BASES DE DATOS

La mayoría de los sistemas computacionales utilizan una base de datos para manejar su información, por lo que es de suma importancia que los desarrolladores de software estén capacitados para su diseño, construcción y uso. Este material tiene como principal objetivo apoyar el curso trimestral introductorio de “Bases de Datos” y puede ser de utilidad a los estudiantes de Ingeniería en Computación, Ingeniería de Software, Tecnologías de la Información y licenciaturas afines.

MÉXICO, D.F. 2013

Dra. María del Carmen Gómez Fuentes

Casa abierta al tiempo

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

9 786074 778809

Notas del curso **BASES DE DATOS**

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA
Casa abierta al tiempo

Dra. María del Carmen Gómez Fuentes

Notas del curso

BASES DE DATOS

UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA
Casa abierta al tiempo

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD CUAJIMALPA

DIVISIÓN DE CIENCIAS NATURALES E INGENIERÍA

MATERIAL DIDÁCTICO

NOTAS DEL CURSO
BASES DE DATOS

AUTORA:
Dra. María del Carmen Gómez Fuentes

Departamento de Matemáticas Aplicadas y Sistemas

IBSN: 978-607-477-880-9

Febrero 2013

Editor
María del Carmen Gómez Fuentes

Departamento de Matemáticas Aplicadas y Sistemas.
División de Ciencias Naturales e Ingeniería
Universidad Autónoma Metropolitana, Unidad Cuajimalpa

Editada por:
UNIVERSIDAD AUTONOMA METROPOLITANA
Prolongación Canal de Miramontes 3855,
Quinto Piso, Col. Ex Hacienda de San Juan de Dios,
Del. Tlalpan, C.P. 14787, México D.F.

NOTAS DEL CURSO: BASES DE DATOS

Primera edición 2013

ISBN: 978-607-477-880-9

Impreso en México
Impreso por Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P.

Contenido.

Objetivos

1.- Conocer las características más importantes de un Sistema Gestor de Bases de Datos (SGBD) y tener la habilidad para plantear modelos de datos que describan problemas reales, así como para implementar dichos modelos usando SGBD relacionales.
2.- Conocer el propósito, ventajas y problemas de la introducción de un SGBD.
3.- Conocer los modelos de los datos y los tipos de bases de datos ubicando al modelo relacional dentro de éstos.
4.- Identificar entidades, atributos y relaciones en un problema específico.
5.- Estudiar técnicas para el diseño de bases de datos relacionales.
6.- Describir las sentencias fundamentales del lenguaje de base de datos SQL para el manejo adecuado de la información.
7.- Aplicar el lenguaje de definición y manipulación de bases de datos SQL en la implementación de modelos relacionales.

Agradecimiento:

Agradezco la generosidad del Dr. Ovidio Peña Rodríguez quien me facilitó su material de clases del trimestre 2007-P de este material utilicé algunos textos y figuras para complementar los capítulos I, III, VIII y IX.

Dra. María del Carmen Gómez Fuentes.

Capítulo I Introducción a las bases de datos.

I.1 Definición de bases de datos y de sistema de administración de bases de datos.

Bases de datos.- El término base de datos surgió en 1963, en la informática una base de datos consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos de datos. En otras palabras, una base de datos no es más que un conjunto de información (un conjunto de datos) relacionada que se encuentra agrupada o estructurada.

I.1.1 Definiciones formales de base de datos.

1.- "Colección de datos, donde los datos están lógicamente relacionados entre sí, tienen una definición y descripción comunes y están estructurados de una forma particular. Una base de datos es también un modelo del mundo real y, como tal, debe poder servir para toda una gama de usos y aplicaciones" [Conference des Statisticiens Européens, 1977].
2.- "Es un conjunto exhaustivo de datos estructurados, fiables y homogéneos, organizados independientemente de su utilización y de su implementación en máquina, accesibles en tiempo real, compartibles por usuarios concurrentes que tienen necesidades de información diferentes y no predecibles en el tiempo" [Access, 2001].

Definición corta de base de datos:
"Es una colección organizada de datos" [Deitel & Deitel, 2008].

Sistema de administración de bases de datos: consiste en un conjunto de programas utilizados para definir, administrar y procesar una base de datos y sus aplicaciones. A los sistemas de administración de bases de datos también se les llama Sistemas de Gestión de Bases de Datos (SGBD). Un sistema de administración de bases de datos es una herramienta de propósito general que permite crear bases de datos de cualquier tamaño y complejidad y con propósitos específicos distintos.

El administrador de una base de datos permite controlar los datos, recuperarlos, ordenarlos, analizarlos, resumirlos y elaborar informes. La base de datos puede combinar datos de varios archivos, por lo que nunca habrá que introducir dos veces la misma información. Incluso puede contribuir a que la entrada de datos sea más eficaz y precisa.

El objetivo principal de un sistema de administración de bases de datos es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente.* Los SGBD se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre varios usuarios, el sistema debe evitar posibles datos contradictorios.

Ejemplo: supongamos que la secretaria de sistemas escolares tiene una lista con los nombres, direcciones de e-mail y teléfono de los alumnos que están en el taller de ajedrez, otra lista de los que están en el taller de teatro, otra lista de los alumnos monitores y otra de los que se van de movilidad en el siguiente trimestre. Si hay uno o varios alumnos muy participativos entonces sus nombres aparecen en varias listas, o incluso en todas. Cuando se da el caso de que uno de este tipo de alumnos cambia su teléfono o su dirección de e-mail, entonces la secretaria tendría que cambiar sus datos en cada una de las listas. Sin embargo, con una base de datos bien estructurada, esto se optimiza y habría que cambiar la dirección en un solo lugar.

I.2 ¿Por qué surgieron las bases de datos?

Antes de las bases de datos se utilizaban los *archivos* para guardar la información, sin embargo, estos presentaban varios problemas [Silberschatz et al., 2002]:

- *Redundancia e inconsistencia de los datos*.- *Redundancia* significa tener el mismo dato guardado varias veces. *Inconsistencia* significa que hay contradicción en el contenido de un mismo dato, es decir, que un mismo dato tiene un valor en una parte de la memoria, mientras que en otra parte contiene otro valor diferente.

- *Dificultad en el acceso a los datos*.- Era difícil que el usuario encontrara rápidamente un dato en especial.

- *No existía el aislamiento de los datos*.- Debido a que los datos estaban dispersos en varios archivos y podían estar en diferentes formatos, era difícil escribir programas nuevos de aplicación para recuperar los datos apropiados.

- *Problemas de integridad*.- Era complicado asegurarse que los valores almacenados satisficieran ciertos tipos de restricciones, por ejemplo, que tuvieran un valor mínimo y/o un valor máximo.

- *Problemas de atomicidad*.- Era muy difícil asegurar que una vez que haya ocurrido alguna falla en el sistema y se ha detectado, los datos se restauraran al estado de consistencia que existía entes de la falla.

- *Anomalías en el acceso concurrente*.- La cuestión de asegurar la consistencia de los datos se complica todavía más cuando se trata de sistemas en los que hay varios usuarios accediendo a un mismo archivo desde diferentes computadoras.

- *Problemas de seguridad*.- No todos los usuarios de un sistema de información deberían poder acceder a todos los datos. En un sistema de archivos es muy difícil garantizar las restricciones de seguridad

Estas dificultades mencionadas, entre otras, motivaron el desarrollo de los sistemas de bases de datos. Los científicos han desarrollado un amplio conjunto de conceptos y técnicas para la gestión de los datos, ya que en la mayoría de las empresas y organizaciones la información es de vital importancia. A continuación se mencionan algunas de las aplicaciones más representativas de las bases de datos:

- *Bancos*.- Para información de los clientes, cuentas, préstamos y transacciones bancarias.
- *Líneas aéreas*.- para reservas e información de planificación.
- *Universidades*.- Para información de los estudiantes, de los profesores y de los cursos.
- *Tarjetas de crédito*.- Para compras con tarjetas de crédito y generación de estados de cuenta.
- *Telecomunicaciones*.- Para llevar registro de las llamadas realizadas, generación mensual de facturas, mantenimiento del saldo de las tarjetas telefónicas de prepago, para almacenar información sobre las redes de comunicaciones.
- *Finanzas*.- Para almacenar información sobre grandes empresas, ventas y compras de documentos financieros como bolsa y bonos.
- *Ventas*.- Para información de clientes, productos y compras.
- *Producción*.- Para la administración de la cadena de producción (inventarios, pedidos, etc.).
- *Recursos humanos*.- Para información sobre los empleados, salarios, impuestos, prestaciones y para la generación de nóminas.

I.3 Características de un sistema de administración de bases de datos.

Las características que definen a un SGBD, [Celma et al., 2003] son las siguientes:

- *Integración de toda la información de la organización*.- La base de datos se crea para dar servicio a toda o a una parte importante de la organización y no para unos usuarios particulares; de esta forma se evita la redundancia de datos dentro del sistema de información y los problemas de inconsistencia derivados de ella.
- *Persistencia de los datos*.- Los datos deben estar disponibles en todo momento, lo que significa que la base de datos debe almacenarse en un dispositivo de memoria secundaria.
- *Accesibilidad simultánea para distintos usuarios*.- Debido al carácter integrador que tiene la base de datos, ésta tendrá que ser compartida por distintos grupos de usuarios, lo que significa que estos podrán acceder simultáneamente a los datos.
- *Independencia de los programas respecto a la representación física de los datos*.- Las aplicaciones que se desarrollen para manipular los datos deben ser independientes de la implementación elegida para las estructuras de la base de datos. A esta característica se le conoce como *independencia de datos.*
- *Definición de vistas parciales de los datos para distintos usuarios*.- Debido también al carácter integrador de la base de datos, en ésta se recogen los datos que interesan a cada grupo de usuarios de la organización, con lo que se incrementa su tamaño y complejidad. Se debe permitir definir vistas parciales de la base de datos que contengan sólo aquellos datos que son relevantes para cada uno de los grupos.
- *Mecanismos para controlar la integridad y la seguridad de los datos*.- Para que la base de datos refleje fielmente la realidad de la cual es una representación, el SGBD bebe asegurar en todo momento la calidad de la información almacenada (*integridad*) evitando que ésta se deteriore por un uso incorrecto (actualizaciones que no son válidas, accesos concurrentes no controlados, etc.). Así mismo, debe asegurar que a la información almacenada sólo acceden las personas autorizadas y en la forma autorizada (*seguridad*).

En resumen: las técnicas de bases de datos se han desarrollado con el objetivo de integrar la información del sistema para evitar redundancias, sin que por ello se pierdan las distintas perspectivas que de ella tienen los usuarios. Además, los SGBD que se construyen para aplicar estas técnicas deben asegurar:

- la *independencia*
- la *integridad* y
- la *seguridad* de los datos

En la *figura 1.1*, se muestra de forma esquemática un sistema de base de datos y sus componentes.

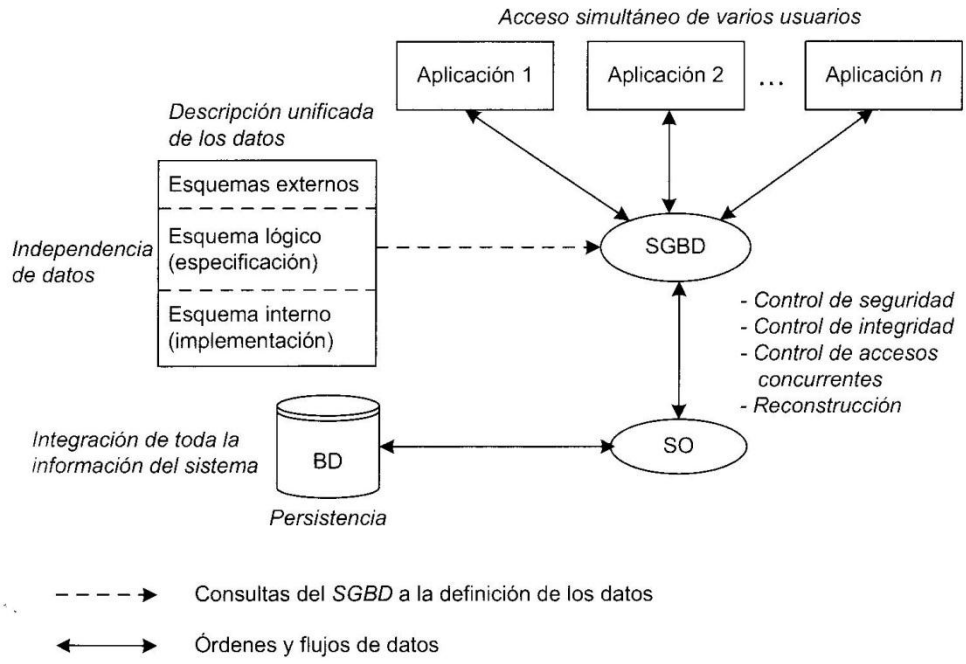


Figura 1.1: Sistema de Base de Datos (Celma et al., 2003)

I.4 Usuarios y administradores de la base de datos.

Las personas que trabajan con una base de datos se pueden catalogar como usuarios de bases de datos o como administradores de bases de datos.

I.4.1 Usuarios de bases de datos e interfaces de usuario.

Podemos distinguir principalmente tres tipos diferentes de usuarios de un sistema de base de datos, en base a la forma en la que interactúan con el sistema, y son los siguientes:

Usuarios normales.- Invocan e interactúan con algún programa de aplicación escrito previamente, por ejemplo, los cajeros de los bancos interactúan con el sistema para realizar las transacciones bancarias que los clientes solicitan en ventanilla. Las personas que trabajan en la inscripción-reinscripción de alumnos en las universidades, es otro ejemplo.

Normalmente la interfaz de usuario que se utiliza en estos casos es a base de formularios, donde el usuario completa los campos apropiados. Los usuarios normales pueden también simplemente leer informes generados de la base de datos.

Programadores de aplicaciones.- Son profesionales que escriben problemas de aplicación. Los programadores de aplicaciones pueden elegir entre muchas aplicaciones para desarrollar interfaces de usuario. Las herramientas de Desarrollo Rápido de Aplicaciones son herramientas que permiten al programador de aplicaciones construir formularios e informes sin escribir un programa. También existen tipos especiales de lenguajes llamados *lenguajes de cuarta generación* que combinan estructuras de control (secuencial, selectivo y cíclico) con instrucciones del lenguaje de manipulación de datos. Estos lenguajes a menudo incluyen características especiales para facilitar la generación de formularios y la presentación de los datos en pantalla. La mayoría de los sistemas de bases de datos comerciales incluyen un lenguaje de cuarta generación.

Los usuarios sofisticados.- Interactúan con el sistema sin programas escritos. En su lugar, ellos realizan sus consultas en un lenguaje de consulta de base de datos. Cada una de estas consultas se envía al procesador de consultas, cuya función es transformar instrucciones LMD (Lenguaje de Manipulación de Datos).

I.4.2 Administrador de bases de datos.

Una de las principales razones de usar un SGBD es tener el control centralizado tanto de los datos como de los programas que tienen acceso a estos datos. La persona que tiene este control central sobre el sistema se llama *administrador de la base de datos*, y sus funciones son las siguientes:

Definir el esquema.- El administrador crea el esquema de la Base de Datos escribiendo un conjunto de instrucciones de definición de datos en un lenguaje especial, llamado Lenguaje de Definición de Datos (DDL: Data Definition Languaje).

Definir la estructura y el método de acceso.- El administrador define como se organiza la información (registros, bloques, estructuras) y define los programas que buscarán la información y harán operaciones sobre ésta.

Modelar el esquema y la organización física.- Los administradores de las bases de datos realizan cambios en el esquema para reflejar las necesidades cambiantes de la organización y realizan cambios en la organización física para mejorar el rendimiento.

Conceden autorizaciones para tener acceso a los datos.- Ellos determinan que partes de la base de datos puede acceder cada usuario. La información de autorización se mantiene en una estructura especial, que el sistema de base de datos consulta cuando se intenta tener acceso a los datos del sistema.

Mantenimiento rutinario. Algunos ejemplos de actividades rutinarias de mantenimiento del administrador de la base de datos son las siguientes:

- Copia de seguridad periódica de la base de datos, ya sea en una cinta magnética o en un servidor remoto, para prevenir la pérdida de datos en caso de desastres como inundaciones, robo o sabotaje.
- Asegurarse de que haya suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en disco según sea necesario.
- Supervisar los trabajos que se ejecutan en la base de datos y asegurarse de que el rendimiento no disminuye por tareas muy costosas iniciadas por algunos usuarios.

I.5 Estructura de un sistema de bases de datos.

Un sistema de bases de datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Las funciones de un sistema de bases de datos se pueden dividir a grandes rasgos en dos componentes:

- Gestor de almacenamiento.
- Procesador de consultas.

El *gestor de almacenamiento* es importante porque las bases de datos requieren normalmente una gran cantidad de espacio de almacenamiento. Las bases de datos de las empresas tienen un tamaño de cientos de gigabytes, y también del orden de terabytes de datos. Un gigabyte son 1,000 megabytes (1,000 millones de bytes), y un terabyte es un millón de megabytes (un billón de bytes). Debido a que la memoria principal de las computadoras no puede almacenar esta gran cantidad de información, ésta se almacena en discos. Los datos se trasladan entre el disco de almacenamiento y la memoria principal cuando es necesario. Como la transferencia de datos a y desde el disco es lenta en comparación con la velocidad del CPU, es fundamental que el sistema de base de datos estructure los datos para minimizar la necesidad de movimiento de datos entre el disco y la memoria principal.

El *gestor de almacenamiento* se divide a su vez en varios módulos, como se indica en la *figura 1.2*. La función principal del gestor de almacenamiento es minimizar el movimiento entre el disco y la memoria principal.

El *procesador de consultas* es importante porque ayuda al sistema de bases de datos a simplificar y facilitar el acceso a los datos. Los componentes del procesador de consultas incluyen:

- El intérprete del Lenguaje de Definición de Datos (DDL)[1]: interpreta las instrucciones del LDD y registra las definiciones en el diccionario de datos.
- Compilador del Lenguaje de Manipulación de Datos (DML)[2]: traduce las instrucciones del LMD en un lenguaje de consultas a instrucciones de bajo nivel que entiende el motor de evaluación de consultas.
- Motor de evaluación de consultas: ejecuta las instrucciones de bajo nivel generadas por el compilador LMD

En la *figura 1.2* también se pueden apreciar los componentes del procesador de consultas.

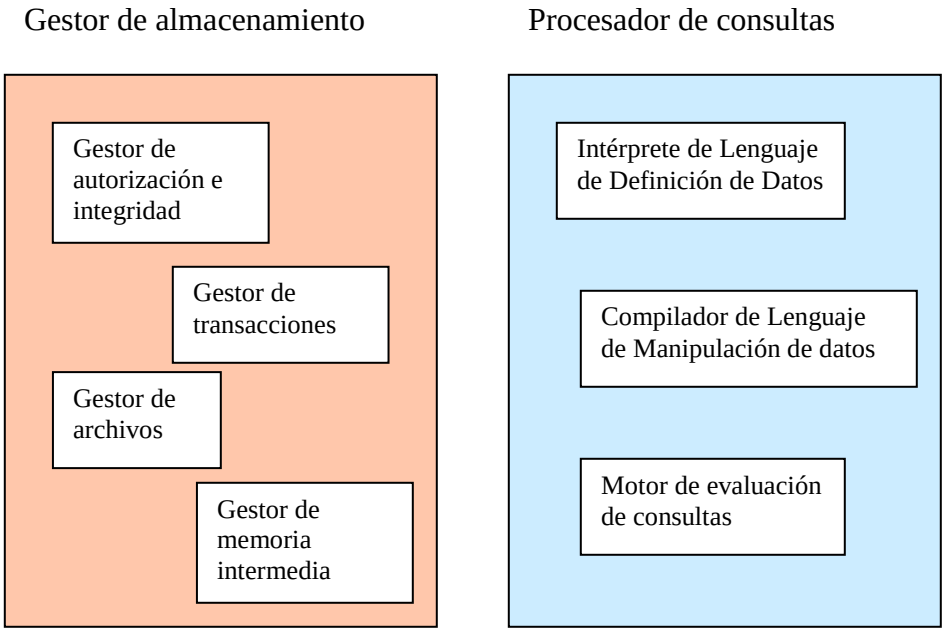


Figura 1.2: Componentes del Gestor de almacenamiento y del procesador de consultas (Material del Curso del Dr. Ovidio Peña Rodríguez).

[1] El Lenguaje de Definición de Datos (LDD) especifica el conjunto de definiciones que forman el *esquema* de una base de datos, es decir, el diseño general de la base de datos

[2] EL Lenguaje de Manipulación de Datos (LMD) es un lenguaje que permite a los usuarios tener acceso a los datos y manipularlos.

Capítulo II Modelos de los datos.

II.1 Abstracción de datos.

Un sistema de bases de datos es una colección de archivos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso y modificar estos archivos. Uno de los propósitos principales de un sistema de bases de datos es proporcionar al usuario una visión *abstracta* de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos.

Para que el sistema sea útil debe recuperar los datos eficientemente, es decir, rápidamente. Con la finalidad de lograr esto, los diseñadores construyen estructuras de datos que a veces son muy complejas. Los desarrolladores esconden la complejidad a los usuarios a través de varios *niveles de abstracción,* esto simplifica la interacción de los usuarios con los sistemas. Los tres niveles de abstracción que se utilizan son los siguientes:

- *Nivel físico*.- Es el nivel más bajo de abstracción y describe como se almacenan realmente los datos, en este nivel se describen con detalle las estructuras complejas de los datos.
- *Nivel lógico*.- Es el siguiente nivel de abstracción, describe que datos se almacenan en la base de datos y que relaciones existen entre esos datos. En este nivel, la base de datos completa se describe en términos de un número pequeño de estructuras relativamente simples. Los que trabajan con este nivel son los administradores de bases de datos, ellos son las personas que deciden que información se mantiene en la base de datos. La implementación de estructuras simples a nivel lógico puede involucrar estructuras complejas a nivel físico.
- *Nivel externo*.- Es el nivel más alto de abstracción, describe solo parte de la base de datos completa, la que el usuario necesita. El sistema puede proporcionar diferentes vistas de una misma base de datos.

Usando una analogía para comprender mejor lo que son los diferentes niveles de abstracción, supongamos que necesitamos un registro que capture los siguientes datos para cada alumno:

Su nombre completo
Su matrícula
Su promedio.

Esto es lo que el usuario ve y con lo que trabaja y sería el equivalente al *nivel externo*. Para implantar esto en un lenguaje de programación, por ejemplo C o C++, podemos declarar una estructura que agrupe toda esta información:

```
struct s_alumno{
  char      nombre[40];
  long int  matricula;
  float     promedio;
};
```

Con esta estructura trabajarían los diseñadores del programa, y sería el equivalente al *nivel lógico* de abstracción. Finalmente, el *nivel físico*, estaría determinado por el orden y la cantidad de bytes que ocupan los tres campos de la estructura s_alumno.

II.1.1 Ejemplares y esquemas.

Las bases de datos van cambiando a lo largo del tiempo conforme la información se inserta y se borra. La colección de información almacenada en la base de datos en un momento particular se denomina un *ejemplar* de la base de datos. El diseño completo de la base de datos se llama el *esquema* de la base de datos. Los esquemas raramente se modifican. Utilizando una vez más la programación para hacer una analogía, un *esquema* de base de datos corresponde a las declaraciones de las variables. Cada variable tiene un valor particular en un instante de tiempo. Los valores de las variables en un programa en un instante de tiempo corresponde a un *ejemplar* de un esquema de bases de datos.

II.2 Definición del modelo de datos.

"Un modelo de datos es una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia" [Silberschatz et al., 2003].

"Un modelo de datos es un conjunto de conceptos y reglas que permiten estructurar los datos resultantes de la observación de la realidad, de forma que queden representadas todas sus propiedades, tanto estáticas como dinámicas" [Celma et al., 2003].

"Un modelo de datos es una notación para describir información acerca de los datos" [García-Molina et al., 2009].

Un modelo de datos es una representación, usualmente gráfica, de estructuras de datos. Los modelos de datos de alto nivel, o conceptuales, utilizan conceptos muy cercanos a la forma en que los usuarios perciben los datos, mientras que Los modelos de bajo nivel o físicos describen detalles de cómo se almacenan los datos en la computadora.

II.3 Clasificación de los modelos de datos.

Los modelos de datos se clasifican en tres grupos:

- Modelos lógicos basados en objetos.
- Modelos lógicos basados en registros.
- Modelos físicos de datos.

Los *modelos lógicos basados en objetos* se usan para describir datos en los niveles conceptuales. Se caracterizan por que proporcionan capacidad de estructuración flexible y permiten especificar restricciones en los datos explícitamente.

Los modelos lógicos basados en objetos más conocidos son:

- Modelo Entidad-Relación.
- Modelo orientado a objetos.
- Modelo binario.
- Modelo semántico de datos. Basado en modelos de redes semánticas con sus raíces en la inteligencia artificial.
- Modelo funcional de datos.

En este curso estudiaremos el modelo Entidad-Relación.

Los *modelos lógicos basados en registros* se utilizan para describir datos en los niveles conceptual y físico, permiten especificar la estructura lógica global de la base de datos y proporcionan una descripción a un nivel más alto en la implantación. Estos modelos se llaman así porque la base de datos está estructurada en registros de formato fijo de varios tipos. Cada registro tiene un número fijo de campos, que su vez son de tamaño fijo. El uso de registros de tamaño fijo facilita la implantación del nivel físico de la base de datos.

Los modelos lógicos basados en registros más conocidos son:

- Modelo relacional.
- Modelo de red.
- Modelo jerárquico.

En este curso estudiaremos el modelo relacional. El modelo de red y el modelo jerárquico son un antecedente del modelo relacional, actualmente se usan muy poco, ya que complican

la tarea del modelado de datos, sin embargo aún están implantados en el código de bases de datos antiguas.

Los *modelos físicos de datos* se usan para describir datos en el nivel más bajo. Son de interés principalmente para los fabricantes de SGBD. Los más conocidos son:

- Modelo unificador.
- Modelo de elementos.

II.4 Tipos de bases de datos.

Existen cuatro tipos diferentes de bases de datos:

- Bases de datos jerárquicas.
- Bases de datos de red.
- Bases de datos relacionales.
- Bases de datos orientadas a objetos.

Las *bases de datos jerárquicas* constituyen el primer modelo lógico de bases de datos que surgió. Es un modelo rígido soportado sobre una estructura de árbol con relaciones exclusivas de padre/hijo, las bases de datos jerárquicas pretenden modelar relaciones jerárquicas del mundo real. Con este tipo de base de datos se obtiene unos excelentes resultados en casos en los que en los modelos donde prevalece el tipo de relación 1:N.

En las bases de datos jerárquicas la representación gráfica se apoya sobre un conjunto de árboles cuyos nodos representan entidades de información y los segmentos de unión representan relaciones 1:N.

El modelo jerárquico utiliza dos conceptos de estructuración: *registros* y *vínculo padre-hijo*. Un *registro* es una colección de valores de campo que proporcionan información sobre una entidad. Un *tipo de vínculo padre-hijo* es un vínculo 1:N entre dos tipos de registros. El tipo de registros del lado 1 se denomina *tipo de registros padre*, y el del lado N se denomina *tipo de registros hijo*.

Un esquema de base de datos jerárquica consiste en varios esquemas jerárquicos. Cada esquema jerárquico consta de varios tipos de registros y varios tipos de vínculo-padre-hijo. En la *figura 2.1* se muestra un ejemplo de esquema jerárquico.

El problema principal del modelo jerárquico es que el mundo real no se adapta fácilmente a este tipo de organización [Marteens, 1999].

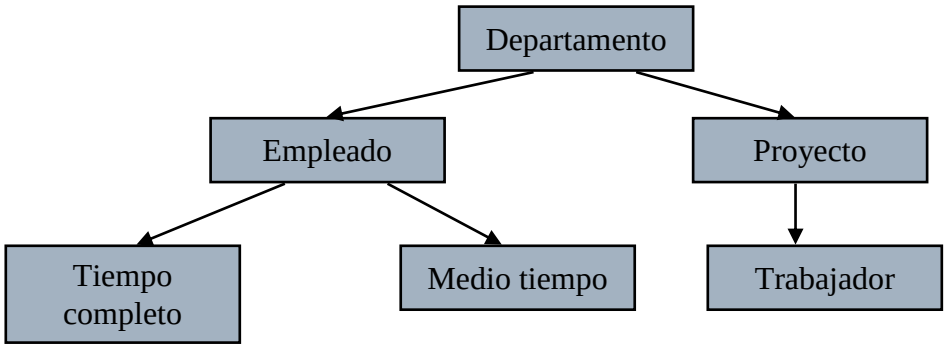


Figura 2.1: Esquema de base de datos de tipo jerárquico (Material del curso del Dr. Ovidio Peña Rodríguez).

Las *bases de datos de red* se basan en dos estructuras básicas: *registros* y *conjuntos*. Cada registro consiste en un grupo de valores de datos relacionados entre sí. Hay diferentes *tipos de registros*, cada uno de los cuáles tiene un nombre. Las relaciones entre los datos se representan mediante enlaces, los cuáles pueden verse como apuntadores. Los registros se organizan como colecciones de grafos arbitrarios.

Un *tipo de conjunto* es un vínculo 1:N entre dos tipos de registros.
Cada definición de tipo de conjuntos consta de 3 elementos:

- Un nombre para el tipo de conjuntos.
- Un tipo de registros propietario.
- Un tipo de registros miembro.

Cada ejemplar del conjunto relaciona un registro de tipo propietario, con el conjunto de registros miembros.

En la *figura 2.2* se muestra un ejemplo de esquema de red.

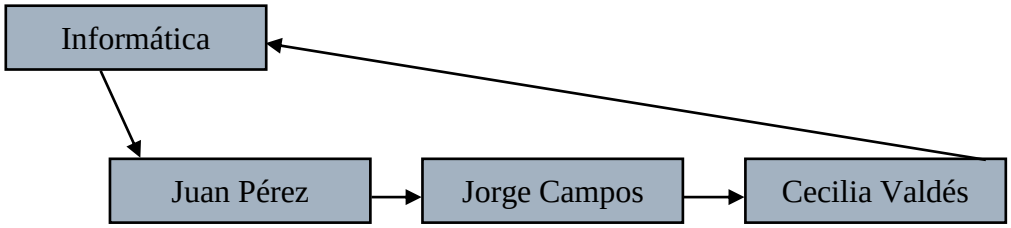


Figura 2.2: Esquema de base de datos de tipo red (Material del curso del Dr. Ovidio Peña Rodríguez).

El modelo en red se desechó debido a dos razones fundamentales, la primera es que para obtener cualquier información era indispensable tener una idea muy clara de cómo estaban

organizados los datos y la segunda razón es que no existían herramientas sencillas que permitieran realizar consultas arbitrarias en una base de datos [Marteens,1999].

Las *bases de datos relacionales* fueron definidas por el matemático Codd en los años 70. La teoría relacional ha ido evolucionando a lo largo del tiempo, incluyendo cada vez nuevas características. Este modelo representa los datos y las relaciones entre los datos mediante una colección de tablas, cada una de las cuáles tiene un número de columnas con nombres únicos. En la sección II.2 de este capítulo, se estudia con más detenimiento las bases de datos relacionales, ya que es un modelo muy utilizado en la actualidad.

El *modelo de datos orientado a objetos* es otro modelo de datos que está recibiendo una atención creciente, éste se puede observar como una extensión del modelo entidad-relación (el cual se estudia en el capítulo III) con los conceptos adicionales de encapsulación, métodos(funciones) e identidad de objetos, que son parte fundamental del diseño orientado a objetos.

Las *bases de datos orientadas a objetos* se propusieron con la idea de satisfacer las necesidades de aplicaciones complejas, como por ejemplo estructuras complejas de datos, transacciones de mayor duración que las tradicionales y accesos a múltiples bases de datos.

Las bases de datos orientadas a objetos permiten al diseñador especificar tanto la estructura de objetos complejos como las operaciones que se pueden aplicar entre los mismos. Una base de datos orientada a objetos provee una *identidad única* a cada objeto independiente almacenado en la base de datos y se parte de la base de que los objetos complejos pueden construirse a partir de otros más simples.

A diferencia de las entidades en el modelo relacional, cada objeto tiene su propia identidad única independiente de los valores que contiene. Así, dos objetos que contienen los mismos valores son, sin embargo, distintos (ver capítulo XII).

Capítulo III Modelo Relacional.

III.1 El modelo relacional.

En el modelo relacional se utiliza un grupo de tablas para representar los datos y las relaciones entre ellos. Cada tabla está compuesta por varias columnas, y cada columna tiene un nombre único. El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se llaman así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo particular. Cada tipo de registro define un número fijo de campos o atributos. Las columnas de la tabla corresponden a los atributos del tipo de registro. El modelo relacional oculta detalles de implementación de bajo nivel a los desarrolladores de bases de datos y a los usuarios.

En las bases de datos *relacionales* los datos se almacenan en distintas tablas por asunto o tarea, pero están relacionados y se pueden combinar de las maneras que se especifique, de tal forma que se puede extraer y unir toda esta información siempre que se desee. Los sistemas relacionales operan conceptualmente sobre *relaciones* o *tablas* de datos y no sobre los datos individuales contenidos en el archivo.

Las relaciones o las tablas permiten representar la información de forma más compacta.

En una base de datos relacional, la información que se encuentra en un conjunto de datos está asociada a la información correspondiente de otro conjunto de datos, la idea es optimizar la manera en la que los usuarios especifican, buscan y generan informes de datos.

A continuación se definen los cuatro objetos básicos de una base de datos:

1. Las relaciones o tablas almacenan los datos en filas y columnas. Todas las bases de datos contienen una o más tablas.
2. Las consultas recuperan y procesan los datos. Pueden combinar datos de distintas tablas, actualizar los datos y realizar cálculos con éstos.
3. Los formularios controlan la entrada de datos y las vistas de datos. Proporcionan indicaciones visuales que simplifican el trabajo con los datos.

4. Los informes resumen e imprimen los datos. Convierten los datos de las tablas y consultas en documentos que comunican ideas.

Una base de datos relacional se compone de tablas independientes que están asociadas por medio de relaciones. Es posible acceder a la información contenida en dos o más tablas simultáneamente.

Antes de comenzar con la siguiente sección, es importante mencionar que existen tres diferentes terminologías dentro del modelo relacional, en la *tabla 3.1* se presenta la equivalencia entre estas terminologías. Dejaremos opcional al lector los términos a utilizar, ya que lo importante es que se entienda el concepto y que se use de la manera adecuada.

Relación	Tabla	Fichero
Tupla	Fila	Registro
Atributo	Columna	Campo
Grado	No. de columnas	No. de Campos
Cardinalidad	No. de filas	No. de registros

Tabla 3.1: comparación de la terminología usada en el modelo relacional (De Miguel et al., 2004).

III.1.1 Relaciones (Tablas).

Las relaciones, también llamadas tablas, son los pilares esenciales de cualquier base de datos, ya que almacenan los datos.

"*Las relaciones* ***vinculan los datos de las distintas tablas*** *para que sean más útiles".*

Una base de datos debería tener una relación distinta para cada asunto principal, como registros de empleados, pedidos de clientes, métodos de entrega o proveedores. No deben duplicarse los datos en varias relaciones. Esto es un error común fácil de evitar si se estructuran bien las tablas.

Como se muestra en la *figura 3.1* cada tabla o relación contiene filas, denominadas *registros* o *tuplas* y columnas, denominadas *campos* o *atributos*.

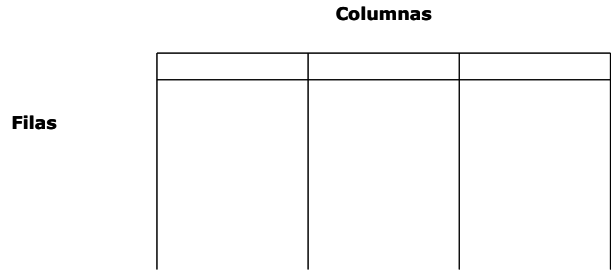


Figura 3.1: Estructura de una relación.

*Registros (tuplas)***.-** una tupla o registro es un conjunto de hechos acerca de una persona, de un evento o de cualquier otro elemento de interés. Por ejemplo el alumno Gulmaro Pérez y su matrícula, promedio, fecha de ingreso, etc. Cada tupla o registro contiene los valores que toma cada uno de los campos de un elemento de la tabla.

En una base de datos bien estructurada, cada tupla o registro debe ser único. Es decir, no deben existir dos o más registros que contengan exactamente la misma información.

*Campos (atributos)***.-** un campo o atributo es una sola clase de hecho que se puede aplicar a cada persona, evento o registro. Por ejemplo, Código postal puede ser un campo de una tabla de Empleados. Teléfono puede ser un campo de una tabla de Transportistas. Matrícula puede ser un campo de una tabla de Alumnos.

Los campos de la base de datos tienen *valores* que determinan el tipo de datos que pueden almacenar, cómo se muestran los datos y qué se puede hacer con ellos.

Por ejemplo, se puede utilizar los valores de campo para asegurar que todos los usuarios especifican la fecha con un formato específico, digamos: con dos números para el mes, dos números para el día, cuatro números para el año, separados por barras diagonales: 03/09/2008.

Un valor importante para los campos es el *tipo de datos*, que puede ser número, texto, moneda (dinero), fecha, hora, etc. El tipo de datos limita y describe la clase de información del campo. También determina las acciones que se pueden realizar en el campo y la cantidad de memoria que utilizan los datos.

Los campos también tienen *propiedades* que controlan los detalles de la información que contienen, incluida la longitud de caracteres, un valor predeterminado y una regla de validación que comprueba que los datos satisfacen ciertos criterios. Las propiedades simplifican la entrada y administración de los datos.

Un *valor nulo* es distinto a la cadena de caracteres vacía o en blanco y distinto de cero o cualquier otro número. Se utilizan en las bases de datos relacionales para representar la falta de información y/o la información que no aplica.

III.1.1.1 Tipos de claves.

Varias definiciones de *clave.*

1.- Una clave es un atributo o conjunto de atributos cuyos valores distinguen a una tupla en una tabla.

2.- Una clave es el conjunto mínimo de atributos cuyos valores le dan una identificación única a la tupla en la relación.

3.- Una clave es una referencia que se utiliza para identificar los registros de forma única y está formada por uno o más campos de los registros.

*Clave primaria o principal***.-** Es un identificador único para cada registro. No puede contener entradas nulas. Para cada tupla de una relación se utiliza un identificador único, denominado clave primaria o clave principal.

Para elegir un campo de una tabla como clave primaria debe cumplir con las siguientes características

- Deberá seleccionarse la que ocupe un menor espacio de almacenamiento.
- Tener una codificación sencilla.
- El contenido de sus valores deben ser conocido y no variar.
- No debe tener valores nulos.
- Podrá utilizarse en otras tablas para construir interrelaciónes.
- Deben ser fácilmente recordables por el usuario.

Por ejemplo, si se tiene una tabla con 2500 alumnos, podríamos poner su *matrícula* como clave principal, si se tiene una tabla con 20 millones de ciudadanos, su *CURP* podría ser la clave principal. Esto significa que si proporcionamos la clave principal la base de datos puede encontrar al registro que contenga la clave proporcionada. Cada alumno tiene una matrícula única, así como cada ciudadano tiene un CURP único, diferente del de todos los demás.

La clave principal debe ser una información que no cambie con frecuencia. Esta clave es importante porque permite acceder a cada uno de los elementos de la Base de Datos mediante la combinación de tres factores:

- El nombre de la Tabla.
- La columna (campo)
- El valor de la clave.

Todas las tablas relacionales deben tener definida una clave principal.

En las *figuras 3.2* y *3.3* se muestran ejemplos de tablas (relaciones).

Atributos

	COD_BAN	BANCO	TELEFONO
	34534	HSBC	5635 6789
Tuplas	56567	Bancomer	5667 8901
	45645	Banamex	5623 4567

Figura 3.2: Tuplas y atributos (campos) de una tabla.

Clave Primaria				
Matrícula	**Nombre**	**Apellido**	**Teléfono**	Nombre Campos
2341167	Pedro	Alonso	5673652	Registro
5625711	María	Gómez	5518234	Registro
6718273	José	López	5615629	Registro
Campo	Campo	Campo	Campo	

Figura 3.3: Tabla con el campo *matrícula* como clave primaria.

Clave secundaria.- Es un atributo (o combinación de atributos) que se usa estrictamente para propósitos de recuperación de información. También se le llama *Clave Alternativa*.

Los atributos que pertenecen a la clave primaria se denominan atributos primarios, los atributos restantes se llaman atributos no primarios o secundarios.

Cuando una clave está formada por un solo atributo se denomina *clave simple*, en caso contrario se denomina *clave compuesta* o concatenada.

Clave externa.- Es la clave primaria de una tabla que se utiliza en otras tablas para crear una interrelación.

Ejemplos de claves de una entidad:

Entidad: *Alumnos de un centro universitario:*
Número de matrícula
Clave Única de Registro de Población
Nombre y apellidos
Fecha de nacimiento
Domicilio
Código postal
Teléfono

- Claves candidatas:
  - Número de matrícula
  - Clave Única de Registro de Población
  - Nombre y apellidos+Fecha de nacimiento+teléfono
- Clave primaria: Número de matrícula
- Claves alternativas:
  - Clave Única de Registro de Población
  - Nombre y apellidos+Fecha de nacimiento+teléfono

III.2 Características de las bases de datos relacionales.

La estructura de una relación puede implementarse de maneras muy diferentes: archivos indexados, archivos invertidos, archivos con direccionamiento calculado por dispersión (hashing), etc. Por esto, no existe un orden entre las tuplas de una relación. La única forma de seleccionar una tupla es especificando el valor de algunos de sus atributos. Los pares de una tupla no están ordenados entre sí, ya que la tupla es también un conjunto, de manera que la única forma de hacer referencia y acceder a un componente de la tupla es mediante el nombre del atributo correspondiente.

Podemos decir que una base de datos relacional tiene las siguientes características:

- La base de datos esta compuesta generalmente de muchas Tablas (Relaciones).
- Cada Tabla contiene un número fijo de Campos (Columnas).
- El nombre de los Campos que componen una Tabla debe ser distinto.
- Cada Registro (Tupla) de una Tabla es único. Es decir, no existen tuplas repetidas en una relación.
- El orden de los Registros y el orden de los Campos de una Tabla no está determinado.
- Para cada Campo existe un conjunto de valores posibles llamado: *Dominio*.

III.3 Manejo de las bases de datos relacionales.

III.3.1 Relacionando tablas: "Interrelaciones".

Una clave principal separa información similar y hace que cada registro sea único, pero también asocia información. Para relacionar dos tablas se utiliza una clave principal. De esta forma las tablas comparten datos sin que se repita la información en ambas.

Las claves principales permiten explotar la eficacia de una base de datos relacional en lugar de trabajar con numerosas listas repetitivas que son difíciles de mantener y no se relacionan entre sí.

Cuando las tablas están relacionadas, la clave principal de una tabla pasa a ser una *clave externa* de la otra tabla.

Los datos deben organizarse en tablas según los asuntos asociados a los datos. Una base de datos bien estructurada tiene una tabla para cada asunto al que pertenecen los datos, como Empleados, Estudiantes o Productos.

Una *clave principal* es un identificador único que distingue un registro de otro y vincula los datos de una tabla a los datos de otras tablas.

Una *interrelación* es una asociación entre tablas que se establece mediante la clave externa de una tabla (tabla hija o tabla dependiente) y la clave principal de la otra (tabla padre o tabla maestra). La *clave externa* es un atributo o conjunto de atributos de la tabla dependiente cuyos valores se corresponden con la clave principal de la tabla maestra. Los dominios de ambas claves deben ser compatibles.

Por ejemplo, si tenemos una tabla de Profesores y una tabla de Grupos. El "número de empleado" es la *clave principal* de la tabla maestra Profesores y es una *clave externa* de la tabla dependiente Grupos. La tabla Grupos tiene su propia clave principal que es la clave de la UEA. Cuando se le pide al profesor Nicodemo Sánchez que imparta una UEA, su número de empleado se introduce en la tabla Grupos, este número de empleado hace referencia a los detalles de Nicodemo en la tabla Profesores, por lo que no es necesario repetir los datos de Nicodemo, en este caso el departamento al que pertenece y su teléfono, en la tabla Grupos.

Nótese en la *figura 3.4* que el número de empleado de Nicodemo puede estar varias veces en la tabla Grupos en el caso de que él sea profesor de varias UEAs.

GRUPOS (tabla dependiente)

CLAVE_UEA	PROFESOR	NUM_EMPL	AULA
15029	Eleazar Martinez	3028	A904
1532	Nicodemo López	3172	A324
12345	Heriberto Jiménez	3075	A217
15112	Maricela Lopez	3194	A220
12463	Nicodemo Sanchez	3172	A912
13947	Moises Caceres	2907	A323

Clave principal

Clave externa

PROFESORES (tabla maestra)

NUM_EMPL	NOMBRE	DEPTO	TELEFONO
2907	Moises Caceres	Quimica	21211304
2962	Arnulfo Díaz	matematicas	91801423
3075	Heriberto Jiménez	matematicas	56754321
3028	Eleazar Martinez	Fisica	57221314
3150	Carlos Pérez	Fisica	55603019
3172	Nicodemo Sanchez	matematicas	53279872
3194	Maricela Lopez	Quimica	58961212

Figura 3.4: Claves en las Tablas.

III.3.2 Consultas, formularios, informes impresos.

Consultas.- Con las consultas se puede obtener información mediante la asociación de los datos almacenados en la base de datos o mediante la realización de cálculos con los datos para proporcionar más información.

Para obtener cierta información, las consultas recuperan, filtran, ordenan y asocian los datos a petición. Otra característica importante de las consultas es que pueden combinar los datos de varias tablas en una sola vista.

Cuando una consulta encuentra datos y los muestra, también puede procesarlos según sus instrucciones. Una consulta puede realizar cálculos con los datos, por ejemplo, obtener el promedio de todas las calificaciones de las asignaturas de un trimestre.

Una consulta también puede quitar datos: eliminar los nombres de los alumnos que no aprobaron el curso.

Utilizando el "Álgebra relacional" que estudiaremos posteriormente veremos con detalle el poder que tienen las consultas dentro de una base de datos.

Formularios.- Los formularios permiten introducir o ver datos en la base de datos fácilmente. Los formularios se pueden considerar ventanas a través de las cuáles las personas pueden trabajar con los datos.

Los formularios controlan y simplifican la entrada de datos. Cuando se introducen datos en un formulario, éstos se guardan en una tabla adicional.

Los formularios hacen que los datos de una tabla o consulta sean más fáciles de comprender, ya que se presentan en diseños visualmente llamativos. Proporcionan listas desplegables, instrucciones, controles de desplazamiento y gráficos que ayudan a los usuarios a trabajar con los datos. De un modo u otro, los formularios hacen que esta tarea sea más agradable.

En algunos casos resulta más práctico utilizar un formulario en lugar de una tabla porque éste puede proporcionar texto con instrucciones, además de gráficos y controles para simplificar la entrada o visualización de los datos.

Informes impresos.- Los informes convierten los datos en documentos. Los informes pueden tener distintas formas y tamaños, pero todos ellos están diseñados para presentar los datos de forma impresa. Los informes proporcionan los medios para dar el mejor formato posible para un fin dado a la apariencia de los datos impresos.

Por medio de los informes se pueden agrupar los datos, realizar cálculos con ellos, y agregar titulares y otros elementos de formato para hacer que tengan más significado y sean más fáciles de leer.

Lo práctico de un informe es que, una vez que se crea, se puede guardar su formato de manera que tenga la misma apariencia cada vez que lo imprima, a pesar de que los datos cambien.

III.4 El concepto de valor nulo en el modelo relacional.

En muchas ocasiones trabajar con dos valores (prendido-apagado, si-no, verdadero-falso, etc.) es suficiente, sin embargo, existen áreas en donde son necesarios tres valores (prendido-apagado-desconectado, si-no-indeterminado, verdadero-falso-no especificado), en el caso del modelo relacional, se introduce el concepto de *valor nulo*, que también se denomina *valor ausente*, para indicar un tercer estado que representa la información: desconocida, inaplicable, inexistente, no válida, no proporcionada, indefinida, etc.

Para el tratamiento de los valores nulos hay que definir:

- Operaciones de comparación.
- Operaciones aritméticas.
- Operaciones algebraicas.
- Funciones de agregación.

En esta sección nos concentraremos en las funciones de comparación, aquí surge la *lógica trivaluada*, en donde, se definen tres valores:

V: Verdadero
F: Falso
Q: Quizás

A continuación, en las *Tablas 3.2, 3.3 y 3.4* se muestran las tablas de verdad para las operaciones NOT, AND y OR de la lógica trivaluada:

	NOT
V	F
F	V
Q	Q

Tabla 3.2: Tabla de verdad de la operación NOT.

		AND
F	F	F
F	V	F
V	F	F
F	Q	F
Q	F	F
V	Q	Q
Q	V	Q
V	V	V
Q	Q	Q

Tabla 3.3: Tabla de verdad de la operación AND.

		OR
F	F	F
F	V	V
V	F	V
F	Q	Q
Q	F	Q
V	Q	V
Q	V	V
V	V	V
Q	Q	Q

Tabla 3.4: Tabla de verdad de la operación OR.

Se define el operador:

IS_NULL (es_nulo)

Este operador toma el valor de "verdadero" si el operando es nulo y "falso" en caso contrario.

En cuanto a las operaciones aritméticas con valores nulos, se considera nulo el resultado de sumar, restar, multiplicar o dividir cuando alguno de los operandos toma el valor nulo. También los valores nulos afectan en algunas operaciones algebraicas.

III.5 Modelo entidad-relación.

El modelo entidad relación, también llamado modelo conceptual de datos, es un modelo semántico que sirve para describir y construir el Esquema Conceptual de una Base de Datos, sirve para modelar un almacenamiento de datos, es una técnica especial de

representación gráfica que incorpora información relativa a los datos y la relación existente entre ellos para dar una visión del mundo real.

En muchos casos, los datos manipulados por el sistema determinan las acciones que se realizan. Puede ser útil definir los requerimientos concentrándose en los datos en lugar de las funciones. La *abstracción de datos* es una técnica para describir para qué son los datos, en lugar de cuál es su apariencia o como se denominan.

El modelo entidad-relación consta de construcciones y convenciones que se utilizan para crear un modelo de datos del usuario. Las cosas en el mundo del usuario están representadas por entidades, y las asociaciones entre éstas están representadas mediante relaciones. Usualmente, los resultados se documentan en un *diagrama entidad-relación*.

Para describir los datos se utiliza el diccionario de tipo de datos. La idea central es categorizar los datos y agrupar los elementos semejantes.

El Modelo Entidad /Relación fue desarrollado por Chen en 1976. Es un modelo muy utilizado en el campo de diseños de bases de datos. Su principal ventaja es que es traducible casi automáticamente a un esquema de bases de datos bajo modelo relacional. Todas las metodologías de diseño de sistemas incorporan esta técnica para el modelado de datos.

Definición formal: "El modelo entidad –relación es una técnica semántica de modelado gráfico de datos basada en la percepción del mundo real como un conjunto de objetos básicos llamados entidades y las relaciones existentes entre ellas".

III.5.1 Características del Modelo Entidad-Relación.

Las características más importantes del modelo Entidad-Relación se pueden enlistar de la siguiente forma:

- Refleja sólo la existencia de los datos, no lo que se vaya a hacer con ellos.
- Se incluyen todos los datos del sistema en estudio, no le afectan las aplicaciones concretas.
- Es independiente de la base de datos a utilizar y del sistema operativo.
- No se tienen en cuenta restricciones de espacio, almacenamiento, ni tiempo de ejecución.
- Está siempre abierto a la evolución del sistema.
- Se basa en la percepción de que el mundo real consiste de una colección de objetos básicos llamados entidades y relaciones entre estos objetos.
- El modelo E-R describe los datos como Entidades, relaciones y atributos.

III.5.2 Elementos del Modelo Entidad-Relación (MER).

El MER tiene sus propias estructuras que son los Diagramas Entidad-Relación (DER).

Entidades: Una entidad es un objeto real o abstracto de interés en una organización acerca del cual se puede y se quiere guardar información, puede ser una persona, un lugar, un concepto o un evento. Por ejemplo, las entidades de la *figura 3.5*:

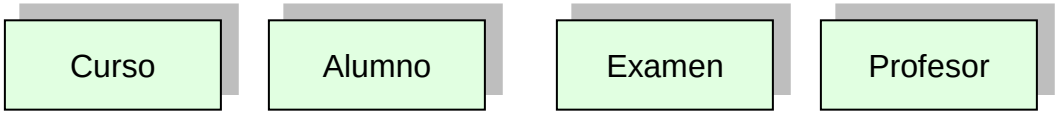


Figura 3.5: Ejemplo de entidades en un modelo E-R.

Asociado al concepto de entidad surge el de *ocurrencia de entidad* que es una realización concreta de la misma. Por ejemplo, si las *entidades* son libro, editorial, autor, documento. Las *ocurrencias para la entidad* editorial serían: McGraw-Hill, Addison-Wesley, Alfaomega.

Una entidad es un objeto que se distingue de otros por medio de un conjunto de específico de propiedades llamadas *atributos.*

Atributos: Un atributo es una propiedad o característica asociada a una entidad y común a todas las ocurrencias de la misma, como se ilustra en la *figuras 3.6*.

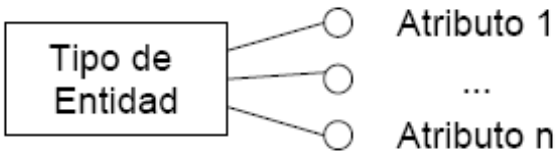


Figura 3.6: Tipo de Entidad y sus Atributos.

Por ejemplo, para la entidad Alumno se pueden tener los atributos: nombre, grupo y calificación, o para la entidad Curso se pueden tener los atributos: unidad, nombre UEA, Carrera, como se indica en el ejemplo de la *figura 3.7*.

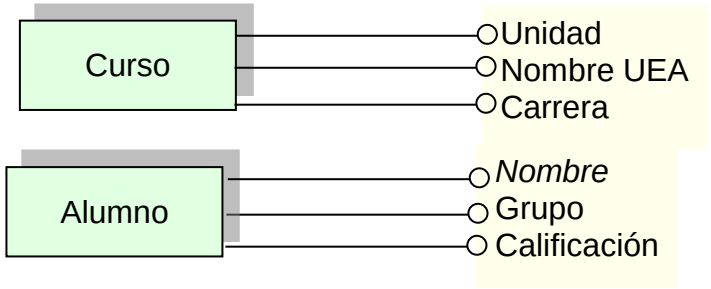


Figura 3.7: Ejemplo de tipo de Entidad y sus Atributos.

Asociado al concepto de Atributo surge el concepto de *dominio*. *Dominio* es el conjunto de valores permitidos para un atributo. En la *figura 3.8* se muestra un ejemplo del dominio del dominio del atributo "Unidad".

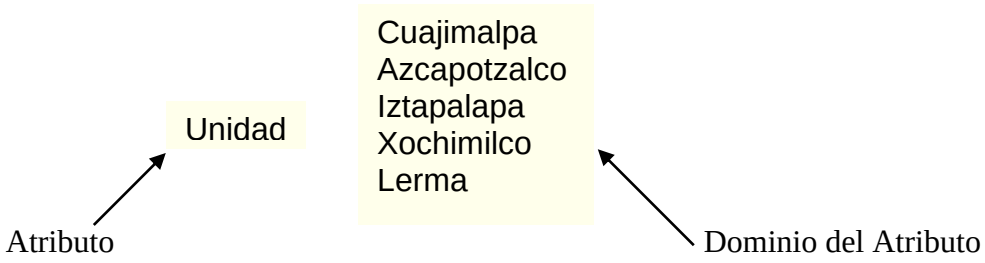


Figura 3.8: Ejemplo de tipo de Atributo y Dominio del Atributo.

Existen 2 tipos de Atributos:

- *Atributo* ***identificador***: Distingue una ocurrencia de entidad del resto de ocurrencias. Por ejemplo *nombre* del alumno.
- *Atributo* ***descriptor***: Caracteriza una ocurrencia pero no la distingue del resto de ocurrencias de la entidad. Por ejemplo *grupo* y *calificación* del alumno.

Debido a que no existe un estándar para la representación de los diagramas Entidad-Relación, existen variaciones en la manera de representar las entidades con sus atributos. La manera más utilizada para representar los atributos es por medio de elipses, como se muestra en la *figura 3.9*.

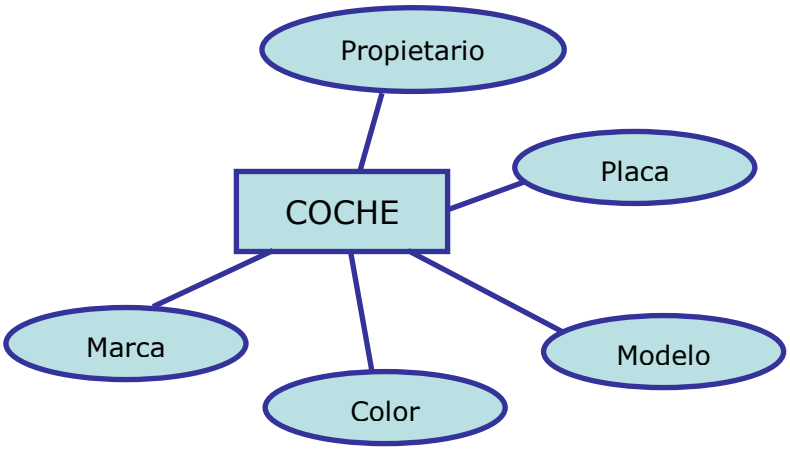


Figura 3.9: Representación de atributos mediante elipses (Material del curso del Dr. Ovidio Peña Rodríguez).

Relaciones: Una relación es una asociación entre entidades. Entre dos entidades puede existir más de un tipo de relación. Un *objeto asociativo* es un elemento que sirve para relacionar objetos. Para que exista una instancia del mismo, deben existir instancias de todos los objetos que relaciona. Asociado al concepto de Relación surge el concepto de *ocurrencia de relación* que es la asociación concreta de ocurrencias de entidad de las diferentes entidades. . Por ejemplo si tenemos las relaciones: "Autor *escribe* Documento" o "Editorial *edita* Libro" (ver *Figura 3.11*) una ocurrencia puede ser: Shakespeare *escribe* "Sueño de una noche de verano", un ejemplo de ocurrencia para la otra relación: Prentice Hall *edita* "Bases de datos relacionales".

En cuanto a las relaciones, hay dos conceptos importantes:

- Conjunto de relaciones.- es la agrupación de todas las relaciones existentes en un conjunto de entidades.
- Dimensión de una relación.- es el número de entidades que participa en ella.

III.5.3 Diagramas entidad-relación.

Para visualizar gráficamente los modelos entidad-relación, se utilizan los *diagramas entidad-relación* en los que el tipo de interrelaciones entre las entidades se representa con un rombo y las entidades con un rectángulo, como se muestra en la *figura 3.10.*

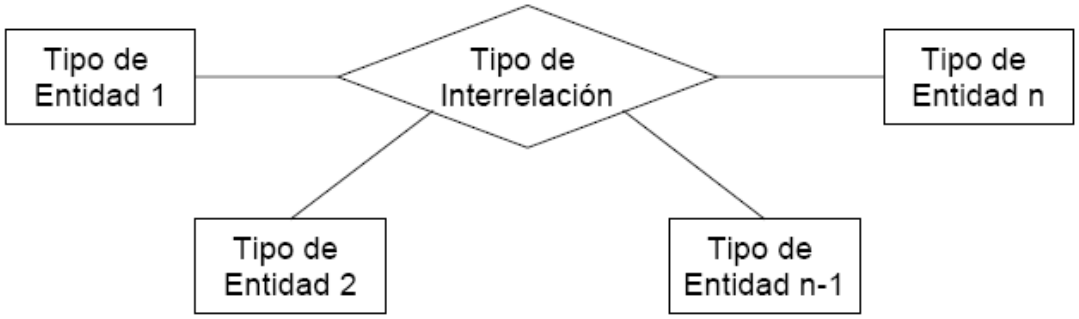


Figura 3.10.: Tipo de Interrelación y Tipos de Entidad Relacionadas
(Material del curso del Dr. Ovidio Peña Rodríguez).

La estructura lógica general de una base de datos se puede expresar gráficamente mediante los diagramas entidad-relación, cuyos los componentes son los siguientes:

- *Rectángulos.*- representan conjuntos de entidades.
- *Elipses*.- representan atributos.
- *Rombos*.- representan relaciones entre conjuntos de entidades.
- *Líneas***.-** unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

Cada componente se etiqueta con la entidad o relación que representa.

En la *figura 3.11* podemos observar dos ejemplos de diagramas muy sencillos de diagramas entidad-relación.

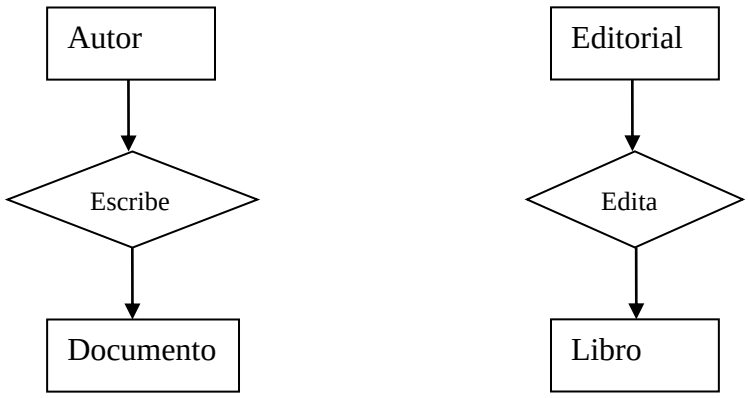


Figura 3.11: Ejemplos sencillos de diagramas entidad-relación.

En la *figura 3.12* tenemos otro ejemplo de diagrama entidad –relación en donde se muestran los atributos de las entidades. Este diagrama pertenece a una parte de un sistema bancario, en él se indica que hay dos conjuntos de entidades: *cliente* y *cuenta*. Cada entidad tiene los atributos mencionados en el diagrama, y están relacionadas con la relación *asigna.*

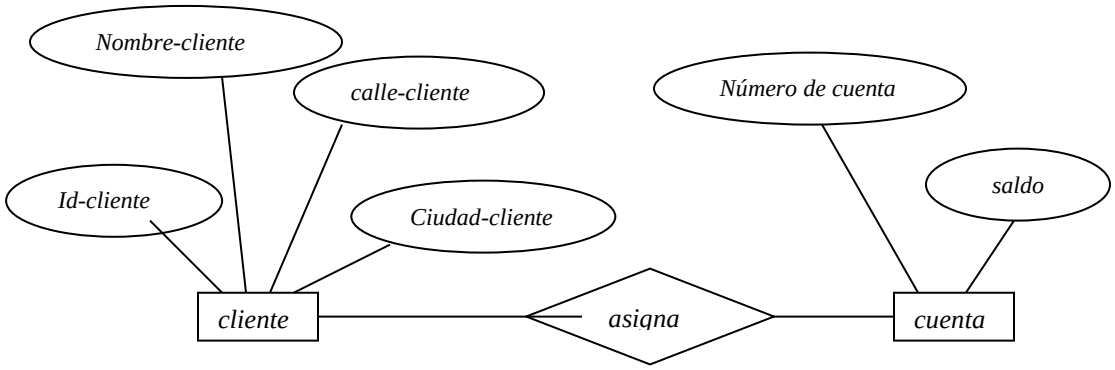


Figura 3.12: Ejemplo de diagrama entidad-relación con atributos (Silberschatz, 2003)

Las relaciones se expresan mediante un verbo, procurando así formar frases que expresan un proceso de gestión, considerando que las entidades son sustantivos que actúan como sujeto y complemento cuando se asocian. En la *figura 3.13* se muestra la relación sujeto-verbo-complemento, en este ejemplo, la ocurrencia de la relación: "Juan García estudia programación" en sentido inverso se lee "Programación es estudiada por Juan García".

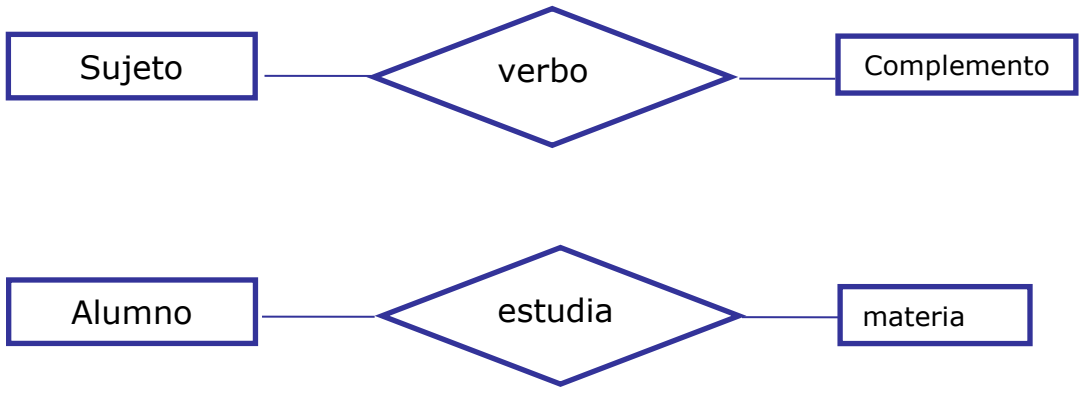


Figura 3.13: Relación sujeto-verbo-complemento.

III.5.4 Cardinalidades de un tipo de entidad.

Una restricción importante es la *cardinalidad de asignación*, también llamada *correspondencia de cardinalidades,* esta expresa el número de entidades con la que otra entidad se puede asociar a través de un conjunto de relaciones. Por ejemplo, si a cada *cuenta* puede pertenecer sólo un *cliente*, el modelo puede expresar esta restricción.

La cardinalidad indica la participación de las entidades asociadas en una relación.

La cardinalidad se representa etiquetando las líneas que unen las entidades con las relaciones en el diagrama entidad-relación. Las cardinalidades máxima y mínima de una entidad participante en un modelo entidad-relación, se definen como "el número máximo y mínimo de ocurrencias de un tipo de entidad que pueden estar interrelacionadas con una ocurrencia del otro, u otros tipo s de entidad que participan en el tipo de interrelación" [Piattini et al., 1998].

Cardinalidad 1:1 (Una a una).-A cada ocurrencia de la primera entidad le corresponde una y solo una ocurrencia de la segunda y viceversa. Ver el ejemplo de la *figura 3.14*.

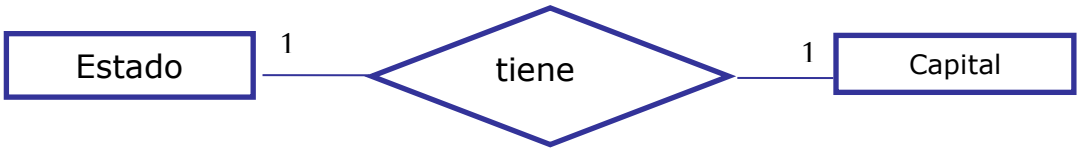


Figura 3.14: Relación 1:1.

Cardinalidad 1:N (Una a muchas).-A cada ocurrencia de la primera entidad pueden corresponderle más de una ocurrencia de la segunda y a cada ocurrencia de la segunda le corresponde no más de una de la primera. Ver el ejemplo de la *figura 3.15*.

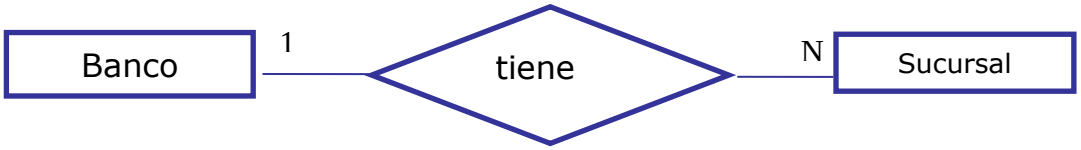


Figura 3.15: Relación 1:N.

Cardinalidad N:N (Muchas a muchas).- A cada ocurrencia de la primera entidad pueden corresponderle más de una ocurrencia de la segunda y viceversa. Ver el ejemplo de la *figura 3.16*.

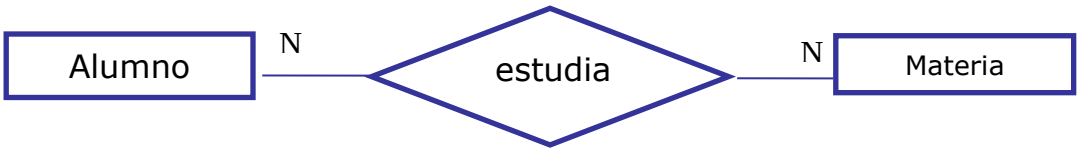


Figura 3.16: Relación N:N.

III.5.5 Clasificación de las entidades del modelo entidad-relación.

Se dice que hay una *dependencia de la existencia* cuando la ocurrencia de una entidad dependiente no puede existir sin la ocurrencia de una entidad de la que depende. En base a lo anterior, se distinguen dos tipos fundamentales de entidades:

- *Entidades fuertes*.- son aquellas cuyas ocurrencias son identificables por si mismas. Los atributos que las identifican son propios de la entidad. Se representan mediante un rectángulo, con el nombre en el interior.
- *Entidades débiles*.- son aquellas cuyas ocurrencias son identificables solamente por estar asociadas a otra u otras entidades. Alguno de los atributos que la identifican está referido a otra entidad. Se representan mediante dos rectángulos inscritos con el nombre de la entidad en el interior.

En la *figura 3.17* se muestran ejemplos de entidades fuertes y débiles.

Entidades fuertes.

Entidades débiles

Figura 3.17: Ejemplos de entidades fuertes y débiles.

En la *figura 3.18* se muestra un ejemplo de cómo se relacionan las entidades fuertes con las entidades débiles:

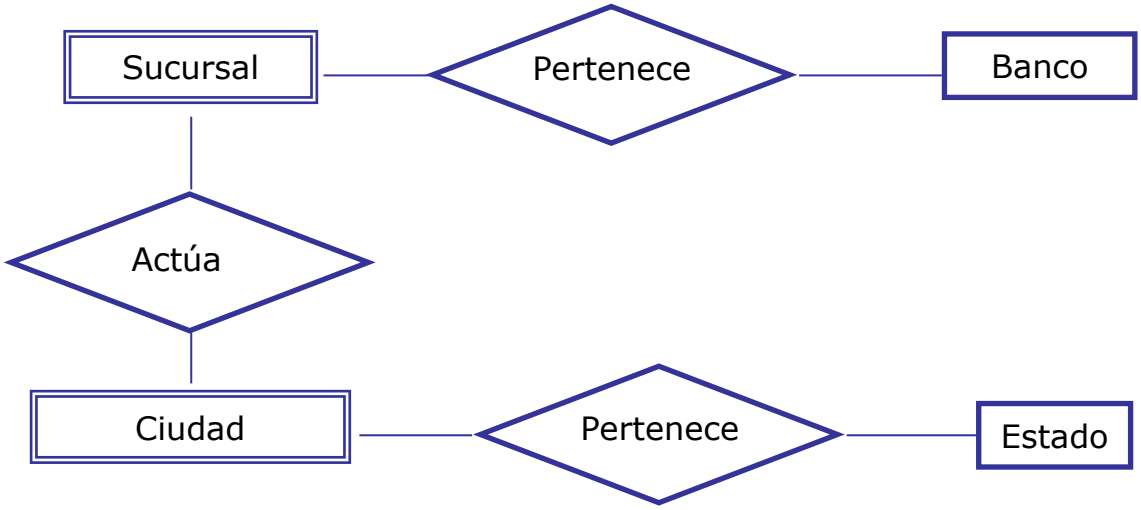


Figura 3.18: Ejemplos de relación entre entidades fuertes y débiles.

Capítulo IV El Lenguaje de Base de Datos SQL.

IV.1 Introducción al SQL.

La característica relacional más importante de SQL es que permite acceder a los datos sin necesidad de especificar cómo se ha de realizar dicho acceso permitiendo así la "navegación automática" por los datos. SQL está diseñado para responder a preguntas del tipo *Qué quiero hacer?* al contrario que los otros lenguajes de programación, como Pascal, Java y C++, que intentan resolver cuestiones del tipo *Cómo lo hago?*

SQL usa los términos *tabla*, *fila* y *columna* para *relación*, *tupla* y *atributo*, respectivamente.

SQL significa Lenguaje Estructurado de Consultas (*Structured Query Languaje*) y se ha establecido Como el lenguaje estándar de bases de datos relacionales, esto significa que su uso esta generalizado a nivel internacional. El lenguaje SQL tiene varios componentes, los básicos son:

- El *lenguaje de definición de datos* para especificar el esquema de la base de datos.
- El *lenguaje de manipulación de datos* para expresar las consultas a la base de datos y las modificaciones.
- La definición de vistas,
- El control de transacciones.
- Integridad.- órdenes para especificar las restricciones de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las restricciones de integridad se rechazan.
- Autorización.- órdenes para especificar derechos de acceso para las relaciones y las vistas.
- SQL incorporado y SQL dinámico.- se pueden incorporar las instrucciones de SQL en lenguajes de programación de propósito general, tales como C, C++, Java, Cobol, Pascal y Fortran.

En este capítulo abordaremos las construcciones y conceptos fundamentales del lenguaje de definición y de manipulación de datos, algunos de los demás aspectos los estudiaremos en capítulos posteriores.

Existen tres maneras de utilizar SQL

1.- *Ejecución directa*.- es el SQL interactivo, las instrucciones SQL se introducen a través de una herramienta que las traduce inmediatamente a la base de datos, por lo que se ejecutan al instante.
2.- *Ejecución dinámica*.- El SQL se incrusta en módulos especiales que pueden ser invocados una y otra vez desde distintas aplicaciones.
3.- *Ejecución incrustada o embebida*.- Las instrucciones SQL se colocan como parte del código de otro lenguaje anfitrión (C, Java, Pascal, Visual Basic,...). Estas instrucciones están separadas del resto del código de forma conveniente. Al compilar el código se utiliza un pre-compilador de la propia base de datos para traducir el SQL

IV.1.1 Algunas reglas sintácticas de SQL.

- En SQL no se distingue entre mayúsculas y minúsculas. Da igual como se escriba.
- El final de una instrucción lo determina el signo del punto y coma.
- Los comandos SQL (SELECT, INSERT,...) pueden ser partidos por espacios o saltos de línea antes de finalizar la instrucción.

En este curso trabajaremos con MySQL, que es un sistema de gestión de base de datos (SGBD) multiusuario, multiplataforma y de código abierto. MySQL pertenece a la compañía sueca MySQL AB, a la que le pertenece casi todos los derechos del código fuente. La compañía desarrolla y mantiene el sistema, vendiendo soporte y servicios, como también las licencias para usar MySQL.

Características de MySQL:

- MySQL está escrito en C y C++.
- Emplea el lenguaje SQL para consultas a la base de datos.
- MySQL Server está disponible como freeware bajo licencia GPL.
- MySQL Enterprise es la versión por suscripción para empresas, con soporte las 24 horas.
- Trabaja en las siguientes plataformas: FreeBSD, HP-UX, GNU/Linux, MacOS X, NetBSD, Novell NetWare, OpenBSD, OS/2 Warp, QNX, SGI IRIX, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64, Microsoft Windows (95, 98, ME, NT, 2000, XP, Vista, Windows7).

IV.1.2 Los tipos de datos de los campos.

Para poder definir el esquema de una tabla, es necesario especificar el tipo de dato de cada uno de sus campos (atributos). SQL define los siguientes tipos de datos:

- Números.

- Cadenas de caracteres.
- Fechas y horas.
- Cadenas de bits.

En este curso trabajaremos con los primeros tres tipos de datos.

A continuación, se reproduce un extracto del manual de MySQL en español, disponible en:
http://dev.mysql.com/doc/refman/5.0/es/index.html

IV.1.3 Tipos numéricos.

Existe una gran variedad de tipos numéricos especificados en el manual de SQL, sin embargo, para fines didácticos solo utilizaremos los que se muestran a continuación.
ZEROFILL sirve para llenar el campo con cero por default, si se especifica **ZEROFILL** para campos numéricos, MySQL añade automáticamente el atributo **UNSIGNED** en el campo, que significa "sin signo", entonces solo se podrá trabajar con el cero y los números positivos, los corchetes cuadrados [ ] indican que el parámetro es opcional.

- INT[(*longitud*)] [UNSIGNED] [ZEROFILL]

Un entero de tamaño normal. *Longitud* es el ancho total del número. El rango con signo es de **-2147483648** a **2147483647**. El rango sin signo es de **0** a **4294967295**.

- BOOL

Un valor de cero se considera falso. Valores distintos a cero se consideran ciertos.

- DOUBLE[(*longitud*,*decimales*)] [UNSIGNED] [ZEROFILL]

Número de punto flotante, es decir, contiene decimales. *Longitud* es el ancho total del número, contando los enteros y los decimales y *decimales* es el número de dígitos que están a la derecha del punto decimal.

Hay más de diez tipos numéricos adicionales, los cuales, en caso de ser necesario, se pueden consultar en el manual de referencia de SQL. El tipo de dato que utilizaremos en el curso es:

NUMERIC[(*longitud*,*decimales*)] [UNSIGNED] [ZEROFILL]

Así, si queremos un entero de 8 dígitos:

NUMERIC (*8*, *0*)

Y si queremos un real con 8 dígitos en los que 3 son para la parte decimal:

NUMERIC (*8*, *3*)

IV.1.4 Tipos de cadenas de caracteres.

Los tipos principales para especificar cadenas de caracteres son:

- **CHAR(*Longitud*) [BINARY | ASCII | UNICODE]**

Una cadena de caracteres de longitud fija que siempre tiene el número necesario de espacios a la derecha para ajustarla a la longitud especificada al almacenarla. *Longitud* representa la longitud de la columna. El rango de *longitud* en MySQL 5.0 es de 0 a 255 caracteres.

Nota: Los espacios a la derecha se borran cuando se obtiene los valores **CHAR** .

- **VARCHAR(*longitud*)**

Cadena de caracteres de longitud variable. *Longitud* representa la longitud de columna máxima. En MySQL 5.0, el rango de *longitud* es de 0 a 255 antes de MySQL 5.0.3, y de 0 a 65,535 en MySQL 5.0.3 y posterior. (La longitud máxima real de un **VARCHAR** en MySQL 5.0 se determina por el tamaño de registro máximo y el conjunto de caracteres que use. La longitud máxima *efectiva* desde MySQL 5.0.3 es de 65,532 bytes.) **VARCHAR** es la abreviación de **CHARACTER VARYING**.

En contraste con **CHAR**, **VARCHAR** almacena los valores usando sólo los caracteres necesarios, más un byte adicional para la longitud (dos bytes para columnas que se declaran con una longitud superior a 255).

Los valores **VARCHAR** no se cortan al almacenarse. El tratamiento de espacios al final depende de la versión. Desde MySQL 5.0.3, los espacios finales se almacenan con el valor y se retornan, según el estándard SQL. Antes de MySQL 5.0.3, los espacios finales se eliminan de los valores cuando se almacenan en una columna **VARCHAR**, esto significa que los espacios también están ausentes de los valores retornados.

- **ENUM('*valor1*','*valor2*',...)**

Una enumeración. Un objeto de cadena de caracteres que **sólo puede tener un valor**, elegido de una lista de valores '*valor1*', '*valor2*', **...**, **NULL** o el valor de error especial '' . Una columna **ENUM** puede tener un máximo de 65,535 valores distintos. Los valores **ENUM** se representan internamente como enteros.

- **SET('*valor1*','*valor2*',...)**

Un conjunto. Un objeto de cadena de caracteres que puede tener **cero o más valores** que deben pertencer a la lista de valores '*valor1*', '*valor2*', **...** Una columna **SET** puede tener un máximo de 64 miembros. Los valores **SET** se representan internamente como enteros.

Al igual que para los tipos de datos numéricos, existen más de diez tipos de cadenas de caracteres que se pueden consultar en el manual de referencia de SQL.

IV.1.5 Tipos de fechas y hora.

Las operaciones más importantes que se pueden hacer con fechas y horas son las siguientes:

- DATE

Una fecha. El rango soportado es de '**1000-01-01**' a '**9999-12-31**'. MySQL muestra valores **DATE** en formato '**YYYY-MM-DD**', pero permite asignar valores a columnas **DATE** usando cadenas de caracteres o números.

- DATETIME

Combinación de fecha y hora. El rango soportado es de '**1000-01-01 00:00:00**' a '**9999-12-31 23:59:59**'. MySQL muestra valores **DATETIME** en formato '**YYYY-MM-DD HH:MM:SS**', pero permite asignar valores a las columnas **DATETIME** usando cadenas de caracteres o números.

- TIMESTAMP[(*longitud*)]

Una marca temporal. El rango es de '**1970-01-01 00:00:00**' hasta el año **2037**.

Una columna **TIMESTAMP** es útil para registrar la fecha y hora de una operación **INSERT** o **UPDATE** . La primera columna **TIMESTAMP** en una tabla se rellena automáticamente con la fecha y hora de la operación más reciente si no le asigna un valor. Puede asignar a cualquier columna **TIMESTAMP** la fecha y hora actual asignándole un valor **NULL** .

En MySQL 5.0, **TIMESTAMP** se retorna como una cadena de caracteres en el formato '**YYYY-MM-DD HH:MM:SS**' cuya anchura de muestra son 19 caracteres. Si quiere obtener el valor como un número, debe añadir **+0** a la columna timestamp .

- TIME

Una hora. El rango es de '**-838:59:59**' a '**838:59:59**'. MySQL muestra los valores **TIME** en formato '**HH:MM:SS**', pero permite asignar valores a columnas **TIME** usando números o cadenas de caracteres.

- YEAR[(2|4)]

Un año en formato de dos o cuatro dígitos. El valor por defecto está en formato de cuatro dígitos. En formato de cuatro dígitos, los valores permitidos son de **1901** a **2155**, y **0000**. En formato de dos dígitos. Las fechas con valores de año de dos dígitos son ambiguas porque no se conoce el siglo. MySQL interpreta los años de dos dígitos usando las siguientes reglas:

- Los valores de años en el rango **00-69** se convierten a **2000-2069**.

- Los valores de años en el rango **70-99** se convierten a **1970-1999**.

Se puede especificar valores **DATETIME**, **DATE**, y **TIMESTAMP** usando cualquier de los siguientes formatos:

- Como cadena de caracteres en formato **'YYYY-MM-DD HH:MM:SS'** o **'YY-MM-DD HH:MM:SS'** . Una sintaxis "relajada" se permite: Cualquier carácter de puntuación puede usarse como delimitador entre partes de fecha o de hora. Por ejemplo, **'98-12-31 11:30:45'**, **'98.12.31 11+30+45'**, **'98/12/31 11*30*45'**, y **'98@12@31 11^30^45'** son equivalentes.
- Como cadena de caracteres en formato **'YYYY-MM-DD'** ó **'YY-MM-DD'** . Se permite una sintaxis "relajada" . Por ejemplo, **'98-12-31'**, **'98.12.31'**, **'98/12/31'**, y **'98@12@31'** son equivalentes.
- Como cadena de caracteres sin delimitadores en formato **'YYYYMMDDHHMMSS'** o **'YYMMDDHHMMSS'**, mientras que la cadena de caracteres tenga sentido como fecha. Por ejemplo, **'19970523091528'** y **'970523091528'** se interpretan como **'1997-05-23 09:15:28'**, pero **'971122129015'** es ilegal (tiene una parte de minutos sin sentido) y se convierte en **'0000-00-00 00:00:00'**.
- Como cadena de caracteres sin delimitadores en formato **'YYYYMMDD'** o **'YYMMDD'** , mientras que el cadena de caracteres tenga sentido como fecha. Por ejemplo, **'19970523'** y **'970523'** se interpretan como **'1997-05-23'**, pero **'971332'** es ilegal (tiene una parte de mes y día sin sentido) y se convierte en **'0000-00-00'**.

- Como número en formato **YYYYMMDDHHMMSS** o **YYMMDDHHMMSS**, mientras que el número tenga sentido como fecha. Por ejemplo, **19830905132800** y **830905132800** se interpretan como **'1983-09-05 13:28:00'**.
- Como número en formato **YYYYMMDD** o **YYMMDD**, mientras que el número tenga sentido como fecha. Por ejemplo, **19830905** y **830905** se interpretan como **'1983-09-05'**.
- Como resultado de una función que retorne un valor aceptable en un contexto **DATETIME**, **DATE**, o **TIMESTAMP** , como **NOW()** o **CURRENT_DATE**.

Los valores ilegales de **DATETIME**, **DATE**, o **TIMESTAMP** se convierten al valor "cero" del tipo apropiado (**'0000-00-00 00:00:00'**, **'0000-00-00'**, o **00000000000000**).

Para valores especificados como cadenas de caracteres que incluyan partes de fecha delimitadas, no es necesario especificar dos dígitos para valores de mes o día menores que **10**. **'1979-6-9'** es lo mismo que **'1979-06-09'**. Similarmente, para valores especificados como cadenas de caracteres que incluyan delimitadores para la parte de hora, no es necesario especificar dos dígitos para horas, minutos o segundos menores que **10**. **'1979-10-30 1:2:3'** es lo mismo que **'1979-10-30 01:02:03'**.

- El formato relajado para valores especificados como cadenas de caracteres puede ser problemático. Por ejemplo, un valor como **'10:11:12'** puede parecer una hora por el delimitador **':'** , pero si se usa en un contexto de fecha se interpreta como **'2010-11-12'**. El valor **'10:45:15'** se convierte a **'0000-00-00'** ya que **'45'** no es un mes legal.

MySQL devuelve y muestra los valores **TIME** en formato **'HH:MM:SS'** (o formato **'HHH:MM:SS'** para valores de hora grandes). **TIME** tiene rango de **'-838:59:59'** a **'838:59:59'**. La razón por la que la parte de hora puede ser tan grande es que el tipo **TIME** puede usarse no sólo para representar una hora del día (que debe ser menor a 24 horas), pero también el tiempo transcurrido o un intervalo de tiempo entre dos eventos (que puede ser mucho mayor a 24 horas, o incluso negativo).

También se puede especificar valores **TIME** en una variedad de formatos:

- Como cadena de caracteres en formato **'D HH:MM:SS.fracción'**. También puede usar una de las siguientes sintaxis "relajadas" : **'HH:MM:SS.fracción'**, **'HH:MM:SS'**, **'HH:MM'**, **'D HH:MM:SS'**, **'D HH:MM'**, **'D HH'**, o **'SS'**. Aquí **D** representa días y puede tener un valor de 0 a 34. Tenga en cuenta que MySQL no almacena la fracción (todavía).
- Como cadena de caracteres sin delimitadores en formato **'HHMMSS'**, mientras que tenga sentido como hora. Por ejemplo, **'101112'** se entiende como **'10:11:12'**, pero **'109712'** es ilegal (no tiene una parte de minutos correcta) y pasa a ser **'00:00:00'**.
- Como número en formato **HHMMSS**, mientras tenga sentido como hora. Por ejemplo, **101112** se entiende como **'10:11:12'**. Los siguientes formatos alternativos también se entienden: **SS**, **MMSS**, **HHMMSS**, **HHMMSS.fracción**. Tener en cuenta que MySQL no almacena la fracción (todavía).
- Como resultado de una función que retorna un valor que es aceptable en un contexto **TIME**, tal como **CURRENT_TIME**.

Para valores **TIME** especificados como cadenas de caracteres que incluyan un delimitador de las partes de hora, no es necesario especificar dos dígitos para horas, minutos o segundos que tengan un valor inferior a **10**. **'8:3:2'** es lo mismo que **'08:03:02'**.

Hay que tener cuidado con asignar valores abreviados a una columna **TIME** . Sin comas, MySQL interpreta los valores asumiendo que los dos dígitos más a la derecha representan segundos. (MySQL interpreta valores **TIME** como tiempo transcurrido en lugar de horas del día.) Por ejemplo puede pensar que **'1112'** y **1112** significan **'11:12:00'** (12 minutos tras las 11 en punto), pero MySQL los interpreta como **'00:11:12'** (11 minutos, 12 segundos). Similarmente, **'12'** y **12** se interpretan como **'00:00:12'**. Los valores **TIME** con comas, por contrario, se tratan siempre como hora del día. Esto es, **'11:12'** significa **'11:12:00'**, no **'00:11:12'**.

Los valores fuera del rango de **TIME** pero que son legales se cambian por el valor límite de rango más cercano. Por ejemplo **'-850:00:00'** y **'850:00:00'** se convierten en **'-838:59:59'** y **'838:59:59'**.

Los valores **TIME** ilegales se convierten a **'00:00:00'**. Hay que tener en cuenta que como **'00:00:00'** es un valor **TIME** legal, no hay forma de decir si un valor **'00:00:00'** almacenado en una tabla, se insertó como **'00:00:00'** o como valor ilegal.

IV.1.6 Operadores relacionales y lógicos.

Antes de comenzar con esta sección, mencionaremos brevemente la instrucción SELECT. Este comando permite realizar consultas sobre los datos de la base de datos, es decir, obtiene datos de la base de datos. A ésta parte del lenguaje se la conoce como DQL (Data Query Language, Lenguaje de consulta de datos); y forma parte del LMD (Data Management Languaje) que se verá posteriormente.

IV.1.6.1 Operadores relacionales o de comparación.

Las operaciones relacionales o de comparación dan un valor de **1** (CIERTO), **0** (FALSO), o **NULL**. Estas operaciones funcionan tanto para números como para cadenas de caracteres. Los más importantes son:

- =

Significa **igual**, por ejemplo:

```
mysql> SELECT 1 = 0;
    -> 0
```

- <>, **!=**

Significa **diferente**, por ejemplo:

```
mysql> SELECT '.01' <> '0.01';
    −> 1
mysql> SELECT .01 <> '0.01';
    −> 0
mysql> SELECT 'zapp' <> 'zappp';
    −> 1
```

- <=

Significa **menor que o igual**, por ejemplo:

```
mysql> SELECT 0.1 <= 2;
    −> 1
```

- <

Significa **menor que**, por ejemplo:

```
mysql> SELECT 2 < 2;
    −> 0
```

- >=

Significa **mayor que o igual**, por ejemplo:

```
mysql> SELECT 2 >= 2;
    −> 1
```

- >

Significa **mayor que**, por ejemplo:

```
mysql> SELECT 2 > 2;
```

−> 0

- **IS** *valor booleano*, **IS NOT** *valor booleano*

Comprueba un valor contra un valor booleano, donde *boolean_value* puede ser **TRUE**, **FALSE**, o **UNKNOWN**.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
    −> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
    −> 1, 1, 0
```

- **IS NULL**, **IS NOT NULL**

Prueba si un valor es o no **NULL**.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
    −> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
    −> 1, 1, 0
```

- **GREATEST(***value1***,***value2***,...)**

Con dos o más argumentos, retorna el argumento mayor (con valor mayor). Los argumentos se comparan usando las mismas reglas que para **LEAST()**.

```
mysql> SELECT GREATEST(2,0);
    −> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
    −> 767.0
mysql> SELECT GREATEST('B','A','C');
    −> 'C'
```

- *expr* **IN (***value***,...)**

Regresa **1** si *expr* es uno de los valores en la lista **IN** , de lo contrario regresa **0**. Si todos los valores son constantes, se evalúan según el tipo y ordenación de *expr* . La búsqueda para el elemento se hace usando búsqueda binaria. Esto significa que **IN** es muy rápido si la lista **IN** consiste en constantes. Si *expr* es una expresión de cadenas de caracteres sensible a mayúsculas, la comparación de cadenas se realiza sensible a mayúsculas.

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
    −> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
    −> 1
```

- **LEAST(***value1***,***value2***,...)**

Con dos o más argumentos, retorna el argumento menor (con un valor menor). Los argumentos se comparan usando las siguientes reglas:

- Si el valor del resultado está en un contexto **INTEGER** o todos los argumentos son enteros, se comparan como enteros.
- Si el valor del resultado se usa en un contexto **REAL** o todos los argumentos son reales, se comparan como reales.
- Si algún argumento es una cadena de caracteres sensible a mayúsculas, los argumentos se comparan como cadenas sensibles a mayúsculas.
- En cualquier otro caso, los argumentos se comparan como cadenas no sensibles a mayúsculas.

```
mysql> SELECT LEAST(2,0);
    -> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
    -> 3.0
mysql> SELECT LEAST('B','A','C');
    -> 'A'
```

IV.1.6.2 Operadores lógicos.

En SQL, todos los operadores lógicos se evalúan a **TRUE**, **FALSE**, o **NULL** (**desconocido**). En MySQL, se implementan como 1 (**TRUE**), 0 (**FALSE**), y **NULL**.

- **NOT**, **!**

NOT lógica. Regresa **1** si el operando es **0**, regresa **0** si el operando es diferente a cero, y **NOT NULL** da como resiltado **NULL**.

```
mysql> SELECT NOT 10;
    -> 0
mysql> SELECT NOT 0;
    -> 1
mysql> SELECT NOT NULL;
    -> NULL
mysql> SELECT ! (1+1);
    -> 0

mysql> SELECT ! 1+1;
    -> 1
```

El último ejemplo produce **1** porque la expresión se evalúa igual que **(!1)+1**.

- **AND**, **&&**

AND lógica. Se evalúa a **1** si todos los operandos son distintos a cero y no **NULL**, a 0 si uno o más operandos son **0**, de otro modo retorna **NULL**.

```
mysql> SELECT 1 && 1;
    -> 1
mysql> SELECT 1 && 0;
    -> 0
mysql> SELECT 1 && NULL;
    -> NULL
mysql> SELECT 0 && NULL;
    -> 0
mysql> SELECT NULL && 0;
    -> 0
```

- **OR**, **||**

OR lógica. Cuando ambos ooperandos son no **NULL**, el resultado es 1 si algún operando es diferente a cero, y 0 de otro modo. Con un operando **NULL** el resultado es 1 si el otro operando no es cero, y **NULL** de otro modo. Si ambos operandos son **NULL**, el resultado es **NULL**.

```
mysql> SELECT 1 || 1;
    -> 1
mysql> SELECT 1 || 0;
    -> 1
mysql> SELECT 0 || 0;
    -> 0
mysql> SELECT 0 || NULL;
    -> NULL
mysql> SELECT 1 || NULL;
    -> 1
```

- **XOR**

XOR lógica. Retorna **NULL** si algún operando es **NULL**. Para operandos no **NULL** , evalúa a 1 si un número par de operandos es distinto a cero, sino retorna 0.

```
mysql> SELECT 1 XOR 1;
    -> 0
mysql> SELECT 1 XOR 0;
    -> 1
mysql> SELECT 1 XOR NULL;
    -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
    -> 1
```

a XOR b es matemáticamente igual a **(a AND (NOT b)) OR ((NOT a) and b)**.

IV.2 Instrucciones básicas del Lenguaje de Definición de Datos (LDD).

En el lenguaje SQL se puede trabajar a nivel del Lenguaje de Definición de Datos (LDD) (en inglés DDL: Data Definition Languaje), en este nivel es donde trabaja el administrador de la base de datos. Con el DDL, el administrador puede crear esquemas de las relaciones, y dar mantenimiento a la base de datos.

El LDD Permite modificar la estructura de las tablas de la base de datos. Lo forman las instrucciones CREATE, ALTER, DROP y RENAME.

IV.2.1 Como crear tablas: CREATE.

Antes de comenzar a trabajar con una base de datos es necesario crearla, esto se hace con el comando:

```
MySQL> CREATE DATABASE nombre_Base_Datos;
```

Como ejemplo crearemos la base de datos llamada escuela.

```
MySQL> CREATE DATABASE escuela;
```

El comando CREATE no es suficiente, una vez que se crea la base de datos, hay que "usarla" con el comando USE. Entonces, para poder trabajar con ella, es necesario el comando:

```
MySQL> USE nombre_Base_Datos;
```

en nuestro caso:

```
MySQL> USE escuela;
```

Cuando se trabajen sesiones posteriores, no habrá que crear la base de datos nuevamente, bastará con "usarla" con el comando USE. Nótese, que si se crea la base de datos con CREATE y después no se utiliza el comando USE, será imposible empezar a trabajar con ella.

Leer la base de datos.- Para saber en que base de datos estamos trabajando se utiliza la función DATABASE(), en nuestro caso, como se muestra en la *figura 4.1* la base de datos se llama "escuela":

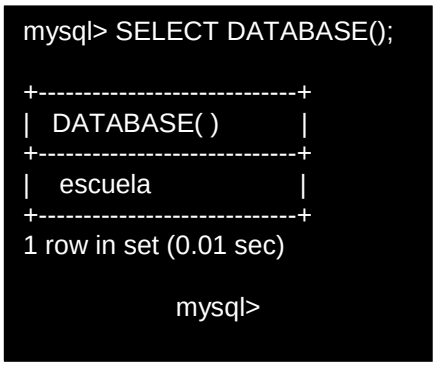


Figura 4.1: La base datos que se esta usando se llama "escuela".

Una vez que nuestra base de datos está lista para trabajar, podemos *crear una tabla dentro de esta base de datos* con el comando CREATE, proporcionando los campos correspondientes:

```
mysql>CREATE TABLE alumnos(nombre        VARCHAR(20),
                           FechaIngreso DATE,
                           sexo          CHAR(1),
                           promedio      DOUBLE(4,2)
                           edad          INT(2));
```

Para consultar todas las tablas que tenemos en una base de datos, podemos usar el comando SHOW TABLES como se muestra a continuación en la *figura 4.2*:

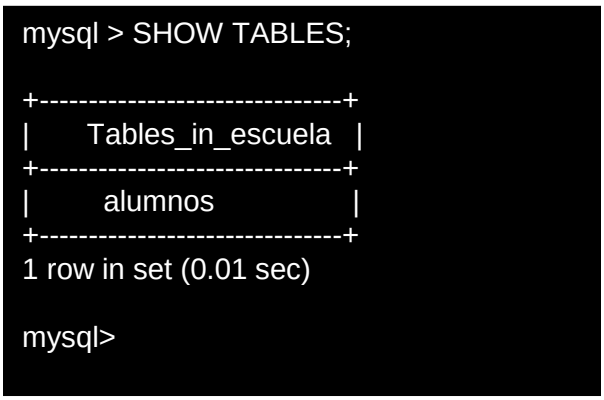


Figura 4.2: La tabla en la base de datos es "alumnos".

Para consultar los campos de una tabla, podemos usar en todo momento el comando DESCRIBE nombre_tabla, esto es útil para acordarnos del nombre, del orden o el tipo de dato de los campos de la tabla, como se muestra en la *figura 4.3*.

```
mysql > DESCRIBE alumnos;


+--------------------+----------------+-------+-------+----------+--------+
|      Field         |    Type        |  Null  | Key | Default | Extra  |
+--------------------+----------------+-------+-------+----------+--------+
|   nombre           | varchar(20) | YES |         | NULL   |         |
|  FechaIngreso   | date           | YES |         | NULL   |         |
|  sexo              | char(1)        | YES |         | NULL   |         |
|  promedio        | double(4,2)  | YES |         | NULL   |         |
|  edad              | int(2)         | YES |         | NULL   |         |
+--------------------+-----------------+-------+-------+----------+--------+
4 row in set (0.01 sec)

mysql>
```

Figura 4.3: Los campos de la tabla "alumnos".

IV.2.2 Cargar datos en una tabla: LOAD….INTO.

Lo más conveniente para cargar datos en una tabla, es usar un archivo con extensión *.txt, entonces, hay que crear un archivo con la extensión txt que contenga un registro por línea, con cada valor separado por un carácter de tabulación, y dispuestos en el orden en el cual se especificaron las columnas en la sentencia CREATE TABLE.

Para valores ausentes, se usan los valores NULL. Para representar estos valores en el archivo de texto, se utiliza **\N** (barra diagonal y N mayúscula). Por ejemplo, si no se tiene su promedio, el registro de Atenogenes se vería del modo siguiente (el espacio en blanco entre cada valor es un solo carácter de tabulación):

Gumaro Perez	2007-02-24	M	8.5	24
Atenogenes Lopez	2007-10-01	M	**\N**	21
Timotea Paredes	2005-06-17	F	8.3	22

Para cargar el archivo alumnos.txt dentro de la tabla alumnos es necesario utilizar el siguiente comando:

MySQL> LOAD DATA LOCAL INFILE 'path/ alumnos.txt' INTO TABLE alumnos;

```
mysql > LOAD DATA LOCAL INFILE 'path/ alumnos.txt' INTO TABLE
alumnos;

Query OK 3 rows affected, 3 warnings (0.01 sec)
Records: 3 Deleted:0  Skipped: 0 warnings: 3

mysql>
```

Figura 4.4: Cargar datos desde un archivo (Comando exitoso).

En la *figura 4.4* se muestra el comando que se utiliza para cargar directamente los datos de una tabla desde un archivo, este comando permite editar primero los datos en un archivo y cerciorarse de que estén correctos.

OJO!

1.- Cuando copiamos una ruta de acceso (path) la computadora usa la diagonal invertida:
c:\.....\....\archivo.txt
MySQL no entiende la diagonal invertida, hay que remplazarla por la diagonal normal:
c:/...../..../archivo.txt

2.- Cuando la última columna de un archivo *.txt se va a cargar a un campo de tipo CHAR o VARCHAR en una tabla de MySQL, es necesario terminarla con un TAB antes de pasar al siguiente renglón, de no ser así la información no se lee correctamente y la tabla de MySQL no contiene la información adecuada.

Otra manera de introducir datos en una tabla es cargarlos línea por línea, esto se hace con el comando INSERT INTO, por ejemplo:

```
MySQL> INSERT  INTO alumnos
    –> VALUES ('Aida Lopez', '2008-03-20', 'F', '9.4','32');
```

IV.2.3 Modificando la estructura y el contenido de una tabla: ALTER TABLE.

Para cambiar la estructura de una tabla existente se utiliza el comando **ALTER TABLE** . Por ejemplo, se puede añadir o borrar columnas, crear o destruir índices, cambiar el tipo de columnas existentes, renombrar columnas o incluso renombrar la misma tabla.

ALTER TABLE funciona creando una copia temporal de la tabla original. La alteración se realiza en la copia, luego la tabla original se borra y se renombra la nueva. Mientras se ejecuta **ALTER TABLE** la tabla original es legible por otros clientes. Las actualizaciones y escrituras en la tabla se esperan hasta que la nueva tabla esté lista, luego se redirigen automáticamente a la nueva tabla para evitar fallas en las actualizaciones.

IV.2.3.1 Borrar un campo de una tabla.

Para borrar un campo de una tabla se usa el comando **ALTER TABLE** añadiendo la cláusula **DROP**. Por ejemplo, si tenemos la tabla **t2** y necesitamos borrar el campo **c**, entonces usamos el comando **ALTER TABLE** con la cláusula **DROP**, como se muestra a continuación:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

La palabra COLUMN es opcional y puede omitirse:

```
mysql> ALTER TABLE t2 DROP c;
```

IV.2.3.2 Para agregar un campo a una tabla.

Para agregar un campo a una tabla se usa el comando **ALTER TABLE** añadiendo la cláusula **ADD**. Por ejemplo, si tenemos la tabla **t2** y necesitamos agregarle el campo **d**, de tipo TIME, entonces usamos el comando **ALTER TABLE** con la cláusula **ADD**, como se muestra a continuación:

```
mysql> ALTER TABLE t2 ADD d TIME;
```

IV.2.3.3 Para modificar el campo de una tabla.

Para hacer modificaciones al campo de una tabla se usa el comando **ALTER TABLE** añadiendo las cláusulas **CHANGE** y/o **MODIFY** según sea el caso.

Para ***cambiar el tipo de dato*** del campo **a** de la tabla **t2** de **INT** a **DOUBLE** dejando el mismo nombre al campo, utilizamos la cláusula **MODIFY**.

```
mysql> ALTER TABLE t2 MODIFY a DOUBLE;
```

Otro ejemplo, si tenemos la tabla **t2** y necesitamos modificar el campo **d**, para que sea de tipo DATE, entonces usamos el comando **ALTER TABLE** con la cláusula **MODIFY**, como se muestra a continuación:

```
mysql> ALTER TABLE t2 MODIFY d DATE;
```

Otro ejemplo, cambiar el campo **b** de **CHAR(10)** a **VARCHAR(20)** y además cambiarle el nombre de **b** a **d**:

Para ***cambiar el nombre del campo*** de una tabla **t2** utilizamos la cláusula **CHANGE**. Por ejemplo si queremos renombrar el campo de **b** a **d**:

```
mysql> ALTER TABLE t2 CHANGE b d;
```

También es posible cambiar el nombre del campo al mismo tiempo que su tipo de datos, por ejemplo:

```
mysql> ALTER TABLE t2 CHANGE b  d VARCHAR(20);
```

Modifica el campo **b** para que se llame **d** y cambia su tipo a VARCHAR(20).

IV.2.3.4 Para agregar un campo a una tabla.

Para agregar un campo a una tabla, usamos el siguiente comando:

```
mysql> ALTER TABLE nombre_tabla ADD nombre_columna  tipo_dato;
```

Por ejemplo:

```
mysql> ALTER TABLE t2 ADD e DATE;
```

IV.2.4 Renombrar una tabla: ALTER TABLE…RENAME.

Para renombrar una tabla se usa el comando **ALTER TABLE**. Por ejemplo, si se crea una tabla **t1** como se muestra a continuación:

```
mysql> CREATE TABLE t1 (a INT,b CHAR(10), c DOUBLE);
```

Para renombrar la tabla de **t1** a **t2** usamos:

```
mysql> ALTER TABLE t1 RENAME t2;
```

IV.2.5 Borrar una tabla: DROP.

Para borrar una tabla de la base de datos:

```
mysql> DROP TABLE nombre_tabla;
```

IV.3 Instrucciones básicas del Lenguaje de Manipulación de Datos (LMD).

Otro nivel del SQL es el Lenguaje de Manipulación de Datos (LMD) (en inglés DML: Data Management Languaje), en este nivel es en donde trabajan los usuarios. El LMD está formado por un conjunto de instrucciones que permiten al usuario hacer consultas, llenar formularios y/o generar reportes, en función de la autorización que tenga para realizar las operaciones.

El LMD es un lenguaje de manipulación de datos, este es un tipo de lenguaje declarativo; esto significa que el usuario expresa qué información desea obtener de la base de datos, pero no indica la forma en la cual esta información se debe localizar y recuperar.

El LMD se usa para modificar tuplas (registros) de la base de datos. Además del DQL, con la instrucción SELECT, lo forman las instrucciones INSERT, UPDATE, MERGE y DELETE.

IV.3.1 Borrar una tupla (registro): DELETE.

Para borrar una tupla o registro de una tabla se usa el comando **DELETE** añadiendo la cláusula **WHERE**, el uso de esta instrucción puede llegar a ser bastante sofisticado, **DELETE** tiene otras cláusulas adicionales que pueden consultarse en el manual, por cuestiones prácticas, aquí no limitaremos a estudiar el uso de la cláusula **WHERE**. La sintaxis es la siguiente:

DELETE FROM *nombre_tabla* [WHERE *condición*]

DELETE borra todos los registros de la tabla *nombre_tabla* que satisfacen la *condición*, y retorna el número de registros borrados.

Más adelante, en la sección IV.3.2, estudiaremos como consultar los datos que contiene una tabla con la instrucción SELECT, adelantándonos un poco, en la *figura 4.5* tenemos que la tabla alumnos contiene los siguientes datos:

```
mysql > SELECT * FROM alumnos;

+-----------------------+-------------------+---------+---------------|---------+
|      nombre           | Fecha Ingreso |  sexo  |  promedio  |  edad |
+-----------------------+-------------------+---------+---------------+---------+
|   Gumaro Perez       |  2007-02-24   |    M     |    8.5        |   24    |
|  Atenogenes López |  2007-10-01   |    M     |    7.1        |    21   |
|  Timotea Paredes    |  2005-06-17   |    F     |    8.3        |   22    |
+-----------------------+-------------------+---------+---------------+---------+
3 row in set (0.01 sec)

mysql>
```

Figura 4.5: Usando SELECT para mostrar los datos de una tabla.

Si queremos borrar el segundo registro de la tabla alumnos, debemos usar una condición en la cláusula en donde el valor de algún campo coincida con el valor del segundo registro, hay que tomar en cuenta que este valor debe ser único del segundo registro, así:

mysql> DELETE FROM alumnos WHERE nombre = 'Atenogenes Lopez';

Borra la segunda tupla de la tabla. Si pedimos que se borren los registros cuyo campo sexo es 'M':

mysql> DELETE FROM alumnos WHERE sexo = 'M';

entonces, se borran los dos primeros registros de la tabla.

Para borrar todos los registros de una tabla se utiliza **DELETE FROM** *nombre_tabla* (sin cláusula **WHERE**).

IV.3.2 Haciendo consultas sencillas: SELECT.

DQL es la abreviatura de "Data Query Language" (lenguaje de consulta de datos) y es parte del LMD. El único comando que pertenece al DQL es SELECT. Este comando permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (tuplas) de una tabla de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (producto cartesiano, join).
- Realizar cálculos sobre los datos.
- Agrupar datos.

Para poder ver el contenido de una tabla, utilizamos el comando SELECT, este se utiliza para mostrar la información de una tabla, la sintaxis general de esta instrucción es la siguiente:

```
SELECT  <lista de atributos>
FROM    <lista de tablas>
WHERE  <condiciónes>;
```

La *lista de atributos* es la información que se requiere visualizar, es una lista de nombres de campos cuyos valores van a ser recuperados por la consulta, puede ser una lista de columnas (campos), o se puede usar el asterisco * para indicar "*todas las columnas*". La *lista de tablas* indica la o las tablas en donde están los datos que se desea recuperar, es una lista de nombres de las relaciones necesarias para procesar la consulta. Un ejemplo muy sencillo:

```
mysql> SELECT * FROM alumnos;
```

La cláusula WHERE es opcional. Si está presente, *condiciones* representa las condiciones que cada registro debe cumplir para ser un resultado de la consulta. Es una expresión condicional (booleana) que identifica las tuplas que van a ser recuperadas por la consulta.

Cuando no se ha introducido ningún valor obtenemos el mensaje que se muestra en la *figura 4.6*:

```
mysql > SELECT * FROM alumnos;
 Empty set (0.0 sec)

mysql>
```

Figura 4.6: Tabla sin datos.

Entonces, antes de consultar en una tabla, es necesario que se le hayan introducido datos previamente!

Esto se puede hacer de dos maneras, una es cargando todos los datos que ya están previamente en un archivo, con el comando:

```
mysql> LOAD DATA LOCAL INFILE 'path/ nombreArchivo.txt' INTO TABLE nombreTabla;
```

Y la otra forma es introduciendo los datos línea por línea, con el comando:

```
mysql > INSERT  INTO nombreTabla
        –>VALUES (‘valor del campo1’, ‘valor del campo2,…,
                  ‘valor del campo’N’’ );
```

Entonces sí se puede observar el contenido de toda la tabla con el commando SELECT * FROM alumnos:

```
mysql > SELECT * FROM alumnos;

+------------------------+--------------------+---------+---------------|---------+
|      nombre            | Fecha Ingreso |  sexo  |  promedio  |  edad |
+------------------------+--------------------+---------+---------------+---------+
|   Gumaro Pérez       |  2007-02-24   |    M     |    8.5        |   24    |
|  Atenógenes López |  2007-10-01   |    M     |    7.1        |    21   |
|  Timotea Paredes    |  2005-06-17   |    F     |    8.3        |   22    |
+------------------------+--------------------+---------+---------------+---------+
3 row in set (0.01 sec)

mysql>
```

Figura 4.7: Datos de la tabla “alumnos”.

Puede observarse que la tabla ya se cargó con el contenido del archivo alumnos.txt del ejemplo de la sección IV.2.2.

Normalmente no se desea ver una tabla completa, especialmente cuando es muy grande, sino solo los registros que cumplen con determinada condición. La *tabla 4.1* resume las operaciones más sencillas que se pueden hacer con los operadores relacionales básicos.

SÍMBOLO	SIGNIFICADO	SINTAXS
=	Igual que	Nombre_campo = ‘cierto_valor’
>	Mayor que	Nombre_campo > ‘cierto_valor’
<	Menor que	Nombre_campo <‘cierto_valor’
>=	Mayor o igual	Nombre_campo >= ‘cierto_valor’
<=	Menor o igual	Nombre_campo <= ‘cierto_valor’
!= , < >	Diferente de	Nombre_campo != ‘cierto_valor’

Tabla 4.1: Operaciones relacionales básicas.

Por ejemplo, para que se despliegue la tupla con todos los datos que pertenecen a una persona con un nombre en particular, agregamos la cláusula WHERE con la condición:

mysql > SELECT * FROM nombre_tabla WHERE nombre = ‘nombre_particular’;

```
mysql > SELECT * FROM alumnos WHERE NOMBRE = ‘Timotea
Paredes’;

+------------------------+-------------------+---------+----------------+
|      nombre            | Fecha Ingreso |  sexo  |  promedio    |
+------------------------+-------------------+---------+----------------+
|  Timotea Paredes    |  2005-06-17   |    F     |    8.3          |
+------------------------+-------------------+---------+----------------+
1 row in set (0.01 sec)

mysql>
```

Figura 4.8: La tupla que cumple con el nombre = Timotea Paredes.

IV.3.2.1 Desplegado por columnas.

Con el comando SELECT es posible desplegar únicamente determinadas columnas (campos). Esto es útil para filtrar los campos que no se requieran. Para hacer esto se usa SELECT seguido de los nombres de los campos que se requiere que aparezcan en pantalla:

SELECT *columna1,…., columnaN*;

Por ejemplo, supongamos que tenemos la tabla llamada empleados:

Código	Nombre	Edad	Depto
1	Jorge Campos	33	1
2	Enrique Muñoz	25	1
3	Esteban Paz	21	1
8	Jorge Arias	30	2
10	Juan Martínez	19	2
12	Anselmo Rodas	28	6

Tabla 4.2: La tabla “empleados”.

Si hacemos la siguiente consulta:

```
mysql >SELECT nombre, edad
        FROM empleados
        WHERE edad >= 28;
```

Entonces tendremos la salida de la *tabla 4.3*:

Nombre	Edad
Jorge Campos	33
Jorge Arias	30
Anselmo Rodas	28

Tabla 4.3: Resultado de la primera consulta.

Así solo se despliegan las columnas de los campos nombre y edad, además, solo tres tuplas de la relación cumplen con la condición de la cláusula WHERE.

IV.3.2.2 Consultas con los operadores lógicos AND, OR y NOT.

Con el comando SELECT se permiten consultas tan complejas como sea necesario, usando conectores **AND**, **OR**, **NOT**. Por ejemplo, si tenemos la tabla empleados del ejemplo anterior y damos el siguiente comando:

```
mysql > SELECT *
        FROM empleado
        WHERE edad < 28 AND depto = 1;
```

Entonces tendremos la siguiente salida:

Código	Nombre	Edad	Depto
2	Enrique Muñoz	25	1
3	Esteban Paz	21	1

Tabla 4.4: resultado de la segunda consulta.

La condición puede ser cualquier expresión relacional válida:

```
SELECT campo1, campo2
FROM tabla1, tabla2
WHERE campo3 = valor AND
tabla2.campo1 = tabla1.campo1;
```

Se pueden usar los siguientes operadores:

- <, <=, >, >=, =, <> (o **!**=)
- +, -, ***, /
- **AND**, **OR**, **NOT**

IV.3.2.3 Ordenando datos.

La cláusula ORDER BY de la instrucción SELECT indica que campo (columna) debe usarse para ordenar las tuplas de una tabla, sintaxis:

SELECT *campo1, campo2, ..., campoN*
FROM tabla1,
ORDER BY campoX;

Por ejemplo, si se requiere obtener el nombre de todos los clientes y su zona, ordenando las tuplas por zona, entonces hacemos:

```
mysql> SELECT ALL nomb_cliente, zona, debe FROM cliente ORDER BY zona;
```

Y el resultado se muestra en la *figura 4.9*.

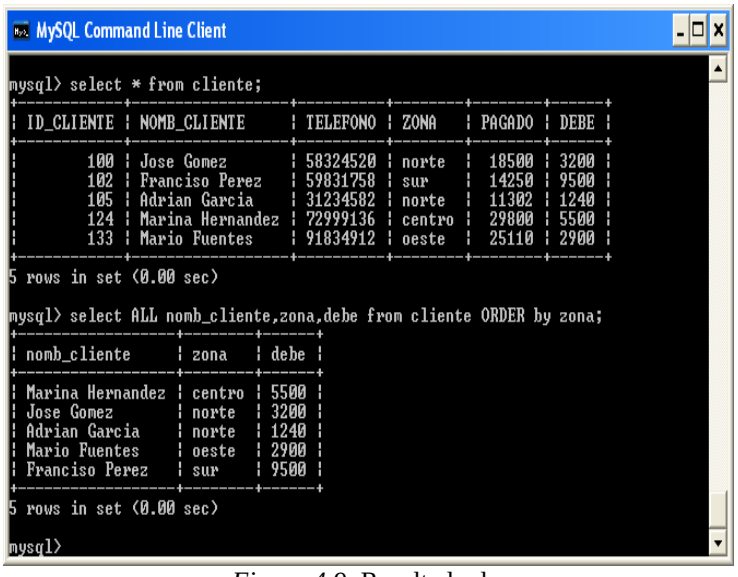


Figura 4.9: Resultado de:
SELECT ALL nomb_cliente, zona, debe FROM cliente ORDER BY zona;

Los campos que son cadenas de caracteres se ordenan por orden alfabético, y los numéricos de menor a mayor, así si lo que deseamos es obtener de la tabla cliente el nombre del cliente, su zona y la cantidad que debe, ordenando respecto a la cantidad que debe el cliente (de más a menos) entonces hacemos:

```
mysql> SELECT nomb_cliente, zona, debe FROM cliente ORDER BY debe DESC;
```

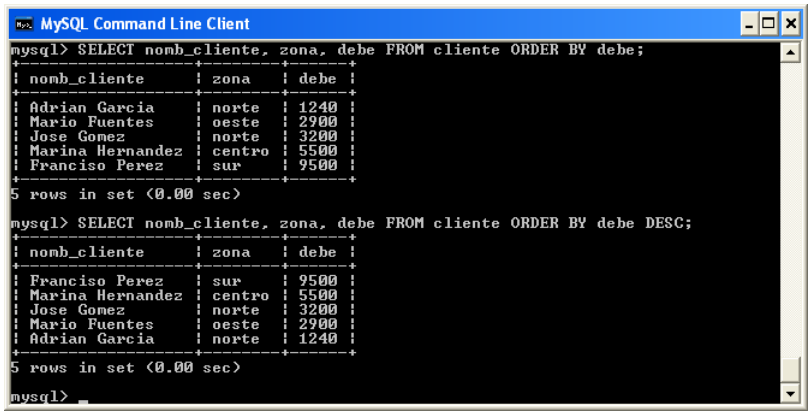


Figura 4.10: Resultado de:
SELECT nomb_cliente, zona, debe FROM cliente ORDER BY debe DESC;

También es posible obtener el valor máximo de una columna, por medio de la cláusula MAX, por ejemplo, si se desea obtener cual es la cantidad máxima que se debe dentro de la tabla cliente:

```
mysql> SELECT MAX(debe) FROM cliente;
```

Ahora, si deseamos que aparezca más información, hacemos lo que se llama una ***subconsulta***:

```
mysql> SELECT nomb_cliente, telefono, debe FROM cliente
    -> WHERE debe = ( SELECT MAX(debe) FROM cliente);
```

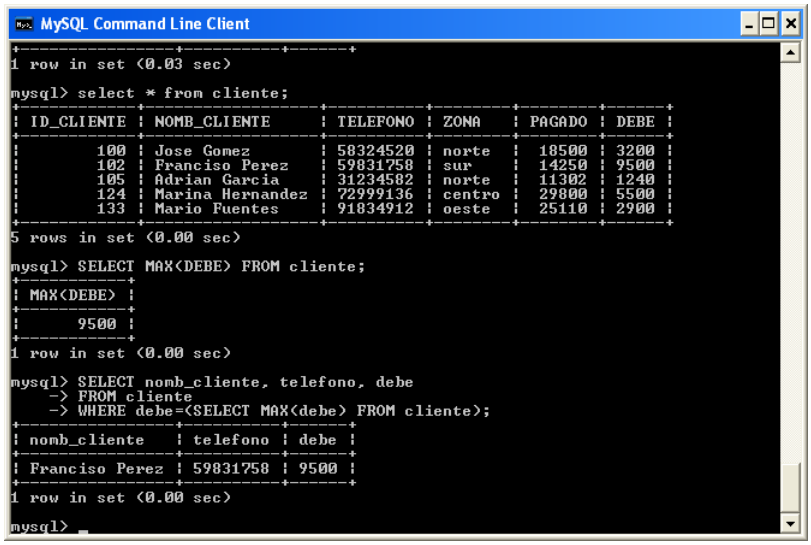


Figura 4.11: Resultado de:
SELECT nomb_cliente, telefono, debe FROM cliente
–> WHERE debe = (SELECT MAX(debe) FROM cliente);

IV.3.2.4 Agrupando datos.

La cláusula GROUP BY de la instrucción SELECT indica que campo (columna) debe usarse para agrupar las tuplas de una tabla, sintaxis:
SELECT *campo1, campo2, ..., campoN*
FROM tabla1,
GROUP BY campoX;

Esto es muy aplicable cuando queremos obtener totales de algún o algunos grupos.

Por ejemplo, si tenemos una tabla llamada "ventas" con los datos de cada venta que hizo un vendedor, podemos obtener la cantidad total de productos que vendió cada vendedor:

```
mysql> SELECT ID_vendedor SUM(cantidad) FROM venta
            GROUP BY ID_vendedor;
```

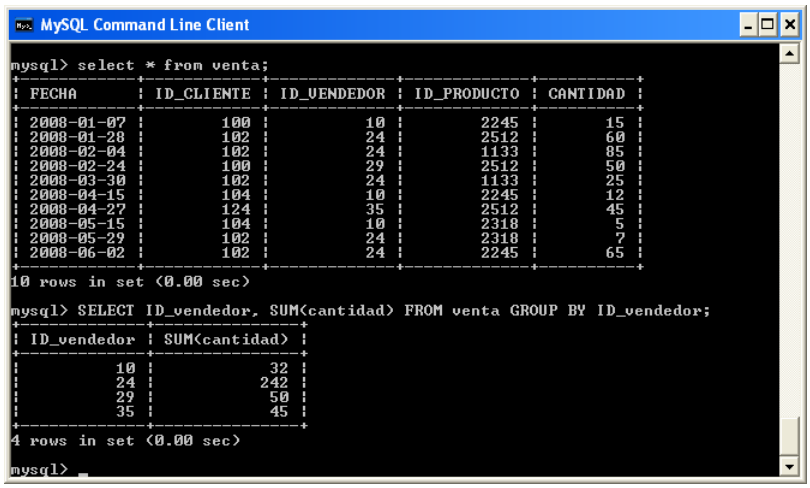


Figura 4.12: Resultado de:
SELECT ID_vendedor SUM(cantidad) FROM venta
GROUP BY ID_vendedor;

IV.3.2.5 NOT IN y subconsultas.

Cuando se agrega la cláusula NOT IN a la instrucción SELECT se despliegan las tuplas que no aparecen en el conjunto especificado por una subconsulta, su sintaxis es:

SELECT *campo1, campo2, ..., campoN*
FROM tabla1,
WHERE *campoX* NOT IN (subconsulta);

Una *subconsulta* es una consulta anidada dentro de otra. Por ejemplo, si tenemos las siguientes dos tablas, una de estudiantes y otra de profesores que se muestran en la *figura 4.13*:

Estudiantes

ID	Nombre	Edad
10	Pedro	14
20	Olga	29
50	Ana	30

Profesores

ID	Nombre	Edad
45	Gloria	24
26	Juan	29
50	Ana	30
34	Cristina	32

Figura 4.13: Tablas “estudiantes” y “profesores”.

y queremos desplegar las ID y los nombres de todos los estudiantes que no sean también profesores, entonces:

```
mysql> SELECT ID,nombre FROM estudiantes
    -> WHERE ID NOT IN (SELECT ID FROM profesores);
```

Como se observa en la *figura 4.14*:

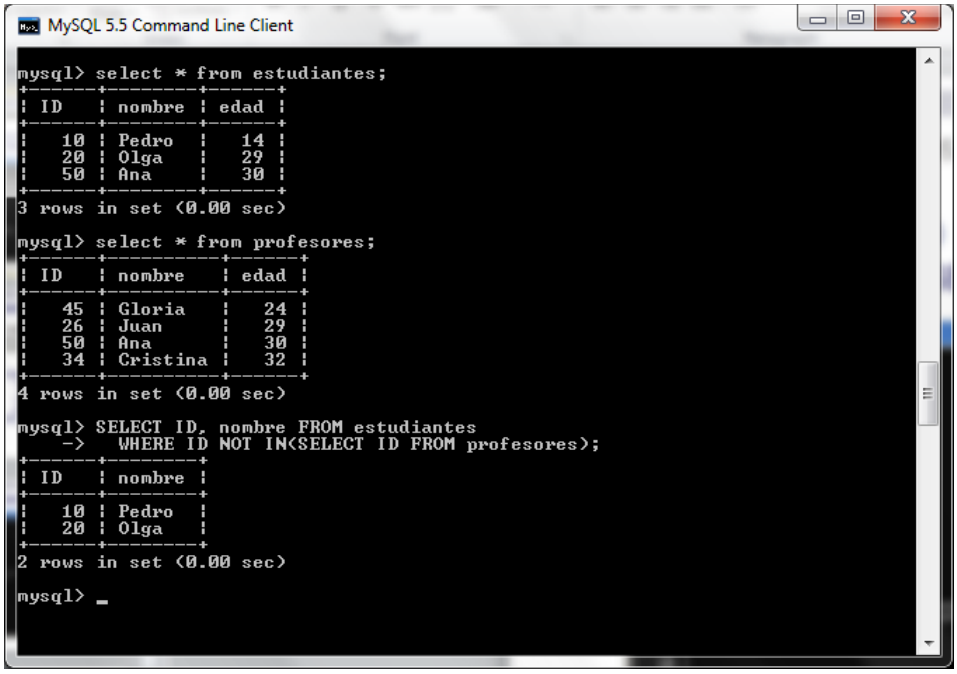


Figura 4.14: Resultado de
SELECT ID,nombre FROM estudiantes
-> WHERE ID NOT IN (SELECT ID FROM profesores);

IV.3.3 Actualizaciones: UPDATE.

En determinadas ocasiones puede ser deseable cambiar un valor dentro de una tupla, sin cambiar todos los valores de la misma. Para este tipo de situaciones se utiliza la instrucción UPDATE. Al igual que ocurre con INSERT y con DELETE, se pueden elegir las tuplas que van a ser actualizadas mediante una consulta, la sintaxis de la instrucción UPDATE es la siguiente:

```
UPDATE nombre_tabla
   SET nombre_columna=nuevo_valor
   [WHERE condición];
```

Por ejemplo, si queremos cambiar el departamento del profesor Nicodemo Sanchez en la tabla de profesores:

mysql> UPDATE profesores SET depto =’Quimca’ WHERE num_empl = 3172;

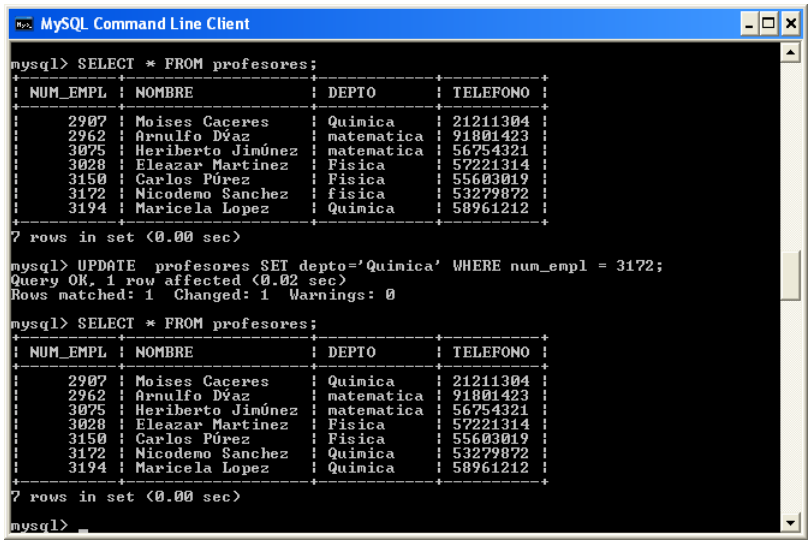


Figura 4.15: Resultado de
UPDATE profesores SET depto =’Quimca’ WHERE num_empl = 3172;

Si la condición se cumple para varias tuplas, entonces se modifican todas ellas, por ejemplo:

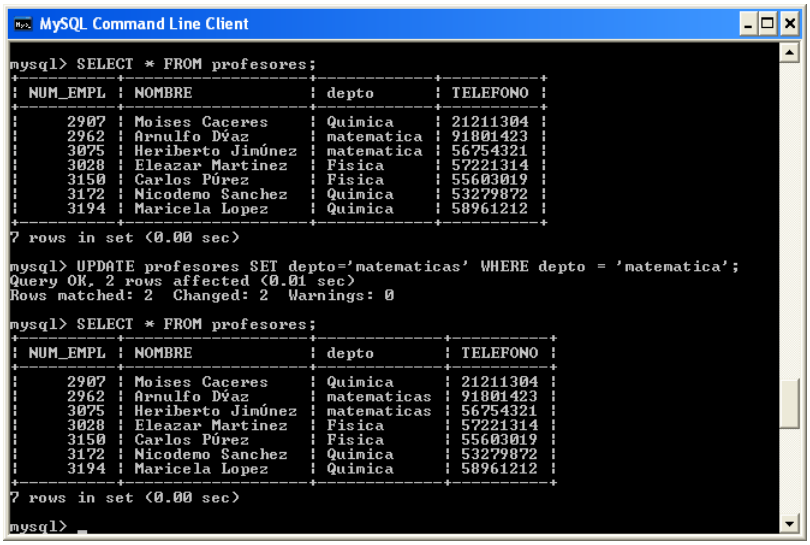


Figura 4.16: Modificación cuando la condición se cumple para varias tuplas.

También es posible realizar operaciones aritméticas, por ejemplo, si queremos multiplicar por 10 todas las calificaciones de la tabla 'alumnos':

mysql> UPDATE alumnos SET promedio = promedio * 10;

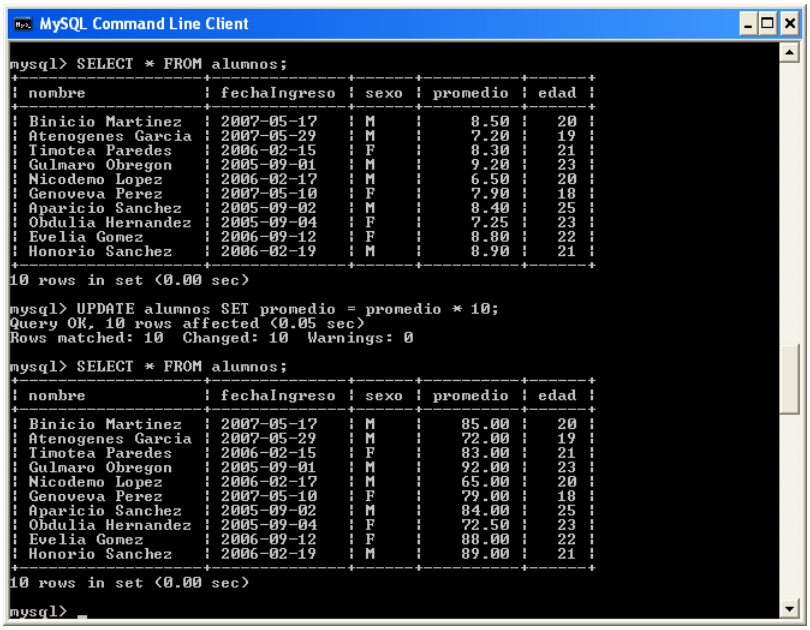


Figura 4.17: Resultado de
UPDATE alumnos SET promedio = promedio * 10;

SQL ofrece la construcción CASE, que se puede usar para formular dos instrucciones de actualización en una única instrucción, la sintaxis es la siguiente:

UPDATE *nombre_tabla*
SET *nombre_columna*=CASE
WHEN *nombre_columna=condición* THEN *valor1*
ELSE *valor2*
END;

Por ejemplo:

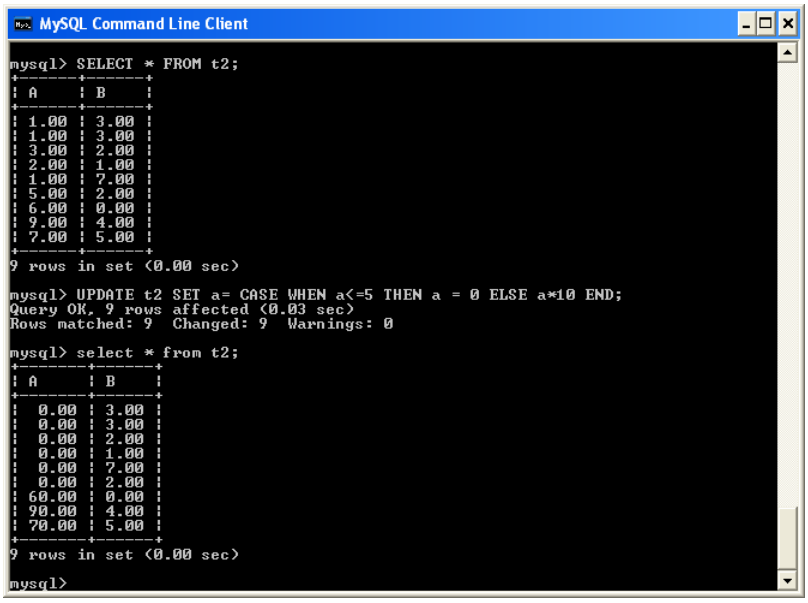


Figura 4.18: Ejemplo de modificación usando CASE.

IV.3.4 Cálculos con fechas.

MySQL provee varias funciones que se aplican a cálculos entre fechas, por ejemplo, para calcular edades u obtener partes de una fecha. La función CURDATE() obtiene la fecha actual Así, por ejemplo, si queremos agregar una nueva columna a la consulta, que se llame "antigüedad" (en años) obteniendo la información de una tabla de alumnos, hacemos la diferencia entre el año actual y el año de su ingreso, por ejemplo:

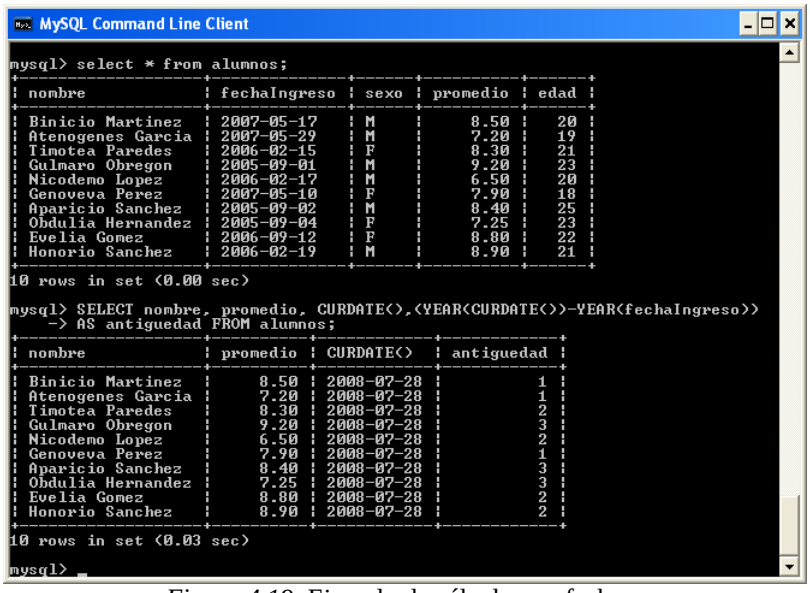


Figura 4.19: Ejemplo de cálculo con fechas.

IV.3.5 Resumen de instrucciones SQL.

- Crear bases de datos
  ```
  mysql> CREATE DATABASE nombre_base_datos;
  ```
- Mostrar bases de datos
  ```
  mysql> SHOW DATABASES;
  ```
- Cambiar a una BD
  ```
  mysql> USE nombre_BD;
  ```
- Mostrar BD actual
  ```
  mysql> SELECT DATABASE();
  ```
- Mostrar tablas de una BD
  ```
  mysql> SHOW TABLES;
  ```
- Mostrar el contenido de una tabla
  ```
  mysql> SELECT * FROM nombre_tabla;
  ```
- Crear una tabla
  ```
  mysql> CREATE TABLE nombre_tabla(
      -> campo1 tipo,
      ::
      -> campoN tipo);
  ```
- Cargar datos en una
  ```
  mysql> LOAD DATA LOCAL INFILE 'path/archivo.txt'
      -> INTO TABLE nombre_tabla;
  ```

- Ver nombres de los campos de una tabla y sus tipos

```
mysql> DESCRIBE nombre_tabla;
```

- Agregar una tupla a una tabla

```
mysql> INSERT INTO nombre_tabla
    -> VALUES( 'valor1',…,'valorN');
```

- Renombrar una tabla

```
mysql> ALTER TABLE nombre1 RENAME nombre2;
```

- Borrar un campo de una tabla

```
mysql> ALTER TABLE nombre DROP columna;
```

- Modificar un tipo de dato

```
mysql> ALTER TABLE nom_tabla
    -> MODIFY  columna tipo;
```

- Cambiar nombre y tipo a una columna

```
mysql> ALTER TABLE nom_tabla
    -> CHANGE nom1_col  nom2_col  tipo;
```

- Añadir un campo a una tabla

```
mysql> ALTER TABLE nom_tabla
    -> ADD campo tipo_dato;
```

- Borrar una tabla

```
mysql> DROP TABLE  nom_tabla;
```

- Borrar una base de datos

```
mysql> DROP DATABASE nom_BaseDatos;
```

- Borrar una tupla

```
mysql> DELETE FROM nom_tabla
    -> WHERE condición;
```

- Modificar tuplas

```
mysql> UPDATE nom_tabla
    ->  SET nom_columna = nuevo_valor
    -> WHERE condición;
```

- Ordenar tuplas

```
mysql> SELECT campo1,…,campoN
    -> FROM nom_tabla
    ->  GROUP BY nom_columna ;
```

Capítulo V Integridad relacional.

V.1 Definición de Integridad relacional.

Una de las características más importantes del modelo relacional de Codd es que "desintegra" los datos que representan a una entidad y los reparte en varias tablas. Para recuperar todos los datos de la entidad es necesario extraerlos de éstas tablas. El término integridad en las bases de datos se refiere a asegurar que los datos sean válidos. La *integridad relacional* es el conjunto de reglas que asegura que se podrán extraer correctamente todos los datos para volver a formar la entidad.

Las tablas que integran la base de datos deben cumplir con *reglas de integridad* que aseguren que los datos que contienen sean exactos y estén siempre accesibles.

Para garantizar la integridad relacional se usan las *restricciones* de dominio, y las *reglas de integridad*.

"Las restricciones de integridad garantizan que las modificaciones realizadas en la base de datos por los usuarios autorizados no den lugar a una pérdida de la consistencia de los datos" [Silberschatz, 2007].

V.2 Restricciones de dominio.

Las *restricciones de dominio* especifican el conjunto de valores y de operaciones permitidas sobre una o columna o dominio. En otras palabras, permiten especificar el rango de un campo en particular.

En versiones avanzadas (MySQL 6.0, SQL92, InterBase, Oracle, etc.) se usan las cláusulas CONSTRAINT y CHECK.

```
CREATE TABLE  tabla (
   clave        NUMERIC(5,0),
   promedio     NUMERIC(4,2),

   CONSTRAINT promedio_valido CHECK (promedio>=0 AND promedio<= 10)
);
```

La cláusula CONSTRAINT promedio_valido es optativa, y se emplea para dar el nombre promedio_valido a la restricción.

V.3 Reglas de integridad.

Las reglas de integridad minimizan los errores de entrada de datos y aseguran que los cambios realizados a una base de datos no provoquen inconsistencia en la información.

Una de las reglas de integridad dice que todos los renglones (filas o tuplas) de una tabla deben ser distintos entre sí. Si hay renglones duplicados puede haber problemas para elegir alguno de los dos. En la mayoría de los sistemas de gestión de bases de datos (SGBD) el usuario puede especificar que no se permiten los renglones duplicados, de tal manera que el SGBD impedirá la adición de una nueva tupla si ésta ya existe dentro de la tabla. Para especificar que no debe haber renglones duplicados en SQL se utiliza la cláusula **DISTINCT.**

Hay una regla de integridad que contempla las situaciones en las que no hay un dato disponible, en este caso se utiliza el concepto de *valor nulo*, el cual indica que no existe el dato. Es importante hacer notar que el valor nulo no es lo mismo que el valor *cero* o el *espacio en blanco*. Dos valores *cero* o dos *espacios en blanco* se consideran iguales. Mientras que dos *valores nulos* se consideran diferentes entre sí.

Cuando cada uno de los renglones de una tabla es diferente de los otros, se puede utilizar una o más columnas para identificar un renglón en particular. Esta columna o grupo de columnas conforma la *clave primaria*. Para indicar que un atributo es la clave primaria se utiliza la cláusula **PRIMARY KEY**, ésta puede usarse a nivel de atributos o a nivel tabla.

Una importante regla llamada *integridad de entidad* dice que los valores de la(s) columna(s) de la clave primaria no deben tener valores nulos.

Una columna o atributo puede o no admitir valores nulos. Para especificar en SQL que no se permiten los valores nulos se utiliza la cláusula **NOT NULL**.

Mediante una clave primaria indicamos que una columna, o una combinación de columnas deben tener valores únicos para cada fila. Por ejemplo, en una tabla de clientes, el código de cliente no debe repetirse en dos filas diferentes. Esto se expresa de la siguiente forma:

```
CREATE TABLE Clientes(
  Codigo integer NOT NULL PRIMARY KEY,
  Nombre varchar(30) NOT NULL,
  /* ... */
);
```

Si una columna pertenece a la clave primaria, debe estar especificada como no nula. Nótese que en este caso hemos utilizado la restricción a nivel de columna. También es posible tener claves primarias compuestas, en cuyo caso la restricción hay que expresarla a

nivel de tabla. Por ejemplo, en la tabla de detalles de pedidos, la clave primaria puede ser la combinación del número de pedido y el número de línea dentro de ese pedido:

```
CREATE TABLE Detalles(
   NumPedido integer NOT NULL,
   NumLinea integer NOT NULL,
   /* ... */
   PRIMARY KEY (NumPedido, NumLinea)
  );
```

Solamente puede haber una clave primaria por cada tabla. De este modo, la clave primaria representa la *identidad* de los registros almacenados en una tabla: la información necesaria para localizar un objeto. No es imprescindible especificar una clave primaria al crear tablas, pero es recomendable como método de trabajo.

En suma, no habrá entradas nulas en la llave primaria y todas las entradas serán únicas.

V.4 Integridad Referencial.

Como ya se vio en el capítulo II, una *interrelación* es una asociación entre tablas que se establece mediante la clave externa de una tabla (tabla hija o tabla dependiente) y la clave principal de la otra (tabla padre o tabla maestra). La *clave externa* es un atributo o conjunto de atributos de la tabla dependiente cuyos valores se corresponden con la clave principal de la tabla maestra. Los dominios de ambas claves deben ser compatibles. Las claves externas son un elemento primordial para garantizar la integridad de las bases de datos.

El valor de la clave externa en una tabla dependiente puede ser nulo o debe corresponder a uno de los valores de una clave primaria de la tabla maestra con la cual se establece la relación.

Para expresar esto se utilizan las cláusulas **REFERENCES** y **FOREIGN KEY**.

Por ejemplo, en una tabla de pedidos se almacena el código del cliente que realiza el pedido. Este código debe corresponder al código de algún cliente almacenado en la tabla de clientes. La restricción puede expresarse de la siguiente manera:

```
CREATE TABLE Pedidos(
Codigo integer NOT NULL PRIMARY KEY,
Cliente integer NOT NULL REFERENCES Clientes(Codigo),
/* ... */
);
```

O, utilizando restricciones a nivel de tabla:

```
CREATE TABLE Pedidos(
Codigo integer NOT NULL,
Cliente integer NOT NULL,
/* ... */
PRIMARY KEY (Codigo),
FOREIGN KEY (Cliente) REFERENCES Clientes(Codigo)
);
```

La columna a la que se hace referencia en la tabla dependiente Pedidos, la columna *Codigo* de Clientes en este caso, debe ser la clave primaria de la tabla maestra Clientes.

Además del ejemplo del capítulo III de las tablas de Profesores y de Grupos, se presenta otro ejemplo de *interrelación*: en este caso tenemos parte de una base de datos para un banco que contiene la tabla Clientes en la que se encuentran los datos personales de los clientes y la tabla Cuentas en la que se lleva el registro de las cuentas que tienen los clientes en el banco. Cada uno de los clientes puede tener una o varias cuentas, la clave primaria de Clientes es el RFC, el atributo RFC también se encuentra en la tabla Cuentas y es la *clave externa* que hace referencia al cliente de Clientes y con la que se pueden encontrar los datos del cliente al que pertenece una cuenta en particular. En este ejemplo, de la *figura 5.1*, la clave externa consiste en una sola columna, sin embargo, en el caso general se puede tener una clave externa compuesta.

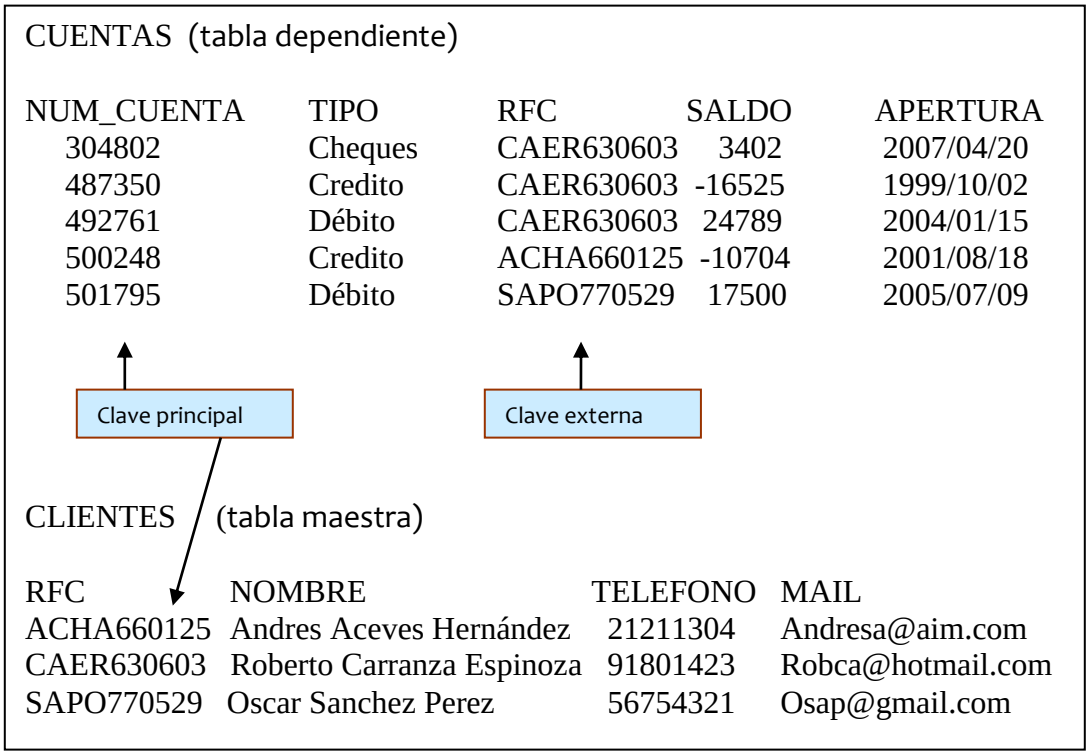


Figura 5.1: Ejemplo de interrelación de tablas.

La integridad referencial garantiza la existencia de las claves externas. En este caso se debe asegurar que el RFC que se ingrese en uno de los renglones de CUENTAS se encuentre dentro de la tabla CLIENTES. En general, no se puede insertar ni modificar una clave externa con un valor que no se encuentre en la tabla maestra.

¿Cómo se crean las tablas Cuentas y Clientes descritas anteriormente garantizando la integridad relacional?

Solución:
Primero hay que crear la base de datos en donde se guardarán las tablas (*figura 5.2*):

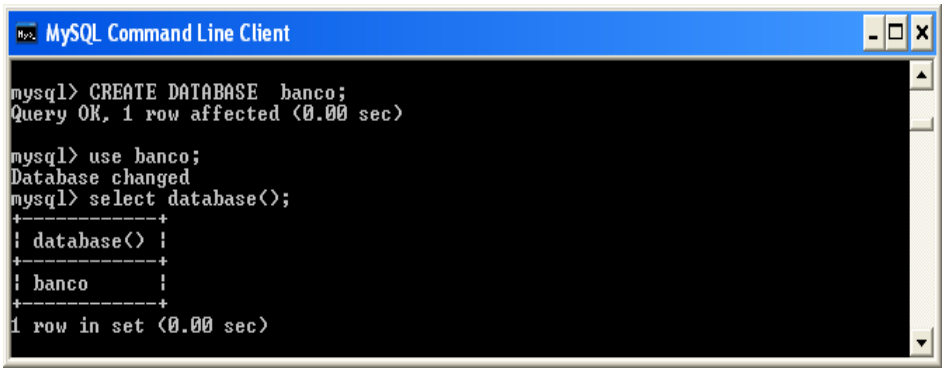


Figura 5.2: Creación de la base de datos "banco".

Después creamos las tablas Cuentas y Clientes (*figura 5.3*):

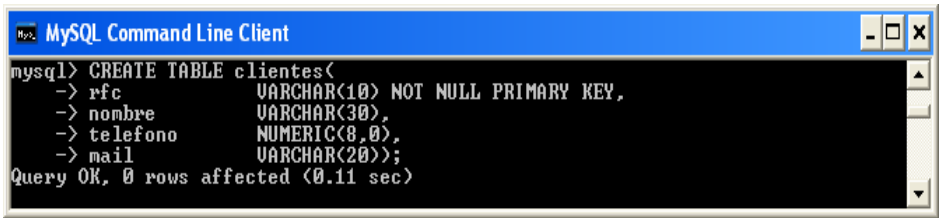


Figura 5.3: Creación de la tabla "clientes".

El campo rfc es la *clave primaria* de la tabla clientes, se indicó también que no se aceptan valores nulos para este campo (*figura 5.4*).

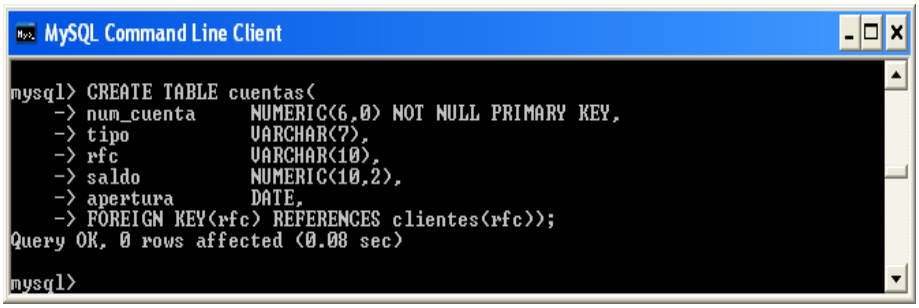


Figura 5.4: Creación de la tabla "cuentas".

El campo num_cuenta es la *clave primaria* de la tabla cuentas, y el campo rfc es la *clave externa* que hace referencia a la tabla clientes.

Podemos ver la estructura de las tablas creadas como se muestra en la *figura 5.5*:

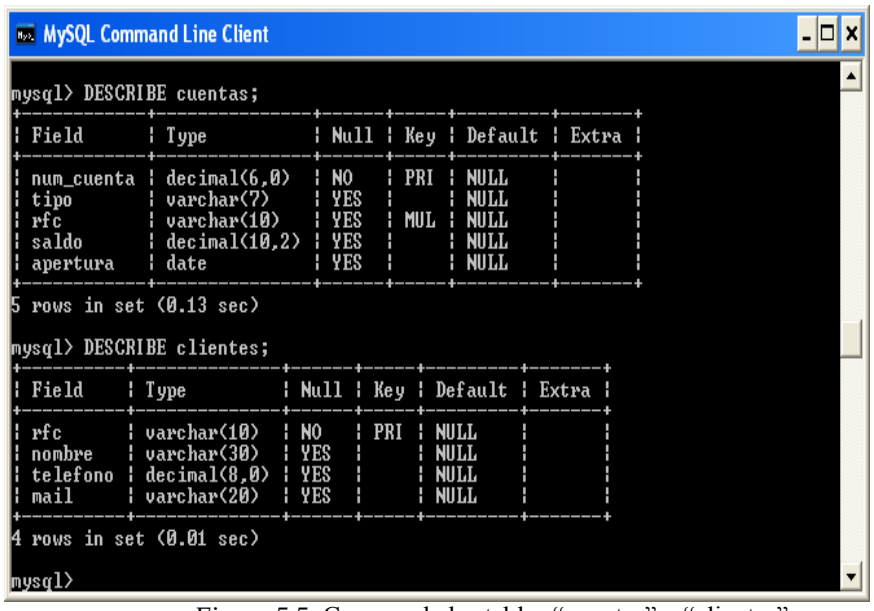


Figura 5.5: Campos de las tablas “cuentas” y “clientes”.

Una vez que ponemos el contenido en las tablas, éste se puede observar con SELECT (*figura 5.6*):

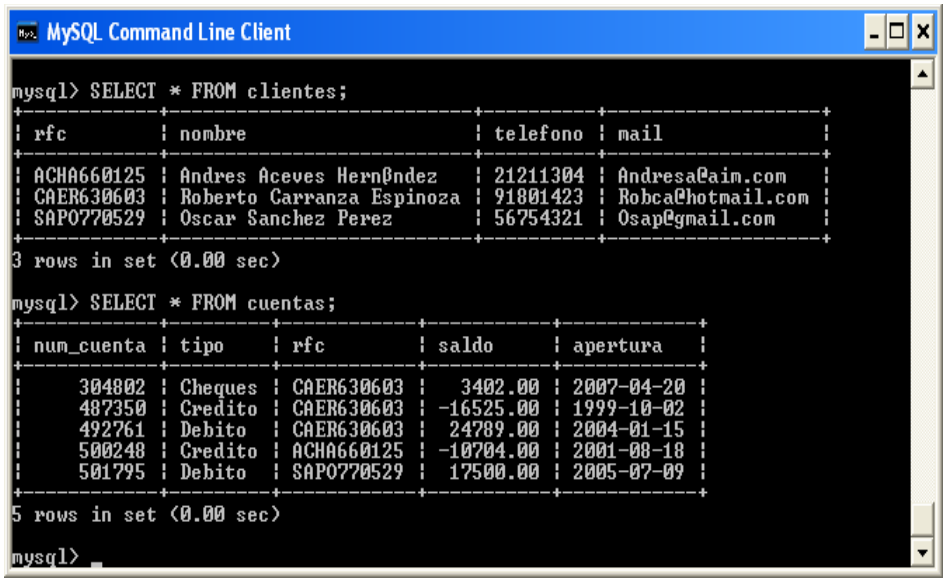


Figura 5.6: Contenido de “clientes” y “cuentas”.

Si quisiéramos agregar en la tabla cuentas el registro:

517203, Cheques, MANS810923,4000,2008-05-29

El Sistema Administrador de la Base de Datos, no nos permite hacerlo, ya que no existe ningún cliente con ese RFC (*figura 5.7*).

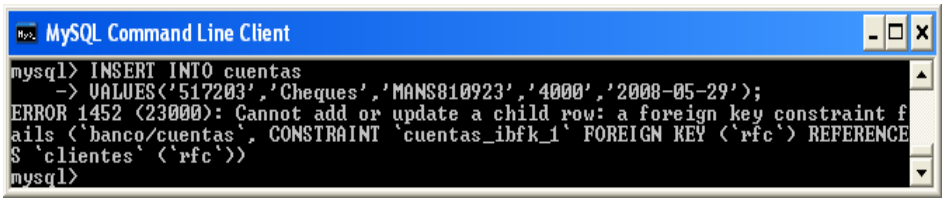


Figura 5.7: Inserción no exitosa de una tupla.

Para poder agregar el registro en la tabla cuentas primero hay que "dar de alta" al nuevo cliente en la tabla clientes:

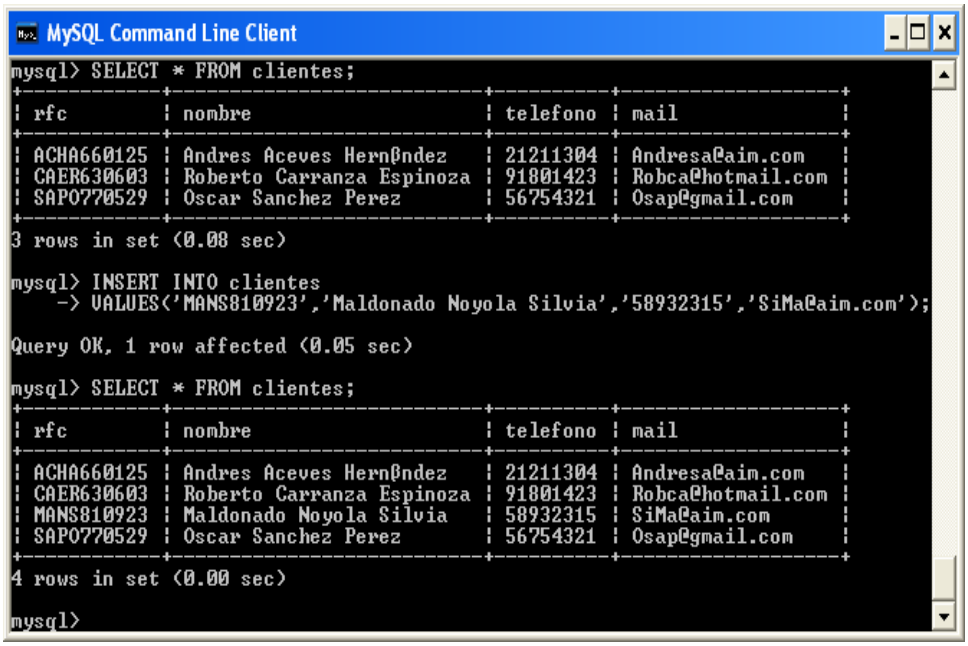


Figura 5.8: Inserción del nuevo cliente.

Entonces ahora si es posible agregar en la tabla cuentas el registro:

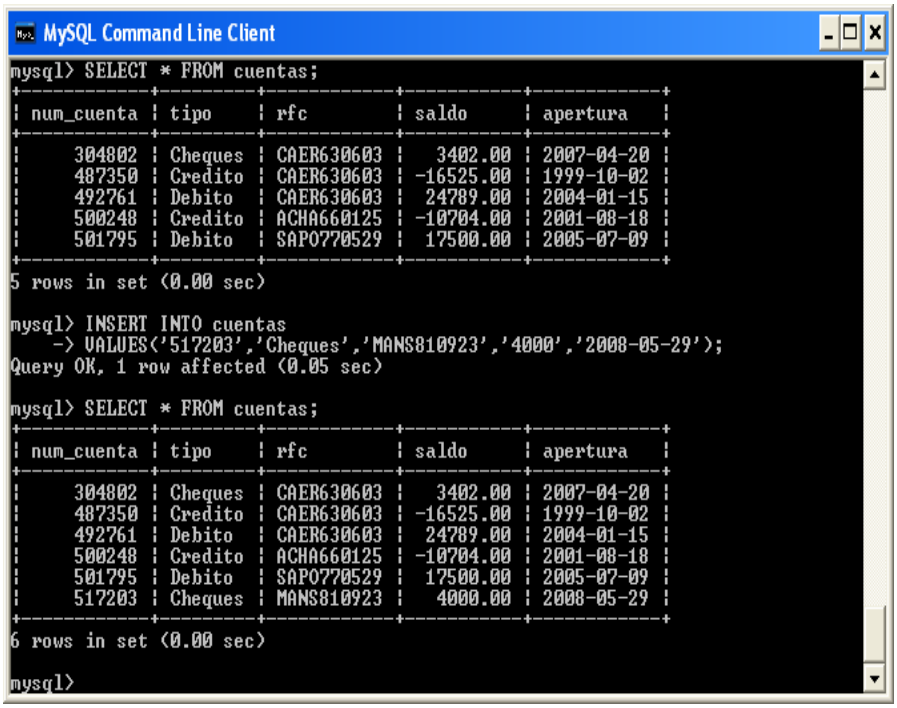


Figura 5.9: Inserción no exitosa de una tupla en la tabla "cuentas".

V.4.1 Acciones referenciales.

Cuando se viola una restricción de integridad referencial, el procedimiento normal es rechazar la acción que ha causado la violación. Por ejemplo, si queremos borrar a un cliente de la tabla Clientes, que tiene ya una cuenta, el sistema no lo permitirá ya que en la tabla Cuentas hay una referencia a ese cliente (*figura 5.10*):

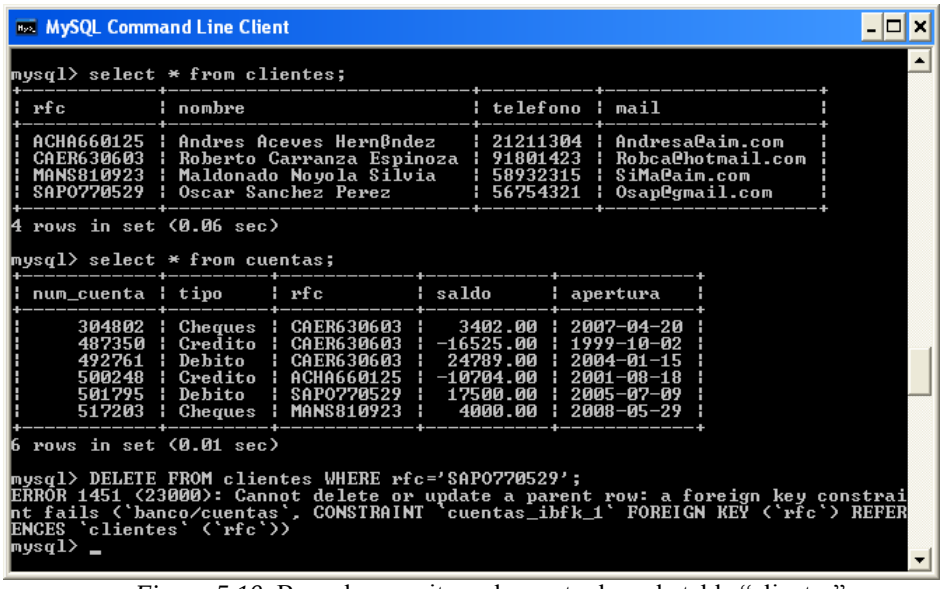


Figura 5.10: Borrado no exitoso de una tupla en la tabla "clientes".

Por el contrario, si el cliente que deseamos borrar de la tabla Clientes, no tiene una cuenta asociada en la tabla Cuentas, entonces el sistema nos permite borrarlo sin problema (*figura 5.11*):

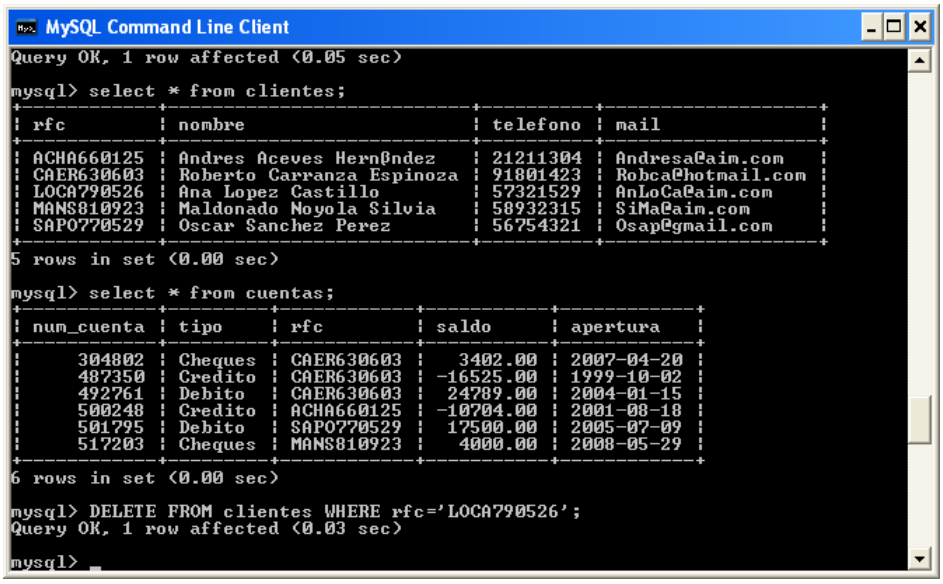


Figura 5.11: Borrado no exitoso de una tupla en la tabla "clientes" que no está asociada a la tabla "cuentas".

.

Una tabla maestra debe asumir su responsabilidad cuando alguien intenta borrar alguna de sus filas o modificar el valor de alguna de sus claves primarias, si se trata de borrado son tres los comportamientos posibles:

- Borrar la información en cadena (borrar en cascada): **CASCADE.**
- No permitir el borrado: **RESTRICT.**
- Permitir el borrado y en lugar de borrar las filas dependientes, convertir los valores de la clave externa en nulos, rompiendo así el vínculo entre las tablas: **SET NULL.**

Siguiendo con el ejemplo anterior, vamos a crear la tabla Cuentas, pero ahora especificando que la acción que se debe llevar a cabo es borrar o actualizar en cascada las tuplas de la tabla padre (Clientes, en este caso).

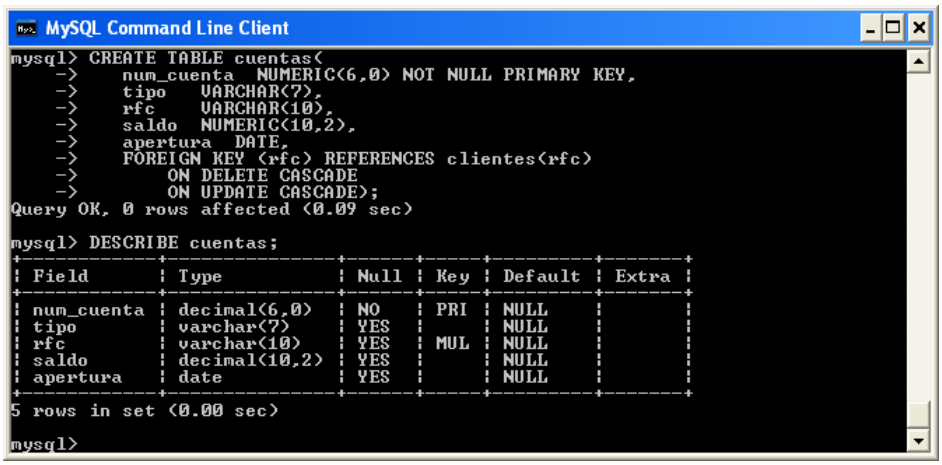


Figura 5.12: Especificación de la acción referencial durante la creación de la tabla.

Si el contenido de las tablas Cuentas y Clientes es el de la *figura 5.13*:

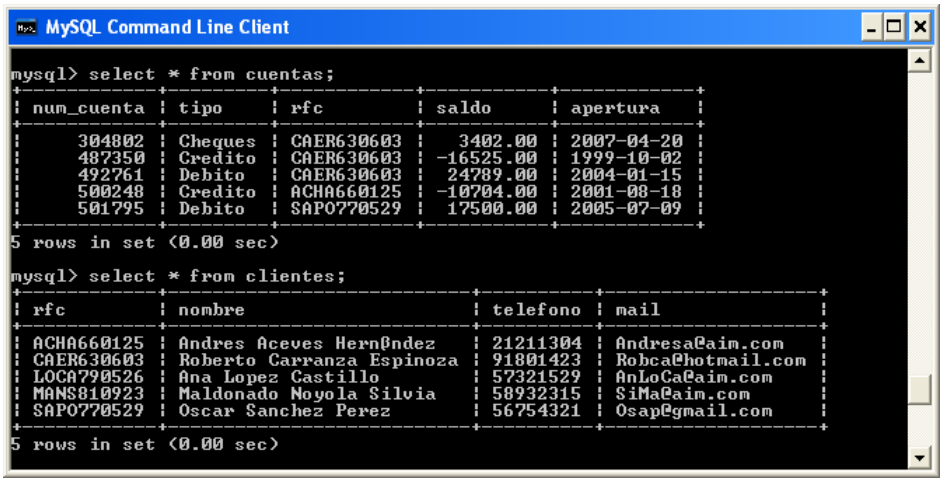


Figura 5.13: Contenido de "cuentas" y "clientes".

Y queremos modificar el RFC de la tabla Cuentas el SGBD no lo permite, ya que es una tabla hija:

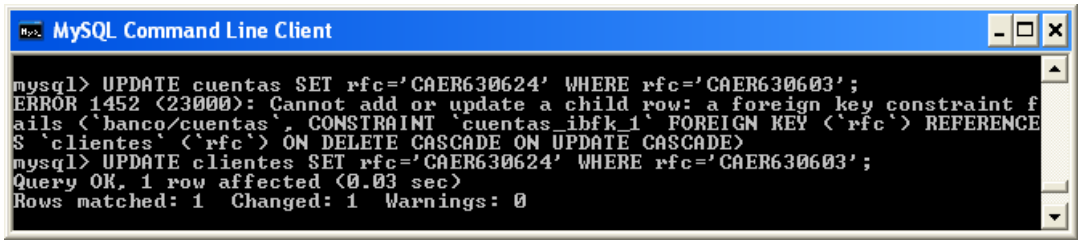


Figura 5.14: Modificación no exitosa de una tupla debido a una acción referencial.

Sin embargo, como especificamos ON UPDATE CASCADE en Cuentas el valor del rfc se modifica automáticamente en cascada al hacer la actualización en la tabla padre Clientes:

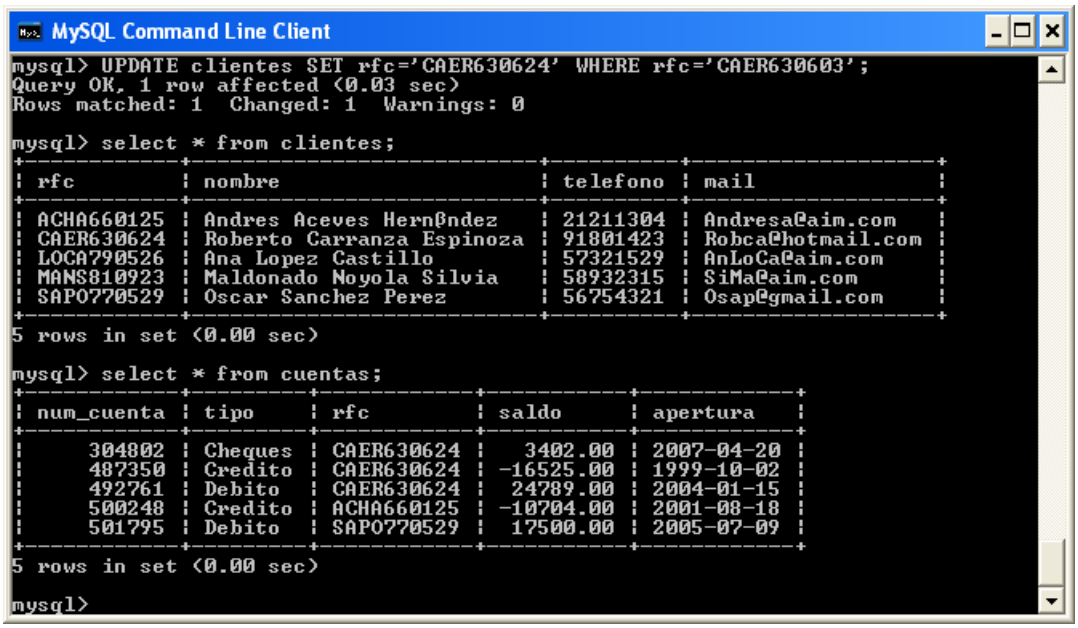


Figura 5.15: Modificación exitosa de una tupla (en cascada).

También es posible declarar diferentes acciones referenciales, por ejemplo:

ON DELETE SET NULL
ON UPDATE CASCADE

Lo anterior hace que la clave externa de la tabla hija (o dependiente) se ponga en NULL, cuando se borre la tupla correspondiente en la tabla padre (o maestra), mientras que la actualización sigue funcionando "en cascada".

V.5 Ejercicios.

Ejercicio 1.- Suponiendo que tenemos una base de datos de un aeropuerto

- Diseña dos tablas llamadas "líneas aéreas" y "destinos" inventando diferentes campos para cada tabla.
- Uno de los campos de cada tabla debe ser la clave principal.
- Interrelaciona las dos tablas haciendo que la clave primaria de destinos sea una clave externa de la tabla líneas aéreas.

Solución:

Este es un ejercicio abierto, en el que el lector puede plantear diversas soluciones, a continuación se presentan 2 ejemplos.

Ejemplo 1 de solución.

Destinos

Clave Primaria

ID_Destino	NombreDest
CUN	Cancun
MER	Merida
TXT	Tuxtla

Destinos es la tabla padre

Lineas_Aereas

Clave Primaria: Clave_Destino + Clave_linea

		Clave Externa
ID_Linea	**Nombre**	**ID_Destino**
MEX	Mexicana	CUN
MEX	Mexicana	MER
AER	Aeromexico	MER
AVC	Aviacsa	TXT
ITRJ	Interjet	CUN

Figura 5.16: primera solución al ejercicio 1.

Ejemplo 2 de solución.

Lineas_Aereas
Clave Primaria

Clave_Linea	**Nombre**
MEX	Mexicana
AER	Aeromexico
AVC	Aviacsa
ITRJ	Interjet

Lineas_Aereas es la tabla padre

Destinos
Clave Primaria: Clave_Destino + Clave_linea
Clave Externa

Clave_Destino	**Clave_linea**	**NombreDest**	**horario**
CUN	MEX	Cancun	matutino
CUN	ITRJ	Cancun	vespertino
MER	MEX	Merida	matutino
MER	AER	Merida	vespertino
TXT	AVC	Tuxtla	matutino

Figura 5.17: Segunda solución al ejercicio 1.

En los dos ejemplos anteriores tenemos un problema: se repiten no solo las claves del destino, sino también el nombre del destino. ¿Qué soluciones propones?

Respuesta

Hacer una tabla aparte que puede llamarse CODIGO_DESTINOS con los campos Clave_Destino y NombreDest, así se elimina este último campo de la tabla DESTINOS

Ejercicio 2.- Diseñar dos tablas para una base de datos que controla las computadoras de la UAM.

- La primera tabla se llama **Computadoras** y debe contener:
  - La marca, el número de serie, capacidad de memoria RAM y tipo (escritorio, LapTop). Puedes agregar campos adicionales.
- La segunda tabla se llama **Inventario** y debe contener:
  - la información de la División a la que pertenece cada computadora (CNI, CDD, CSH)
  - el sistema operativo que tiene (Unix, Windows XP, Windows Vista)
  - y el estado en el que se encuentra (Funcionando, Descompuesta).
- Determinar los atributos de cada tabla.
- Indicar que atributo es la clave primaria de la tabla 1.
- La clave primaria de la tabla 1 deberá ser también la clave primaria de la tabla 2.
- Inventar 3 tuplas de cada tabla.

Solución - Ejemplo:

Computadoras

	Clave Primaria				
Marca	**No. Serie**	**Procesador**	**DiscoDuro**	**RAM**	**Tipo**
Dell	39827	AMD	120 GB	2GB	Escritorio
HP	59821	Core Duo	300 GB	1GB	LapTop
Compaq	36915	Sparc	60 GB	512MB	Escritorio

Inventario

Clave Externa

No. Serie	**División**	**SistemaOperativo**	**Estado**
59821	CDD	Windows Vista	Funcionando
39827	CNI	Windows XP	Funcionando
36915	CSH	Unix	Descompuesta

Ejercicio 3.- Diseñar dos tablas para una base de datos para una tienda de abarrotes.

- La primera tabla se llama **Ventas** y contiene el código de la venta, se registra una venta por cada producto diferente, una venta puede contener varios artículos del mismo producto, también debe registrarse la ganancia por la venta hecha.
- La segunda tabla se llama **Productos** y debe contener el código de cada producto, su descripción, su costo y su precio de venta.
- Indicar que atributo es la clave primaria de la tabla 1.
- La clave primaria de la tabla 2 deberá ser la clave externa de la tabla 1.

- Inventar 3 tuplas de cada tabla.

Solución - Ejemplo:

Ventas

Clave Primaria	**Clave Externa**		
CodVenta	**CodProducto**	**ArticulosVendidos**	**Ganancia**
Marzo17-1	25032	3	96
Marzo17-2	20198	1	13
Marzo17-3	31456	10	20

Productos

Clave Primaria			
CodProducto	**Descripción**	**Costo**	**PrecioVenta**
20198	Detergente	11	24
25032	Aceite	19	32
31456	Leche	9	11

Ejercicio 4.- Diseñar tres tablas para la base de datos de un hospital.

- La primera tabla se llama **Pacientes** en ésta se encuentran todos los nombres de los pacientes junto con sus datos personales (Edad, Sexo, Teléfono). Cada paciente tiene asignada una clave llamada "ClavePaciente".
- La segunda tabla se llama **Doctores** y en esta se encuentran los datos de cada doctor con su respectivo nombre, número de empleado y especialidad.
- La tercera tabla se llama **Internos** y debe contener la clave del paciente, la enfermedad diagnosticada (Diagnóstico), el número de empleado del doctor que lo atiende, el número de habitación en la que se encuentra internado y el estado en el que se encuentran (Alta, baja, interno)
- Indicar cual es la clave primaria de la tabla **Pacientes.**
- Indicar cual es la clave primaria de la tabla **Doctores.**
- La tabla **Internos** tiene dos claves externas: ¿Cuáles son?
- Inventar 3 tuplas para cada tabla.

Solución - Ejemplo:

Pacientes

Clave Primaria				
ClavePaciente	**Nombre**	**Edad**	**Sexo**	**Teléfono**
0752	Eleazar Martínez	8	M	58263214
0836	Obdulia Sánchez	25	F	52347025
0911	Isidoro Hernández	41	M	91148302

Doctores

Clave Primaria		
NumDoctor	**Nombre**	**Especialidad**
22341	Prudencio Mata	Cancerología
24932	Raúl Díaz	Ginecología
25480	Leonor Barroso	Pediatría

Internos

Clave Externa		**Clave Externa**		
ClavePaciente	**Enfermedad**	**NumDoctor**	**Habitación**	**Estatus**
0911	Cáncer pulmonar	22341	521	Baja
0752	Apendicitis	25480	203	Alta
0836	Aborto	24932	415	Interno

Capítulo VI Álgebra relacional y SQL.

VI.1 Introducción al álgebra relacional.

Las operaciones de álgebra relacional manipulan relaciones. Esto significa que estas operaciones usan una o dos relaciones existentes para crear una nueva relación. Esta nueva relación puede entonces usarse como entrada para una nueva operación. Este poderoso concepto: *la creación de una nueva relación a partir de relaciones existentes*, hace posible una variedad infinita de manipulaciones sobre los datos, y también hace considerablemente más fácil la solución de las consultas, debido a que se puede experimentar con soluciones parciales hasta encontrar la proposición con la que se trabajará.

Las tablas resultantes sólo se generan en memoria temporal para visualizar sus datos u operar con ellas, pero nunca se almacenan en memoria permanente a no ser que se indique expresamente.

Entender las operaciones del algebra relacional es la clave para entender los lenguajes de consulta relacionales (SQL)

El álgebra relacional consta de nueve operaciones, las cuáles se clasifican en tres tipos:

Las operaciones de origen matemático: se toman de la teoría de conjuntos, las relaciones en sí mismas son conjuntos, así es que se les pueden aplicar las operaciones de conjunto, las cuatro operaciones son:

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

Las operaciones del lenguaje relacional: son operaciones nuevas, específicas del modelo relacional, a continuación se enlistan las cuatro operaciones:

- Selección.
- Proyección.
- Reunión (join).

- División.

La operación de los lenguajes de programación: esta operación es estándar de los lenguajes de computación y consiste en darle valor a un nombre, la operación en cuestión es:

- Asignación.

En este caso, la asignación se usa para dar un nombre a una nueva relación que se crea de relaciones existentes. Se usará ":=" para indicar la asignación de nombres a relaciones. A continuación se explica en que consiste cada operación por grupos.

Algebra Relacional y SQL

La propiedad fundamental del álgebra relacional es la de cierre. Esto significa que el resultado de cualquier expresión algebraica es una relación. SQL maneja un operador llamado GROUP BY (agrupar por). El resultado de utilizar la cláusula GROUP BY, en alguna consulta, no es otra tabla, sino que un conjunto de tablas, no cumpliéndose el principio de clausura. Hay aspectos del modelo relacional que no están soportados en SQL, esto constituye un problema y un reto para el desarrollo futuro de SQL.

VI.2 Operaciones de origen matemático.

VI.2.1 Unión.

La operación **unión** (∪**)** permite combinar los datos de dos relaciones. El resultado de aplicar una operación de unión es una tabla formada con las columnas de una de ellas y las filas de ambas tablas, excluyendo las que sean duplicadas, que sólo aparecen una vez. La unión de las relaciones T1 y T2 es el conjunto de tuplas que están en T1, en T2 o en ambas.

Las dos tablas deben tener el mismo grado y los dominios de sus atributos deben ser compatibles. En la *figura 6.1* se ilustra gráficamente la unión de dos tablas.

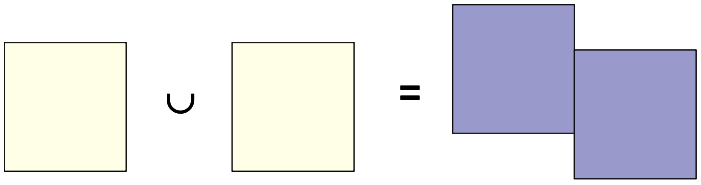


Figura 6.1: Unión de dos tablas.

En la *figura 6.2* se da un ejemplo concreto de la unión de dos tablas.

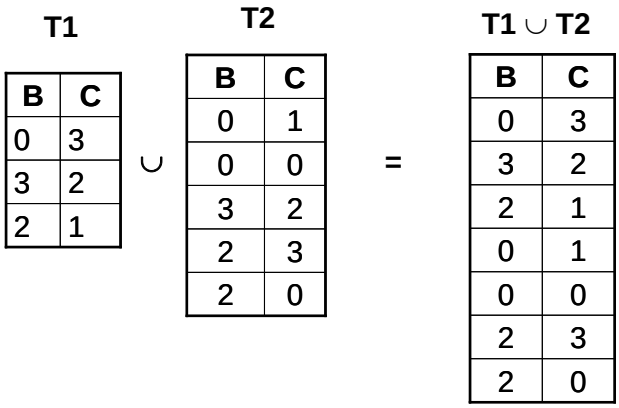


Figura 6.2: Ejemplo concreto de la unión de dos tablas.

La sintaxis de la **unión** en SQL es la siguiente:

(SELECT *<lista de atributos>*
FROM *nombre_ tabla1*)
UNION
(SELECT *<lista de atributos>*
FROM *nombre_ tabla2*)

Por ejemplo:

```
mysql>  (SELECT * FROM T2) UNION (SELECT * FROM T3);
```

En la *figura 6.3* se hizo la unión de las dos tablas T2 y T3.

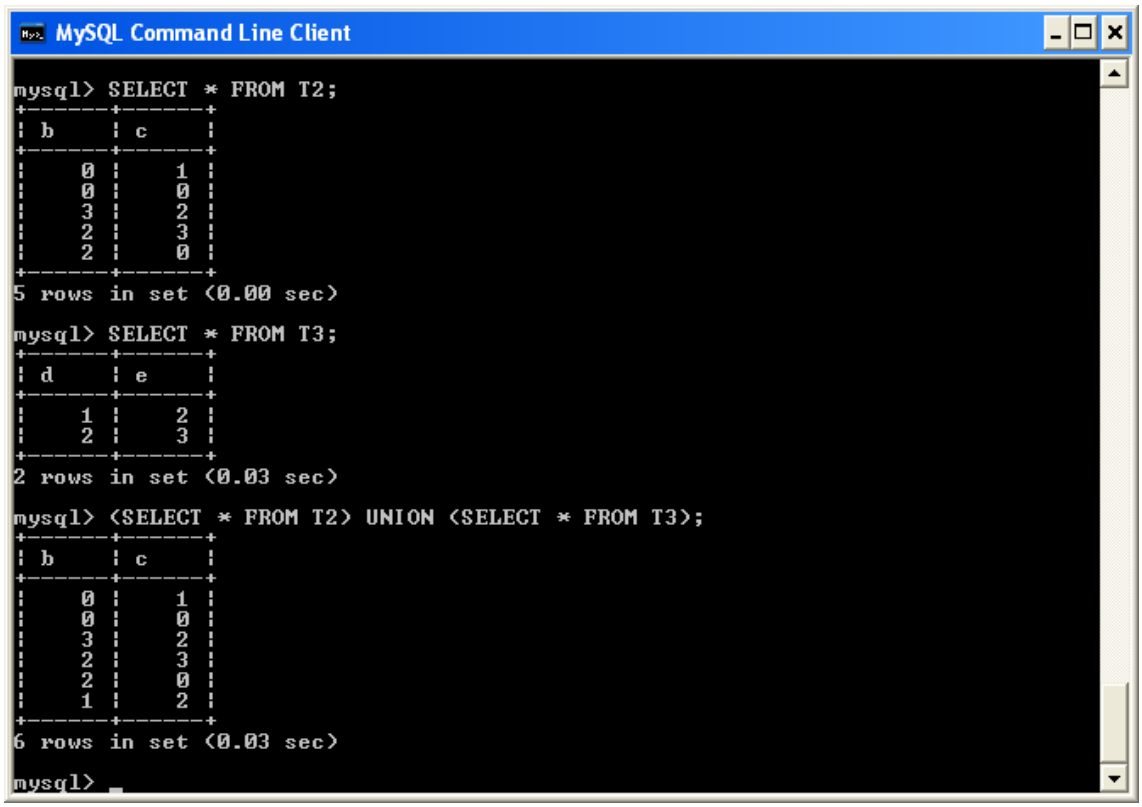


Figura 6.3: Unión de dos tablas en MySQL.

VI.2.2 Intersección.

La operación **intersección** (∩ **)** permite identificar las tuplas que son comunes a dos relaciones. Al igual que para la unión, las dos tablas deben tener el mismo grado y los dominios de sus atributos deben ser compatibles.

La intersección de las relaciones T1 y T2 es el resultado de las tuplas comunes en ambas tablas, la nueva tabla tiene las columnas de una de ellas y las filas comunes a ambas tablas. La intersección de las relaciones T1 y T2: T1 ∩ T2 es el conjunto de tuplas que están tanto en T1 como en T2. En la *figura 6.4* se ilustra gráficamente la intersección de dos tablas.

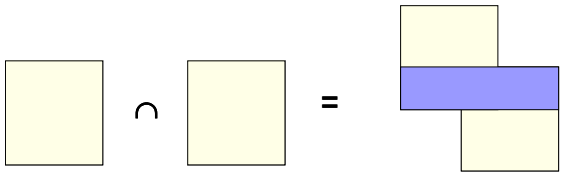


Figura 6.4: Intersección de dos tablas.

Un ejemplo concreto de intersección es el de la *figura 6.5*:

T1

B	C
0	3
3	2
2	1

T2

B	C
0	1
0	0
3	2
2	3
2	0

T1 ∩ T2

B	C
3	2

Figura 6.5: Ejemplo concreto de la intersección de dos tablas.

La sintaxis de la **intersección** en SQL es la siguiente:

(SELECT *<lista de atributos>*
FROM *nombre_ tabla1*)
INTERSECT
(SELECT *<lista de atributos>*
FROM *nombre_ tabla2*)

Por ejemplo:

```
mysql>  (SELECT * FROM T1) INTERSECT (SELECT * FROM T2);
```

Nota: este operador no está implantado en MySQL.

VI.2.3 Diferencia.

La operación **diferencia** (-) permite identificar tuplas que están en una tabla T1 que no se encuentran en otra tabla T2. De la misma manera que las operaciones anteriores, las dos tablas deben tener el mismo grado y los dominios de sus atributos deben ser compatibles.

En otras palabras, sean dos relaciones A y B, la diferencia de A - B es el conjunto de tuplas que están en A, pero que no están en B. En la *figura 6.6* se ilustra gráficamente la diferencia de dos tablas.

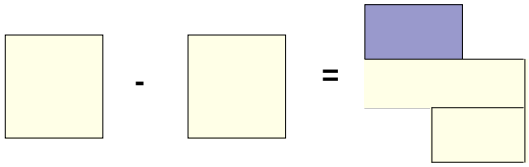


Figura 6.6: Diferencia de dos tablas.

La *figura 6.7* es un ejemplo concreto de la diferencia de dos tablas:

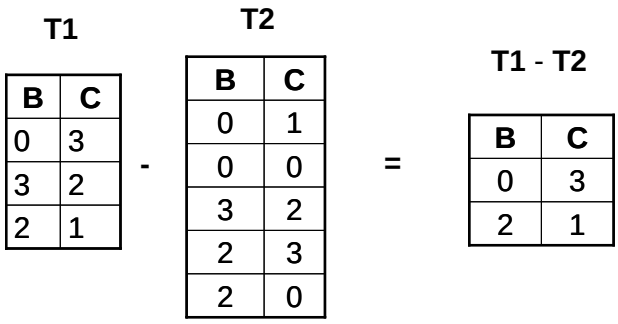


Figura 6.7: Ejemplo concreto de la diferencia de dos tablas.

La sintaxis de la **diferencia** en SQL es la siguiente:

```
(SELECT  <lista de atributos>
FROM     nombre_ tabla1)
MINUS
(SELECT  <lista de atributos>
FROM     nombre_ tabla2)
```

Por ejemplo:

```
mysql>  (SELECT * FROM T1) MINUS (SELECT * FROM T2);
```

Nota: este operador no está implantado en MySQL.

VI.2.4 Producto cartesiano.

La operación **producto cartesiano**, que se indica por el símbolo ***** o por el símbolo **x**, es idéntica a la operación en matemáticas que crea el producto cartesiano de dos conjuntos. El producto de T1* T2 se crea:

1.- Uniendo los campos (columnas) de las dos relaciones.
2.- Concatenando todas las tuplas (filas) de T1, con cada una de las tuplas de T2.

Las tablas T1 y T2 pueden ser de grados diferentes (diferente numero de columnas), la tabla resultante tiene la suma de las columnas de T1 más las de T2, y su cardinalidad será el resultado de multiplicar la cardinalidad de T1 por la cardinalidad de T2.

Si hacemos que:

T3 := T1 * T2

Entonces los campos de T3 son todos los campos de T1 y T2 juntos, y el número de tuplas de T3 será el número de filas de T1 por el número de filas de T2. En la *figura 6.8* se muestra un ejemplo del producto cartesiano de dos tablas.

Ejemplo 1:

T1

A
a1
a2

x

T2

B
b1
b2
b3

=

T1 x **T2**

A	B
a1	b1
a1	b2
a1	b3
a2	b1
a2	b2
a2	b3

Figura 6.8: Ejemplo 1 del producto cartesiano de dos tablas.

En la *figura 6.9* se muestra otro ejemplo del producto cartesiano de dos tablas.

T1

B	C
0	3
3	2
2	1

x

T2

D	E
0	1
0	0

=

T1 x **T2**

B	C	D	E
0	3	0	1
0	3	0	0
3	2	0	1
3	2	0	0
2	1	0	1
2	1	0	0

Figura 6.9: Ejemplo 2 del producto cartesiano de dos tablas.

La sintaxis del **producto cartesiano** en SQL es la siguiente:

SELECT *<lista de atributos>*
FROM *nombre_ tabla1*
CROSS JOIN *nombre_ tabla2*

En la *figura 6.10* se muestra un ejemplo con MySQL del producto cartesiano de dos tablas.

```
mysql>  SELECT * FROM t1 CROSS JOIN t2;
```

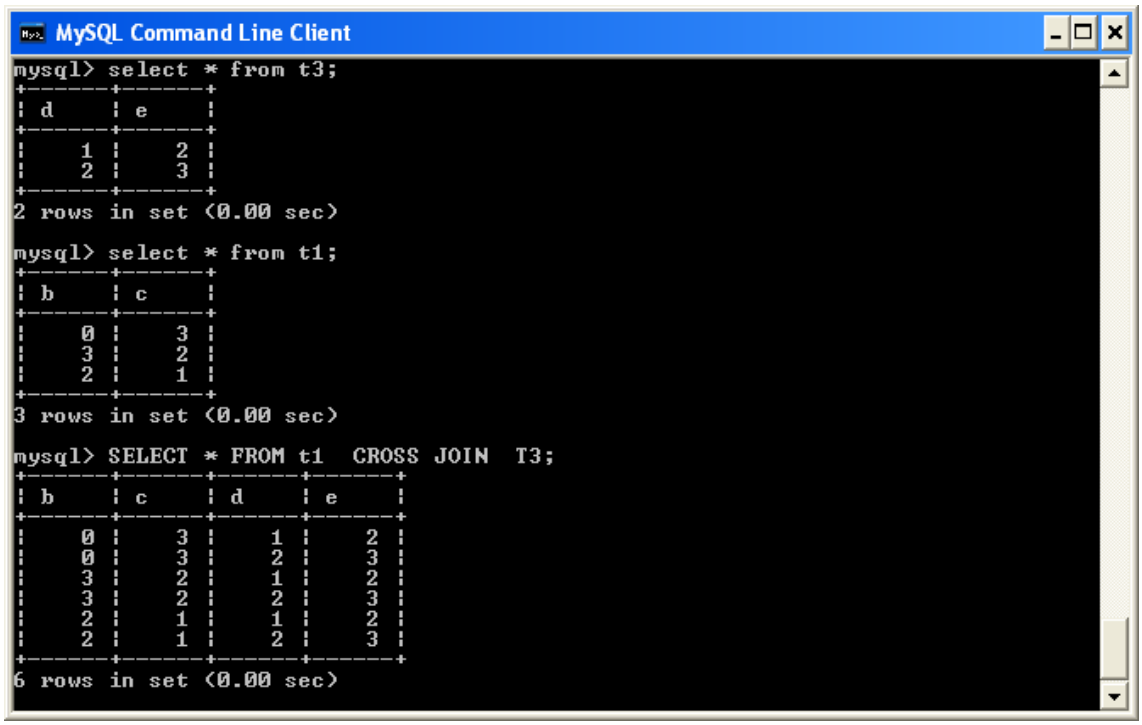


Figura 6.10: Ejemplo de producto cartesiano con MySQL

VI.3 Operaciones del lenguaje relacional.

VI.3.1 Selección.

La **selección (σ)** es una operación del álgebra relacional que usa una condición para seleccionar filas de una tabla. En otras palabras, la selección extrae tuplas de una relación dada que satisfagan una condición especificada. Se usa para crear una relación a partir de otra relación, seleccionando sólo aquellas filas que satisfacen una condición específica.

Las condiciones de selección son esencialmente las mismas condiciones usadas en la instrucción IF en los lenguajes tradicionales de programación. Sin embargo, los nombres de campos usados en una condición de selección dada deben encontrarse en la tabla nombrada en la operación de selección.

La operación de **selección** puede pensarse como la eliminación de las filas no deseadas. En la *figura 6.11* se ilustra gráficamente la selección en una tabla.

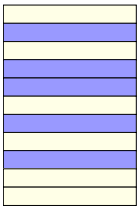


Figura 6.11: Ilustración de la operación selección (solo se eligen algunas tuplas).

Por ejemplo:

σ_{cond} **(T1)**
Resultado:= $\sigma_{(c>1 \text{ AND } B>2}$ **(T1)**
Resultado ⇨ **0, 3, 2**

T1

A	B	C
1	0	1
1	0	0
0	3	2
1	2	3
0	2	0

Figura 6.12: Selección de las tuplas en las que C>1 y B>2.

La sintaxis (simplificada) de la **selección** en MySQL es la siguiente:

SELECT *<lista de atributos>*
FROM *nombre_ tabla1* WHERE *condición*

En la *figura 6.13* se muestra un ejemplo de selección con MySQL:

```
mysql>  SELECT * FROM t1 WHERE C >1 AND B >2;
```

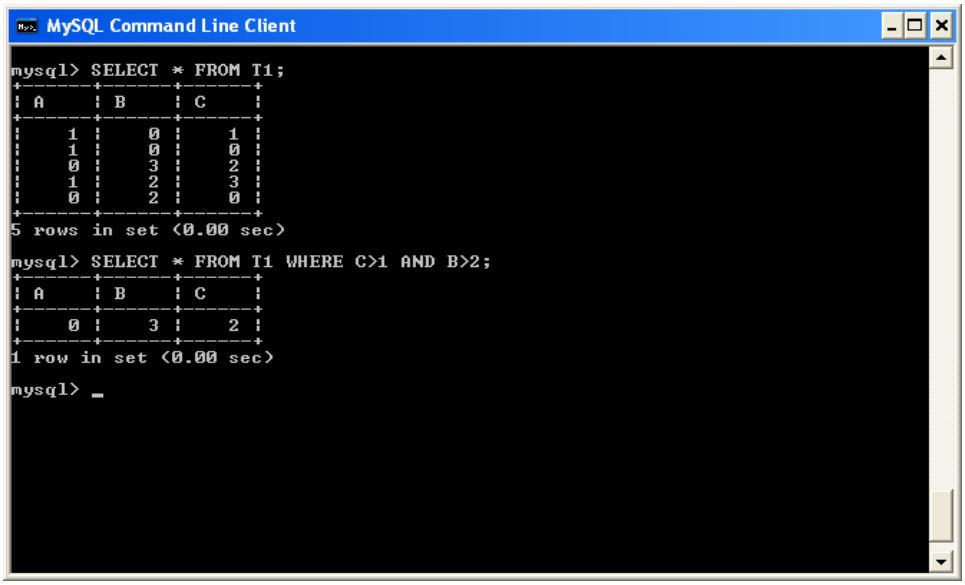


Figura 6.13: Ejemplo de selección con MySQL.

OJO!

No hay que confundir la palabra reservada SELECT de MySQL con el operador de selección del álgebra relacional. SELECT de MySQL se puede usar para realizar una proyección, para seleccionar, y para especificar otras acciones.

VI.3.2 Proyección.

La **proyección** es una operación del álgebra relacional que crea una tabla borrando columnas de una tabla existente.

La operación de **proyección** puede pensarse como la eliminación de las columnas no deseadas, a diferencia de otras operaciones del álgebra relacional, la operación de proyectar no requiere de una palabra clave especial o símbolo. Más bien, para crear una proyección, se lista simplemente la relación original seguida de las columnas que se quieren conservar encerradas entre corchetes.

En una proyección sobre una relación R obtenemos las tuplas de la entrada con sólo algunos atributos, como se ilustra en la *figura 6.14*.

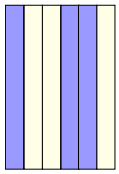


Figura 6.14: Ilustración de la operación de proyección (solo se eligen algunos campos).

Por ejemplo:

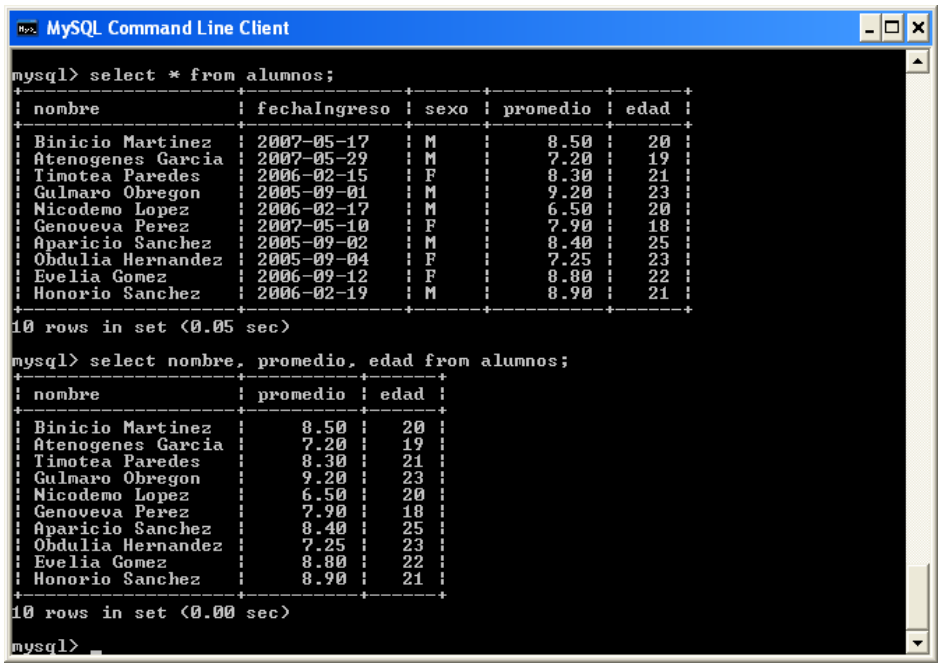


*Figura 6.*15: Ejemplo de proyección en MySQL.

VI.3.3 Reunión (Join).

La operación **reunión (join),** también llamada **combinación**, se usa para conectar datos a través de distintas relaciones, es quizás la función más importante de cualquier lenguaje de base de datos. Las columnas de la tabla resultante, serán las columnas de las tablas participantes. Las columnas que se utilicen en la condición deben tener dominios compatibles y se denominan atributos o columnas de join.

Existen varias versiones de la reunión:

1.- Reunión natural (natural join).
2.- Reunión theta (theta join).
3.- Reunión externa (outer join).

De estas tres, la reunión natural es la más importante y es la que estudiaremos en este curso.

Con la operación **reunión** se construye una relación concatenando cada tupla de la primera relación con cada una de las tuplas de la segunda, siempre que ambas tuplas satisfagan la condición dada. Esta condición se establece usando operadores relacionales (>, <, =) y operadores lógicos (AND, OR, NOT).

La **reunión natural** conecta relaciones cuando las columnas comunes tienen valores iguales.

En la **reunión externa** se incluyen las tuplas de una tabla que no están relacionadas con la otra tabla, concatenando con nulos. Es una variante del Join en la que se intenta mantener toda la información de los operandos, incluso para aquellas filas que no participan en el Joinn las tuplas que no tienen correspondencia en el Join se "rellenan con nulos".

La **reunión θ** (theta) es un producto cartesiano seguido de una restricción.

T1

A	B
a1	b1
a2	b1
a3	b3
a3	b4

*

T2

A	C
b1	c1
b2	c2
b3	c3
b1	c2

=

T1 * T2

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c2
a3	b3	c3

*Figura 6.*16: Ejemplo de reunión natural.

La sintaxis (simplificada) de la **reunión natural** en MySQL es la siguiente:

```
SELECT  *
FROM    tabla1
NATURAL JOIN
tabla2;
```

Por ejemplo:

```
mysql>  SELECT * FROM R1 NATURAL JOIN R2;
```

Este ejemplo se puede ver en la *figura 6.17.*

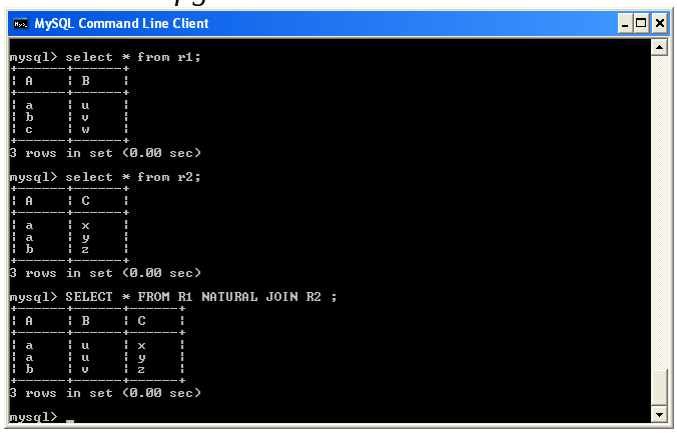


*Figura 6.*17: Ejemplo 1 de reunión natural en MySQL

Ver ahora el ejemplo 2 en la *figura 6.18*:

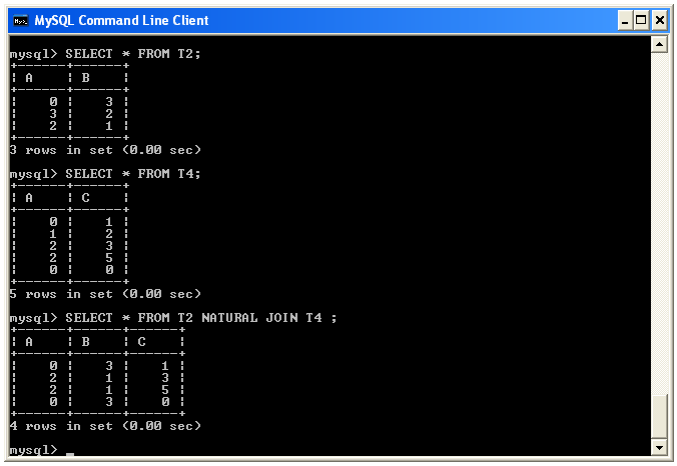


*Figura 6.*18: Ejemplo 2 de reunión natural en MySQL

Ejercicio: si hacemos la operación **natural join** sobre las tablas de la *figura 6.19*:

R

A	B	C
1	0	1
1	0	0
0	3	2
1	2	3
0	2	0

*

S

B	D
0	3
3	0
2	1

=

*Figura 6.*19: Tablas R y S.

Utilizando la instrucción:

```
mysql>  SELECT * FROM R NATURAL JOIN S;
```

entonces tendremos la tabla R*S de la *figura 6.20*:

R

A	B	C
1	0	1
1	0	0
0	3	2
1	2	3
0	2	0

*

S

B	D
0	3
3	0
2	1

=

R * S

A	B	C	D
1	0	1	3
1	0	0	3
0	3	2	0
1	2	3	1
0	2	0	1

*Figura 6.*20: Reunión natural de las tablas R y S.

Ejemplo práctico.
Sea la base de datos con las tablas de la *figura 6.21*:

GRUPOS

CLAVE_UEA	NUM_EMPL	AULA
15029	3028	A904
15328	3172	A324
12345	3075	A217
15112	3194	A220
12463	3172	A912
13947	2907	A323

PROFESORES

NUM_EMPL	NOMBRE	DEPTO	TELEFONO
2907	Moises Caceres	Quimica	21211304
2962	Arnulfo Díaz	matematicas	91801423
3075	Heriberto Jiménez	matematicas	56754321
3028	Eleazar Martinez	Fisica	57221314
3150	Carlos Pérez	Fisica	55603019
3172	Nicodemo Sanchez	matematicas	53279872
3194	Maricela Lopez	Quimica	58961212

*Figura 6.*21: Tablas de la base de datos “escuela”.

Podemos hacer una reunión natural (natural join) para reunir los datos de las dos tablas, como se muestra en la *figura 6.22*:

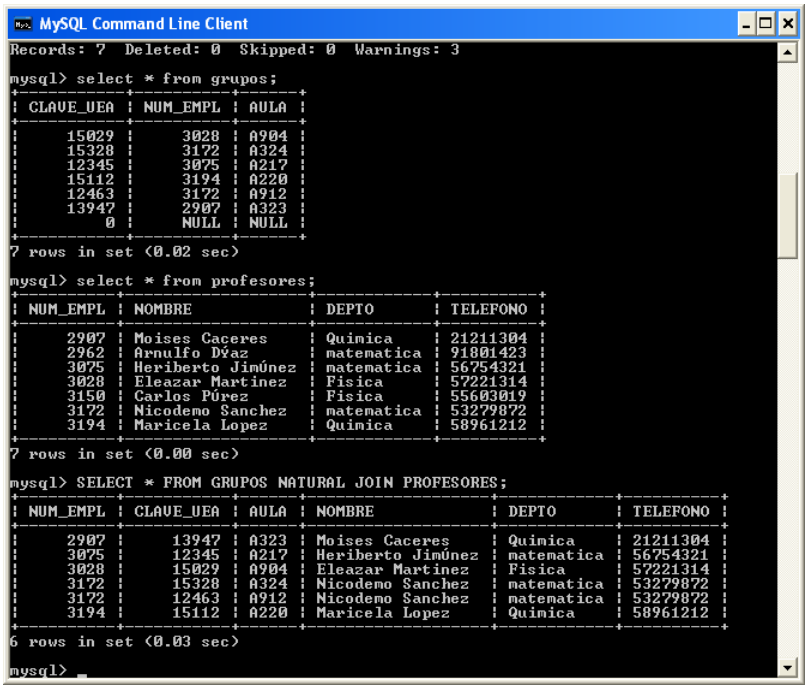


Figura 6.22: Reunión natural de las tablas Grupos y Profesores en MySQL.

En la práctica, el **Join** es muy útil cuando necesitamos extraer datos de dos tablas al mismo tiempo, por ejemplo, si tenemos las tablas de la *figura 6.23*, entonces por medio de un **Join** no necesitamos hacer dos accesos si queremos la IdUsuario, el nombre y apellido paterno de los integrantes de un proyecto.

Usuarios

IdUsuario	Nombre	APaterno	AMaterno	Mail
MGomez	Mario	Gómez	Pérez	mg@aim.com
AEspinoza	Alfredo	Espinoza	García	aeg@hotmail.mx
CRamos	Claudia	Ramos	Sanchez	clar@geoline.mx
ERodriguez	Evaristo	Rodriguez	Fernández	evar@aim.com

Integrantes

Idusuario	IdProyecto	rol
ERodriguez	2010p03	lider
AEspinoza	2010p04	lider
MGomez	2010p03	alumno
CRamos	2010p04	alumno

*Figura 6.*23: Tablas “usuarios” e “integrantes”.

Y la instrucción es la siguiente:

```
mysql>  SELECT IdUsuario, Nombre, Apaterno FROM Usuarios
            NATURAL JOIN  Integrantes
            WHERE Integrantes.IdProyecto = ‘2010p03’;
```

En la *figura 6.24* se ilustra como crear la tabla **Usuarios** en la base de datos **proyectos** y cargar los datos desde un archivo.

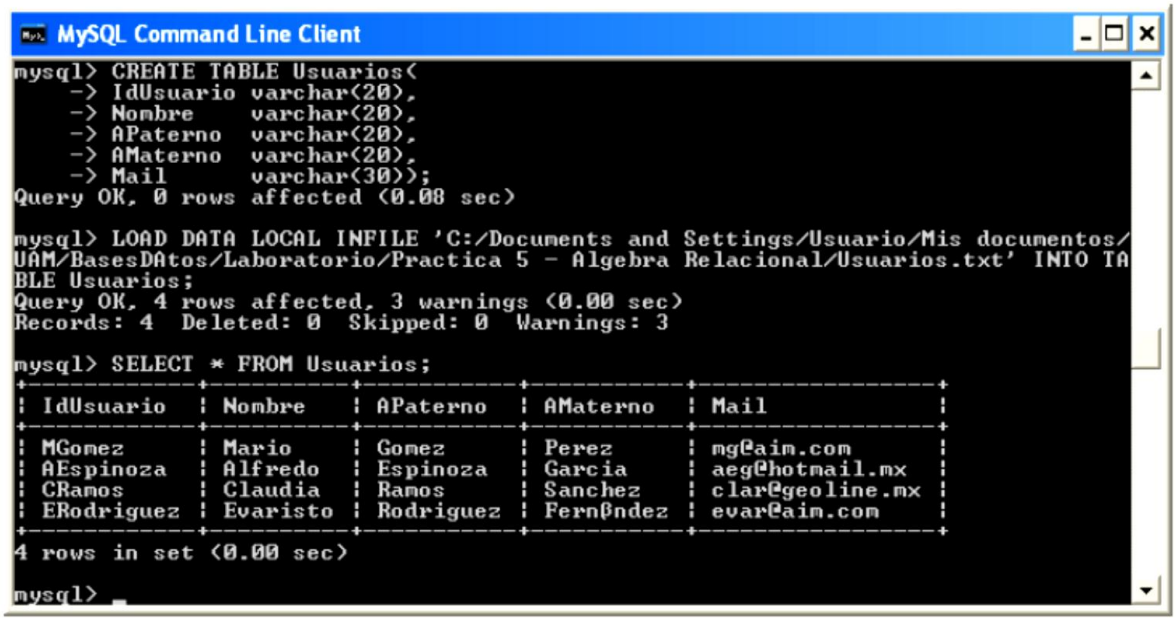


*Figura 6.*24: Creación de la Tabla "usuarios".

En la *figura 6.25* se muestra la creación de la tabla **Integrantes** en la base de datos **proyectos** y como se cargan sus datos desde archivo.

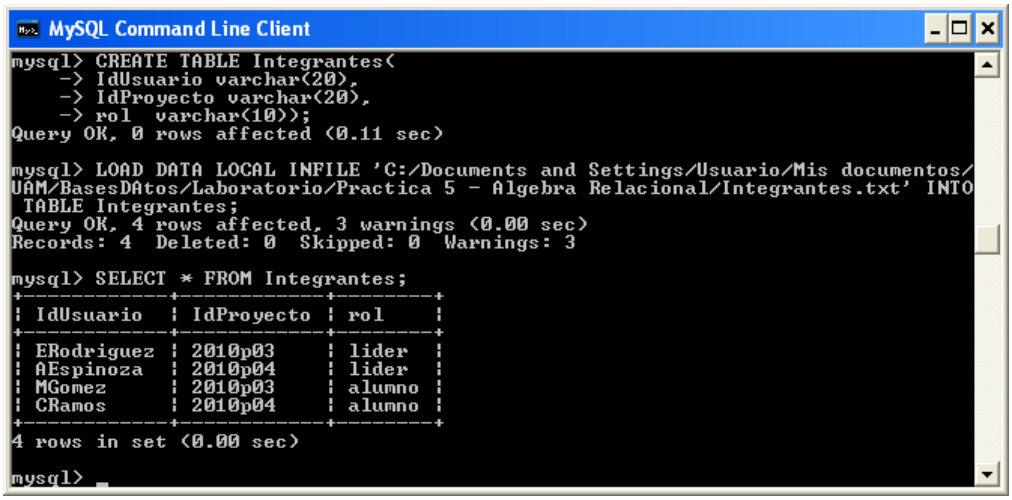


Figura 6.25: Creación de la tabla "Integrantes".

En la *figura 6.26* se muestra como obtener la IdUsuario, el nombre y apellido paterno de los integrantes del proyecto 2010p03

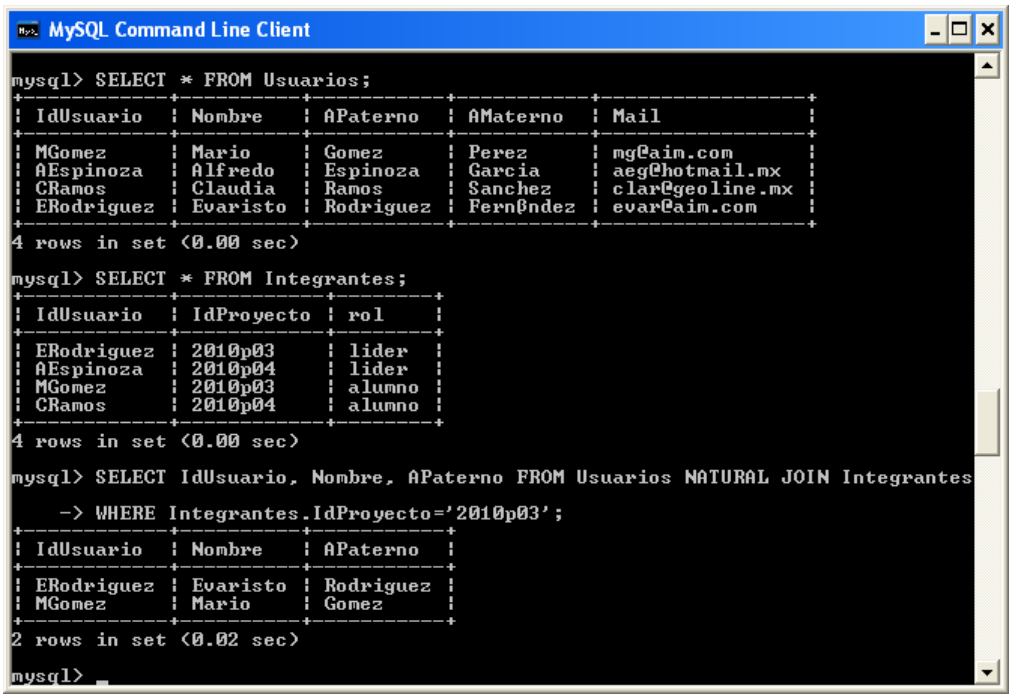


Figura 6.26: Proyección 1 de la Tabla "usuarios".

Si quisiéramos tener en la consulta, un campo adicional de la tabla **Integrantes** solo hay que añadirlo en la lista, como se muestra en la *figura 6.27*.

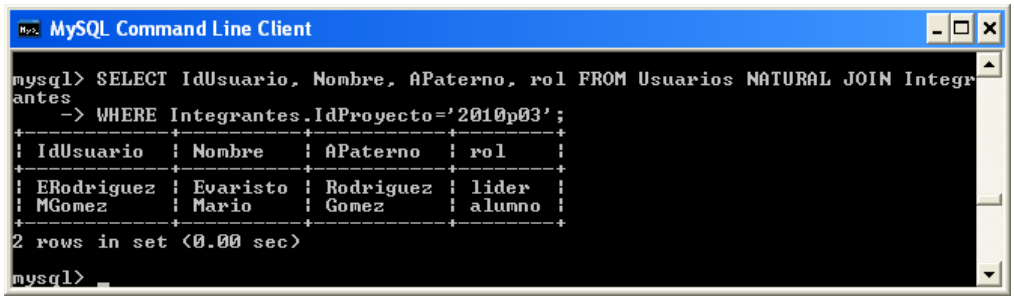


*Figura 6.*27: Proyección 2 de la Tabla "usuarios".

VI.3.4 División.

Esta operación del álgebra relacional crea una nueva relación seleccionando las tuplas de una relación R1 que se correspondan con *todas* las tuplas de otra relación R2, es decir, selecciona las tuplas de R1 que están relacionadas con *todas* las tuplas de R2. En la *figura 6.28* se ilustra gráficamente la división de la tabla T1 entre la tabla T2.

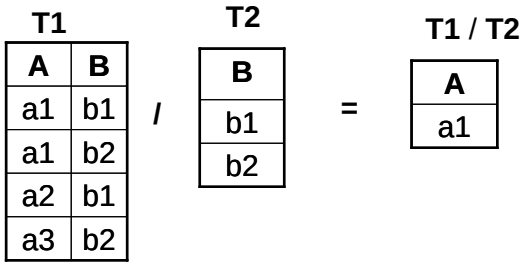


*Figura 6.*28: División de la tabla T1 entre la tabla T2.

En otras palabras, la operación **división (/)** construye una relación con los valores de un campo de una primera tabla T1(dividendo) que concuerden con *todos* los valores de los campos de otra tabla T2 (divisor).

En la *figura 6.29* se muestran algunos ejemplos de división: R/S.

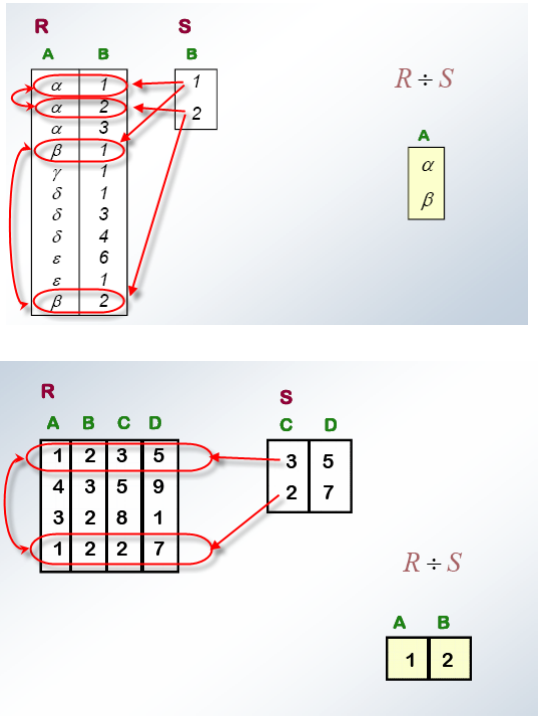


Figura 6.29: División
(Material del Curso del Dr. Ovidio Peña Rodríguez)

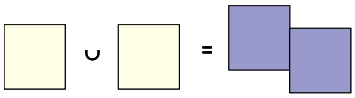
Nótese que los campos de la tabla que corresponde al divisor, deben estar incluidos en la tabla del dividendo. Además la cardinalidad de la tabla del divisor debe ser distinta de cero.

Para la División no existe una sentencia en SQL, por lo que si se requiere obtener la tabla resultado es necesario hacer una combinación de instrucciones y condiciones.

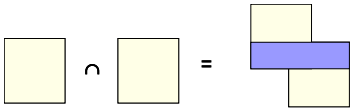
VI.4 Resumen del álgebra relacional.

Operaciones de origen matemático:

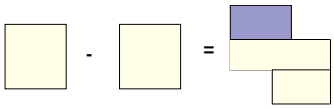
Unión.- Une las tuplas de una tabla T1 con las de otra tabla T2 (T1 y T2 deben ser tablas compatibles).



Intersección.- Son las tuplas que se encuentran tanto en una tabla T1 como en otra tabla T2 (T1 y T2) deben ser tablas compatibles).



Diferencia.- Son las tuplas que están en una tabla T1 pero que no se encuentran en otra tabla T2 (T1 y T2 deben ser tablas compatibles).



Producto cartesiano.- Forma una tercera relación uniendo los campos de las tablas T1 y T2 y concatenando todas las tuplas de T1 con cada una de las tuplas de T2.

T1

A
a1
a2

x

T2

B
b1
b2
b3

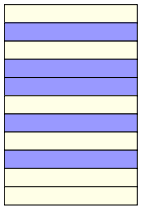
=

T1 x T2

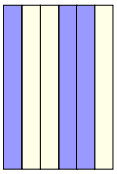
A	B
a1	b1
a1	b2
a1	b3
a2	b1
a2	b2
a2	b3

Operaciones de lenguaje relacional:

Selección.- Se usa para aplicar una condición a una relación y obtener una nueva relación que consiste de las tuplas que satisfacen la condición.



Proyección.- Crea una nueva relación usando solo las columnas deseadas de una relación existente. Borra los atributos que no están en la lista de proyección.



Reunión (join).- Conecta relaciones a través de columnas que contienen información similar.

T1

A	B
a1	b1
a2	b1
a3	b3
a3	b4

*

T2

A	C
b1	c1
b2	c2
b3	c3
b1	c2

=

T1 * T2

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c2
a3	b3	c3

División.- Identifica las tuplas de una relación que corresponde con cada una de las tuplas de otra relación.

T1

A	B
a1	b1
a1	b2
a2	b1
a3	b2

/

T2

B
b1
b2

=

T1 / T2

A
a1

Los campos de la tabla que corresponde al divisor, deben estar incluidos en la tabla del dividendo, y su cardinalidad debe ser distinta de cero.

Operación de lenguajes de programación:

Asignación.- Da un nombre a una relación.

VI.5 Ejercicios de consultas en MySQL.

Sea la base de datos “Tienda” con las tablas de las *figuras 6.30* y 6.31.

CLIENTE

ID_CLIENTE	NOMB_CLIENTE	TELEFONO	ZONA	PAGADO	DEBE
100	Jose Gomez	58324520	norte	18500	3200
102	Franciso Perez	59831758	sur	14250	9500
105	Adrian Garcia	31234582	norte	11302	1240
124	Marina Hernandez	72999136	centro	29800	5500
133	Mario Fuentes	91834912	oeste	25110	2900

VENDEDOR_JEFE

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	SUCURSAL
24	Eduardo Martínez		centro
29	Fanny Galindo	24	satelite
31	Ana Sanchez	29	churubusco
35	Oscar Lemus	24	centro

VENDEDOR_SUBORDINADO

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	SUCURSAL
8	Alejandra Lopez	29	churubusco
10	Tomas Jimenez	35	centro
11	Elena Rodriguez	31	observatorio
14	Elias Bravo	29	observatorio
19	Patricio Perez	24	satelite
29	Fanny Galindo	24	satelite
31	Ana Sanchez	29	churubusco
35	Oscar Lemus	24	centro

PRODUCTO

ID_PRODUCTO	DESCRIPCION	ID_PROVEDOR	COSTO	PRECIO
1133	balon de foot ball	312	80	120
2245	raqueta de tenis	210	280	350
2318	palos de golf	223	3100	3800
2512	balon de basquet	312	70	100

PROVEEDORES

ID_PROVEDOR	NOMB_PROVEDOR	TELEFONO
210	Andres Olvera	53822517
223	Lorenzo Gutierrez	92315880
312	Erika Peña	17221234

*Figura 6.*30: Base de datos “Tienda” (I).

VENTA

FECHA	ID_CLIENTE	ID_VENDEDOR	ID_PRODUCTO	CANTIDAD
01/07	100	10	2245	15
01/28	102	24	2512	60
02/04	102	24	1133	85
02/24	100	29	2512	50
03/30	102	24	1133	25
04/15	104	10	2245	12
04/27	124	35	2512	45
05/15	104	10	2318	5
05/29	102	24	2318	7
06/02	102	24	2245	65

*Figura 6.*31: Base de datos “Tienda” (II).

VI.5.1 Consultas sencillas.

Si queremos una relación que se llame VENDEDORES que contenga tanto a los vendedores jefes como a los subordinados, entonces utilizamos el operador **unión**, antes de aplicar la unión a dos relaciones hay que verificar que ambas tengan exactamente las mismas columnas (mismo nombre y mismo tipo). Si este es el caso se dice que las relaciones son *unión compatible*:

VENDEDORES := VENDEDOR_JEFE ∪ VENDEDOR_SUBORDINADO

En la *figura 6.32* se muestra la relación resultante: VENDEDORES.

VENDEDORES

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	SUCURSAL
8	Alejandra Lopez	29	churubusco
10	Tomas Jimenez	35	centro
11	Elena Rodriguez	31	observatorio
14	Elias Bravo	29	observatorio
19	Patricio Perez	24	satelite
24	Eduardo Martínez		centro
29	Fanny Galindo	24	satelite
31	Ana Sanchez	29	churubusco
35	Oscar Lemus	24	centro

*Figura 6.*32: Tabla (relación) VENDEDORES.

Si lo que queremos saber es la lista de vendedores que son subordinados y jefes, entonces podemos utilizar el operador **intersección**, ya que la intersección permite identificar las tuplas que son comunes a las dos relaciones, al igual que para la unión, es necesario que ambas relaciones tengan exactamente las mismas columnas. Así, podemos crear una nueva relación llamada VENDEDOR_SUBORDINADO_JEFE con el operador de intersección:

VENDEDOR_SUBORDINADO_JEFE := VENDEDOR_SUBORDINADO ∩ VENDEDOR_JEFE

En la *figura 6.33* se muestra la relación resultante: VENDEDOR_SUBORDINADO_JEFE, la cual contiene todas las tuplas que están en ambas relaciones.

VENDEDOR_SUBORDINADO_JEFE

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	SUCURSAL
29	Fanny Galindo	24	satelite
31	Ana Sanchez	29	churubusco
35	Oscar Lemus	24	centro

*Figura 6.*33: Tabla (relación) VENDEDOR_SUBORDINADO_JEFE.

Para saber cuáles son los vendedores jefes que no están subordinados, podemos utilizar la operación **diferencia**, ya que ésta operación permite identificar las tuplas de una tabla que no están en otra tabla, así tendremos:

VENDEDOR_JEFE_JEFE := VENDEDOR_JEFE – VENDEDOR_SUBORDINADO;

La relación resultante de esta diferencia: VENDEDOR_JEFE_JEFE, consta de una sola tupla:

VENDEDOR_JEFE_JEFE

ID_VENDEDOR	NOMB_VENDEDOR	ID_JEFE	SUCURSAL
24	Eduardo Martínez		centro

*Figura 6.*34: Tabla (relación) VENDEDOR_JEFE_JEFE.

Para hacer consultas en la base de datos, aplicamos el operador **selección**, que en SQL se llama SELECT, existe una amplia gama de posibilidades para hacer consultas, aquí daremos solo unos ejemplos.

a).- Si necesitamos saber la información de los vendedores de la sucursal de churubusco:

```
mysql>  SELECT * FROM vendedor_sub WHERE sucursal =’churubusco’;
```

En la *figura 6.35* se muestra esta operación en MySQL.

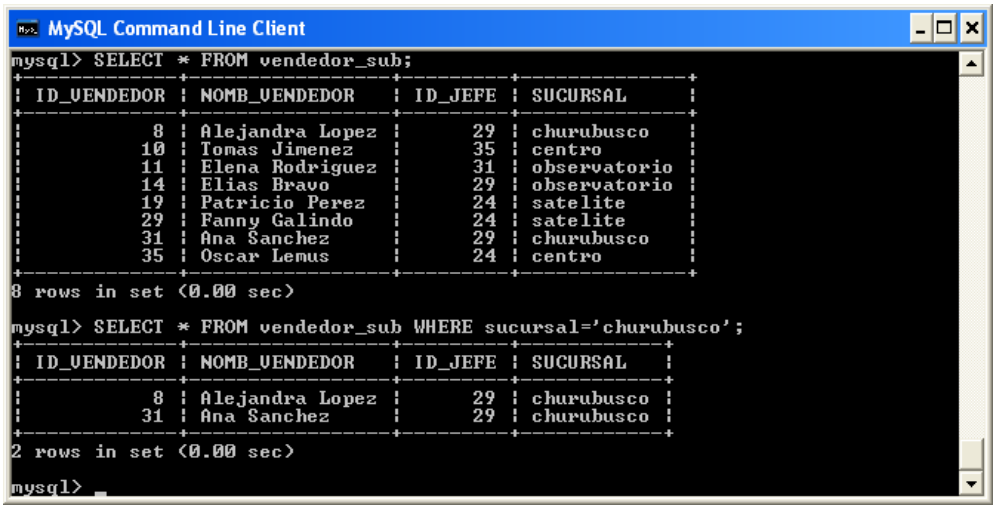


*Figura 6.*35: Resultado de:
SELECT * FROM vendedor_sub WHERE sucursal =’churubusco’;
.

Las condiciones de selección son esencialmente las mismas condiciones usadas en las instrucciones IF de los lenguajes tradicionales de programación. Sin embargo, los nombres

de campos (columnas) usados en una condición de selección dada deben encontrarse en la relación nombrada en la operación de selección. Algunos ejemplos de condiciones de selección que podrían usarse en las tablas de la base de datos en la que estamos trabajando son las siguientes:

b).- Para saber cuáles son los clientes que deben más de 3000:

```
mysql>  SELECT * FROM cliente WHERE debe > 3000;
```

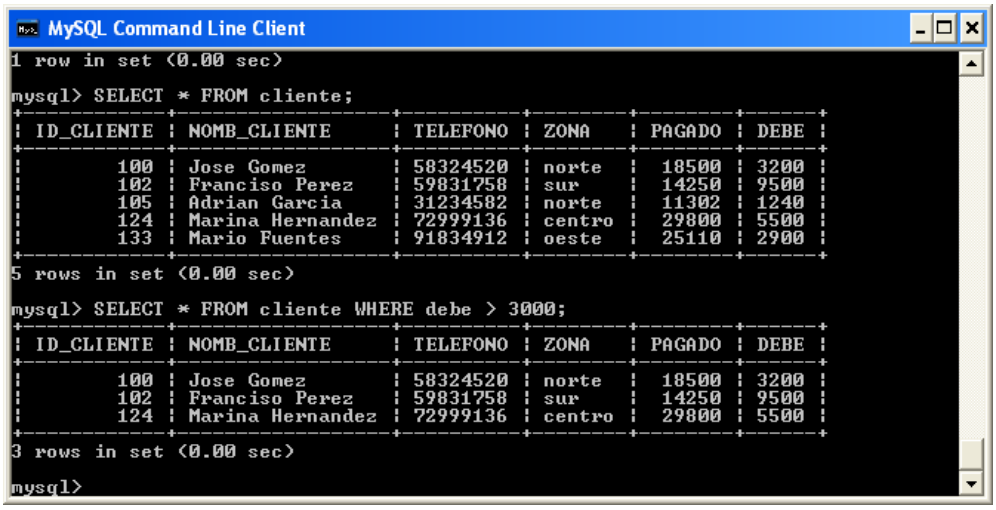


*Figura 6.*36: Resultado de:
SELECT * FROM cliente WHERE debe > 3000;

c).- Para saber cuáles son los clientes que deben más de 3000 sean de la zona centro:

```
mysql>  SELECT * FROM cliente WHERE (debe > 3000);
```

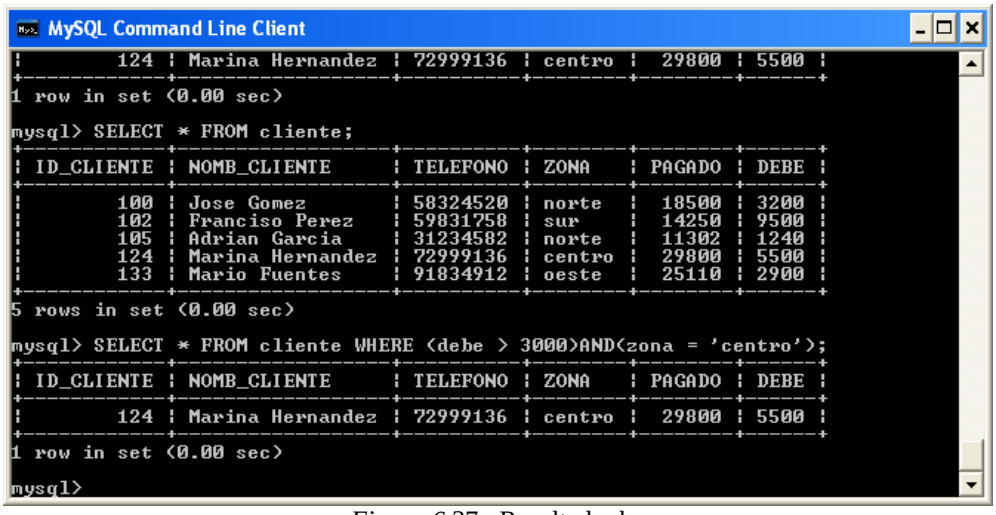


Figura 6.37: Resultado de:
SELECT * FROM cliente WHERE (debe > 3000);

d).- Podemos proyectar solo las columnas que nos interesan y hacer al mismo tiempo una selección, por ejemplo:

mysql> SELECT descripcion, precio FROM producto WHERE id_provedor = 312;

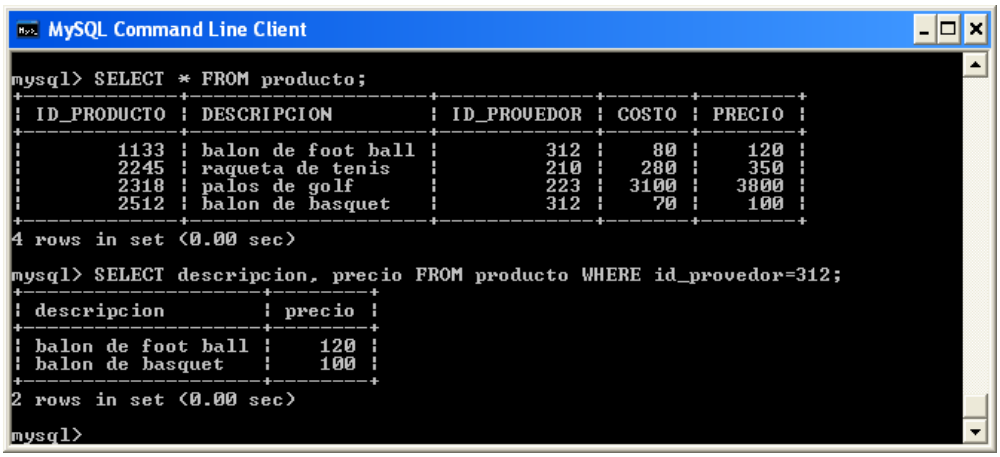


Figura 6.38: Resultado de:
SELECT descripcion, precio FROM producto WHERE id_provedor = 312;

VI.5.2 Consultas usando varias tablas.

Supongamos que queremos saber cuáles son los clientes que le han hecho compras al vendedor 10. Entonces usamos:

```
mysql>  SELECT DISTINCT nomb_cliente FROM cliente, venta
    ->  WHERE cliente.Id_cliente = venta.Id_cliente AND venta.Id_vendedor = '10';
```

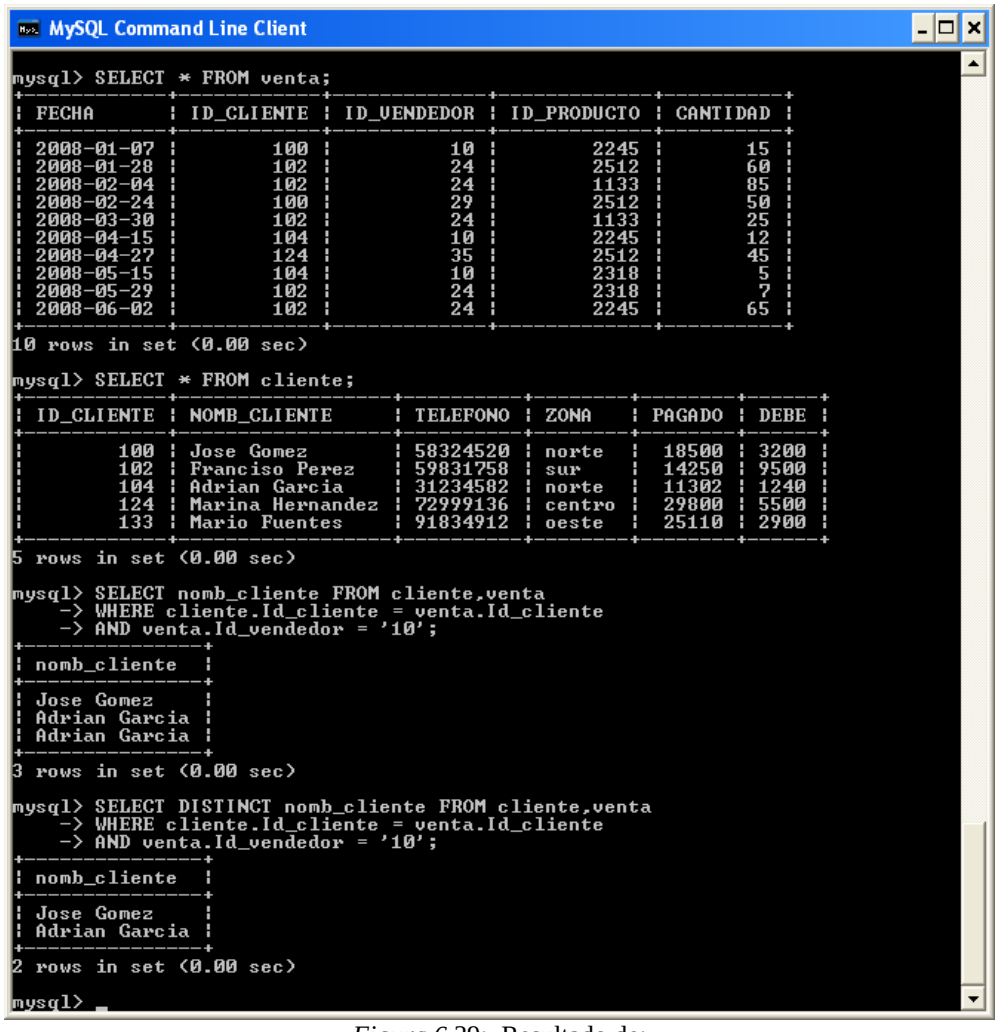


*Figura 6.*39: Resultado de:
SELECT DISTINCT nomb_cliente FROM cliente, venta
-> WHERE cliente.Id_cliente = venta.Id_cliente AND venta.Id_vendedor = '10';

Como se observa en la *figura 6.39*, **DISTINCT** sirve para que no se muestren los valores duplicados.

En la *figura 6.40* se muestra como consultar los nombres de los clientes que han comprado el producto 2318.

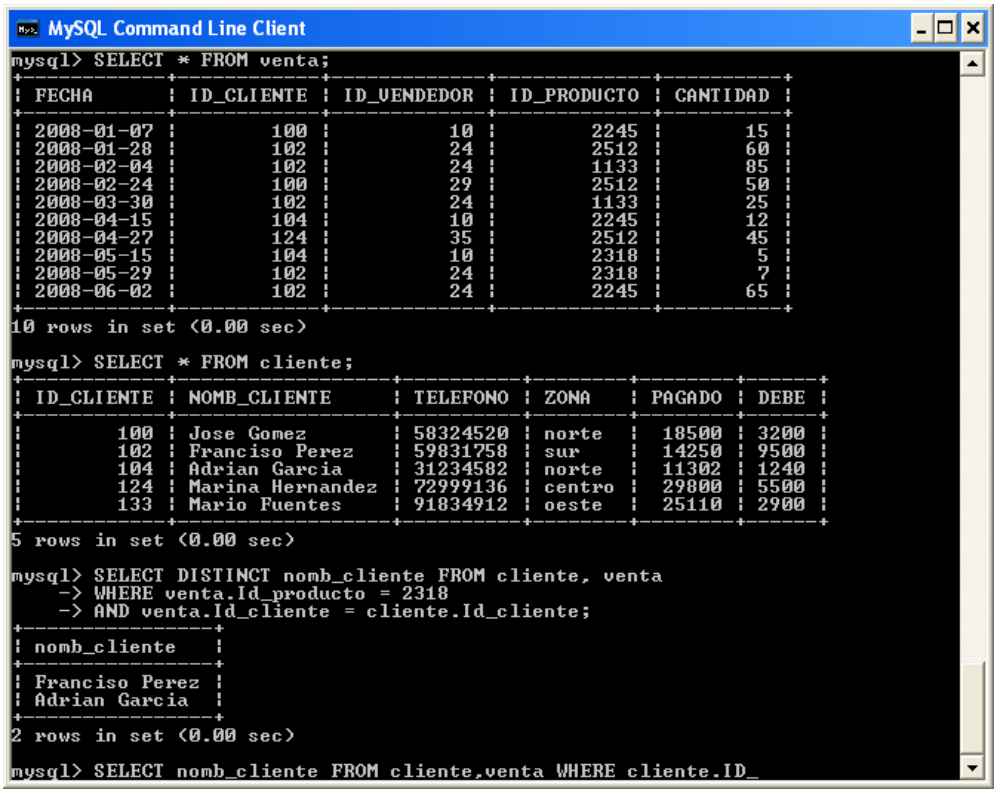


*Figura 6.*40: Consultando los clientes que compraron el producto 2318.

En la *figura 6.41* se muestra la consulta delos clientes que compraron balones de basquet-ball.

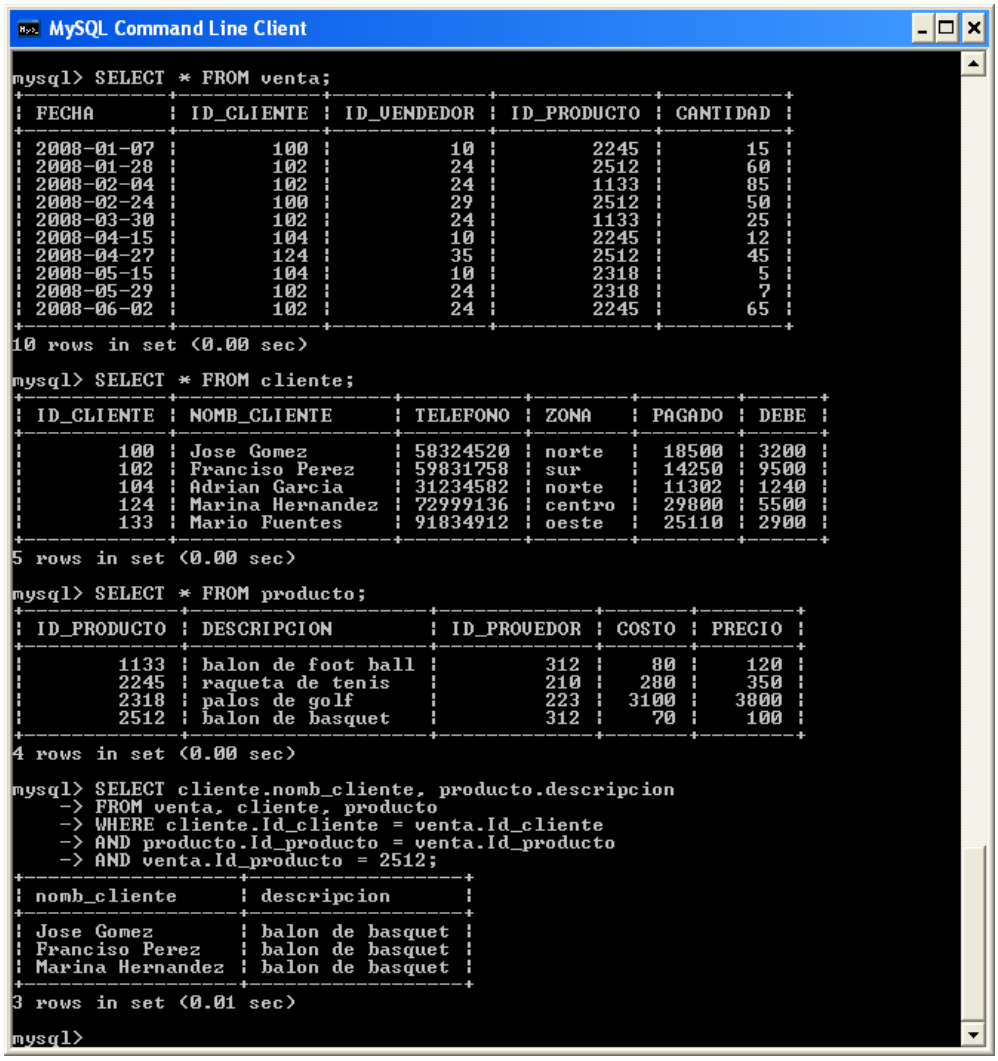


*Figura 6.*41: Consultando los clientes que compraron balones de básquet-ball.

VI.5.3 Creación de vistas.

Se dice que las tablas que forman una base de datos son las *tablas base*, porque contienen los datos básicos de la base de datos. Además de estas tablas base, es posible definir tablas adicionales, las cuáles basan su contenido de la información que obtienen de las *tablas base*, a estas tablas se les llama *vistas* de la base de datos.

Las vistas son "tablas virtuales" creadas a partir de datos almacenados en "tablas físicas". Una *vista* (*view*) es como una "ventana" que permite ver una porción de la base de datos. Las vistas son útiles para mantener la confidencialidad al restringir el acceso a ciertas partes de la base de datos. También son útiles para simplificar consultas que son utilizadas con frecuencia. La sintaxis para la creación de una vista es la siguiente:

CREATE VIEW nombre_vista AS SELECT…..*condiciones de la consulta*;

Por ejemplo, si queremos crear una vista que contenga los nombres de los clientes y el producto que compraron, entonces usamos el comando que se muestra en la *figura 6.42*:

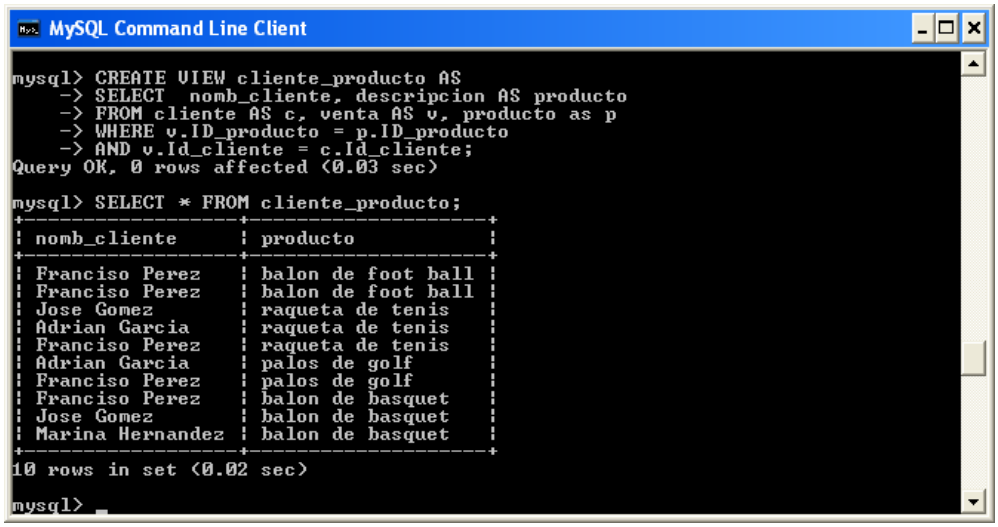


Figura 6.42: Creación de una vista "cliente_producto".

Como puede apreciarse en la *figura 6.42*, una vez creada la vista, solo es necesario hacer un "select" para poder ver su contenido. Ya no es necesario un comando largo cada vez que se requiera visualizar estos datos.

En la *figura 6.43* se muestra otro ejemplo en el que se tiene una vista con todos los vendedores (los jefes y los subordinados):

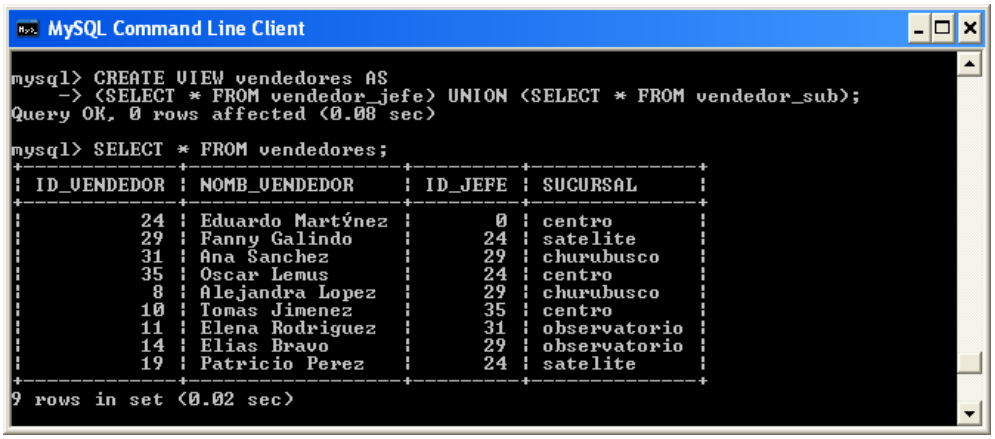


Figura 6.43: Creación de la vista "vendedores".

Hay que tomar en cuenta que:

- Si damos de baja una sesión y luego volvemos a entrar, las vistas permanecen en la base de datos.
- Las vistas no se pueden manejar como tablas.
- Para borrar una vista se usa el comando:
  DROP VIEW *nombre_vista*

Capítulo VII Diseño de bases de datos relacionales.

VII.1 Diseño de las tablas.

Las tablas son los componentes básicos de las bases de datos. La configuración de una sólida estructura de tablas es un paso clave en la creación de una base de datos eficaz y fácil de mantener.

Antes de crear una base de datos, es necesario analizar los datos y determinar cómo pueden dividirse en tablas independientes bien estructuradas.

Las bases de datos relacionales almacenan los datos en tablas independientes, basándose en el asunto, pero las tablas se combinan a través de relaciones. Por ejemplo, una tabla de clientes se relaciona con una tabla de pedidos por medio de un campo de "identidad de cliente" que contiene cada una de estas tablas. Otro ejemplo, una tabla de asignaturas se relaciona con una tabla llamada profesores por medio del campo "numero de empleado" que contiene cada una de estas tablas. Un ejemplo más, una tabla de lineas_aereas se relaciona con una tabla de destinos por medio de un campo que contiene la "clave de la línea aérea".

Normalmente, los datos no deben repetirse en más de una tabla, excepto en el caso en que los campos estén relacionados, con esta filosofía, se obtienen las siguientes ventajas:

- **Eficacia** No es necesario almacenar información redundante, como por ejemplo: el nombre o la dirección de un cliente, en cada pedido que realice el cliente.
- **Control** Es más fácil actualizar, eliminar y ampliar los datos en una base de datos bien estructurada que no contiene duplicaciones.
- **Exactitud** Al evitar las repeticiones, disminuye la posibilidad de errores. Si los datos están bien una vez, aparecen bien en todas partes.

- **Integridad de los datos** Se puede agregar o quitar campos o registros en tablas únicas sin que la estructura de los datos se vea afectada, y no será necesario volver a diseñar la base de datos.

La separación protege la estructura original. Cuando se diseñan las tablas, hay que pensar en maneras de estructurar los datos que los hagan fáciles de especificar y mantener.

VII.2 Del modelo entidad-relación a la base de datos relacional.

Antes de comenzar a diseñar un sistema de base de datos, es importante que se realice un buen análisis de requerimientos y una especificación de requerimientos muy clara, para poder identificar cuáles son las entidades, sus atributos y las relaciones que deben existir. Cuando los requisitos son claros y precisos, se facilita elaborar un diagrama entidad relación. A continuación ponemos un ejemplo de los requisitos necesarios para un sistema de base de datos que lleva el control de películas, este sistema debe:

- Guardar los datos de cada película (título, año, duración).
- Indicar en donde se produjo la película (nombre del estudio).
- Llevar un registro de los actores de cada película (cédula, nombre, apellido, nombre artístico).
- Llevar un registro del reparto por película, es decir, los roles y los actores que los desempeñaron (si hubo algún premio se indica).

A continuación, en la figura *7.1* se muestra el diagrama entidad relación:

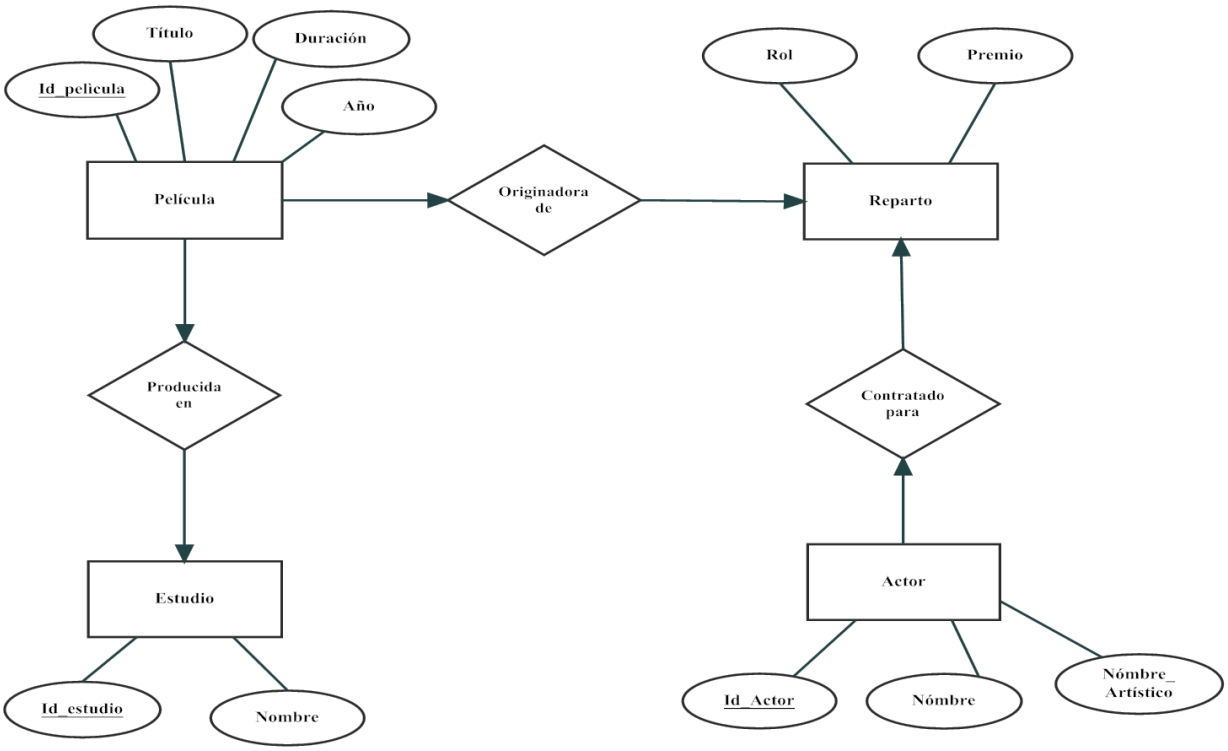


Figura 7.1: Modelo entidad-relación para la base de datos “Películas”.

Cabe señalar que no existe una forma única de elaborar un diagrama entidad-relación, el diseño depende de la experiencia y la visión particular, varios diseñadores puedes proponer diagramas diferentes en donde habría que valorar las ventajas y desventajas de cada uno.

Una vez que se tiene el diagrama entidad-relación, podemos elaborar el modelo relacional en base a lo siguiente:

- Convertir las entidades en tablas.
  - Una tabla por cada entidad.
  - El nombre de la tabla puede ser igual al de la entidad del diagrama E-R, sin embargo, algunos autores recomiendan usar plural, por ejemplo si la entidad es “película” entonce la tabla se llamará “películas”.

- Convertir los atributos en columnas
  - Los atributos de una entidad se convierten en columnas de la tabla.
  - Los atributos obligatorios deben ser “No Nulos”, es decir, no deben admitir un valor ausente o no especificado.

- Es recomendable usar nombres cortos pero significativos para cada una de las columnas de la tabla, pueden ser los mismos nombres que se usan en el modelo E-R, o pueden ser abreviaturas consistentes, por ejemplo, la clave de una película puede llamarse ID_pelicula o clave_pelicula.

- Convertir los identificadores únicos en claves primarias:
  - Cuando el identificador único esta formado por varios atributos, entonces se tiene una clave primaria compuesta.
  - Si el **identificador único está formado por relaciones con otras entidades**, se deben generar las claves externas respectivas y éstas formarán la clave primaria, en el ejemplo del sistema de las películas, podemos observar que la clave principal de la tabla "reparto" esta formada por dos claves externas, que son: la identidad del actor (ID_actor) y la identidad de la película (ID_pelicula).

- Convertir las relaciones entre entidades en claves Externas:
  - Asignar un nombre de columna para la Clave Externa y rotularlo "CE" en las especificaciones.
  - Relaciones 1 a muchos: La Clave Externa se coloca en la entidad a la que "le llega" cardinalidad muchos.
  - Si la relación es obligatoria (en el lado de la entidad **que posee** la Clave Externa), la Clave Externa no debe admitir valores nulos.

Una de las formas de trabajar es elaborar un *cuadro de especificaciones* para cada tabla del modelo relacional, este cuadro debe contener la información que se muestra en la *figura 7.2*.

Nombre de la tabla.

Nombre columna	Atributo 1	Atributo 2	…..	Atributo N
Tipo Clave				
Nulos				
Ejemplo				

Figura 7.2: Datos de una tabla.
(Material del curso del Dr. Ovidio Peña Rodríguez).

La clave primaria se abrevia como CP y la clave externa como CE, en el caso de las claves primarias compuestas se indica CP en cada una de las columnas que pertenecen a los atributos que la componen.

Cuando es obligatorio que el campo tenga un valor, es decir, que no admite valores nulos, se indica con la abreviatura NN (No Nulo).

Ejemplo 1.- Sea la tabla de la *figura 7.3*.

Películas

Nombre columna	Id_pelicula	titulo	Año	duración	Producida
Tipo clave	CP				CE
Nulos	NN	NN	NN	NN	
Ejemplo	F14	"Código x"	2004	120	E1
	F97	"Zeta"	2005	150	E35
	F28	"Mar adentro"	2004	100	E11

Figura 7.3: Datos de una tabla "Películas".
(Material del curso del Dr. Ovidio Peña Rodríguez).

Omitiendo las tuplas de ejemplo, las demás tablas del modelo relacional dado por el diagrama entidad-relación de la *figura 7.1*, quedarían definidas como se muestra en la *figura 7.4*.

Actores

Nombre columna	ID_actor	Nombre	Nombre_artístico
Tipo Clave	CP		
Nulos	NN	NN	

Reparto

Nombre columna	ID_pelicula	ID_actor	rol	premio
Tipo Clave	CE, CP	CE, CP		
Nulos	NN	NN		

Estudios

Nombre columna	ID_estudio	Nombre
Tipo Clave	CP	
Nulos	NN	NN

Figura 7.4: Tablas de la base de datos "Películas".
(Material del curso del Dr. Ovidio Peña Rodríguez).

Ejemplo 2.- Hacer un diagrama E-R de una BD para las carreras de Ingeniería en Computación y Matemáticas Aplicadas de la UAM Cuajimalpa, con las siguientes carácterísticas:

- Que guarde los datos de cada alumno (matricula, nombre, mail, teléfono)
- Que contenga las UEAs (Id_UEA, nombre) así como los profesores disponibles (num_empl, nombre, departamento, teléfono).
- Se debe llevar una relación de los cursos ofrecidos (grupo, UEA, profesor, salón) y de las inscripciones (matrícula, grupo, calificación)

El diagrama entidad-relación que se propone es el de la *figura 7.5.*

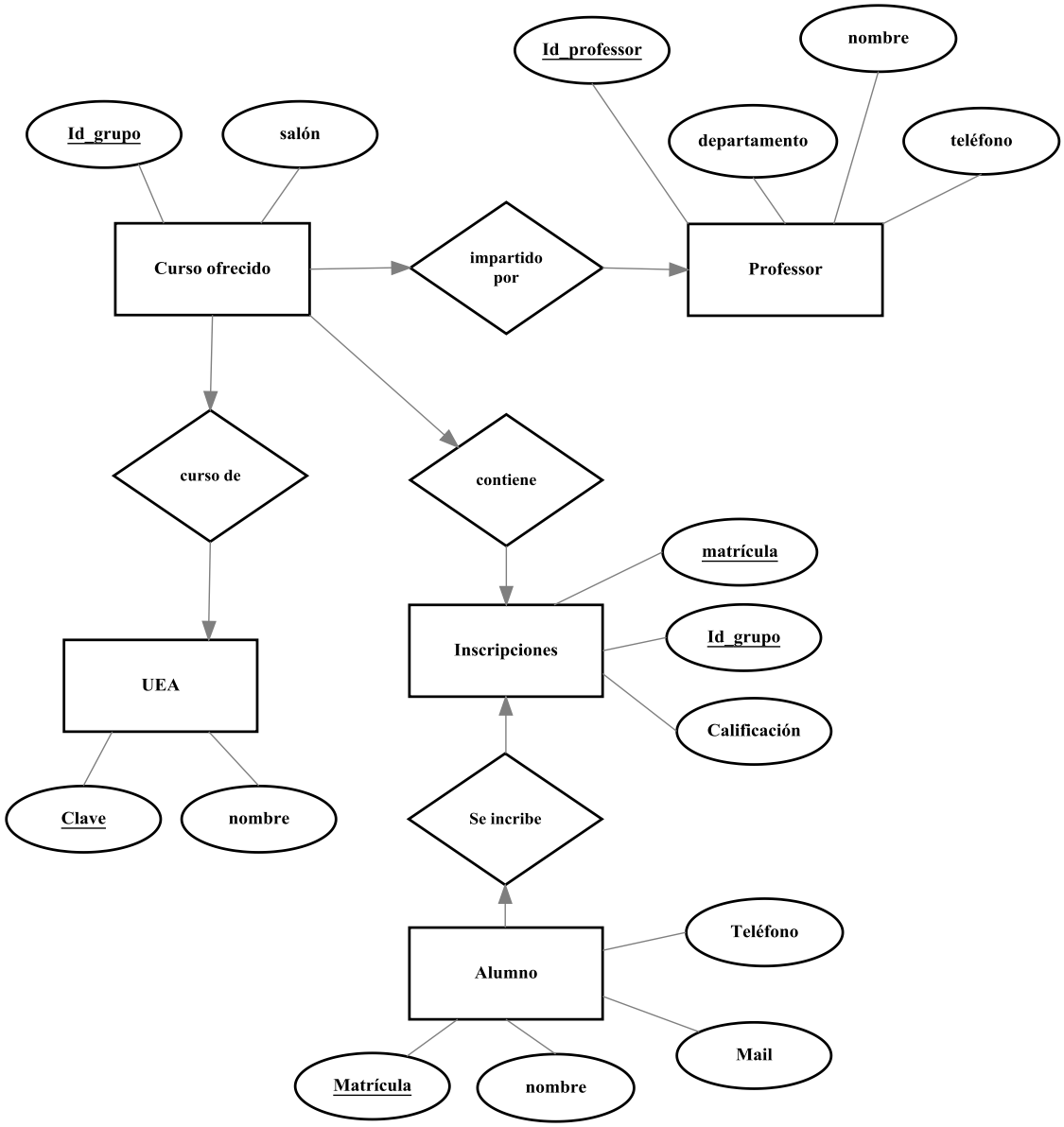


Figura 7.5: Diagrama E-R de “Escuela”.

En base al diagrama entidad-relación propuesto, configuramos el modelo relacional de la *figura 7.6*.

Cursos_ofrecidos

Nombre columna	ID_grupo	UEA	ID_profesor	Aula
Tipo Clave	CE, CP	CE, CP	CE	
Nulos	NN	NN	NN	

Profesores

Nombre columna	ID_profesor	Nombre	Departamento	Teléfono
Tipo Clave	CP			
Nulos	NN	NN	NN	

Alumnos

Nombre columna	Matricula	Nombre	Mail	Teléfono
Tipo Clave	CP			
Nulos	NN	NN		

Inscripciones

Nombre columna	Matricula	ID_grupo	Calificación
Tipo Clave	CE, CP	CE, CP	
Nulos	NN	NN	

UEAs

Nombre columna	Clave	nombre
Tipo Clave	CP	
Nulos	NN	NN

Figura 7.6: Modelo relacional de “Escuela”.

Capítulo VIII Privilegios.

VIII.1 El sistema de privilegios.

La función principal del sistema de privilegios es identificar al usuario que se conecta a la base de datos mediante una cuenta de usuario y una contraseña, y posteriormente asociar a cada usuario en particular privilegios para consultar y/o hacer modificaciones en las tablas.

El sistema de privilegios actúa mediante dos etapas:

Etapa 1: El servidor comprueba si debe permitir conectarse a un usuario con una cuenta y una contraseña dadas.

Etapa 2: Asumiendo que se conecta, el servidor comprueba cada comando que ejecuta para ver si tiene suficientes permisos para hacerlo. Por ejemplo, si intenta seleccionar registros de una tabla en una base de datos o eliminar una tabla de la base de datos, el servidor verifica que tenga el permiso **SELECT** para la tabla o el permiso **DROP** para la base de datos.

Para configurar las cuentas de usuario y controlar los privilegios disponibles para cada una, se utilizan los comandos **GRANT** y **REVOKE.**

VIII.2 Los tipos de permisos.

Hay una extensa variedad de permisos que se pueden otorgar o quitar a los usuarios, estos pueden ser válidos a diferentes niveles:

- Nivel global.
- Nivel base de datos.
- Nivel de tabla.
- Nivel de columna.
- Nivel de rutina (no los veremos en este curso)

En la *tabla 8.1* se muestran los permisos más importantes con los que se trabaja, en el manual de referencia puede consultarse la lista completa de tipos de permisos.

Permiso	Significado
`ALL [PRIVILEGES]`	Da todos los permisos simples excepto `GRANT OPTION`
`ALTER`	Permite el uso de `ALTER TABLE`
`ALTER ROUTINE`	Modifica o borra rutinas almacenadas
`CREATE`	Permite el uso de `CREATE TABLE`
`CREATE ROUTINE`	Crea rutinas almacenadas
`CREATE TEMPORARY TABLES`	Permite el uso de `CREATE TEMPORARY TABLE`
`CREATE USER`	Permite el uso de `CREATE USER`, `DROP USER`, `RENAME USER`, y `REVOKE ALL PRIVILEGES`.
`CREATE VIEW`	Permite el uso de `CREATE VIEW`
`DELETE`	Permite el uso de `DELETE`
`DROP`	Permite el uso de `DROP TABLE`
`EXECUTE`	Permite al usuario ejecutar rutinas almacenadas
`FILE`	Permite el uso de `SELECT ... INTO OUTFILE` y `LOAD DATA INFILE`
`INDEX`	Permite el uso de `CREATE INDEX` y `DROP INDEX`
`INSERT`	Permite el uso de `INSERT`
`SELECT`	Permite el uso de `SELECT`
`SHOW DATABASES`	`SHOW DATABASES` muestra todas las bases de datos
`SHOW VIEW`	Permite el uso de `SHOW CREATE VIEW`
`UPDATE`	Permite el uso de `UPDATE`
`GRANT OPTION`	Permite dar permisos

Tabla 8.1: Los privilegios principales (http://dev.mysql.com/doc/refman/5.0).

En MySQL es posible crear diversos usuarios, al inicio se crea un usuario que puede utilizar todos los comandos que existen en MySQL sobre una base de datos en particular, a este usuario se le llamamos administrador (root) y posee todos los privilegios. Los privilegios se almacenan en tablas que existen por "default" dentro del servidor de MySQL a estas tablas se les llama ***tablas Grant***.

Cuando se inicia una sesión en MySQL las tablas Grant se cargan en memoria. Cuando se recargan las tablas grant en memoria los privilegios se ven afectados de la siguiente manera:

- Los cambios en los privilegios de tabla y columna surten efecto en la siguiente petición del usuario.
- Los cambios en privilegios de la base de datos surten efecto en la siguiente sentencia USE *db_name*.

- Los cambios a los privilegios globales y las claves de acceso surten efecto la próxima vez que el usuario inicie una nueva sesión.

El único usuario que tiene la capacidad de dar o revocar permisos a los usuarios será siempre el usuario root o administrador (solo existe uno por servidor), este puede usar dos comandos:

- GRANT: Para otorgar ciertos privilegios al usuario
- REVOKE: Para quitar o revocar los privilegios de un usuario en particular.

VIII.3 La sintaxis de GRANT y de REVOKE.

Los comandos GRANT y REVOKE sirven para dar (con GRANT) o quitar (con REVOKE) a los usuarios los permisos mencionados en la *tabla 9.1*, y algunos otros más. La sintaxis simplificada del comando GRANT es la siguiente:

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
   ON [object_type] {tbl_name | * | *.* | db_name.*}
   TO user [IDENTIFIED BY [PASSWORD] 'password']
      [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

Y para REVOKE:

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
   ON [object_type] {tbl_name | * | *.* | db_name.*}
   FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Para determinar que privilegios tiene la cuenta en la que se esta trabajando:

```
MySQL> SHOW GRANTS;
```

Se puede asignar permisos globales usando sintaxis **ON *.*** o permisos a nivel de base de datos usando la sintaxis **ON** *nombre_bd***.***. Si especificamos **ON *** y tenemos seleccionada una base de datos por defecto, los permisos se dan en esa base de datos. (***Atención:*** Si especificamos **ON *** y *no* hemos seleccionado una base de datos por defecto, los permisos dados son globales.).

A nivel tabla, los únicos permisos que pueden especificarse son **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **CREATE**, **DROP**, **GRANT OPTION**, **INDEX**, y **ALTER**.
Mientras que para una columna, solo son **SELECT**, **INSERT** y **UPDATE**,

Para crear un usuario se utiliza:

```
GRANT USAGE ON BaseDatos.Tabla TO usuario IDENTIFIED BY ‘ clave
```

En la *figura 8.1* se muestra un ejemplo:

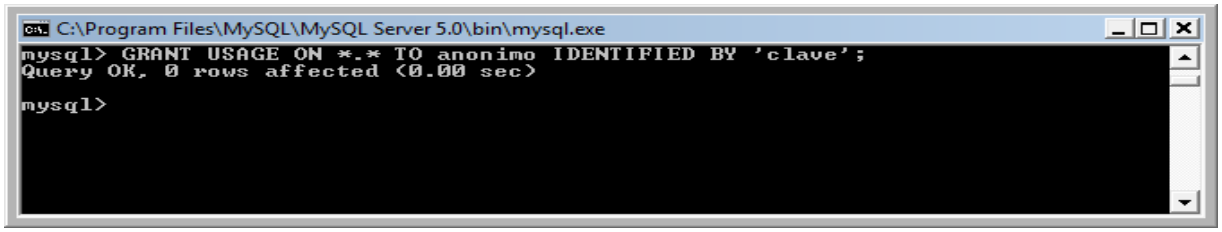


Figura 8.1: Dando de alta al usuario “anónimo”.

Con lo anterior, lo único que podrá hacer el usuario *anonimo* es conectarse a una sesión de MySQL, no tiene privilegios. En la *figura 8.2* se observa como *anónimo* entra a su sesión. El comando para entrar es el siguiente:

```
mysql –h localhost  -u usuario -p
```

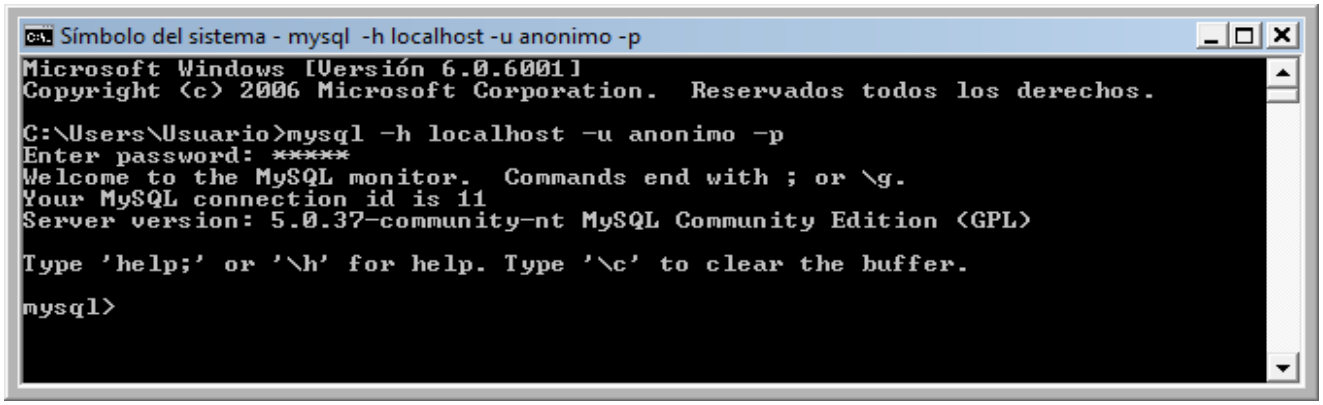


Figura 8.2: “Anonimo” entra a su sesión MySQL.

Nótese que para que el usuario pueda conectarse a una sesión, es necesario entrar desde el símbolo del sistema. Además es necesario estar situado en el directorio donde se encuentra MySQL.exe, o bien proporcionar la ruta de acceso (*figura 8.3*).

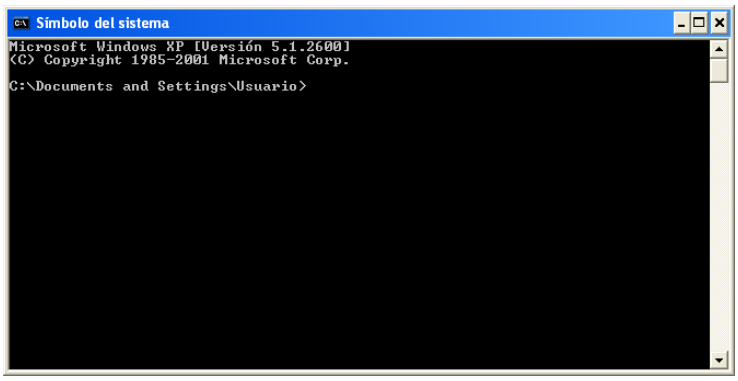


Figura 8.3: Es necesario estar situado en el directorio donde se encuentra MySQL.exe.

Para otorgar, por ejemplo, el privilegio del comando SELECT en la base de datos "miguelin" al usuario "anonimo" el administrador debe dar el comando de la *figura 8.4.*

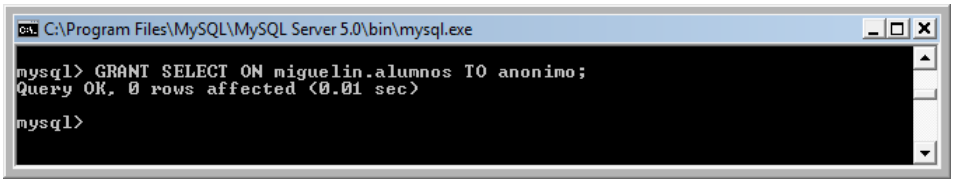


Figura 8.4: Otorgando el privilegio SELECT al usuario "anónimo".

De esta forma, el usuario "anonimo" podrá hacer consultas sobre la base de datos "miguelin".

Para quitar este privilegio de consulta, el administrador debe usar el comando "revoke" como en la *figura 8.5*.

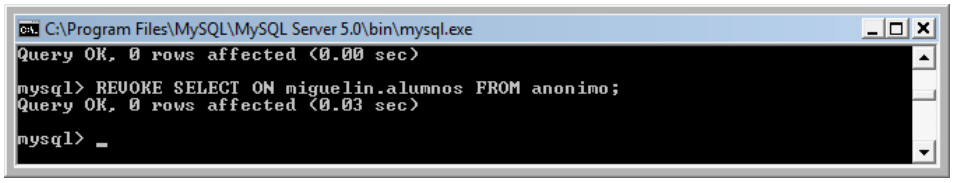


Figura 8.5: Eliminando el privilegio SELECT al usuario "anónimo".

Para eliminar a un usuario se utiliza el comando DROP USER, como se indica en la *figura 8.6*.

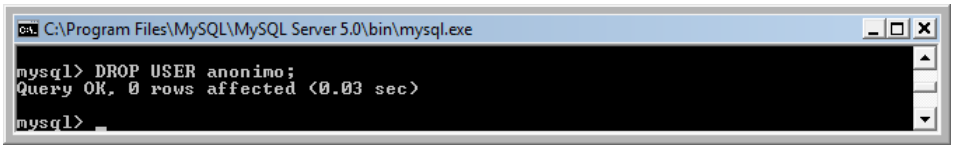


Figura 8.6: Eliminando al usuario "anónimo".

Capítulo IX Transacciones.

IX.1 Definición de transacción y sus características.

Cuando dos o más usuarios trabajan simultáneamente sobre una base de datos se pueden presentar problemas de consistencia. Por ejemplo, si un usuario está actualizando los datos de una tabla y al mismo tiempo otro usuario está consultando estos datos, es posible que el usuario que los consulta obtenga datos inconsistentes si no se toman medidas al respecto. Para que un Sistema Administrador de Base de Datos permita que varios usuarios trabajen sobre la base de datos de manera concurrente se utiliza el concepto de *transacción*.

Una *transacción* es la secuencia de una o más comandos que se ejecutan de una manera conjunta, formando una unidad lógica de trabajo.

Existen tres tipos de transacciones:

- *Transacciones de recuperación.*- Consultan la información para elaborar reportes.
- *Transacciones de actualización.*- Se insertan, borran o actualizan datos de la base de datos.
- *Transacciones mixtas.*- Se mezclan operaciones de recuperación de datos y de actualización.

Además, las transacciones tienen las siguientes propiedades:

- *Atomicidad.*- Una transacción debe ser una unidad indivisible de trabajo. Si una transacción no se termina, el SGBD recupera los valores anteriores desde disco y los presenta como actuales.
- *Consistencia.*- Transforman la base de datos de un estado consistente a otro.
- *Independencia.*- Los efectos parciales de una transacción incompleta no son visibles al resto de las transacciones.
- *Durabilidad:* Los efectos de una transacción enviada se almacenan permanentemente en la base de datos y no pueden ser cancelados.
- *Aislamiento.*- Cuando se tienen dos transacciones simultáneas, una transacción reconoce los datos en el estado en que estaban antes de que la otra transacción los

modificara o después de que la segunda transacción haya concluido, pero no reconoce un estado intermedio.

Para garantizar la coherencia de una transacción, el administrador de la base de datos utiliza el "motor de base de datos". Una vez que se inicia una transacción ésta debe concluirse correctamente, en caso contrario, la instancia del Motor de base de datos deshace todas las modificaciones de datos realizadas desde que se inició la transacción.

Una transacción que abarca dos o más bases de datos es una transacción distribuida. La transacción distribuida se administra internamente de forma que para el usuario funciona como una transacción local.

En la *figura 9.1* se muestra el flujo de datos en una transacción, como puede observarse, las operaciones de lectura y escritura se realizan sobre una página de memoria intermedia, de tal forma que se puede controlar que los datos que se leen sean los que estaban antes o después de la escritura, dependiendo de la prioridad que se les dé a las operaciones.

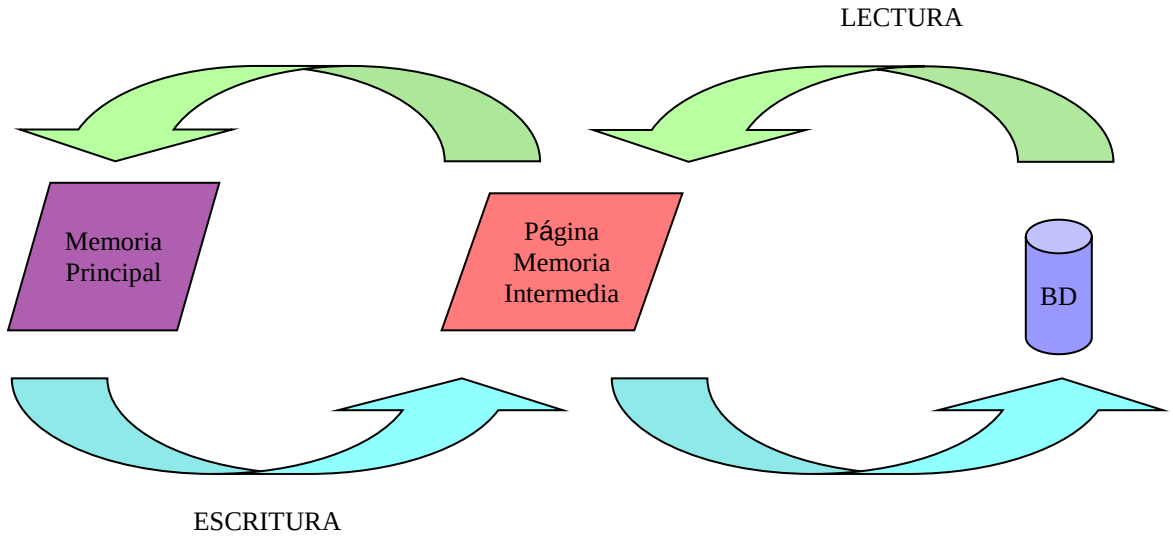


Figura 9.1: Flujo de datos en una transacción.
(Material del curso del Dr. Ovidio Peña Rodríguez).

En el diagrama de flujo de la *figura 9.2* se pueden apreciar las principales instrucciones que se utilizan en una transacción.

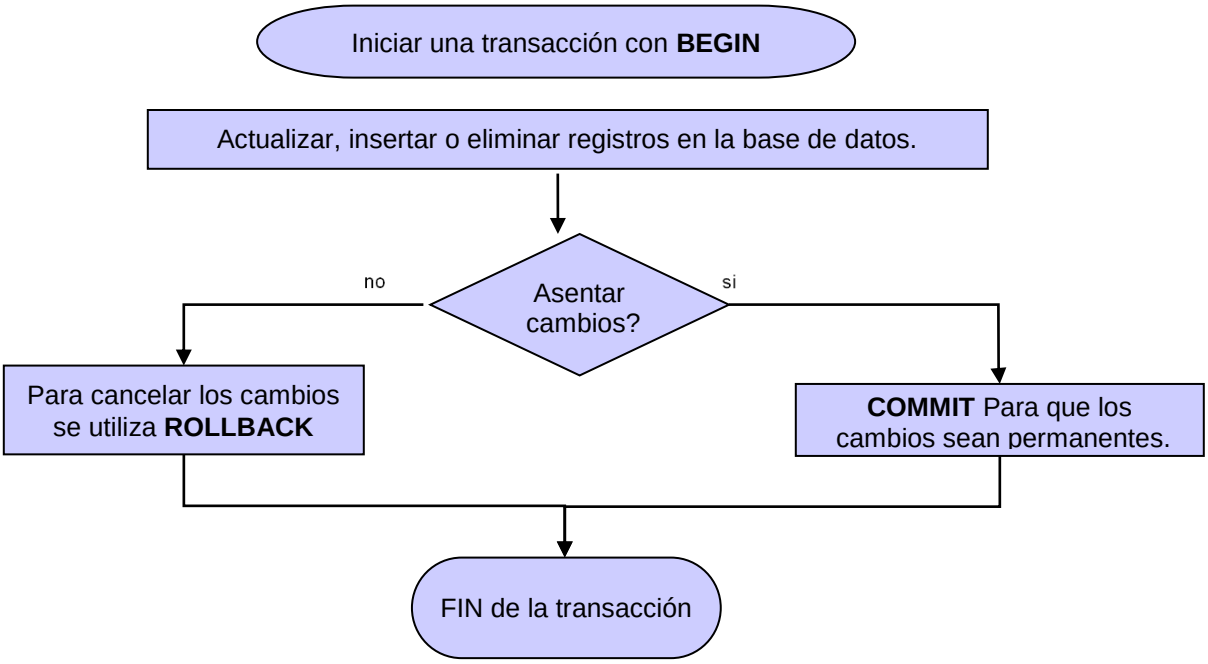


Figura 9.2: Secuencia de una transacción.

Un *candado* es un mecanismo que prohíbe que dos transacciones manipulen los mismos datos al mismo tiempo. Por ejemplo, un candado evita que se borre una tabla si en ese momento se está llevando a cabo otra transacción sobre ésta.

Durante la ejecución de una transacción pueden existir fallas de diversa índole, a continuación se especifican.

- Fallas durante la transferencia de datos:
  - Error en la programación: la transacción no termina.
  - Interbloqueo.
  - Error en un comando.
- Fallas del sistema.
  - Cuando se pierden datos de memoria principal.
  - Cuando se reinicia el sistema.
- Falla de memoria secundaria.
  - Defectos de software.
  - Defectos del hardware.
  - Pérdida parcial de la Base de Datos.
- Fallas no recuperables.
  - Virus.
  - Factores externos.

IX.2 Estados de una transacción.

Una vez que una transacción se inicia, ésta pasa por diferentes estados, dependiendo de su éxito o fracaso:

- *Activa:* Se dice que una transacción esta "activa" durante su ejecución.
- *Abortada*: Se dice que una transacción esta "abortada" cuando su ejecución no termina de manera satisfactoria, en este caso, las modificaciones parciales que hizo a la Base de Datos deben deshacerse.
- *Fallida*: Cuando, por alguna causa, su ejecución no puede continuar.
- *Parcialmente comprometida*: Una transacción se encuentra "parcialmente comprometida" luego de ejecutar la última instrucción.
- *Comprometida*: Cuando la transacción se completa con éxito, es decir, una vez que se asentaron los cambios, ésta se encuentra en el estado de "comprometida".

En la *Figura 9.3* se muestra el diagrama de transición de estados de una transacción

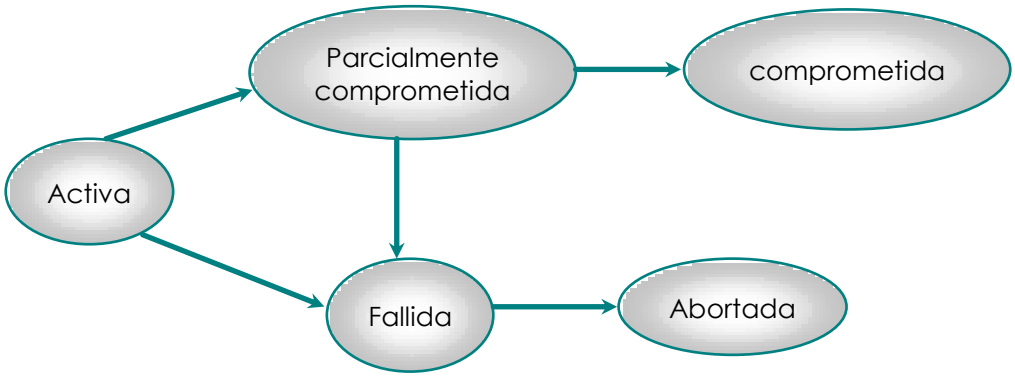


Figura 9.3: Diagrama de transición de estados de una transacción.
(Material del curso del Dr. Ovidio Peña Rodríguez).

IX.2.1 El motor de la base de datos.

El servidor de bases de datos MySQL soporta distintos tipos de tablas, tales como *ISAM*, *MyISAM*, *InnoDB*, y *DBD (Berkeley Database)*. De éstos, InnoDB es el tipo de tabla más importante

Las tablas del tipo InnoDB almacenan en un sólo archivo, en lugar de tres y sus principales características son que permite trabajar con transacciones y definir reglas de integridad referencial. Para asegurar que se tiene soporte para el tipo de tablas InnoDB se usa la instrucción, como en la *figura 9.4*:

```
mysql> SHOW VARIABLES LIKE '%innodb%';
```

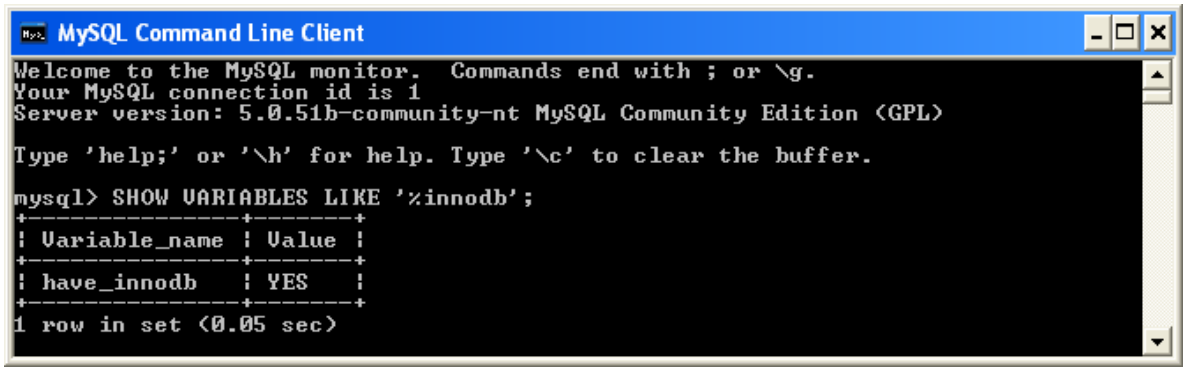


Figura 9.4: **have_innodb** que tiene el valor YES indica que sí se tiene el motor **innodb**

Ejemplo.

A continuación crearemos una tabla llamada Transacc con el motor InnoDB, con un solo atributo llamado campo, posteriormente agregaremos 3 tuplas con los valores 1, 2 y 3. Los comandos en MySQL para hacer lo anterior son los siguientes:

- **CREATE TABLE** transacc
  (campo **INT NOT NULL PRIMARY KEY**) **TYPE** = InnoDB;
- **INSERT INTO** transacc **VALUES** (1), (2), (3);
- **SELECT** * **FROM** transacc;

Este comando se muestra en la *figura 9.5*.

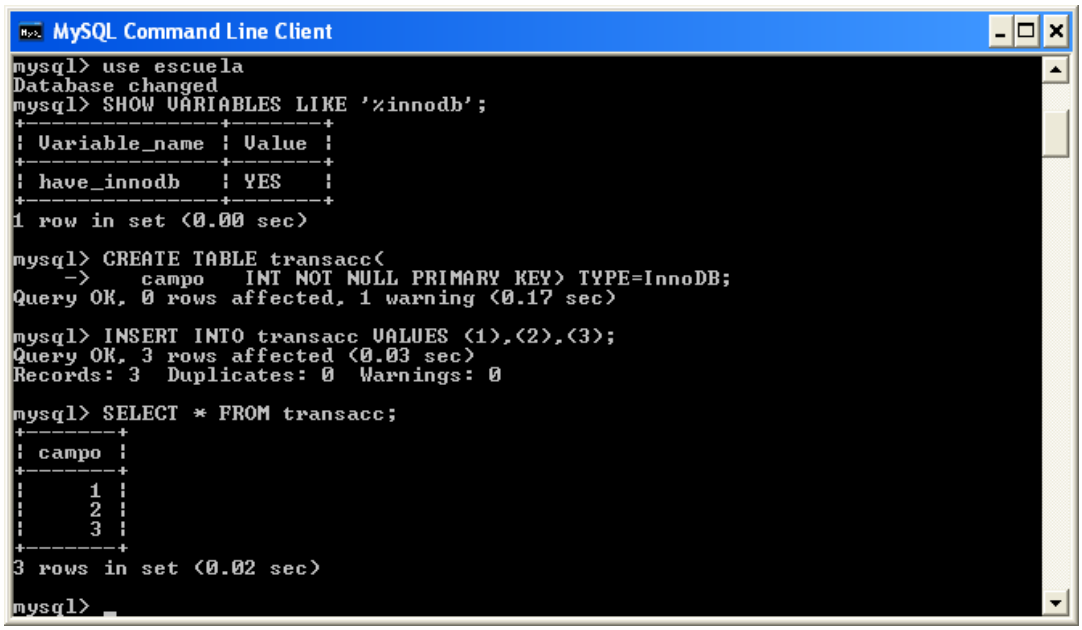


Figura 9.5: Creación de una tabla con el motor de base de datos InnoDB.

IX.2.2 Iniciando una transacción.

Ahora, con la instrucción BEGIN pondremos *Activa* la transacción. Una vez *Activa* agregaremos una tupla con el valor: 4, como se muestra en la *figura 9.6*.

BEGIN;

- **INSERT INTO** transacc **VALUES**(4);
- **SELECT** * **FROM** transacc;

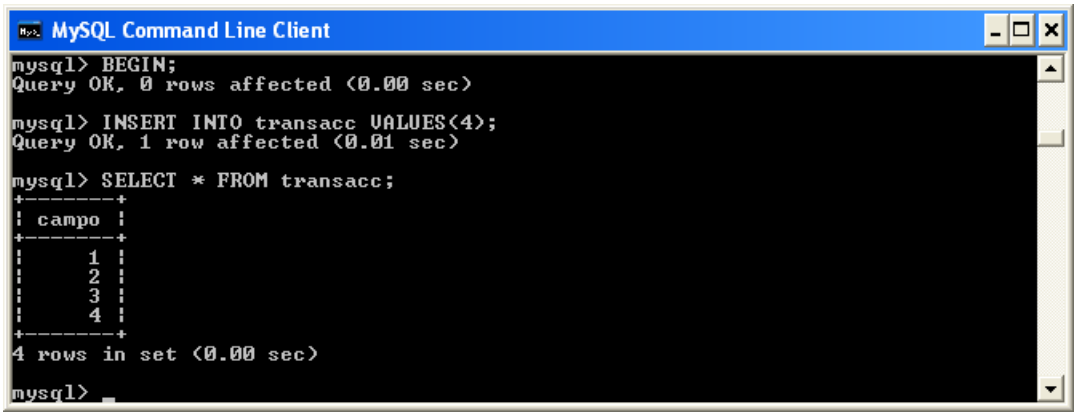


Figura 9.6: Inicio de una transacción.

Mientras la transacción esté *Activa* ésta se puede abortar con la instrucción ROLLBACK y entonces el cambio se pierde, como se muestra en la *figura 9.7*.

- **ROLLBACK**;
- **SELECT** * **FROM** transacc;

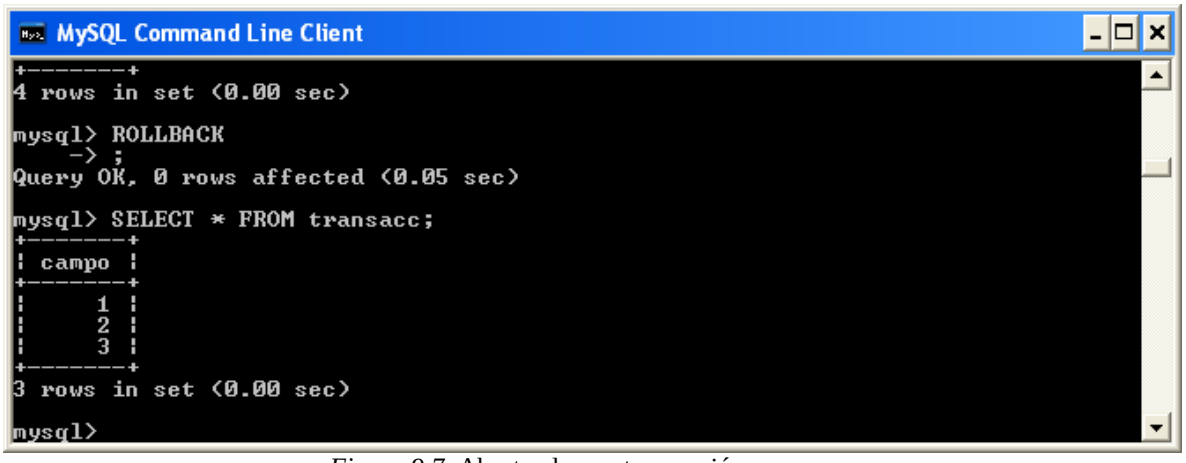


Figura 9.7: Abortando una transacción.

Comencemos de nuevo una transacción agregando la tupla con el valor “4” a la tabla Transacc:

- **BEGIN**;
- **INSERT INTO** transacc **VALUES**(4);
- **SELECT** * **FROM** transacc;
- **EXIT**;

Hasta este momento la transacción queda en el estado *Parcialmente Comprometida*. Si en lugar de abortar la transacción nos salimos con EXIT, como en la *figura 9.8.*

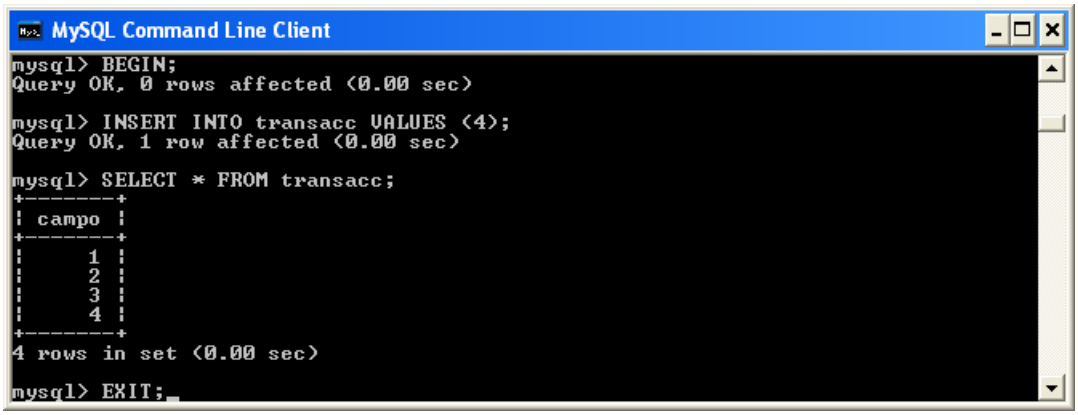


Figura 9.8: Salida de una transacción cuando aún no esta “comprometida”.

Entonces como la transacción no está en estado de *Comprometida*, al salir de la sesión los cambios se pierden, como se observa en la *figura 9.9.*

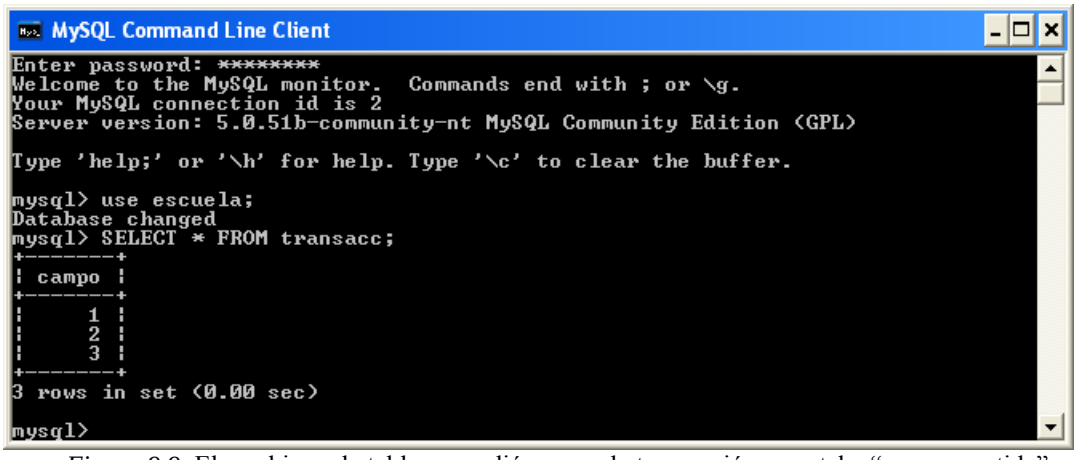


Figura 9.9: El cambio en la tabla se perdió porque la transacción no estaba “comprometida”.

Finalmente, para que la transacción pase al estado de *Comprometida* y que entonces, al salir de la sesión los cambios no se pierdan, es necesario dar el comando **COMMIT**.

- **BEGIN**;
- **INSERT INTO** transacc **VALUES**(4);
- **SELECT** * **FROM** transacc;
- **COMMIT**

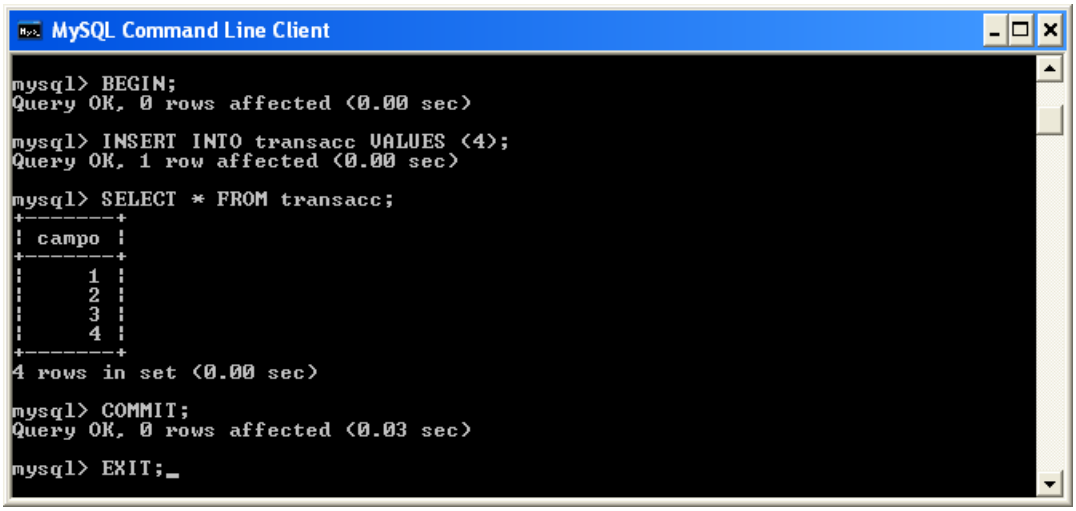


Figura 9.10: El cambio

De esta forma, cuando se da de baja la sesión y luego volvemos a entrar, los cambios hechos por la transacción sobre la tabla Transacc son permanentes.

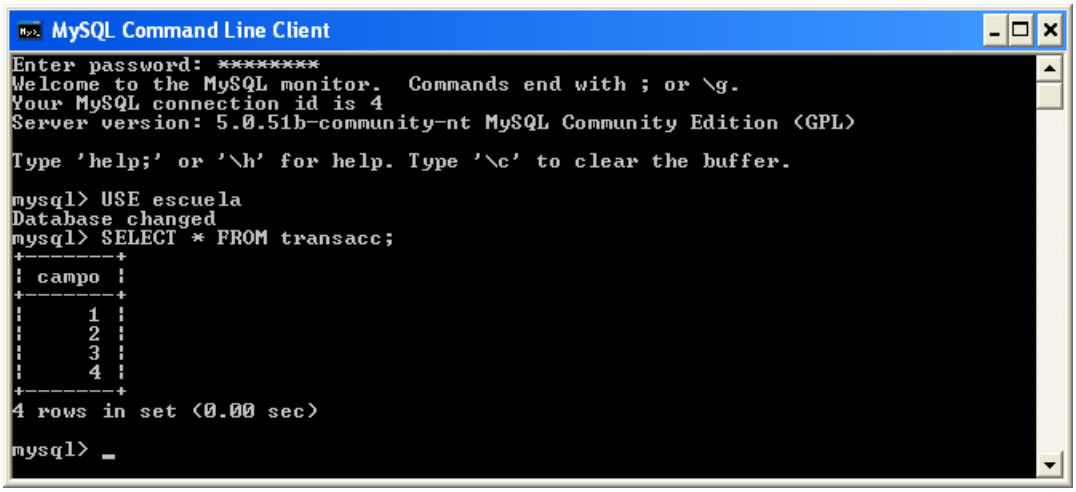


Figura 9.11: El cambio

Capítulo X Normalización.

X.1 Que es y para qué sirve la normalización.

Normalizar las tablas de una base de datos significa optimizar su diseño para no repetir datos innecesariamente y para prevenir problemas de inconsistencia. Con la normalización, los datos complejos se transforman en un conjunto de tablas simples que son más fáciles de entender y mantener. Como se ilustra en la *figura 10.1*, es mejor invertir un esfuerzo importante al diseño de la base de datos, esto ahorra mucho esfuerzo en las etapas posteriores del desarrollo de un sistema de software.

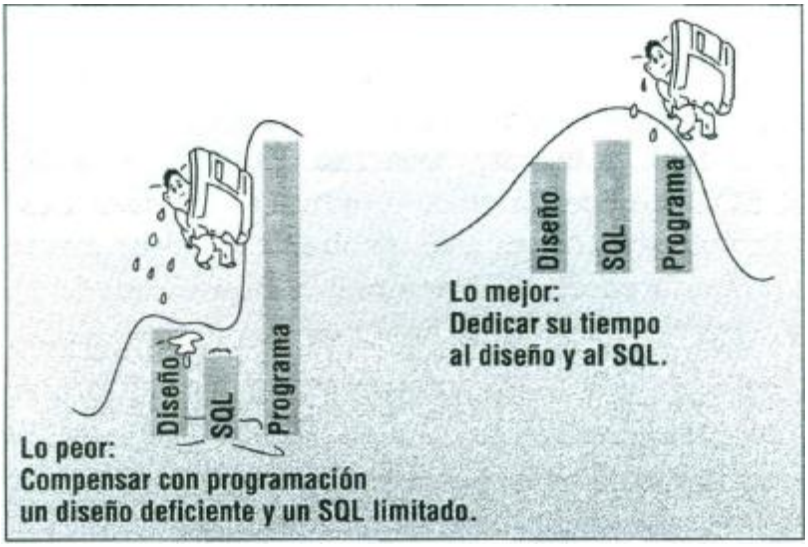


Figura 10.1: Es mejor invertir tiempo y esfuerzo a un buen diseño de la base de datos (Post, 2006).

E. F. Codd, fue el creador de las bases de datos relacionales, en 1970 definió las tres primeras formas normales, que son:

- 1FN (primera Forma Normal).

- 2FN (segunda Forma Normal).
- 3FN (tercera Forma Normal).

Se dice que una tabla cumple una determinada forma normal cuando satisface las restricciones impuestas por dicha norma. La idea es evitar los problemas que surgen cuando una base de datos está mal diseñada.

Durante el proceso de normalización hay que tener siempre en mente la posibilidad e descomponer una tabla en otras más pequeñas.

La normalización tiene como ventajas principales:

- *La precisión*.- Las interrelaciones entre las tablas consiguen mantener información diferente relacionada con toda exactitud.
- *La mínima redundancia*.- la información no está duplicada innecesariamente.

X.1.1 Dependencia funcional.

Antes de comenzar a estudiar las formas normales, definiremos el concepto de *dependencia funcional* ya que la normalización se basa en este concepto.

Primero definiremos un *esquema de una tabla* T(A, D), como un como un conjunto compuesto por:

- Un conjunto de atributos **A** sobre los que se definen todas las *ocurrencias* del esquema, es decir, las tuplas.
- Un conjunto de restricciones o *dependencias* **D** que debe satisfacer cualquier ocurrencia.

Dependencia funcional.- Sean $X \subseteq A$, $Y \subseteq A$ dos atributos. Existe *Dependencia funcional*, $X \rightarrow Y$, de Y con respecto a X cuando a todo valor de X le corresponde uno y sólo uno de Y.

Se dice que un atributo o conjunto de atributos B *dependen funcionalmente* del atributo o conjunto de atributos A ($A \rightarrow B$), si y sólo si a cada valor de A le corresponde un único valor de B. Por ejemplo, si se tiene el CURP de una persona, se puede saber su fecha de nacimiento, y si es hombre o mujer.

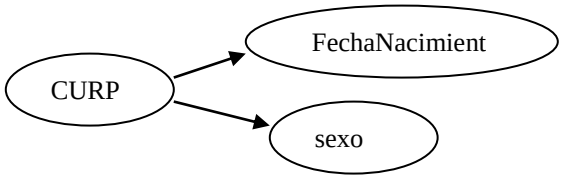


Figura 10.2: Dependencia funcional de los campos FechaNacimiento y sexo, del campo CURP.

En la *Figura 10.2* se dice que la `FechaNacimiento` y el `sexo` dependen funcionalmente del `CURP`.

Transitividad de una dependencia funcional.- Si Y depende funcionalmente de X y Z depende funcionalmente de Y, entonces se cumple que Z depende funcionalmente de X:

$(X \rightarrow Y)$ y $(Y \rightarrow Z) \Rightarrow (X \rightarrow Z)$.

Ejemplo: (Matricula → Grupo) y (Grupo → Aula) ⇒ (Matricula → Aula)

Es decir, un atributo Z es transitivamente dependiente de otro X si se conoce por diferentes vías, una directamente y otra a partir de otro atributo intermedio Y.

X.2 La primera forma normal (1NF).

Una tabla se encuentra en primera forma normal si y sólo si los valores que componen el atributo de una tupla son *atómicos*. Los atributos *atómicos*, son valores únicos y lo más pequeño posible, es decir, indivisibles. Además tiene una clave primaria, atributos no nulos y no tiene tuplas repetidas.

En la tabla de la *figura 10.3* se muestra el ejemplo de una tabla que no esta en su primera forma normal porque el campo Nombre puede dividirse en 3 campos pequeños y porque el campo Telefonos tiene muchos valores para una sola tupla, es decir, no hay un valor único para este campo.

CLIENTES

Clave Primaria		
ID_Cliente	**Nombre**	**telefonos**
038	Gervasio Perez Flores	58967123, 58967112
124	Eduardo Molina Saenz	23181534, 91801435

Figura 10.3: la tabla CLIENTES no está en la primera forma normal (1NF).

Para convertir la tabla a la primera forma normal:

CLIENTES

Clave Primaria					
ID_Cliente	**Nombre**	**ApellidoMat**	**ApellidoPat**	**telefono1**	**telefono2**
038	Gervasio	Perez	Flores	58967123	58967112
124	Eduardo	Molina	Saenz	23181534	91801435

Figura 10.4: la tabla CLIENTES está en la primera forma normal (1NF).

Ejercicio.- ¿Porqué la tabla MATERIALES de la *figura 10.5* no esta en la Primera forma normal 1FN?

MATERIALES

COD_MAT	DESCRIPCION	MEDIDAS
039	TORNILLO	3,5-5-7
067	ARANDELA	2-5
461	BROCA	5-6-7-8-9
057	TACO	5-6-7-8-9-10

Figura 10.5: Tabla MATERIALES.

Solución: como puede observarse, un material puede tener varias medidas, para cumplir con 1FN es necesario dividir la información en dos tablas, de esta forma, el campo MEDIDAS no tiene varios valores, como se observa en la tabla de la *figura 10.6*.

MATERIALES

COD_MAT	DESCRIPCION
039	TORNILLO
067	ARANDELA
461	BROCA
057	TACO

MAT-MED

COD_MAT	MEDIDAS
039	3,5
039	5
039	7
067	2
067	5
461	5
461	6
. . .	. . .
057	5
057	6
. . .	. . .

Figura 10.6: Normalización mediante la adición de otra tabla.

X.3 La segunda forma normal (2NF).

Una tabla está en la segunda forma normal cuando está en la primera forma normal (1NF) y además cada campo secundario (aquel que no pertenece a la clave principal) depende de la clave principal en su totalidad y no de una parte de ella.

Lo anterior se cumple cuando ninguna clave primaria es compuesta, es decir, esta forma normal sólo se considera si la clave principal de la tabla es compuesta, ya que si la clave principal es simple entonces la tabla ya se encuentra en segunda forma normal (si se encuentra en primera forma normal).

Ejercicio.- ¿Porqué la tabla PRESTAMOS de la *figura 10.7* no esta en la Segunda forma normal 2FN?

PRESTAMO

N. Usu	C. Libro	Título	Fecha Préstamo
1	3421	Fisica I	02/04/2009

Figura 10.7: Tabla "Prestamos".

Solución: El campo Título no depende del Número de usuario, sino de la clave del libro. Rediseñando tenemos las tablas de la *figura 10.8.*

PRESTAMOS

N usuario	Cod Libro	Fecha Préstamo
1	3421	02/04/2009

TITULOS

Cod Libro	Título
3421	Física I

Figura 10.8: Normalización mediante la adición de otra tabla.

X.4 La tercera forma normal (3NF).

Una tabla está en la tercera forma normal cuando esta en la segunda forma normal (2NF) y además cada campo que no sea llave primaria solo depende de la llave primaria o de las claves secundarias de la tabla y no depende de otro campo de tal forma que "no existen atributos no primarios que son *transitivamente dependientes* de cada posible clave de la tabla".

El objetivo de la tercera forma normal es eliminar cualquier dependencia transitiva. Una *dependencia transitiva* implica que se puede saber un campo secundario a través de otro campo que no es la clave principal.

En suma, para que una tabla cumpla con la tercera forma normal, un atributo secundario sólo se debe conocer a través de la clave principal o claves secundarias de la tabla y no por medio de otro atributo no primario.

A continuación se da un ejemplo de una dependencia transitiva, consideremos la tabla:

ALUMNO (MATRICULA, GRUPO, AULA) con los siguientes requerimientos:

- Un alumno sólo tiene asignado un grupo.
- A un grupo siempre le corresponde una única aula.

El atributo AULA es transitivamente dependiente de MATRICULA ya que se puede conocer además a través de GRUPO.

En la tabla de la *figura 10.9* se muestra el ejemplo de una tabla que no está en su tercera forma normal porque los campos Nombre y Telefono no dependen de la identidad de la orden (ID_Orden), sino del campo ID_Cliente.

ORDENES

Clave Primaria					
ID_Orden	**Fecha**	**ID_Cliente**	**Apellido1**	**Nombre**	**Telefono**
15	2009-01-25	038	Perez	Gervasio	58967123
17	2009-02-02	124	Molina	Eduardo	23181534

Figura 10.9: la tabla ORDENES no está en la tercera forma normal (3NF).

Para obtener la tercera forma normal es necesario tener las dos tablas de la *figura 10.10*.

ORDENES

Clave Primaria		Clave Externa
ID_Orden	**Fecha**	**ID_Cliente**
15	2009-01-25	038
17	2009-02-02	124

CLIENTES

Clave Primaria			
ID_Cliente	**Nombre**	**Apellido1**	**Telefono**
038	Gervasio	Perez	58967123
124	Eduardo	Molina	23181534

Figura 10.10: la tabla CLIENTES y la tabla ORDENES están en la tercera forma normal (3NF).

Capítulo XI Creación de índices.

XI.1 Conceptos básicos.

Los índices en las bases de datos tienen como objetivo agilizar el acceso a los datos. El índice es una estructura que mejora la velocidad de acceso a los registros de una tabla dentro de la base de datos. Una analogía es el acceso a un capítulo de un libro. Si la búsqueda se hace de forma secuencial habría que recorrer página por página hasta encontrar el capítulo buscado, sin embargo, si se consulta primero el índice, entonces podemos ir directamente a la página en donde se encuentra el capítulo. Con los índices en las bases de datos se recorren mucho menos tuplas en cada búsqueda, por lo tanto hay menor carga y mayor velocidad de acceso. Sin embargo hay que tomar en cuenta que se requiere el trabajo adicional de la generación de los índices. Además los índices ocupan espacio, y cuando la cantidad de datos es tal que se alcanza el límite de ocupación de la memoria, la creación de un índice puede representar un problema.

El índice consta de parejas en la que se asocia el elemento que se desea indexar con su posición en la base de datos. Para buscar un elemento que esté indexado, es necesario buscar en el índice este elemento, una vez que éste se encuentra, se devuelve el registro que se encuentra en la posición marcada por el índice.

Los índices pueden crearse utilizando uno o más campos de una tupla. Para decidir los campos que se usarán para construir el índice es mejor elegir aquellos sobre los cuáles se hacen búsquedas más frecuentes.

Los índices se construyen generalmente con árboles B, B+, B* o con una mezcla de ellos, aunque también existen otros métodos.

Los índices son una desventaja en las tablas las que se utiliza frecuentemente operaciones de escritura (Insert, Delete, Update), esto es porque los índices se actualizan cada vez que se modifica una columna, y esto requiere de tiempo. En tablas demasiado pequeñas no es necesario ganar tiempo en las consultas, por lo que no vale la pena el esfuerzo de crear el índice. Los índices tampoco son muy aconsejables cuando pretendemos que la tabla sobre la que se aplica devuelva una gran cantidad de datos en cada consulta. Por último hay que tener en cuenta que ocupan espacio y en determinadas ocasiones incluso más espacio que los propios datos aunque lo más común es que los índices ocupen menos espacio que las tablas.

XI.2 Indexación.

Los índices se organizan en base a *claves de búsqueda*. Una clave de búsqueda es un atributo o un conjunto de atributos que se utilizan para localizar una tupla en una tabla de la base de datos. Un índice se compone de dos campos, el primero contiene la clave de búsqueda y el segundo un apuntador con la dirección en la que se encuentra el elemento de la tabla asociado con la clave de búsqueda.

Los índices pueden ser *ordenados* o *asociativos*. Como su nombre lo indica, los índices *ordenados* se basan en la disposición "ordenada" de valores, es decir, los elementos se ordenan secuencialmente a partir de una clave de búsqueda principal. Mientras que los índices *asociativos* tienen sus claves de búsqueda distribuidas uniformemente por medio de alguna función de asociación.

En la *figura 11.1* se tiene un ejemplo de un *índice ordenado denso*, para localizar una tupla por medio de este tipo de índices se elige una clave de búsqueda cuyo valor sea igual al de la clave que se está buscando. En el caso de que existan varias tuplas con la misma clave, se comienza a buscar a partir de la primera tupla apuntada por el índice. En el índice ordenado denso, aparece un registro índice para cada valor de clave de búsqueda de la tabla indexada.

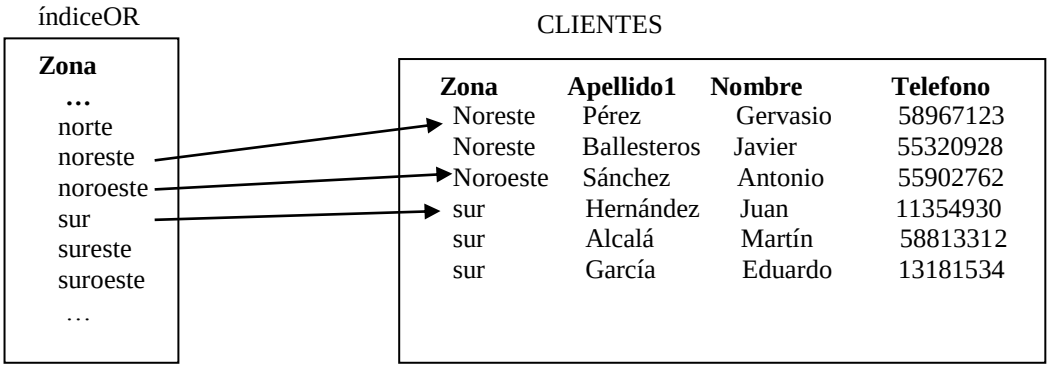


Figura 11.1: índice ordenado denso.

En el ejemplo de la *figura 11.1*, las búsquedas dentro de la tabla comienzan a partir de la zona (norte, sur,…) que se elija en el índice.

En la *figura 11.2* se tiene un ejemplo de un *índice ordenado disperso*, para localizar una tupla por medio de este tipo de índices se elige una clave de búsqueda cuyo valor sea igual al de la clave que se está buscando. En el caso de que la clave buscada no se encuentre en el índice, entonces se toma el elemento con el valor más grande que sea menor al del elemento buscado y entonces se busca en la base de datos a partir de la dirección de éste valor.

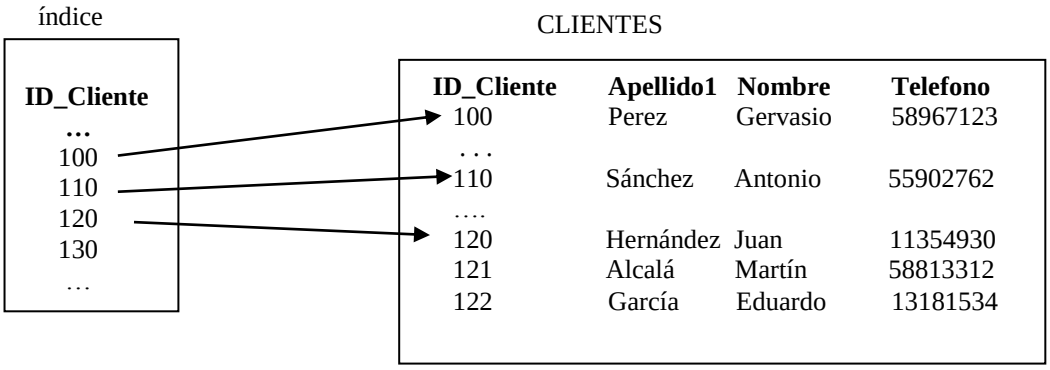


Figura 11.2: índice ordenado disperso.

En el ejemplo de la *figura 11.2*, si el elemento buscado es la tupla con ID_Cliente = 122, entonces se busca en la base de datos a partir del elemento cuya ID_Cliente es 120.

XI.2.1 Índices densos vs. Índices dispersos.

En general es más rápido localizar una tupla con índice denso que una con índice disperso, sin embargo un índice disperso ocupa menos espacio que uno denso y las operaciones de inserción y borrado son más rápidas que con el denso.

XI.3 Árboles B y árboles B+.

Un árbol es un grafo con un nodo raíz, aristas (o ramas) que conectan un nodo con otro estableciendo una relación de padre-hijo con la condición de que cada hijo tenga un solo padre. Un ejemplo de árbol se ilustra en la *figura 11.3*.

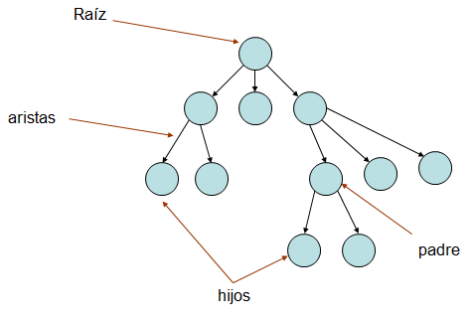


Figura 11.3: Partes de un árbol.

Además un árbol tiene las siguientes características:

- No existen nodos aislados (es un grafo conectado).
- Cada nodo tiene 0 o más nodos hijos hacia los cuáles tiene una arista dirigida.
- Hay un solo nodo que no tiene padre al cual se le llama *nodo raíz*.
- Cada nodo tiene exactamente un padre (excepto el *nodo raíz*).
- A los nodos que no tienen hijos se les denomina *hojas.*
- Los nodos hijos de un mismo padre son *hermanos.*
- Los nodos que no son hojas son *nodos internos* o *nodos no terminales*.
- No hay ciclos.

Cuando se trabaja con árboles se utilizan los siguientes términos (los cuáles se ilustran en la *figura 11.4*).

- *Camino* es una secuencia de ramas contiguas que van de un nodo n_x a otro nodo n_y.
- *Longitud de un camino* es el número de ramas entre el nodo n_x y el nodo n_y.
- *Nivel* de un nodo es la longitud del camino que lo conecta con el nodo raíz.
- *Profundidad del árbol* es la longitud del camino más largo que conecta el nodo raíz con una hoja.

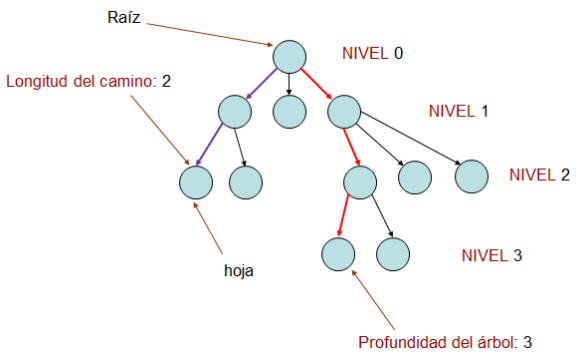


Figura 11.4: Términos para definir a un árbol.

Los árboles B y B+ son *árboles de búsqueda*. Un árbol de búsqueda es "un tipo especial de árbol que sirve para guiar la búsqueda de un registro (tupla) dado el valor de uno de sus campos (atributos)" [Elmasri & Navathe, 2008]. En un árbol de búsqueda de orden *n* cada nodo contiene cuando mucho *n*-1 valores de búsqueda y *n* apuntadores a un nodo hijo.

Los árboles de búsqueda suelen utilizarse para localizar tuplas que están almacenadas dentro de un archivo en disco. El valor de cada nodo del árbol corresponde con uno de los valores de uno de los campos de la tupla.

Un árbol B tiene las siguientes características:

- Cada nodo tiene la forma

  $[P_1, (k_1, Ptr_1), P_2, (k_2, Ptr_2)],\ldots, P_{n-1}, (k_{n-1}, Ptr_{n-1}), P_n]$

  Donde: P_i = apuntador a un nodo hijo, se le llama *apuntador de datos*.
  k_i = valor al cual se le asocia un apuntador Ptr_i
  Ptr_i = dirección en el disco donde se encuentra guardada una tupla cuyo campo contiene el valor k_i, Se le llama *apuntador a datos.*

- Dentro de cada nodo: $k_1 < k_2 < \ldots < k_{n-1}$
- Para todos los valores de un campo clave de búsqueda con valor = *x* del subárbol al que apunta P_i se debe cumplir que $k_{i-1} < x < k_i$ $(1 < i < n)$

- Cada nodo que no es raíz ni hoja tiene por lo menos n/2 apuntadores de árbol.
- Un nodo con *n* apuntadores de árbol tiene *n*-1 valores del campo clave de búsqueda y *n*-1 apuntadores a datos.
- Todos los nodos hoja se encuentran en el mismo nivel.

A continuación, en la *figura 11.5* se muestra un ejemplo de árbol B de búsqueda.

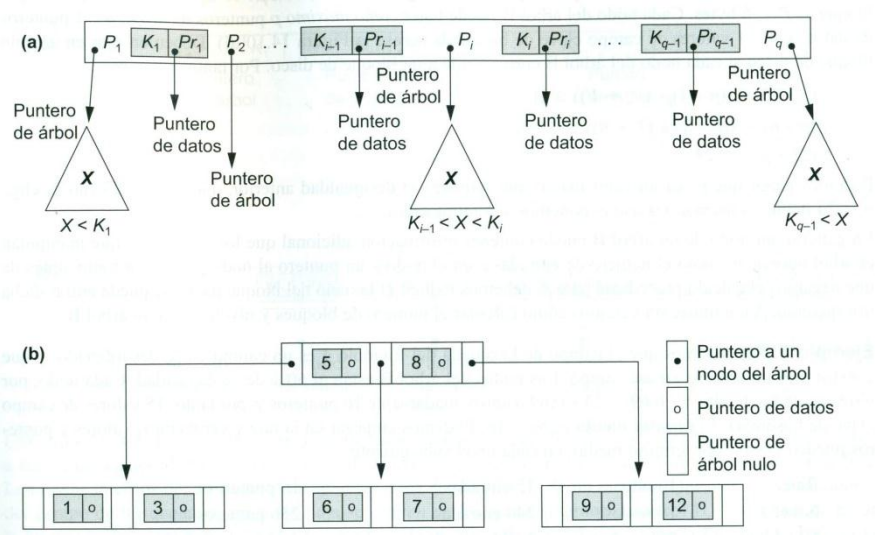


Figura 11.5: Árbol B de búsqueda [Elmasri & Navathe, 2008].

Cuando se inserta una nueva tupla, es necesario actualizar el árbol de búsqueda, para que se incluya un nuevo valor con su respectivo apuntador a esta nueva tupla. Los algoritmos para hacer esto en un árbol B no garantizan que el árbol siga equilibrado, es decir, que todas las hojas queden al mismo nivel. Cuando un árbol de búsqueda está equilibrado la velocidad de búsqueda es uniforme, ya que es independiente del valor de la clave de búsqueda.

Los árboles B+ son una variante de los árboles B. Mientras que en un árbol B cada valor del campo de búsqueda aparece una vez en algún nivel del árbol junto con su apuntador a datos, en el árbol B+ los apuntadores a datos se guardan solo en los nodos hoja del árbol, por lo que la estructura de los nodos hoja es diferente de la estructura de los nodos internos.

Un árbol B+ satisface estas propiedades:

- Todos los caminos de la raíz a las hojas tienen la misma longitud.
- Cada nodo que no es raíz ni hoja tiene entre n/2 y n hijos, donde n está fijo para cada árbol en particular.
- Un nodo hoja tiene entre (n-1)/2 y (n-1) valores.
- Casos especiales:
  – si la raíz no es una hoja, tiene como mínimo 2 hijos.
  – Si la raíz es una hoja, puede tener entre 0 y (n-1).

Para ver con detalle los algoritmos de búsqueda, inserción y eliminación para los árboles B y B+ se puede consultar [Elmasri & Navathe, 2008, pag. 442-453] y [Silberschatz et al., 2006, pag. 408-418].

XI.4 Definición de índice en SQL.

La sintaxis para crear un índice en SQL es la siguiente:

```
CREATE [UNIQUE] INDEX <nombre_indice> ON <nombre_tabla>(
                        <nombre_campo> [ASC | DESC]
                       {,<nombre_campo> [ASC | DESC]})
              );
```

La pálabra clave UNIQUE especifica que no pueden existir claves duplicadas en el índice.

ASC | DESC especifican el criterio de ordenación elegido, ascendente o descendente, por defecto es ascendente

Por ejemplo, si ya tenemos creada la tabla Clientes, podemos crear un índice único en el campo IdCliente Esto nos permitirá buscar mucho más rápido por el campo IdCliente asegurando que no se tengan dos IdCliente iguales.

```
CREATE UNIQUE INDEX UIX_Clientes_IdCliente ON Clientes (IdCliente);
```

Tomar en cuenta que las claves primarias son índices y que los nombres de los índices deben ser únicos.

Para agregar un índice a una tabla:

```
ALTER TABLE nombre-de-la-tabla ADD INDEX nombre-del-indice (columna(s)-que-forma(n)-el-indice)
```

Para eliminar un índice debemos emplear la sentencia DROP INDEX.

```
DROP INDEX <nombre_tabla>.<nombre_indice>;
```

Capítulo XII Tendencias en bases de datos.

XII.1 Nuevos retos. Factores y líneas de evolución.

Las bases de datos sirven para el manejo de la información. Sabemos que la información es muy valorada en cada una de las instituciones que utilizan las computadoras para su manejo, como empresas, dependencias del gobierno, hospitales, escuelas, museos, etc. Las bases de datos deben atender a múltiples usuarios y a diferentes aplicaciones.

Se dice que estamos en la tercera generación de las bases de datos, ésta se caracteriza por: "Proporcionar capacidades de gestión de datos al igual que sus predecesores, permitiendo que grandes cantidades de datos persistentes sean compartidos por muchos usuarios. También proporcionan gestión de objetos, permitiendo tipos de datos mucho más complejos, objetos multimedia, datos derivados, encapsulamiento de la semántica de los datos, así como otras nuevas capacidades. Algunos proporcionan incluso gestión de conocimiento, soportando un gran número de reglas complejas para inferencia automática de información y también para mantener las restricciones de integridad entre datos" [Cattell, 1991].

Actualmente existe una fuerte competencia entre las diferentes empresas telefónicas y de comunicaciones. Para ganar mercado se ha desencadenado una búsqueda constante de nuevas tecnologías que proporcionen al usuario acceso a cada vez más información. Es de esperarse que en un futuro cercano los usuarios puedan tener acceso a cada vez más bases de datos desde sus dispositivos móviles. Esto implica el desarrollo de bases de datos con una gran cantidad de usuarios que utilizan su información al mismo tiempo (bases de datos paralelas). Tecnologías que aumenten la velocidad de acceso a los datos (datos en memoria principal) y una evolución en los sistemas de seguridad de los datos. Se requieren además, otras formas de modelar el mundo real complejo, es decir, el que no se puede modelar con bases de datos relacionales (bases de datos orientadas a objetos). También se requieren sistemas que ayuden al ser humano a interpretar la información (minería de datos, bases de datos deductivas). En la siguiente sección se mencionan las tendencias de la tecnología en lo que respecta a las bases de datos.

XII.2 Otros tipos de bases de datos.

Bases de datos distribuidas.- Son bases de datos ubicadas en diferentes computadoras y conectadas a través de la red. En este tipo de bases de datos se puede procesar la información de forma local, es decir, cada base de datos hace sus operaciones de forma independiente de las demás, o bien se pueden realizar operaciones de forma distribuida, lo que significa hacer uso de las bases de datos de las otras computadoras (nodos). El usuario tiene acceso a los datos de las otras computadoras de la misma forma en la que tiene acceso a los de la suya.

Bases de datos orientadas a objetos.- El modelo de base de datos orientado a objetos es un modelo de datos lógico que captura la semántica de los objetos que se utilizan en la programación orientada a objetos. Por lo tanto manejan los conceptos básicos del diseño orientado a objetos que son: la abstracción, el encapsulamiento, la herencia y el polimorfismo.

- La *abstracción* es el paso del mundo real al mundo virtual, con ella se hace una descripción especial simplificada de un sistema. Para modelar al objeto se toman ciertos rasgos y se suprimen otros. Una buena abstracción es aquella que logra hacer énfasis en los detalles significativos o relevantes del objeto y discrimina cualquier otra característica. Con la abstracción se consigue un buen mapeo de los objetos del mundo real a los objetos del sistema.

- El *encapsulamiento* es el proceso mediante el cual se ocultan todos los detalles de un objeto que no son esenciales para su uso desde el exterior. Este principio nos indica que debemos ocultar al exterior la complejidad con la que está construido nuestro objeto ya que para que los otros objetos lo usen o se comuniquen con él, el conocimiento de esta complejidad es innecesaria. La principal ventaja del encapsulamiento es la seguridad. Con la seguridad se promueve la integridad del objeto. Una falla en la integridad produce datos incoherentes (datos corrompidos). Con el encapsulamiento los usuarios se dan cuenta de las operaciones que pueden solicitar al objeto pero desconocen los detalles de cómo se lleva a cabo la operación.

- La *herencia* ayuda a clasificar ordenadamente un sistema complejo ya que es una forma de clasificar u ordenar las abstracciones por niveles.

- El *polimorfismo* es la propiedad que tiene un objeto de tipo A de mostrarse como de tipo B. Por ejemplo, Si tenemos dos clases X y Z que tienen el mismo ancestro W, entonces podemos usar una variable de clase W que pueda tomar el valor de un objeto de clase X o también el valor de un objeto de clase Z.

Bases de datos objeto-relacionales.- Estas bases de datos comparten las características de las bases de datos relacionales y las orientadas a objetos, ya que son capaces de almacenar objetos en bases de datos relacionales. Este tipo de bases de datos tienen la intención de ayudar a los programadores que trabajan con programación orientada a objetos y que utilizan bases de datos relacionales.

Bases de datos deductivas.- Por medio de la programación lógica y un sistema de almacenamiento de los datos eficiente y fiable que también almacena reglas deductivas, estos SGBD tienen la capacidad de deducir, inferir u obtener información nueva a partir de los datos almacenados

Bases de datos multimedia.- Son las que soportan los datos multimedia (videos, sistemas de información geográficos, enciclopedias electrónicas, música, televisión,…). Los datos multimedia tienen la característica de ser datos muy voluminosos por lo que requieren de un manejo especial para que la operación con estos datos no sea demasiado lenta. Además, son necesarias interfaces especiales para poder escuchar, y ver su contenido, de tal forma que se sincronice el video con el audio, que no se degrade la calidad de las imágenes ni se distorsionen los colores.

Bases de datos temporales.- Distinguen dos aspectos de los datos en función del tiempo; el "tiempo de validez", que indica el período en el cual un hecho es verdad en el mundo real, y el "tiempo de transacción", que indica el período en el cual un hecho está guardado en la base de datos. En este tipo de base de datos se almacenan datos históricos y datos actuales, esta información es muy útil en el área de finanzas (operaciones bursátiles, contabilidad, cuentas bancarias,…), en el área de reservas (vuelos, trenes, hoteles), y en el monitoreo del clima, entre otras.

Bases de datos activas.- Contienen un subsistema que permiten la definición y gestión de reglas. El conocimiento que provoca una reacción se elimina de los programas de aplicación y se codifica en forma de reglas activas. De este modo, al encontrarse las reglas definidas como parte del esquema de la base de datos, se comparten por todos los usuarios, en lugar de estar replicadas en todos los programas de aplicación. Cualquier cambio sobre el comportamiento reactivo se puede llevar a cabo cambiando solamente las reglas activas, sin necesidad de modificar las aplicaciones. Un SGBD de este tipo debe ser capaz de monitorear y reaccionar oportunamente ante ciertos eventos. Es decir, cuando se producen ciertas condiciones, el SGBD ejecuta automáticamente ciertas acciones. Uno de los problemas que ha limitado el uso extensivo de reglas activas, a pesar de su potencial para simplificar el desarrollo de bases de datos y de aplicaciones, es el hecho de que no hay técnicas fáciles de usar para diseñar, escribir y verificar reglas.

Bases de datos seguras.- Son bases de datos organizadas en múltiples niveles, cada nivel contiene objetos con diferentes niveles de confidencialidad y usuarios que pueden agruparse por roles, cada rol implica distintos privilegios sobre la información. El diseño y manejo de estas bases de datos esta orientado a atender los aspectos más importantes de la seguridad, que son:

- La confidencialidad: no mostrar ciertos datos a usuarios no autorizados y protección de datos personales.
- Accesibilidad: la información debe estar disponible y también el acceso a los servicios.
- Integridad: Los datos deben ser confiables, se debe garantizar que no se modificaron indebidamente.

Bases de datos difusas.- Una de las características del lenguaje natural, que hace difícil su utilización en sistemas computacionales es su imprecisión. Por ejemplo conceptos como alto y bajo, o pequeño y grande, tienen significados diferentes de acuerdo al contexto en el que se estén utilizando, e incluso dentro del mismo contexto, pueden significar cosas diferentes para diferentes individuos. El modelo relacional no permite el procesamiento de consultas del tipo *"Encontrar a todos los alumnos cuya calificación no sea muy baja"* dado que ni el cálculo ni el álgebra relacional, que establecen el resultado de cualquier consulta como una nueva relación, tienen la capacidad de permitir consultas de una manera difusa. Las bases de datos difusas encajan en el campo de la Inteligencia Artificial, ya que permiten el manejo de información con una terminología que es muy similar a la del lenguaje natural.

Almacenes de datos (data warehouses).- Un almacén de datos es una colección integrada de datos clasificados por temas. Almacenan información histórica que se extrae de fuentes múltiples, heterogéneas, autónomas y distribuidas. Su contenido es variable en el tiempo y tiene la finalidad de ayudar en el proceso de la toma de decisiones. Tiene las características que le permiten proporcionar idóneamente los datos a la minería de datos, ya que en un almacén de datos la información está limpia, independiente, coherente y unificada. El almacén de datos puede compararse con el expediente de la institución o empresa, sin embargo no debe utilizarse para trabajar con los datos de uso actual.

Minería de datos.- Parte del hecho de que las bases de datos tienen un conocimiento implícito el cual se puede extraer y encontrar patrones de comportamiento que permitan hacer predicciones. Este conocimiento también puede servir para segmentar los datos con el objeto de aumentar la velocidad de acceso. Además, con este conocimiento también se pueden hacer clasificaciones. La minería de datos usa técnicas de la estadística (análisis de varianza, regresión, prueba chi-cuadrado, análisis de agrupamiento,…) y de la inteligencia artificial (redes neuronales, árboles de decisión, algoritmos genéticos, sistemas expertos,…). La clasificación y predicción de los datos puede ser de gran utilidad en los negocios para entender como se comportan los clientes y así aumentar las ventas, evitar pérdidas o contactar clientes potenciales. La minería de datos tiene una amplia gama de aplicaciones en diversas áreas, por ejemplo en el estudio del comportamiento de los empleados de una empresa, de los usuarios de internet, de los fraudes con tarjetas bancarias, el monitoreo de los cables de alta tensión, etc.

Las áreas de aplicación de las bases de datos se extenderán conforme los avances de los Sistemas Gestores de Bases de Datos permitan satisfacer las exigencias de la globalización de la información de la cultura, de la ciencia, de la industria y del comercio.

Apéndice A: conectando Java con una base de datos MySQL.

El objetivo de este apéndice es proporcionar a los alumnos lo necesario para establecer una conexión entre una base de datos MySQL y Java, esto les será de gran utilidad para las UEAs: Proyecto de SW 2, Proyecto de SW 3, Proyecto Terminal I, Proyecto Terminal II.

Lo primero que se necesita para establecer la conexión es el *conector*, el cual es un programa ejecutable llamado "MySQL Connnector/J" que se encuentra en: http://dev.mysql.com/downloads/connector/j/. Cuando el ambiente en el que se trabaja Java es Netbeans, el conector debe estar en la carpeta `Libraries` del proyecto en cuestión, que es la carpeta en la que se encuentra el JDK.

A continuación se presenta la clase `Conecta` [Deitel & Deitel, 2008] que sirve para establecer la conexión, en la *figura 1* se explican cuáles son los atributos de la clase que contienen la información necesaria: el nombre del usuario, en este caro el administrador de la base de datos, llamado "root", el password para entrar a la base de datos y su URL

.

```
package com.util;

import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Connection;

public class Conecta {
  private static Connection con;
  private static String usuario = "root";
  private static String pwd     = "ueadb01";
  private static String url = "jdbc:mysql://localhost/baseDatos";
  private Conecta conecta;

  private Conecta(){}
  public static Connection getCon()
  {
      try {
      …

      return con;
  }
```

Usuario administrador

Password para entrar a BD

URL de la base de datos

Figura 1: Atributos necesarios para la conexión.

En la *figura 2*: se puede apreciar como funciona el método `getCon()`. Primero se instancia el controlador de la base de datos y posteriormente como se establece la conexión por medio del método `getConnection` al que se le envían como parámetros el URL de la base de datos, el usuario y el password. Además, puede observarse que el método `getCon()` de la clase `Conecta`, regresa el objeto `con` de la clase `Connection`. También se contemplan dos excepciones; una para el caso en el que no se puede instanciar el controlador de la base de datos y la otra para cuando no se pudo establecer la conexión con la base de datos.

```
public class Conecta {
  private static Connection con;
  private static String usuario = "root";
  private static String pwd     = "ueadb01";
  private static String url = "jdbc:mysql://localhost/baseDatos";
  private Conecta conecta;

  private Conecta(){}
  public static Connection getCon()
  {
      try { //carga la clase controlador
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url,usuario,pwd);
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
        catch(SQLException ex) {
            ex.printStackTrace();
        }
      return con;
  }
```

Controlador de la BD

Establece conexión con la BD

El método getCon() regresa una "conexión"

Figura 2: Estableciendo la conexión.

En la *figura 3* se presenta otro método de la clase `Conecta`, llamado `Close` el cual sirve para cerrar la conexión una vez que ya se ha hecho la operación deseada en la base de datos. Es importante cerrar la conexión después de cada operación para evitar problemas.

```
public class Conecta {
  private static Connection con;
  private static String usuario = "root";
  private static String pwd     = "ueadb01";
  private static String url = "jdbc:mysql://localhost/baseDatos";
  private Conecta conecta;

  private Conecta(){}
  public static Connection getCon()
  {
     try {

      ...
      return con;
  }

  public static void close()
  {
      try {
          if(con != null)
             con.close();
      } catch (SQLException ex) {
          ex.printStackTrace();
      }
  }
}
```

El método close cierra la "conexión"

Figura 3: Método para cerrar la conexión.

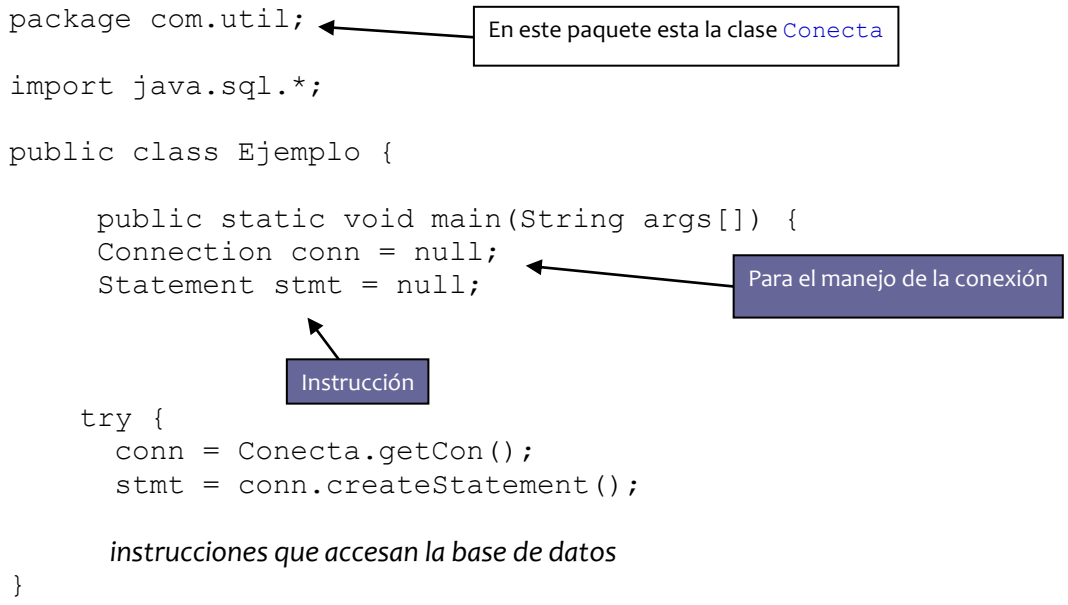
La clase `Conecta` se utiliza desde otras clases para hacer diversas operaciones en una base de datos MySQL, a continuación, en la *figura 4* y en la *figura 5* se da un ejemplo de cómo se hace un `UPDATE` en la base de datos desde Java.

Primero es necesario incluir el paquete en donde se encuentre la clase `Conecta`, en nuestro caso, esta clase se elaboró dentro del paquete `com.util`. El programa que utilice la clase `Conecta` necesita un objeto de clase `Connection` y es este objeto el que permite hacer las operaciones para establecer la conexión, cerrarla, y ejecutar comandos de lectura y actualización.

Para ejecutar una instrucción MySQL, se requiere de un objeto de clase `Statement`. La instrucción a ejecutar es un método del objeto `stmt` (instrucción), `executeUpdate` para el caso de hacer una actualización y `executeQuery` para el caso de una consulta, estas son las dos instrucciones utilizadas más comúnmente.

En la *figura 4* se muestra la forma de instanciar la conexión y el objeto `stmt` (instrucción). En la *figura 5* se observa un ejemplo de cómo se hace una actualización en una base de datos con el

método `executeUpdate`, también se puede ver el manejo de las excepciones y como hay que cerrar la conexión una vez que se llevó a cabo la operación deseada.




```
package com.util;

import java.sql.*;

public class Ejemplo {

     public static void main(String args[]) {
     Connection conn = null;
     Statement stmt = null;

    try {
      conn = Conecta.getCon();
      stmt = conn.createStatement();

      instrucciones que accesan la base de datos
}
```

Figura 4: Atributos necesarios para hacer uso de la clase `Conecta`.

```
public class Ejemplo {
   public static void main(String args[]) {
   Connection conn = null;
   Statement stmt = null;

   try {
    conn = Conecta.getCon();
    stmt = conn.createStatement();
    stmt.executeUpdate("insert into nombreTabla(atrib1, atrib2,…,
atribN)
      values('Valor1','Valor2',…,'valor N")");
   } catch (SQLException e) {
      System.out.println("error: no se pudo establecer conexión con
BD");
      e.printStackTrace();
   } catch (Exception e) {
      System.out.println("other error:");
      e.printStackTrace();
   } finally {
      try {
        stmt.close();
        conn.close();
      } catch (Exception e) {
      }
    }
  }
}
```

Ejemplo de cómo se hace un UPDATE en la BD

Hay que cerrar la instrucción y la conexión

Figura 5: Uso del método `executeUpdate()`

En la *figura 6* podemos apreciar que para hacer una consulta a la base de datos se requiere de un objeto adicional de clase `ResultSet` (conjunto de resultados), el cual sirve para guardar los resultados de la consulta. Al terminar la operación con la base de datos, no solo hay que cerrar la instrucción y la conexión, sino también el conjunto de resultados.

```
public class Ejemplo {
   public static void main(String args[]) {
   Connection conn = null;
   Statement stmt = null;
   ResultSet resultSet = null; // para manejar los resultados

   try {
    conn = Conecta.getCon();
    stmt = conn.createStatement();
    resultSet = stmt.executeQuery(
                            "SELECT atrib1, atrib2,…,FROM nombreTabla");
     ... Procesamiento de los resultados de la consulta . . .
   } catch (SQLException e) {
     System.out.println("error: no se pudo establecer conexión con BD");
      e.printStackTrace();
   } catch (Exception e) {
      System.out.println("other error:");
      e.printStackTrace();
   } finally {
      try {
        resultSet.close();
        stmt.close();
        conn.close();
      } catch (Exception e) {
      }
    }
  }
}
```

Ejemplo de cómo se hace un SELECT en la BD

Hay que cerrar la instrucción y la conexión pero primero el conjunto de resultados.

Figura 6: `executeQuery`, método para ejecutar una consulta (SELECT).

Finalmente, en la *figura 7* podemos observar la manera en la que se obtienen los resultados de la consulta. En el objeto `metaDatos` de clase `ResultSetMetaData` están guardados datos como el nombre y número de columnas. Los valores producto de la consulta se encuentran en el objeto `resultset` y se obtienen mediante el método `getObject`. El método `next` indica si todavía hay otro valor en `resultset` devolviendo "verdadero", devuelve "falso" cuando ya no existe otro elemento más.

```
public class Ejemplo {
   public static void main(String args[]) {
   Connection conn = null;
   Statement stmt = null;
   ResultSet resultSet = null; // para manejar los resultados

   try {
    conn = Conecta.getCon();
    stmt = conn.createStatement();
    resultSet = stmt.executeQuery(
                          "SELECT atrib1, atrib2,…,FROM nombreTabla");
     ... Procesamiento de los resultados de la consulta . . .
     ResultSetMetaData metaDatos = resultSet.getMetaData();
     int numCols = metaDatos.getColumnCount();
     System.out.println( "Tabla nombreTabla de la BD nombreBD: \n");
     for(int i = 1; i<= numCols; i++)
      System.out.printf("%-8s\t",metaDatos.getColumnName(i));

     while(resultSet.next())
     {
       for(int i = 1; i<= numCols; i++)
          System.out.printf("%-8s\t", resultSet.getObject(i));
       System.out.println();
     }

   } catch (SQLException e) {
```

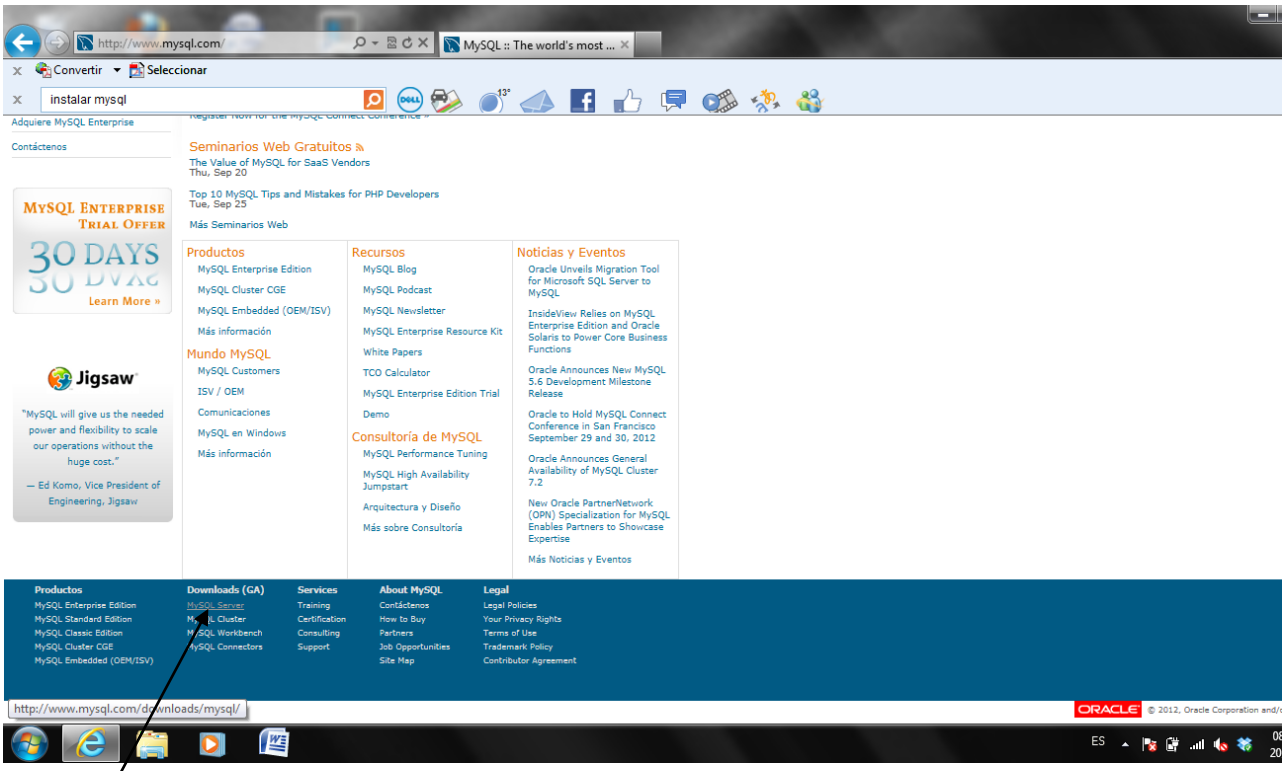
Ejemplo de cómo se hace un SELECT en la BD

Figura 7: Procesamiento de los resultados de la consulta.

Apéndice B: como instalar una base de datos MySQL.

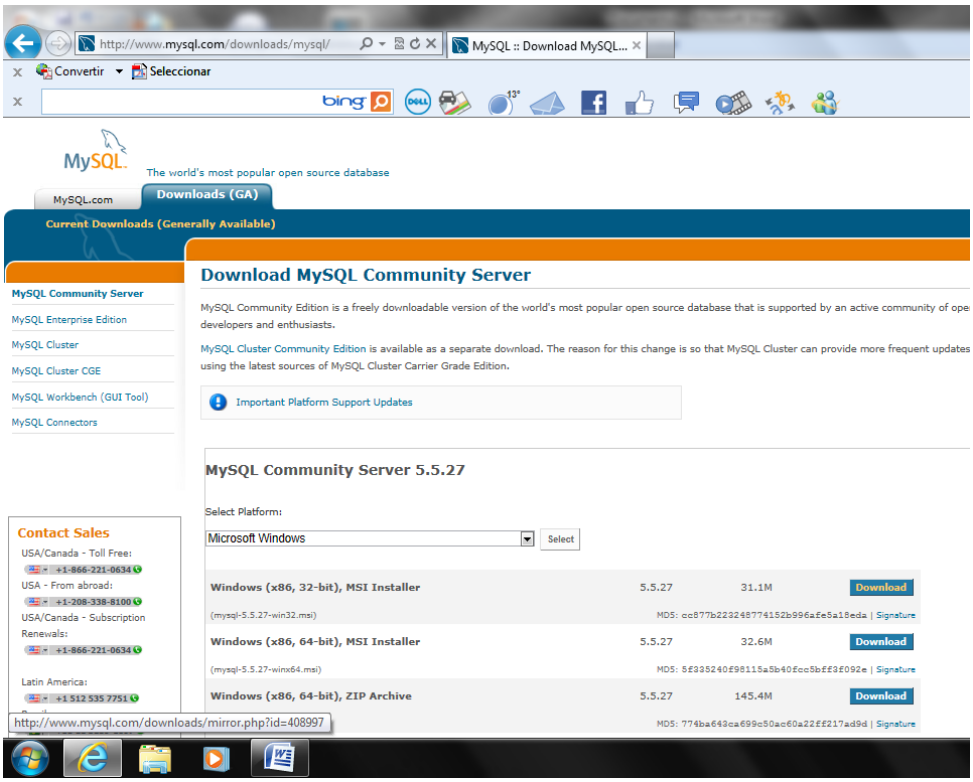
A continuación se señalan los pasos que deben seguirse para instalar MySQL en una computadora con Windows, los pasos para instalarlo en otros sistemas operativos son muy similares, solo se debe elegir la opción con el sistema operativo deseado al principio.

El sitio oficial MySQL es: http://www.mysql.com/, hasta debajo de la página principal se encuentras las opciones de descargas (downloads), hay que seleccionar **MySQLServer.**

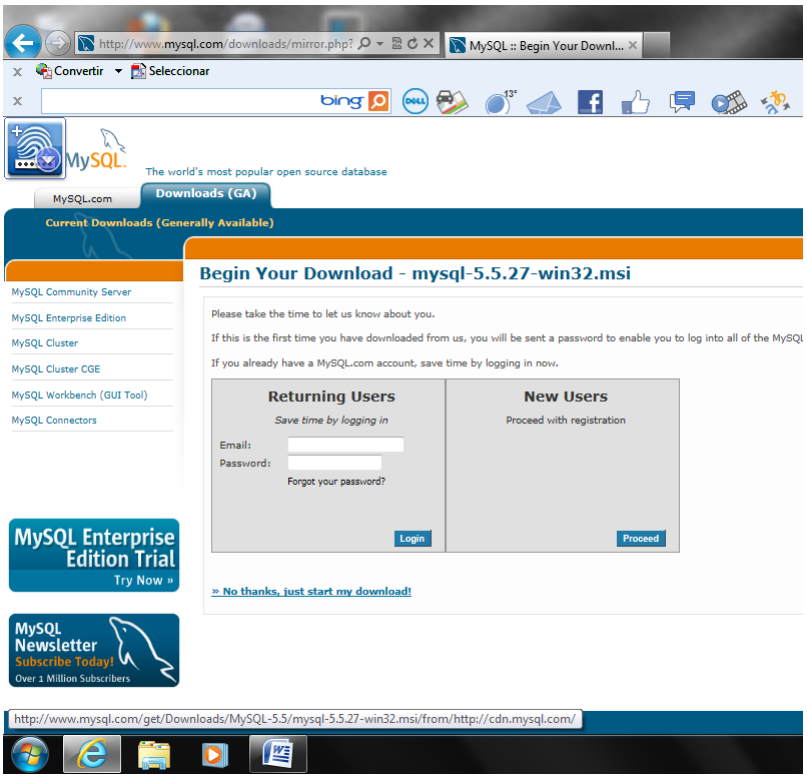


MySQLServer.

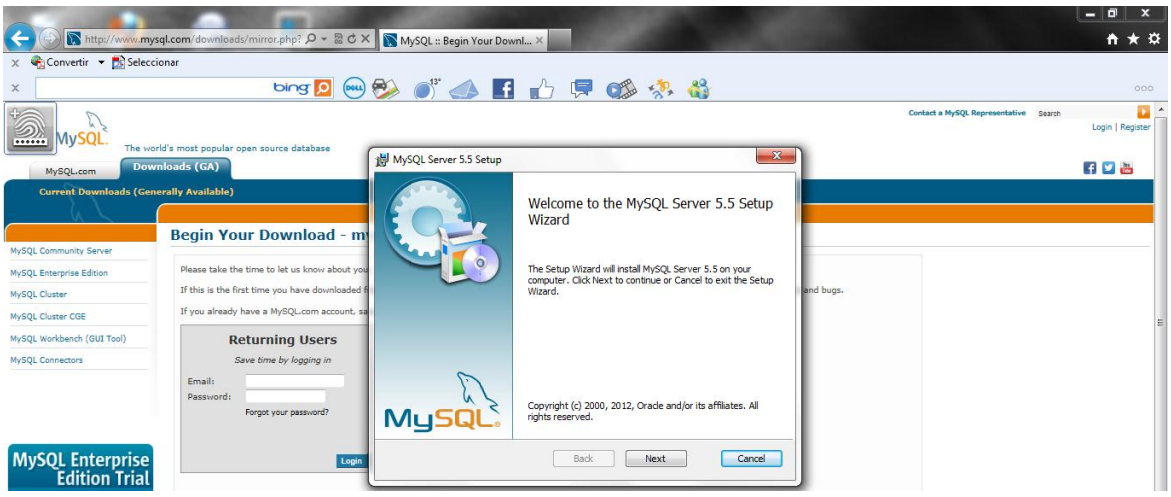
En el siguiente ejemplo se seleccionó la opción para Windows de 32 bits.



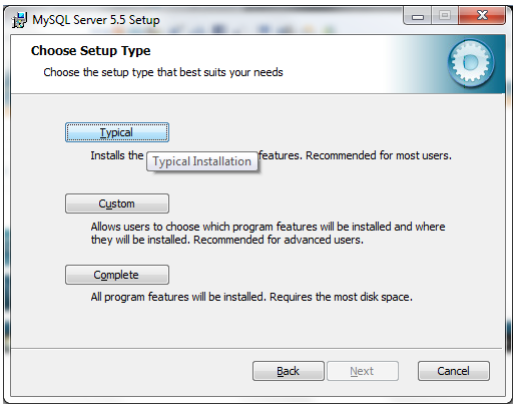
No es necesario registrarse en la siguiente pantalla, basta con seleccionar “just start my download”.

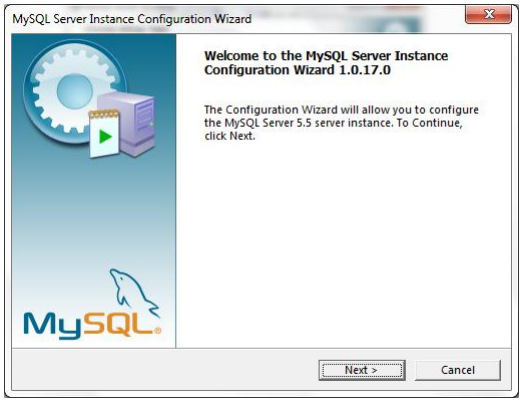


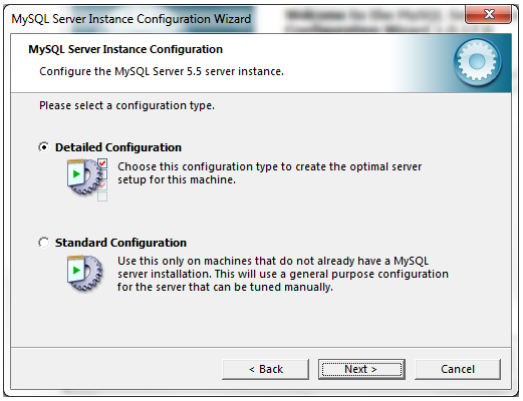
Una vez que se realizó la descarga, aparece la ventana del asistente de configuración:

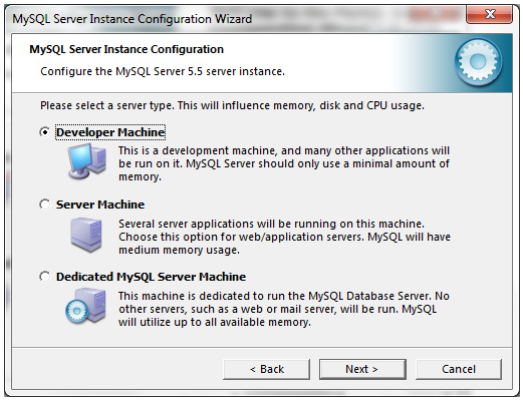


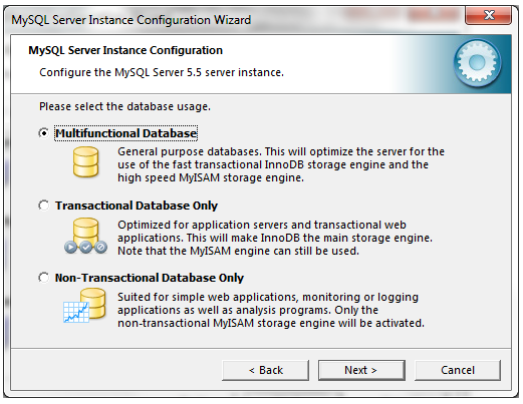
Con las selecciones que el asistente de configuración muestra por defecto es suficiente para proceder con la instalación, como se muestra en la siguiente secuencia de pantallas:

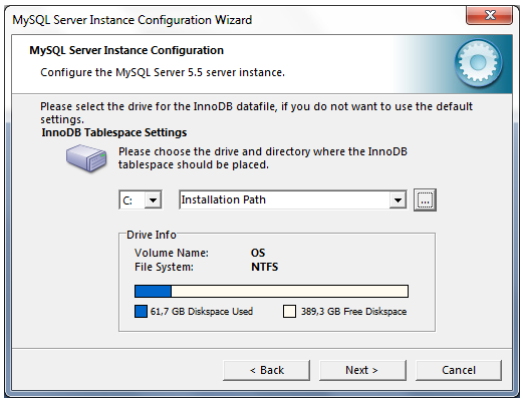

MySQL Server 5.5 Setup
Choose Setup Type
Choose the setup type that best suits your needs
Typical
Installs the Typical Installation features. Recommended for most users.
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.
Complete
All program features will be installed. Requires the most disk space.
Back
Next
Cancel

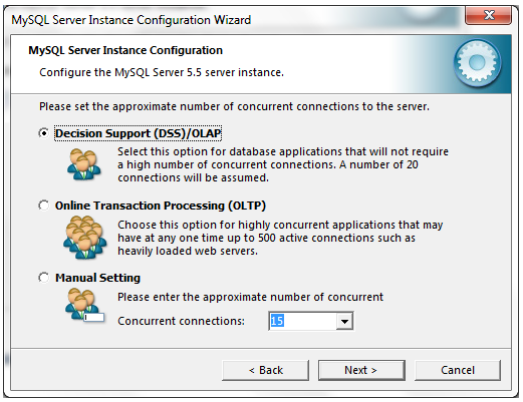

MySQL Server Instance Configuration Wizard
Welcome to the MySQL Server Instance Configuration Wizard 1.0.17.0
The Configuration Wizard will allow you to configure the MySQL Server 5.5 server instance. To Continue, click Next.
MySQL
Next >
Cancel

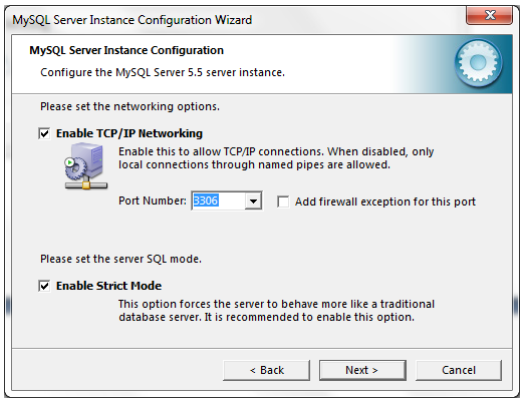

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.5 server instance.
Please select a configuration type.
Detailed Configuration
Choose this configuration type to create the optimal server setup for this machine.
Standard Configuration
Use this only on machines that do not already have a MySQL server installation. This will use a general purpose configuration for the server that can be tuned manually.
< Back
Next >
Cancel

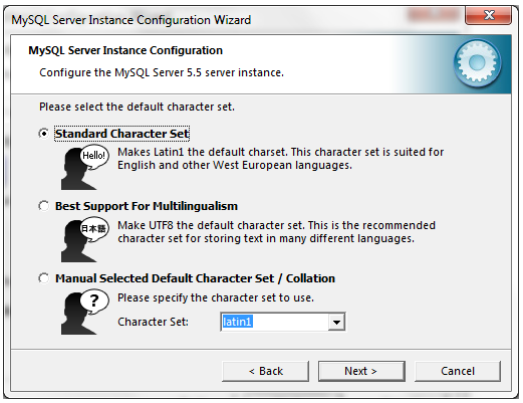

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.5 server instance.
Please select a server type. This will influence memory, disk and CPU usage.
Developer Machine
This is a development machine, and many other applications will be run on it. MySQL Server should only use a minimal amount of memory.
Server Machine
Several server applications will be running on this machine. Choose this option for web/application servers. MySQL will have medium memory usage.
Dedicated MySQL Server Machine
This machine is dedicated to run the MySQL Database Server. No other servers, such as a web or mail server, will be run. MySQL will utilize up to all available memory.
< Back
Next >
Cancel

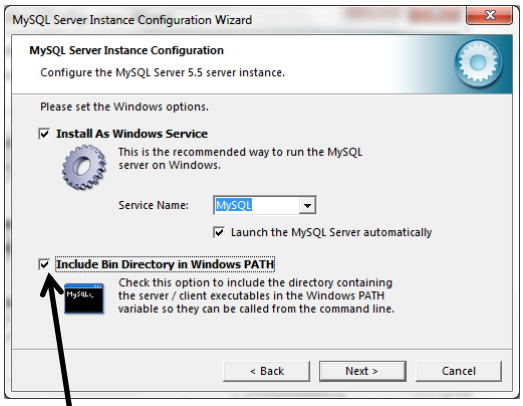

MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.5 server instance.
Please select the database usage.
Multifunctional Database
General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.
Transactional Database Only
Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.
Non-Transactional Database Only
Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.
< Back
Next >
Cancel


MySQL Server Instance Configuration Wizard
MySQL Server Instance Configuration
Configure the MySQL Server 5.5 server instance.
Please select the drive for the InnoDB datafile, if you do not want to use the default settings.
InnoDB Tablespace Settings
Please choose the drive and directory where the InnoDB tablespace should be placed.
C:
Installation Path
Drive Info
Volume Name: OS
File System: NTFS
61,7 GB Diskspace Used
389,3 GB Free Diskspace
< Back
Next >
Cancel

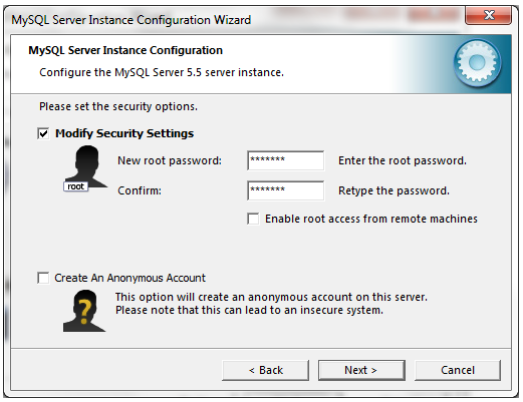




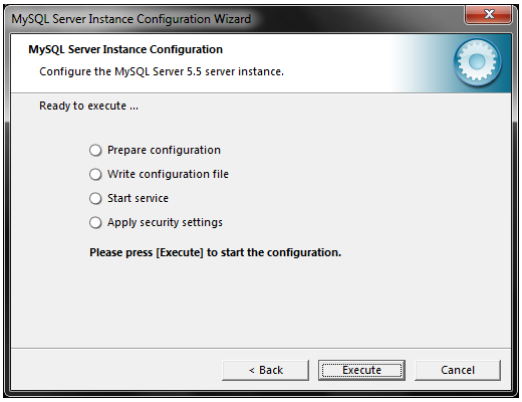




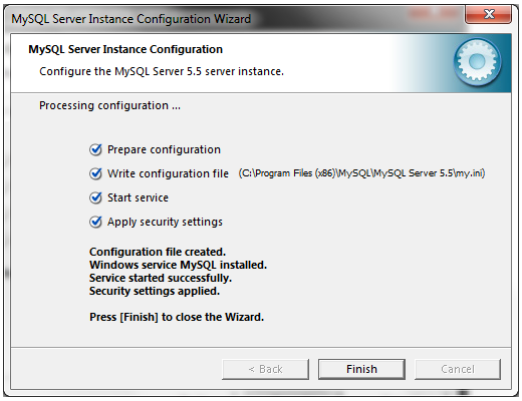
Nótese que se eligió adicionalmente la segunda opción!



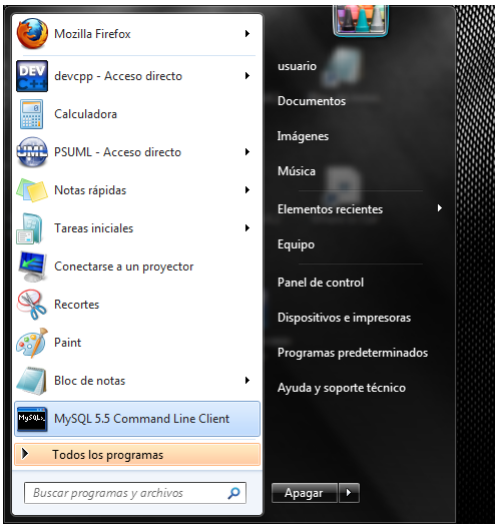
Es muy importante recordar siempre este password, ya que servirá para iniciar la sesión de trabajo en MySQL.



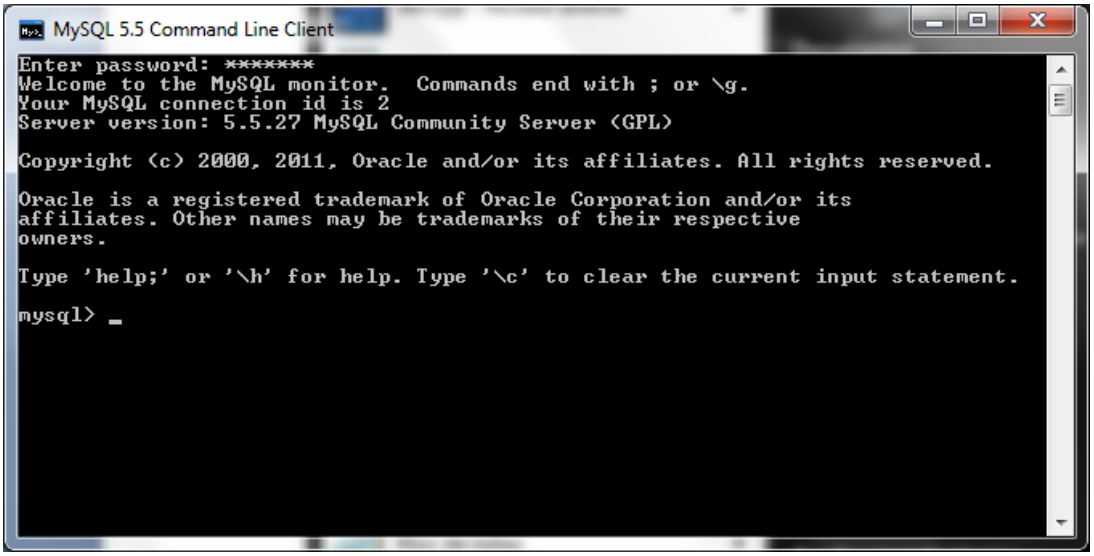
Esta última ventana mostrará el progreso de la configuración, cuando termina la instalación todas los pasos quedan señalados como se muestra a continuación.



Para tener acceso a MySQL es necesario seleccionarlo de entre los programas del menú de inicio.



Para iniciar la sesión es necesario introducir el password que se proporcionó durante la instalación. En caso de no recordar el password es necesario desinstalar todo y volver a empezar.



Glosario.

(Extracto del glosario de [Kroenke, 2003], con algunas modificaciones y adiciones)

Abstracción.- Generalización de algo que oculta ciertos detalles sin importancia, pero que permite trabajar con una clase más amplia de tipos.

Aplicación. Sistema computacional que procesa parte de una base de datos para satisfacer las necesidades de información del usuario. Contiene menús, formas, reportes, consultas, páginas Web y programas de aplicación.

Atributo.- (1) En el contexto relacional es la columna de una relación, también se le llama columna, campo o elemento de datos. (2) en el modelo entidad-relación es una propiedad de una entidad.

Base de datos.- Es una colección estructurada de datos.

Base de datos distribuida.- Es una base de datos almacenada en dos o más computadoras.

Base de datos relacional.- Base de datos que consta de relaciones, también llamadas tablas.

Campo.- (1) en el contexto relacional es sinónimo de atributo. (2) un grupo lógico de bytes en un registro utilizado para el procesamiento de archivos.

Cardinalidad.- En una relación binaria, el número máximo o mínimo de elementos permitidos en cada lado de la relación.

Cardinalidad máxima.- (1) El número máximo de valores que puede tener un atributo dentro de un objeto. (2) en una relación entre tablas, el número de máximo de filas con las que puede relacionarse un renglón de una tabla en la otra tabla.

Cardinalidad mínima.- (1) El número mínimo de valores que puede tener un atributo dentro de un objeto. (2) en una relación entre tablas, el número de mínimo de filas con las que puede relacionarse un renglón de una tabla en la otra tabla.

DDL.- Véase Lenguaje de Definición de Datos (LDD).

DML.- Véase Lenguaje de Manejo de Datos (LMD).

Diagrama de entidad-relación (DER).- Gráfica utilizada para representar las entidades y sus relaciones. Por lo general las entidades se muestran en cuadrados o rectángulos, y las relaciones, en diamantes o rombos. La cardinalidad de la relación se muestra dentro del diamante.
Diccionario de Datos.- Es un catálogo de bases de datos y de metadatos de aplicación al que un usuario puede ingresar. Un diccionario de datos activo es aquel cuyos contenidos los actualiza automáticamente el SGBD cada vez que se realiza algún cambio en la estructura de base de datos a la aplicación. Un diccionario de datos pasivo es aquél cuyos contenidos deben actualizarse manualmente cuando se realizan los cambios.

Dominio.- (1) en el contexto del modelo entidad-relación, es el conjunto de todos los valores posibles que puede tener un atributo. (2) En el contexto relacional es la descripción del formato (tipo de datos, longitud) y la semántica (significado) de un atributo.

Entidad.- (1) Algo de importancia para el usuario, lo cual se necesita representar en una base de datos. (2) en un modelo entidad-relación, las entidades se restringen a cosas que pueden representarse mediante una tabla.

Entidad débil.- En un modelo entidad-relación, una entidad cuya existencia lógica en la base de datos depende de la existencia de otra entidad. Véase también Entidad dependiente de ID y Entidad Fuerte.

Entidad dependiente de la existencia.- Sinónimo de entidad débil. Una entidad que no puede aparecer en la base de datos, a menos que también aparezca una ocurrencia de una u otras entidades en la base de datos. Una subclase de las entidades dependientes de la existencia son las entidades dependientes de ID.

Entidad dependiente de ID.- Entidad que no puede existir lógicamente sin la existencia de otra entidad. Por ejemplo, una CITA no puede existir sin un CLIENTE que concerte la cita. La entidad dependiente de ID siempre contiene la clave de la entidad de la cual depende. Dichas entidades son un subconjunto de la entidad débil.

Entidad Fuerte.- En un modelo entidad-relación, cualquier entidad cuya existencia en la base de datos no depende de la existencia de alguna otra entidad.

Esquema.- Vista lógica completa de la base de datos.

Esquema relacional.- Conjunto de relaciones con restricciones interrelacionales.

Grado.- Para las relaciones en el modelo entidad-relación, el número de entidades que participan en la relación. En casi todos los casos, dichas relaciones son de grado 2.

Índice.- Datos significativos utilizados para mejorar y clasificar la ejecución. Los índices pueden construirse para una columna o grupos de columnas. Son especialmente útiles para las columnas que se utilizan para saltos de control en reportes y para especificar las condiciones en los "join".

Join (reunión).- Operación algebraica relacional en dos relaciones, A y B, que produce una tercera relación, C. Un renglón de A se conecta con uno de B para formar un nuevo renglón en C, si es que las filas en A y B satisfacen las relaciones concernientes a sus valores. Por ejemplo, A1 es un atributo en A y B1 es un atributo en B. La junta de A con B en la cual A1<B1 resultará en una relación, C, que tiene la concatenación de las filas en A y B, en las cuáles el valor de A1 es menor que el valor de B1.

Lenguaje de Definición de Datos (LDD).- Lenguaje utilizado para describir la estructura de una base de datos.

Lenguaje de Manipulación de Datos (LMD).- Un lenguaje utilizado para describir el procesamiento de una base de datos.

Llave.- Un grupo de uno o más atributos que identifican un renglón único en una relación. Debido a que las relaciones no pueden tener filas duplicadas, cada relación debe tener cuando menos una llave, que es la combinación de todos los atributos de la relación

Metadatos.- Datos referentes a la estructura de datos en una base almacenada en el diccionario de datos. Los metadatos se utilizan para describir tablas, columnas, restricciones, índices, etc.

Metadatos de aplicación.- Diccionario de datos; datos referentes a la estructura y contenido de los menús de aplicación, formas y reportes.

Modelo de datos.- (1) modelos de los requerimientos de datos de los usuarios, expresado en términos del modelo entidad-relación, o del modelo objeto semántico. Algunas veces se denomina modelo de datos de los usuarios. (2) lenguaje para la descripción de la estructura y procesamiento de una base de datos.

Modelo de datos relacional.- Modelo de datos en el cual se almacenan los datos en relaciones y éstas se representan en los renglones de la relación mediante valores de datos.

Modelo entidad-relación.- Las construcciones y convenciones que se utilizan para crear un modelo de datos del usuario. Las cosas en el mundo del usuario están representadas por entidades, y las asociaciones entre éstas están representadas mediante relaciones. Usualmente, los resultados se documentan en un diagrama entidad-relación.

Normalización.- Proceso por el que se evalúa una relación para determinar si está o no en una forma normal especificada y, si es necesario, convertirla en relaciones que estén en dicha forma normal especificada.

Partición horizontal.- Subconjunto de una tabla que consta de filas completas de la tabla, por ejemplo, en una tabla con 10 filas, las primeras cinco filas.

Partición mezclada.- Una combinación de una partición vertical y horizontal; por ejemplo, en una tabla con cinco columnas y cinco filas, las primeras tres columnas de las primeras tres filas.

Partición vertical.- Un subconjunto la las columnas de una tabla. Por ejemplo, en una tabla con 10 columnas, las primeras cinco columnas.

Procesamiento concurrente.- En las aplicaciones de teleprocesamiento, es el hecho de que varias transacciones compartan el CPU. A cada transacción se le asigna el CPU de manera circular o en alguna otra manera durante cierto período de tiempo. Las operaciones se llevan a cabo tan rápidamente que a los usuarios les parecen simultáneas. En las redes de área local y otras aplicaciones distribuidas, se utiliza el procesamiento concurrente para referirse al procesamiento de aplicaciones (posiblemente simultáneo) en computadoras múltiples.

Procesamiento de base de datos distribuida.- Procesamiento de base de datos en el cual se recuperan y actualizan los datos de las transacciones a través de dos o as computadoras independientes, y por lo general geográficamente distribuidas.

Producto.- Una operación relacional en dos relaciones, A y B, que produce una tercera C, con C que contiene la concatenación de cada renglón en con cada renglón de B.

Producto cartesiano.- Operación relacional en dos relaciones, A y B, que produce una tercera relación, C; con C que contiene la concatenación de cada renglón en A con cada renglón en B.

Programa de aplicación.- Un programa desarrollado de acuerdo con las necesidades del cliente para el procesamiento de una base de datos. Puede escribirse en un lenguaje de procedimientos estándar, como C++, C, Pascal o Visual Basic, o en un lenguaje único para el SGBD.

Registro.- (1) Un grupo de campos pertenecientes a la misma entidad; se utiliza en los sistemas de procesamiento de archivos. (2) En un modelo relacional, un sinónimo de renglón y tupla.

Relación.- Un arreglo bidimensional que contiene información en la que las columnas tienen el mismo significado en cada renglón y los renglones no están duplicados.

Relación 1:N.- Abreviatura de una relación uno a muchos entre los renglones de una tabla.

Relación binaria.- Una asociación entre dos entidades, objetos o renglones relacionales.

Relación ES UN.- Una relación entre dos entidades del mismo tipo lógico.

Relación recursiva.- Relaciones entre entidades o tuplas del mismo tipo. Por ejemplo, si CLIENTES se refiere a otros CLIENTES, la relación a la que *hace referencia* es recursiva.

Relación TIENE UN.- Una relación entre dos entidades que son de diferentes tipos lógicos, por ejemplo EMPLEADO TIENE UN (n) AUTO.

Renglón.- Un grupo de columnas de una tabla que pertenecen a una misma entidad. Es lo mismo que una tupla o un registro.

Reporte.- Extracción de datos de una base de datos. Los reportes se pueden imprimir, desplegar en un monitor de computadora o almacenar en un archivo. Un reporte es parte de una aplicación de base de datos.

Repositorio.- Colección de metadatos sobre la estructura de base de datos, aplicaciones, páginas Web, usuarios y otros componentes de aplicación. Los repositorios activos se mantienen automáticamente mediante las herramientas en el ambiente de desarrollo de la aplicación. Los repositorios pasivos se deben mantener manualmente.

Respaldo consistente.- Archivo de respaldo desde el cual se ha eliminado todos los cambios que no se realizaron.

Respaldo diferencial.- Archivo de respaldo que sólo contiene los cambios realizados después de un respaldo anterior.

Restricción.- Regla que concierne a los valores de atributos permitidos cuya veracidad se puede evaluar. Por lo general, una restricción no incluye reglas dinámicas, como “el pago al vendedor nunca puede disminuir”, o “el salario de ahora debe ser mayor al del trimestre pasado”.

Restricción de cardinalidad de una relación.- Una restricción en el número de renglones que puede haber en una relación. Las restricciones de cardinalidad mínima determinan el número de renglones que debe haber; las restricciones de cardinalidad máxima especifican el mayor número de renglones que puede haber.

Restricción de integridad referencial.- Una restricción en los valores de la clave ajena. Una restricción de integridad referencial especifica que los valores de una llave externa deben ser un subconjunto apropiado de los valores de la llave primaria a los que hace referencia.

Restricción de interrelación.- Restricción que requiere el valor de un atributo en un renglón de una relación para asociar el valor de un atributo encontrado en otra relación. Por ejemplo, Número de cliente en PEDIDO debe ser igual a Número de Cliente en CLIENTE.

Restricción de la relación.- Restricción de integridad referencial o restricción de cardinalidad de la relación.

Selección.- Operación algebraica relacional ejecutada en una relación, A, que produce una relación, B, con B que contiene solamente las filas en A que satisfacen las restricciones especificadas en la selección.

SGBD.- Sistema de Gestión de Bases de Datos, en ingles Data Base Management System (SGBD). Es un sistema de administración de bases de datos, consiste en un conjunto de programas utilizados para definir, administrar y procesar la base de datos y sus aplicaciones.

Sistema de administración de base de datos distribuida.- En una base de datos distribuida, es la colección de la transacción distribuida y los administradores de base de datos en todas las computadoras.

Sistema de base de datos distribuida.- Sistema distribuido en el cual una base de datos, o porciones de ésta, se distribuyen en dos o más computadoras.

Sistema distribuido.- Sistema en el que los programas de aplicación de una base de datos se procesan en dos o más computadoras.

SQL.- Lenguaje de consulta estructurado, un lenguaje para la definición de la estructura y el procesamiento de una base de datos relacional. Se utiliza como un lenguaje de consulta autónomo, o puede implementarse en programas de aplicación. El SQL es aceptado como un estándar por el American National Standards Institute (Instituto Nacional Americano de Estándares). Fue desarrollado por IBM.

Sublenguaje de datos.- Lenguaje para la definición y procesamiento de una base de datos diseñada para implementarse en programas escritos en otro lenguaje, en la mayoría de los

casos COBOL, C++, o Visual Basic. Un sublenguaje de datos es un lenguaje de programación incompleto, puesto que solo contiene construcciones para el acceso de datos.

Subsistema de definición de herramientas.- Parte del programa SGBD utilizado para definir y cambiar la estructura de la base de datos.

Subsistema de diccionario de datos y de administración de bases de datos.- Colección de programas en el SGBD utilizados para acceder el diccionario de datos y ejecutar las funciones de la administración de las bases tales como el mantenimiento de contraseñas y la ejecución del respaldo y recuperación.

Subsistema de interfaz de procesamiento.- Una porción de las rutinas SGBD que ejecuta instrucciones para el procesamiento de la base de datos. Acepta el ingreso de los programas de consulta interactivos y de programas de aplicación escritos en lenguajes estándar, o en lenguajes específicos del SGBD.

Subtipo.- En jerarquías de generalización, una entidad que es una subespecie o subcategoría de un tipo de nivel superior. Por ejemplo, INGENIERO es un subtipo de EMPLEADO.

Supertipo.- En jerarquías de generalización, una entidad que lógicamente contiene subtipos. Por ejemplo, EMPLEADO es un supertipo de INGENIERO, CONTADOR y VENDENDOR.

Tabla base.- En las implementaciones relacionales, la tabla desde donde se definen las vistas relacionales.

Transacción.- El registro de un suceso, ya sea en una empresa, en una escuela, en un centro de investigación, etc.

Tupla.- Lo mismo que renglón.

Unión.- Una operación algebraica relacional realizada en dos relaciones compatibles con la unión, por ejemplo A y B, que forma una tercera relación, C, con C que contiene cada renglón de en A y B, menos algún renglón duplicado.

Valor calculado.- Una columna de una tabla que se calcula a partir de los valores de otras columnas. Los valores no se almacenan, pero se calculan cuando se despliegan.

Valor nulo.- Valor de un atributo que nunca se ha proporcionado. Dichos valores son ambiguos y pueden significar que: (a) el valor es desconocido, (b) el valor no es apropiado, o (c) el valor está en blanco.

Vista.- Lista estructurada de elementos de datos de entidades definidos en el modelo de datos.

Vista de usuario.- Vista de una base de datos de un usuario en particular.

BIBLIOGRAFÍA

[1] Access 7.0 para Windows, Editorial Anaya, Madrid, 2001.
[2] Cattell, "What are next-generation database systems?" CACM, octubre, Vol. 34, N2 10, pp.31-33, 1991.
[3] Celma M.,Casamayor J., Mota L. "Bases de Datos" Relacionales Pearson Education, España, 2003.
[4] De Miguel A., Piattini M., "Fundamentos y Modelos de Bases de Datos". 2ª ed., Alfaomega-Ra-ma, México, 2004.
[5] De Miguel A., Martínez P., Castro E., Cavero J., Cuadra D., Iglesias A., Nieto C., "Diseño de Bases de Datos". Problemas resueltos, Alfaomega-Ra-ma, México, 2001.
[6] De Miguel A., Piattini M., Marcos E., "Diseño de Bases de Datos Relacionales". Alfaomega-Ra-ma, Colombia 2000.
[7] Deitel P.J. & Deitel H. M., "Java, cómo programar", 7ª ed., Pearson-Prentice Hall, México, 2008.
[8] Elmasri R., Navathe S.B., "Fundamentos de Sistemas de Bases de Datos". 5a ed. Pearson-Addison Wesley, España, 2008.
[9] García-Molina H., Ullman J. & Widom J., "Database systems, the complete book", 2nd ed., Pearson-Prentice Hall, U.S.A., 2009.
[10] Hansen G.W., Hansen J.V., "Diseño y Administración de Bases de Datos", 2ª ed., Prentice Hall, España, 1998.
[11] Jimenez C., Armstrong T, "El rol del lenguaje SQL en los SGBDR y en la implementación del Modelo Relacional" http://www.inf.udec.cl/~revista/ediciones/edicion1/armstrong.PDF
[12] Kroenke David, "Procesamiento de Bases de Datos", 8ª ed. Pearson Education, México, 2003.
[13] Marteens Ian, "La cara oculta de Borland C++ Builder con SQL, España, 1999.
[14] López-Fuensalida A., "Metodologías de Desarrollo". Producción automática de software con herramientas CASE, Macrobit-Ra-ma, México,1991.
[15] Lucas A., "Diseño y Gestión de Sistemas de Bases de Datos", Ed. Parainfo, Madrid 1993.
[16] Pastor O. & Blesa P., "Gestión de Bases de Datos", Universidad Politécnica de Valencia, España, 2000.
[17] Piattini M., Calvo-Manzano J., Cervera J., Fernandez L. "Análisis y diseño de Aplicaciones Informáticas de Gestión". Una perspectiva de Ingeniería de Software. Alfaomega-Rama, México, 2004.

[18] Piatinni M., "Líneas de evolución de las bases de datos", NOVATICA, Especial 25 aniversario, edición digital, ATI, may-jun, 2000.
[19] Piattini M., Castaño, De Miguel A., "Fundamentos y Modelación de Bases de Datos". Computec Ra-ma, Colombia, 1998.
[20] Post G., "Sistemas de administración de bases de datos", McGraw-Hill, 3era ed., 2006.
[21] Rob P., Coronel C., "Sistemas de Bases de Datos". Diseño, implementación y administración, ed. Thomson, México, 2004.
[22] Sampalo de la Torre M.A, Leyva E., Garzón M.L, Prieto J., "Informática" Volumen III ed. MAD.
[23] Silberschatz, Korth, Sudarshan, "Fundamentos de Bases de Datos", McGraw-Hill/Interamericana, 5ª ed., China, 2006.
[24] http://dev.mysql.com/doc/refman/5.0/es/index.html

Notas del curso: Bases de Datos
Se terminó de imprimir el 28 de febrero de 2013 en
Publidisa Mexicana S. A. de C.V.
Calz. Chabacano No. 69, Planta Alta
Col. Asturias C.P. 06850.
50 ejemplares en papel bond 90 gr.