
FROM CLIENTES
ORDER BY nombre DESC;
```

dni	nombre
18253698	YAPURA, LUIS
19234568	SAENZ, DARIO
10253698	RUIZ, CARLO
14231564	MENDEZ, ARIEL
12356897	GOMEZ, LUIS

FUNCIONES DE AGREGADOS

SQL ofrece una serie de funciones de agregados especiales para ampliar su capacidad básica de recuperación de información. Esta funciones son:

- COUNT (cuenta).
- SUM (suma).
- AVG (promedio).
- MAX (máximo).
- MIN (mínimo).

FACTURAS				
NUM_FACT@	FECHA	TOTAL	DNI	DNI_VEND
1568	23/01/2005	$460	12356897	15235487
1569	23/01/2005	$85	14231564	14254368
1570	25/01/2005	$170	12356897	14254368

-Indicar cuántas compras realizó el cliente cuyo dni es 12356897.

```
SELECT count(*)
FROM FACTURAS
WHERE dni= "12356897";
```

Expr1000
2

Como no se muestra un campo en sí, aparece la palabra Expr1000, si deseamos que figure un texto debemos usar AS de la siguiente forma:

```
SELECT count(*) AS "TOTAL DE COMPRAS DEL CLIENTE 12356897"
FROM FACTURAS
WHERE dni="12356897";
```

"TOTAL DE COMPRAS DEL CLIENTE 12356897"
2

También podemos usar un campo con Etiquetas:

```
SELECT count(*) , " TOTAL DE COMPRA"
FROM FACTURAS AS FAC
WHERE FAC.dni="12356897";
```

Expr1000	Expr1001
2	TOTAL DE COMPRA

FAC se usa en cuenta de FACTURA, es como un sinónimo.

-Indicar cuánto gasto en total el cliente cuyo dni es 12356897.

```
SELECT sum(total)
FROM FACTURAS
WHERE dni="12356897";
```

Expr1000
$630,00

-Indicar el promedio de todas las compras efectuadas por el cliente 12356897.

```
SELECT AVG(total)
FROM FACTURAS
WHERE dni="12356897";
```

Expr1000
$315,00

-Indicar cuál fue la máxima compra realizada por el cliente 12356897.

```
SELECT max(total)
FROM FACTURAS
WHERE dni="12356897";
```

Expr1000
$460,00

-Indicar cuál fue la mínima compra realizada por el cliente 12356897.

```
SELECT min(total)
FROM FACTURAS
WHERE dni="12356897";
```

Expr1000
$170,00

RECUPERACIÓN DE DATOS CON LIKE(COMO).

-Mostrar el documento de todos los clientes cuyo Ciudad de Envío empiece con "S".

```
SELECT dni, ciudad_envio
FROM CLIENTES
WHERE ciudad_envio  LIKE  "S*";
```

Dni	ciudad_envio
12356897	SALTA
10253698	SALTA

-Mostrar el documento de todos los clientes cuyo Ciudad de Envío tenga exactamente 5 letras y termine con "Y".

```
SELECT dni, ciudad_envio
FROM CLIENTES
WHERE ciudad_envio  LIKE  "????Y";
```

dni	ciudad_envio
18253698	JUJUY
14231564	JUJUY

USO DE IN y NOT IN

-Mostrar el nombre y dni de los clientes cuyo dni sea 18253698 o 20000000.

```
SELECT dni, nombre
FROM CLIENTES
WHERE dni  in  ("18253698", "20000000");
```

dni	nombre
18253698	YAPURA, LUIS

-Mostrar el nombre y dni de los clientes cuyo dni no sean 18253698 o 20000000 o 30000000.

```
SELECT dni, nombre
FROM CLIENTES
WHERE dni not  in  ("18253698", "20000000", “30000000”);
```

dni	nombre
12356897	GOMEZ, LUIS
10253698	RUIZ, CARLO
19234568	SAENZ, DARIO
14231564	MENDEZ, ARIEL

EMPLEO DE GROUP BY (AGRUPAR POR)

-Mostrar el número de cada ciudad de envío de los clientes.

```
SELECT ciudad_envio, count(*) AS "CANTIDAD"
FROM CLIENTES
GROUP BY ciudad_envio;
```

ciudad_envio	"CANTIDAD"
CATAMARCA	1
JUJUY	2
SALTA	2

Tenga cuidado, al usar GROUP BY ya no trabajamos con 5 registro (como posee clientes al principio) sino con tres:

	dni	nombre	direccion	telefono	direc_envio	ciudad_envio
REG 1	10253698	RUIZ, CARLO	SAN LUIS 35	4253698	SAN LUIS 35	**SALTA**
	12356897	GOMEZ, LUIS	AV. PARAGUAY 234	4234568	ZABALA 345	**SALTA**
REG 2	14231564	MENDEZ, ARIEL	COLON 34	4215874	MENDOZA 324	**JUJUY**
	18253698	YAPURA, LUIS	SANTA FE 345	4213875	SANTA FE 345	**JUJUY**
REG 3	19234568	SAENZ, DARIO	BUENOS AIRES 23	421536	LA RIOJA 456	**CATAMARCA**

De modo que si quiero mostrar algún campo, estos tiene que corresponder con los agrupamientos, en el ejemplo, al poner ciudad_envio no suscita ningún inconveniente puesto que REG 1 posee SALTA en ciudad _envio, REG 2 posee JUJUY y REG 3 posee CATAMARCA.

Si quisiera mostrar, por ejemplo, dni daría error, pues para el primer registro REG 1 ¿qué mostraría, el dni 10253698 o el dni 12356897?

```
SELECT dni, count(*) AS "CANTIDAD"
FROM CLIENTES
GROUP BY ciudad_envio;
```

EMPLEO DE HAVING (CON):

Having es para el group by lo que where es para select, es decir having se utiliza con group by.

Si deseamos poner una condición a los grupos seleccionados, entonces usamos having.

En el ejemplo anterior tenemos como resultado tres registros (luego del group by), si ahora decimos que nos muestre la ciudad de envio cuya cantidad sea superior a 1(no incluir a CATAMARCA), tenemos que usar having.

-Mostrar el número de cada ciudad de envío de los clientes, cuya cantidad sea mayor que uno.

```
SELECT ciudad_envio, count(*) AS "CANTIDAD"
FROM CLIENTES
GROUP BY ciudad_envio
HAVING count(*) > 1;
```

ciudad_envio	"CANTIDAD"
JUJUY	2
SALTA	2

En Having puede ir :

1) Algún campo colocados luego del SELECT.
2) Una función de agregado.

En el ejemplo la función de agregado count(*) se haya en los dos lugares (SELECT y HAVING), pero puede que sólo esté al lado de HAVING.

```
SELECT ciudad_envio
FROM CLIENTES
GROUP BY ciudad_envio
HAVING count(*) > 1;
```

ciudad_envio
JUJUY
SALTA

UTILIZACIÓN DE DOS TABLAS.

Al colocar dos (o mas tablas) en el FROM, se realiza el producto cartesiano de las mismas, luego se realiza la proyección y selección.

Por ejemplo si vemos las tablas VENDEDORES y FACTURAS:

VENDEDORES			
DNI_VEND@	**NOMBRE**	**DIREC_VEND**	**TELEF_VEND**
15235487	MERILES, DARIO	MENDOZA 600	4235687
14254368	GUAYMAS, ROGELIO	BROWN 254	4251235

FACTURAS				
NUM_FACT@	**FECHA**	**TOTAL**	**DNI**	**DNI_VEND**
1568	23/01/2005	$460	12356897	15235487
1569	23/01/2005	$85	14231564	14254368
1570	25/01/2005	$170	12356897	14254368

Si ejecutamos la siguiente consulta:

SELECT *
FROM VENDEDORES, FACTURAS;

VENDEDORES.dni_vend	nombre	direc_vend	telef_vend	num_fact	fecha	total	dni	FACTURAS.dni_vend
15235487	MERILES, DARIO	MENDOZA 600	4235687	1568	23/01/2005	$460,00	12356897	15235487
14254368	GUAYMAS, ROGELIO	BROWN 254	4251235	1568	23/01/2005	$460,00	12356897	15235487
15235487	MERILES, DARIO	MENDOZA 600	4235687	1569	23/01/2005	$85,00	14231564	14254368
14254368	GUAYMAS, ROGELIO	BROWN 254	4251235	1569	23/01/2005	$85,00	14231564	14254368
15235487	MERILES, DARIO	MENDOZA 600	4235687	1570	25/01/2005	$170,00	12356897	14254368
14254368	GUAYMAS, ROGELIO	BROWN 254	4251235	1570	25/01/2005	$170,00	12356897	14254368

Nos muestra el producto cartesiano, es decir seis registros (2x3=6) (dos de VENDEDORES y tres de FACTURAS). Aparte muestra los campos de ambas tablas (*).
Esta consulta carece de significado, por lo que es necesario:

a) Que las tablas estén relacionadas (tengan un campo en común)
b) Que se haga una Proyección. (se coloquen nombres de campos luego del SELECT).
c) Que se haga una selección, de ser necesario.

Por ejemplo:

-Mostrar el nombre de los vendedores que hicieron al menos una venta en el año 2005.

SELECT distinct(VENDEDORES.nombre)
FROM VENDEDORES, FACTURAS
WHERE VENDEDORES.dni_vend=FACTURAS.dni_vend
and FACTURAS.fecha > 31/12/2004;

nombre
GUAYMAS, ROGELIO
MERILES, DARIO

Observe que en el where se coloca la relacion entre las dos tablas, le decimos que VENDEDORES.dni_vend=FACTURAS.dni_vend.
La proyección es distinct(VENDEDORES.nombre)
La selección es FACTURAS.fecha > 31/12/2004.

CONSULTAS DE UNION

Los registros que se muestran pertenecen a dos tablas diferentes, por lo que tienen que ser iguales.

```
SELECT nombre
FROM CLIENTES
        UNION
                SELECT nombre
                FROM VENDEDORES;
```

nombre
GOMEZ, LUIS
GUAYMAS, ROGELIO
MENDEZ, ARIEL
MERILES, DARIO
RUIZ, CARLO
SAENZ, DARIO
YAPURA, LUIS

Recuerde que definimos cinco clientes y dos vendedores.

UTILIZACIÓN DE TRES (o mas) TABLAS

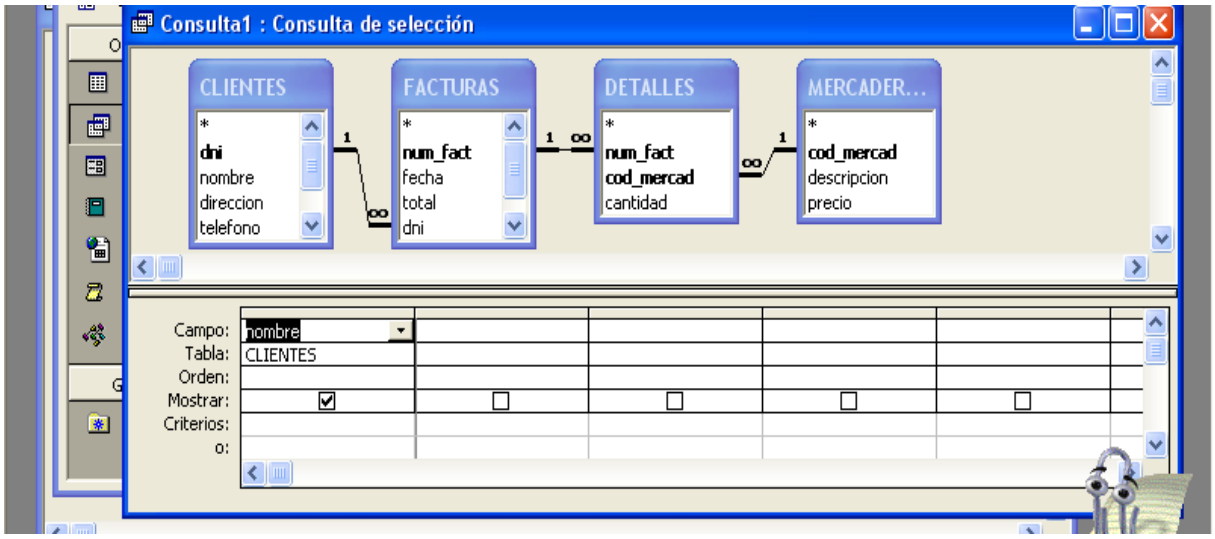
Solo tiene sentido si las tablas están relacionadas, es decir poseen campos en común.
Por ejemplo:

-Mostrar el nombre de los clientes que adquirieron "YERBA AMANECER x 1KILO".

```
SELECT CLIENTES.nombre
FROM CLIENTES, FACTURAS, DETALLES, MERCADERIAS
WHERE (CLIENTES.dni = FACTURAS.dni)
        and (FACTURAS.num_fact = DETALLES.num_fact)
        and ( DETALLES.cod_mercad= MERCADERIAS.cod_mercad)
        and MERCADERIAS.descripcion="YERBA AMANECER x 1KILO";
```

Para responder hay que recorre cuatro tablas, pues nombre del cliente se encuentra en la primera tabla CLIENTES y descripción de la mercadería se haya en la cuarta tabla MERCADERIAS (ver DER).
Hay que recorrer CLIENTES, FACTURAS, DETALLES y finalmente MERCADERIAS.

En el where colocamos la relación entre cada tabla, por ejemplo (CLIENTES.dni = FACTURAS.dni) indica que CLIENTES se relaciona con FACTURAS a través del campo dni.



nombre
GOMEZ, LUIS
MENDEZ, ARIEL

Access utiliza el JOIN, el principio es el mismo que explicamos anteriormente:

```
SELECT CLIENTES.nombre
 FROM MERCADERIAS INNER JOIN ((CLIENTES INNER JOIN FACTURAS
     ON CLIENTES.dni = FACTURAS.dni) INNER JOIN DETALLES ON
     FACTURAS.num_fact = DETALLES.num_fact) ON
     MERCADERIAS.cod_mercad = DETALLES.cod_mercad
  WHERE  MERCADERIAS.descripcion="YERBA AMANECER x 1KILO";
```

ANIDAMIENTOS IN

Otra forma de realizar el ejercicio anterior es usando anidamientos con la cláusula IN.

```
SELECT CLIENTES.nombre
FROM CLIENTES
WHERE CLIENTES.dni  IN
          (SELECT  FACTURAS.dni
               FROM FACTURAS
              WHERE FACTURAS.num_fact  IN
                  ( SELECT DETALLES.num_fact
                        FROM DETALLES
                        WHERE DETALLES.cod_mercad IN
                          ( SELECT  MERCADERIAS.cod_mercad
                            FROM MERCADERIAS
                             WHERE MERCADERIAS.descripcion =
                                        "YERBA AMANECER x 1KILO")));
```

Cada SELECT produce una salida que es evaluada por el siguiente SELECT anidado .

Por ejemplo:

```
                          ( SELECT  MERCADERIAS.cod_mercad
                            FROM MERCADERIAS
                             WHERE MERCADERIAS.descripcion =
                                  "YERBA AMANECER x 1KILO")
```

Dá como reultado 053 por lo que sería similar colocar:

```
SELECT CLIENTES.nombre
FROM CLIENTES
WHERE CLIENTES.dni  IN
          (SELECT  FACTURAS.dni
               FROM FACTURAS
              WHERE FACTURAS.num_fact  IN
                  ( SELECT DETALLES.num_fact
                        FROM DETALLES
                        WHERE DETALLES.cod_mercad IN (053)
```

Y así sucesivamente.

NORMALIZACION: EJEMPLO:
Normalice la siguiente tabla de libros:

LIBROS

@ISBN	@UBICACION	TITULO	AUTORES	IMPRENTA	ESTADO	DIREC_IMP
15	F1	BASE DATO	DATE y SANCHEZ	Mc Grawn	BUENO	SAN JUAN 72 BS AS.
15	F2	BASE DATO	DATE y SANCHEZ	Mc Graun	MALO	SAN JUAN 72 BS AS.

Suponemos que el ISBN se repite para libros de una misma partida.

PRIMERA FORMA NORMAL:
Autores no es un campo atómico. Si deseo ubicar a un solo autor, se dificulta la búsqueda.
Genero la tabla AUTORES:

AUTORES

@ISBN	AUTORES
15	DATE

SEGUNDA FORMA NORMAL:
TITULO, IMPRENTA, DIREC_IMP solo dependen de ISBN.
ESTADO depende de ISBN y de UBICACIÓN
Genero las tablas LIBROS y ESTADO_LIBROS

LIBROS

@ISBN	TITULO	IMPRENTA	DIREC_IMP
15	BASE DATO	Mc Grawn	SAN JUAN 72 BS AS.

ESTADO_LIBROS

@ISBN	@UBICACION	ESTADO
15	F1	BUENO
15	F2	MALO

TERCERA FORMA NORMAL:
DIREC_IMPR depende de IMPRENTA.
Genero la tabla IMPRENTA y modifico la tabla LIBROS

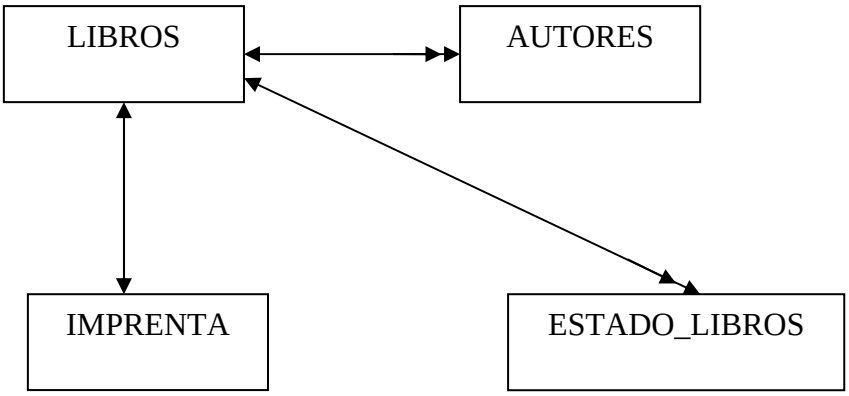
LIBROS

@ISBN	TITULO	IMPRENTA
15	BASE DATO	Mc Grawn

IMPRENTA

@IMPRENTA	DIREC_IMP
Mc Grawn	SAN JUAN 72 BS AS.

Si desea puede colocar un Cod_IMPRENTA.



Las flechas de libros – autores unen el pk ISBN y el FK ISBN.
Las flechas de libros – estado_libros unen el pk ISBN y el FK ISBN.

Fíjese que la relación uno a uno implica que los datos de imprenta se pueden guarda en libros generando una sola tabla.

EJERCICIO INTERESANTE:

Dado un PRESUPUESTO PUBLICO, construir la tabla para guardar los valores del presupuesto. Genere consulta sql pertinentes al caso.
Aclaración: Un presupuesto implica los números de cuentas y montos, por ejemplo:

CUENTA	**CONCEPTO**	**MONTO**	
3	TOTAL DE EROGACIONES	6.213.680,00	1
3.1	EROGACIONES CORRIENTE	6.158.680,00	2
3.1.1	PERSONAL	4.748.480,00	3
3.1.1.1	SUELDO BASICO	2.402.000,00	4
3.1.1.1.6	PLANTA PERMANENTE	919.640,00	5
3.1.1.1.7	PLANTA TRANSITORIA	1.482.360,00	6
3.1.1.2	ANTIGÜEDAD	362.800,00	7
3.1.1.2.5	PLANTA PERMANENTE	362.800,00	8
3.1.1.2.6	PLANTA TRANSITORIA	0	9
3.1.1.3	BONIFICACIONES Y SUPLEMENTOS	1.499.080,00	10
……			….
3.1.1.4	APORTES PATRONALES	484.600,00	15
…….			…
3.1.2	BIENES Y SERVICIOS	1.340.200,00	25
3.1.3	TRANSFERENCIAS	70.000,00	26
……			….

Tiene que comprobarse, por ejemplo: La suma de 3.1.1+3.12+ 3.13 = 3.1
(4.748.480,00 + 1.340.200,00 +70.000,00 = 6.158.680,00)

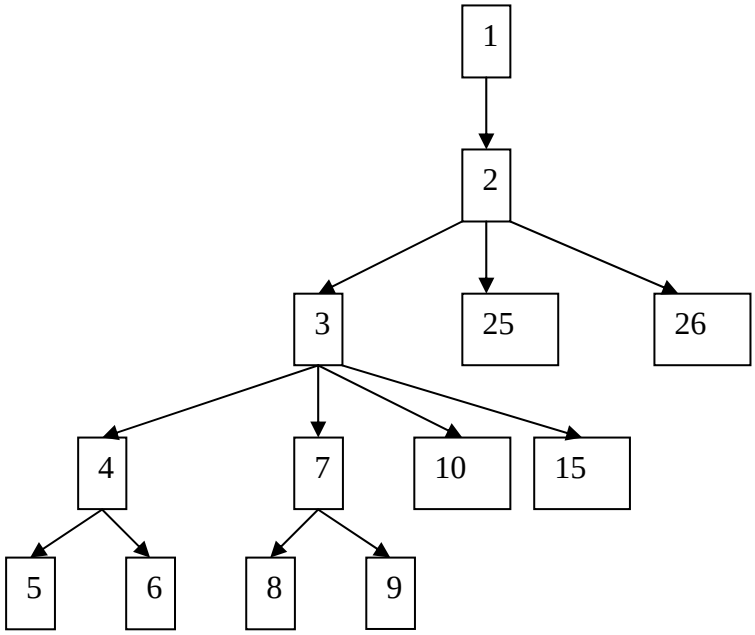
ESTRUCTURA DE ARBOL

PRESUPUESTOS

COD	COD_PADRE	CUENTA	CONCEPTO	MONTO
1	0	3	TOTAL DE EROGACIONES	6.213.680,00
2	1	3.1	EROGACIONES CORRIENTE	6.158.680,00
3	2	3.1.1	PERSONAL	4.748.480,00
4	3	3.1.1.1	SUELDO BASICO	2.402.000,00
5	4	3.1.1.1.6	PLANTA PERMANENTE	919.640,00
6	4	3.1.1.1.7	PLANTA TRANSITORIA	1.482.360,00
7	3	3.1.1.2	ANTIGÜEDAD	362.800,00
8	7	3.1.1.2.5	PLANTA PERMANENTE	362.800,00
9	7	3.1.1.2.6	PLANTA TRANSITORIA	0
10	3	3.1.1.3	BONIFICACIONES Y SUPLEMENTOS	1.499.080,00
….	….	……		
15	3	3.1.1.4	APORTES PATRONALES	484.600,00
…	…	…….		
25	2	3.1.2	BIENES Y SERVICIOS	1.340.200,00
26	2	3.1.3	TRANSFERENCIAS	70.000,00

Aquí decimos: El cod 1 es el primero ya que su padre no existe es 0.
El cod 2 tiene como padre al cod 1 (depende del 1)
El cod 3 tiene como padre al cod 2.

Si sumanos los COD_PADRE iguales, por ejemplo COD_PADRE=2 tenemos (4.748.480,00 + 1.340.200,00 +70.000,00 = 6.158.680,00) donde 6.158.680,00 es justamente el monte del COD=2.



1)
SELECT *
FROM PRESUPUESTO
WHERE COD_PADRE = 2; DEVUELVE REGISTRO 3,25,26

→ MUESTRA LOS REGISTROS CON COD_PADRE = 2

2)
SELECT *
FROM PRESUPUESTO
WHERE COD_PADRE IN (SELECT COD
FROM PRESUPUESTO
WHERE COD_PADRE = 2)

EL SELECT DE MAYOR PROFUNDIDA DEVUELVE 3,25 y 26
EL PRIMER SELECT DEVUELVE REGISTROS 4,7,10 Y 11 (POR COD_PADRE 3)
(25 Y 26 NO TIENEN HIJOS)

1) DEVUELVE REGISTROS 3,25,26 (PROFUNDIDAD 3 DEL ARBOL)
2) DEVUELVE REGISTROS 4,7,10,15 (PROFUNDIDAD 4 DEL ARBOL)

Si uso UNION entre 1) y 2) muestro los registros 3,25,26,4,7,10,15.

Si sigo usando select anidados podré recorrer todo el árbol.
SELECT *
FROM PRESUPUESTO
WHERE COD_PADRE IN (SELECT COD

```
FROM PRESUPUESTO
WHERE COD_PADRE IN ( SELECT COD
                     FROM PRESUPUESTO
                     WHERE COD_PADRE = 2)
```

Muestra los registros del último nivel de profundidad.