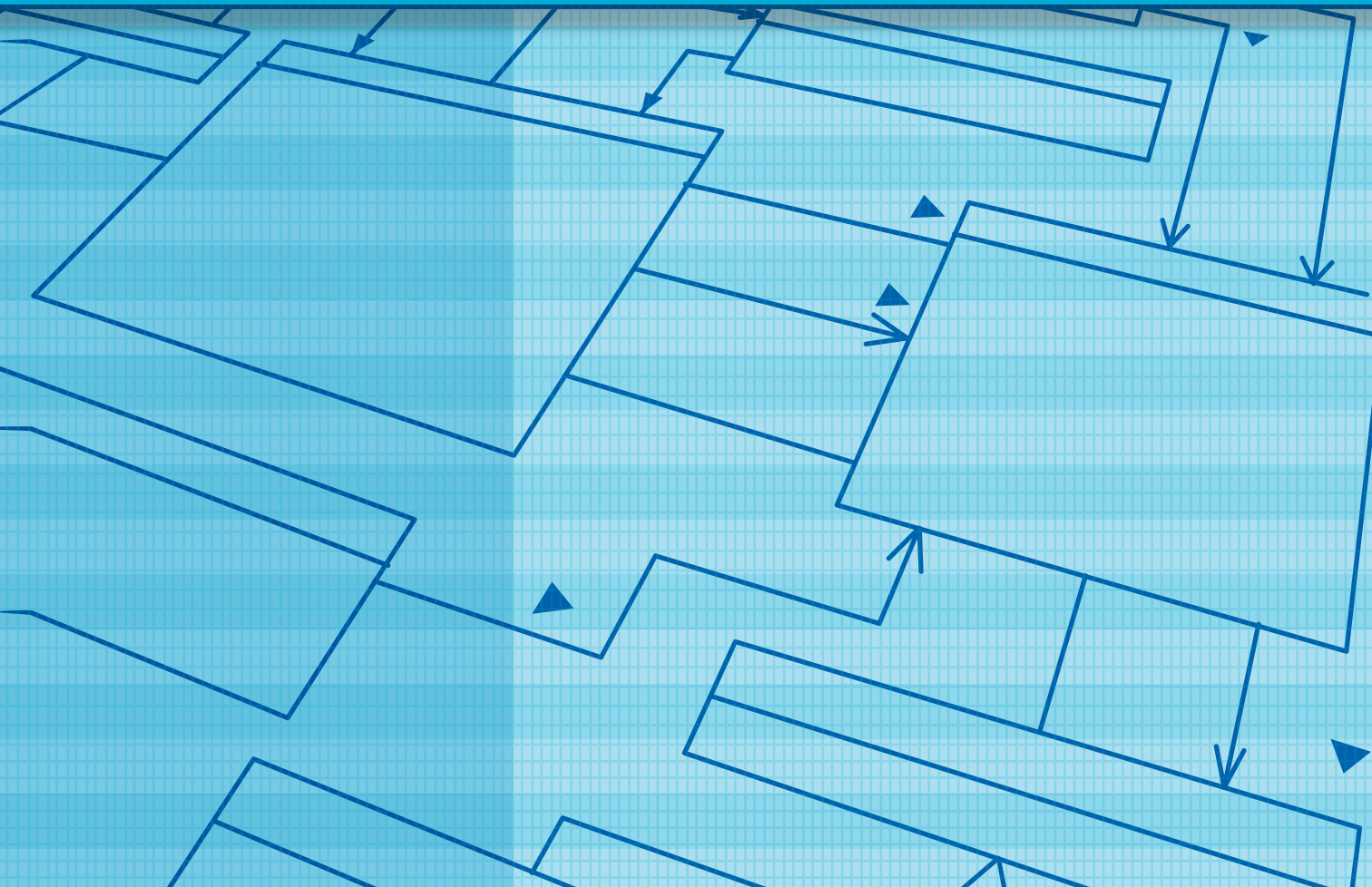
DISEÑO DE BASE DE DATOS

ESCUELA DE INGENIERÍA DE SISTEMAS

Universidad del Azuay

Oswaldo Merchán Manzano



Diseño de bases de datos

Oswaldo Merchán Manzano

Autor

PhD. Francisco Salgado Arteaga

Rector de la Universidad del Azuay

PhD. Martha Cobos Cali

Vicerrectora de la Universidad del Azuay

Mgt. Jacinto Guillén García

Vicerrector de Investigaciones

1ra Edición: Universidad del Azuay
Av. 24 de Mayo 7-77 y Hernán Malo
Apartado 01.01.981
PBX: (593) 4091000
Fax: (593) 7815-997
www.uazuay.edu.ec
Cuenca – Ecuador

Carrera de Ingeniería de Sistemas y Telemática
Escuela de Ingeniería de Sistemas
Universidad del Azuay
Apartado 01.01.981

Diseño y diagramación: Mariela Barzallo León

ISBN: 978-9978-325-94-0

9 789978 325940

Impreso en Cuenca – Ecuador, Noviembre 2016

PRESENTACIÓN

El diseño y la gestión de bases de datos son aspectos fundamentales para el éxito de cualquier sistema de información. El adecuado diseño que posibilite una gestión ágil y segura es clave para garantizar una información confiable y oportuna, que apoye la toma de decisiones.

El conocimiento y la experiencia profesional junto con la experiencia docente de Oswaldo Merchán Manzano se reflejan en el presente libro, el que aborda, con una visón fundamentalmente práctica, el diseño de bases de datos relacionales, sin descuidar ni restarles importancia a los temas de orden conceptual y sobre todo destacando las ventajas que las bases de datos relacionales tienen sobre otros sistemas.

Oswaldo manifiesta que el libro está destinado principalmente a los estudiantes que se inician en el campo de las tecnologías de la información, y creo que la forma didáctica en que se abordan los temas a lo largo de los seis capítulos, así como el desarrollo de un caso práctico cumplen con el propósito planteado. Sin embargo, los temas abordados y la forma de hacerlo, no sólo lo convierten en un texto recomendable para la docencia, sino que también se convierte en un punto de partida para el desarrollo de proyectos de investigación.

Las horas de trabajo adicionales que fueron asignadas por la Universidad del Azuay al Ing. Merchán, han sido eficiente y cabalmente cumplidas y se materializaron en este libro, que será un referente en la formación de futuros profesionales.

Carlos Cordero Díaz

DEDICATORIA

A mi esposa Julia,

a mis hijos Ana Belén y Esteban.

PRÓLOGO

Este libro trata sistemáticamente el diseño de las bases de datos relacionales, analizando los modelos de datos, el lenguaje y la teoría conceptual subyacente. La obra se centra en el diseño de bases de datos desde un punto de vista eminentemente práctico; emplea principios metodológicos que ayudan al análisis, diseño y elaboración del esquema conceptual y lógico. Se proporcionan ilustraciones y una serie de ejemplos reales, que clarifican los conceptos del diseño, creación y gestión de una base de datos. Adicionalmente al final de cada capítulo se plantean ejercicios que le sirven al lector para poner en práctica los conceptos teóricos; estos ejercicios se basan en el caso de estudio denominado **La Ferretería**, que será analizado a lo largo del libro.

Los conceptos fundamentales, algoritmos y ejercicios prácticos que se presenta en este libro no están ligados a un sistema de gestión de bases de datos comercial específico.

Este libro está dirigido a los estudiantes universitarios que cursan carreras relacionadas con las tecnologías de la información, los sistemas de información o con la informática. El libro está pensado como texto para un curso de un cuatrimestre sobre el diseño de bases de datos. El libro también se dirige a los profesionales como analistas de sistemas, programadores de aplicaciones y personas en general que están relacionadas con el uso de bases de datos o interesadas en aprender esta tecnología.

CONTENIDO

El texto está dividido en seis capítulos. El Capítulo 1 presenta una introducción a los conceptos de los sistemas de bases de datos; se analizan las ventajas con respecto a los sistemas basados en archivos, lenguajes de base de datos, usuarios tipo y los componentes de un SGBD. El Capítulo 2 ilustra los conceptos sobre el modelo entidad - relación que es utilizado para el diseño conceptual de una base de datos relacional; se presenta la notación para los diagramas ER, los algoritmos y procedimientos para crear un esquema relacional. En el Capítulo 3 se describen los fundamentos del modelo de datos relacional, las propiedades de una relación que sirven para el diagrama del esquema de una base de datos. El Capítulo 4 introduce los conceptos del álgebra relacional con una serie de ejemplos prácticos que ilustran las operaciones necesarias para el procesamiento de consultas. El Capítulo 5 es de tipo práctico, aquí se trata el lenguaje de consultas estructurado SQL; se analizan las diferentes instrucciones de definición y manipulación de datos. Además se revisan los conceptos de integridad de datos. Finalmente, el Capítulo 6 presenta las técnicas de normalización, que incluyen los conceptos de dependencia funcional, transitiva, y las formas normales. Con el propósito de ilustrar la técnica de normalización, este capítulo concluye con un ejercicio práctico que describe paso a paso el proceso de normalización.

Un curso de diseño de bases de datos se puede impartir de diferentes formas. Como directrices para el uso del libro se recomienda utilizar en el orden en que aparecen los capítulos; sin embargo, es posible modificarlo impartiendo los conceptos de la normalización del Capítulo 6 luego de revisar el diseño de la bases de datos a través del modelo Entidad – Relación y modelo relacional de los Capítulos 2 y 3 respectivamente, para continuar estudiando en el Capítulo 4 el álgebra relacional, que describe la manipulación de datos y que es la base del lenguaje de consultas SQL del Capítulo 5.

En el libro no se enseña un sistema de gestión de base de datos (SGBD) específico, por dos razones: la primera, existe una gama de SGBD comerciales y para cada uno están disponibles muchos recursos para enseñar; y la segunda, es importante que los estudiantes se den cuenta de la independencia que existe entre el diseño de las bases de datos y el software para implementarlas.

AGRADECIMIENTO

Es grato reconocer el apoyo de muchas personas en este proyecto. Empezaré agradeciendo al economista Carlos Cordero, Rector de la Universidad del Azuay, al ingeniero Jacinto Guillén, Decano de Investigación y a las autoridades de la Facultad de Ciencias de la Administración y de la Escuela de Ingeniería de Sistemas, ingenieros Xavier Ortega y Marcos Orellana.

Este libro se ha beneficiado de la contribución, el aporte y sugerencias de los revisores: Dr. Francisco Salgado Arteaga, *Universidad del Azuay* y Dr. Lisandro Solano Quínde, *Subdecano de la Facultad de Ingeniería de la Universidad de Cuenca,* a quienes les expreso mi agradecimiento.

Gracias al Dr. Oswaldo Encalada, *Universidad del Azuay,* por su valiosa aportación de experiencia y voluntad para la revisión de estilo.

También expreso mi gratitud a los estudiantes Renato Ávila, Christina Cabrera y Andrea Trujillo que colaboraron con la parte práctica y los ejercicios resueltos.

Por último, mi agradecimiento a quienes comparten mi vida, mi familia, que me proporcionó el tiempo y el apoyo para preparar este libro.

Oswaldo Merchán

Julio de 2016

TABLA DE CONTENIDOS

CAPÍTULO 3
Modelo relacional 103

CAPÍTULO 4
Álgebra relacional 123

CAPÍTULO 5
SQL 175

FUNDAMENTOS DE LOS SISTEMAS DE BASES DE DATOS

En este capítulo se presentan los conceptos fundamentales de la tecnología de base de datos, y se definen los términos básicos, características y componentes. En la Sección 1.1 se realiza una revisión general de los sistemas de bases de datos y se presentan algunos ejemplos de aplicaciones. La Sección 1.2 compara la antigua técnica basada en el procesamiento de archivos con los sistemas de procesamiento de bases de datos. En la Sección 1.3 y 1.4 se examinan los objetivos de un sistema de base de datos y se presentan para el análisis una serie de definiciones de bases de datos. En la Sección 1.5 se examinan los tres niveles de abstracción de datos, la independencia de datos y sus beneficios asociados, y los conceptos de esquema e instancia. En la Sección 1.6 se introducen los conceptos de los modelos de datos que se aplicarán en capítulos posteriores. En la Sección 1.7 se describen los tipos de usuarios que interactúan con los sistemas de bases de datos. En la Sección 1.8 se consideran los diferentes lenguajes utilizados en un sistema de gestión de bases de datos. En la Sección 1.9 se analiza los SGBD y su relación con la base de datos y las aplicaciones. Por último en la Sección 1.10 se examinan los componentes de un sistema de base de datos típico.

1.1 INTRODUCCIÓN

Las bases de datos han sido un tema importante en el estudio de los sistemas de información, permanentemente en nuestra vida diaria estamos relacionándonos con bases de datos. En la actualidad el rápido avance de las tecnologías de internet y las telecomunicaciones, han hecho del conocimiento de la tecnología de bases de datos un área de estudio fundamental en el campo de las tecnologías de información. Con la revolución tecnológica de internet aumentó el acceso directo de los usuarios a las bases de datos, permitiendo a muchas organizaciones mediante interfaces web dejar en línea sus servicios y el acceso a los datos. Por ejemplo, en una biblioteca en línea, al buscar un determinado libro o artículo científico se está accediendo a una base de datos, o cuando los estudiantes de la universidad a través de su teléfono celular acceden al sitio web a revisar sus calificaciones, la información es recuperada mediante un sistema de base de datos.

Como parte introductoria de este capítulo y con el objeto de familiarizarnos con la terminología, se consideran definiciones generales de los componentes de un sistema de bases de datos. Una *base de datos* es una colección de datos interrelacionados y un *sistema de gestión de bases de datos* SGBD (DBMS por sus siglas en inglés *DataBase Management System*) es el software que gestiona y controla el acceso a los datos de la base de datos. *Una aplicación de base de datos* es un programa que interactúa con la base de datos. En la Figura 1.1 se muestra la estructura general de un sistema

de bases de datos, en donde el usuario interactúa con una aplicación que hace de interfaz con el SGBD, para acceder a los datos. En los siguientes capítulos del libro se proporcionan definiciones más precisas de estos conceptos.

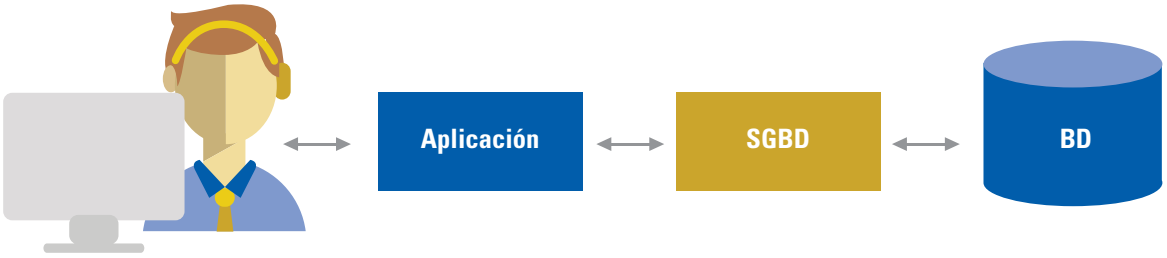


Figura 1.1. Relación usuario, aplicación de bases de datos, SGBD y base de datos.

A continuación de describen algunos ejemplos de aplicaciones de bases de datos:

Universidades. Para la gestión de las matrículas de los estudiantes, sus materias, calificaciones, asistencia, profesores, registro de graduados. Adicionalmente es posible incluir información para la gestión de recursos humanos y contables.

Producción. Para la gestión de la cadena de producción, los suministros, el manejo del inventario de bodega y pedidos.

Biblioteca. Para el registro de los libros disponibles en la biblioteca, la información sobre los lectores y la gestión de reservas. La información generada en la base de datos facilitará a los lectores la búsqueda de un libro a partir de su código, título, autor o tema. El sistema controlará la entrada y salida de los libros en la biblioteca y enviará recordatorios a los lectores que tienen un libro prestado.

Campeonato de fútbol. En el desarrollo de un campeonato de fútbol, posiblemente la información más relevante que se tiene que reportar, es la actualización periódica de la tabla de posiciones, para lo cual, con la ayuda de la base de datos, es posible registrar información de cada uno de los partidos, el marcador, los jugadores que anotaron los goles, etcétera. Para la gestión de la información del campeonato también se pueden registrar los datos relacionados

con los árbitros que dirigieron el partido, el estadio en donde se jugó, e incluso las amonestaciones a los jugadores.

Supermercado. Para comprar en un supermercado, lo más probable es que se acceda a una base de datos. Por lo general, el cajero ingresa los productos adquiridos a través de la lectura del código de barras. Este programa se enlaza con otra aplicación, que mediante el código de barras accede a la base de datos para obtener la descripción y precio del producto, que junto con la cantidad vendida, genera la factura correspondiente.

La base de batos es un depósito único de datos para toda la organización, diseñada para gestionar grandes cantidades de información, por lo que debe ser capaz de integrar los distintos sistemas y aplicaciones, atendiendo los requerimientos de los usuarios en los niveles: operativo, táctico y estratégico.

Según Kroenke (2003), una base de batos es un conjunto de registros integrados que jerárquicamente está caracterizada de la siguiente manera: los bits conforman los bytes o caracteres; los caracteres constituyen campos; los campos integran registros y los registros forman los denominados **archivos de datos del usuario**. Adicionalmente una base de datos contiene una descripción de su propia estructura, denominada **diccionario de datos o metadatos**, incluye también **índices** que se usan para representar las relaciones entre los datos y para mejorar el desempeño de las aplicaciones de la base de datos. El último tipo de información que con frecuencia se almacena en la base de datos, son los llamados **metadatos de aplicación**, que contienen datos acerca de la aplicación, como la estructura de la forma de entrada de los datos o de un reporte. En la Figura 1.2 se ilustra esta caracterización de una base de datos.

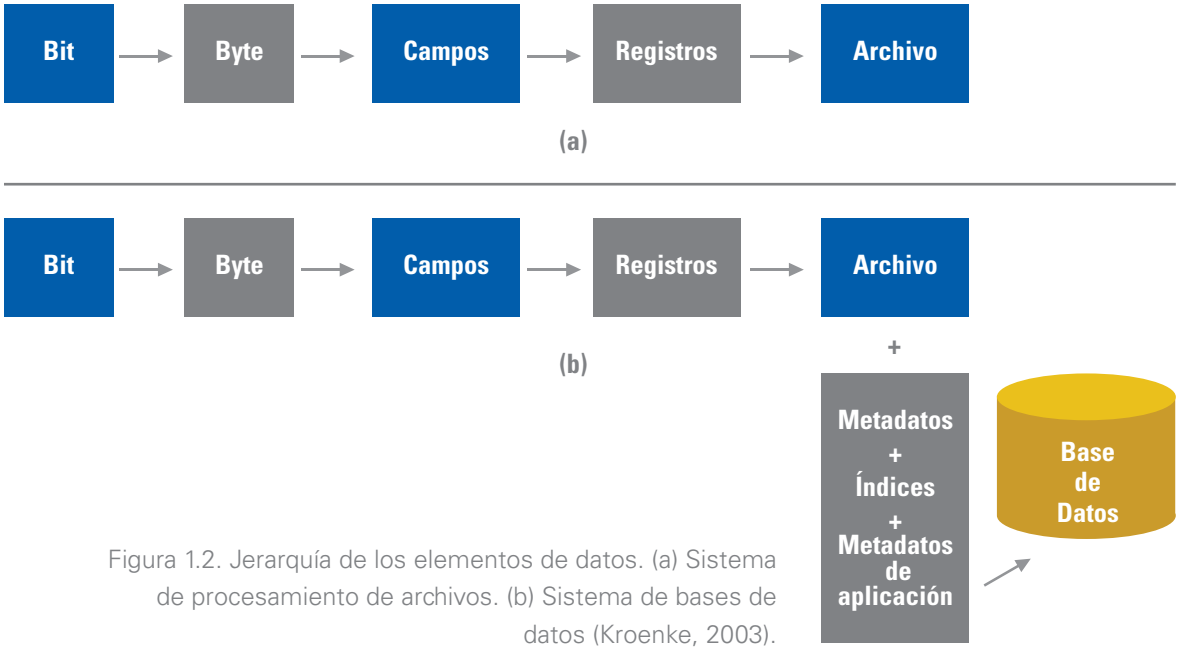


Figura 1.2. Jerarquía de los elementos de datos. (a) Sistema de procesamiento de archivos. (b) Sistema de bases de datos (Kroenke, 2003).

1.2 SISTEMAS DE PROCESAMIENTO DE ARCHIVOS Y SISTEMAS DE PROCESAMIENTO DE BASES DE DATOS

- **Sistemas tradicionales basados en archivos.** Los sistemas de archivos manuales no resultaban adecuados para el manejo de grandes cantidades de información, peor aún para la gestión permanente de la información que requiere generar reportes mensuales, semanales y hasta diarios. Resulta también complejo el manejo de archivos manuales cuando se necesita establecer referencias cruzadas o procesar la información contenida en documentos físicos.

  El sistema basado en archivos fue uno de los primeros intentos para informatizar los archivos manuales, éstos estaban orientados a cubrir necesidades específicas de procesamiento, por lo que, tanto los lenguajes de programación como las estructuras de datos se centraban en realizar de manera más eficiente una tarea específica.

  Los sistemas informáticos tradicionales se denominan sistemas orientados a procesos, debido a que en ellos se pone énfasis en los tratamientos que reciben los datos, los cuales se almacenan en ficheros que son diseñados para una determinada aplicación. Por ejemplo, en la Figura 1.3 se muestran tres necesidades del mundo real de una organización: la administración del personal, la gestión financiera y el control de vehículos. Para cada una de estas necesidades se crean sus archivos individuales y junto a éstos, las aplicaciones orientadas a los procesos. Diferentes son los problemas que se presentan en estos sistemas: datos aislados y separados, información duplicada, formatos de archivos incompatibles, proliferación de programas de aplicación y consultas fijas.

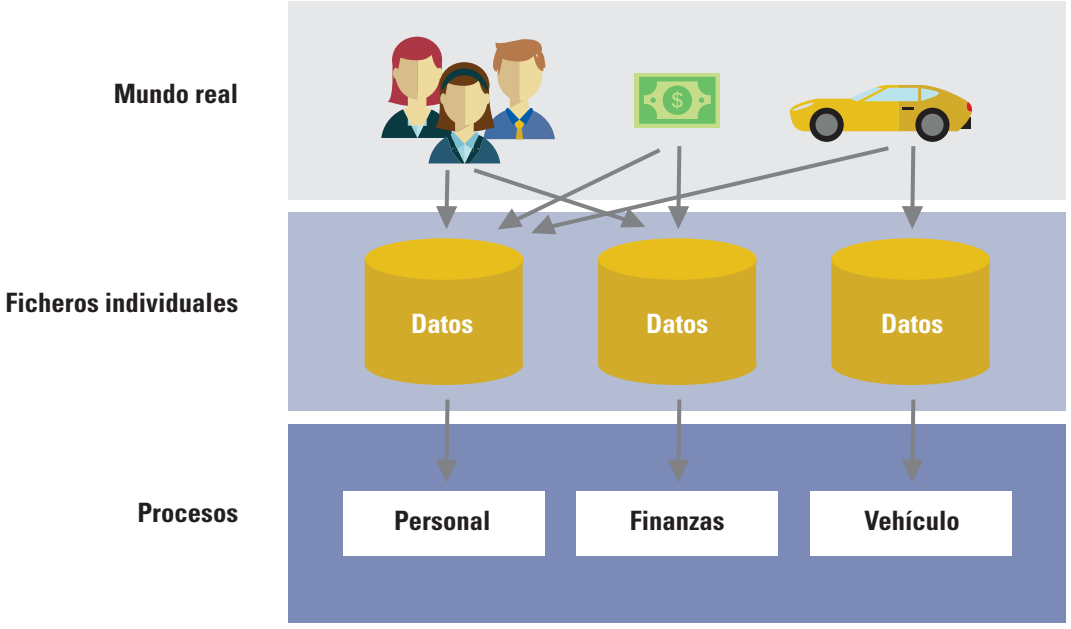


Figura 1.3. Sistema tradicional basado en archivos.

- **Sistemas basados en datos.** Con el fin de resolver los problemas que se presentan con los sistemas informáticos orientados a procesos y con el objeto de alcanzar una mejor gestión del conjunto de datos, nace un nuevo enfoque apoyado sobre una "base de datos", en donde los datos son recogidos y almacenados al menos lógicamente una sola vez, con independencia de los tratamientos, como se muestra en la Figura 1.4. En los sistemas basados en datos, el análisis comienza por determinar la lógica de los datos organizacionales como un todo independiente, para después involucrarlos con las aplicaciones que lo utilizan.

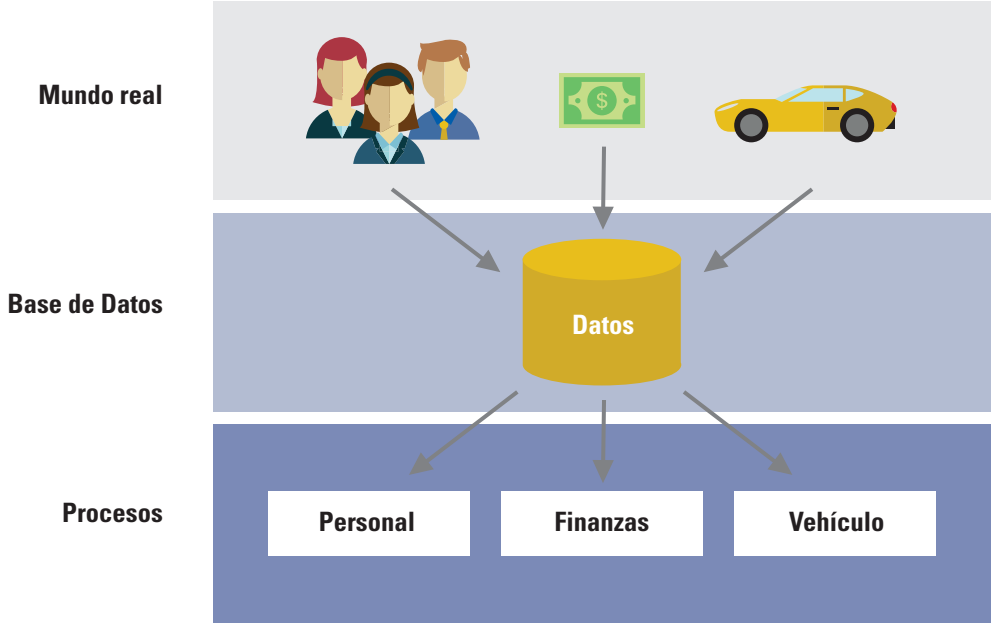


Figura 1.4. Sistema orientado a bases de datos.

1.3 OBJETIVOS DE LOS SISTEMAS DE BASES DE DATOS

Un sistema de procesamiento de archivos apoyado por un sistema operativo convencional, tiene un número de desventajas importantes que tienen que ser evitadas por un sistema de base de datos. Los sistemas operativos almacenan los datos en diferentes archivos y necesitan de varios programas de aplicaciones para acceder o añadir datos a esos archivos.

La tecnología de las bases de datos surge con el propósito de superar las limitaciones de los primeros métodos de gestión basados en los sistemas de procesamiento de archivos. Una importante diferencia entre estos dos métodos de gestión de datos está caracterizada en el acceso a los datos. Los programas de procesamiento de archivos acceden directamente a los archivos de los datos almacenados; por el contrario, los programas para el procesamiento de bases de datos, demandan al SGBD para tener acceso a los datos almacenados.

Guardar y gestionar la información de una organización en un sistema de procesamiento de archivos, lleva a una serie de inconvenientes significativos, que serán salvados con los sistemas de bases de datos; tema que se analizará a continuación:

- **Datos integrados.** En un sistema de base de datos, los datos de las aplicaciones se almacenan en un medio denominado **base de datos.** Por ejemplo, en un programa de facturación, la aplicación puede acceder a los datos del cliente, los datos de los artículos o ambos. En caso de necesitar ambos datos, el programador sólo especifica cómo deben combinarse los datos y encarga al SGBD para que realice las operaciones necesarias para conseguirlo.

  Cuando los datos se encuentran aislados, éstos están distribuidos en varios archivos, y pueden tener diferentes formatos, complicando la creación de nuevas aplicaciones para obtener los datos apropiados.

- **Independencia programa / datos.** Una característica importante de los sistemas de bases de datos es la independencia entre los datos y los tratamientos que se hacen sobre ellos. En los sistemas orientados a procesos, la estructura física y el almacenamiento de los archivos y registros de datos están definidos en el código del programa, por lo que resulta difícil realizar cambios a la estructura existente.

  En un sistema de bases de datos, los programas dependen menos de los formatos de archivo. Los formatos de registro se almacenan en la misma base de datos y son accedidos por el SGBD y no por los programas de aplicación.

  La independencia de programas y datos disminuye el efecto de los cambios en el formato de los datos de los programas de aplicación. En la mayoría de casos los programas de aplicación no se enteran de que el formato de los datos ha sido modificado.

- **Redundancia e inconsistencia de datos.** En un sistema de procesamiento de archivos, éstos y los programas de aplicación son creados por diferente personal durante un período largo de tiempo, en donde probablemente los archivos tengan diferentes formatos y puede ser que la información esté duplicada en varios sitios. Este problema ocasiona redundancia de la información, que aumenta los costos de almacenamiento y además lleva a inconsistencias de los datos, porque las diversas copias de los mismos no concuerdan entre sí.

  Por el contrario, la técnica de bases de datos elimina la redundancia de datos, integrando los archivos, de tal forma que no se almacenen diferentes copias con los mismos datos. No obstante, esta técnica no elimina por completo la redundancia, sino que la controla. Por ejemplo, en el diseño de la base de datos, es necesario duplicar las claves, para garantizar las relaciones entre archivos.

La redundancia genera inconsistencia en los datos, causando problemas serios, produciendo resultados incoherentes que crean incertidumbre. Si el reporte de una aplicación presenta resultados diferentes, ¿quién decide cuál es el correcto?, con lo que se crea la duda de los usuarios en la confiabilidad de los datos almacenados.

Por ejemplo, si la dirección de un empleado está registrada en el archivo `PERSONAL`, que contiene la información del personal de la empresa y además esta dirección está registrada en el archivo CONTROL, que almacena la información del control de asistencia de los empleados, un cambio en la dirección del empleado puede estar reflejada en el registro del personal, pero no estarlo en el registro del control de asistencia, produciendo una inconsistencia de los datos, debido a que las dos copias de la dirección no coinciden.

La duplicación de datos involucra desperdicio de recursos. Se consume un espacio de almacenamiento innecesario y el ingresar datos más de una vez, significa tiempo y dinero.

- **Problemas de integridad.** La integridad de datos hace referencia a los valores que serán almacenados en la base de datos. Estos valores deben satisfacer ciertos tipos de restricciones de consistencia, que son reglas de coherencia que no serán violadas en la base de datos. Las restricciones son aplicadas a los elementos de datos contenidos en un registro o a las relaciones entre registros.

  La integridad de datos se garantiza mediante la implementación por parte de los desarrolladores, de ciertos conceptos clave como: validación de datos, normalización, integridad referencial, entre otros; que serán añadidos a través del código correspondiente en los diferentes programas, con el objeto de hacer cumplir las restricciones de integridad. Por ejemplo, en una empresa se determina que el valor máximo del límite de crédito para los clientes no debe ser mayor a 3.000 dólares.

- **Acceso concurrente.** Uno de los principales objetivos de un sistema de base de datos, es permitir que varios usuarios accedan simultáneamente a una serie de datos compartidos. El acceso concurrente no presenta problemas si la operación que se realiza con los datos es una lectura, sin embargo; el problema se presenta cuando al menos uno de los usuarios que accede al mismo tiempo realiza una actualización a la base de datos. Sin el debido control estas operaciones pueden producir interferencias que ocasionan incoherencias en los resultados.

  El acceso concurrente aumenta el rendimiento global del sistema y permite obtener respuestas más rápidas, para lo cual, los SGBD deben proporcionar mecanismos para controlar la interacción entre las transacciones concurrentes y evitar que se destruya la consistencia de la base de datos. Por ejemplo, si se analiza el siguiente caso: la cuenta bancaria A tiene un saldo

de 1.000 dólares y dos usuarios acceden para retirar 100 y 50 dólares. Si las dos transacciones se ejecutan en serie, una después de la otra, sin que se entrelacen las operaciones, el saldo final independiente de la que se ejecutó primero será de 850 dólares. Sin embargo, si las dos transacciones se ejecutan al mismo tiempo, el resultado de esta ejecución concurrente puede dar lugar a obtener un saldo incorrecto o inconsistente. Se supone que las dos transacciones que se ejecutan para cada retiro, en primer lugar, leen el saldo, a continuación reducen el monto correspondiente a cada uno de sus retiros y luego escriben el resultado. Las dos transacciones que se ejecutan concurrentemente, leen el valor de 1.000 dólares y luego escriben 900 dólares y 950 dólares, en lugar de registrar el valor correcto de 850 dólares, con lo que se produce una inconsistencia en el saldo final.

El SGBD debe garantizar que, cuando los usuarios accedan de manera concurrente a una base de datos, no se produzcan interferencias de ningún tipo para garantizar la consistencia de los datos.

- **Control de seguridad.** El término seguridad hace referencia a los mecanismos que protegen la base de datos frente a accesos no autorizados, sean estos intencionados o accidentales. Ningún usuario puede tener acceso a toda la información de la base de datos. El SGBD debe proporcionar mecanismos que garanticen la seguridad de los datos, mediante medidas como nombres de usuarios y contraseñas que identifiquen a la persona autorizada a usar la base de datos o una parte de ella.

  La restricción de seguridad para los usuarios autorizados se define a nivel de acceso a los datos y según el tipo de operación que se realice (selección, actualización, inserción y borrado). Por ejemplo, en una empresa, el personal del departamento de contabilidad debería acceder exclusivamente a la parte de la base de datos con información contable, no necesita acceder a la información de la base de datos del control médico del personal.

- **Problemas de atomicidad.** Los sistemas de información están sujetos a fallos que pueden presentarse durante el procesamiento de una transacción, si las causas y efectos que generan estos fallos no son controlados, se puede producir inconsistencia en la base de datos. Un mecanismo que implementa el SGBD para garantizar la consistencia de los datos, está definido en la propiedad de atomicidad que debe poseer una transacción, garantizando que la ejecución de ésta se realice en su totalidad o no se realice en lo absoluto, asegurando que si se produce un fallo, los datos se restauren al estado consistente que existía antes del fallo. Por ejemplo, si se requiere transferir 1.000 dólares de la cuenta `A` la cuenta `B.` Si durante la ejecución de la transacción se produce un fallo en el sistema, es posible que se haya debitado los 1.000 dólares en la cuenta A pero todavía no se haya acreditado en la cuenta B, generando

en la base de datos un estado inconsistente, debido a que la transacción no se realizó en su totalidad. La transacción debió ser atómica, es decir realizarse en su totalidad o no realizarse en absoluto, para garantizar la consistencia de la base de datos.

1.4 CONCEPTO DE BASES DE DATOS

En este apartado se revisa el concepto de una base de datos, que en términos generales **representa una colección de datos relacionados, luego se analiza el conjunto de programas** para gestionar la base de datos denominado SGBD, que junto con los programas de aplicaciones y los usuarios, constituyen el sistema de base de datos.

Un sistema de base de datos se define como una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso a esos datos y modificarlos. (Silberschatz , A. Korth, H. Sudarshan, S., 2014)

Las definiciones de base de datos son numerosas y han ido cambiando y configurándose a lo largo del tiempo. A continuación se enuncian algunas definiciones planteadas por diferentes autores:

Definición 1:

"Colección de datos interrelacionados almacenados en conjunto, sin redundancia perjudicial o innecesaria. Su finalidad es servir a una o más aplicaciones de la mejor forma posible. Los datos se almacenan de modo que resulten independientes de los programas que lo usan. Se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer los datos almacenados ¨ (Martin, 1995)

Definición 2:

"Colección o depósito de datos donde los mismos se encuentran 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 servir para toda una gama de usos y aplicaciones". Citado en (Moratalla, 2001)

Definición 3:

"Una colección o depósito de datos integrados, con redundancia controlada y con una estructura que refleje las interrelaciones y restricciones existentes en el mundo real. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse inde-

pendientes de éstas, y su definición y descripción, han de estar almacenadas junto con los mismos. Los procedimientos de actualización y recuperación, comunes y bien determinados, habrán de ser capaces de conservar la seguridad (integridad, confiabilidad y disponibilidad) del conjunto de datos" (De Miguel, A., Piattini, M., 1993)

Definición 4

"Una colección compartida de datos lógicamente relacionados, junto con una descripción de estos datos, que están diseñados para satisfacer las necesidades de información de una organización" (Connolly T., Begg C., 2005)

Si se examina las definiciones se puede colegir que una base de datos es un depósito centralizado, posiblemente de gran tamaño, formado por datos que pueden ser utilizados al mismo tiempo por varios usuarios. Los elementos de datos están integrados, evitando que se produzca redundancia. En la base de datos no se almacenan únicamente datos operacionales, sino también la definición y descripción de dichos datos, que le dan la característica de auto descriptiva, a esta descripción se la conoce con el nombre de **catálogo del sistema** o **diccionario de datos** que contiene metadados, es decir datos acerca de los datos. Al diccionario de datos se lo puede considerar como un tipo especial de tabla, al que accede y actualiza únicamente el sistema de bases de datos y no el usuario normal. La característica auto descriptiva de la base de datos es aquella que proporciona la independencia entre datos y programas.

Los sistemas de bases de datos separan los programas de aplicación de la estructura de los datos, almacenando esta estructura en la propia base de datos. La acción de incrementar nuevas estructuras o modificar las existentes no afecta a los programas de aplicación, siempre y cuando éstos no dependan de la información que haya sido modificada. Este concepto recibe el nombre de **abstracción** de datos, y se refiere a la posibilidad de modificar la definición interna de un objeto sin afectar a los usuarios de dicho objeto, siempre y cuando la definición externa continúe siendo la misma. (Connolly T., Begg C., 2005)

En el análisis de las necesidades de información de una organización, para la gestión de los datos se identifican y almacenan diferentes objetos (entidades) y sus propiedades (atributos), que deben estar interrelacionadas conformando una colección de datos lógicamente relacionados.

Las bases de datos son creadas para servir a toda la organización, es decir, a varias aplicaciones y múltiples usuarios.

De otra parte, un SGBD es un conjunto de programas que permite a los usuarios definir, crear, mantener y controlar el acceso a la base de datos. Es el software que interactuará entre la base de

datos y los programas de aplicación de los usuarios. A continuación se presenta un resumen de las funciones de un SGBD, tema que será revisado con mayor detalle en la Sección 1.9.

- Permite a los usuarios insertar, borrar, actualizar y consultar datos de la base de datos.
- Permite a los usuarios definir la base de datos mediante la especificación de las estructuras, tipos de datos, y sus restricciones.
- Permite mantener la coherencia de los datos almacenados.
- Permite el acceso compartido a los datos mediante el control de concurrencia.
- Evita el acceso a la base de datos a los usuarios no autorizados mediante el sistema de seguridad.
- Si se presentan fallos en el hardware o software, permite restaurar la base de datos a un estado previo coherente, mediante el sistema de control de recuperación.
- Proporciona un catálogo que contiene la descripción de los datos y que se almacena en la base de datos.

Por último se analizan los programas de aplicación que interactúan con la base de datos, formulando las solicitudes (por lo general son instrucciones SQL) dirigidas al SGBD. Estos programas de aplicación pueden estar escritos en alguna herramienta de programación y se utilizan para mantener la base de datos y generar información.

1.5 ABSTRACCIÓN DE DATOS

Un objetivo importante de los sistemas de bases de datos es proporcionar a los usuarios una visión *abstracta* de los datos, es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos. Considerando que muchos usuarios no están familiarizados y no tienen su

formación en los sistemas de bases de datos, se les oculta la complejidad de los datos a través de diversos niveles de abstracción para facilitar su interacción con el sistema.

En los sistemas de información tradicionales se puede observar la existencia de dos estructuras distintas:

- Lógica - externa (vista de usuario)
- Física - interna (forma en la que se encuentran almacenados los datos)

1.5.1 Niveles de abstracción

En los sistemas de bases de datos, aparece un nuevo nivel de abstracción denominado nivel conceptual o intermedio, que permite una representación global de los datos entre las estructura lógica y física, pero con independencias del equipo y de los usuarios. En términos generales, el *nivel externo* es la visión que cada usuario en particular tiene de la base de datos, el *nivel conceptual* corresponde al enfoque del conjunto de la empresa, y el *nivel interno* es la forma cómo los datos se organizan en el almacenamiento físico. En la Figura 1.5 se muestra los tres niveles de abstracción de datos.

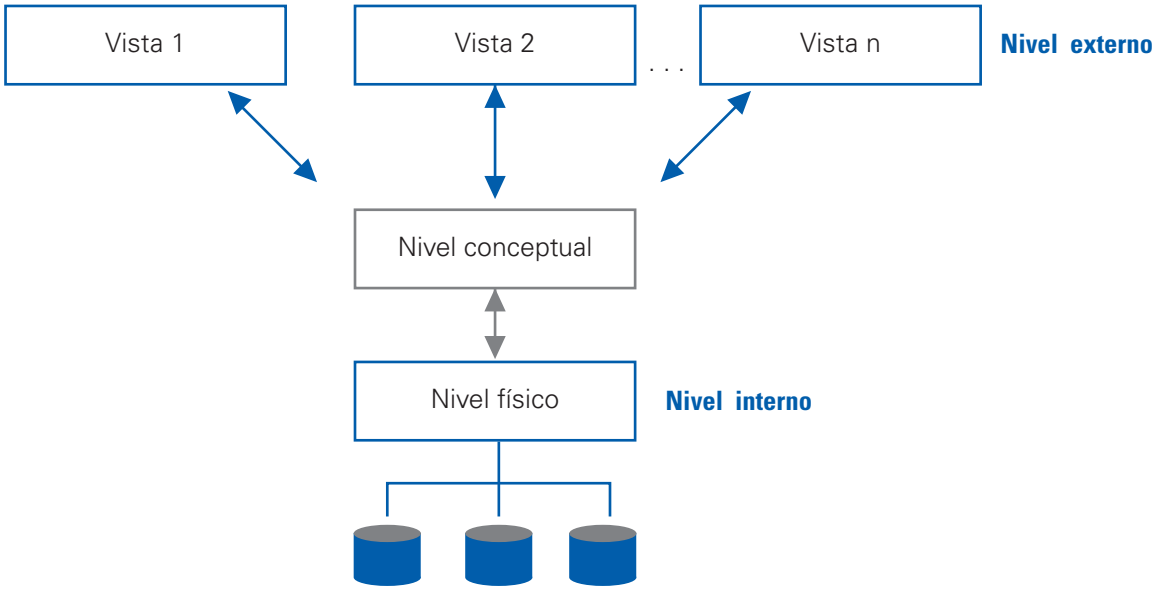


Figura 1.5. Arquitectura de tres niveles de abstracción de datos.

- **Nivel físico o interno.** Es el nivel más bajo de abstracción. Corresponde a la representación física de la base de datos en la computadora, describe *cómo* se almacenan realmente los datos. Se distinguen tres aspectos:

  Estrategia de almacenamiento. Incluye básicamente la asignación de espacio de almacenamiento para el conjunto de datos e índices, las estrategias para optimizar tiempo y espacio en memoria secundaria, y el tratamiento de desbordamientos.

  Caminos de acceso. Compete a las especificaciones de claves primarias y secundarias, así como la de punteros e índices.

  Misceláneos. Corresponde a las técnicas de cifrado y compresión de datos.

- **Nivel conceptual.** Este nivel describe *qué* datos son realmente almacenados en la base de datos y las relaciones que existen entre los mismos. El nivel conceptual contiene la estructura lógica de toda la base de datos. Este nivel usan los administradores de bases de datos, quienes deciden qué información se almacenará en la base de datos.

  En el nivel conceptual debe incluirse todas las entidades, atributos y sus relaciones, las restricciones aplicadas a los datos, la semántica y la información de seguridad e integridad de los datos.

- **Nivel de visión o externo:** Es el nivel más alto de abstracción. Describe la parte de la base de datos que es de interés particular para cada usuario en un número de vistas.

  Las vistas pueden constituir diferentes representaciones de los mismos datos. Por ejemplo, los diversos formatos de presentación de la fecha, un usuario puede verlos con el formato (dd, mm, aaaa), en tanto que otro puede verlos como (aaaa, mm, dd). Las vistas pueden también incluir datos derivados o calculados, que son creados cada vez que se necesitan, a partir de los datos almacenados en la base de datos.

1.5.2 Independencia de datos

La independencia de datos es uno de los objetivos de la arquitectura de tres niveles. Esta proporciona la capacidad para modificar una definición de esquema en un nivel inferior sin que afecte a una definición de esquema en el siguiente nivel más alto. Existen dos tipos de independencia de datos:

- **Independencia física de datos:** este concepto hace referencia a la capacidad de modificar el esquema físico (interno) sin tener que cambiar los esquemas conceptual o externo. En

algunas ocasiones son necesarias las modificaciones en el nivel físico para mejorar el funcionamiento, como por ejemplo, utilizar distintas estructuras de almacenamiento, modificar y eliminar índices, utilizar diferentes organizaciones de archivos, cambios de tamaño de bloques, cambios en las direcciones relativas y absolutas de almacenamiento.

- **Independencia lógica de datos:** es la capacidad de modificar el esquema conceptual, sin necesidad de modificar los esquemas externos existentes, ni reescribir los programas de aplicación. Las modificaciones en el nivel lógico son necesarias siempre que la estructura lógica de la base de datos se altere. Por ejemplo, incluir o eliminar entidades, introducir nuevos atributos, o realizar cambios de nombre y tipo.

1.5.3 Esquema e instancia

La información almacenada en la base de datos en un determinado momento en el tiempo, recibe el nombre de **instancia o ejemplar** (Silberschatz , A. Korth, H. Sudarshan, S., 2014) , por otra parte, la descripción global de la base de datos se denomina esquema. El **esquema** se especifica durante el proceso de diseño de la base de datos y no suele modificarse con frecuencia.

Los sistemas de bases de datos tienen varios esquemas que se definen de acuerdo con los niveles de abstracción. En el nivel más bajo de abstracción está el esquema físico o interno, nivel que describe las estructuras utilizadas para el almacenamiento, la definición de los registros y los índices. En el nivel lógico o conceptual se encuentra el esquema conceptual, en el que se describen todas las entidades, atributos y relaciones. En el nivel más alto de abstracción, nivel de vistas, se tienen diferentes esquemas externos, a veces denominados subesquemas, que describen las diferentes vistas de los datos. Solo existe un esquema físico y un esquema conceptual en la base de datos.

1.6 MODELOS DE DATOS

Un modelo de datos consiste en una colección integrada de herramientas conceptuales que sirven para: describir los datos, sus relaciones, las restricciones de consistencia y la semántica asociada a cada uno de ellos.

Según Connolly (2005), los modelos de datos comprenden tres componentes:

- El primero corresponde a un componente **estructural**, conformado por un conjunto de reglas que definen cómo se construyen las bases de datos.

- El segundo componente de **manipulación**, corresponde a las operaciones para extraer y actualizar datos y para modificar la estructura de la base de datos.

- El último componente es el conjunto de **restricciones de integridad** derivadas de las reglas del mundo real y que sirven para garantizar la precisión de los datos.

Los modelos se clasifican en tres categorías: lógicos basados en objetos, lógicos basados en registros y físicos.

- **Modelo lógico basado en objetos:** este modelo describe los datos a nivel conceptual y de visión, los más conocidos son:

  Modelo entidad – relación

  Modelo orientado a objetos

  Modelo semántico de datos

  El modelo de datos entidad – relación (ER) se ha afianzado como una de las principales técnicas para el diseño de bases de datos. Utiliza conceptos tales como entidades, atributos y relaciones que se examinan a detalle en el Capítulo 2.

  El modelo de datos orientado a objetos se puede considerar como una extensión del modelo ER, en él se incluyen métodos, encapsulamientos e identidad de objetos. En este modelo no se describe únicamente el estado de los objetos, sino también las acciones que se encuentran asociadas con ellos.

- **Modelo lógico basado en registros:** los modelos lógicos basados en registros se utilizan para describir datos en los niveles conceptual y físico.

  La base de datos de este modelo está estructurada en registros de formato fijo de varios tipos, cada tipo de registro define un número fijo de campos y cada campo es de longitud fija. Existen tres tipos de modelos basados en registros.

Modelo de datos relacional

Modelo de datos en red

Modelo de datos jerárquico

Modelo Relacional: representa los datos y las relaciones entre ellos mediante una colección de tablas, cada una de las cuales tiene un número de columnas con nombres únicos. Las tablas también son conocidas como relaciones. En la Figura 1.6 se muestran dos tablas para el manejo de la información de un cliente y el saldo de su cuenta bancaria. Las dos tablas están interrelacionadas mediante la columna *Número.* En el Capítulo 3 se revisa el modelo relacional en detalle.

Nombre	Calle	Ciudad	Número
Pérez	Sucre	Cuenca	900
Torres	Amazonas	Quito	556
Torres	Colón	Quito	647
Jara	9 de Octubre	Guayaquil	801
Jara	Machala	Guayaquil	647

Tabla Cliente

Número	Saldo
900	55
556	10000
647	15000
801	60520

Tabla Saldo

Figura 1.6. Tablas de un modelo relacional.

- **Modelo físico basado en datos:** se usa para describir datos en el nivel más bajo, representan información como: rutas de acceso, el ordenamiento y las estructuras de registros. Los modelos físicos son limitados comparados con los modelos lógicos.

1.7 USUARIOS DE UNA BASE DE DATOS

En esta sección se identifican las personas que están involucradas en el diseño, uso y mantenimiento de una base de datos.

- **Administrador de una base de datos (ABD):** (DBA por sus siglas en inglés *database administrator*) es el responsable del diseño, control y administración de la base de datos, actividades que pueden ser desempeñadas por una persona o un grupo de personas dependiendo de la envergadura del proyecto. En un sistema de base de datos es responsabilidad del administrador gestionar los siguientes recursos: la base de datos, el SGBD y los programas relacionados.

  Las tareas específicas de un administrador de la base de datos se resumen en las siguientes:

  - *La estructura de la base de datos.* Determina qué información va a ser necesaria almacenar en la misma, después de haber analizado los requisitos de los distintos usuarios.

  - *La descripción conceptual y lógica de la base de datos.* Una vez definidos los requisitos de la información, es preciso realizar el diseño conceptual de la base de datos, para luego, adecuar la estructura conceptual a un SGBD específico.

  - *La descripción física de la base de datos.* Encontrar una estructura interna que soporte el esquema lógico y los objetivos de diseño. Es una labor que se extiende a lo largo de la vida de la base de datos. El ABD tendrá que variar parámetros, reorganizar los datos, modificar estructuras de almacenamiento, realizar nuevas distribuciones de los ficheros en los soportes, entre otras.

  - *Definición de estándares con los que se va a regir la organización.*

  - *Administrar los aspectos relacionados con la seguridad.*

  - *Gestionar el control y la interacción entre la red y la base de datos.*

  - *Ocuparse de los procedimientos de explotación y uso.*

- **Diseñador de la base de datos:** es responsable de identificar las entidades, los atributos, las relaciones y las restricciones que se tienen que aplicar a los datos que serán almacenados en la base de datos, y escoger las estructuras apropiadas.

En grandes proyectos de diseño de bases de datos se puede considerar dos tipos de diseñadores: lógicos y físicos. Los primeros se encargan de la definición de entidades y sus relaciones, en tanto que los segundos son los que materializan físicamente el diseño lógico de la base de datos, seleccionando estructuras apropiadas, métodos de acceso y medidas de seguridad.

- **Programadores de aplicaciones:** una vez implementada la base de datos se requiere escribir los programas de aplicaciones con las funcionalidades requeridas por los usuarios finales, en las que se incluyen comandos del SGBD para realizar operaciones sobre la base de datos, tales como: extraer, insertar, borrar y actualizar datos. Los desarrolladores de aplicaciones, para programar la interfaz del usuario, eligen de acuerdo a su conveniencia, de entre muchas herramientas de software disponibles.

- **Usuarios finales:** son los "clientes" de la base de datos, personas que en su trabajo requieren el acceso a los datos para realizar consultas, actualizaciones y generar reportes. Existen dos categorías de acuerdo a la forma de acceso a la base de datos:

  - *Usuarios normales:* usuarios que hacen consultas a la base de datos, habitualmente no están familiarizados con los SGBD y no están conscientes de su existencia. Acceden a los datos a través de un programa de aplicación creado a propósito, esto implica que el usuario no necesita tener conocimiento de la estructura de la base de datos ni del funcionamiento del SGBD. Por ejemplo, un vendedor de supermercado que tiene que añadir un artículo en una factura, usa un programa de aplicación que puede llamarse *nueva_factura*. Este programa le pide al vendedor que ingrese el código del artículo y la cantidad vendida, para luego elaborar la factura sin que el usuario advierta la existencia de la base de datos.

  - *Usuarios sofisticados:* pueden realizar consultas y modificaciones a la base de datos utilizando un lenguaje de alto nivel, como SQL (*Structured Query Language,* lenguaje estructurado de consulta). Para cumplir con este propósito, el usuario debe estar familiarizado con la estructura de la base de datos y con las funcionalidades que ofrece el SGBD.

1.8 LENGUAJE DE BASES DE DATOS

Para llevar a cabo las distintas funciones que cumple un SGBD es necesario contar con diferentes lenguajes y procedimientos que permitan la comunicación con la base de datos. Los siste-

mas de bases de datos proporcionan dos tipos de lenguaje, uno para especificar el esquema de la base de datos y el otro para expresar las consultas y actualizaciones de los datos. En la práctica estos dos lenguajes no funcionan en ambientes diferentes, sino que forman parte de un único lenguaje denominado SQL. En la Figura 1.7 se ilustran estos dos conceptos de lenguaje.

Lenguaje de definición de datos (LDD): DDL por sus siglas en inglés (*Data definition language*) es un lenguaje especial basado en un conjunto de definiciones que sirve para especificar el esquema de una base de datos. El LDD permite a los usuarios describir y nombrar las entidades, atributos y relaciones, junto con sus restricciones de integridad asociadas como: asertos, restricciones de dominio e integridad referencial y las autorizaciones para el control de seguridad.

La restricción de dominio declara los posibles valores que pueden adoptarse para un determinado atributo. El sistema comprueba esta restricción, siempre que se ingresa un nuevo elemento de dato en la base de datos.

La integridad referencial garantiza que un valor que aparece en una tabla para un conjunto de atributos dado, aparezca también para un conjunto de atributos en otra tabla. El tema de integridad referencial se estudia con mayor detalle en el Capítulo 5.

Los **asertos** son condiciones que la base de datos debe cumplir siempre. Hay ciertas restricciones que no pueden expresarse empleando las restricciones de integridad referencia o de dominio. Por ejemplo, para asignarle un límite de crédito a un cliente, éste debe tener al menos cinco facturas de compra. Esta condición debe expresarse en forma de aserto.

El resultado de la compilación de sentencias LDD es un conjunto de tablas que contienen metadatos, es decir datos de los datos, que se almacenan en un archivo llamado *Diccionario de datos* (DD).

Lenguaje de manipulación de datos (LMD). DML por sus siglas en inglés (Data manipulation language). Su objetivo es proporcionar una interacción eficiente entre los usuarios y el sistema que permita la recuperación, supresión, inserción y modificación de la información almacenada en la base de datos. Existen dos tipos de lenguajes de manipulación de datos:

No procedimentales: también conocidos como declarativos, requieren que el usuario determine qué datos se necesitan, sin especificar cómo obtenerlos. Los SGBD para manipular los registros requeridos, traducen la instrucción LMD a uno o más procedimientos. Estos procedimientos eximen al usuario de la obligación de conocer cómo están implementadas las estructuras de datos y qué algoritmo necesita para extraerlos y tratarlos.

Los sistemas de gestión de bases de datos relacionales, para la manipulación de datos, generalmente incluyen algún tipo de lenguaje no procedimental, el más utilizado es el lenguaje de consultas estructurado SQL.

Procedimentales: este lenguaje permite al usuario decirle al sistema *qué* datos se necesitan y *cómo* obtenerlos. En este tipo de lenguajes el programador expresa todas las operaciones de acceso a los datos que hay que utilizar, recurriendo a los procedimientos apropiados para obtener la información. Por ejemplo, los lenguajes de los sistemas jerárquico y de red, son de tipo procedimental.

En ciertos SGBD en donde hay una marcada separación entre los niveles conceptual e interno, los lenguajes de definición de datos se utilizan únicamente para especificar el esquema conceptual. Para especificar el esquema interno se usa el lenguaje de definición de almacenamiento LDA (SDL por sus siglas en inglés *Storage definition language*) y para especificar las vistas de los usuarios se necesita un cuarto lenguaje, el de definición de vistas LDV (VDL por sus siglas en inglés *View definition language*).

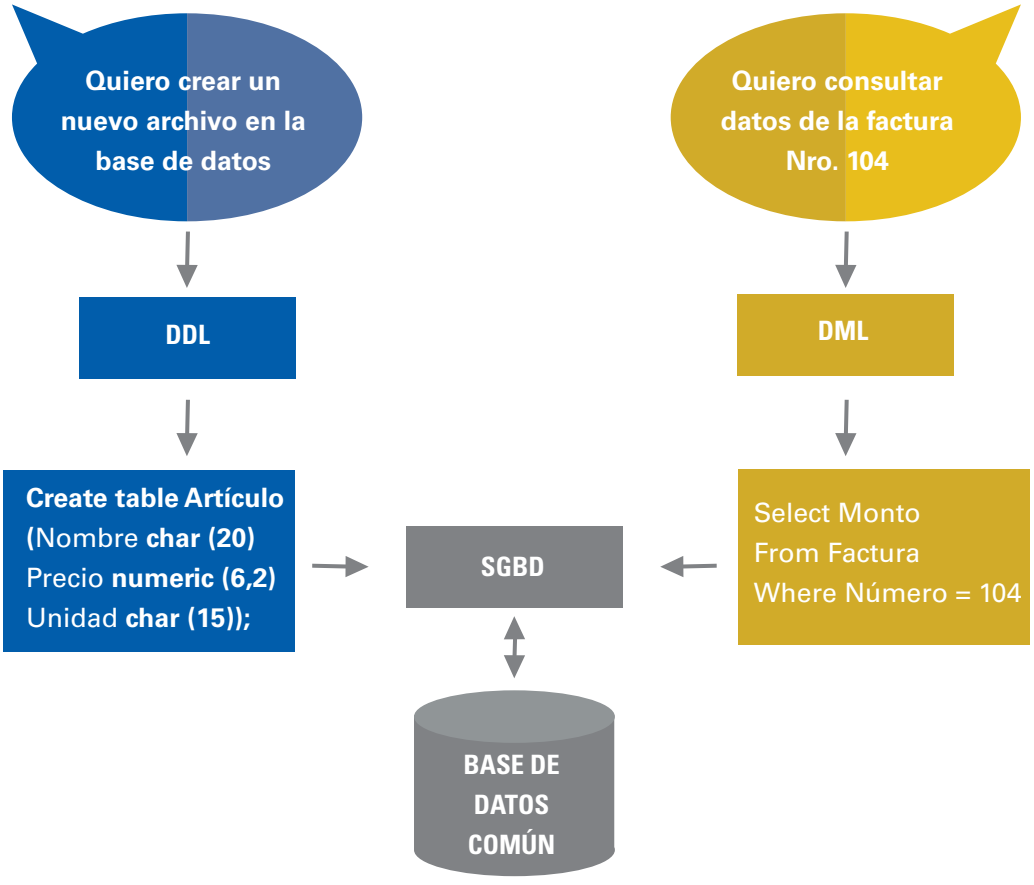


Figura 1.7. Representación de la acción de un lenguaje de base de datos.

1.9 SISTEMA DE GESTIÓN DE BASE DE DATOS (SGBD)

DE MIGUEL y otros (1993) definen a un sistema de gestión de base de datos como: "*Al conjunto coordinado de programas, procedimientos, lenguajes, etc., que suministra, tanto a los usuarios no informáticos como a los analistas, programadores, o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su seguridad.*"

Al SGBD se lo considera como un modelo de programa, que sirve como interfaz entre los datos de bajo nivel almacenados en la base de datos y los diferentes programas de aplicación y consultas para acceder a los datos, como se muestra en la Figura 1.8.

Un gestor de bases de datos es responsable de las siguientes funciones:

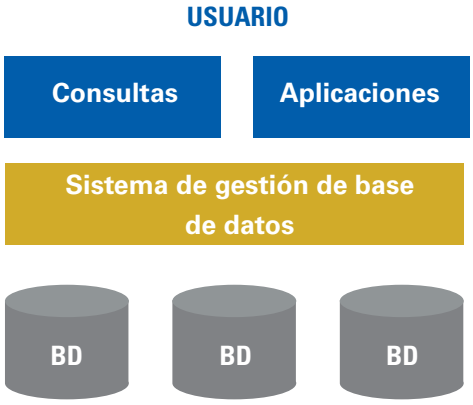


Figura 1.8. Relación entre el SGBD, los datos y sus aplicaciones.

- **Interacción con el gestor de archivos:** es responsable de proporcionar al usuario la capacidad de almacenamiento, recuperación y actualización de los datos en la base de datos. El gestor de base de datos traduce las sentencias LMD a comandos del sistema de archivos de bajo nivel, para procesar los datos almacenados en el disco.

- **Control de concurrencia:** es responsabilidad del sistema de gestión de base de datos conservar la consistencia de los datos, cuando varios usuarios actualizan la base de datos de manera concurrente.

  El acceso concurrente no presenta mayor inconveniente cuando la acción que realizan los usuarios sobre la base de datos es únicamente de lectura, puesto que no hay forma de que puedan interferirse unos con otros; no obstante, pueden producirse interferencias y

generar incoherencia en los datos, cuando varios usuarios acceden al mismo tiempo a la base de datos y al menos uno de ellos está realizando una acción de escritura (insertar, modificar o eliminar). El SGBD debe controlar y garantizar que no se produzcan interferencias entre las transacciones concurrentes a una serie de datos compartidos.

- **Implantación de la integridad:** controla que todos los datos almacenados en la base de datos así como los cambios, cumplan con ciertas restricciones de consistencia. La integridad de los datos se expresa mediante el término de restricciones, que no son más que reglas de coherencia que el gestor de base de datos no debe permitir que sean violadas. Por ejemplo, se podría especificar una restricción que señale que los clientes no pueden tener un límite de crédito mayor a 2.000 dólares. En este caso, el SGBD comprobará que cuando se ingresa o modifica un límite de crédito, éste no podrá ser superior a 2.000 dólares.

- **Catálogo del sistema:** el SGBD genera un catálogo del sistema llamado también diccionario de datos, en donde se almacena información que describe los datos contenidos en la base de datos, es decir, metadatos a cerca de la estructura de la base de datos, información a la que no solo debe acceder el SGBD, sino también puedan acceder cierto tipo de usuarios.

  Por lo general el catálogo del sistema maneja la siguiente información, que varía de un SGBD a otro:

  - nombres de las tablas

  - nombre, tipo y tamaño de cada una de las columnas

  - restricciones de integridad

  - nombre de los usuarios autorizados

  - datos a los que puede acceder el usuario y los tipos de acceso autorizados (eliminación, inserción, lecturas y modificación)

  - información estadística de uso de los datos, como número de accesos realizados y la frecuencia

- **Implantación de la seguridad:** no todos los usuarios de la base de datos necesitan tener acceso a todo su contenido. Es función del SGBD administrar estas seguridades, garantizando que sólo los usuarios autorizados puedan acceder a la base de datos. Se entiende

como seguridad, a la protección de la base de datos frente a accesos no autorizados, sean estos intencionados o accidentales.

- **Función de recuperación:** es responsabilidad del gestor de bases de datos detectar las fallas ocasionadas por el hardware o software, y restaurar la base de datos al estado coherente en el que se encontraba antes de ocurrido el fallo.

1.10 COMPONENTES DE UN SISTEMA DE GESTIÓN DE BASES DE DATOS

En este apartado se revisarán los diferentes programas (software) que forman parte de los componentes de un SGBD, cada uno de los cuales cumplen una operación específica y se encuentran relacionados entre sí o con otros componentes, como por ejemplo el sistema operativo subyacente.

No es posible generalizar la estructura de los componentes de un SGBD. Estos varían de un sistema a otro, sin embargo, en la Figura 1.9 se visualiza una arquitectura en la que se ilustran los componentes y sus relaciones.

En la parte superior de la Figura 1.9 se muestran las interfaces para los usuarios. Los usuarios normales, cuya labor principal es la de consultar y actualizar constantemente la base de datos mediante programas de aplicación, que han sido creados y probados por los desarrolladores. Un segundo grupo de usuarios son los programadores, que escriben aplicaciones utilizando algún lenguaje host. El tercer grupo de usuarios son los denominados sofisticados, que trabajan con interfaces interactivas para formular consultas. Por último están los administradores, que a través de un conjunto de instrucciones de definición de datos LDD crean el esquema original de la base de datos.

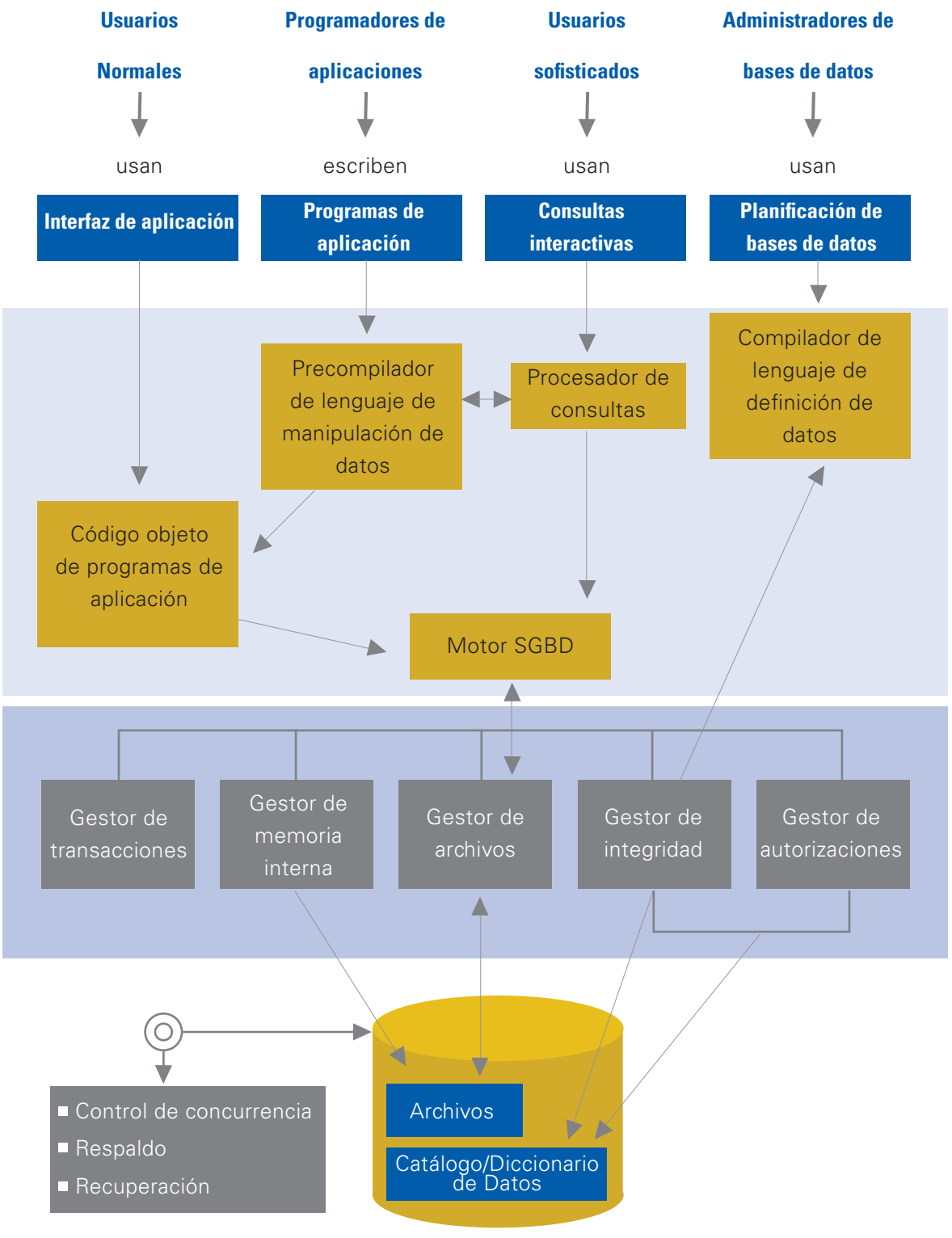


Figura 1.9. Componentes de un sistema de base de datos.

Procesador de consultas. Este componente se encarga de analizar a la consulta sintácticamente, validarla, optimizarla y ejecutarla. Transforma la consulta escrita en un lenguaje de alto nivel SQL, a una serie de instrucciones de bajo nivel (algebra relacional), para luego ejecutar una estrategia y recuperar los datos.

El procesador de consultas inicia con el analizador de léxico, en el que se identifican los elementos del lenguaje como por ejemplo, los nombres de las columnas, de las tablas y las palabras reservadas de SQL; luego el analizador sintáctico comprueba la sintaxis de la consulta, la cual tiene que ser validada; comprobándose que todos los nombres de columnas y tablas sean válidos y que tengan significado semántico en el esquema de la base de datos. A continuación el SGBD construye un árbol del análisis de la consulta que se transformará en una expresión del álgebra relacional. El sistema dispondrá de varias estrategias distintas para la ejecución de la consulta, la más adecuada recibe el nombre de *plan de ejecución de la consulta* y el proceso que realiza el SGBD para la elección se denomina *optimización de consultas*. A continuación, en función del plan de ejecución, se prepara el código con la secuencia de operaciones para la evaluación de la consulta, que será ejecutado por el procesador de base de datos para generar el resultado.

Motor SGBD. El motor SGBD se comunica con las consultas enviadas por el usuario y con los programas de aplicación, recibe los requerimientos en términos de tablas y columnas y luego traduce en órdenes dirigidas al sistema operativo para leer y escribir datos en medios físicos. Este componente del SGBD también administra el manejo de transacciones, bloqueos, respaldos y recuperación. Al motor del SGBD se lo conoce también como gestor de base de datos o DM por sus siglas en inglés (database manager)

Precompilador de LMD. Los programadores de aplicaciones escriben las aplicaciones en un lenguaje host como por ejemplo java, luego el programa es enviado al precompilador en donde se extraen las instrucciones LMD integradas a la aplicación. En esta instancia se ejecutan dos acciones, la primera es generar el código objeto del programa y la segunda enviar las instrucciones al procesador de consultas para extraer los datos.

Compilador LDD. Los administradores de la base de datos definen e introducen cambios en la base de datos mediante el LDD. Este compilador procesa las definiciones de esquema especificadas en el lenguaje de definición de datos y almacena en el diccionario de datos la descripción de los esquemas o metadatos.

Gestor de almacenamiento. El SGBD cuenta con un componente responsable del almacenamiento, recuperación y actualización de la base de datos, denominado gestor de almacenamien-

to. Este componente de software proporciona la interfaz entre los programas de aplicaciones, las consultas enviadas al sistema y los datos de bajo nivel almacenados en la base de datos. Los principales componentes del software del gestor de almacenamiento son los siguientes:

- *Gestor de transacciones.* El SGBD debe garantizar que la ejecución de las transacciones se realice correctamente a pesar de la existencia de fallos en el sistema, asegurando que la base de datos permanezca en un estado consistente.

- *Gestor de autorizaciones.* Este componente comprueba que los usuarios tengan las autorizaciones necesarias para el acceso a los datos y lleva a cabo las operaciones requeridas.

- *Gestor de integridad.* Para las operaciones de actualización de la base de datos, este componente verifica que se satisfagan todas las restricciones de integridad.

- *Gestor de archivos.* Administra la asignación de espacio de almacenamiento en disco, establece y mantiene las estructuras de datos e índices usadas para representar la información almacenada en el espacio físico.

- *Gestor de memoria interna.* Este componente controla el acceso a la información almacenada (datos y metadatos), es responsable de la transferencia de datos desde el disco de almacenamiento hasta la memoria principal, empleando servicios básicos del sistema operativo.

1.11 CUESTIONARIO

1. Defina los términos base de datos y sistema de gestión de base de datos.
2. Explique la diferencia entre un sistema basado en archivos y un sistema basado en datos.
3. Enumere los objetivos de un sistema de base de datos.
4. Explique el término abstracción de datos.
5. ¿Cuáles son los niveles de abstracción? Explique brevemente cada uno de ellos.
6. Explique los términos independencia física e independencia lógica de datos.
7. Enumere y explique los tres componentes de los modelos de datos.
8. Enumere los usuarios de un sistema de base de datos.
9. Cuáles son las tareas de un administrador de base de datos (ABD).
10. Explique el significado de lenguaje de definición de datos y lenguaje de manipulación de datos.
11. ¿Qué son los lenguajes no procedimentales?
12. ¿Cuáles son las funciones de un sistema de gestión de base de datos?
13. Enumere los componentes de un sistema de gestión de base de datos.

MODELO ENTIDAD - RELACIÓN

Una vez concluida la recopilación y análisis de requisitos del usuario para un sistema de base de datos, la siguiente etapa de acuerdo al ciclo vital del sistema de información (Elmasri R., Navathe S., 2016), corresponde al diseño de la base de datos, a través de un modelo que contenga herramientas conceptuales y metodologías comprensibles, fáciles de entender. El modelo Entidad – Relación (ER) cumple con estas condiciones y es el tema que se tratará en el presente capítulo.

En 1976 Peter Chen propuso este modelo conceptual de alto nivel, que no examina las dificultades del almacenamiento y consideraciones de eficiencia que son tratadas en el diseño físico de la base de datos. El modelo ER utiliza una técnica de diseño denominada "arriba abajo", que parte de la identificación de los datos más importantes a los que se los conoce como *entidades*, y las relaciones entre los datos que debe modelarse, para luego, determinar los *atributos*, que son las características o propiedades de las entidades. A estos conceptos se aplicarán las reglas de coherencia o restricciones, que servirán para garantizar la integridad de la base de datos.

En este capítulo se presentan los conceptos del modelo Entidad – Relación (ER) y la metodología para el diseño conceptual de las aplicaciones de base de datos. En la Sección 2.1 se presenta un caso del mundo real denominado **La Ferretería**, que nos servirá para ejemplificar los diferentes conceptos de la estructura de datos, las restricciones del modelo y las técnicas de diagramación. Este caso de estudio se utilizará a lo largo del presente libro. En la Sección 2.2 se analizan los conceptos de entidad, atributo y relación. En la Sección 2.3 se revisa una notación alternativa para los conceptos de cardinalidad y participación. La Sección 2.4 introduce los conceptos de dependencia de existencia y su relación con las entidades débiles y fuertes. En la Sección 2.5 se analizan los atributos que forman parte de una relación. Una vez revisados los conceptos y la metodología del modelo Entidad - Relación, en la Sección 2.6 a manera de repaso, se perfecciona el modelo de nuestro caso de estudio La Ferretería. En la Sección 2.7 se presenta un resumen de la notación utilizada para los diagramas de modelo Entidad – Relación. La Sección 2.8 presenta los algoritmos y procedimientos para crear un esquema relacional a partir de un esquema conceptual (Entidad – Relación). Al final de esa sección se incluye el esquema relacional como resultado de la transformación a tablas del caso de estudio La Ferretería. En la Sección 2.9 se introduce un concepto abstracto para el diseño de bases de datos denominado agregación. El capítulo concluye con la Sección 2.10 en la que se plantea un cuestionario de repaso y casos del mundo real con ejercicios resueltos y propuestos para el desarrollo de los estudiantes.

2.1 EJEMPLO DE APLICACIÓN DE BASES DE DATOS

El caso que se analiza en esta sección sirve como ejemplo para introducir los conceptos del modelo Entidad – Relación y cómo aplicarlos en el diseño de una base de datos. Partimos de una descripción de los requisitos de datos de los usuarios (mundo real) para luego gradualmente crear el esquema conceptual. La base de datos denominada La Ferretería presenta los siguientes requerimientos:

- Para automatizar su sistema de facturación, las compras que realiza un cliente se registran en una factura que será elaborada por un vendedor. La factura debe registrar el número y fecha de elaboración, los artículos vendidos y las cantidades de cada uno de ellos.

- Para los clientes es necesario que se almacene su nombre, apellidos, cédula de ciudadanía, dirección y número de teléfono o teléfonos. Si tiene asignado un límite de crédito, se debe registrar su valor.

- Los datos que se registren para los vendedores son: nombre, apellido, sexo, teléfono, objetivo de ventas y la fecha del contrato de trabajo.

- Cada cliente tendrá asignado a un vendedor como representante, esto no significa que únicamente el vendedor asignado sea quien elabore la factura a su cliente. Adicionalmente se almacenará la fecha de asignación.

- Para el control del subsidio familiar se requiere contar con datos de las personas a cargo de un vendedor. Para cada carga familiar se registrará su nombre, fecha de nacimiento, sexo y parentesco con el vendedor.

- Con el objeto de asignar descuentos a los clientes, se requiere llevar un registro de los clientes que recomendaron comprar en **La Ferretería.**

En función de los requisitos del usuario, en la Figura 2.1 se muestra el diagrama de un esquema ER para la base de datos **La Ferretería**. A partir de este modelo se irán gradualmente explicando los conceptos del modelo ER.

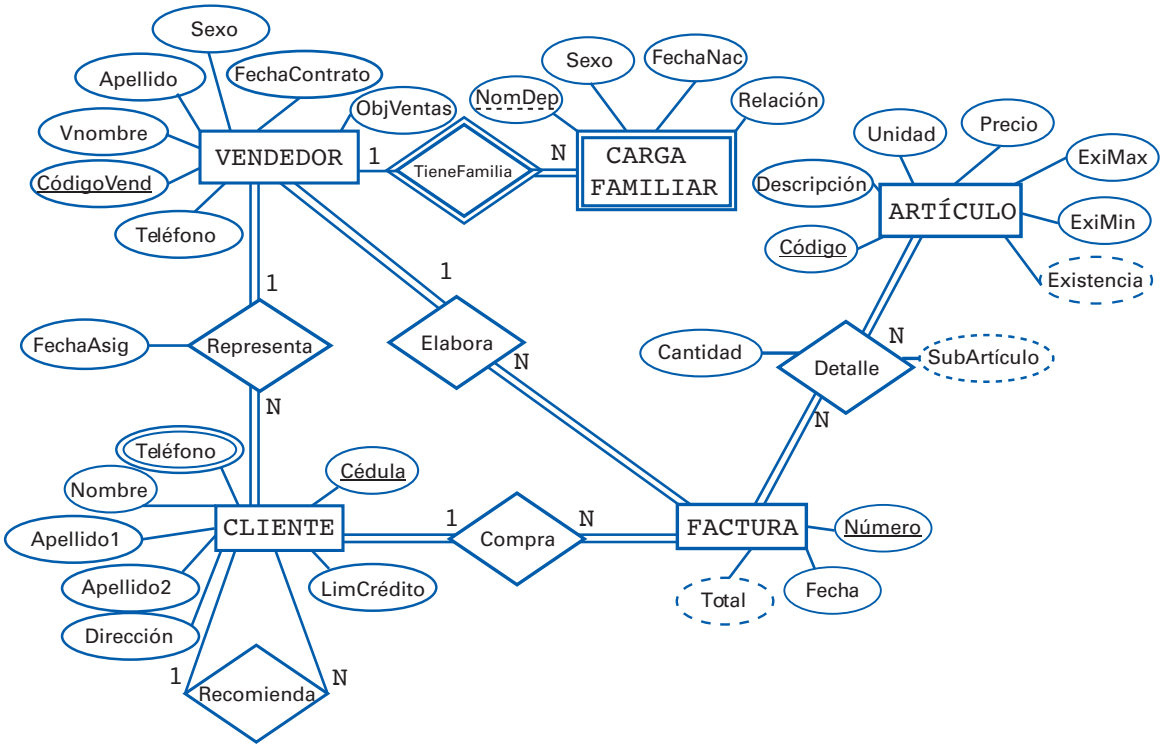


Figura 2.1. Diagrama ER para el caso del mundo real **La Ferretería.**

2.2 ELEMENTOS DEL MODELO ENTIDAD – RELACIÓN

El modelo Entidad – Relación describe el mundo real mediante un conjunto de conceptos que representan los datos de manera gráfica y lingüística. En un principio se trataba únicamente los conceptos de entidades, relaciones y atributos, pero luego, con las nuevas aplicaciones de la tecnología de bases de datos, los sistemas de información geográfica, las telecomunicaciones, los sistemas complejos de software, se han generado requisitos más complejos que los necesarios para las aplicaciones tradicionales, incluyéndose los conceptos de clases, subclases, especialización, agregación. El modelo (ER) incluido estos nuevos conceptos semánticos recibe el nombre de Modelo Entidad Mejorado (EER por sus siglas en inglés, *Enhanced* ER).

2.2.1 Entidades

Una entidad es cualquier objeto o concepto que puede identificarse en el mundo real, debe ser un objeto que existe y es distinguible de otros objetos. Desde el punto de vista de la existencia,

una entidad puede tener una existencia física (real), por ejemplo: vehículo, cliente, artículo, estudiante o una existencia conceptual (abstracta), por ejemplo: empresa, facultad, materia.

Cada objeto en particular que está representado unívocamente dentro de una entidad recibe el nombre de **instancia de una entidad** (Connolly T., Begg C., 2005).

En nuestro caso de estudio se determina la entidad `CLIENTE` con sus posibles propiedades: nombre, apellidos, cédula de ciudadanía, dirección y su límite de crédito, que se analizarán posteriormente cuando se trate el tema de los atributos. Una instancia de esta entidad es un cliente en particular, por ejemplo, un cliente de nombre Víctor y apellido Castro que está representado unívocamente por su cédula de ciudadanía 0102030405.

Una entidad en el esquema conceptual debe aparecer solo una vez y se representa mediante un rectángulo con el nombre en singular en su interior. En nuestro caso de estudio se definen cinco entidades que se describen en la Figura 2.2.

Figura 2.2. Representación gráfica de las entidades del caso **La Ferretería.**

2.2.2 Atributos

Los atributos son propiedades que describen características que posee cada instancia de una entidad. Por ejemplo, la entidad `VENDEDOR` podría estar representada por los atributos *CódigoVend* (código del vendedor) que para nuestro caso correspondería al identificador unívoco, *Vnombre* (nombre del vendedor), *Apellido* (apellido del vendedor), *FechaContrato* (fecha en la que fue contratado), *Sexo* (sexo del vendedor), *Objventas* (objetivo de ventas) y *Teléfono* (número telefónico).

Es importante resaltar que los atributos de cada entidad se determinan desde el punto de vista de las necesidades del usuario, siendo él quien describe la problemática a ser analizada, y no desde una óptica general que abarque todas las características posibles. Así por ejemplo, para la entidad `CARGAFAMILIAR`, varias son las características que pueden ser consideradas como atributos: fecha de nacimiento, estatura, peso, primer nombre, segundo nombre, sexo, relación o parentesco, edad, historia clínica, entre otros, sin embargo, para el modelado de datos del caso de **La Ferretería** se definen únicamente los atributos nombre, relación, sexo y fecha de nacimiento. Gráficamente los atributos se representan mediante un óvalo.

2.2.2.1 Tipos de atributos

En el modelo Entidad – Relación los atributos pueden clasificarse como: simples o compuestos, simple valor o multi-valor, derivados o almacenados y atributo clave:

- **Atributo simple:** recibe también el nombre de atómicos y se caracterizan por no ser divisibles en partes más pequeñas, por ejemplo, en la entidad VENDEDOR los atributos simples que le caracterizan a cada instancia de la entidad son: *Nombre, Apellido, Sexo, ObjVentas, FechaContrato, Teléfono*.

- **Compuestos:** un atributo es compuesto cuando es dividido en pequeñas subpartes, es decir, en otros atributos. Los atributos compuestos pueden formar una jerarquía de atributos. Por ejemplo, en la entidad `CLIENTE`, el atributo *Dirección*, puede ser modelado como un atributo compuesto, dividiéndole en componentes más pequeñas: *Ciudad*, *Parroquia*, *Barrio*, *Calle*. Adicionalmente al atributo *Calle* es posible dividirle en subpartes: *Nombre*, *Número* y *NroDep*, como se muestra en la Figura 2.3.

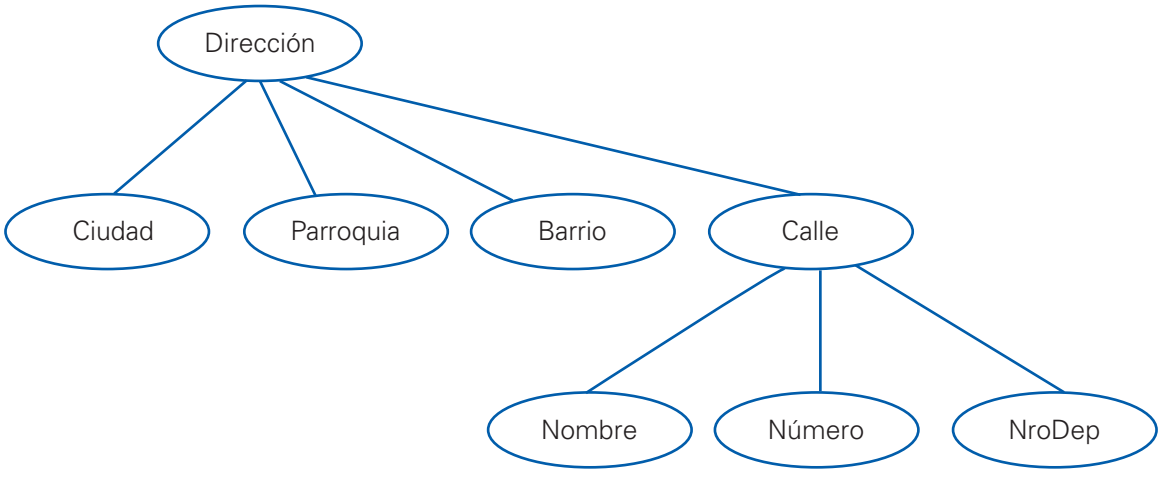


Figura 2.3. Atributos compuestos.

El nombre de una persona es otro ejemplo de atributo compuesto. Es posible dividirlo en cuatro subpartes: *Nombre1*, *Nombre2, Apellido1 y Apellido2.*

Al momento de diseñar una base de datos, la decisión para definir un valor como simple o compuesto depende de la necesidad del dato. Por ejemplo, si se quiere referir al atributo *Nombre* de una persona como una sola unidad o diferenciarlo mediante sus componentes individuales.

- **Atributo con simple valor**: son aquellos atributos que contienen un solo valor para cada instancia de una entidad. Por ejemplo, una persona tiene un único valor para su edad y la edad es un simple valor de la persona. Estos atributos también reciben el nombre de monovalorados o univaluados.

  En nuestro caso de estudio, la entidad `CARGAFAMILIAR` tiene cuatro atributos con simple valor: *NomDep, Sexo, FechaNac* y *Relación* y los valores para una instancia pueden ser: "María", "F", "27-Oct-04" e "Hija" respectivamente.

- **Atributo multivalor:** cuando un atributo tiene multiples valores para identificarse, por ejemplo, cada instancia de la entidad `Persona`, puede tener varias profesiones. Para representar un atributo multivalor se utiliza un doble óvalo, como se muestra en la Figura 2.4.

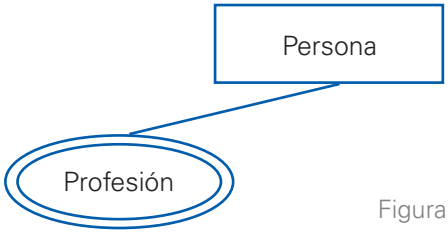


Figura 2.4. Representación de un atributo multivalor.

  En la entidad `CLIENTE`, el atributo *Teléfono*, está modelado como atributo multivalor, puesto que un cliente puede tener asignado uno o más número telefónicos. Es posible, cuando el caso lo amerite, definir límites inferior y superior en el número de valores asignados a un atributo multivalor, por ejemplo, si **La Ferretería** decide limitar que los clientes deben tener entre cero y dos números telefónicos.

- **Atributos derivados y almacenados:** se denominan atributos derivados, cuando los valores de un atributo están relacionados. Si se conoce el valor de uno o varios atributos, a partir de éstos se puede obtener el valor de otro atributo que esté relacionado. Por ejemplo, existe relación entre la edad y la fecha de nacimiento de una persona; si se conoce la fecha de nacimiento, se puede determinar su edad, en este caso se dice que la edad es derivada de la fecha de nacimiento. Al atributo *Edad* se lo denomina **atributo derivado** que no es almacenado sino es calculado cuando sea necesario, y al atributo *FechaNacimiento* se lo llama **atributo almacenado.** (Elmasri R., Navathe S., 2016).

En el caso de **La Ferretería** el atributo Total de la entidad FACTURA es un ejemplo de atributo derivado, éste se calcula sumando cada uno de los subtotales obtenidos al multiplicar el precio del artículo por la cantidad vendida en la factura. Los atributos *Precio y Cantidad* son atributos almacenados.

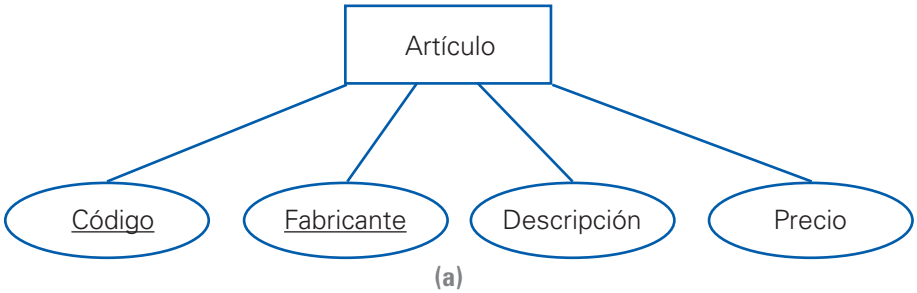
Otro ejemplo se presenta si se considera el atributo *Tiempo-de-servicio*, que indicaría el tiempo en años que un vendedor presta sus servicios. Este atributo se deriva a partir de la diferencia entre la *FechaContrato* que expresa la fecha en la que el vendedor comenzó a trabajar en La Ferretería y la fecha actual.

El valor de un atributo derivado se calculará cuando éste se requiera, es por esta razón que no es necesario almacenarlo. Estos atributos en el diagrama ER se representan mediante un óvalo con línea entrecortada.

- **Atributo clave:** una entidad tiene un atributo cuyos valores son distintos y nos permite identificar de manera unívoca cada instancia de una entidad. Por ejemplo, en la entidad CLIENTE, el atributo clave es la cédula de ciudadanía, *Cédula* que nos permite diferenciar a cada uno de los clientes, garantizando que no existirán dos con el mismo número de cédula. Para la entidad FACTURA, el atributo clave es Número, para la entidad VENDEDOR su clave es *CódigoVend*. (ver Figura 2.1).

  En el diagrama Entidad – Relación un atributo clave se representa con el nombre subrayado dentro del óvalo

  En algunos casos varios atributos simples en conjunto forman una clave, que se la denomina **clave compuesta**. La combinación de los valores de estos atributos simples tomados en conjunto, debe ser distinta para cada instancia de la entidad. Por ejemplo, si en la entidad ARTÍCULO, a cada artículo se quiere identificarlo de manera única con su código y el dato del fabricante que lo elaboró, la clave compuesta se forma del par de atributos *Código y Fabricante*, combinación que debe ser distinta para cada instancia de la entidad. En la Figura 2.5 (a) se muestra el diagrama de una clave compuesta y en la Figura 2.5 (b) se describen 7 instancias de la entidad ARTÍCULO, identificadas con una clave compuesta.



(a)

ARTÍCULO

Código	Fabricante	Descripción	Precio
1	Acero S.A	Perfil T	6,00
1	ANE	Perfil T	6,50
5	Acero S.A	Perfil L	7,20
5	ANE	Perfil L	7,80
5	FabAcero	Perfil L	8,00
8	Acero S.A	Perfil G	9,50
8	ANE	Perfil G	9,70

b

Figura 2.5. (a) Clave compuesta. (b) Instancias de la entidad ARTÍCULO con clave compuesta.

Atributos con valor nulo: "representa un valor para un atributo que es actualmente desconocido o no es aplicable" (Connolly T., Begg C., 2005). Un valor nulo significa ausencia de valor, contexto que resulta contradictorio, es por esta razón que algunos autores prefieren utilizar el término "nulo" en remplazo de "valor nulo".

Un valor nulo tiene la categoría de "no aplicable" cuando una instancia de entidad no tenga un valor aplicable para un atributo. Por ejemplo, en el atributo *LímCrédito* de la entidad CLIENTE, en ciertas instancias se van a registrar valores nulos, considerando que no todos los clientes serán beneficiarios de un crédito por La Ferretería, como se muestra en la Figura 2.6. (a)

Un valor nulo tiene la categoría de "desconocido", cuando no se conoce el valor de un atributo. Esta categoría se clasifica en dos: El primero "no conocido" cuando no se conoce si el valor existe. Por ejemplo, si no se conoce el número de teléfono de un vendedor. El segundo caso "no se encuentra" o "perdido", cuando el valor existe pero no se tiene la información. Por ejemplo, el peso de una persona, se sabe que existe, pero no se conoce el valor. En la Figura 2.6(b) se muestra esta categoría de valor nulo.

Es necesario diferenciar entre una cadena de texto con espacios en blanco y un valor numérico cero con un valor nulo, éste último representa ausencia de valor.

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito
Víctor	Castro	Torres	0102030405	Sucre 1-12	600
Juan	Polo	Ávila	0122334455	Bolívar 5-67	1000
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500
Pablo	Brito	Parra	0123456789	Alfaro 1-23	Null
Marcia	Mora	Durazno	0987654321	Olmedo 1-10	1000

(a)

VENDEDOR

Nombre	Apellido	CódigoVend	Sexo	Peso	ObjVentas	Teléfono
Diego	Loja	1	M	60	3000	2472713
Antonio	Calle	2	M	65	2500	2876549
Lucía	Serrano	3	F	58	2000	Null
Arturo	Salto	4	M	Null	4000	2887643

(b)

Figura 2.6. (a) Valor nulo no aplicable. (b) Valor nulo desconocido.

2.2.2.2 Dominio de un atributo

Cada atributo de una entidad está asociado con un conjunto de valores permitidos, llamado **dominio**. En otros términos, dominio de un atributo es el conjunto de posibles valores que puede tomar. Por ejemplo, en la entidad `CLIENTE` el dominio del atributo *LímCrédito* se restringe a valores mayores a 600 y menores a 2000; para el atributo *Nombre* el dominio está compuesto de una cadena de caracteres de una determinada longitud.

Para registrar los datos de los puntos cardinales el dominio del atributo está definido por los siguientes valores permitidos: "norte", "sur", "este" y "oeste".

2.2.3 Relaciones

Una **relación** es cualquier asociación o correspondencia del mundo real que pueda establecerse entre entidades. A cada relación se le asigna un nombre, normalmente un verbo o una frase corta que incluya un verbo, que describe su función (Connolly T., Begg C., 2005, pág. 317). Gráficamente las relaciones se representan a través de un rombo con el nombre en su interior, conectado mediante líneas a cada entidad participante.

Recibe el nombre de **instancia de relación** cada combinación de las instancias de las entidades relacionadas que constituyen una ocurrencia en la relación.

Un ejemplo de relación se muestra en la Figura. 2.7(a), en la que se asocia las entidades `PERSONA` y `VEHÍCULO` generando la relación `CONDUCE`. En la Figura 2.7(b), se muestra tres instancias de la relación `CONDUCE`.

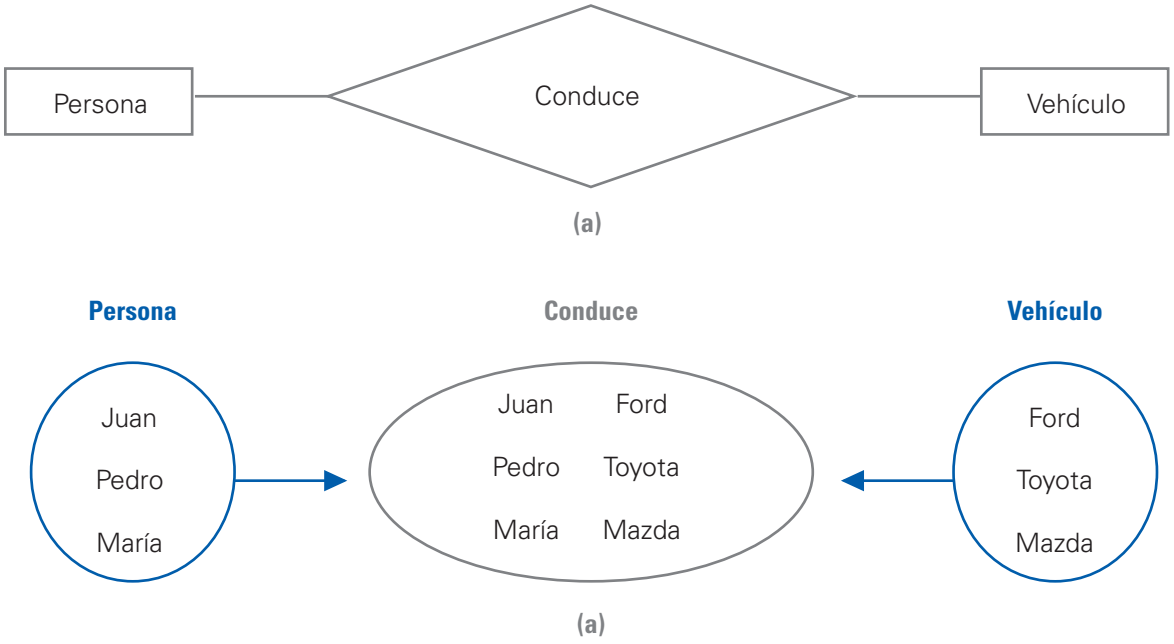


Figura 2.7. (a) Representación gráfica de una relación. (b) Tres instancias de una relación.

Una relación puede incluir varias entidades. Se llama **grado** de una relación al número de entidades que participan en la relación.

2.2.3.1 Tipos de relaciones

Dependiendo del número de entidades que intervengan en la relación, éstas pueden clasificarse en:

- **Binarias:** cuando la relación se establece entre dos entidades diferentes. En un sistema de bases de datos, estas relaciones son las más frecuentes. La relación `CONDUCE` que se analizó en los párrafos anteriores es un ejemplo de relación binaria. Otros ejemplos de relaciones de grado dos que corresponden a nuestro caso de estudio, se muestran en la Figura 2.8, que representan la relación `REALIZA` en la que participan las entidades `VENDEDOR` y `FACTURA`; y la relación `DETALLE`, formada por la asociación de las entidades, `ARTÍCULO` y `FACTURA`.

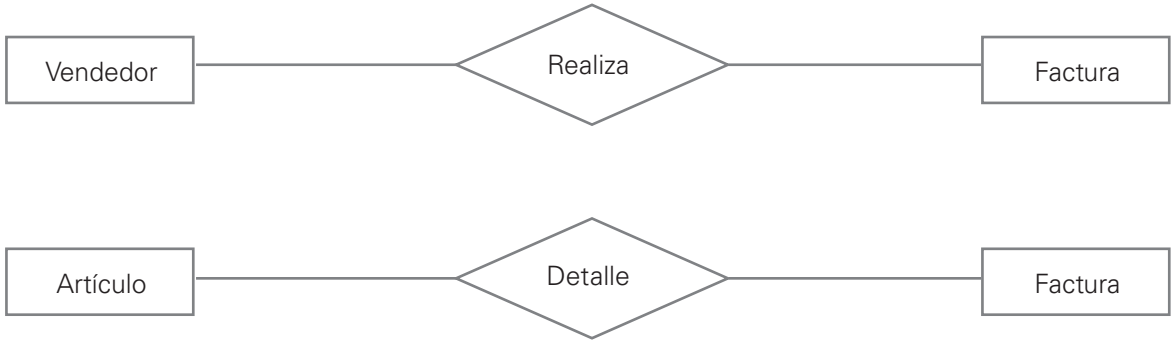


Figura 2.8. Ejemplos de relaciones binarias o de grado dos.

- **Relaciones unitarias (Recursivas):** se establece entre entidades de la misma clase. Por ejemplo, las personas son hijos de personas; un empelado supervisa a otros empleados. Para nuestro caso de estudio se tiene la relación recursiva `RECOMIENDA`, que representa los clientes que recomendaron a otros clientes. Estos ejemplos de recursividad se muestran en la Figura 2.9

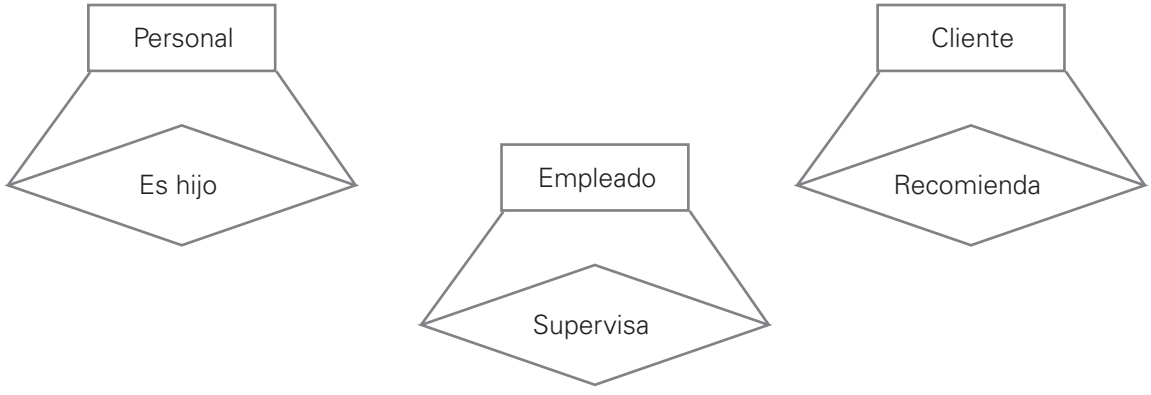


Figura 2.9. Ejemplos de relación recursiva.

Relación N-arias: son relación que se establecen entre N entidades, siendo N > 2. En la práctica estas relaciones suele ser sustituida por relaciones binarias. "La mayor parte de las relaciones de los sistemas de bases de datos son binarias" (Silberschatz , A. Korth, H. Sudarshan, S., 2014). La relación entre las entidades `PADRE, MADRE` e `HIJO`, es un ejemplo de relación ternaria o de grado 3, como se muestra en la Figura 2.10.

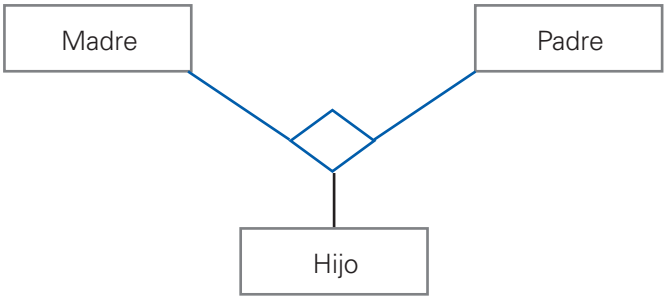


Figura 2.10. Ejemplos de relación ternaria.

2.2.3.2 Propiedades de una relación

Se denomina propiedades de una relación a las restricciones que se asignan a las entidades que participan en la relación y que reflejan las limitaciones que se presentan en el mundo real. Por ejemplo, en el caso particular de La Ferretería, una limitación representa el hecho de que un cliente tiene exactamente a un vendedor como representante, situación que se debe reflejarla como restricción en el modelo.

Se analizarán tres tipos de restricciones: el rol de la relación, la correspondencia de cardinalidad y la participación.

Roles de una relación

Son las funciones que desempeñan cada una de las instancias de las entidades asociadas. En toda relación binaria existen dos roles diferentes correspondientes a las entidades participantes. Por ejemplo, en la relación `CONDUCE` la entidad `PERSONA` cumple un rol de "*conduce el*" en tanto que "*es conducido*" corresponde al rol de la entidad `VEHÍCULO`. Gráficamente el rol de la relación se escribe sobre la línea de la relación de las entidades, como se indica en la Figura. 2.11.

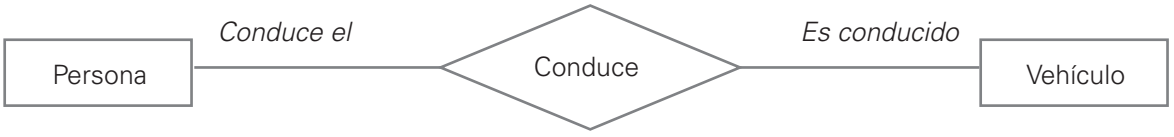


Figura 2.11. Rol de una relación.

Para ilustrar este concepto se supone que A representa la entidad formada por todos los profesores de un centro de estudios universitario, y B la formada por todos los alumnos de dicho centro; entre las instancias de estas dos entidades se puede establecer varias tipos de relaciones caracterizadas por los siguientes roles:

- Imparte clase a / Recibe clase de
- Es tutor de / Es dirigido por
- Es director de / Es dirigido por

Para el caso de la relación recursiva RECOMIENDA de nuestro caso **La Ferretería**, es necesario establecer el rol o papel que cada entidad juega en cada instancia de relación. La entidad CLIENTE tiene dos roles en la relación, uno de quien "*recomienda a*" un cliente y el otro, de quien "es *recomendado*" por un cliente, como se muestra en la Figura 2.12.

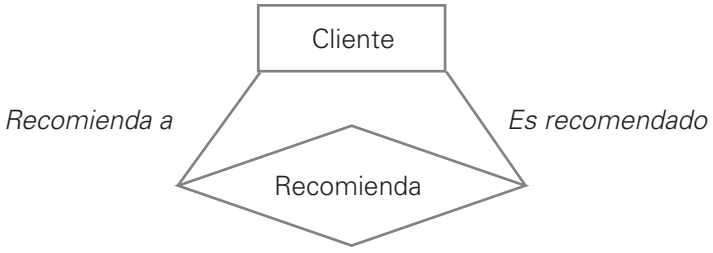


Figura 2.12 Roles de una relación.

Cardinalidad

En una relación binaria expresa, "*el número de posibles instancias de una entidad que pueden relacionarse con una única instancia de otra entidad asociada a través de una relación concreta*" (Connolly T., Begg C., 2005, pág. 325). Sobre la base de esta propiedad se determinan tres tipos de relaciones con cardinalidad uno a uno (1:1), uno a muchos (1:N) o muchos a muchos (N:M).

A continuación se analizan estos conceptos, partiendo de la relación binaria CONDUCE, que representa la asociación de las entidades PERSONA y VEHÍCULO:

Uno a uno.- En el mundo real es posible suponer que una persona conduce un vehículo y un vehículo es conducido por una sola persona. La representación de esta regla como una restricción de cardinalidad en el modelo Entidad - Relación se ilustra en la Figura.2.13(a), que contiene cuatro instancias de la relación CONDUCE representadas por la letra "c", forma-

da por una instancia de la entidad PERSONA, asociada con una única instancia de la entidad VEHÍCULO. En la Figura 2.13(b) se muestra el diagrama de cardinalidad 1:1.

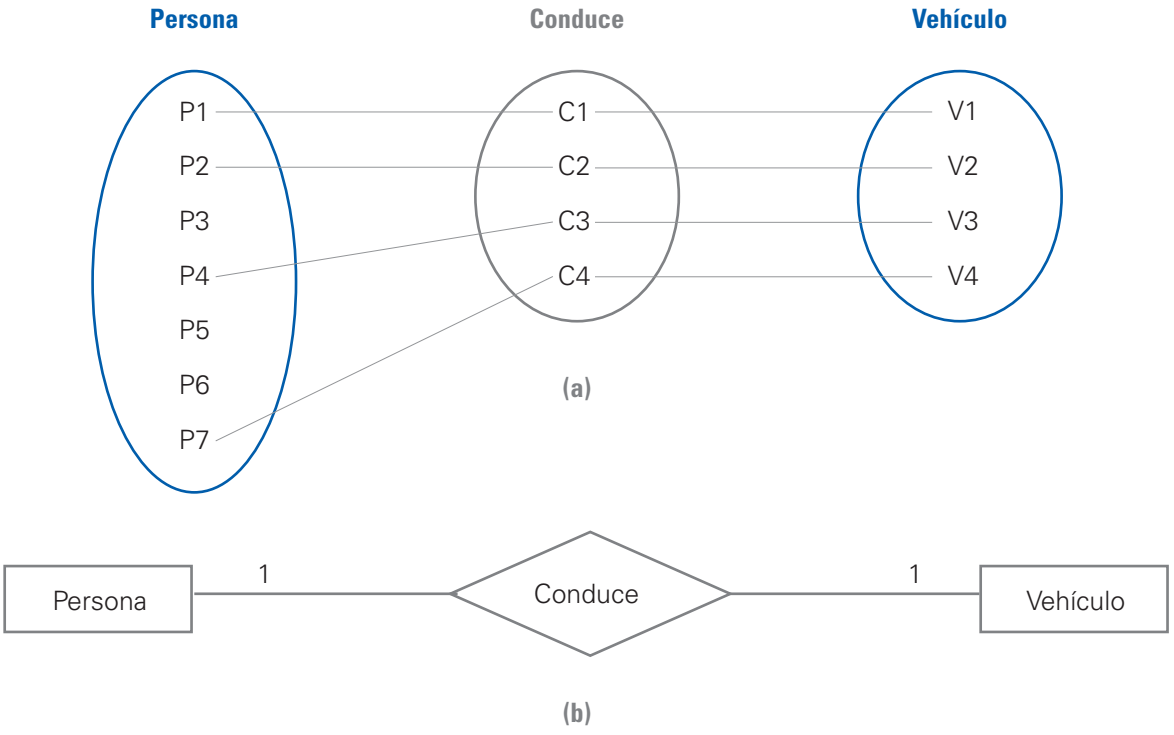
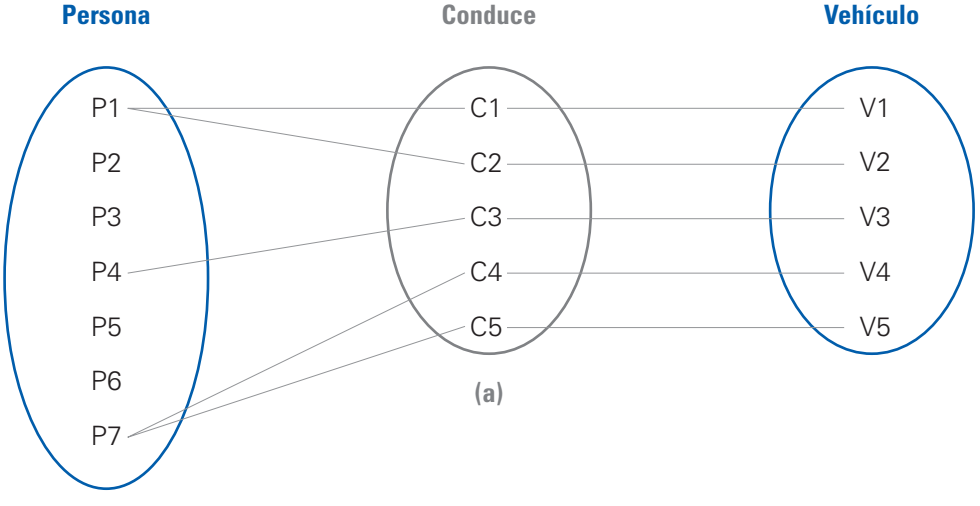
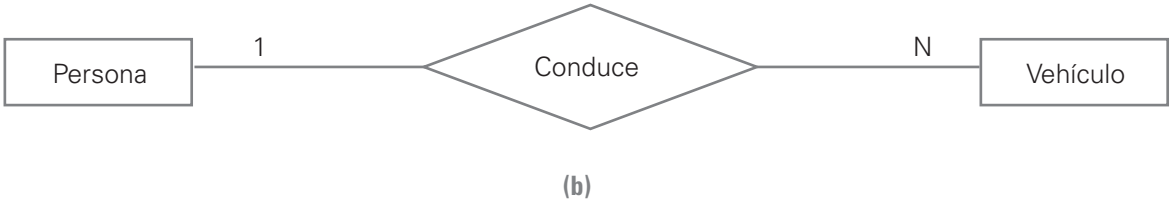


Figura 2.13. (a) Instancias de una relación 1:1. (b) Relación con cardinalidad 1:1.

- **Uno a muchos.-** Para analizar la cardinalidad uno a muchos se supone que una persona puede manejar varios vehículos y que un vehículo es manejado por una sola persona. La Figura 2.14(a) describe cinco instancias de la relación CONDUCE, que están formadas por una instancia de la entidad PERSONA asociada con una o varias instancias de la entidad VEHÍCULO. En la Figura 2.14(b) se muestra el diagrama de cardinalidad 1:N.

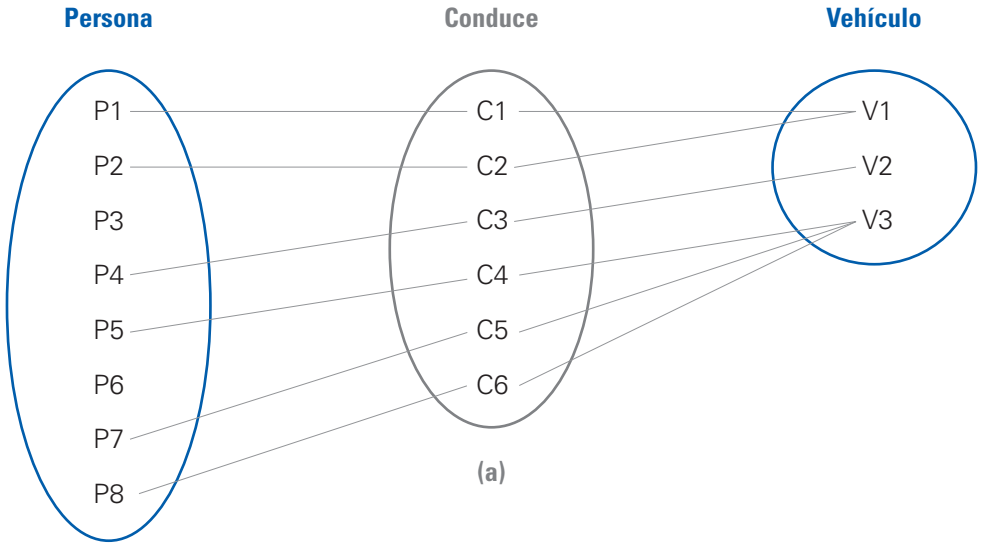




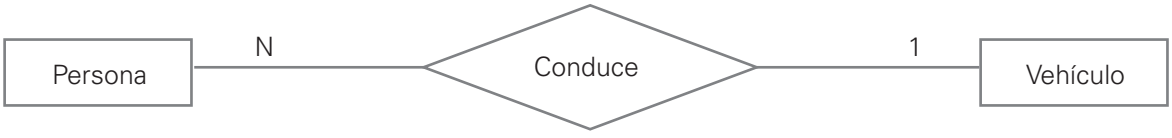
(b)

Figura 2.14. (a) Instancias de una relación 1:N. (b) Relación con cardinalidad 1:N

El concepto es similar cuando se analiza la cardinalidad **muchos a uno**. Para este caso se considera que una persona conduce un vehículo y un vehículo es conducido por varias personas. Por ejemplo, la Figura 2.15(a) muestra seis instancias de la relación `CONDUCE`, definidas por la asociación de una única instancia de la entidad `VEHÍCULO` con una o varias instancias de la entidad `PERSONA`. La Figura 2.15(b) muestra el diagrama con cardinalidad N:1.



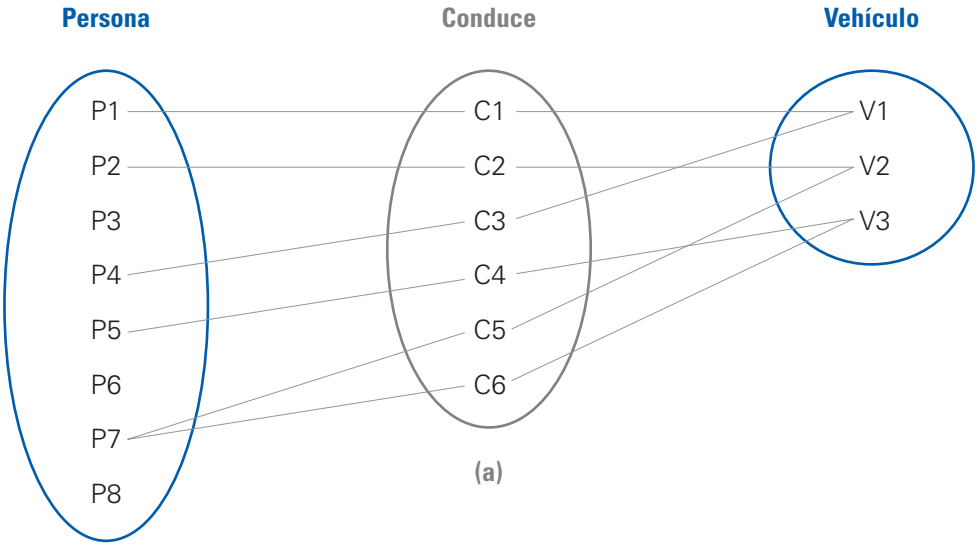
(a)



(b)

Figura 2.15. (a) Instancias de una relación N:1. (b) Relación con cardinalidad N:1.

Muchos a muchos.- Para la cardinalidad N:N se supone que una persona puede conducir varios vehículos y que un vehículo es conducido por varias personas. En la Figura 2.16(a) se representa seis instancias de la relación `CONDUCE`, formadas por la asociación entre una instancia de la entidad `PERSONA` con una o varias instancias de la entidad `VEHÍCULO` y una instancia de la entidad `VEHÍCULO` asociada con varias instancias de la entidad `PERSONA`. Como en los casos anteriores, la Figura 2.16(b) muestra el diagrama con cardinalidad N:N



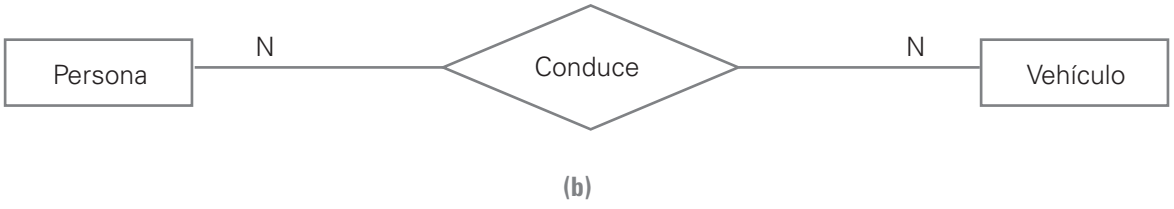


Figura 2.16. (a) Instancias de una relación N:N. (b) Relación con cardinalidad N:N.

De manera práctica, para determinar la cardinalidad de una relación, es posible mediante el planteamiento de dos preguntas basadas en el rol que desempeña cada entidad en la relación, las respuestas corresponderán a la cardinalidad buscada. Esta práctica se ilustra con el siguiente ejemplo. En el caso de estudio de **La Ferretería**, la relación `REPRESENTA` (ver Figura 2.17), que asocia las entidades `CLIENTE` y `VENDEDOR` tiene una cardinalidad de N:1, esto significa que cada vendedor puede representar a varios clientes, pero un cliente puede ser representado por un solo vendedor. Esta representación del mundo real se determina con el siguiente análisis:

Pregunta 1. ¿Un cliente es *representado* por cuántos vendedores?, la respuesta es "por un vendedor", definiéndose como cardinalidad uno (1).

Pregunta 2. ¿Un vendedor *representa* a cuántos clientes?, la respuesta es "a varios clientes", determinándose como cardinalidad muchos (N).

Figura 2.17. Relación Representa con cardinalidad N:1.

A continuación se analiza un segundo ejemplo, a partir de la relación **ELABORA** de la Figura 2.1 que asocia las entidades **VENDEDOR** y **FACTURA**. Las restricciones del mundo real especifican que un vendedor realiza varias facturas, pero una factura es realizada únicamente por un vendedor. Las preguntas para determinar la cardinalidad se definen de la siguiente manera:

Pregunta 1: ¿Un vendedor cuántas facturas realiza?, la respuesta es varias facturas, determinándose como muchos (N).

Pregunta 2: ¿Una factura es realizada por cuántos vendedores?, la respuesta es por un solo vendedor, definiéndose con uno (1). Como resultado de estas dos preguntas se establece la cardinalidad 1:N para la relación **ELABORA.**

Participación y dependencia de existencia

La propiedad de participación, llamada también restricción de participación, especifica si todas las instancias de una entidad se relacionan necesariamente con instancias de otra entidad. En una relación se presentan dos restricciones de participación: *total* si todas las instancias de una entidad participan en la relación; y *parcia*l, si sólo participan algunas. La participación total se conoce también como **dependencia de existencia** que se analizará más delante.

Por ejemplo, en la relación **CONDUCE** se define que todos los vehículos serán conducidos por al menos una persona, es decir, todas las instancias de la entidad **VEHÍCULO** participan en la relación, sin embargo, no toda las personas tendrán asignado un vehículo, lo que significa que no

todas las instancias de la entidad PERSONA participan en la relación. Para este caso, las entidades VEHÍCULO y PERSONA tendrán una participación *total y parcial* respectivamente.

Este ejemplo se ilustra en la Figura 2.18(a). De la entidad PERSONA únicamente participan en la relación las instancias p1, p2, p4, p5 y p6, de tal manera que la participación de esta entidad en la relación CONDUCE es parcial, sin embargo, de la entidad VEHÍCULO participan en la relación todos los vehículos registrados v1, v2 y v3, de modo que la participación de esta entidad en la relación CONDUCE es *total.*

Gráficamente en el modelo, la relación con participación total se representa con doble línea y la parcial con una línea, como muestra la Figura 2.18(b).

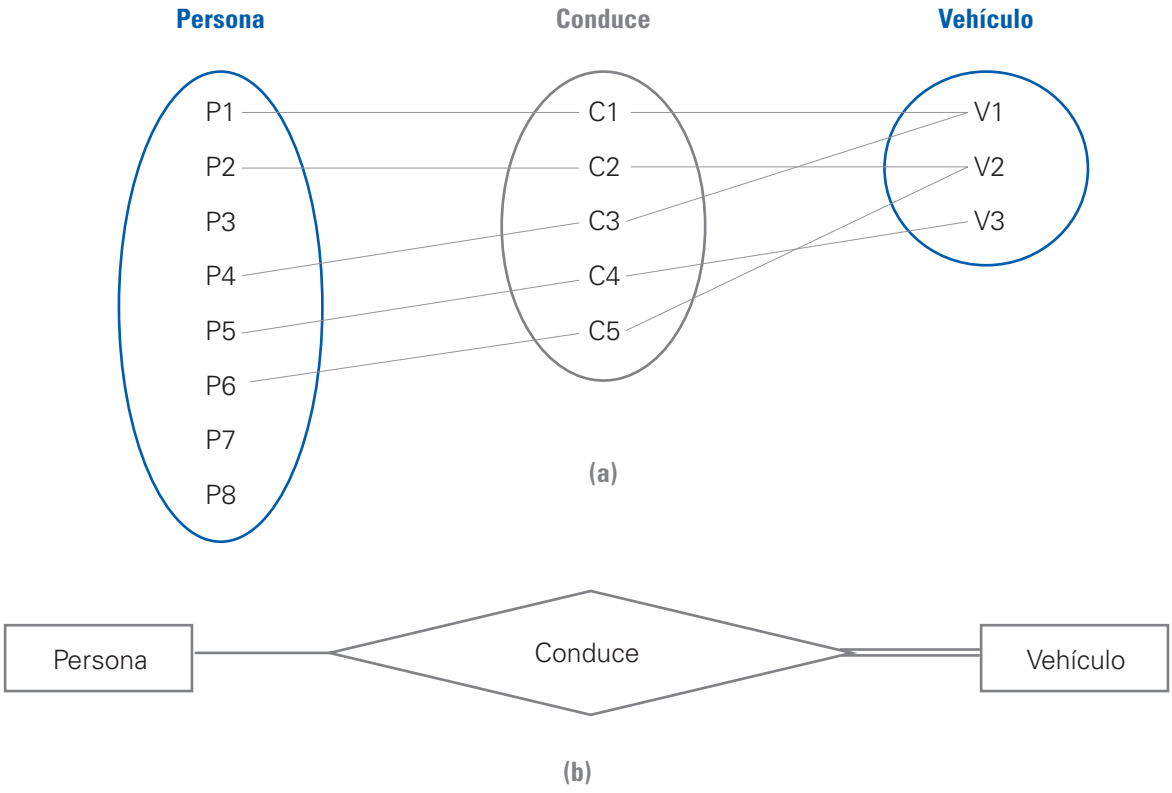


Figura 2.18. (a) Instancias de participación de la relación CONDUCE. (b) Relación con participación parcial para PERSONA y total para VEHÍCULO.

En la práctica, para determinar la restricción de participación, se analiza a través de un proceso análogo al utilizado para definir la cardinalidad, respondiendo las preguntas de acuerdo a las restricciones del mundo real. Por ejemplo, si como política el propietario de **La Ferretería** decide que cada cliente debe tener asignado un vendedor que lo represente, pero que no todos los vendedores serán considerados como representantes. El análisis de la participación se describe de la siguiente manera, y su resultado se muestra en la Figura 2.19.

Pregunta 1: ¿Todos los clientes tienen un representante?, si la respuesta es afirmativa como en este caso, la participación es total (o dependencia existente).

Pregunta 2: ¿Todos los vendedores representan a clientes?, en este caso la respuesta es negativa, entonces la participación es parcial.

Figura 2.19. Relación con participación total para cliente y parcial para vendedor.

A la cardinalidad y a las restricciones de participación en conjunto se las denomina **restricciones estructurales** de una relación. (Elmasri R., Navathe S., 2007, pág. 65)

2.3 NOTACIÓN ALTERNATIVA PARA LA CARDINALIDAD Y PARTICIPACIÓN

Esta notación implica asociar un par de números enteros mínimo y máximo (mín, máx) a cada una de las entidades que forman parte de la relación. Los valores mínimos a ambos lados de la relación corresponden a la restricción de participación, el valor 0 (cero) representa la participación *parcial* y el valor 1 (uno) la *total*. En tanto que la cardinalidad (1:1, 1:N y N:N) de la relación se representa con el valor máximo.

Si se considera la relación `REPRESENTA` entre las entidades `VENDEDOR` y `CLIENTE`. En la Figura 2.20(a) se muestra que el vendedor v1 representa al cliente c5 y c7, el vendedor v4 no tiene representando y el vendedor v8 representa únicamente al cliente c12. En este ejemplo, al no formar parte de la relación todos los vendedores, su participación es *parcial* representada por el valor mínimo de 0, sin embargo, para el caso de los clientes, todos forman parte de la relación, su participación es *total* y se representa con el valor mínimo de 1. Es importante considerar que la participación de una entidad en una relación, está representada por el valor mínimo en el lado opuesto de la relación. (Connolly T., Begg C., 2005, pág. 331)

Por otra parte, la relación posee una cardinalidad de 1:N, porque la norma del mundo real determina que un vendedor representa a varios clientes y un cliente es representado por un vendedor. En la Figura 2.20(b) se ilustra la cardinalidad y la participación de la relación `REPRESENTA`.

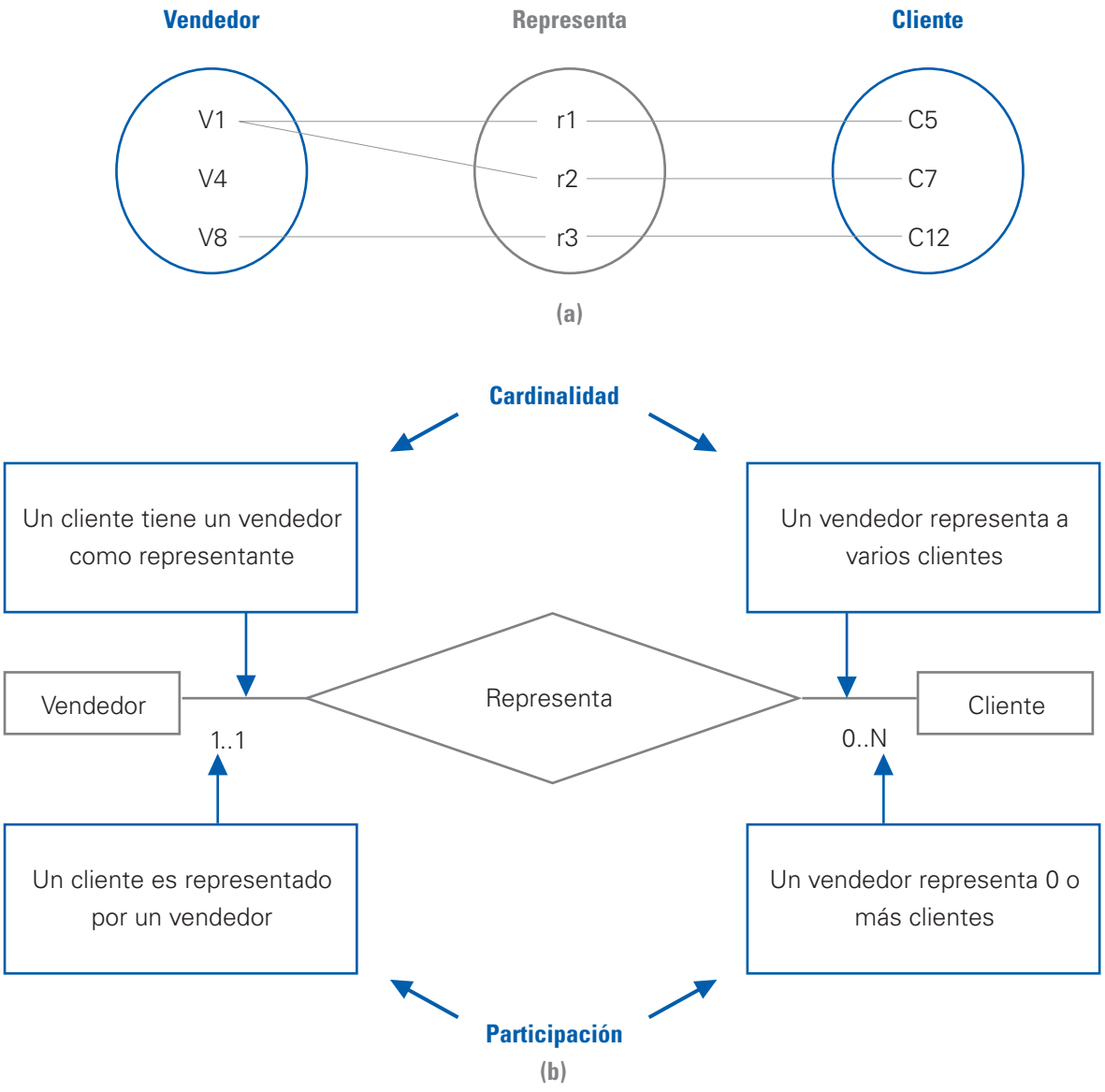


Figura 2.20. (a) Tres instancias de la relación REPRESENTA. (b) Cardinalidad y participación de la relación REPRESENTA

Para un segundo ejemplo de análisis se modifican las restricciones del mundo real, asumiendo que todos los vendedores deben tener asignados clientes para representar y que todos los clientes tengan un vendedor como representante, la participación para las dos entidades VENDEDOR y CLIENTE es *total* simbolizada con el valor mínimo de 1. La cardinalidad no se modifica y se mantiene 1:N. En la Figura 2.21(a), se detalla la representación gráfica de este ejemplo y en la Figura 2.21(b), se muestra el modelo ER con los valores de cardinalidad y participación.

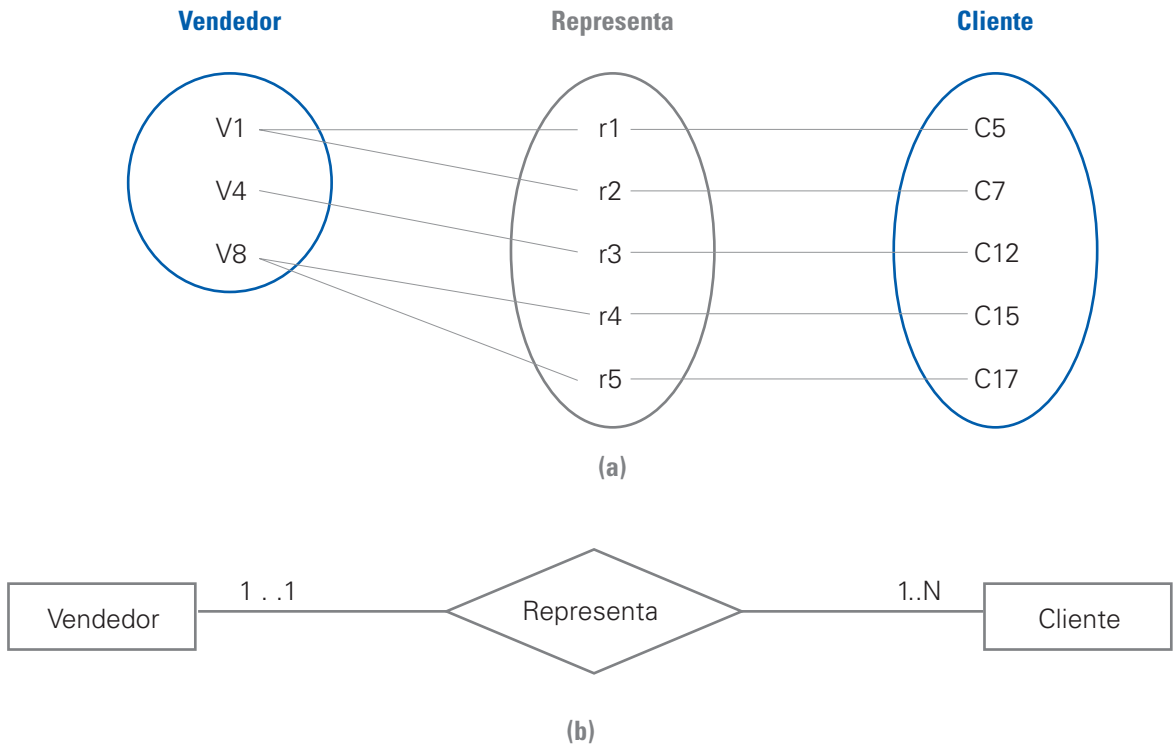


Figura 2.21. (a) Cinco instancias de la relación REPRESENTA. (b) Participación total – total con cardinalidad 1:N.

Para un tercer ejemplo se asume que a todos los vendedores se les asigna uno o más clientes para representar (participación total), pero que no todos los clientes tienen asignado a un vendedor como representante (participación parcial). En la Figura 2.22(a), se observa que el vendedor v1 representa a los clientes c5 y c7 y que el vendedor v4 representa al cliente c15, sin embargo, el cliente c12 no es representado por ningún vendedor. Como en los casos anteriores la cardinalidad se mantiene.

La Figura 2.22(b) ilustra el diagrama ER de la relación REPRESENTA. Para simbolizar que un vendedor representa a *uno o más* clientes se coloca la notación ´1..N´ al lado de la entidad CLIENTE, y para especificar que un cliente es representado por *cero* o *un* vendedor, se coloca la notación ´0..1´ al lado de la entidad VENDEDOR.

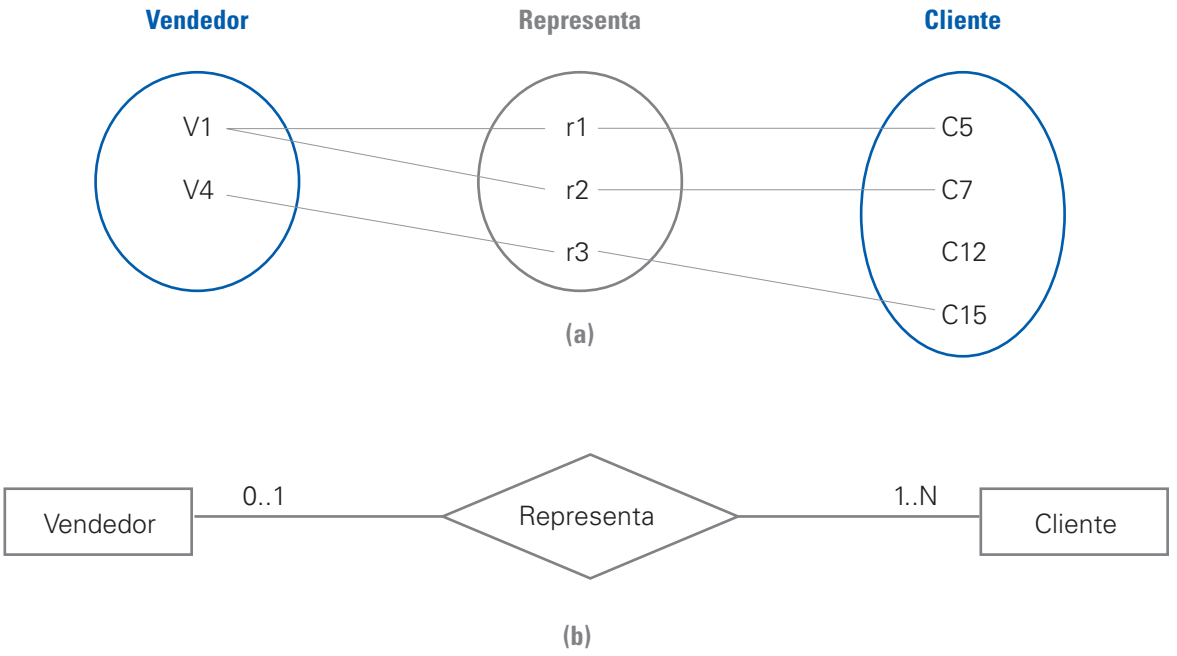


Figura 2.22. (a) Tres instancias de la relación REPRESENTA.
(b) Participación total – parcial con cardinalidad 1:N.

2.4 DEPENDENCIA DE EXISTENCIA, ENTIDADES FUERTES Y ENTIDADES DÉBILES

Dependencia de existencia. Si la existencia de la entidad x depende de la existencia de la entidad y, entonces se dice que x es *dependiente por existencia de y*. Operativamente significa que si se suprime *y*, también se suprime *x*. En este caso la entidad *y* es **dominante** o **fuerte** y *x* es **subordinada** o **débil**.

Una **entidad débil** se define como aquella cuya existencia depende de la existencia de otra entidad. Una característica de este tipo de entidades es no poseer atributo clave propio; esto significa que cada instancia no puede identificarse unívocamente en función de sus atributos, adicionalmente una entidad débil siempre tiene una participación total en la relación y gráficamente se representa mediante un rectángulo con doble línea.

Como ejemplo se analiza la relación TIENEFAMILIAR del caso de estudio **La Ferretería** que se muestra en la Figura 2.23. De acuerdo a las restricciones del mundo real no es una condición que los vendedores tengan cargas familiares (participación parcial), pero sí es necesario que una carga familiar esté vinculada con un vendedor (participación total). En términos del modelo ER esto significa que la existencia de la entidad VENDEDOR no depende de la existencia de la entidad CARGAFAMILIAR; por el contrario, la existencia de la entidad CARGAFAMILIAR depende de la existencia de la entidad VENDEDOR.

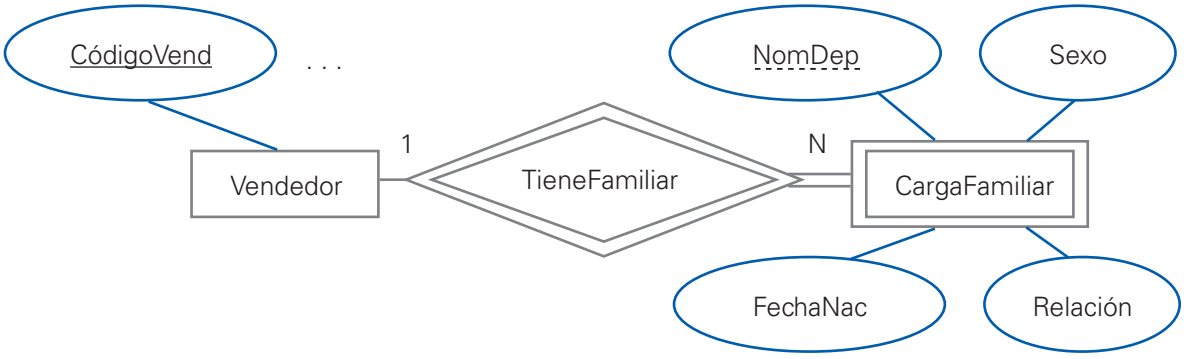


Figura 2.23. Dependencia de existencias.

Nótese que en la entidad `CARGAFAMILIAR` no se incluye un atributo clave, por lo que no es posible identificar de manera unívoca a las instancias de entidad en el caso de que dos cargas familiares por eventualidad tengan los mismos valores en sus atributos. En realidad, los datos hacen referencia a dos cargas familiares que se identifican como distintas a través de la relación con su vendedor asociado. Adicionalmente la relación `TIENEFAMILIAR`, que relaciona una entidad débil recibe el nombre de **relación identificativa** (Elmasri R., Navathe S., 2016) y se representa gráficamente con un rombo con doble línea.

Si bien una entidad débil no tiene atributo clave, no obstante, para identificar las instancias de entidad se requiere contar con un conjunto de atributos denominado **clave parcial** o **discriminante**. Gráficamente el atributo de clave parcial se representa subrayado con una línea discontinua.

En el ejemplo que se muestra en la Figura 2.23, se define el atributo *NomDep* como clave parcial, suponiendo que no existirán dos nombres repetidos de cargas familiares (entidad subordinada) que pertenezcan a un mismo vendedor (entidad dominante).

Es importante tener en cuenta que no toda dependencia de existencia genera una entidad débil. Por ejemplo, si se requiere facturar el consumo de energía eléctrica, la entidad `FACTURA` no puede existir si no está relacionada con una entidad fuerte `MEDIDOR`, sin embargo para identificar de manera única a cada una de las facturas, se requiere de un atributo clave que se lo puede denominar *NúmeroFactura*. En este contexto la entidad `FACTURA` pierde su condición de entidad débil.

Una **entidad fuerte** es aquella cuya existencia no depende de la existencia de otra entidad y se caracteriza por tener un atributo clave.

En la Figura 2.24 se ilustra otro ejemplo de dependencia de existencia. El modelo describe los movimientos realizados por una determinada cuenta corriente mediante la relación `BITÁCO–`

RA. Se presenta dependencia de existencia entre la entidad TRANSACCIÓN (subordinada) y la entidad CUENTA (dominante), puesto que para que exista una instancia de entidad TRANSACCIÓN, necesariamente debe haber una instancia de entidad CUENTA; además, si se elimina una instancia de entidad CUENTA, la o las instancias de entidad TRANSACCIÓN asociada también se eliminarán. Por otra parte, la entidad TRANSACCIÓN tiene asignado un atributo clave llamado Número, en consecuencia, no es una entidad débil.

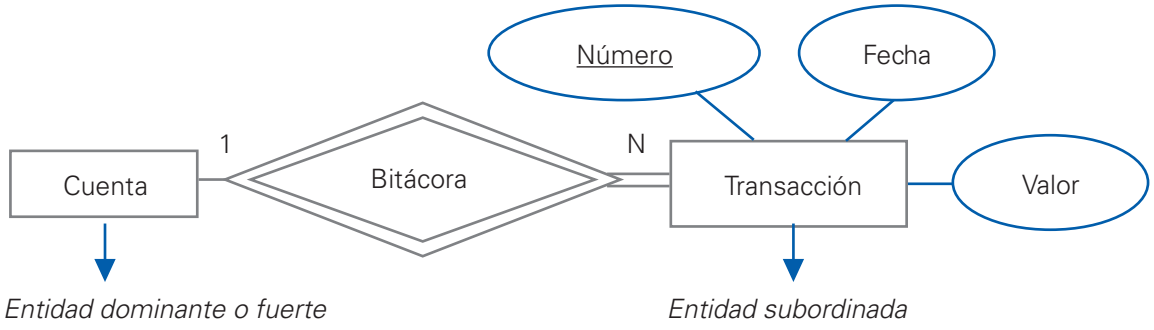
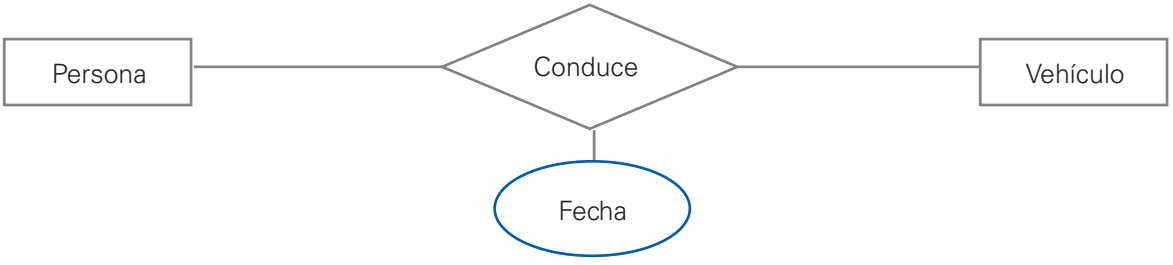


Figura 2.24. Dependencia de existencia sin entidad débil.

2.5 ATRIBUTOS DE UNA RELACIÓN

Una relación puede tener sus propios atributos similares a los que se manejan en las entidades. Por ejemplo, si se requiere registrar las fechas en las que un vehículo fue utilizado por una determinada persona, se incluye el atributo *Fecha* a la relación CONDUCE, como se muestra en la Figura 2.25(a). Los atributos de una relación se representan gráficamente mediante un óvalo, de manera análoga a los utilizados en una entidad.

En la Figura 2.25(b) se observa que a cada instancia de la relación CONDUCE se le asigna un valor del atributo, que corresponde a la fecha en la que una persona utilizó un determinado vehículo.



(a)

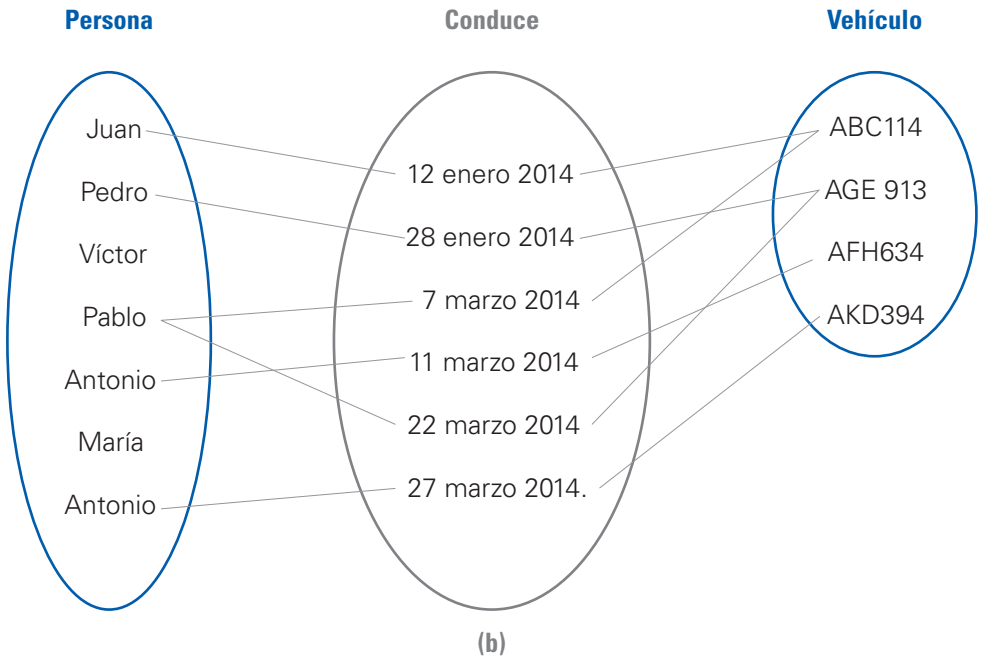


Figura 2.25. (a) Atributo de la relación CONDUCE. (b) Instancias del atributo *Fecha* de la relación CONDUCE.

Para el caso del ejemplo **La Ferretería**, en la relación DETALLE se registran dos atributos. El primero *Cantidad* que almacena el número de unidades compradas de un determinado artículo en una factura, y el segundo, el atributo derivado *Subtotal,* que calcula el precio del artículo multiplicado por el número de unidades compradas, como se muestra en la Figura 2.26.

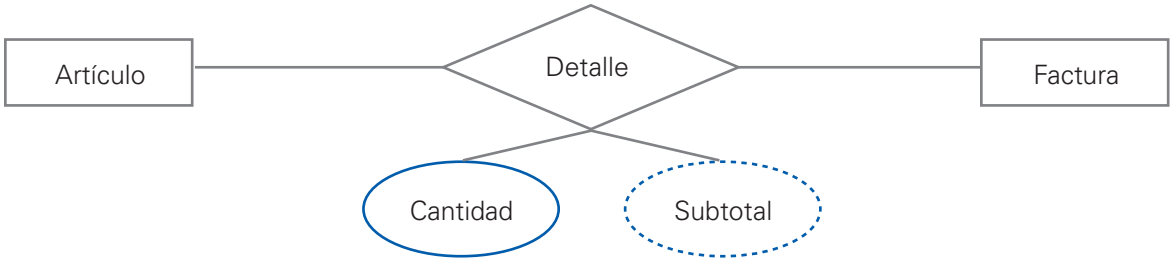


Figura 2.26 Atributos de una relación.

2.6 REVISIÓN DEL DISEÑO DEL MODELO ENTIDAD - RELACIÓN DE LA BASE DE DATOS LA FERRETERÍA

Una vez estudiados los conceptos del modelo Entidad – Relación, en esta sección se realiza una revisión general del diseño de la base de datos **La Ferretería** de la Figura 2.1, que de acuerdo con los requerimientos del sistema se han generado las relaciones indicadas a continuación:

Para modelar el requerimiento "Cada cliente tendrá asignado a un vendedor como representante", se crea la relación *Representa* con cardinalidad 1:N entre las entidades `VENDEDOR` y `CLIENTE`, debido a que un vendedor tendrá asignado a varios clientes a quien represente y un cliente será representado por un solo vendedor. Para definir la restricción de participación, en los requerimientos no se especifica si todos los vendedores tendrán a su cargo clientes, o si todos los clientes tendrán asignados a un vendedor como representante. En estas circunstancias es necesario consultar al usuario para determinar las condiciones y modelar el requerimiento. Para el caso del ejemplo, se considera una restricción de participación *Total – Total.*

La relación identificativa *TieneFamilia* con cardinalidad 1:N que asocia las entidades `VENDEDOR` y `CARGAFAMILIAR` es creada para modelar el requerimiento del mundo real entre los vendedores y sus cargas familiares. La participación para la entidad `VENDEDOR` es parcial, ya que no todos los vendedores tendrán cargas familiares y para `CARGAFAMILIAR` su participación es total por tratarse de una entidad subordinada. Como se explicó anteriormente, `CARGAFAMILIAR` es una entidad débil por no tener un atributo clave propio.

Para modelar los artículos adquiridos mediante una determinada factura se crea la relación `DETALLE` entre las entidades `ARTÍCULO` y `FACTURA`, con cardinalidad N:N considerando que un artículo está en varias facturas y una factura tiene varios artículos. Su participación es *Total - Total* según se define que todas las instancia de las entidades `ARTÍCULO` y `FACTURA` en algún momento formarán parte de la relación `DETALLE.` Se incluye el atributo de la relación *Cantidad*, que corresponde al número de unidades de un artículo adquirido en una factura. Por otra parte, en la relación `DETALLE` se especifica el atributo derivado *Subtotal*, que representa el producto de las cantidades vendidas de cada artículo por el precio unitario.

La relación *Elabora*, modela el requerimiento del mundo real que corresponde al vendedor que realiza la factura. Tiene una cardinalidad 1: N entre las entidades `VENDEDOR` y `FACTURA`, y su participación es *Total – Total.*

La relación `COMPRA` entre las entidades `CLIENTE` y `FACTURA` tiene una cardinalidad 1:N, puesto que un cliente tiene varias facturas, pero una factura pertenecerá a un solo cliente. Se determina que la participación es *Total – Total.*

Para modelar el requerimiento de los clientes que recomendaron a otros clientes, se crea una relación recursiva `RECOMIENDA`, en la entidad `CLIENTE`, con cardinalidad 1:N y los roles de "recomienda a" y "es recomendado". La restricción de participación es *Parcial – Parcial*, porque de acuerdo con las condiciones del mundo real, no todos los clientes cumplen los roles de recomendar y ser recomendados.

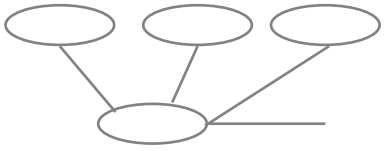
En la entidad `CLIENTE` se determina un atributo multivalor *Teléfono*, considerando el requerimiento, de que un cliente puede tener varios números telefónicos.

2.7 NOTACIÓN UTILIZADA PARA EL MODELO ER

A continuacion se presenta un resumen de la simbología del modelo Entidad - Relación, (Elmasri R., Navathe S., 2016), (Silberschatz, 2014)

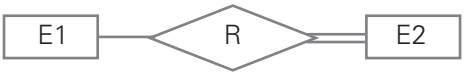
Notación	Significado
Nombre entidad	Entidad fuerte
Nombre entidad	Entidad débil
Nombre	Relación
Nombre	Relación asociada con una entidad débil - relación identificativa
Nombre	Atributo
Nombre	Atributo clave
Nombre	Clave parcial de una entidad débil

Atributo multivalor

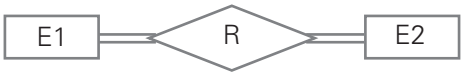


Atributo compuesto

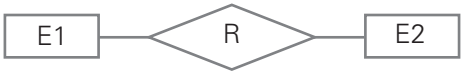
Atributo derivado



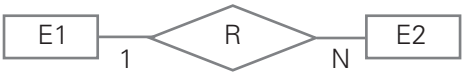
Participación parcial de E1 y total de E2 en R



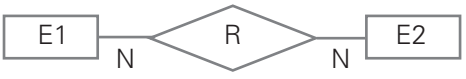
Participación total de E1 y total de E2 en R



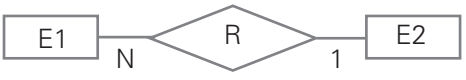
Participación parcial de E1 y parcial de E2 en R



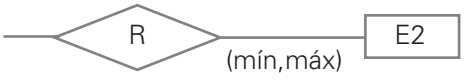
Cardinalidad 1:N para E1:E2 en R (relación uno a muchos)



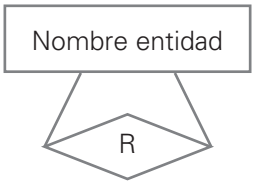
Cardinalidad N:N para E1:E2 en R (relación muchos a muchos)



Cardinalidad N:1 para E1:E2 en R (relación muchos a uno)



Restricción estructural (mín, máx)



Relación recursiva

2.8 TRANSFORMACIÓN DE UN ESQUEMA ENTIDAD – RELACIÓN

Un esquema conceptual de una base de datos obtenido mediante un diseño conceptual sirve como entrada para generar un esquema lógico. Los procesos de diseño generalmente parten de un modelo conceptual de alto nivel como el modelo ER, para luego transformarlos al modelo relacional. Estos dos modelos se fundamentan en los mismos principios de diseño, en consecuencia, es posible mediante procedimientos y algoritmos que serán tratados en el presente apartado, convertir un esquema ER en un esquema relacional que representa la base de datos como una colección de relaciones. Este procedimiento de transformación recibe también el nombre de "diseño de bases de datos relacionales utilizando el mapeado ER a relacional" (Elmasri R., Navathe S., 2007, pág. 190).

El concepto matemático de relación, físicamente se representa en forma de una **tabla** bidimensional en la que las columnas de la tabla corresponden a atributos y las filas equivalen a tuplas. Con el objeto de evitar ambigüedad y confusión con el uso del término formal *relación*, en este apartado se utiliza la terminología *tabla* para referirnos a una relación y *columna* a un atributo.

En los siguientes apartados se explica estos procesos de transformación, para lo cual se utiliza el modelo ER de nuestro caso de estudio **La Ferretería** que se muestra en la Figura 2.1.

2.8.1 Transformación de entidades fuertes

Cada entidad fuerte *E* se representa mediante una tabla *T,* y cada uno de sus atributos simples a_1, a_2, a_3,...a_n, se transforman en *n* columnas distintas. El atributo clave definido en el modelo corresponderá a la clave primaria de la tabla (ver sección 3.5.2).

Para ilustrar este proceso se analiza la entidad `VENDEDOR` del modelo de nuestro caso de ejemplo, que contiene siete atributos: *CódigoVend, Nombre, Apellido, Sexo, FechaContrato, ObjVentas y Teléfono*. Esta entidad se representa mediante una tabla llamada *Vendedor* con siete columnas de las cuales *CódigoVend* corresponde a la clave primaria, que para diferenciarlo irá subrayada tal como muestra en la Figura 2.27.

VENDEDOR

Nombre	Apellido	CódigoVend	Sexo	FechaContrato	ObjVentas	Teléfono

Figura 2.27 Transformación de la entidad `VENDEDOR` a tabla.

Con un proceso similar, a partir de las entidades CLIENTE, FACTURA y ARTÍCULO, del modelo ER, se crea las correspondientes tablas CLIENTE, FACTURA y ARTÍCULO con sus respectivas claves primarias *Cédula*, *Número* y *Código*, como se muestra en la Figura 2.28.

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito

FACTURA

Número	Fecha

ARTÍCULO

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín

Figura 2.28. Transformación de entidad fuerte.

En este proceso no se han considerado los atributos derivados y multivalor que serán analizados posteriormente.

Si en el modelo se presenta una clave compuesta, ésta se representa en la tabla mediante el conjunto de los atributos simples que lo forman. Por ejemplo, si se revisa el modelo de la Figura 2.5(a), en donde la entidad ARTÍCULO tiene cuatro atributos, de los cuales, el par de atributos simples *Código* y *Fabricante* formarán la clave compuesta que será representada en la tabla mediante las columnas *Código* y *Fabricante* que en conjunto formarán la clave primaria, como se detalla en la Figura 2.29.

ARTÍCULO

Código	Fabricante	Descripción	Precio
1	Fábrica 1	Perfil T	6,00
1	Fábrica 2	Perfil T	6,50
5	Fábrica 1	Perfil L	7,20
5	Fábrica 2	Perfil L	7,80
5	Fábrica 3	Perfil L	8,00
8	Fábrica 1	Perfil G	9,50
8	Fábrica 2	Perfil G	9,70

Figura 2.29. Tabla con clave primaria compuesta.

2.8.2 Transformación de entidades débiles

Si se considera el caso de la entidad débil `A` cuyos atributos son a_1, a_2, a_3,...a_n, y depende de la entidad dominante `B`, cuyo atributo clave es b_1. La transformación de estas entidades consiste en crear una tabla `A` que contenga como columnas todos los atributos simples `A` incluido el atributo clave b_1 de `B`, que se los denomina clave foránea o extranjera (en inglés *foreign key*).

La clave primaria de una tabla generada a partir de una entidad débil se forma con la clave primaria de la entidad fuerte, más la clave parcial de la entidad débil. Por ejemplo, para transformar la entidad débil `CARGAFAMILIAR` se crea la tabla `CARGAFAMILIAR` con las cuatro columnas que corresponden a sus atributos *Nombre, Sexo, FechaNac y Relación*, y se incluye como clave foránea la clave principal CódigoVend de la entidad dominante `VENDEDOR`, que se le renombra como *CódVendedor*.

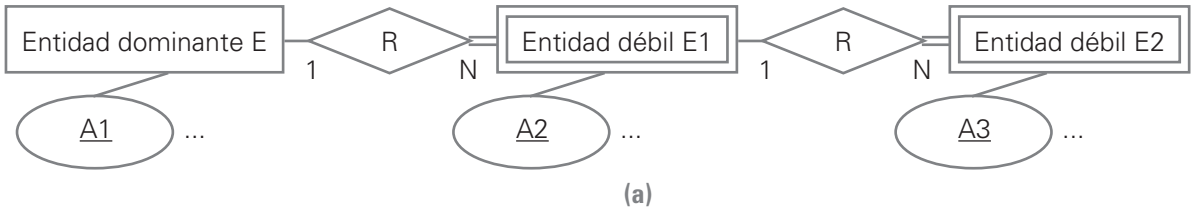
Cómo se analizó en la Sección 2.4, al atributo *NomDep* se le definió como clave parcial de la entidad `CARGAFAMILIAR`, que en combinación con la clave foránea *CódVendedor* forman la clave principal de la tabla `CARGAFAMILIAR`, como se ilustra en la Figura 2.30.

CARGAFAMILIAR

CódVendedor	NomDep	FechaNac	Relación	Sexo
1	María	27-oct-04	Hija	F
1	Isabel	03-dic-12	Hija	F
2	Antonio	16-abr-02	Hijo	M
2	Maricela	18-may-92	Cónyuge	F
3	Teodoro	25-oct-88	Cónyuge	M

Figura 2.30. Tabla transformada de una entidad débil.

Es posible que se presenten requerimientos en donde una entidad débil E2 sea subordinada de otra entidad débil E1 como se muestra en la Figura 2.31(a). La clave primaria de la tabla E2 será la combinación de la clave primaria de la entidad fuerte más las claves parciales de las entidades débiles E1 y E2. En la Figura 2.31(b) se ilustra el resultado de esta transformación.



(a)

Tabla E

A1	...

Tabla E1

A1	A2	...

Tabla E2

A1	A2	A3	...

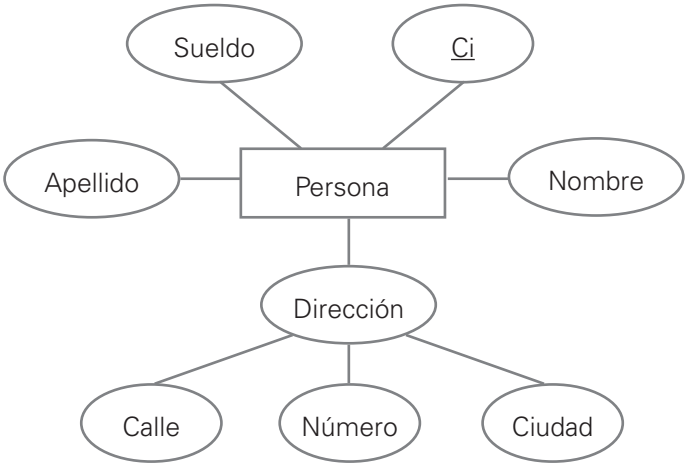
(b)

Figura 2.31. (a) Entidad débil E2 subordinada de la entidad débil E1. (b) Transformación de las entidades débiles E1 y E2 a tablas.

2.8.3 Transformación de atributos compuestos

En la Figura 2.32(a) se muestra la entidad `PERSONA` con el atributo compuesto *Dirección* y sus atributos simples que lo conforman: *Calle, Número y Ciudad*. Dos alternativas se plantean para transformar los atributos compuestos: la primera consiste en eliminar el atributo compuesto *Dirección* considerando todas su subpartes *Calle, Número y Ciudad*, como atributos simples de `PERSONA`. La tabla generada contendrá una columna para la clave primaria *Ci* y cinco columnas adicionales para los atributos simples. No se considera una columna para *Dirección*. La Figura 2.32(b) muestra el resultado de la tabla `PERSONA`.

La segunda alternativa consiste en crear dos tablas, una `PERSONA` que corresponde a la entidad del mismo nombre, con su clave primaria *Ci* y sus atributos propios *Nombre, Apellido y Sueldo* y otra tabla `DIRECCIÓN` que tiene como clave foránea el identificador *Ci* de la entidad `PERSONA` y las tres columnas que corresponden a la subpartes del atributo compuesto. Esta alternativa se muestra en la Figura 2.32(c).



(a)

PERSONA

CI	Nombre	Apellido	Sueldo	Calle	Número	Ciudad

(b)

PERSONA

CI	Nombre	Apellido	Sueldo

DIRECCIÓN

CI	Calle	Número	Ciudad

(c)

Figura 2.32. (a) Diagrama ER con atributo compuesto. (b) Transformación mediante eliminación de atributo compuesto. (c) Transformación que genera una nueva tabla con clave foránea.

2.8.4 Transformación de relaciones

2.8.4.1 Relación N:N

Para cada relación N:N se crea una tabla que tiene como clave primaria la combinación de los atributos clave de cada una de las entidades participantes. A esta tabla se le incluye cualquier tipo de atributo de la relación. La nueva tabla tiene la finalidad de crear una referencia cruzada entre las claves primarias. Es por este motivo que a este tipo de tablas se las denomina **tablas de relaciones** o **referencias cruzadas**. (Elmasri R., Navathe S., 2007, pág. 193).

La Figura 2.33 muestra la transformación a tabla de una relación `R` con cardinalidad N:N, en la que participan las entidades `E1` y `E2`, con sus atributos clave *C1* y *C2* respectivamente. Adicionalmente se incluye el atributo simple *A* en la relación `R`.

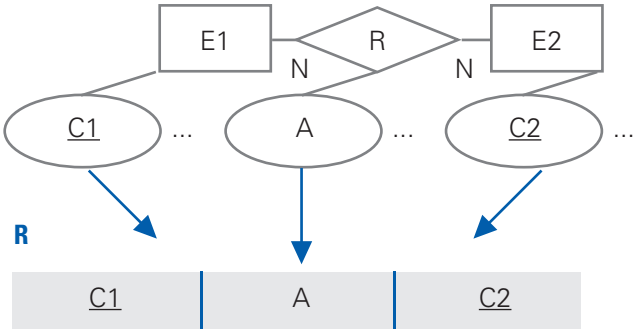


Figura 2.33. Transformación a tablas de un tipo de relación N:N.

Para el caso de nuestro ejemplo **La Ferretería**, para transformar la relación `DETALLE` con cardinalidad N:N, se crea la tabla `DETALLE`, que contiene dos claves foráneas *FacNúmero* y *ArtCódigo* que corresponde a los atributos clave *Número* y *Código* de las entidades `FACTURA` y `ARTÍCULO`. La combinación de estas dos claves foráneas forma la clave primaria de la nueva tabla `DETALLE`. Adicionalmente se incluye el atributo *Cantidad* de la relación `DETALLE`. La Figura 2.34 (Página 75) ilustra el resultado de la transformación de una de relación N:N.

2.8.4.2 Relación 1:N

Cuando se trata de una relación `R` con cardinalidad 1:N, se propaga como clave foránea el atributo clave de la entidad que tiene cardinalidad 1 a la entidad que tiene cardinalidad N. Como ejemplo, se realiza el proceso de transformación de las relaciones 1:N que se presentan en nuestro caso de estudio **La Ferretería**.

Para el caso de la relación `REPRESENTA` se crea una clave foránea propagando el atributo clave *CódigoVend* de la entidad `VENDEDOR`, renombrado como *CódRepre* en la tabla `CLIENTE`. El atributo simple de la relación *FechaAsignación*, renombrado como *FechaAsig* se propagará conjuntamente con el atributo clave a la tabla `CLIENTE`, como se detalla en la Figura 2.35. (Página 76)

DETALLE

FactNúmero	ArtCódigo	Cantidad
100	2	5
100	9	1
100	10	5
101	6	4
101	8	5
102	5	6
103	11	1
103	9	4
103	1	4
104	7	2
104	8	2
105	1	5
106	2	8
107	5	4
107	2	3
108	9	4
108	10	10
109	11	1
110	10	3
110	2	1
111	6	5
111	9	2

Figura 2.34. Tabla que representa la relación N:N y un atributo de la relación.

CLIENTE

Número	Apellido1	Apellido2	Cédula	...	CódRepre	FechaAsig
Víctor	Castro	Torres	0102030405	. . .	1	05-jul-04
Juan	Polo	Ávila	0122334455	. . .	1	16-dic-07
Elena	Tapia	Barrera	0111555666	. . .	3	13-mar-10
Pablo	Brito	Parra	0123456789	. . .	3	17-may-11
Marcia	Mora	Durazno	0987654321	. . .	1	21-feb-08
Jaime	Pérez	Guerrero	0777888999	. . .	2	28-oct-09
Manuel	Bonilla	León	0123123123	. . .	2	09-sep-09
Alberto	Torres	Viteri	0999998888	. . .	4	22-sep-11
Humberto	Pons	Coronel	0456456456	. . .	3	11-jun-11

Figura 2.35. Resultado del mapeo de una relación con cardinalidad 1:N y un atributo de la relación.

Para las relaciones con cardinalidad 1:N, `ELABORA` y `COMPRA` se realiza un proceso similar, incluyendo los atributos clave *CódigoVend* y *Cédula* de las entidades correspondientes `VENDEDOR` y `CLIENTE`, en la tabla `FACTURA`, creando dos claves foráneas que se las renombra como *CódVendedor* y *CcCliente* respectivamente. El resultado de este proceso se detalla en la Figura 2.36. (Página 77)

Existen requerimientos del mundo real en los que conviene que una relación con cardinalidad 1:N se transforme en una tabla, como si se tratase de una relación N:N, estos casos se detallan a continuación:

- Para evitar valores nulos en la clave foránea de la nueva tabla. Estos valores se presentan cuando el número de ocurrencias de la entidad que propaga la clave es muy pequeño. Por ejemplo, en la Figura 2.37 se detalla la transformación de la relación `REPRESENTA`, creando una clave foránea en la tabla `CLIENTE` a partir de la propagación de la clave primaria de la tabla `VENDEDOR`; sin embargo, como se observa en la gráfica, existe un limitado número de instancias de la relación, lo que implica que en la tabla `CLIENTE` se generen valores nulos, en este caso, se recomienda crear una nueva tabla de la relación.

FACTURA

Número	Fecha	CódVendedor	CcCliente
100	06-ene-14	1	0102030405
101	13-ene-14	3	0122334455
102	28-ene-14	1	0123456789
103	05-feb-14	4	0122334455
104	19-feb-14	2	0987654321
105	20-feb-14	2	0111555666
106	25-feb-14	4	0777888999
107	13-mar-14	3	0999998888
108	17-mar-14	3	0123123123
109	20-mar-14	2	0123123123
110	21-mar-14	4	0456456456
111	25-mar-14	1	0123123123

Figura 2.36. Resultado de la transformación de la relación ELABORA y COMPRA.

CLIENTE

Nombre	Apellido1	Cédula	...	CódRepre
Víctor	Castro	0102030405	. . .	1
Juan	Polo	0122334455	. . .	1
Elena	Tapia	0111555666	. . .	3
. . .	. . .	. . .	. . .	null
. . .	. . .	. . .	. . .	null
. . .	. . .	. . .	. . .	null
. . .	. . .	. . .	. . .	null
Pablo	Brito	0123456789	. . .	3
Marcia	Mora	0987654321	. . .	1
Jaime	Pérez	0777888999	. . .	null
Manuel	Bonilla	0123123123	. . .	2
Alberto	Torres	0999998888	. . .	4
Humberto	Pons	0456456456	. . .	null

VENDEDOR

Vnombre	Apellido	CódigoVend	...
Diego	Loja	1	. . .
Antonio	Calle	2	. . .
Lucía	Serrano	3	. . .
Arturo	Salto	4	. . .
Maricela	Vera	5	. . .

Figura 2.37. Caso de cardinalidad 1:N que recomienda crear una tabla de la relación.

- Cuando se presenten posibilidades de modificaciones del mudo real, es decir, que en el futuro la relación 1:N se cambie a cardinalidad N:N

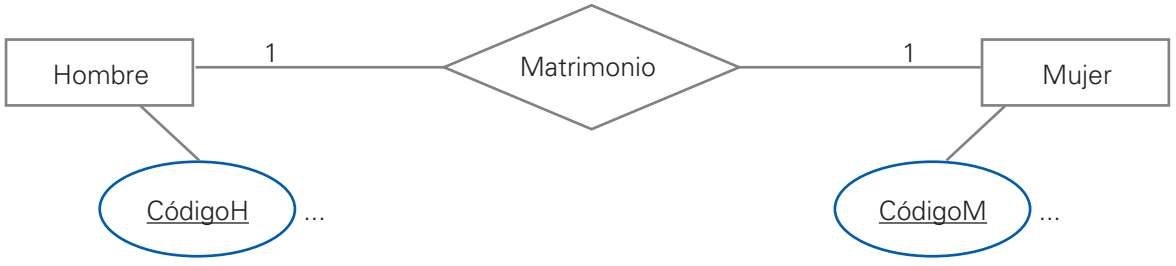
- Cuando la relación tiene un considerable número de simples atributos.

2.8.4.3 Relación 1:1

Para la relación con cardinalidad 1:1 se tienen tres alternativas de transformación que dependerán de la restricción de participación de las entidades asociadas.

- Participación parcial - parcial. Si ambas entidades tienen una participación parcial en la relación, y el número de instancias de la relación es mínimo, es recomendable crear una nueva tabla para evitar valores nulos.

  Si se considera el modelo del ejemplo de la Figura 2.38, que muestra dos entidades `HOMBRE` y `MUJER` asociados con una relación `MATRIMONIO`, su restricción de cardinalidad es 1:1 y la propiedad de participación *Parcial – Parcial*. Si las instancias de la relación `MATRIMONIO` comparadas con el número de instancias de entidad de cada una de las entidades asociadas es mínimo, en este caso se crea una tabla con similar metodología de la que se utiliza para la relación N:N.



HOMBRE

CódigoH	...
H1	. . .
H2	. . .
H3	. . .
H4	. . .
. . .	. . .
. . .	. . .
. . .	. . .
.....	. . .
H15	. . .
H16	. . .
. . .	. . .
. . .	. . .
H150	. . .
H151	. . .
. . .	. . .
. . .	. . .
. . .	. . .
H400	. . .
H401	. . .
. . .	. . .
. . .	. . .
. . .	. . .
H500	. . .

MUJER

CódigoM	...
M1	. . .
M2	. . .
M3	. . .
M4	. . .
. . .	. . .
. . .	. . .
. . .	. . .
. . .	. . .
. . .	. . .
M20	. . .
M21	. . .
. . .	. . .
. . .	. . .
M200	. . .
M201	. . .
M202	. . .
. . .	. . .
. . .	. . .
M450	. . .
. . .	. . .
. . .	. . .
. . .	. . .
M500	. . .

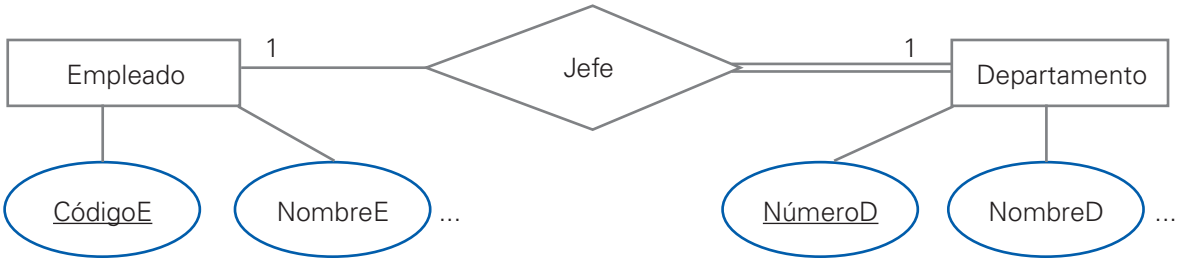
MATRIMONIO

CódHombre	CódMujer
H1	M3
H4	M450
H150	M20
H401	M202
H500	M450
H16	M4

Figura 2.38. Tabla generada a partir de una cardinalidad 1:1 con participación Parcial – Parcial.

- Participación Total – Parcial. En este caso lo conveniente es propagar el atributo clave de la entidad con participación parcial, creando una clave foránea en la tabla con participación total. Realizar lo contrario, es decir, propagar el atributo clave desde la entidad con participación total, genera valores nulos.

  Como ejemplo de esta alternativa de transformación se consideran las entidades `EMPLEADO` y `DEPARTAMENTO`, asociados con la relación `JEFE`. En este caso la participación de `EMPLEADO` en la relación es parcial, puesto que no todos los empleados serán jefes de departamento, pero sí, todos los departamentos tendrán un empleado como jefe. Para transformar este requerimiento del mundo real, se propaga la clave principal *CódigoE* de `EMPLEADO` a la tabla `DEPARTAMENTO`, creando una clave foránea *CódEmp*, como se muestra en la Figura 2.39.



EMPLEADO

CódigoE	NombreE	...
E1	Juan	. . .
E2	Pedro	. . .
E3	María	. . .
E4	Eduardo	. . .
E5	Juana	. . .
E6	Alberto	. . .
. . .	. . .	. . .
. . .	. . .	. . .
. . .	. . .	. . .
. . .	. . .	. . .
. . .	. . .	. . .
E99	Mauricio	. . .
E100	Carmen	. . .

DEPARTAMENTO

NúmeroD	NombreD	...	CódEmp
1	Financiero	. . .	E2
2	Administrativo	. . .	E6
3	Técnico	. . .	E99

Figura 2.39. Transformación de una relación1:1 con participación Parcial – Total.

La tercera alternativa de transformación corresponde a la relación con una restricción de participación Total - Total. En este caso es posible propagar cualquiera de los atributos clave de las dos entidades participantes y generar una clave foránea en la tabla correspondiente.

Para ilustrar esta alternativa se analiza la relación `SERVICIO` con la participación de las entidades `PREDIO` y `MEDIDOR`. Se supone que en cada predio se instala únicamente un medidor de servicio eléctrico, es decir, la relación tiene una restricción de cardinalidad 1:1. Adicionalmente se considera que todas las instancias de predios y medidores se encuentran relacionados generando una restricción de participación Total – Total. El modelo ER aparece en la Figura 2.40(a).

A continuación se analizan las dos posibilidades de propagación de los atributos clave. La primera consiste en propagar el atributo clave *NúmeroC* de la entidad `PREDIO` y crear una clave foránea renombrada como *NúmCatastro* en la tabla `MEDIDOR`, y como segunda posibilidad, propagar el atributo clave *NúmeroM* de la entidad `MEDIDOR` y crear una clave foránea renombrada como *NúmMedidor* en la tabla `PREDIO`. En la Figura 2.40(b) y 2.40(c) se muestran las dos opciones de transformación.

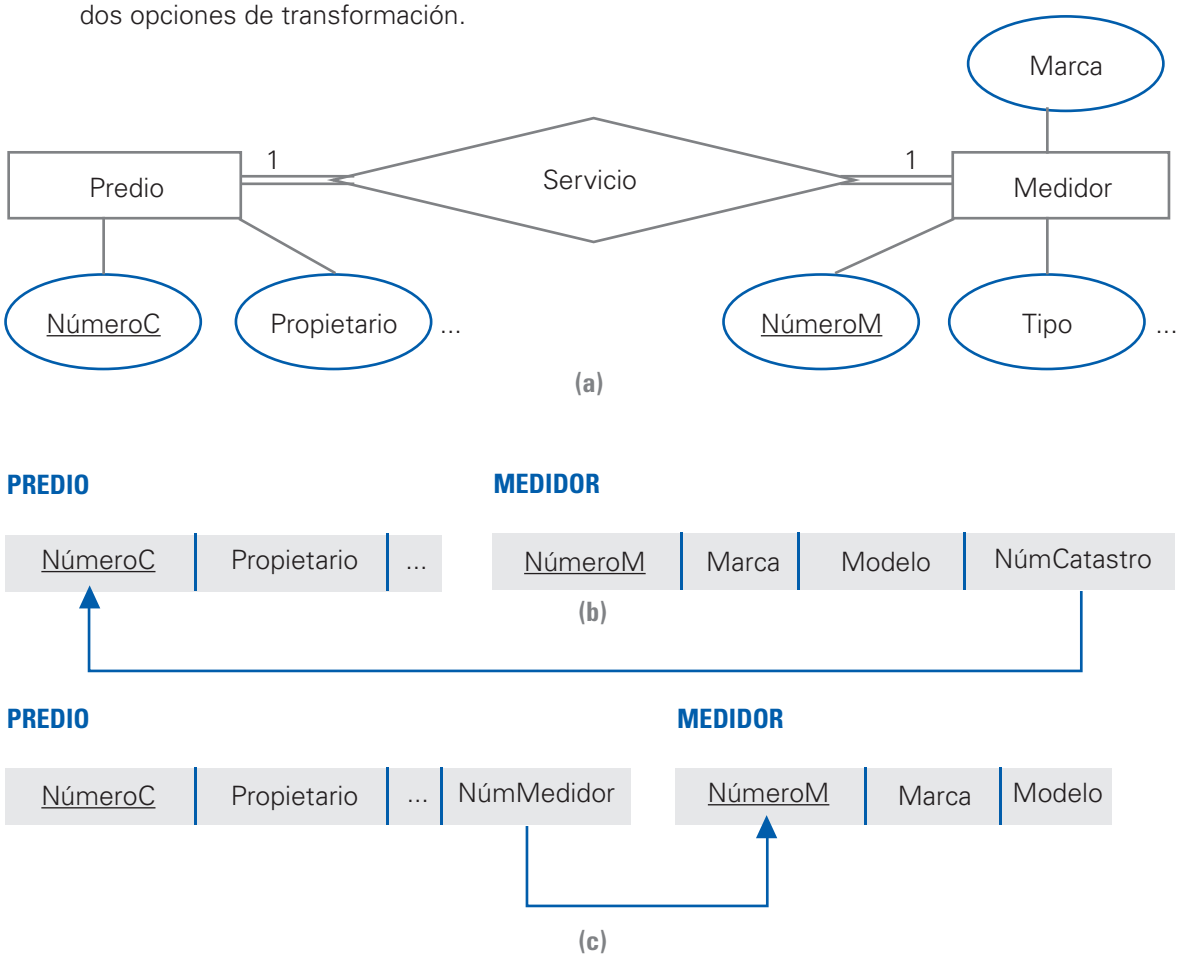
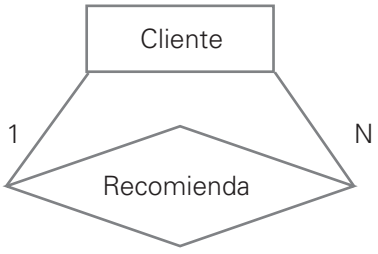


Figura 2.40. Transformación a tablas de una relación con cardinalidad 1:1 y participación Total – Total. (a) Modelo ER. (b) Alternativa 1 de propagación. (c) Alternativa 2 de propagación.

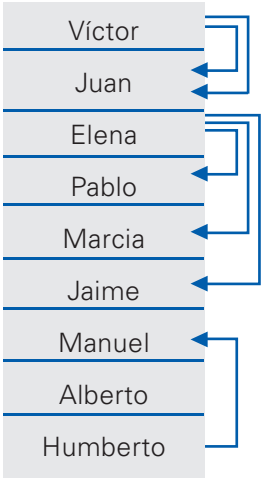
2.8.5 Transformación de una relación recursiva

Para ilustrar el proceso de transformación se utiliza la relación recursiva `RECOMIENDA` de la entidad `CLIENTE` de nuestro caso de estudio **La Ferretería**. La Figura 2.41(a) muestra el modelo ER de esta relación y la Figura 2.41(b) ejemplifica las instancias de entidades que participan en la relación.

Para transformar esta relación se crea una nueva columna como clave foránea en la tabla Clientes, que resulta de la propagación de la clave primaria *Cédula* de la misma relación y se la renombra como *CédulaReco*. En la Figura 2.41(c) se muestra la transformación de esta relación recursiva y las instancias de la relación.



(a)



(b)

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CédulaReco
Víctor	Castro	Torres	0102030405	Sucre 1-12	600	Null
Juan	Polo	Ávila	0122334455	Bolívar 5-67	1000	0102030405
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500	0102030405
Pablo	Brito	Parra	0123456789	Alfaro 1-23	Null	0111555666
Marcia	Mora	Durazno	0987654321	Olmedo 1-10	1000	0111555666
Jaime	Pérez	Guerrero	0777888999	Tálbot 4-89	1200	0111555666
Manuel	Bonilla	León	0123123123	B. Malo 5-98	600	0456456456
Alberto	Torres	Viteri	0999998888	Cordero 10-10	1200	null
Humberto	Pons	Coronel	0456456456	Córdova 3-45	2000	null

(c)

Figura 2.41. Transformación de una relación recursiva. (a) Relación recursiva en el modelo ER. (b) Instancias de la relación. (c) Tabla generada de la relación recursiva.

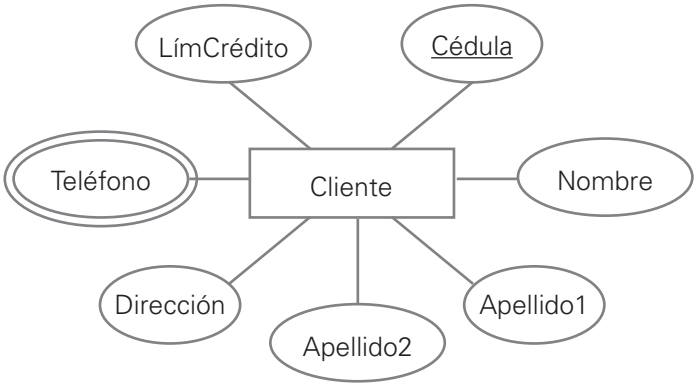
2.8.6 Transformación de atributos multivalor

El proceso para transformar este tipo de atributos consiste en crear una nueva tabla por cada atributo multivalor, que contiene la clave foránea correspondiente al atributo clave de la entidad y adicionalmente una columna con el atributo multivalor. La clave principal de la nueva tabla está conformada por la clave foránea y el atributo multivalor.

Como ejemplo de este proceso de transformación se revisa el atributo multivalor *Teléfono* de la entidad `CLIENTE` de nuestro caso **La Ferretería**. Se crea una nueva tabla `CLIENTE_TELÉF` formada por la clave foránea *CédulaCli* a partir del atributo clave *Cédula* de la entidad `CLIENTE` y la columna *NúmeroTeléf* que representa el atributo multivalor *Teléfono*. La clave principal de la nueva tabla está integrada por el par *CédulaCli*, *NúmeroTeléf*. En la Figura 2.42(a) y 2.42(b) se muestra el modelo ER de una entidad con atributo multivalor y la transformación a tablas respectivamente.

2.8.7 Atributos de una relación

Si una relación tiene un atributo propio se analizó que éste se propague junto con el atributo clave a la tabla donde se crea la clave foránea, sin embargo, es conveniente que aquellas relaciones que contienen varios atributos propios se transformen en una tabla, en donde los atributos pasan a ser columnas de dicha tabla, como si se tratase de una relación N:N.



(a)

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito
Víctor	Castro	Torres	0102030405	Sucre 1-12	600
Juan	Polo	Ávila	0122334455	Bolívar 5-67	1000
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500
Pablo	Brito	Parra	0123456789	Alfaro 1-23	Null
Marcia	Mora	Durazno	0987654321	Olmedo 1-10	1000
Jaime	Pérez	Guerrero	0777888999	Tálbot 4-89	1200
Manuel	Bonilla	León	0123123123	B. Malo 5-98	600
Alberto	Torres	Viteri	0999998888	Cordero 10-10	1200
Humberto	Pons	Coronel	0456456456	Córdova 3-45	2000

CLIENTE_TELÉF

CédulaCli	NúmeroTeléf
0102030405	2876543
0102030405	4789673
0111555666	2874763
0123456789	2865434
0123123123	4776289
0123123123	4774567
0999998888	2889884
0456456456	4968322
0456456456	4058011
0777888999	2810083
0122334455	2890890
0987654321	4853510

(b)

Figura 2.42. Transformación de un atributo multivalor: (a) Modelo ER con atributo multivalor. (b) Tabla `CLIENTE_TELÉF` con una fila por cada número telefónico.

2.8.8 Resultado de la transformación a tablas del caso La Ferretería

Luego de realizar los procesos de transformación del modelo ER a tablas, un posible diagrama del esquema relacional de la base de datos **La Ferretería** se muestra en la Figura 2.43(a), y en la Figura 2.43(b) se muestra una alternativa de base de datos.

Es preciso indicar que en este esquema no se han dibujado los arcos que van desde cada clave foránea a la tabla a la que se refieren. Estos arcos serán considerados más adelante cuando se analicen los conceptos de *integridad referencial*.

ARTÍCULO

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín

DETALLE

FactNúmero	ArtCódigo	Cantidad

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CédulaReco	CódRepre	Fecha_Asig

VENDEDOR

Vnombre	Apellido	CódigoVend	Sexo	FechaContrato	ObjVentas	Teléfono

CARGAFAMILIAR

NomDep	Sexo	FechaNac	Relación	CódVendedor

FACTURA

Número	Fecha	CódVendedor	CcCliente

CLIENTE_TELÉF

CédulaCli	NúmeroTeléf

(a)

VENDEDOR

Vnombre	Apellido	CódigoVend	Sexo	FechaContrato	ObjVentas	Teléfono
Diego	Loja	1	M	03-mar-12	3000	2472713
Antonio	Calle	2	M	12-dic-11	2500	2876549
Lucía	Serrano	3	F	03-may-11	2000	Null
Arturo	Salto	4	M	15-jul-12	4000	2887643
Maricela	Vera	5	F	18-ago-08	3500	4657890

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CédulaReco	CódRepre	FechaAsig
Víctor	Castro	Torres	0102030405	Sucre 1-12	600	0122334455	1	13-oct-12
Juan	Polo	Ávila	0122334455	Bolívar 5-67	1100	0456456456	1	02-mar-13
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500	0777888999	Null	04-jun-12
Pablo	Brito	Parra	0123456789	Alfaro 1-23	Null	0456456456	3	15-dic-11
Marcia	Mora	Durazno	0987654321	Olmedo 1-10	1000	0777888999	1	23-dic-12
Jaime	Pérez	Guerrero	0777888999	Colón 4-98	1200	0122334455	2	02-oct-12
Humberto	Pons	Coronel	0456456456	Borrero 3-45	2000	Null	3	10-sep-13
Alberto	Torres	Viteri	0999998888	Luque 10-10	1200	0122334455	4	13-ene-13
Manuel	Bonilla	León	0123123123	Montalvo 5-98	600	0122334455	2	12-feb-12

CARGAFAMILIAR

NomDep	Sexo	FechaNac	Relación	CódVendedor
María	F	27-oct-04	Hija	1
Isabel	F	03-dic-12	Hija	1
Antonio	M	16-abr-02	Hijo	2
Maricela	F	18-may-92	Cónyuge	2
Teodoro	M	25-oct-88	Cónyuge	3

ARTÍCULO

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín
1	Cemento	6,0	saco	50	10
2	Clavos	4,8	kilo	70	5
5	Alambre	3,0	libra	20	5
6	Perfil T	15,0	unidad	10	3
7	Perfil L	15,5	unidad	15	5
8	Perfil G	16,0	unidad	10	3
9	Pintura	19,5	galón	15	4
10	Lija	0,6	unidad	30	10
11	Suelda	20,0	kilo	10	5

FACTURA

Número	Fecha	CódVendedor	CcCliente
100	06-ene-14	1	0102030405
101	13-ene-14	3	0122334455
102	28-ene-14	1	0123456789
103	05-feb-14	4	0122334455
104	19-feb-14	2	0987654321
105	20-feb-14	2	0111555666
106	25-feb-14	4	0777888999
107	13-mar-14	3	0999998888
108	17-mar-14	3	0123123123
109	20-mar-14	2	0123123123
110	21-mar-14	4	0456456456
111	25-mar-14	1	0123123123

CLIENTE_TELÉF

CédulaCli	NúmeroTeléf
0102030405	2876543
0102030405	4789673
0111555666	2874763
0123456789	2865434
0123123123	4776289
0123123123	4774567
0999998888	2889884
0456456456	4968322
0456456456	4058011
0777888999	2810083
0122334455	2890890
0987654321	4853510

DETALLE

FactNúmero	ArtCódigo	Cantidad
100	2	5
100	9	1
100	10	5
101	6	4
101	8	5
102	5	6
103	11	1
103	9	4
103	1	4
104	7	2
104	8	2
105	1	5
106	2	8
107	5	4
107	2	3
108	9	4
108	10	10
109	11	1
110	10	3
110	2	1
111	6	5
111	9	2

(b)

Figura 2.43. (a) Esquema de la base de datos La Ferretería. (b) Instancia de la base de datos del caso La Ferretería.

2.9 AGREGACIÓN

En las secciones anteriores se analizaron conceptos del modelo ER, con los que se puede modelar la mayoría de las características de una base de datos relacional; sin embargo, existen requisitos más complejos que los necesarios para el diseño de las aplicaciones tradicionales, que implican el desarrollo de nuevos conceptos semánticos de modelado, que han sido incluidos en el modelo ER, obteniéndose el modelo Entidad Relación Extendido (EER por sus siglas en inglés *Enhanced ER* o *Extended ER*). La agregación, es uno de estos conceptos que serán tratados a continuación.

En el modelo ER no es posible formular relaciones entre relaciones, para expresar esta necesidad, se usa el concepto abstracto de la agregación, mediante el cual la relación se trata como una entidad de nivel superior.

Para ilustrar el uso de la agregación se consideran las entidades `SALA` y `EVENTO`, en donde, una sala será reservada por un cliente para realizar un determinado evento social. Tanto el precio del alquiler de la sala como su capacidad estarán en función del evento que se realice. El cliente está definido por la entidad `CLIENTE` y es quien realiza una reservación del par de entidades relacionadas `SALA` con `EVENTO` denominada `UTILIZAPARA`, lo que implica modelar una relación con otra relación, como se muestra en la Figura 2.44.

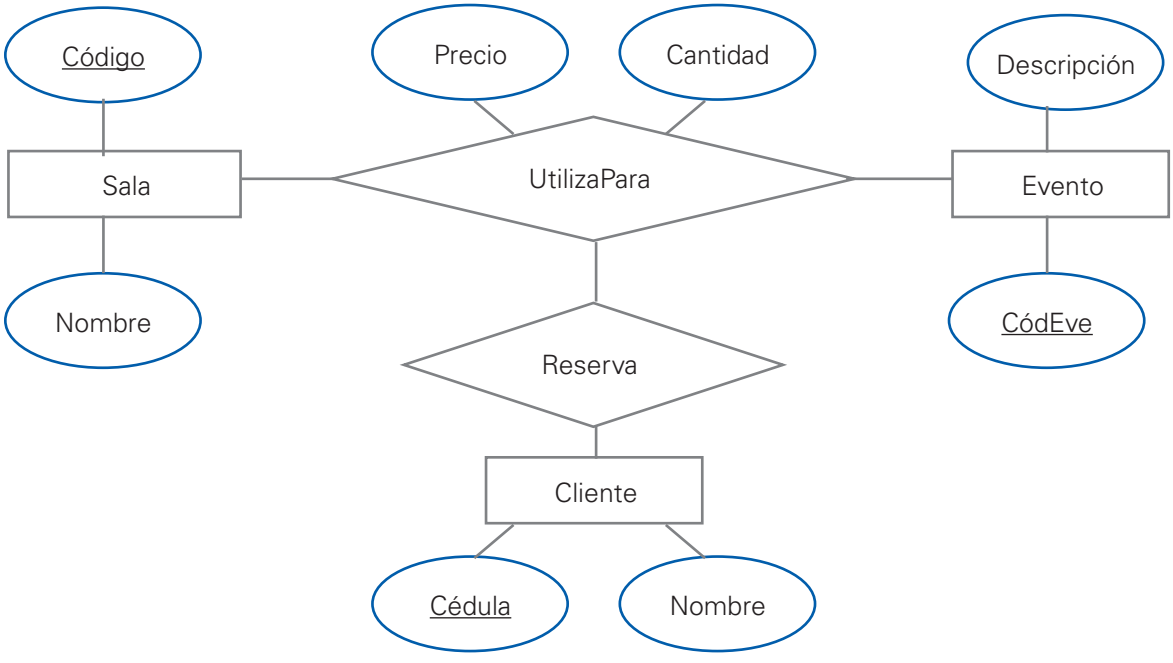
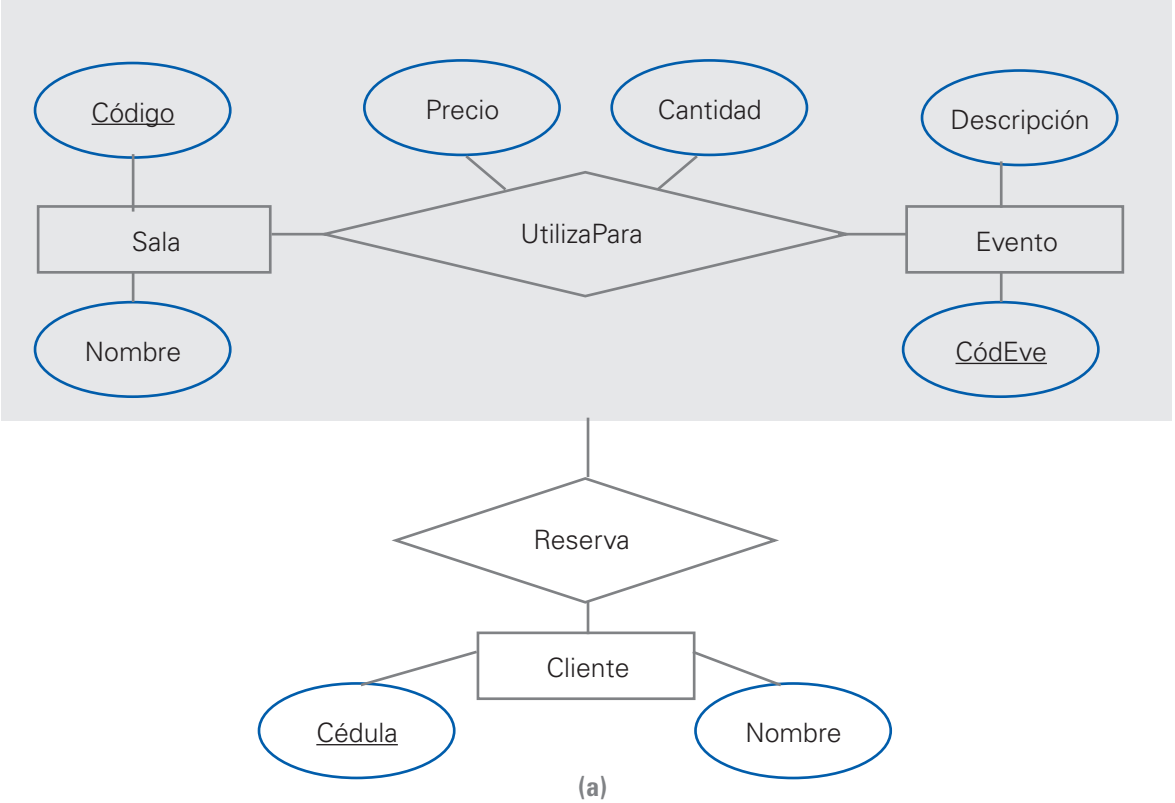


Figura 2.44. Relación no permitida `UTILIZAPARA` con `RESERVA`.

Al no ser posible una relación de relación, la mejor forma de modelar la situación descrita en la Figura 2.44 es a través de una agregación, que consiste en generar una entidad de nivel superior compuesta por: `SALA`, `EVENTO` y `UTILIZAPARA` y a esta agregación relacionarla con la entidad `CLIENTE`. El tratamiento de este grupo de entidades es similar al de cualquier entidad.

Para la transformación a tabla la clave primaria de la agregación está formada por la clave primaria del conjunto de entidades y relaciones que la define. En la Figura 2.45(a) se muestra el modelo ER con el uso de la agregación, y en la Figura 2.45(b) se describen sus correspondientes tablas.



(a)

SALA

Código	Nombre

EVENTO

CódEve	Descripción

UTILIZAPARA

Código	CodEve	Precio	Capacidad

RESERVA

Código	CodEve	Cédula

CLIENTE

Cédula	Nombre

(b)

Figura 2.45. (a) Modelo ER con agregación. (b) Generación de tablas a partir de una agregación.

2.10 Cuestionario y ejercicio

Preguntas de repaso

1. Defina los términos entidad, atributo y dominio.

2. Enumere los tipos de atributos y proporcione ejemplos de cada uno.

3. Explique el significado de una relación en el modelo Entidad - Relación y dé un ejemplo de relaciones unarias, binarias y ternarias.

4. Explique mediante un ejemplo el significado de instancia de una relación.

5. ¿A qué se denomina propiedades de una relación? Enumere las propiedades.

6. Explique mediante ejemplos las restricciones de cardinalidad en el modelo Entidad - Relación.

7. Describa el significado de dependencia de existencia.

8. Explique la diferencia entre una entidad débil y una entidad fuerte.

9. Explique mediante un ejemplo el tipo de relación recursiva.

10. Describa las alternativas de transformación a tablas de una relación con cardinalidad 1:1.

11. ¿Qué es la propiedad de participación? Dé un ejemplo para ilustrar este concepto.

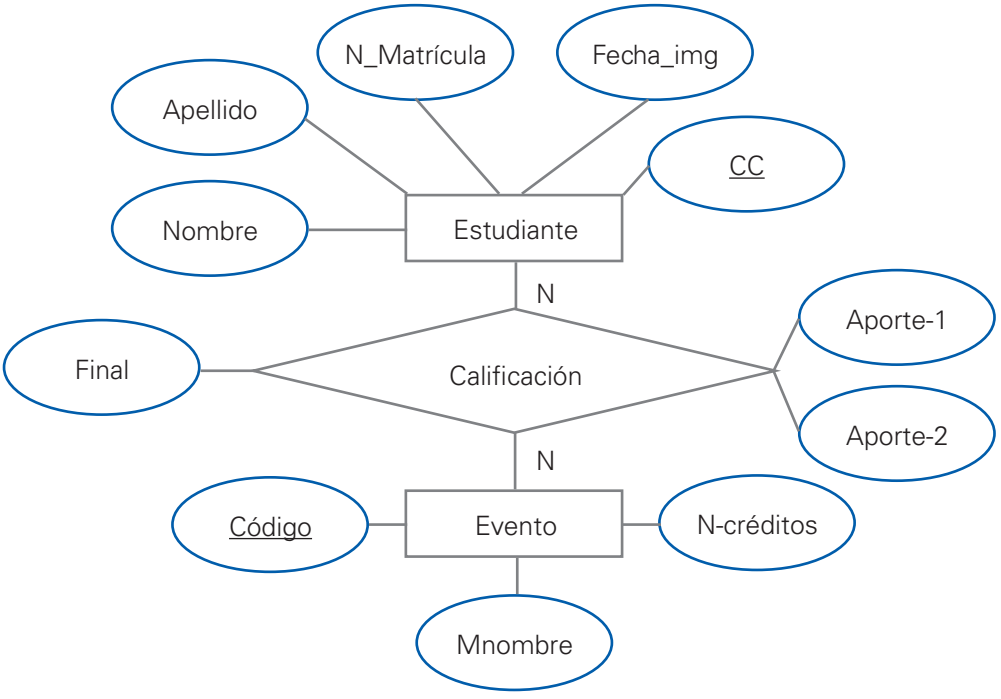
12. Proporcionar un ejemplo de atributos de una relación y explique el proceso de la transformación a tablas.

13. Describa el término agregación y proporcione un ejemplo.

Ejercicios resueltos

14. Registro de calificaciones

> Diseñar un esquema ER para el registro de las calificaciones (exámenes y aportes) de los estudiantes en las diferentes materias que están cursando. A lo largo del período de clases se realizan dos aportes y un examen. La base de datos registra para cada estudiante el nombre, apellido, número de cédula de ciudadanía y la fecha en la que se matriculó en el establecimiento educativo. También se registra el nombre de la materia y el número de créditos. Dibuje el esquema ER y transforme a tablas.



ESTUDIANTE	Cc	Nombre	Apellido	Fecha_ing	N_Matrícula
	1101775849	Nelly	Barrera	01-oct-95	5630
	1234567890	Mario	Fajardo	05-nov-98	7810
	9876543210	Luis	Arias	09-sep-99	9315
	1122334455	Francisco	Cabrera	07-may-99	9148
	. . .	. . .	. . .	. . .	. . .

MATERIA	Código	Mnombre	N_créditos
	ISI515	Bases de Datos	5
	ISI213	Lenguaje I	6
	ISI 432	Estructura de Datos	3
	ISI 313	Lenguaje II	6
	ISI 712	Sistemas Expertos	4
	. . .	. . .	. . .

CALIFICACIÓN	Cédula	CódMat	Aporte-1	Aporte-2	Final
	1101775849	ISI515	8	6	7
	1234567890	ISI515	9	7	7
	9876543210	ISI712	7	4	8
	1122334455	ISI213	6	9	6
	1101775849	ISI432	6	8	7
	1101775849	ISI712	5	6	9
	1234567890	ISI313	9	8	8
	. . .	. . .	. . .	. . .	. . .

15. Eliminatorias

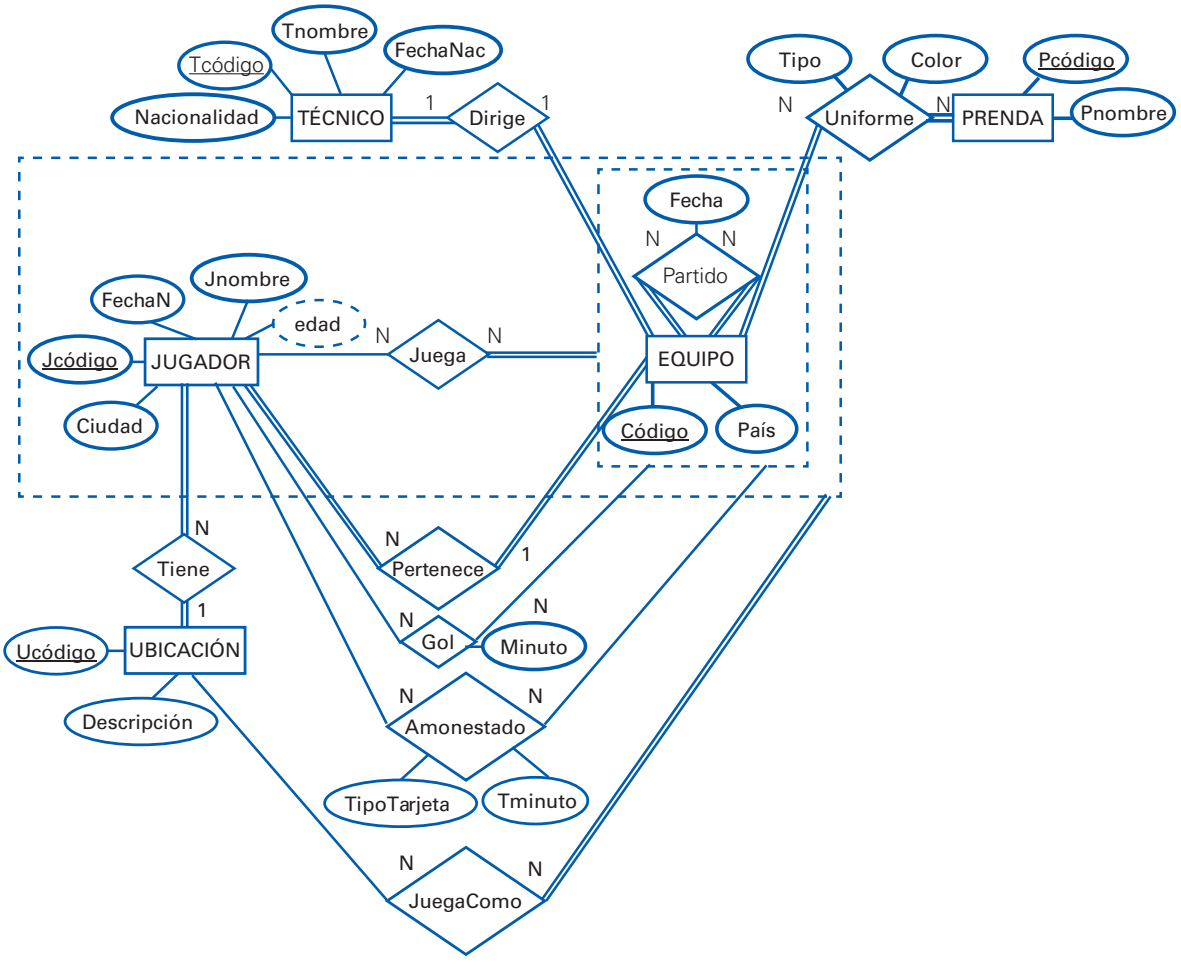
Elaborar el modelo entidad - relación y las tablas para la gestión de la información de las eliminatorias sudamericanas para el campeonato mundial de fútbol.

Para cada equipo de fútbol (País) se requiere saber el nombre de los jugadores, los datos del director técnico, los colores de su uniforme (camiseta, pantaloneta y medias) tanto el uniforme principal como el alterno. Se considerará un solo color para cada prenda. Para el director técnico registrar el nombre, la fecha de nacimiento y su nacionalidad.

De cada jugador es necesario registrar su nombre, la fecha de nacimiento, la edad, la ciudad donde nació y su ubicación como jugador en el campo de juego (arquero, defensa, medio campo o delantero). Si bien el jugador tiene su ubicación específica, no obstante, el director técnico en un determinado partido, puede asignarle otra ubicación, la misma que tendrá que ser almacenada en la base de datos.

Para cada partido registrar la fecha, los goles anotados y la lista de jugadores que formaron parte de ese partido. Para los goles registrar el nombre del jugador que marcó el gol y el minuto. Adicionalmente se requiere saber en qué minuto y qué jugador fue amonestado con tarjeta amarilla o roja.

Modelo Entidad – Relación:



Resultado de mapear el esquema ER del caso eliminatorias.

TÉCNICO

Tcódigo	Tnombre	Nacionalidad	FechaNac	CódEquipoDirige

PRENDA

Pcódigo	Pnombre

UNIFORME

CódEquipo	CódUniforme	Tipo	Color

EQUIPO

Código	País

PARTIDO

CódEquipo2	CódEquipo1	Fecha

JUGADOR

Jcódigo	Jnombre	Ciudad	FechaN	CódigoEquip	CódUbicación

UBICACIÓN

Ucódigo	Descripción

JUEGA

CódEquip1	CódEquip2	CódJugador

GOL

CódEqp1	CódEqp2	CódJugad	Minuto

AMONESTADO

CódEqpo1	CódEqpo2	CódJugd	TipoTarjeta	Tminuto

JUEGACOMO

CódigEqpo1	CódigEqpo2	CódJgd	CódUbicación

- Sentencia del comando `CREATE TABLE` para definir el esquema del caso Eliminatorias.

```
create table EQUIPO(
Codigo varchar (10) not null,
Pais varchar (20) not null,
primary key (Codigo)
);

create table PARTIDO(
CodEquipo1 varchar(10) not null,
CodEquipo2 varchar(10) not null,
Fecha date not null,
foreign key (CodEquipo1) references EQUIPO(Codigo) on update cascade on
delete restrict,
foreign key (CodEquipo1) references EQUIPO(Codigo) on update cascade on
delete restrict
);
```

```
create table TECNICO(
Tcodigo varchar(10) not null,
Nacionalidad varchar(20) not null,
Tnombre varchar(50) not null,
FechaNac date not null,
CodEquipoDirige varchar (10) not null,
primary key (Tcodigo),
foreign key (CodEquipoDirige) references EQUIPO(Codigo) on update cascade
on delete restrict
);

create table PRENDA(
Pcodigo varchar (10) not null,
Pnombre varchar (20) not null,
primary key (Pcodigo)
);

create table UNIFORME(
Color varchar (15) not null,
Tipo varchar (20) not null,
CodUniforme varchar (10) not null,
CodEquip varchar (10) not null,
foreign key (CodUniforme) references PRENDA(Pcodigo) on update cascade
on delete restrict,
foreign key (CodEquip) references EQUIPO(Codigo) on update cascade on
delete restrict
);

create table UBICACION(
Ucodigo varchar (10) not null,
Descripcion varchar (150) not null,
primary key (Ucodigo)
);
```

```
create table JUGADOR(

Jcodigo varchar (10) not null,

Ciudad varchar (20) not null,

Jnombre varchar (50) not null,

FechaN date not null,

CodigoEquip varchar (10) not null,

CodUbicacion varchar (10) not null,

primary key (Jcodigo),

foreign key (CodigoEquip) references EQUIPO(Codigo) on update cascade
on delete restrict,

foreign key (CodUbicacion) references UBICACION(Ucodigo) on update
cascade on delete restrict

);


create table JUEGA(

CodJugador varchar (10) not null,

CodEquip1 varchar (10) not null,

CodEquip2 varchar (10) not null,

foreign key (CodJugador) references JUGADOR(Jcodigo) on update cascade
on delete restrict,

foreign key (CodEquip1) references EQUIPO(Codigo) on update cascade on
delete restrict,

foreign key (CodEquip2) references EQUIPO(Codigo) on update cascade on
delete restrict,

);


create table GOL(

minuto int not null,

CodJugad varchar(10) not null,

CodEqp1 varchar(10) not null,

CodEqp2 varchar(10) not null,

foreign key (CodJugad) references JUGADOR(Jcodigo) on update cascade on
delete set null,

foreign key (CodEqp1) references EQUIPO(Codigo) on update cascade on
delete restrict,

foreign key (CodEqp2) references EQUIPO(Codigo) on update cascade on
delete restrict,

);
```

```
create table AMONESTADO(
Tminuto int not null,
TipoTarjeta varchar(10) not null,
CodJugd varchar(10) not null,
CodEqpo1 varchar(10) not null,
CodEqpo2 varchar(10) not null,
foreign key (CodJugd) references JUGADOR(Jcodigo) on update cascade on
delete set null,
foreign key (CodEqpo1) references EQUIPO(Codigo) on update cascade on
delete restrict,
foreign key (CodEqpo2) references EQUIPO(Codigo) on update cascade on
delete restrict,
);

create table JUEGACOMO(
codigoUbicacion varchar(10) not null,
CodJgd varchar(10) not null,
CodigEqpo1 varchar(10) not null,
CodigEqpo2 varchar(10) not null,
FchJueg date not null,
foreign key (codigoUbicacion) references UBICACION(Ucodigo) on update
cascade on delete restrict,
foreign key (CodJgd) references JUGADOR(Jcodigo) on update cascade on
delete set null,
foreign key (CodigEqpo1) references EQUIPO(Codigo) on update cascade on
delete restrict,
foreign key (CodigEqpo2) references EQUIPO(Codigo) on update cascade on
delete restrict,
);
```

16. La Empresa

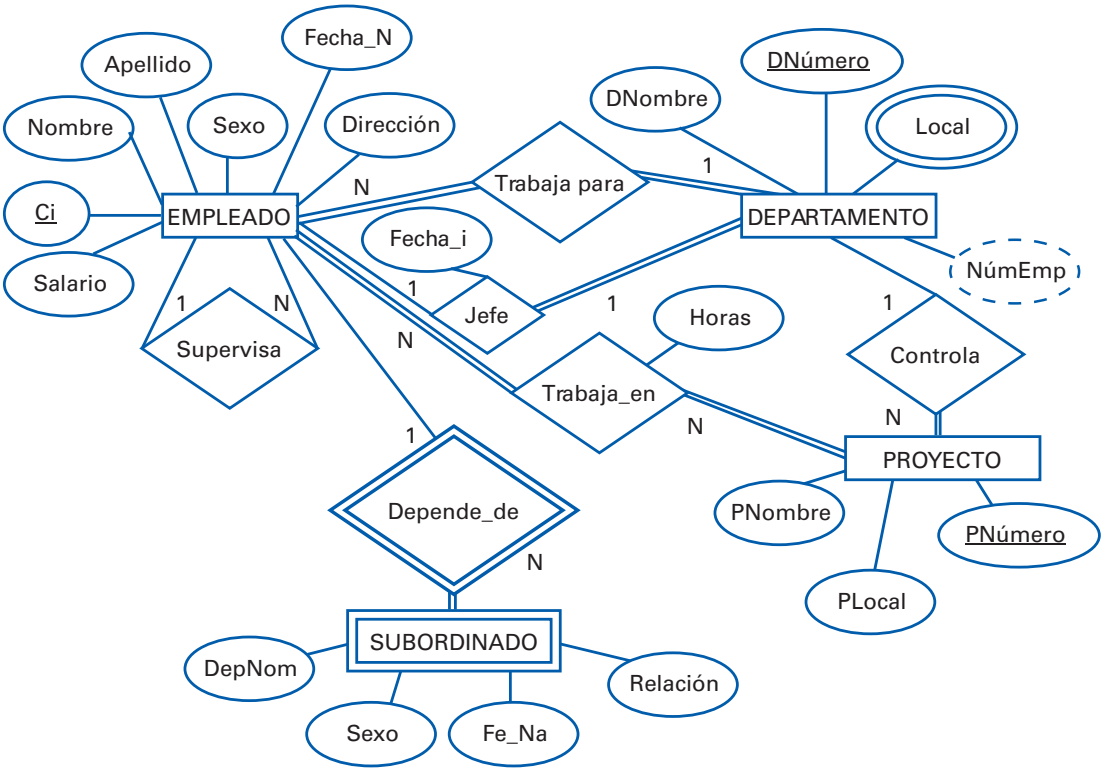
El presente ejemplo corresponde a un caso de estudio del libro "Fundamentos de Sistemas de Bases de Datos" de (Elmasri R., Navathe S., 2016). La empresa, mediante un sistema de información, requiere llevar el control de los empleados, departamentos y proyectos, para lo cual es necesario diseñar la base de datos que almacenará los requerimientos del usuario.

La empresa está organizada en departamentos. Cada departamento tiene un nombre único, un número único, y un empleado en particular que es el jefe del departamento. Se llevará el registro de la fecha en la que el empleado fue designado como jefe del departamento. Un departamento puede tener varias localidades.

En relación con los proyectos, cada uno tiene un número único y un nombre único, y se ejecuta en un solo lugar. Cada departamento controla un cierto número de proyectos. No necesariamente todos los departamentos controlan proyectos.

Los datos que se registrarán para los empleados son los siguientes: el nombre, el apellido, la cédula, la fecha de nacimiento, la dirección, el sexo y su salario. Los empleados trabajan en un departamento en particular, pero pueden trabajar en varios proyectos que no están necesariamente controlados por su mismo departamento. Para cada empleado se registrará además el número de horas semanales que trabaja en cada proyecto. También se llevará el registro del supervisor directo de cada empleado.

De los empleados se tiene que registrar los datos de sus dependientes (subordinados), se almacenarán el nombre de cada persona dependiente, el sexo, fecha de nacimiento, y el parentesco con el empleado.



Resultado de mapear el esquema ER del caso la empresa

EMPLEADO

Nombre	Apellido	Ci	Fecha_N	Dirección	Sexo	Salario	SuperCi	Dno
Juan	Polo	123456789	3-Mar-59	Sucre 7-12	M	3000	333445555	5
Humberto	Pons	333445555	25-Dic-60	Bolívar 5-67	M	4000	888665555	5
Irma	Vega	999887777	13-Nov-50	P. Córdova 3-45	F	2500	987654321	4
Elena	Tapia	987654321	3-May-61	Ordóñez 7-29	F	4300	888665555	4
Pablo	Castro	666884444	15-Sept-55	Bolívar 1-50	M	3800	333445555	5
Marcia	Mora	453453453	29-Mar-60	Colombia 4-23	F	2500	333445555	5
Manuel	Bonilla	987987987	16-Jul-58	B. Malo 1-10	M	2500	987654321	4
Jaime	Pérez	888665555	5-Abr-57	Sangurima 8-34	M	5500	Null	1

DEPARTAMENTO

DNombre	DNúmero	JefeCi	Jefe_Fi
Investigación	5	333445555	12-May-80
Administrativo	4	987654321	5-Dic-82
Compras	1	888665555	6-Jun-78

LOCALIZACIÓN

DNúmero	DepLoca
1	Cuenca
4	Guayaquil
5	Quito
5	Manta
5	Cuenca

TRABAJA_EN

Eci	Pno	Horas
123456789	1	12,5
123456789	2	15,6
666884444	3	14,7
453453453	1	10
453453453	2	10
333445555	2	20
333445555	3	10
333445555	10	10
333445555	20	10
999887777	30	30
999887777	10	5
987987987	10	15
987987987	30	17
987654321	30	10
987654321	20	12
888665555	20	Null

PROYECTO

PNombre	PNúmero	PLocal	Dnum
ProductoX	1	Quito	5
ProductoY	2	Manta	5
ProductoZ	3	Cuenca	5
Computadora	10	Guayaquil	4
Reorganizar	20	Cuenca	1
Beneficios	30	Guayaquil	4

SUBORDINADO

Eci	Dep_Nom	Sexo	Fe_Na	Relación
333445555	María	F	2-feb.-86	Hija
333445555	Teodoro	M	10-oct.-90	Hijo
333445555	Ana	F	15-sep.-65	Cónyuge
987654321	Alberto	M	6-jul.-67	Cónyuge
123456789	Miguel	M	5-nov.-84	Hijo
123456789	María	F	9-ene.-87	Hija
123456789	Elizabeth	F	12-dic.-60	Cónyuge

Ejercicios propuestos

17. Hoteles

Construir un diagrama Entidad - Relación para la gestión de la información hotelera. Los datos almacenados para los hoteles serán su nombre, la dirección, los números telefónicos, el nombre del administrador y la categoría del hotel.

Para cada una de las habitaciones, a más del tipo, se incluirá su tamaño y los servicios que tiene, por ejemplo: baño, teléfono, TV, aire acondicionado, etcétera.

Existen ciertos hoteles que tienen uno o más restaurantes, de los que se tiene que registrar el nombre y su capacidad.

Los hoteles cuentan también con salas de alquiler para ser usadas en diferentes eventos, siendo necesario llevar información del nombre y la superficie de la sala; su capacidad y el precio. Estas dos últimas características dependen del tipo de uso que se le dé a la sala, que entre otros pueden ser eventos para: banquetes, teatro, montaje en U, conferencias, fiestas, etcétera.

Para información de los clientes se almacenarán los tipos de actividades turísticas que ofrecen cada uno de los hoteles. Adicionalmente se tiene que incluir los servicios generales con los que cuenta el hotel, por ejemplo: piscina, gimnasio, bar, parqueadero…

Es necesario que se registren las reservas realizadas por los clientes tanto de las habitaciones como de las salas con sus respectivos eventos.

18. Partiendo del modelo del caso "Eliminatorias", se pide adicionar los siguientes requerimientos: registrar los datos de los estadios en donde se juegan los partidos. Almacenar el nombre y la nacionalidad de los árbitros y registrar para cada partido el árbitro principal y los dos asistentes.

Para efectos de analizar los conceptos de agregación se requiere para los uniformes, registrar los diferentes colores en una entidad denominada COLOR.

19. Partiendo del modelo Entidad - Relación del caso de estudio de La Ferretería, adicionar y modificar de acuerdo a los siguientes requerimientos:

Registrar el porcentaje de descuento que La Ferretería realizará a sus clientes en cada uno de los artículos registrados en la factura.

Los artículos disponibles para la venta cuentan con el código del artículo y el fabricante que elaboró. Un artículo siempre tendrá el mismo código, pero será diferenciado por el fabricante, además el precio será diferente de acuerdo al fabricante. Por ejemplo, el artículo Platina T puede ser construido por diferentes fabricantes A, B, C. Para los fabricantes se tiene que registrar la siguiente información: nombre, dirección y teléfonos.

20. Elaborar un modelo Entidad - Relación y las tablas para el manejo de la información de una librería. Se tendrá que manejar información propia del libro como: el título, el año de publicación, la edición, el ISBN, el o los autores, los idiomas a los que está traducido. Para los autores se tendrá que registrar el nombre y nacionalidad. Para la editorial del libro, se registrará el nombre, su dirección y el URL.

La librería cuenta con varias sucursales de las cuales se tiene que registrar información de su dirección, teléfono, los libros almacenados y el número de ejemplares de cada uno.

Los empleados responsables para cada sucursal no son asignados de manera exclusiva, cada cierto período son cambiados de local. Para el control de la librería es necesario llevar un registro de la fecha en la que un empleado fue asignado a una determinada sucursal.

MODELO RELACIONAL

Este capítulo comienza con una breve introducción e historia del modelo relacional. En la Sección 3.2 se revisan los conceptos fundamentales del modelo: relaciones, atributos, tuplas y dominio. En la Sección 3.3 se analizan los términos relacionados con el esquema y la instancia, tanto de una base de datos como de una relación. La Sección 3.4 trata sobre las propiedades que posee una relación. En la Sección 3.5 se describen las restricciones de integridad de datos, que restringen los valores que pueden ser insertados o creados durante el proceso de actualización de la base de datos. La Sección 3.6 trata sobre la forma cómo se puede mostrar gráficamente el esquema de la base de datos, junto con las dependencias de clave primaria y externa, a través de un diagrama de esquema. En la Sección 3.7 se revisa en forma general las diferentes operaciones relacionales para la manipulación de los datos.

Para la ilustración de los conceptos y ejemplos que se presentan en este capítulo, se utiliza el caso de estudio del mundo real denominado La Ferretería.

3.1 INTRODUCCIÓN

El modelo relacional en la actualidad se ha convertido en el modelo de datos más dominante para las aplicaciones comerciales de procesamiento de datos, debido a su simplicidad y fundamentación matemática, que facilita el trabajo del programador comparando con los modelos anteriores, como el jerárquico y el de red.

El modelo relacional fue propuesto por E.F. Cood (1970) en su artículo original titulado "*A relational model of data for large share data banks*". Está fundamentado en el concepto de una relación matemática y su base teórica, en la teoría de conjunto y la lógica de predicados de primer orden. En el modelo relacional los datos se encuentran almacenados en tablas (registros), a cada una le corresponde un nombre. Las tablas están formadas por columnas, cada cabecera de una columna corresponde a un atributo. Una fila de la tabla recibe el nombre de tupla y cada tupla contiene un valor por cada columna.

Los principales objetivos que se plantearon en el modelo fueron:

- Conseguir la independencia de datos y programas.
- Contar con una base teórica para fundamentar y tratar la semántica de los datos y los problemas de coherencia y redundancia.

- Proporcionar un lenguaje para la manipulación de datos orientados a conjuntos.

En los estudios realizados sobre el modelo relacional, tres proyectos de investigación fueron los más significativos:

El primer proyecto realizado en el laboratorio San José de Research Laboratory de IBM, en California, fue el SGBD prototipo System R., que trajo consigo dos desarrollos principales: el primero el lenguaje de consultas estructurado, convirtiéndose en el lenguaje estándar formal de ISO y el segundo el desarrollo de varios productos comerciales de SGBD relacionales como DB2, Oracle, entre otros.

El segundo proyecto de importancia fue el desarrollo del modelo relacional denominado sistema gráfico interactivo de extracción (INGRES por sus siglas en inglés *Interactive Graphics Retrieval System*). Este proyecto implicaba el desarrollo de un prototipo de SGBDR (sistema de gestión de bases de datos relacional) que condujo a una versión académica de INGRES, y sirvió para la popularidad de los conceptos relacionales, que dieron como resultado el producto comercial INGRES de *Relational Tecnology Inc.*

El tercer proyecto denominado *Peterlee Relational Test Vehicle*, tuvo una visión más teórica que los dos primeros proyectos System R e INGRES y su objetivo se orientaba a la investigación en el procesamiento y optimización de consultas.

3.2 CONCEPTOS DEL MODELO RELACIONAL

Una base de datos relacional está basada en el concepto matemático de **relación**, la cual está representada físicamente en forma de tabla bidimensional, a la que se le asigna un nombre único. Las columnas de la **tabla** corresponden a los **atributos** y las filas de la tabla corresponden a cada registro individual. Por ejemplo, si se analiza la relación `VENDEDOR` que almacena información de todos los vendedores de La Ferretería. La relación tiene siete columnas, *CódigoVend, Vnombre, Apellido, Sexo, FechaContrato, ObjVentas y Teléfono*. Cada fila o **tupla** de la tabla registra información de un vendedor, que consiste en su código, el nombre, su apellido, el sexo, la fecha en la que fue contratado, su objetivo de ventas y su número telefónico. Los atributos de una tabla, pueden estar en cualquier orden y la relación seguirá siendo igual, transmitiendo el mismo significado.

Con un criterio similar al de los atributos, el orden en que aparecen las tuplas en la relación es irrelevante, puesto que la relación es un conjunto de tuplas. Si a la relación `VENDEDOR` se le organiza por el atributo *CódigoVend* o por el *Apellido*, no tiene ninguna importancia, puesto que la relación es la misma ya que los dos criterios de organización contienen las mismas tuplas.

Se define como **dominio** al conjunto de valores permitidos para cada atributo de una relación. Los dominios permiten al usuario definir el origen y el significado de los valores que los atributos pue-

den asumir. Dos o más atributos pueden estar definidos sobre el mismo dominio. Un dominio está conformado por un nombre, un tipo de dato y un formato. En la Figura 3.1 se muestran los dominios para algunos de los atributos de la relación *Vendedor*.

Atributo	Nombre de dominio	Significado	Definición del dominio
Vnombre	Nombre del cliente	Conjunto de caracteres que representa el nombre de un vendedor.	Carácter: tamaño 20
CódigoVend	Código del vendedor	Conjunto de todos los posibles códigos de un vendedor.	Entero
Sexo	Sexo	Sexo del vendedor	Carácter: tamaño 1, valor "M" o "F"
FechaContrato	Fecha de contrato	Posible fecha en la que el vendedor fue contratado	Fecha: formato dd-mmm-aa
ObjVentas	Objetivo de ventas	Posibles objetivos de venta de un vendedor	Numérico: tamaño 8,2

Figura 3.1. Dominios para atributos de la relación `VENDEDOR`.

A los significados del dominio se lo conoce también como *lógicas de dominio* (Elmasri R., Navathe S., 2016). Un dominio es **atómico** si los elementos del mismo se consideran unidades indivisibles. Por ejemplo, el atributo *Teléfono* de la relación `VENDEDOR`; si se trata cada número de teléfono como un valor único e indivisible, entonces el atributo *Teléfono* tiene un dominio atómico, sin embargo, si al valor del número telefónico se le divide en código internacional del país, código del área y número del abonado, se consideraría un atributo con dominio no atómico.

Se denomina **grado** de una relación, al número de atributos o columnas que contiene. Por ejemplo, la relación `VENDEDOR`, tiene siete atributos, por lo que su grado es siete. A una relación con grado uno se la denomina unaria, si es de grado dos se la llama binaria y a la de tres, terciaria; si el grado es superior a tres, generalmente se utiliza el término n-aria.

La **cardinalidad** es otro concepto que forma parte del modelo relacional y se define como el número de tuplas que contiene una relación. La cardinalidad cambia permanentemente en la base de datos, en la medida en que se incrementen o borren tuplas. Se utiliza el término **ejemplar de relación** para referirse a una relación que contiene un determinado conjunto de filas, es decir a una instancia específica de una relación. En la Figura 3.2 se muestran la relación `VENDEDOR` y sus diferentes componentes.

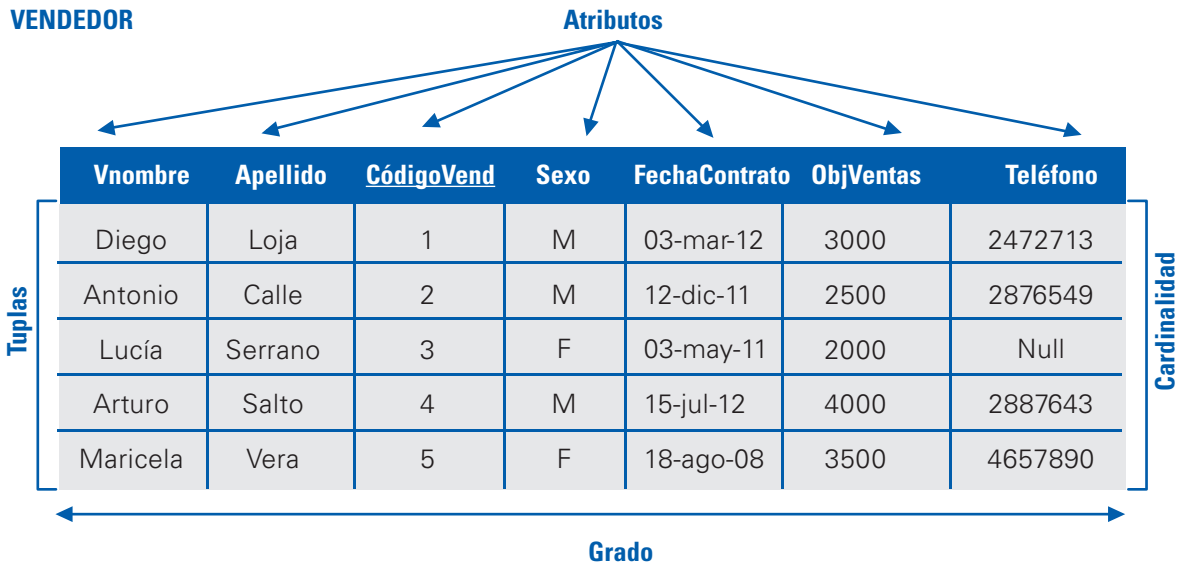


Figura 3.2. Instancia o ejemplar de la relación VENDEDOR.

En los párrafos anteriores se hizo referencia a dos terminologías del modelo, las relaciones o tablas, los atributos o columnas y las tuplas o filas, sin embargo, desde el punto de vista físico el SGBDR almacena cada relación en un archivo, razón por la cual a una relación se la puede llamar también archivo, a una tupla se la denomina registro y a un atributo se lo puede llamar campo. En la Figura 3.3 se resumen las diferentes terminologías utilizadas en el modelo relacional.

Terminos formales	Alternativa 1	Alternativa 2
Relación	Tabla	Archivo
Tupla	Fila	Registro
Atributo	Columna	Campo

Figura 3.3. Terminologías alternas del modelo relacional (Connolly T., Begg C., 2005).

3.3 ESQUEMA Y EJEMPLAR DE LA BASE DE DATOS

En una base de datos se tiene que diferenciar entre el **esquema de la base de datos** y el **ejemplar de una base de datos.** El primero representa el diseño lógico de la misma y corresponde a una colección de esquemas de relaciones, sobre los cuales se encuentran definidas las restricciones de integridad. El segundo se refiere a una instancia de los datos en un momento determinado.

Por otra parte, el **esquema de una relación** representa los atributos y el dominio asociado a cada uno de ellos. El esquema de una relación normalmente no cambia. Se utiliza el término **ejemplar de relación** para referirse a una relación que contiene un determinado conjunto de filas, es decir, a una instancia específica de una relación que puede cambiar con el tiempo, cuando la relación se actualice. Este concepto tiene su correspondencia con el significado de una variable que se utiliza en el lenguaje de programación.

Por ejemplo, para registrar los datos de cada uno de los clientes de La Ferretería, como: el nombre, los apellidos, la cédula, la dirección, el límite de crédito asignado, si fue recomendado o no por otros clientes, quien es su representante y la fecha en la que fue asignado; se necesita crear una relación que almacene los valores de cada uno de los atributos. Al esquema de la relación creado se le denomina `CLIENTE` y está definido de la siguiente manera:

`CLIENTE` *(Nombre, Apellido1, Apellido2, Cédula, Dirección, LímCrédito, CédulaReco, CódRepre, FechaAsig)*

El ejemplar de la relación `CLIENTE` se muestra en la Figura 3.4.

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CedulaReco	CódRepre	FechaAsig
Victor	Castro	Torres	0102030405	Sucre 1-12	600	0122334455	1	13-Oct-12
Juan	Polo	Ávila	0122334455	Bolívar 5-67	1100	0456456456	1	2-Mar-13
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500	0777888999	Null	4-Jun-12
Pablo	Brito	Parra	0123456789	Alfaro 1-23	Null	0456456456	3	15-Dic-11
Marcia	Mora	Durazno	0987654321	Olmedo 1-10	1000	0777888999	1	23-Dic-12
Jaime	Pérez	Guerrero	0777888999	Colón 4-98	1200	0122334455	2	2-Oct-12
Humberto	Pons	Coronel	0123123123	Borrero 3-45	2000	Null	3	10-Sept-13
Alberto	Torres	Viteri	0999998888	Luque 10-10	1200	0122334455	4	13-Ene-13
Manuel	Bonilla	León	0456456456	Montalvo 5-98	600	0122334455	2	12-Feb-12

Figura 3.4. Ejemplar o instancia de la relación `CLIENTE`.

Para el caso de la relación `VENDEDOR`, de la Figura 3.2. El esquema está definido por:

VENDEDOR (Vnombre, Apellido, CódigoVend, Sexo, FechaContrato, ObjVentas, Teléfono)

Nótese que cada una de estas relaciones tiene un identificador, a cada cliente se le identifica por el valor del atributo *Código*, y a cada vendedor por el valor del atributo *CódigoVend*. De otra parte los atributos *CódigoVend* y *CódRepre* de las relaciones `VENDEDOR` y `CLIENTE` respectivamente son

atributos comunes y sirven para relacionar tuplas de relaciones diferentes. Por ejemplo, si se requiere buscar información de los clientes que tienen como representante a Diego Loja. En primer lugar se busca en la relación `VENDEDOR` el valor del código correspondiente a Diego Loja, para nuestro ejemplo el valor es de 1, a continuación, con estos datos se busca en la tabla `CLIENTE` las coincidencias con los valores de la columna *CódRepre*, que corresponden a los clientes representados por el vendedor especificado.

Diversas son las relaciones que forman parte del modelo relacional del caso de estudio La Ferretería. Adicionalmente a las relaciones antes indicadas se incluyen las siguientes:

- *CARGAFAMILIAR (NomDep, Sexo, FechaNac, Relación, CódVendedor)*
- *ARTÍCULO (Código, Descripción, Precio, Unidad, ExiMáx, ExiMín)*
- *FACTURA (Número, Fecha, CódVendedor, CcCliente)*
- *CLIENTE_TELÉF (CédulaCli, NúmeroTeléf)*
- *DETALLE (FactNúmero, ArtCódigo, Cantidad)*

3.4 PROPIEDADES DE UNA RELACIÓN

3.4.1 Valor atómico

Cada celda de la relación (fila, columna) contiene exactamente un valor único (atómico), es decir no es divisible. Bajo este concepto no son permitidos los dos tipos de atributos que se analizaron en el Capítulo 2; los atributos compuestos que en el modelo relacional se representarán como simples atributos y los atributos multivalor que deben representarse mediante relaciones separadas. Este concepto de valor atómico fue desarrollado bajo el principio de **primera forma normal,** tema que se tratará en el Capítulo 6 cuando se revisen las formas normales.

Por las características de atomicidad antes descritas, el modelo relacional también recibe el nombre de **modelo relacional plano**. (Elmasri R., Navathe S., 2016).

3.4.2 Orden de tuplas

La definición de una relación no especifica ningún orden para las tuplas, puesto que las mismas están definidas como un conjunto y desde el punto de vista matemático, los elementos de un conjunto

no tienen un orden entre ellos. Una relación puede tener diferentes criterios de ordenamiento, por ejemplo, la relación ARTíCULO de la Figura 3.5 está ordenada de forma ascendente por el atributo *Código*, sin embargo, podría ordenarse lógicamente por *Descripción, Precio, Unidad,* o por cualquier otro atributo, sin que se especifique una preferencia por un orden con respecto a otro.

Teóricamente el orden de las tuplas no tiene importancia, no obstante, en la práctica el orden puede afectar a la eficiencia en el acceso a las tuplas. (Connolly T., Begg C., 2005)

ARTÍCULO

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín
1	Cemento	6,0	saco	50	10
2	Clavos	4,8	kilo	70	5
5	Alambre	3,0	libra	20	5
6	Perfil T	15,0	unidad	10	3
7	Perfil L	15,5	unidad	15	5
8	Perfil G	16,0	unidad	10	3
9	Pintura	19,5	galón	15	4
10	Lija	0,6	unidad	30	10
11	Suelda	20,0	kilo	10	5

Figura 3.5. Relación ARTÍCULO ordenada por el atributo *Código*.

3.4.3 Orden de atributos

A nivel lógico el orden de los atributos y sus respectivos valores no son importantes, siempre que se mantenga la correspondencia entre ellos. Para ejemplificar que en una relación es innecesaria la ordenación de los valores de los atributos en una tupla se considera que una tupla es un conjunto de pares (<atributo> , <valor>), que representan la asignación de un valor a un atributo, por ejemplo (descripción, cemento). Bajo este criterio la ordenación de los atributos en una relación no es importante, debido a que el nombre del mismo aparece acompañado de su valor. En la Figura 3.6 se presentan dos tuplas que son idénticas, a pesar de que el par (<atributo> , <valor>) están en desorden.

Tupla = (Código, 1),(Descripción, Cemento),(Precio, 6,0),(Unidad, saco),(ExiMáx, 50),(ExiMín, 10)

Tupla = (Descripción, Cemento),(Unidad, saco),(Código, 1),(ExiMín, 10),(Precio, 6,0),(ExiMáx, 50)

Figura 3.6. Dos tuplas idénticas con los atributos en desorden.

Adicionalmente a las propiedades antes descritas, una relación debe cumplir con las siguientes características:

- En el esquema relacional, cada tabla (relación) tienen un nombre.
- Una relación no contiene tuplas duplicadas, cada tupla es diferente.
- Cada uno de los atributos tiene un nombre único.
- Los valores para cada atributo deben pertenecer al mismo dominio.

3.5 RESTRICCIONES DE INTEGRIDAD

En la Sección 1.6 se revisó los componentes de un modelo de base de datos, siendo uno de ellos, el conjunto de restricciones (*constraints*) de integridad que aseguran la precisión de los datos en un estado de la base de datos. Estas restricciones dependen de las reglas del mundo real, que deben ser representadas en la base de datos.

En esta sección se examinará las diferentes restricciones que pueden expresarse directamente en el esquema de un modelo relacional, a través del lenguaje de definición de datos LDD. Entre las principales se incluye las restricciones de dominio, las de clave, las restricciones en valor nulo. Existen también dos reglas de integridad importantes, que se aplican a todas las instancias de la base de datos y se conocen con los nombres de integridad de entidad e integridad referencia. Adicionalmente en esta sección se revisarán otras restricciones que no se pueden aplicar directamente en los esquemas del modelo de datos, las que se denominan restricciones generales.

Otro importante tipo de restricción de integridad son las de dependencia de datos, tema que se analizará en el Capítulo 6 cuando se revisen los procesos de normalización, y corresponden básicamente a las restricciones de dependencia funcional y dependencia multivalor.

3.5.1 Restricción de dominio

En la Sección 3.2 se introdujo el concepto de dominio como la asociación de los valores posibles a cada atributo. Los tipos de datos asociados a ellos pueden incluir números enteros o reales, caracteres, cadenas de longitud fija o variable, valores lógicos, monetarios, de fecha y hora.

La declaración de un atributo como parte de un dominio concreto, que limita al conjunto de valores para los atributos de las relaciones, actúa como restricción y se denomina **restricción de dominio.**

Ésta es la más elemental de las restricciones de integridad y es comprobada mediante el sistema cuando se ingresa un nuevo dato a la base de datos.

3.5.2 Claves

Como se indicó anteriormente, en el modelo relacional no se permiten tuplas duplicadas en una relación, es decir, que tengan los mismos valores en todos sus atributos, por tanto, es necesario determinar un conjunto de uno o más atributos que identifiquen de manera unívoca cada tupla de una relación.

Se define como **superclave** a un atributo o conjunto de atributos que identifican de forma unívoca una tupla de la relación. Por ejemplo, el atributo *Código* de la relación `ARTÍCULO` es una superclave, porque es suficiente para distinguir un artículo de otro; sin embargo, el atributo *Descripción* de un artículo no es una superclave, porque podrían existir varios artículos con el mismo nombre. Una superclave también puede estar formada por un conjunto de atributos. Por ejemplo, la combinación de los atributos *Código* y *Descripción* es una superclave para la relación `ARTíCULO`. Bajo este concepto, toda relación tiene al menos una superclave predeterminada, formada por el conjunto de todos sus atributos.

La superclave cuenta con atributos redundantes, es por esta razón que autores como (Elmasri R., Navathe S., 2016) consideran más importante el concepto de clave. La **clave** de una relación es una superclave de ésta, con la propiedad adicional que eliminando cualquier atributo del conjunto que forma la superclave, ésta pierde su condición de superclave.

Esta restricción recibe el nombre de **superclave mínima**, es decir, de una superclave no se puede eliminar ningún atributo, caso contrario, perderá su condición de identificar unívocamente una tupla de la relación. Por ejemplo, en la relación `ARTÍCULO` de la Figura 3.5, el conjunto de atributos {Código} es una clave, porque dos tuplas de artículos distintos no pueden tener el mismo valor para el código (se recuerda que *Código* es también una superclave). Cualquier conjunto de atributos que incluya al *Código* es una superclave, por ejemplo {*Código*, *Descripción*, *Precio*}; sin embargo, esta superclave no es una clave de `ARTÍCULO`, porque si se elimina la descripción, el precio o ambos, ésta no deja de ser una superclave. En términos generales una superclave formada por un solo atributo, también es una clave.

La condición de clave debe mantenerse fija en el tiempo, es decir, al insertar nuevos valores en el atributo clave, estos tienen que guardar la condición de unicidad de las tuplas. Por ejemplo, en la relación `CLIENTE` no se podría considerar al atributo Nombre como clave, porque es posible que en algún momento dos clientes tengan nombres idénticos.

Una relación puede tener más de una clave, y cada una recibirá el nombre de **clave candidata**. Por ejemplo, si a la relación `VENDEDOR` se le incluye el atributo *CédulaCiudadanía*, cuyo domino es el conjunto de dígitos que componen el número de identificación, en este caso, la relación tendrá dos claves candidatas: *CódigoVend* y *CédulaCiudadanía*.

Para la designación de una clave candidata no puede utilizarse la instancia de una relación, para demostrar que un atributo o un conjunto de atributos cumplen con la condición de identificador único. Si en una instancia determinada no hay duplicados en los valores, esto no garantiza que en algún momento se presenten valores duplicados. Por el contrario, la presencia de valores repetidos en una instancia concreta, sí garantiza que al atributo o conjunto de atributos no se consideren como una clave candidata.

El término **clave primaria** es utilizado para denotar la clave candidata que fue seleccionada para identificar las tuplas de una relación. Las claves candidatas que no fueron seleccionadas como claves primarias, reciben el nombre de **claves alternas.** Por ejemplo, para el caso de la relación `VENDEDOR`, que se analizó en el párrafo anterior, que cuenta con las claves candidatas *CódigoVend* y *CédulaCiudadanía*. Si se selecciona al atributo *CódigoVend* como clave principal, el atributo *CédulaCiudadanía* sería la clave alterna.

Para escoger una clave primaria se recomienda que los valores de sus atributos no se modifiquen nunca o muy rara vez. Por ejemplo, no es recomendable definir la dirección de un cliente como clave primaria, puesto que ésta se modificará cuando el cliente se cambie de dirección; por el contrario, una buena alternativa a ser considerada como clave primaria, es el número de la cédula de ciudadanía del cliente, que no cambia nunca. Si bien la elección de una clave candidata como clave primaria es algo arbitrario, no obstante, es preferible elegir aquella que tenga un solo atributo, o un pequeño número de ellos.

Para la relación `DETALLE` que se muestra en la Figura 3.7, sólo hay una clave candidata, compuesta por los atributos {*FacNúmero*, *ArtCódigo*}, por lo que este conjunto de atributos pasará a ser automáticamente la clave primaria. Para representar la clave primaria en una relación, el atributo o conjunto de atributos que forman parte de la clave, van subrayados.

DETALLE

FactNúmero	ArtCódigo	Cantidad
100	2	5
100	9	1
100	10	5
101	6	4
101	8	5
102	5	6
103	11	1
...	...	...
110	10	3
110	2	1
111	6	5
111	9	2

Figura 3.7. Instancia de la relación `DETALLE`.

Cuando en una relación aparecen entre sus atributos la clave primaria de otra relación (en ocasiones puede ser la misma relación), su aparición suele representar algún tipo de correspondencia entre las tuplas de las dos relaciones. Este atributo o conjunto de atributos que se refiere a una clave primaria se denomina clave externa. Por ejemplo, el atributo *CódRepre* de `CLIENTE`, es una **clave externa** que hace referencia a la clave primaria *CódigoVend* de la relación `VENDEDOR`.

En ciertos casos la clave externa de una relación hace referencia a la clave primaria de la misma relación. Por ejemplo, si se requiere expresar la correspondencia entre los clientes de La Ferretería que fueron recomendados por otros clientes, en este caso, el atributo *CédulaReco* de `CLIENTE`, es una clave externa que hace referencia a la clave primaria *Cédula* de la misma relación `CLIENTE`.

En términos generales, "el esquema de una relación por ejemplo r1, puede incluir entre sus atributos la clave primaria de otra relación, por ejemplo r2. La relación r1 se denomina **relación referenciante** de la dependencia de clave externa, y r2 se denomina **relación referenciada** de la clave externa". (Silberschatz , A. Korth, H. Sudarshan, S., 2014)

3.5.3 Valores nulos

En la relación VENDEDOR que se muestra en la Figura 3.2, el valor NULL de la columna *Teléfono* significa que para la vendedora Lucía Serrano el valor del teléfono no es conocido o no existe. Los valores NULL pueden tener varias representaciones: *valor desconocido, valor existente pero no disponible o el de un atributo no aplicable a una tupla*. Para ilustrar este último caso se considera que los clientes de La Ferretería son nacionales y extranjeros. Esto implica incluir un atributo adicional *Pasaporte* a la relación CLIENTE, en el que se registrará el número del pasaporte de los extranjeros, pero este atributo no aplicaría para los clientes nacionales.

El nombre de valor nulo está sujeto a controversia, algunos autores prefieren llamarlo "nulo" en lugar de "valor nulo", debido a que nulo representa la ausencia de valor.

Sin la incorporación del concepto de valores nulos el usuario estaría obligado a introducir datos falsos para representar cualquiera de los significados de nulo, datos que podrían ser no significativos y traerían confusión al momento de procesarlos. Por ejemplo, para el caso de que no exista el número de teléfono, se podría representar el nulo con el valor de "no" o para el caso de que el valor existe pero no esté disponible, se podría representar como "pendiente".

El uso de valores nulos puede causar en el modelo problemas de implementación, que se presentan debido a que el modelo relacional está basado en la lógica booleana, que permite únicamente los valores verdadero o falso. Si se permiten adicionalmente valores nulos, se tendría que considerar una lógica de orden superior, como la lógica trivaluada.

Los valores nulos pueden generar ambigüedades. Por ejemplo, en la relación CLIENTE, los clientes Víctor Castro y Humberto Pons tienen registrado un valor NULL en el atributo *Dirección*, esto significa que ambos clientes comparten ese valor de dirección.

En el diseño de base de datos es recomendable en lo posible evitar el uso de valores nulos.

3.5.4 Integridad de entidad

La restricción de integridad de entidad en una relación declara que ningún atributo de una clave primaria puede ser nulo, esto se debe a que por definición, una clave primaria se utiliza para identificar de manera unívoca las tuplas y si se permite un valor nulo, implica que no se podría identificar ciertas tuplas. Por ejemplo, al ser el atributo *Código* la clave primaria de la relación ARTíCULO, no se podría insertar una tupla en la relación ARTíCULO que tenga un atributo *Cédula* con valor nulo.

3.5.5 Integridad referencial

A diferencia de las restricciones de integridad de entidad y la de clave, que se especifican en relaciones individuales, la restricción de integridad referencial se especifica entre dos relaciones y garantiza que un valor que aparece en una relación para un conjunto de atributos, también aparece para un determinado conjunto de atributos en otra relación. Por ejemplo, si se considera la base de datos de la Figura 3.8., en la relación `CLIENTE`, el atributo *CódRepre* (clave externa) hace referencia al atributo *CódVendedor* (clave primaria) de la relación `VENDEDOR`. Esta correspondencia expresa que un valor de *CódRepre* en cualquier tupla de la relación `CLIENTE`, debe coincidir con otro del atributo *CódVendedor* en alguna tupla de `VENDEDOR`, o puede ser `NULL` si el cliente no tiene un representante o será asignado posteriormente. En el lenguaje de consultas estructurado a la clave externa se la denomina *foreign key* y a la clave principal *primary key*.

Las restricciones de integridad referencial en el diagrama se pueden mostrar mediante un arco que parte de la *foreign key* y termina con la punta de flecha en la relación referenciada que contiene la *primary key*. En el Capítulo 5 se aborda con mayor detalle los conceptos de integridad referencial.

3.5.6 Restricciones generales

En el mundo real se presentan ciertas reglas adicionales definidas por el usuario, que restringen algunos aspectos de la organización, llamadas a veces *restricciones* de *integridad semántica.* Por ejemplo, si en nuestro caso de estudio La Ferretería, se especifica la condición o restricción que un cliente para acceder a un límite de crédito debe haber adquirido mínimo cinco facturas con un monto superior a 1.000 dólares. Un segundo ejemplo podría considerarse el hecho de que un vendedor puede tener máximo seis clientes a quien representa. Este tipo de restricciones, el usuario puede implementarlas dentro de la aplicación que actualiza la base de datos o utilizar el mecanismo de *aserciones* del SGBD. Otra función relacionada con este tipo de restricciones son los triggers, que se utilizan para especificar la acción que se realizará cuando se cumpla cierta condición o se produzca cierto evento. Por ejemplo, si la existencia en bodega de un determinado artículo alcanzó su límite mínimo, el SGBD activará un *triggers* (disparador) con un mensaje al usuario o con la acción de ejecutar un procedimiento determinado.

3.6 DIAGRAMA DEL ESQUEMA DE UNA BASE DE DATOS

El esquema de la base de datos junto con las dependencias de clave primaria y externa, se puede representar gráficamente mediante un diagrama de esquema. En la Figura 3.8 se muestra el diagrama de esquema de nuestro caso de estudio La Ferretería, en el que se describe el nombre de cada relación con sus atributos. El atributo o conjunto de atributos que forman la clave primaria (*primary key*) se expresa subrayado. La flecha que aparece desde el atributo de la clave externa hasta la clave primaria, representa la dependencia de clave externa.

A continuación se analiza el nombre que se le asigna a un atributo. En la Figura 3.8, el código del vendedor definido en las relaciones **FACTURA** y **CARGAFAMILIAR**, con igual nombre de atributo *CódVendedor*, representa el mismo concepto del mundo real; sin embargo, en las relaciones **CLIENTE** y **VENDEDOR**, a este concepto se le asigna otros nombres de atributos *CódRepre* y *CódigoVend* respectivamente. Estos atributos que tienen el mismo concepto del mundo real pueden adoptar o no iguales nombres en diferentes relaciones. Por otra parte, para los atributos que representan diferentes conceptos, es posible asignarles el mismo nombre en diferentes relaciones. Por ejemplo, el atributo *Sexo* en la relación **VENDEDOR** tiene el mismo nombre que el atributo *Sexo* en la relación **CARGAFAMILIAR**, sin embargo, en el primer caso hace referencia al sexo del vendedor y en el segundo al sexo de la carga familiar.

En un inicio el modelo relacional sugirió que el nombre de la clave externa fuera idéntico al de la clave primaria, pero este criterio ha creado problemas cuando la correspondencia se crea en la misma tabla. Por ejemplo, para nuestro caso de estudio, el concepto cédula de ciudadanía aparece dos veces en la relación **CLIENTE**, una como cédula del cliente y otra como cédula del cliente que recomienda. A estos atributos, para distinguirlos se los asigna el nombre de *Cédula* y *CédulaReco* respectivamente.

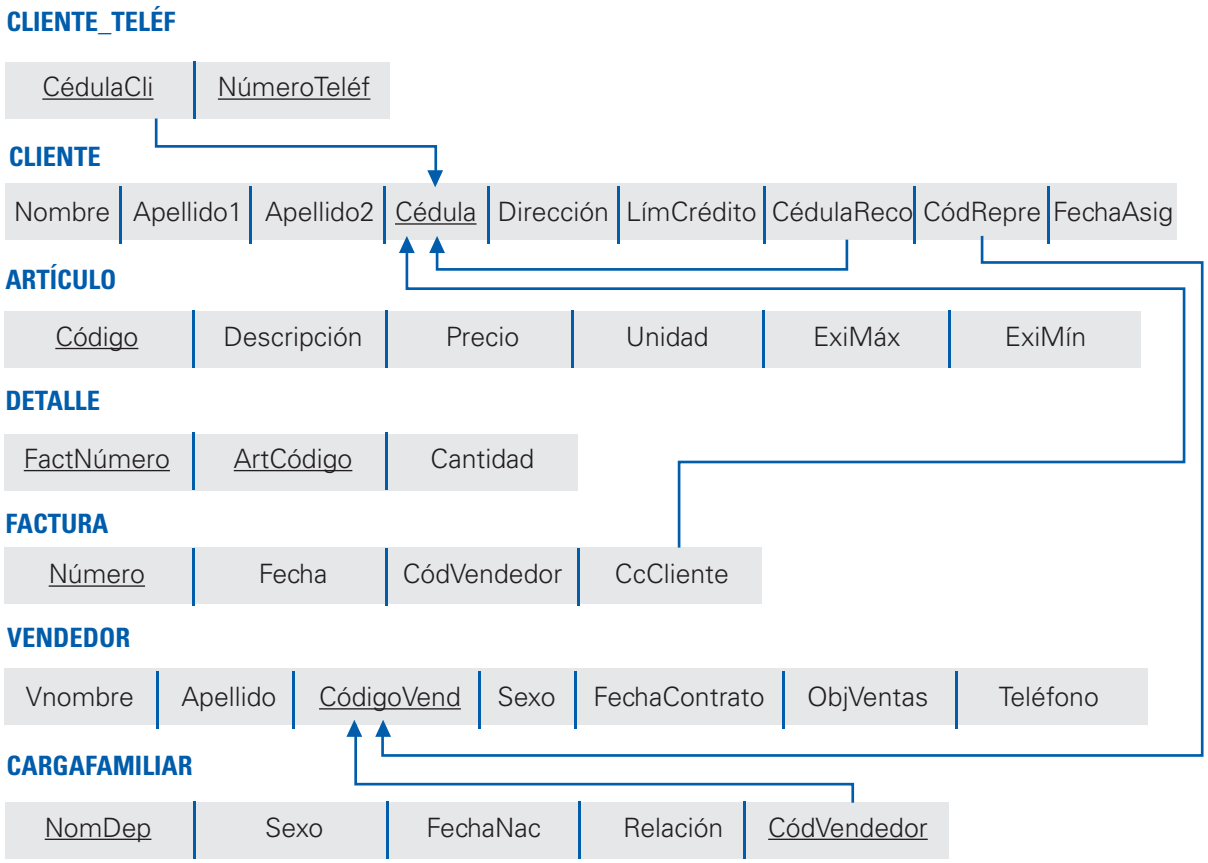


Figura 3.8. Diagrama de esquema para la base de datos de La Ferretería.

3.7 OPERACIONES RELACIONALES

La parte manipulativa es otro de los componentes de un modelo de datos que se revisó en la Sección 1.6, que define los tipos de operaciones sobre los datos. Los lenguajes de consulta relacional de tipo procedimentales proveen un conjunto de operaciones que actúan sobre relaciones o pares de relaciones, obteniendo como resultado una única relación. A este resultado se le puede aplicar operaciones de la misma forma que a las relaciones originales de la base de datos.

En esta sección se trata, de forma general, las diferentes operaciones relacionales, las que serán revisadas a detalle en el Capítulo 4.

La operación usada con mayor frecuencia es la selección de tuplas de una relación que cumpla con cierta condición. Por ejemplo, si de la relación `CLIENTE` que se muestra en la Figura 3.4 se requiere un reporte de todos los clientes que tengan un límite de crédito mayor a 1.100 dólares (*LímCrédito* > 1.100). El resultado que se muestra en la Figura 3.9, es una nueva relación y ésta es un subconjunto de la relación original `CLIENTE`.

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CédulaReco	CódRepre	FechaAsig
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500	0777888999	Null	4-Jun-12
Jaime	Pérez	Guerrero	0777888999	Colón 4-98	1200	0122334455	2	2-Oct-12
Humberto	Pons	Coronel	0456456456	Borrero 3-45	2000	Null	3	10-Sept-13
Alberto	Torres	Viteri	0999998888	Luque 10-10	1200	0122334455	4	13-Ene-13

Figura 3.9. Resultado de la consulta con la operación selección.

Otra operación frecuente es la selección de ciertos atributos de una relación, que genera una nueva relación con los atributos especificados. Por ejemplo, para obtener un listado de los clientes, únicamente con el nombre, el primer apellido, el número de cédula y el límite de crédito. La nueva relación contiene todas las tuplas de la relación `CLIENTE`, pero únicamente con las columnas seleccionadas, como se muestra en la Figura 3.10.

Nombre	Apellido1	Cédula	LímCrédito
Víctor	Castro	0102030405	600
Juan	Polo	0122334455	1100
Elena	Tapia	0111555666	1500
Pablo	Brito	0123456789	Null
Marcia	Mora	0987654321	1000
Jaime	Pérez	0777888999	1200
Humberto	Pons	0456456456	2000
Alberto	Torres	0999998888	1200
Manuel	Bonilla	0123123123	600

Figura 3.10. Resultado de la consulta que contiene los atributos seleccionados.

También forman parte de las operaciones relacionales, las operaciones de la teoría de conjuntos: *unión, intersección y diferencia*, que se aplican sobre dos relaciones de estructuras similares.

Otra operación de uso frecuente es la *reunión*, que genera una única relación a partir de dos relaciones que combinan pares de tuplas, una de cada relación. Por ejemplo, para obtener un reporte del nombre del vendedor junto con la información de sus cargas familiares. El resultado de esta consulta se muestra en la Figura 3.11. Existen diferentes formas de reunión que se detallan en el Capítulo 4.

Vnombre	Apellido	NomDep	Sexo	FechaNac	Relación	CódVendedor
Diego	Loja	María	F	27-Oct-04	Hija	1
Diego	Loja	Isabel	F	3-Dic-12	Hija	1
Antonio	Calle	Antonio	M	16-Abr-02	Hijo	2
Antonio	Calle	Maricela	F	18-May-92	Cónyuge	2
Lucía	Serrano	Teodoro	M	25-Oct-88	Cónyuge	3

Figura 3.11. Resultado de la operación reunión entre `VENDEDOR` y `CARGAFAMILIAR`.

El **producto cartesiano** es otra operación relacional que permite combinar información de dos relaciones cualquiera. La diferencia con la operación reunión radica en que el resultado contiene todos los pares de tuplas de las dos relaciones, sin considerar si sus atributos coinciden o no.

Se denomina álgebra relacional al lenguaje de consultas procedimental, que consta de un conjunto de operaciones relacionales, que toma una o dos relaciones de entrada y genera una de salida como resultado. Este lenguaje es la base del lenguaje de consultas estructurado SQL que se revisa en el Capítulo 5.

3.8 CUESTIONARIO Y EJERCICIOS

Preguntas de repaso

1. Explique los conceptos del modelo de datos relacional: relación, atributo, tupla, dominio, grado y cardinalidad.

2. Explique la diferencia entre esquema y ejemplar de la base de datos.

3. Explique la diferencia entre esquema y ejemplar de una relación.

4. Explique las propiedades de una relación.

5. ¿Por qué hay tuplas sin orden en una relación?

6. Defina los términos superclave, clave, clave candidata y clave alterna.

Ejercicios resueltos

7. Diseñar una base de datos para el registro de la información de un sistema de carreteras, en función de las siguientes consideraciones[1]:

- Las carreteras se encuentran divididas en tramos.
- Un tramo pertenece siempre a una carretera y no puede cambiar de carretera.
- Adicionalmente los tramos pueden pasar por varios terminales municipales, debiendo registrarse dos datos de interés, el kilómetro de entrada y el kilómetro de salida de dicho término municipal.
- Los tramos se agrupan en áreas y cada uno de los cuales no puede pertenecer a más de un área.

[1] Tomado de los apuntes del curso de Gestión de bases de datos del profesor Humberto Ramos [2002]. CEPADE – Universidad Politécnica de Madrid

Solución:

MUNICIPIO (*CódigoMunicipio, Nombre)*

CARRETERA *(CódigoCarretera, NombreCarr)*

TRAMO *(CódTramo, Descripción, CódigoCarretera, CódigoÁrea)*

ÁREA *(CódigoÁrea, DescripciónÁrea)*

TÉRMINOMUNICIPIO *(CódTramo, CódigoMunicipio, KmEntrada, KmSalida)*

Ejercicios propuestos

8. Considere la siguiente base de datos relacional e indique para cada relación las superclaves, claves primarias y externas apropiadas. En caso de existir, determine las claves candidatas y alternas. Explique sus decisiones.

MATERIA *(NombreMateria, NúmeroCréditos)*

ESTUDIANTE *(Cédula, Nombre, Apellido, Fecha_Ing, N_Matrícula)*

CALIFICACIÓN *(NombreMateria, Cédula, Nota)*

9. Diseñar una base de datos para la gestión de la información de la secretaría de la universidad, que requiere llevar un registro de las notas de los aportes (uno o más de uno), de los exámenes finales y exámenes supletorios de cada una de las materias que cursan los estudiantes.

Se requiere además saber, en qué facultad y escuela se imparten las materias, cuál es el número de créditos, a qué nivel de la carrera corresponde y el nombre del profesor o de los profesores que imparten la materia.

10. En función de las relaciones VUELO_PILOTO y PILOTO, determinar las superclaves, claves primarias, candidatas, foráneas y alternas. Argumente todas sus decisiones.

VUELO_PILOTO

Nro_Vuelo	Cód_piloto	Fecha

PILOTO

Cód_piloto	Nom_piloto	Total_H_Vuelo

11. En función de las siguientes relaciones determine las superclaves, claves primarias, foráneas, candidatas y alternas. Rellene las relaciones con datos, generando varias tuplas para comprobar su decisión.

ARTÍCULO1

Fabricante	Descripción

LOCAL

Estantería	Bodega

FABRICANTE2

Fabricante	Nro_Casilla	Estantería

ÁLGEBRA RELACIONAL

En los capítulos anteriores se revisó el fundamento conceptual del modelo Entidad - Relación y del modelo relacional. En función de esta base teórica, en este capítulo se tratará las técnicas de gestión de datos, que mediante un conjunto de operaciones del álgebra relacional, permite consultar y actualizar los datos subyacentes.

Se examinará el concepto y simbología de cada una de las operaciones, para luego ilustrar su funcionalidad a través de ejemplos y ejercicios prácticos que se presentarán al final del capítulo. En la Sección 4.1 se analiza los conceptos del álgebra relacional y su analogía con la teoría de conjuntos. La Sección 4.2 se centra en las operaciones unarias que operan en relaciones individuales. En la Sección 4.3 se examinan las operaciones de conjuntos `UNIÓN, INTERSECCIÓN, DIFERENCIA y PRODUCTO CARTESIANO` aplicadas a una base de datos. En la Sección 4.4 se analizan las operaciones binarias que operan sobre dos tablas, en tanto que en la Sección 4.5 se presentan las operaciones de agregación y el cierre recursivo. El capítulo concluye con ejercicios resueltos y propuestos, que se presentan en la Sección 4.6.

En el capítulo anterior se revisó la terminología del modelo relacional: una relación recibe el nombre de *tabla*, una cabecera de columna corresponde a un *atributo* y una fila es una *tupla*. En el presente capítulo, para referirnos a los elementos de una base de datos, se utilizará la terminología: relación, atributo y tupla.

Para el desarrollo de los ejemplos y ejercicios de las operaciones del álgebra relacional, se utilizará la base de datos de nuestro caso de estudio **La Ferretería**, que se muestra en la Figura 2.42.

4.1 CONCEPTOS DEL ÁLGEBRA RELACIONAL

Se inicia analizando la similitud entre el álgebra relacional y el álgebra de conjuntos. Matemáticamente un conjunto es una colección de elementos y el estudio de sus operaciones básicas recibe el nombre de álgebra de conjuntos.

Si se aplica a los siguientes conjuntos `A` = {1, 3, 5, 8, 10} y `B` = {1, 4, 5, 7, 8, 11} las operaciones del algebra de conjuntos, como por ejemplo la de unión, intersección y diferencia, de acuerdo a cada una de sus definiciones, se obtienen los siguientes resultados:

C = A ∪ B donde C = {1, 3, 4, 5, 7, 8, 10, 11}

C = A ∩ B donde C = {1, 5, 8}

C = A – B donde C = {3, 10}

En el álgebra relacional, las relaciones corresponden a los conjuntos y las tuplas son los elementos de ese conjunto. Bajo esta consideración las operaciones que se realizan en el álgebra de conjuntos son aplicables al algebra relacional. Por ejemplo, si se tiene las relaciones R y S definidas sobre los dominios [A, B, C], como se muestra en la Figura 4.1.

R	A	B	C
	a1	b1	c1
	a1	b2	c1
	a2	b1	c2

S	A	B	C
	a1	b1	c1
	a2	b2	c1
	a2	b2	c2

Figura 4.1. Relaciones R y S con dominio A, B, C.

Si a estas relaciones se les aplica las operaciones de unión, intersección y diferencia definidas en el álgebra de conjuntos, se obtiene nuevas relaciones que se muestran a continuación en la Figura 4.2.

T = R ∪ S donde

T	A	B	C
	a1	b1	c1
	a1	b2	c1
	a2	b1	c2
	a2	b2	c1
	a2	b2	c2

T = R ∩ S donde

T	A	B	C
	a1	b1	c1

T = R - S donde

T	A	B	C
	a1	b2	c1
	a2	b1	c2

Figura 4.2. Resultado de las operaciones de unión, intersección y diferencia.

Sobre esta base se define que el álgebra relacional es un lenguaje teórico, basado en un conjunto de operadores que se aplican a una o más relaciones con el propósito de obtener otra relación, sin que se modifiquen las originales. El hecho de obtener siempre una relación como resultado de una o más operaciones, recibe el nombre de **propiedad de cierre.**

Las operaciones fundamentales del álgebra relacional son: selección, proyección, unión, diferencia, renombramiento y producto cartesiano, que nos permiten realizar la mayoría de operaciones de extracción de datos de una base de datos. Además, existen otras operaciones que son definidas en términos de las operaciones fundamentales, intersección, combinación, división y asignación.

Desde otro punto de vista, Elmasri y Navathe (2016) dividen a las operaciones del álgebra relacional en dos grupos: Al primero corresponden las operaciones propias del álgebra de conjuntos que se aplican a relaciones y tuplas como la unión, la intersección, la diferencia de conjuntos y el producto cartesiano. Al segundo grupo pertenecen las operaciones creadas específicamente para el manejo de bases de datos relacionales; incluyen la selección (*select*), proyección (*project*) y combinación (*join*).

4.2 OPERACIONES UNARIAS

4.2.1 Selección (*Select*)

Esta operación es utilizada para seleccionar tuplas en una relación R, que satisfacen una condición de selección. El resultado es otra relación con el mismo número de atributos y con igual o menor número de tuplas que la relación R, como se muestra en la Figura 4.3.

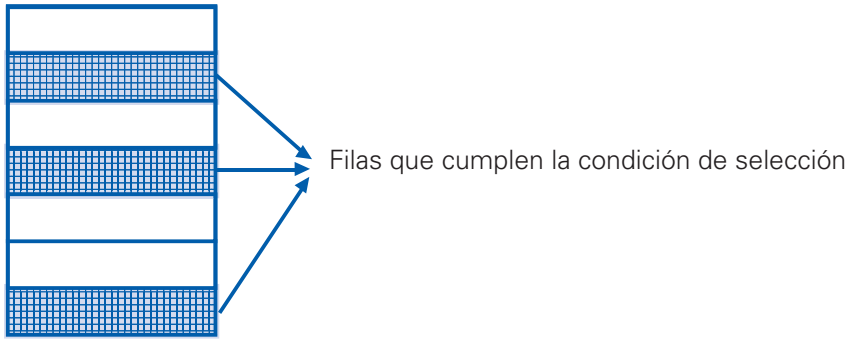


Figura 4.3. Ilustración de la operación selección σ

Operación: unaria, se aplica a una sola relación.

Formato: **σ <condición de selección>(R)**, en donde:

- **σ** (sigma): símbolo del operador

- **<Condición de selección>** expresión booleana de la forma:

  <nombre atributo> <*comparación*> <valor constante>, o cuando la comparación se realiza con otro atributo

  <nombre atributo> <*comparación*> <nombre atributo> donde:

  <nombre atributo> nombre de una columna de la relación R

  <***comparación***> corresponde a uno de los operadores {=,<, ≥,>,≤,≠}.

  <valor constante> valor del dominio del atributo

En la operación selección es posible conectar varias condiciones mediante los **operadores lógicos:** `AND, OR y NOT,` para obtener una expresión de la forma:

σ <*condición*> **Operador lógico** <*condición*> **Operador lógico ... Operador lógico** <*condición*> **(R)** analizada de la siguiente manera:

- <*condición 1*> **AND** <*condición 2*> Produce un resultado *verdadero* si la condición 1 y la condición 2 son verdaderas, en caso contrario el resultado es *falso*.
- <*condición 1*> **OR** <*condición 2*> el resultado es verdadero si al menos una de las dos condiciones es *verdadera*, en caso contrario es *falso*.
- <**NOT** *condición*> el resultado es *verdadero* si la condición es *falso*.

Ejemplos:

- Seleccionar las tuplas de los clientes con un límite de crédito mayor a 1.200 dólares.

  σ `LímCrédito > 1200(CLIENTE)`

- Seleccionar las tuplas de los vendedores de sexo masculino que tienen el objetivo de ventas mayor a 2.500 dólares.

  σ `Sexo = "M" AND ObjVentas > 2500(CLIENTE)`

- Enlistar las tuplas de los clientes que tiene como representante al vendedor de código 1 y límite de crédito mayor a 1.000 dólares o clientes que tienen como representante al vendedor de código 3 y límite de crédito mayor a 1.200 dólares.

  `σ(CódRepre=1 AND LímCrédito > 1000) OR (CódRepre=3 AND LímCrédito > 1200)(CLIENTE)`

4.2.2 Proyección (*Project*)

Esta operación se aplica a una relación `R`, creando otra relación que contiene un subconjunto de columnas seleccionadas, y excluyendo las no consideradas. Si en esta selección existen tuplas duplicadas, éstas se eliminan. En la Figura 4.4 se ilustra esta operación.

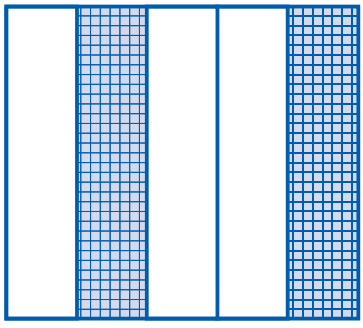


Figura 4.4. Ilustración de la operación proyección π.

Operación: unaria, se aplica a una sola relación.

Formato: **π<lista de atributos>(R)**, donde:

- **π** (Pi): símbolo del operador.
- **<lista de atributos>** Lista de columnas de la relación R que se requiere seleccionar.
- **R**: nombre de la relación.

Por ejemplo, si se necesita seleccionar el nombre, el primer apellido y la cédula de ciudadanía de los clientes, la operación proyección se especifica de la siguiente manera:

```
π Nombre, Apellido1, Cédula(CLIENTE)
```

Se define como **grado** de una relación al número de atributos especificados en <lista de atributos>. En el resultado de la instrucción anterior, el grado de la nueva relación es tres. En lo referente al número de tuplas, para este ejercicio, el resultado tendrá el mismo número que la relación original `CLIENTE`, debido a que, como parte de la <lista de atributos> se selecciona la clave primaria *Cédula*.

En caso de que en la <lista de atributos> de la relación `R` no figure la clave primaria, es posible se presente tuplas duplicadas, que para el resultado de la operación serán eliminadas. Por ejemplo, para recuperar el primer apellido y el límite de crédito de los clientes, se utiliza la operación proyección de la siguiente forma:

```
π Apellido1, LímCrédito(CLIENTE)
```

La relación `CLIENTE` tiene 9 tuplas, que luego de aplicar la operación proyección, como resultado se tiene únicamente 8 tuplas. Se elimina el duplicado de la tupla <´Castro´,600>. El resultado de la consulta se muestra en la Figura 4.5.

Apellido1	LímCrédito
Castro	600
Polo	1100
Tapia	1500
Brito	Null
Mora	1000
Pérez	1200
Torres	1200
Pons	2000

Figura 4.5. Resultado de la operación: π Apellido1, LímCrédito(`CLIENTE`).

Existen ciertas consultas que requieren de la aplicación de una función en la <lista de atributos> de una proyección. Esta operación recibe el nombre de **proyección generalizada**, que se expresa en términos del álgebra relacional de la siguiente forma: π *F1,F2,…, Fn* (R), donde *F1, F2, …. Fn*, son funciones aplicadas a los atributos. Por ejemplo, para obtener un reporte que incluya el nombre, el apellido1, la cédula y el valor del límite de crédito incrementado en 15 por ciento, la operación se puede escribir:

```
π Nombre, Apellido1, Cédula, LímCrédito * 1,15(CLIENTE)
```

La función de esta operación se aplica también entre columnas de la <lista de atributos>. Por ejemplo, se requiere calcular la diferencia entre la existencia máxima y mínima de cada uno de los artículos. La expresión en álgebra relacional y el resultado de la consulta se muestran en la Figura 4.6.

```
π Código, Descripción, ExiMáx – ExiMín(ARTÍCULO)
```

Código	Descripción	ExiMáx - ExiMín
1	Cemento	40
2	Clavos	65
5	Alambre	15
6	Perfil T	7
7	Perfil L	10
8	Perfil G	7
9	Pintura	11
10	Lija	20
11	Suelda	5

Figura 4.6. Consulta con proyección generalizada.

4.2.3 Operación renombrar (*Rename*)

En los ejemplos anteriores el nombre de los atributos de la relación resultante, continúa con el mismo nombre de los atributos de la relación original. En el álgebra relacional con la operación **Renombrar** es posible al resultado de una operación, asignar un nuevo nombre tanto a la nueva relación como a sus atributos. A continuación se analizan los siguientes casos:

a) Renombrar únicamente la relación. Por ejemplo, para obtener un reporte de los datos de todos los vendedores de sexo masculino. Al resultado de la operación selección, se le asigna un nuevo nombre `VEN_MASC`, como se muestra en la siguiente expresión.

```
VEN_MASC ← σSexo = "M" (VENDEDOR)
```

b) Renombrar la relación y sus atributos. Para el caso de los atributos se enlista el nuevo nombre de los atributos dentro de los paréntesis. Si la relación original R tiene los atributos *a1, a2, y a3*, y a la nueva relación se la quiere llamar S, con los atributos renombrados *b1, b2 y b3* respectivamente, la forma generalizada se expresa de la siguiente manera:

$$S(b1,b2,b3) \leftarrow \pi\, a1,\ a2,\ a3(R)$$

Por ejemplo, para conseguir un reporte de los clientes con el nombre, el apellido y su límite de crédito incrementado el 15 por ciento, es posible a la nueva relación renombrarle como INCREMENTO, y a sus atributos asignarle un nuevo nombre *Nnombre, Napellido y Nuevolímite* respectivamente. La expresión de esta consulta y su resultado se muestran en la Figura 4.7.

$$INCREMENTO(Nnombre,\ Napellido,\ Nuevolímite) \leftarrow$$
$$\pi\ Nombre,\ Apellido1,\ LímCrédito*1,15(CLIENTE)$$

Como una alternativa para especificar la operación renombrar se utiliza el símbolo ρ con el siguiente formato:

$$\rho S(a1,a2,a3\ .\ .\ .\ an)(R)$$

donde a la relación R se le asigna un nuevo nombre S y los atributos son renombrados como a_1, a_2, a_3 ... a_n. Por ejemplo, la expresión anterior se puede formular de la siguiente manera y su resultado se muestra en la Figura 4.7.

$$INCREMENTO \leftarrow \rho(Nnombre,\ Napellido,\ Nuevolímite)$$
$$\pi\ Nombre,\ Apellido1,\ LímCrédito*1,15(CLIENTE)$$

INCREMENTO

Nnombre	Napellido	NuevoLímite
Víctor	Castro	690
Juan	Polo	1265
Elena	Tapia	1725
Pablo	Brito	Null
Marcia	Mora	1150
Jaime	Pérez	1380
Manuel	Castro	690
Alberto	Torres	1380
Humberto	Pons	2300

Figura 4.7. Expresión y resultado con la operación renombrar.

4.2.4 Secuencia de operaciones

Las operaciones del álgebra relacional es posible escribirlas en una única expresión o creando una secuencia de operaciones con resultados intermedios. Por ejemplo, si se quiere recuperar el nombre y primer apellido de los clientes que tienen un límite de crédito mayor o igual a 1.200 dólares, se aplican los operadores proyección y selección, que agrupados en una sola expresión se escriben de la siguiente manera:

$$\text{CONSULTA} \leftarrow \pi_{\text{Nombre, Apellido1, LímCrédito}}(\sigma_{\text{LímCrédito} \geq 1200}(\text{CLIENTE}))$$

Alternativamente es posible crear una secuencia de operaciones con resultados intermedios y renombrarlos a cada uno de ellos. Para el caso del ejemplo anterior se obtiene un resultado cuando se aplica la operación selección y otra con la proyección. En la Figura 4.8 se muestra la secuencia de las instrucciones y los resultados de cada una.

$$\text{CRÉDITO\_MAYOR} \leftarrow \sigma_{\text{LímCrédito} \geq 1200}(\text{CLIENTE})$$

$$\text{CONSULTA} \leftarrow \pi_{\text{Nombre, Apellido1, LímCrédito}}(\text{CRÉDITO\_MAYOR})$$

CRÉDITO_MAYOR

Nombre	Apellido1	Apellido2	Cédula	Dirección	LímCrédito	CédulaReco	CódRepre	FechaAsig
Elena	Tapia	Barrera	0111555666	Amazonas 3-45	1500	0777888999	Null	4-Jun-12
Jaime	Pérez	Guerrero	0777888999	Colón 4-98	1200	0122334455	2	2-Oct-12
Alberto	Torres	Viteri	0999998888	Luque 10-10	1200	0122334455	4	13-Ene-13
Humberto	Pons	Coronel	0456456456	Borrero 3-45	2000	Null	3	10-Sept-13

CONSULTA

Nombre	Apellido1	LímCrédito
Elena	Tapia	1500
Jaime	Pérez	1200
Alberto	Torres	1200
Humberto	Pons	2000

Figura 4.8. Secuencia de operaciones y resultado de la consulta.

4.3 OPERACIONES DE CONJUNTOS

En esta sección se revisan las operaciones del álgebra de conjuntos: unión, intersección, diferencia y producto cartesiano, que son aplicadas en el álgebra relacional. Para ello se consideran las relaciones R y S con sus atributos A = *(a1, a2, a3,..., an) y B = (b1, b2, b3,...., bn)* respectivamente.

4.3.1 Unión (*Union*)

La **unión** de dos relaciones R y S, expresada por $R \cup S$, define una relación que incluye todas las tuplas que están en R o en S o en ambas, eliminando las tuplas duplicadas. En la Figura 4.9 se ilustra este concepto. La operación unión tiene la propiedad de ser conmutativa $R \cup S = S \cup R$.

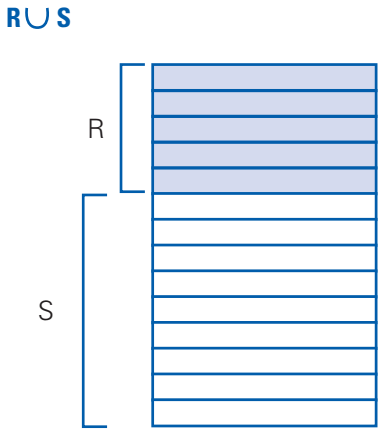


Figura 4.9. Función de la operación unión.

Para que sea posible la unión las dos relaciones R y S deben ser compatibles, es decir, tener el mismo número de atributos y cada pareja de atributos correspondientes de R y S ser del mismo dominio. Estas dos condiciones reciben el nombre de **compatibilidad con respecto a la unión** (Connolly T., Begg C., 2005, pág. 83). Las relaciones R y S pueden ser el resultado de operaciones intermedias, en las que, con el objeto de formar dos relaciones compatibles, por lo general se aplica la operación proyección, como se explica en el siguiente ejemplo. Para extraer el nombre y el primer apellido de los clientes que tienen un límite de crédito mayor a 1.000 dólares o tienen como representante al vendedor de código 3. La secuencia de instrucciones para esta consulta, en la que se incluye la operación UNIÓN se muestra a continuación.

```
CLIENTE_CRÉDITO ← π Nombre, Apellido1 (σ LímCrédito > 1000(CLIENTE))

CLIENTE_VENDEDOR ← π Nombre, Apellido1 (σ CódRepre = 3(CLIENTE))

RESULTADO ← CLIENTE_CRÉDITO ∪ CLIENTE_VENDEDOR
```

Previo a la ejecución de la operación unión, se generan dos relaciones intermedias, la primera CLIENTE_CRÉDITO como resultado de aplicar la operación selección con la condición *LímCrédito* > 1.000 y, la operación proyección para seleccionar los atributos *Nombre* y *Apellido1*, el resultado de esta expresión se muestra en la Figura 4.10(a). Con el mismo criterio se obtiene el segundo resultado intermedio CLIENTE_VENDEDOR, que corresponde a la condición *CódRepre* = 3 como se muestra en la Figura 4.10(b). Las dos relaciones tienen el mismo número de atributos y cada pareja correspondiente tiene el mismo dominio, con lo que se cumple con la condición de compatibilidad con respecto a la unión.

Por último, para dar respuesta a la consulta, se combinan estas dos relaciones con la utilización de la operación unión, obteniendo tuplas con los nombres y apellidos de los clientes que aparecen en alguna de las dos relaciones o en ambas, las tuplas <Elena, Tapia> y <Humberto, Pons> aparecen sólo una vez en el resultado.

Para el caso de este ejemplo, los nombres de los atributos de las dos relaciones que participan en la operación unión son los mismos, sin embargo, ésta no es una condición que deben cumplir las relaciones. Adicionalmente en este ejemplo, el nombre de los atributos de la relación `RESULTADO_U` son los mismos que los de la relación `CLIENTE_CRÉDITO` y `CLIENTE_VENDEDOR`, pudiendo ser renombrados con la operación renombrar. En la Figura 4.10(c) se muestra la relación `RESULTADO_U` como producto de la operación unión.

CLIENTE_CRÉDITO

Nombre	Apellido1
Juan	Polo
Elena	Tapia
Jaime	Pérez
Alberto	Torres
Humberto	Pons

(a)

CLIENTE_VENDEDOR

Nombre	Apellido1
Elena	Tapia
Pablo	Brito
Humberto	Pons

(b)

RESULTADO_U

Nombre	Apellido1
Juan	Polo
Elena	Tapia
Jaime	Pérez
Alberto	Torres
Humberto	Pons
Pablo	Brito

(c)

Figura 4.10. (a) Operación selección con la condición *LímCrédito* > 1000. (b) Operación selección con la condición *CódRepre* = 3. (c) Resultado de la operación unión.

4.3.2 Intersección (*Intersection*)

La operación **intersección** de dos relaciones `R y S,` expresada por `R ∩ S,` define una relación que incluye todas las tuplas que están en `R` y en `S`, siempre que las dos relaciones sean compatibles con respecto a la unión. En la Figura 4.11 se muestra el resultado de la intersección. Esta operación cumple con la propiedad de ser conmutativa `R ∩ S = S ∩ R`

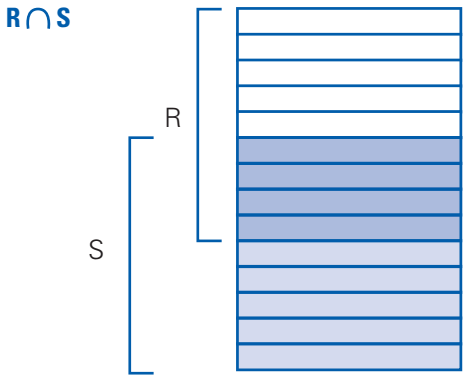


Figura 4.11. Función de la operación intersección.

Para ejemplificar la operación intersección se utiliza las relaciones `CLIENTE_CRÉDITO` y `CLIENTE_VENDEDOR` de la Figura 4.10(a) y (b). Si se requiere un reporte de todos los clientes que tienen un límite de crédito mayor a 1.000 dólares y que tengan como representante al vendedor de código 3. La secuencia de instrucciones será:

```
CLIENTE_CRÉDITO ← π Nombre, Apellido1 (σ LímCrédito > 1000(CLIENTE))
CLIENTE _VENDEDOR ← π Nombre, Apellido1 (σ CódRepre = 3(CLIENTE))
RESULTADO_I ← CLIENTE_CRÉDITO ∩ CLIENTE_VENDEDOR
```

El resultado de la operación intersección se muestra en la Figura 4.12.

RESULTADO_I

Nombre	Apellido1
Elena	Tapia
Humberto	Pons

Figura 4.12. Resultado de la operación intersección.

4.3.3 Diferencia o menos (*Minus*)

La **diferencia** de dos relaciones R y S, expresada por R - S, se define como una relación que incluye las tuplas que están en R, pero no en S, siempre que las dos relaciones sean compatibles con respecto a la unión. La diferencia o menos no es conmutativa, es decir, $R - S \neq S - R$. En la Figura 4.13 se muestra la función de esta operación.

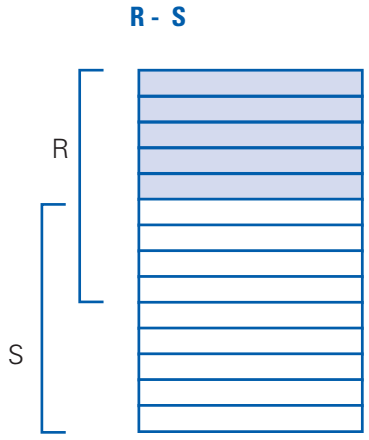


Figura 4.13. Función de la operación diferencia.

Como en el ejemplo anterior se utilizan las relaciones de la Figura 4.10(a) y (b) para generar la siguiente consulta. Recuperar todos los clientes que tienen un límite de crédito mayor a 1.000 dólares, pero no están representados por el vendedor de código 3. Se combinan las dos relaciones utilizando la operación diferencia, como se muestra a continuación.

```
RESUL_DIF1 ⟵ CLIENTE_CRÉDITO - CLIENTE_VENDEDOR
```

El resultado de la consulta precedente se muestra en la Figura 4.14(a).

Como se indicó anteriormente, la operación diferencia no es conmutativa; para ilustrar esta propiedad se considera la siguiente consulta. Se quiere recuperar todos los clientes que tienen como representante el vendedor de código 3, pero no tienen un límite de crédito mayor a 1.000 dólares. Se combinan las dos relaciones como se muestra en la siguiente operación.

```
RESUL_DIF2 ⟵ CLIENTE_VENDEDOR - CLIENTE_CRÉDITO
```

El resultado de la consulta precedente se muestra en la Figura 4.14(b).

RESUL_DIF1

Nombre	Apellido1
Juan	Polo
Jaime	Pérez
Alberto	Torres

(a)

RESUL_DIF2

Nombre	Apellido1
Pablo	Brito

(b)

Figura 4.14. Resultados de la operación diferencia.

4.3.4 Producto cartesiano o producto cruzado (*Cartesian product*)

La operación binaria **producto cartesiano** de las relaciones `R y S`, expresada por `R x S`, define una relación que incluye la combinación de cada tupla de la relación `R` con cada tupla de la relación `S.` No es necesario que las dos relaciones sean compatibles. En la Figura 4.15 se ilustra la función del producto cartesiano

R
1
2

S
a
b
c

R X S	
1	a
2	a
1	b
2	b
1	c
2	c

Figura 4.15. Función de la operación producto cartesiano.

Si la relación `R` tiene m atributos y p tuplas y la relación `S` tiene n atributos y `q` tuplas, la operación producto cartesiano `R` x `S` contendrá *m + n* atributos y *p * q* tuplas. Los nombres de los atributos de la nueva relación serán los mismos que los atributos de `R` y `S`, sin embargo, en caso de que las dos relaciones tengan atributos con el mismo nombre, para diferenciarlos y cumplir con la condición de unicidad de atributos dentro de una relación, tendrán que ser renombrados.

La operación producto cartesiano tiene sentido siempre que esté seguida de una operación selección que cree una correspondencia entre los valores de los atributos de las dos relaciones. Por ejemplo, para obtener un reporte con el nombre y apellido de los vendedores y el nombre de sus cargas familiares la secuencia de operaciones es la siguiente.

```
VENDEDOR_NOMBRE ← π Vnombre, Apellido, CódigoVend(VENDEDOR)

CARGAFA_NOMBRE ← π NomDep, CódVendedor(CARGAFAMILIAR)

VENDEDOR_CARGA ← VENDEDOR_NOMBRE x CARGAFA_NOMBRE

CONDICIÓN ← σ CódigoVend = CódVendedor(VENDEDOR_CARGA)

REPORTE ← π Nombre, Apellido, NomDep(CONDICIÓN)
```

La relación VENDEDOR_NOMBRE contiene 3 atributos y 4 tuplas y la relación CARGAFA_NOMBRE contiene 2 atributos y 5 tuplas, como se muestran en la Figura 4.16(a) y (b) respectivamente, y son el resultado de aplicar la operación proyección a las relaciones VENDEDOR y CARGAFAMILIAR, con el objeto de eliminar los atributos que no se utilizarán en las subsiguientes secuencias de operaciones.

La relación VENDEDOR_CARGA que se muestra en la Figura 4.16(c) contiene 5 atributos y 20 tuplas, y es el resultado del producto cartesiano entre las relaciones VENDEDOR_NOMBRE y CARGAFA_NOMBRE.

La siguiente operación selecciona las tuplas que cumplan la condición *CódigoVend = CódVendedor,* es decir, selecciona a cada uno de los vendedores con sus respectivas cargas familiares. El resultado de la instrucción es la relación denominado CONDICIÓN que se muestra en la Figura 4.16 (d).

Por último para obtener el resultado requerido se realiza una proyección con la que se extraen los atributos *Vnombre, Apellido, NomDep,* creando la relación REPORTE que se muestra en la Figura 4.16(e).

Si en la secuencia de instrucciones no se realizan las dos primeras operaciones de PROYECCIÓN, el número de atributos de la relación resultante del producto cartesiano de las relaciones VENDEDOR y CARGAFAMILIAR será igual a doce, que corresponde a la suma de todos los atributos de las dos relaciones.

VENDEDOR_NOMBRE

Vnombre	Apellido	CódigoVend
Diego	Loja	1
Antonio	Calle	2
Lucía	Serrano	3
Arturo	Salto	4

(a)

CARGAFA_NOMBRE

NomDep	CódVendedor
María	1
Isabel	1
Antonio	2
Maricela	2
Teodoro	3

(b)

VENDEDOR_CARGA

Vnombre	Apellido	CódigoVend	NomDep	CódVendedor
Diego	Loja	1	María	1
Diego	Loja	1	Isabel	1
Diego	Loja	1	Antonio	2
Diego	Loja	1	Maricela	2
Diego	Loja	1	Teodoro	3
Antonio	Calle	2	María	1
Antonio	Calle	2	Isabel	1
Antonio	Calle	2	Antonio	2
Antonio	Calle	2	Maricela	2
Antonio	Calle	2	Teodoro	3
Lucía	Serrano	3	María	1
Lucía	Serrano	3	Isabel	1
Lucía	Serrano	3	Antonio	2
Lucía	Serrano	3	Maricela	2
Lucía	Serrano	3	Teodoro	3
Arturo	Salto	4	María	1
Arturo	Salto	4	Isabel	1
Arturo	Salto	4	Antonio	2
Arturo	Salto	4	Maricela	2
Arturo	Salto	4	Teodoro	3

(c)

CONDICIÓN

Vnombre	Apellido	CódigoVend	NomDep	CódVendedor
Diego	Loja	1	María	1
Diego	Loja	1	Isabel	1
Antonio	Calle	2	Antonio	2
Antonio	Calle	2	Maricela	2
Lucía	Serrano	3	Teodoro	3

(d)

REPORTE

Vnombre	Apellido	NomDep
Diego	Loja	María
Diego	Loja	Isabel
Antonio	Calle	Antonio
Antonio	Calle	Maricela
Lucía	Serrano	Teodoro

(e)

Figura 4.16. Secuencia de operaciones que incluye un producto cartesiano:

(a) VENDEDOR_NOMBRE ← π Vnombre, Apellido, CódigoVend(VENDEDOR)

(b) CARGAFA_NOMBRE ← π NomDep, CódVendedor(CARGAFAMILIAR)

(c) VENDEDOR_CARGA ← VENDEDOR_NOMBRE x CARGAFA_NOMBRE

(d) CONDICIÓN ← σ CódigoVend = CódVendedor(VENDEDOR_CARGA)

(e) REPORTE ← π Nombre, Apellido, NomDep(CONDICIÓN)

4.4 OPERACIONES BINARIAS

4.4.1 Combinación o concatenación (*Join*)

La operación de combinación, denotada por el símbolo ⋈ , combina tuplas similares a partir de dos relaciones para formar una nueva relación. En otras palabras, la combinación es una derivación del producto cartesiano seguido de una operación de selección. Su formato está definido por:

Q ← R⋈<condición de selección> S

Si las relaciones R y S tienen *n* y *m* atributos respectivamente, la relación Q tendrá *n* + *m* atributos y una tupla por cada combinación de las tuplas de R y S que satisfagan la condición de selección. Las tuplas combinadas que no satisfacen la condición de selección o tienen en los atributos de conexión valores nulos, no formarán parte del resultado. En general si las relaciones R y S están formadas por *p* y *q* tuplas respectivamente, el resultado de la combinación tendrá entre cero y *p* * *q* tuplas.

Para ilustrar este concepto se utiliza el ejemplo que se analizó en la Sección 4.3.4. La relación CONDICIÓN es el resultado de aplicar un producto cartesiano, seguido de una operación que selecciona las tuplas que cumplen la condición *CódigoVend = CódVendedor*, como se transcribe a continuación.

```
VENDEDOR_CARGA ← VENDEDOR_NOMBRE x CARGAFA_NOMBRE
CONDICIÓN ← σ CódigoVend = CódVendedor(VENDEDOR_CARGA)
```

Este par de operaciones se remplazan por una única operación de combinación, de la siguiente forma:

```
CONDICIÓN ← VENDEDOR_NOMBRE ⋈ CódigoVend = CódVendedor CARGAFA_NOMBRE
```

El resultado de esta instrucción se mostró en la Figura 4.16(d)

La función combinación tiene diferentes formas de operar, con una ligera variación entre una y otra, que se explica a continuación.

4.4.2 Combinación theta (*Theta join*) y equicombinación (*Equijoin*)

Recibe el nombre de combinación theta, cuando en la condición de selección se utiliza uno de estos operadores de comparación ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$) .

Si la condición de selección contiene el operador de igualdad (=), a la operación de combinación se la denomina **equicombinación o *Equijoin***. Por ejemplo, si se requiere recuperar el nombre y los apellidos de los clientes y el nombre y apellido de su representante, la secuencia de operaciones es la siguiente:

```
CLI_VEND ← CLIENTE ⋈ CódRepre = CódigoVend VENDEDOR

RESULTADO ← π Nombre, Apellido1, Apellido2, Vnombre, Apellido(CLI_VEND)
```

La relación `CLI_VEND` es el resultado de la operación equicombinación y contendrá quince atributos, ocho corresponden a la relación `CLIENTE` y siete a la relación `VENDEDOR`, como se muestra en la Figura 4,17(a). La relación final con los datos, objeto de la consulta, aparece en la Figura 4.17(b).

CLI_VEND

Nombre	Apellido1	Apellido2	...	CódRepre	FechaAsig	Nombre	Apellido	CódigoVend	...
Victor	Castro	Torres	...	1	13-oct-12	Diego	Loja	1	...
Juan	Polo	Ávila	...	1	02-mar-13	Diego	Loja	1	...
Pablo	Brito	Parra	...	3	15-dic-11	Lucía	Serrano	3	...
Marcia	Mora	Durazno	...	1	23-dic-12	Diego	Loja	1	...
Jaime	Pérez	Guerrero	...	2	02-oct-12	Antonio	Calle	2	...
Manuel	Castro	León	...	2	12-feb-12	Antonio	Calle	2	...
Alberto	Torres	Viteri	...	4	13-ene-13	Arturo	Salto	4	...
Humberto	Pons	Coronel	...	3	10-sep-13	Lucía	Serrano	3	...

(a)

RESULTADO

Vnombre	Apellido1	Apellido2	Vnombre	Apellido
Victor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Pablo	Brito	Parra	Lucía	Serrano
Marcia	Mora	Durazno	Diego	Loja
Jaime	Pérez	Guerrero	Antonio	Calle
Manuel	Castro	León	Antonio	Calle
Alberto	Torres	Viteri	Arturo	Salto
Humberto	Pons	Coronel	Lucía	Serrano

(b)

Figura 4.17. (a) Resultado de la operación equicombinación, (b) Proyección.

4.4.3 Combinación natural *(Natural Join)*

La operación de **combinación natural** denotada por * es una equicombinación en la que los dos atributos de conexión tienen el mismo nombre, razón por la cual, no es necesario nombrarlos en la instrucción. En caso de no cumplir con esta condición se requerirá previamente realizar una operación de renombramiento. En la relación resultante sólo se mantendrá uno de los atributos de la conexión. "En general, una `CONCATENACIÓN NATURAL` (combinación natural) se lleva a cabo equiparando *todos* los pares de atributos que tengan el mismo nombre en las dos relaciones". (Elmasri R., Navathe S., 2007, pág. 157). Esta operación recibe también el nombre **combinación interna (*Inner join*).** La forma general de la operación combinación natural es:

$$Q \leftarrow R * S$$

Por ejemplo, para recuperar la descripción y la cantidad de artículos vendidos en la factura número 103, la secuencia de operaciones es la siguiente:

```
FAC_103 ← σ FactNúmero = 103(DETALLE)

TEMP FactNúmero, Código, Cantidad ← π FactNúmero, ArtCódigo, Cantidad(FAC_103)

ART_FACT ← ARTÍCULO * TEMP

REPORTE ← π Descripción, Cantidad(ART_FACT)
```

El primer resultado intermedio `FAC_103` recupera los códigos de artículos y las cantidades de la factura 103 de la relación `DETALLE`. El resultado se muestra en la Figura 4.18(a). En la siguiente instrucción, al atributo *ArtCódigo* de la relación `FAC_103`, se le renombra como *Código*, para que el nombre coincida con el de la relación `ARTÍCULO`, cumpliendo con el requisito de tener el mismo nombre los atributos de conexión. El resultado se muestra en la Figura 4.18(b). En la relación intermedia `ART_FACT`, se realiza la operación combinación, que contendrá una tupla de cada combinación de las dos relaciones `ARTÍCULO` y `TEMP`, siempre que cumpla con la condición de igualdad. El resultado se muestra en la Figura 4.18(c). Por último se aplica la operación proyección para obtener el `REPORTE` de la consulta requerida. En la Figura 4.18(d) se muestran los resultados de la secuencia de operaciones.

FACT_103

FactNúmero	ArtCódigo	Cantidad
103	11	1
103	9	4
103	1	4

(a)

TEMP

FactNúmero	ArtCódigo	Cantidad
103	11	1
103	9	4
103	1	4

(b)

ART_FACT

Factura	Código	Cantidad	Descripción	Precio	Unidad	ExiMáx	ExiMín
103	11	1	Suelda	20,0	Kilo	10	5
103	9	4	Pintura	19,5	Galón	15	4
103	1	4	Cemento	6,0	Saco	50	10

(c)

REPORTE

Descripción	Cantidad
Suelda	1
Pintura	4
Cemento	4

(d)

Figura 4.18. Secuencia de operaciones para aplicar la operación de combinación natural.

4.4.4 Combinación externa (*Outer join*)

Generalmente en una combinación interna (*inner join*) no todas las tuplas; de las relaciones R y S cumplen con la condición de selección. Estas tuplas y las que en el atributo de selección contienen un valor nulo (NULL), son eliminadas del resultado; sin embargo, en ciertas ocasiones es necesario que en el resultado también aparezcan todas las tuplas de R, o S, o de ambas, que no satisfacen la condición de selección. Este tipo de condiciones que se presentan en una consulta se realizan utilizando un conjunto de tres operaciones de combinación externa (*outer join*), que se detallan a continuación.

4.4.4.1 Combinación externa izquierda (*Left outer join*)

Denotada por R ⟕ S, es una combinación cuyo resultado incluye todas las tuplas de la relación R independientemente si cumplen o no la condición de conexión. A las tuplas que no tengan coincidencia, en el atributo de selección de la relación S, se les asigna un valor nulo.

Para ilustrar esta operación se toma como referencia el ejemplo de la Sección 4.4.2 que requiere un reporte de todos los clientes y sus vendedores que fueron asignados como representantes. El resultado de esta consulta se muestra en la Figura 4.17(a), en la que se observa que no se incluyen dos tuplas: la una de la vendedora "Maricela Vera" que no tiene asignados clientes a quienes representar; y la otra, del cliente "Elena Tapia" que no tiene un vendedor como representante. Para el caso de requerir un reporte en el que no se eliminen de la relación resultante las tuplas de los clientes que no tienen valores correspondientes con vendedores, se utiliza una combinación externa como se muestra a continuación:

```
EXT_IZQ ← CLIENTE ⟕ CódRepre = CódigoVend VENDEDOR

CLI_VEND ← π Nombre, Apellido1, Apellido2, Vnombre, Apellido(EXT_IZQ)
```

La relación resultante se muestra en la Figura 4.19, que mantiene todas las tuplas de la primera relación o izquierda. Obsérvese que en este resultado se incluye al cliente "Elena Tapia Barrera", y a los atributos de esta tupla, correspondientes al vendedor, se les asigna un valor nulo.

CLI_VEND

Nombre	Apellido1	Apellido2	Vnombre	Apellido
Victor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Elena	Tapia	Barrera	Null	Null
Pablo	Brito	Parra	Lucía	Serrano
Marcia	Mora	Durazno	Diego	Loja
Jaime	Pérez	Guerrero	Antonio	Calle
Manuel	Castro	León	Antonio	Calle
Alberto	Torres	Viteri	Arturo	Salto
Humberto	Pons	Coronel	Lucía	Serrano

Figura 4.19. Combinación externa izquierda de las relaciones `CLIENTE` y `VENDEDOR`.

4.4.4.2 Combinación externa derecha (*Right outer join*)

Esta operación está denotada por `R` ⟖ [`S`. A diferencia de la combinación externa izquierda, ésta incluye en el resultado todas las tuplas de la relación derecha. La siguiente secuencia de instrucciones ejemplifica esta operación, que incluirá en el resultado todas la tuplas de la relación `VENDEDOR`, inclusive las que no son coincidentes, como es el caso de la vendedora "Marcia Vera". La Figura 4.20 muestra el resultado de la operación.

```
EXT_DER ← CLIENTE ⟖ [ CódRepre = CódigoVend VENDEDOR

CLI_VEND ← π Nombre, Apellido1, Apellido2, Vnombre, Apellido (EXT_DER)
```

CLI_VEND

Nombre	Apellido1	Apellido2	Nombre	Apellido
Victor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Pablo	Brito	Parra	Lucía	Serrano
Marcia	Mora	Durazno	Diego	Loja
Jaime	Pérez	Guerrero	Antonio	Calle
Manuel	Castro	León	Antonio	Calle
Alberto	Torres	Viteri	Arturo	Salto
Humberto	Pons	Coronel	Lucía	Serrano
Null	Null	Null	Null	Null

Figura 4.20. Combinación externa derecha de las relaciones `CLIENTE` y `VENDEDOR`.

4.4.4.3 Combinación externa completa (*Full outer join*)

Operación denotada por R] ⋈ [S, cuyo resultado incluirá todas las tuplas de las dos relaciones, colocando nulo en los atributos de las tuplas que no tengan correspondencia. La Figura 4.21 muestra el resultado de la siguiente secuencia de instrucciones de una operación de combinación externa.

```
EXT_COMP ⟵ CLIENTE ] ⋈ [ CódRepre = CódigoVend VENDEDOR

CLI_VEND ⟵ π Nombre, Apellido1, Apellido2, Vnombre, Apellido(EXT_COMP)
```

CLI_VEND

Nombre	Apellido1	Apellido2	Vnombre	Apellido
Victor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Elena	Tapia	Barrera	Null	Null
Pablo	Brito	Parra	Lucía	Serrano
Marcia	Mora	Durazno	Diego	Loja
Jaime	Pérez	Guerrero	Antonio	Calle
Manuel	Castro	León	Antonio	Calle
Alberto	Torres	Viteri	Arturo	Salto
Humberto	Pons	Coronel	Lucía	Serrano
Null	Null	Null	Maricela	Vera

Figura 4.21. Combinación externa completa de las relaciones `CLIENTE` y `VENDEDOR`.

4.5 OPERACIONES COMPLEMENTARIAS

4.5.1 Operaciones de agregación y agrupación

Las operaciones de **función de agregación** nos ayudan a obtener resúmenes de datos a partir de la base de datos, algo similar a los totales que se puede encontrar en un reporte. Si a estos resúmenes se les aplica el concepto de agrupación se obtendrán subtotales agrupados de acuerdo con un determinado criterio. Con las operaciones básicas del algebra relacional no es posible obtener estos resúmenes, no obstante, se han creado otras operaciones, las principales funciones son las que se detallan a continuación:

`SUMA (SUM):` totaliza los valores en el atributo asociado.

`PROMEDIO (AVG):` calcula el promedio de los valores en un atributo.

`MÁXIMO (MAX):` obtiene el máximo valor en el atributo asociado.

`MÍNIMO (MIN):` obtiene el mínimo valor en un atributo asociado.

`CONTAR (COUNT):` cuenta el número de valores en el atributo asociado.

Las funciones están denotadas por el símbolo $\Im$. Su formato es:

<atributos de agrupación> $\Im$ <lista funciones>(<nombre relación>)

Donde <atributos de agrupación> es una lista de atributos de la relación especificada en <nombre relación>, y <lista funciones> es una lista de pares (<función agregación><atributo>). La relación resultante contiene los <atributos de agrupación>, y el resultado de la función de agregación aplicado al atributo. Si no se incluye el atributo de agrupación, la función de agregación se aplica a todas las tuplas de la relación, obteniéndose una sola tupla como resultado.

A continuación se revisan una serie de ejemplos que ilustran el funcionamiento de las operaciones de agregación y agrupación:

Si se quiere contar el número de clientes de **La Ferretería,** la operación es la siguiente, y su resultado se muestra en la Figura 4.22(a).

$\Im$ COUNT Cédula(CLIENTE)

Para el caso en el que se requiera un reporte con información del vendedor que incluya su código, el número de clientes asignados y el promedio del límite de crédito de sus clientes. En esta consulta se considera al atributo *CódRepre* como atributo de agrupación, la función **COUNT** cuenta el número de clientes por cada vendedor y la función **AVG** calcula el promedio del límite de crédito, agrupados por vendedor. El resultado de esta operación que se detalla a continuación se muestra en la Figura 4.22(b).

CódRepre $\Im$ COUNT Cédula, AVG LímCrédito(CLIENTE)

En las operaciones anteriores, los atributos de la nueva relación toman el nombre del <atributo de agrupación> y del par (<función>,<atributo>) , sin embargo, es posible en el resultado cambiar el nombre con la operación **RENOMBRAR**. Por ejemplo, si se quiere un reporte que incluya el código del vendedor y la cantidad de facturas realizadas, renombrando los atributos de la relación resultante como *Vendedor* y *Núm_Factura* y asignándole a la nueva relación el nombre de **REPORTE**. El resultado de esta operación que se detalla a continuación se muestra en la Figura 4.22(c).

REPORTE (Vendedor, Núm_Factura) ← (CódVendedor ℑ COUNT Número(FACTURA))

Para la resolución de una función se debe considerar que las tuplas duplicadas no se eliminan y el resultado de la función no es un número escalar, sino una relación.

COUNT cédula
9

(a)

CódRepre	COUNTCédula	AVG LímCrédito
1	3	766,66
2	2	850
3	3	1750
4	1	1200

(b)

REPORTE

Vendedor	Núm_Factura
1	3
2	3
3	3
4	3

(c)

Figura 4.22. Consultas con funciones de agregación y agrupación:

a) ℑ COUNT cédula (CLIENTE).

b) CódRepre ℑ COUNT cédula, AVG (LímCrédito) (CLIENTE).

c) REPORTE (Vendedor, Núm_Factura) ← (CódVendedor ℑ COUNT Número(FACTURA)).

4.5.2 Cierre recursivo

Esta operación se aplica en los tipos de relaciones recursivas, como las que se analizaron en la Sección 2.2.3.1, en donde se modelan los clientes que recomendaron a otros clientes, mediante el tipo de relación RECOMENDADO. Sobre la base de los datos de la tabla CLIENTE, en la Figura 4.22, se muestra en forma de un árbol invertido los niveles de las relaciones de los clientes y sus recomendados.

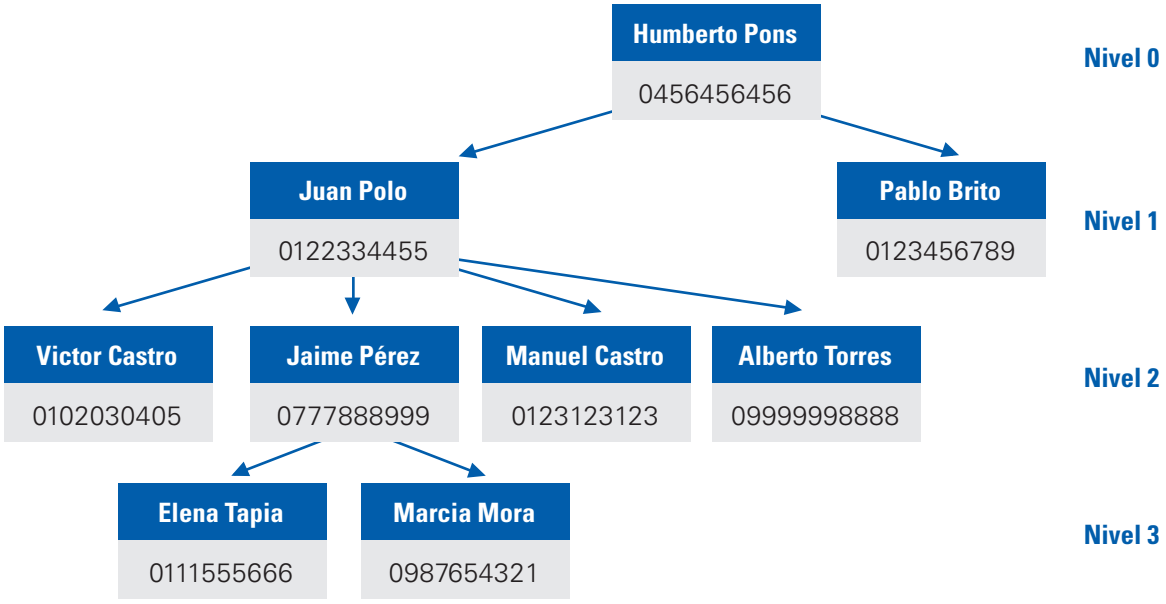


Figura 4.23. Árbol invertido del tipo de relación RECOMENDADO.

Para ejemplificar el cierre recursivo, se analiza la siguiente consulta: Recuperar la cédula de ciudadanía de los clientes recomendados directamente por Humberto Pons. En la Figura 4.23 se observa que Juan Polo y Pablo Brito son los clientes directamente recomendados por Humberto Pons, y que corresponden al primer nivel de la relación recursiva RECOMIENDA. La secuencia de operaciones para obtener el resultado de la consulta es la siguiente:

De la relación CLIENTE se recupera la cédula de Humberto Pons. Figura 4.24(a) con la siguiente instrucción.

```
CÉDULA_HPONS(CédPons) ← π Cédula(σ Nombre ='Humberto',
                                  Apellido1='Pons'(CLIENTE))
```

Luego se recupera la cédula de ciudadanía del cliente y la cédula de quien lo recomendó. Figura 4.24 (b)

```
RECOMENDADO ← π Cédula, CédulaReco(CLIENTE)
```

A continuación se realiza la operación de combinación entre las relaciones *Recomendado y Cédula_Hpons*, para luego aplicar la operación proyección y obtener el número de cédula de los clientes recomendados por el cliente Humberto Pons en la relación denominada `REC_NIVEL1.`

$$\texttt{REC\_NIVEL1(Cc)} \leftarrow \pi_{\texttt{Cédula}}(\texttt{RECOMENDADO} \bowtie_{\texttt{CédulaReco = CédPons}} \texttt{CÉDULA\_HPONS})$$

En la Figura 4.24(c), se muestra el resultado de esta operación que, con el objeto de ilustrar el proceso, se incluye la relación intermedia R, que corresponde al producto cartesiano que forma parte de la operación combinación.

CÉDULA_HPONS

CédPons
0456456456

(a)

RECOMENDADO

Cédula	CédulaReco
0102030405	0122334455
0122334455	0456456456
0111555666	0777888999
0123456789	0456456456
0987654321	0777888999
0777888999	0122334455
0456456456	Null
0999998888	0122334455
0123123123	0122334455

(b)

R

Cédula	CédulaReco	Céd_Pons
0102030405	0122334455	0456456456
0122334455	0456456456	0456456456
0111555666	0777888999	0456456456
0123456789	0456456456	0456456456
0987654321	0777888999	0456456456
0777888999	0122334455	0456456456
0456456456	Null	0456456456
0999998888	0122334455	0456456456
0123123123	0122334455	0456456456

REC_NIVEL1

Cc
0122334455
0123456789

(c)

Figura 4.24. Procesos para una consulta de cierre recursivo.

La operación de cierre recursivo se la puede realizar a un nivel superior. Por ejemplo, para obtener un reporte de los clientes recomendados por todos los clientes que recomendó Humberto Pons, es decir, los clientes que en la Figura 4.23 se encuentran en el nivel 2, se puede implementar esta consulta con la siguiente instrucción:

$$\texttt{REC\_NIVEL2} \leftarrow \pi\ \texttt{Cc}\ (\texttt{RECOMENDADO} \bowtie \texttt{CédulaReco} = \texttt{Cc}\ \texttt{REC\_NIVEL1})$$

El resultado se muestra en la Figura 4.25, en donde consta la cédula de los cuatro clientes que fueron recomendados por Juan Polo, y que éste a su vez fue recomendado por Humberto Pons.

REC_NIVEL2

Cédula
0102030405
0777888999
0123123123
0999998888

Figura 4.25. Segundo nivel de consulta de un cierre recursivo.

Si se requiere obtener un listado de todos los clientes recomendados a partir de Humberto Pons en los niveles uno y dos, se realiza la operación de `UNIÓN` entre las dos relaciones `REC_NIVEL1 y REC_NIVEL2`. El resultado de esta operación se muestra en la Figura 4.26.

RESULTADO

Cédula
0122334455
0123456789
0102030405
0777888999
0123123123
0999998888

Figura 4.26. Resultado de la operación unión entre `REC_NIVEL1 y REC_NIVEL2.`

Para realizar la consulta a un nivel 3 o hasta donde se presenta la recursividad, se utiliza un proceso de bucle similar al descrito.

4.5.3 Unión externa

En el numeral 4.3.1 se analizaron las condiciones para realizar la operación unión entre dos relaciones; sin embargo, es posible realizar la operación cuando las dos relaciones no son compatibles respecto de la unión, mediante la unión externa.

Para ilustrar esta operación se supone que las relaciones `CLIENTE`, tienen los atributos *Nombre, Apellido1, Cédula y Límite de crédito* y la relación `VENDEDOR` los atributos *Nombre, Apellido1, Cédula y Objetivo de Ventas.* Estas dos relaciones son compatibles parcialmente con respecto a la unión, debido a que, los atributos *Nombre, Apellido1 y Cédula* forman parejas de atributos correspondientes con el mismo dominio, pero no existe correspondencia de dominio con los atributos que almacenan el límite de crédito y el objetivo de ventas. Adicionalmente, para completar el ejercicio, se considera que un vendedor también puede ser cliente, para lo cual se incluye en la relación `CLIENTE` dos vendedores: Marcia Vera y Diego Loja.

El resultado de la unión externa entre las relaciones `CLIENTE` y `VENDEDOR` está formado por todas las tuplas de las dos relaciones que cumplan la condición de unión entre los atributos compatibles, las tuplas duplicadas se eliminan. Para el caso de los vendedores que también cumplen el rol de clientes, las tuplas tendrán valores en todos los atributos, en tanto que, para los clientes que no son vendedores, los atributos *ObjVentas* tienen valor nulo, y de igual manera, para los vendedores que no son clientes en el atributo *LímCrédito* se colocan valores *null*.

En la Figura 4.27 se muestra las relaciones y el resultado de la unión externa entre `CLIENTE` y `VENDEDOR`.

CLIENTE

Nombre	Apellido1	Cédula	LímCrédito
Víctor	Castro	0102030405	600
Juan	Polo	0122334455	1100
Elena	Tapia	0111555666	1500
Pablo	Brito	0123456789	Null
Marcia	Mora	0987654321	1000
Jaime	Pérez	0777888999	1200
Humberto	Pons	0456456456	2000
Alberto	Torres	0999998888	1200
Manuel	Castro	0123123123	600
Maricela	Vera	0555555555	1500
Diego	Loja	0111111111	1500

VENDEDOR

Nombre	Apellido1	Cédula	ObjVentas
Diego	Loja	0111111111	3000
Antonio	Calle	0222222222	2500
Lucía	Serrano	0333333333	2000
Arturo	Salto	0444444444	4000
Maricela	Vera	0555555555	3500

CLIENTE UNIÓN EXTERNA VENDEDOR

Nombre	Apellido1	Cédula	LímCrédito	ObjVentas
Víctor	Castro	0102030405	600	Null
Juan	Polo	0122334455	1100	Null
Elena	Tapia	0111555666	1500	Null
Pablo	Brito	0123456789	Null	Null
Marcia	Mora	0987654321	1000	Null
Jaime	Pérez	0777888999	1200	Null
Humberto	Pons	0456456456	2000	Null
Alberto	Torres	0999998888	1200	Null
Manuel	Castro	0123123123	600	Null
Maricela	Vera	0555555555	1500	3500
Diego	Loja	0111111111	1500	3000
Antonio	Calle	0222222222	Null	2500
Lucía	Serrano	0333333333	Null	2000
Arturo	Salto	0444444444	Null	4000

Figura 4.27. Operación unión externa sin repetición.

Cuando dos relaciones tienen atributos con el mismo nombre, pero su significado es diferente, la operación de unión externa genera resultados incorrectos. En estos casos es necesario antes de ejecutar la operación, renombrar estos atributos. Por ejemplo, si en las relaciones `CLIENTE` y `VENDEDOR` del ejercicio anterior, al atributo *Cédula* se le reemplaza por *Ciudad*, si bien, en estas dos relaciones el nombre coincide, sin embargo, el significado es diferente, considerando que, para los clientes representa la ciudad en donde tienen su negocio y para los vendedores la ciudad de nacimiento, razón por la cual, los atributos *Ciudad*, de las dos relaciones no son coincidentes.

Bajo este contexto, al realizar la operación de unión externa entre clientes y vendedores, se generan inconsistencias, sin que se pueda determinar si el dato de la ciudad corresponde al cliente o al vendedor, como se observa en la Figura 4.28, en donde para Diego Loja que cumple los roles de vendedor y cliente se crean dos tuplas con valores diferentes, Cuenca y Guayaquil para el atributo *Ciudad*.

VENDEDOR

Nombre	Apellido1	Ciudad	ObjVentas
Diego	Loja	Cuenca	3000
Antonio	Calle	Quito	2500
Lucía	Serrano	Cuenca	2000
Arturo	Salto	Cuenca	4000
Maricela	Vera	Cuenca	3500

CLIENTE

Nombre	Apellido1	Ciudad	LímCrédito
Víctor	Castro	Cuenca	600
Juan	Polo	Loja	1100
Elena	Tapia	Quito	1500
Pablo	Brito	Guayaquil	Null
Marcia	Mora	Cuenca	1000
Jaime	Pérez	Machala	1200
Humberto	Pons	Loja	2000
Alberto	Torres	Ambato	1200
Manuel	Castro	Quito	600
Maricela	Vera	Cuenca	1500
Diego	Loja	Guayaquil	1500

CLIENTE UNIÓN EXTERNA VENDEDOR

Nombre	Apellido1	Ciudad	LímCrédito	ObjVentas
Víctor	Castro	Cuenca	600	Null
Juan	Polo	Loja	1100	Null
Elena	Tapia	Quito	1500	Null
Pablo	Brito	Guayaquil	Null	Null
Marcia	Mora	Cuenca	1000	Null
Jaime	Pérez	Machala	1200	Null
Humberto	Pons	Loja	2000	Null
Alberto	Torres	Ambato	1200	Null
Manuel	Castro	Quito	600	Null
Maricela	Vera	Cuenca	1500	3500
Diego	Loja	Guayaquil	1500	Null
Diego	Loja	Cuenca	Null	3000
Antonio	Calle	Quito	Null	2500
Lucía	Serrano	Cuenca	Null	2000
Arturo	Salto	Cuenca	Null	4000

Figura 4.28. Operación unión externa con repetición.

Para obtener un resultado de la operación unión externa que diferencie el significado de la ciudad del cliente con la ciudad del vendedor, previamente se renombra el atributo *Ciudad* en cada una de las relaciones, para que éstos no formen parte de los atributos coincidentes *Nombre* y *Apellido1*. Por ejemplo, se asigna un nuevo nombre *Ciu_Negocio* y *Ciu_Nacim*, como se muestra en la Figura 4.29(a) y el resultado de la unión externa entre las dos relaciones se detalla en la Figura 4.29(b).

CLIENTE

Nombre	Apellido	Ciu_Negocio	LímCrédito

VENDEDOR

Nombre	Apellido1	Ciu_Nacim	ObjVentas

(a)

CLIENTE UNIÓN EXTERNA VENDEDOR

Nombre	Apellido1	Ciu_Negocio	Ciu_Nacim	LímCrédito	ObjVentas
Víctor	Castro	Cuenca	Cuenca	600	Null
Juan	Polo	Loja	Loja	1100	Null
Elena	Tapia	Quito	Quito	1500	Null
Pablo	Brito	Guayaquil	Guayaquil	Null	Null
Marcia	Mora	Cuenca	Cuenca	1000	Null
Jaime	Pérez	Machala	Machala	1200	Null
Humberto	Pons	Loja	Loja	2000	Null
Alberto	Torres	Ambato	Ambato	1200	Null
Manuel	Castro	Quito	Quito	600	Null
Maricela	Vera	Cuenca	Cuenca	1500	3500
Diego	Loja	Guayaquil	Guayaquil	1500	Null
Diego	Loja	Cuenca	Cuenca	Null	3000
Antonio	Calle	Quito	Quito	Null	2500
Lucía	Serrano	Cuenca	Cuenca	Null	2000
Arturo	Salto	Cuenca	Cuenca	Null	4000

(b)

Figura 4.29. (a) Relaciones con atributos renombrados. (b) Resultado de la unión externa.

4.6 CUESTIONARIO Y EJERCICIOS

Preguntas de repaso

1. Explique el objetivo de cada una de las operaciones unarias del álgebra relacional.

2. ¿Qué significa en el álgebra relacional la propiedad de cierre?

3. Explique la diferencia entre el álgebra relacional y el álgebra de conjuntos.

4. Explique el concepto *compatibilidad con respecto a la unión*, e indique en qué operaciones del álgebra relacional se aplica.

5. Proporcione un ejemplo de dos relaciones que tengan compatibilidad respecto de la unión y dos que estén en incompatibilidad respecto de la unión.

6. Mediante un ejemplo explique la operación producto cartesiano del álgebra relacional.

7. ¿Qué combinación de operaciones del álgebra relacional derivan de la operación JOIN?

8. ¿En qué se diferencia la combinación interna de la combinación externa?

9. Mediante un ejemplo explique la operación unión externa.

Ejercicios resueltos

Los siguientes ejercicios se plantean en función del caso de estudio **La Ferretería** que se detalla en la instancia de la base de datos de la Figura 2.43(b).

10. Recuperar el nombre y el apellido de los vendedores de sexo masculino que tienen un objetivo de ventas mayor a 2.500 dólares.

$$R1 \leftarrow \sigma_{\text{Sexo = "M" and ObjVentas > 2500}}(\text{VENDEDOR})$$

$$R2 \leftarrow \pi_{\text{Vnombre, Apellido}}(R1)$$

11. Al precio de todos los artículos incrementarles el 15%.

$$R1 \leftarrow \pi_{\text{Código, Descripción, Precio*1.15, Unidad, ExiMáx, ExiMín}}(\text{ARTÍCULO})$$

12. Seleccionar los clientes que tienen como representante al vendedor de código 1.

$$R1 \leftarrow \sigma_{CódRepre = 1}(CLIENTE)$$

$$R2 \leftarrow \pi_{Nombre,\ Apellido1}(R1)$$

13. Seleccionar el nombre y los apellidos de los clientes que tienen como representante al vendedor Diego Loja.

$$R1 \leftarrow \sigma_{Vnombre = \text{“Diego”}\ and\ Apellido = \text{“Loja”}}(VENDEDOR)$$

$$R2 \leftarrow R1 \bowtie_{CódigoVend = CódRepre}(CLIENTE)$$

$$R3 \leftarrow \pi_{Nombre,\ Apellido1,\ Apellido2}(R2)$$

14. Enlistar la cédula de ciudadanía y el nombre de los clientes que tienen un límite de crédito mayor a 1.000 dólares.

$$R \leftarrow \pi_{Cédula,\ Nombre}\ (\sigma_{LímCrédito > 1000}(CLIENTE))$$

15. Seleccionar el nombre y el primer apellido de los clientes que tienen como representante al vendedor Diego Loja y tienen un límite de crédito mayor o igual a 1.000 dólares.

$$R1 \leftarrow \sigma_{Vnombre = \text{“Diego”}\ and\ Apellido = \text{“Loja”}}\ (VENDEDOR)$$

$$R2 \leftarrow R1 \bowtie_{CódigoVend = CódRepre}(CLIENTE)$$

$$R3 \leftarrow \pi_{Nombre,\ Apellido1}\ (\sigma_{LímCrédito >= 1000}(R2))$$

16. Para todos los vendedores de sexo masculino, recuperar el nombre y el apellido de los clientes a quienes representan y tienen un límite de crédito mayor a 1.000 dólares.

$$R1 \leftarrow \sigma_{Sexo = \text{“M”}}(VENDEDOR)$$

$$R2 \leftarrow R1 \bowtie_{CódigoVend = CódRepre}\ CLIENTE$$

$$R3 \leftarrow \pi_{Nombre,\ Apellido1,}\ (\sigma_{LímCrédito > 1000}(R2))$$

17. Para cada cliente, recuperar su nombre y primer apellido y el nombre y primer apellido de quien lo recomendó.

```
CLIE NomClie, ApellClie, CédulaReco ← π Nombre, Apellido1, CédulaReco (CLIENTE)
RECO NomRec, ApellRec, Cédula ← π Nombre, Apellido1, Cédula(CLIENTE)
RELACIÓN ← CLIE ⋈ CédulaReco = Cédula RECO
RESULTADO ← π NomClie, ApellClie, NomReco, ApellRec(RELACIÓN)
```

Los resultados intermedios de esta consulta se muestran en la Figura 4.30.

CLIE

NomClie	ApelliClie	CédulaReco
Víctor	Castro	0122334455
Juan	Polo	0456456456
Elena	Tapia	0777888999
Pablo	Brito	0456456456
Marcia	Mora	0777888999
Jaime	Pérez	0122334455
Humberto	Pons	null
Alberto	Torres	0122334455
Manuel	Castro	0122334455

RECO

NomRec	ApellClie	Cédula
Víctor	Castro	0102030405
Juan	Polo	0122334455
Elena	Tapia	0111555666
Pablo	Brito	0123456789
Marcia	Mora	0987654321
Jaime	Pérez	0777888999
Humberto	Pons	0456456456
Alberto	Torres	0999998888
Manuel	Castro	0123123123

RELACIÓN

NomClie	ApellClie	CédulaReco	NomRec	ApellRec	Cédula
Víctor	Castro	0122334455	Juan	Polo	0122334455
Jaime	Pérez	0122334455	Juan	Polo	0122334455
Alberto	Torres	0122334455	Juan	Polo	0122334455
Manuel	Castro	0122334455	Juan	Polo	0122334455
Elena	Tapia	0777888999	Jaime	Pérez	0777888999
Marcia	Mora	0777888999	Jaime	Pérez	0777888999
Juan	Polo	0456456456	Humberto	Pons	0456456456
Pablo	Brito	0456456456	Humberto	Pons	0456456456

RESULTADO

NomClie	ApellClie	NomRec	ApellRec
Víctor	Castro	Juan	Polo
Jaime	Pérez	Juan	Polo
Alberto	Torres	Juan	Polo
Manuel	Castro	Juan	Polo
Elena	Tapia	Jaime	Pérez
Marcia	Mora	Jaime	Pérez
Juan	Polo	Humberto	Pons
Pablo	Brito	Humberto	Pons

Figura 4.30. Resultado de una consulta con relación recursiva.

18. Enlistar el número de cada una de las facturas de las compras realizadas por el cliente Pablo Brito.

R1 ← σ Nombre = "Pablo" and Apellido1 = "Brito"(CLIENTE)

R2 ← R1 ⋈ Cédula = CcCliente FACTURA

R3 ← π Número(R2)

19. Determinar cuántas facturas tiene el cliente Pablo Brito.

R1 ← σ Nombre = "Pablo" and Apellido1 = "Brito"(CLIENTE)

R2 ← R1 ⋈ Cédula = CcCliente(FACTURA)

R3 ← ℑcount Cédula(R2)

20. Enlistar el nombre de todos los artículos comprados en las diferentes facturas por el cliente Manuel Bonilla.

R1 ← σ Nombre = "Manuel" and Apellido1 = "Bonilla"(CLIENTE)

R2 ← R1 ⋈ Cédula = CcCliente(FACTURA)

R3 ← R2 ⋈ Número = FactNúmero(DETALLE)

R4 ← R3 ⋈ ArtCódigo = Código(ARTÍCULO)

R5 ← π Descripción(R4)

21. Enlistar el nombre y el apellido de los clientes a quienes facturó el vendedor Arturo Salto.

R1 ← σ Vnombre = "Arturo" and Apellido = "Salto"(VENDEDOR)

R2 ← R1 ⋈ CódigoVend = CódVendedor FACTURA

R3 ← R2 ⋈ CcCliente = Cédula CLIENTE

R4 ← π Nombre, Apellido1(R3)

22. Recuperar el nombre de las cargas familiares del vendedor Diego Loja.

R1 ← σ Vnombre = "Diego" and Apellido = "Loja"(VENDEDOR)

R2 ← R1 ⋈ CódigoVend = CódVendedor CARGAFAMILIAR

R3 ← π NomDep(R2)

23. Recuperar el nombre y el apellido de los vendedores que tienen cargas familiares.

R1 ← CARGAFAMILIAR ⋈ CódVendedor = CódigoVend VENDEDOR

R2 ← π Vnombre, Apellido(R1)

24. Para cada vendedor enlistar el nombre, el apellido y el número de cargas familiares.

R1 ← CódVendedor ℸ count CódVendedor (CARGAFAMILIAR)

R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR

R3 ← π Vnombre, Apellido, count CódVendedor(R2)

25. Enlistar el nombre y el apellido de los vendedores que tienen más de una carga familiar.

R1 ← CódVendedor ℸ count CódVendedor (CARGAFAMILIAR)

R2 ← σ count CódVendedor > 1(R1)

R3 ← R2 ⋈ CódVendedor = CódigoVend VENDEDOR

R4 ← π Vnombre, Apellido, count CódVendedor(R3)

26. Enlistar el nombre, el apellido y el sexo del vendedor e incluir el nombre y el sexo de su carga familiar.

```
R1 NomDep, CargaSexo, CódVendedor ← π NomDep, Sexo, CódVendedor (CARGAFAMILIAR)
R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR
R2 ← π Vnombre, Apellido, Sexo, NomDep, CargaSexo(R1)
```

El nombre del atributo *Sexo* se presenta tanto en la relación VENDEDOR como en la relación CARGA-FAMILIAR. Con el objeto de evitar ambigüedades con este nombre común para las dos relaciones, se utiliza la operación RENOMBRAR para cambiar el nombre del atributo *Sexo* en la relación CARGA-FAMILIAR y llamarlo *CargaSexo*.

27. Recuperar el nombre y el apellido de los vendedores que tienen cargas familiares con el mismo nombre y el mismo sexo.

```
R1 NomDep, C.Sexo, CódVendedor ← π NomDep, Sexo, CódVendedor (CARGAFAMILIAR)
R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR
R3 ← σ Vnombre = NomDep and Sexo = C.Sexo (R2)
R4 ← π Vnombre, Apellido (R3)
```

28. Seleccionar el nombre y el apellido de todos los vendedores que tienen únicamente cargas familiares de sexo femenino.

```
R1 ← π CódVendedor (σ Sexo = "M"(CARGAFAMILIAR))
R2 ← π CódVendedor (CARGAFAMILIAR)
R3 ← R2 – R1
R4 ← R3 ⋈ CódVendedor = CódigoVend VENDEDOR
R5 ← π Vnombre, Apellido (R4)
```

Los resultados intermedios de esta consulta se muestran en la Figura 4.31.

R1

CódVendedor
2
3

R2

CódVendedor
1
2
3

R3

CódVendedor
1

R4

CódVendedor	Vnombre	Apellido	CódigoVend	Sexo	. . .
1	Diego	Loja	1	M	. . .

R5

Vnombre	Apellido
Diego	Loja

Figura 4.31. Consulta que incluye la operación DIFERENCIA.

29. Recuperar el nombre y apellido de los vendedores que no tienen cargas familiares.

```
R1 ← π CódigoVend(VENDEDOR)
R2 ← π CódVendedor(CARGAFAMILIAR)
R3 ← R1 – R2
R4 ← R3 ⋈ CódVendedor = CódigoVend VENDEDOR
R5 ← π Vnombre, Apellido(R4)
```

30. Recuperar el nombre y el apellido de los vendedores que tienen al menos una carga familiar mujer.

```
R1 ← σ Sexo = "F"(CARGA_FAMILIAR)
R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR
R3 ← π Vnombre, Apellido(R2)
```

31. Enlistar el nombre y apellido de los vendedores que tienen como carga familiar a su esposa.

```
R1 ← σ Sexo = "F" and Relación = "Cónyuge"(CARGAFAMILIAR)
R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR
R3 ← π Vnombre, Apellido(R2)
```

32. Recuperar el nombre, el apellido y el número de facturas realizadas por cada vendedor.

```
R1 ← CódVendedor ℸ count Número (FACTURA)
R2 ← R1 ⋈ CódVendedor = CódigoVend VENDEDOR
R3 ← π Vnombre, Apellido, count Número(R2)
```

33. Determinar cuántos artículos diferentes se vendieron en la factura número 100.

```
R1 ← σ FactNúmero = 100(DETALLE)
R2 ← ℸ count ArtCódigo(R1)
```

34. Para cada uno de los artículos vendidos en la factura número 100, recuperar la descripción y el monto total.

```
R1 ← σ FactNúmero = 100 (DETALLE)
R2 ←  R1 ⋈ ArtCódigo = Código ARTÍCULO
R3 Descripción, MontoTotal ← π Descripción, Cantidad * Precio (R2)
```

Los resultados intermedios de esta consulta se muestran en la Figura 4.32.

R1

FactNúmero	ArtCódigo	Cantidad
100	2	5
100	9	1
100	10	5

R2

FactNúmero	ArtCódigo	Cantidad	Código	Descripción	Precio	Unidad	ExiMáx	ExiMín
100	2	5	2	Clavos	4,8	Kilo	20	5
100	9	1	9	Pintura	19,5	Galón	15	4
100	10	5	10	Lija	0,6	Unidad	30	10

R3

Descripción	MontoTotal
Clavos	24
Pintura	19,5
Lija	3

Figura 4.32. Resultado de subtotales por artículo en una factura.

A partir de esta consulta se puede calcular el monto total de la factura número 100, añadiendo la función suma.

```
R4 ← ℑ sum MontoTotal(R3)
```

35. Para todos los artículos vendidos, recuperar el código, la descripción, la unidad y las cantidades vendidas.

```
R1 ← ArtCódigo ℑ sum Cantidad(DETALLE)
R2 ← R1 ⋈ ArtCódigo = Código ARTÍCULO
R3 ← π Código, Descripción, Unidad, sum Cantidad(R2)
```

36. Enlistar los diferentes límites de crédito de los clientes.

```
R1 ← π LímCrédito(CLIENTE)
```

37. Recuperar la cédula de ciudadanía del cliente y el monto total de compras realizadas en La Ferretería.

```
R1 ← FACTURA ⋈ Número = FactNúmero DETALLE
R2 ← R1 ⋈ ArtCódigo = Código ARTÍCULO
R3 Cédula, Total ← π CcCliente, Cantidad * Precio(R2)
R4 ← CcCliente ℑ Sum (Total)(R3)
```

El resultado de esta consulta se muestra en la Figura 4.33.

R4

CcCliente	SumTotal
0102030405	46,5
0111555666	30
0122334455	262
0123123123	218
0123456789	18
0456456456	84
0777888999	38.4
0987654321	63
0999998888	26.4

Figura 4.33. Resultado de la consulta con la cédula y el monto total de compras.

38. Determinar el nombre y el primer apellido del cliente que tiene el mayor monto total (dólares) de compras.

Para realizar este ejercicio se parte del resultado R4 de la consulta 37, que se muestra en la Figura 4.33.

```
R5 ← ℸ Máx(SumTotal)(R4)
R6 ← R4 ⋈ SumTotal  = Máx(SumTotal)R5
R7 ← R6 ⋈ CcCliente = Cédula CLIENTE
R8 ← π Nombre, Apellido1, Máx(SumTotal)(R7)
```

En la función que tiene como resultado la relación `R5` se obtiene la tupla con el máximo valor de 262 dólares correspondientes a la suma total de las facturas agrupadas por la cédula de identidad; sin embargo, en esta operación se pierde la columna de la cédula de identidad del cliente, que se la puede recuperar mediante la operación combinación entre las relaciones `R4` y `R5`, para luego concatenar con la relación `CLIENTE` y obtener su nombre y apellido.

Los resultados intermedios de esta consulta se muestran en la Figura 4.34.

R5

Máx(SumTotal)
262

R6

CcCliente	SumTotal	Máx(SumTotal)
0122334455	262	262

R7

CcCliente	SumTotal	Máx(SumTotal)	Nombre	Apellido1	Apellido2	Cédula	. . .
0122334455	262	262	Juan	Polo	Ávila	0122334455	. . .

R8

Nombre	Apellido1	Máx(SumTotal)
Juan	Polo	262

Figura 4.34. Resultados intermedios de la consulta que calcula el mayor monto de compras.

39. Para el cliente que tiene la cédula de ciudadanía 0102030405, enlistar el número de facturas y el monto total.

```
R1 ⟵ σ CcCliente =  0102030405(FACTURA)
R2 ⟵ R1 ⋈ Número = FactNúmero DETALLE
R3 ⟵ R2 ⋈ ArtCódigo = Código ARTÍCULO
R4 Número, Monto ⟵ π Número, Cantidad * Precio(R3)
R5 ⟵ Número ℸ Sum (Monto)(R4)
```

40. Recuperar el monto total de cada una de las facturas del cliente Víctor Castro.

Este ejercicio es similar al número 39, su diferencia radica en el hecho de que en esta consulta se parte del nombre y el apellido del cliente, lo que implica incluir como parte de las secuencia de operaciones la instrucción de `SELECCIÓN`, para recuperar la cédula del cliente en función de los atributos nombre y apellido.

R1 ← σ Nombre = "Víctor" and Apellido = "Castro"(CLIENTE)

R2 ← R1 ⋈ Cédula = CcCliente FACTURA

R3 ← R2 ⋈ Número = FactNúmero DETALLE

R4 ← R3 ⋈ ArtCódigo = Código ARTÍCULO

R5 Número, Monto ← π Número, Cantidad * Precio(R4)

R6 ← Número ℸ Sum(Monto)(R5)

41. Recuperar el número de artículos diferentes vendidos en cada factura.

R1 ← FactNúmero ℸ Count(FactNúmero)(DETALLE)

42. Recuperar el nombre de los clientes que compraron cemento, y el nombre de quien los vendió.

R1 ← σ Descripción= "Cemento"(ARTÍCULO)

R2 ← R1 ⋈ Código = ArtCódigo DETALLE

R3 ← R2 ⋈ FactNúmero = Número FACTURA

R4 ← R3 ⋈ CcCliente = Cédula CLIENTE

R5 ← R4 ⋈ CódVendedor = CódigoVend VENDEDOR

R6 ← π Nombre, Apellido1, Vnombre, Apellido(R5)

43. Recuperar el nombre y el apellido de los clientes y sus números telefónicos.

R1 ← CLIENTE ⋈ Cédula = CédulaCli CLIENTE_TELÉF

R2 ← π Nombre, Apellido, NúmeroTeléf(R1)

44. Recuperar el nombre y el apellido1 de los clientes que tienen un límite de crédito mayor a 1.000 dólares y han comprado más de una factura.

R1 ← CcCliente ℸ count (CcCliente)(FACTURA)

R2 ← σ Count CcCliente > 1(R1)

R3 ← R2 ⋈ CcCliente = Cédula CLIENTE

R4 ← σ LímCrédito > 1000(R3)

R5 ← π Nombre, Apellido1(R4)

Los resultados intermedios de esta consulta se muestran en la Figura 4.35.

R1

CcCliente	Count CcCliente
0102030405	1
0111555666	1
0122334455	2
0123123123	3
0123456789	1
0456456456	1
0777888999	1
0987654321	1
0999998888	1

R2

CcCliente	Count CcCliente
0122334455	2
0123123123	3

R3

CcCliente	Count CcCliente	Nombre	Apellido1	Apellido2	Cédula	LímCrédito	. . .
0122334455	2	Juan	Polo	Ávila	0122334455	1100	…
0123123123	3	Manuel	Castro	León	0123123123	600	…

R4

CcCliente	Count CcCliente	Nombre	Apellido1	Apellido2	Cédula	LímCrédito	. . .
0122334455	2	Juan	Polo	Ávila	0122334455	1100	…

R5

Nombre	Apellido1
Juan	Polo

Figura 4.35. Resultados intermedios de la consulta 44.

Una segunda alternativa para esta consulta se puede resolver mediante el uso de la operación binaria intersección.

$$R1 \leftarrow \pi_{\text{Nombre, Apellido1}}(\sigma_{\text{LímCrédito} > 1000}(\text{CLIENTE}))$$

$$R2 \leftarrow {}_{\text{CcCliente}}\tau_{\text{count (CcCliente)}}(\text{FACTURA})$$

$$R3 \leftarrow \sigma_{\text{Count CcCliente} > 1}(R2)$$

$$R4 \leftarrow \pi_{\text{Nombre, Apellido1}}(R3 \bowtie_{\text{CcCliente = Cédula}} \text{CLIENTE})$$

$$R5 \leftarrow R1 \cap R4$$

Los resultados del proceso de esta consulta se detallan en la Figura 4.36.

R1

Nombre	Apellido1
Juan	Polo
Elena	Tapia
Jaime	Pérez
Humberto	Pons
Alberto	Torres

R2

CcCliente	Count CcCliente
102030405	1
111555666	1
122334455	2
123123123	3
123456789	1
456456456	1
777888999	1
987654321	1
999998888	1

R3

CcCliente	Count CcCliente
122334455	2
123123123	3

R4

Nombre	Apellido1
Juan	Polo
Manuel	Castro

R5

Nombre	Apellido1
Juan	Polo

Figura 4.36. Alternativa con la operación intersección.

Si a esta consulta se le modifica la condición "y" por "o", es decir recuperar el nombre y el apellido de los clientes que tienen un límite de crédito mayor a 1.000 dólares o tienen más de una factura, la secuencia de operaciones es igual a la alternativa dos, cambiando únicamente la última instrucción, en remplazo de la operación intersección (∩) se coloca unión (∪).

$$R5 \leftarrow R1 \cup R4$$

El resultado de la consulta se muestra en la Figura 4.37, que incluye todos los clientes que cumplan una de las dos condiciones o las dos.

R5

Nombre	Apellido1
Juan	Polo
Elena	Tapia
Jaime	Pérez
Humberto	Pons
Alberto	Torres
Manuel	Castro

Figura 4.37. Operación R1 ∪ R4.

Ejercicios propuestos

45. Recuperar el nombre y la cédula de los clientes que, al menos en una de sus facturas, ha comprado un monto que supera el 10 % de su límite de crédito.

46. Recuperar el nombre y el código de los vendedores cuyo monto total de sus ventas supera el 40% de su objetivo.

47. Determinar el nombre del vendedor que más facturas ha realizado.

48. Determinar el nombre del vendedor que más ha vendido.

49. Recuperar el nombre de los vendedores que tienen facturas con un monto total mayor al promedio de ventas de todas las facturas.

50. Para la factura con el mayor monto de venta, enlistar el nombre y el apellido del cliente y de su vendedor.

51. Enlistar el nombre y apellido1 de los clientes que tienen un límite de crédito mayor a 1000 dólares y tienen como representante a la vendedora Lucía Serrano.

52. Enlistar el nombre y apellido de los clientes que tienen un límite de crédito mayor a 1.000 dólares y en total han comprado un monto mayor a su límite de crédito.

53. Recuperar el nombre y el apellido de los vendedores que tienen un objetivo de ventas mayor a 2.500 dólares o en total han vendido un monto mayor a 1.000 dólares.

54. Enlistar la cédula de todos los clientes recomendados directamente por el cliente Humberto Pons.

55. Enlistar el nombre de los clientes que tienen como representantes a los vendedores de código 1, 3 o 4.

56. Enlistar el nombre, el apellido y la cédula de ciudadanía de todos los clientes cuyos apellidos están comprendidos entre los apellidos Mora y Tapia.

57. Para cada factura, recuperar su número, el cliente que compró, el vendedor que realizó y el monto total, todo ordenado por el apellido del cliente.

58. Para los clientes representados por el vendedor Diego Loja, determinar el límite de crédito máximo, el mínimo, el límite de crédito total, y el número de clientes.

59. Para cada vendedor que tenga más de dos clientes a quien representar, recuperar el límite de crédito máximo, mínimo, el límite de crédito total y el número de clientes a quien representa.

60. Recuperar el monto total de las compras realizadas por cada cliente.

61. Recuperar el monto total de las compras realizadas por cada cliente en cada factura.

62. Para cada factura con más de dos artículos vendidos, recuperar el nombre del cliente y el nombre del vendedor.

63. Determinar cuántos clientes de La Ferretería recomendaron a otros clientes.

64. Determinar cuántos clientes de La Ferretería fueron recomendados.

65. Determinar cuántos clientes de La Ferretería no fueron recomendados.

66. Determinar a cuántos clientes recomendó Juan Polo.

67. Recuperar el nombre y el primer apellido de los clientes que recomendó Juan Polo.

68. Recuperar el nombre de los clientes que tienen un límite de crédito mayor a todos los límites de crédito de los clientes representados por el vendedor de código 1.

69. Recuperar el nombre de los clientes que tienen un límite de crédito mayor al menos a uno de los límites de crédito de los clientes representados por el vendedor de código 1.

70. Recuperar el promedio del límite de crédito de los clientes agrupados por su vendedor.

71. Recuperar el nombre de los vendedores en donde todos sus representados tienen un límite de crédito mayor a 1.000 dólares.

72. Enlistar el nombre de los clientes que tienen un límite de crédito mayor al promedio del límite de crédito de los clientes representados por el vendedor de código 3.

73. Enlistar el nombre de los clientes que tienen un límite de crédito mayor al promedio del límite de crédito de los clientes representados por la vendedora Lucía Serrano.

74. Recuperar el nombre de los vendedores cuyo objetivo de ventas es mayor al promedio del objetivo de ventas de todos los vendedores.

75. Determinar el mayor límite de créditos promedio de los clientes agrupados por su representante.

SQL

Este Capítulo presenta los elementos y conceptos principales del estándar SQL para los sistemas de gestión de bases de datos relacionales comerciales (SGBDR). No se pretende dar una guía de usuario o un manual de referencia, el objetivo es proporcionar las principales características de SQL con ejemplos prácticos basados en el caso de estudio La Ferretería.

En la Sección 5.1 se describe brevemente la historia y los estándares de SQL. En la Sección 5.2 se analizan las instrucciones para la recuperación de datos, partiendo de consultas simples para luego pasar a consultas más complejas con acceso a varias tablas y con el uso de múltiples condiciones; en esta Sección también se describe la construcción de consultas con el uso de funciones de agregación y concluye con la revisión de las diferentes opciones para subconsultas. La Sección 5.3 explica las sentencias para la actualización de datos denominadas consultas de acción, encargadas de insertar, modificar y eliminar datos. En la Sección 5.4 se revisará uno de los principales papeles de un SGBD, el mantenimiento de la integridad de los datos almacenados. La Sección 5.5 describe los comandos básicos del lenguaje de definición de datos (LDD), para crear, modificar y eliminar una tabla. El capítulo concluye con la Sección 5.6 que presenta una serie de ejercicios resueltos y propuestos.

5.1 INTRODUCCIÓN A SQL

El lenguaje estructurado de consultas (SQL por sus siglas en inglés *Structured Query Language)* se utiliza para interactuar con una base de datos relacional y sirve para organizar, gestionar y recuperar los datos almacenados.

El propósito inicial de SQL estuvo orientado a las consultas, de ahí el nombre de lenguaje estructurado de consultas; sin embargo, con su desarrollo se incluyen otras funciones orientadas a definir estructuras, modificar datos, definir controles de acceso y especificaciones de integridad.

Desde el punto de vista del contexto de un lenguaje estructurado como Java o C++, SQL en realidad no es un lenguaje informático completo, no contiene sentencias para el control del programa, como la instrucción `IF` para el manejo de condiciones, o la sentencias `FOR` para el control de flujos; SQL es un lenguaje con instrucciones especializadas cuya sintaxis se parece a las frases escritas en inglés.

La versión original de SQL denominada "Sequel", fue desarrollada a principios de los años 70 por IBM como parte del proyecto System/R, cuyo objetivo fue el de comprobar la operatividad del concepto relacional y aportar con experiencias en la implementación de un sistema real de gestión de bases de datos relacional (SGBD). El lenguaje "Sequel" fue evolucionando y pasó a denominarse SQL, convirtiéndose en la actualidad en el estándar para el manejo de bases de datos relacionales.

Uno de los alcances más importantes para la aceptación de SQL, es la estandarización oficial del lenguaje adoptada por los organismos internacionales, el Instituto Nacional Americano de Normalización (ANSI por sus siglas en inglés *American National Standard Institute*) y la Organización Internacional para la Normalización (ISO por sus siglas en ingles *International Standards Organization*), que en 1986 publicaron la primera norma de SQL denominada SQL-86. En 1989, ANSI publicó una extensión de la norma denominada SQL89. Con el surgimiento de nuevas características del lenguaje se fueron aprobando nuevas versiones de la norma, SQL-92, SQL:1999, SQL:2003, SQL:2006 hasta la última versión SQL:2008.

SQL es un lenguaje de bases de datos que cuenta con varios componentes para definir datos, consultas y actualizaciones. El componente LDD (Lenguaje de definición de datos) incluye comandos para creación, modificación y eliminación de tablas; el componente LMD (Lenguaje de manipulación de datos) proporciona comandos para consultar, insertar, modificar y eliminar datos. Incluye también comandos para especificar las restricciones de integridad, la definición de vistas, el control de transacciones y autorizaciones.

En este Capítulo se presentan las principales características del LMD y una revisión general del LDD, basados principalmente en la norma SQL-92, sin hacer referencia a un gestor de bases de datos específico. Es posible que ciertas características o ejemplos que se describen no funcionen en el sistema de gestión de bases de datos que el lector utiliza, razón por la cual, se recomienda consultar el manual de usuario para determinar las características particulares del gestor.

5.2 RECUPERACIÓN DE DATOS

5.2.1 Consultas simples

La declaración más potente y compleja dentro del lenguaje SQL, sin duda es la instrucción `SELECT`, que en conjunto con otras palabras claves y cláusulas, se usa para expresar una consulta a la base de datos. Es ilimitada la forma de ver la información, casi cualquier pregunta con respecto a quién, qué, dónde, cuándo, se puede responder con la instrucción `SELECT`, siempre y cuando se cuente con un buen diseño de la base de datos y una recopilación adecuada de los datos, se puede obtener la respuesta que se necesita para tomar decisiones acertadas en la organización.

Muchas de las técnicas que se aprenderán sobre la instrucción **SELECT** serán aplicadas cuando se revisen las instrucciones **UPDATE, INSERT** y **DELETE** diseñadas para actualizar datos.

La instrucción **SELECT** es la base de todas las preguntas que se plantean a la base de datos y ofrece muchas opciones, desde consultas simples que recuperan datos de una única tabla, tema que será tratado en esta sección, hasta consultas complejas a varias tablas, con resúmenes y subconsultas que se revisarán en las siguientes secciones.

Dependiendo del sistema de gestión de bases de datos (SGBD), una instrucción **SELECT** puede ejecutarse interactivamente desde una línea de comando, o desde el interior de un bloque de código de un programa. Independientemente de las dos alternativas que se escoja para ejecutar una consulta, la sintaxis de la instrucción **SELECT** es siempre la misma.

En la Figura 5.1 se muestra el proceso de una consulta en SQL, partiendo de la solicitud que se realiza al **SGBD**, luego, en función de esta solicitud el **SGDB** ejecuta una acción sobre la base de datos para finalmente obtener el resultado. El resultado de una consulta en **SQL** es una tabla de datos. Si la instrucción **SELECT** es interactiva, el resultado se muestra en la pantalla del computador, y si ésta se realiza mediante programa, la tabla resultante es devuelta por el **SGBD** al programa.

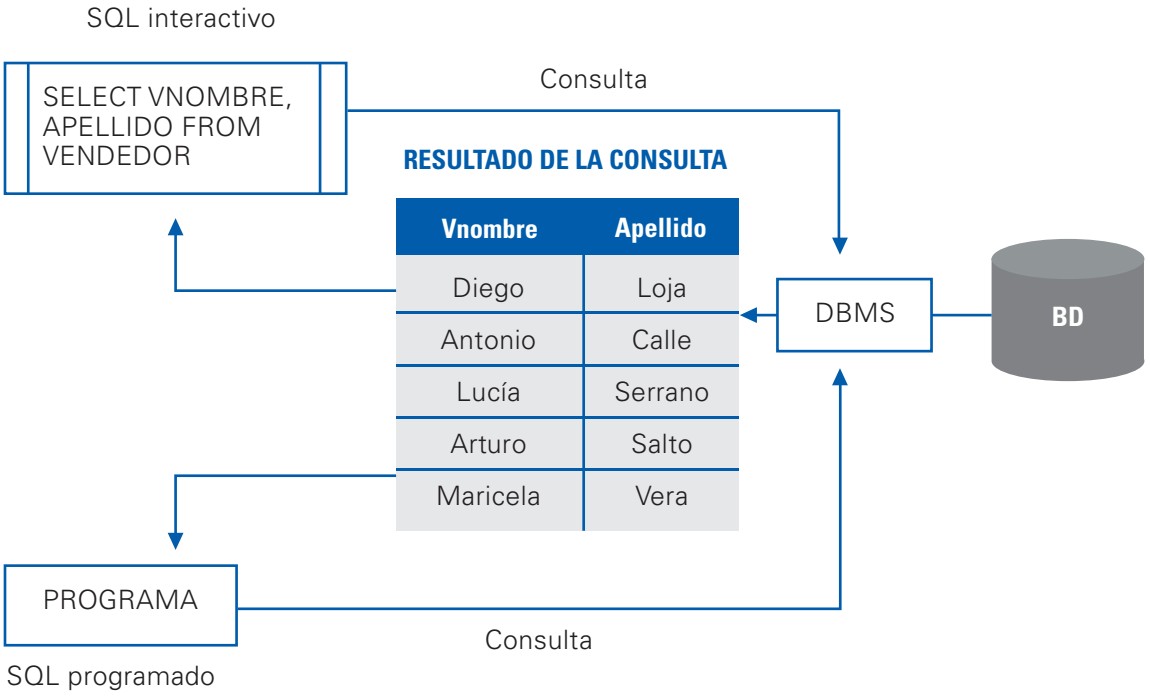


Figura 5.1. Proceso de una consulta en SQL.

Una instrucción SELECT se compone de varias palabras claves conocidas como cláusulas. En la Figura 5.2 se muestra el formato general simplificado de la instrucción SELECT que consiste en seis cláusulas, de las cuales SELECT y FROM son obligatorias, en tanto que las restantes son opcionales.

```
SELECT [ALL, DISTINCT] <lista de selección>
FROM   <lista de tablas>
WHERE <condición>
GROUP BY <nombre de la columna>
HAVING <condición de búsqueda>
ORDER BY <nombre de columna> ASC, DESC.
```

Figura 5.2. Formato de la instrucción SELECT.

A continuación se presenta un breve resumen de las cláusulas de la instrucción SELECT:

- La cláusula SELECT es absolutamente necesaria y se utiliza para enlistar los elementos de datos o columnas que se desean recuperar en el conjunto de resultados de la consulta. Los elementos pueden ser columnas extraídas de las tablas especificadas en la cláusula FROM o también, pueden ser, el resultado de expresiones matemáticas o de funciones de agregación.
- La cláusula FROM de la instrucción SELECT también es obligatoria. Se usa para especificar las tablas desde donde se extraen los datos de las columnas que han sido enumeradas en la cláusula SELECT.
- La cláusula WHERE es opcional y se utiliza para filtrar las filas de datos en los resultados de una consulta. La palabra clave WHERE es seguida por una expresión que técnicamente es conocida como *predicado* y se evalúa a través de un operador, obteniendo los resultados verdadero, falso o desconocido.
- La cláusula GROUP BY se utiliza para dividir la información en grupos distintos. Especifica una consulta de resumen o agregación, agrupando filas idénticas y generando una fila de resumen de los resultados de cada grupo. La cláusula GROUP BY es opcional.
- La cláusula HAVING permite aplicar una condición de búsqueda a los grupos de datos definidos por la cláusula GROUP BY. El resultado contiene todos los grupos que satisfacen la condición. No confundir las cláusulas WHERE y HAVING. La primera permite, seleccionar las filas antes de formar grupos, en tanto que, la segunda recupera ciertos grupos formados por la cláusula GROUP BY.

- La cláusula `ORDER BY` es opcional y se utiliza para ordenar los resultados de una consulta sobre la base de una o más columnas de ordenamiento. Si se prescinde de la cláusula `ORDER BY`, las filas de la tabla resultante se presentan en un orden indeterminado.

Antes de revisar de manera detallada la instrucción `SELECT`, es necesario tener clara la diferencia entre *dato e información*. En esencia un dato es lo que se almacena en la base de datos y la *información* es lo que se recupera de la base de datos. (Groff J., Weinberg P., 2003) (Viescas J., Hernández M., 2014, pág. 79). No obstante la información puede ser proporcionada únicamente si existen los datos adecuados en la base de datos y si ésta ha sido debidamente estructurada.

Por ejemplo, en la Figura 5.3 se muestra una colección de datos, en donde es difícil determinar qué representan: 0122334455, Bolívar 5-67, o cualquiera de los otros datos de cada una de las filas de las diferentes tablas de la base de datos. No hay forma de saberlo hasta procesarlos, de manera que sean significativos y se conviertan en información.

Lucía	Serrano	3	F	03-may-11	2000	Null

Juan	Polo	Ávila	0122334455	Bolívar 5-67	1100	0456456456	1	02-mar-13

6	Perfil T	15,0	Unidad	10	3
8	Perfil G	16,0	Unidad	10	3

101	13-ene-14	3	0122334455

101	6	4
101	8	5

Figura 5.3. Ejemplo de datos relacionados con una factura.

En función de los datos de la Figura 5.3, se ejecuta una instrucción `SELECT` que manipula los datos y genera información relacionada con la factura número 101, el cliente, el vendedor, los artículos comprados, la fecha y los valores facturados. En la Figura 5.4 se muestra un ejemplo de información.

La Ferretería

Factura Nro: 101 **Vendedor**: Lucía Serrano

Cliente: Juan Polo **Fecha:** 13-ene-14

Cédula: 0122334455

Artículo	Cantidad	Precio	Total
Perfil T	4	15,00	60,00
Perfil G	5	16,00	80,00
		TOTAL	140,00

Figura 5.4. Ejemplo de datos de la factura tratados como información.

5.2.1.1 Cláusula `SELECT` y `FROM`

Para una consulta simple la instrucción `SELECT` requiere únicamente de dos cláusulas. La cláusula `SELECT` seguida de una lista de elementos de selección separados por comas, permite especificar los elementos de datos que se desean recuperar en una consulta. Cada elemento de selección genera en el resultado una única columna en el orden especificado de izquierda a derecha. Los elementos de selección pueden ser: el nombre de una columna, una expresión o una constante como se verá posteriormente.

La cláusula `FROM` es obligatoria en una consulta y está seguida de una <lista de tablas> que especifica las tablas fuentes necesarias para procesar la consulta.

5.2.1.2 Selección con una o varias columnas

La instrucción de consulta más simple contiene en la lista de elementos de selección únicamente una columna. Por ejemplo, enlistar el nombre de todas las cargas familiares. La instrucción se escribe de la siguiente manera:

```
SELECT NomDep
FROM CARGAFAMILIAR
```

NomDep
María
Isabel
Antonio
Maricela
Teodoro

SQL examina cada fila de la tabla especificada en la cláusula **FROM** y toma el valor de la columna requerida en el **SELECT**, devolviendo como resultado una fila por cada fila de datos de la tabla.

Si se necesita consultar varias columnas de una tabla, en la lista de elementos de selección se especifica el nombre de las columnas requeridas, separadas por comas. Por ejemplo, para obtener un listado con el nombre, la fecha de nacimiento y la relación de las cargas familiares.

```
SELECT NomDep, FechaNac, Relación
FROM CARGAFAMILIAR
```

NomDep	FechaNac	Relación
María	27-oct-04	Hija
Isabel	03-dic-12	Hija
Antonio	16-abr-02	Hijo
Maricela	18-may-92	Cónyuge
Teodoro	25-oct-88	Cónyuge

5.2.1.3 Selección con columnas calculadas

SQL permite incluir en la lista de selección de la cláusula **SELECT**, expresiones aritméticas como suma, resta, multiplicación y división, que se denominan ***columnas calculadas***. Es posible utilizar paréntesis para generar expresiones más complejas. Las columnas de la tabla referenciadas en la lista de selección deben ser de tipo numérico, caso contrario si una columna contiene datos tipo

texto, SQL advertirá un error, sin embargo, existen ciertos productos SQL que facilitan el uso de operaciones sobre cadenas de caracteres.

Por ejemplo, si se requiere un listado de los vendedores, que incluya, el nombre, el apellido y el objetivo de ventas incrementado en un veinte por ciento, la consulta se expresa de la siguiente manera:

```
SELECT Vnombre, Apellido, ObjVentas * 1.2
FROM VENDEDOR
```

Vnombre	Apellido	ObjVentas * 1.2
Diego	Loja	3600
Antonio	Calle	3000
Lucía	Serrano	2400
Arturo	Salto	4800
Maricela	Vera	4200

El lenguaje SQL, para procesar esta consulta, accede a la tabla `VENDEDOR` y por cada fila de la tabla crea una fila de resultados. Para generar las dos primeras columnas, toma directamente los datos de la tabla `VENDEDOR`, en tanto que, para la columna calculada toma el dato *ObjVentas* de cada fila y le incrementa el veinte por ciento.

En una columna calculada es posible construir expresiones aritméticas que incluyan únicamente operaciones con columnas. Por ejemplo, la siguiente consulta calcula la diferencia entre la existencia máxima y existencia mínima de cado producto:

```
SELECT Descripción, ExiMáx – ExiMín
FROM ARTÍCULO
```

Descripción	ExiMáx - ExiMín
Cemento	40
Clavos	65
Alambre	15
Perfil T	7
Perfil L	10
Perfil G	7
Pintura	11
Lija	20
Suelda	5

Con el objeto de facilitar la interpretación y lectura de los resultados de una consulta, SQL permite incluir constantes en la cláusula `SELECT` como un elemento de la lista de selección. En el siguiente ejemplo se ilustra este concepto.

```
SELECT      Descripción, ´Diferencia de existencia:´,
ExiMáx – ExiMín
FROM  ARTÍCULO
```

Descripción	Diferencia de existencia:	ExiMáx – ExiMín
Cemento	Diferencia de existencia:	40
Clavos	Diferencia de existencia:	65
Alambre	Diferencia de existencia:	15
Perfil T	Diferencia de existencia:	7
Perfil L	Diferencia de existencia:	10
Perfil G	Diferencia de existencia:	7
Pintura	Diferencia de existencia:	11
Lija	Diferencia de existencia:	20
Suelda	Diferencia de existencia:	5

El resultado de la consulta muestra tres columnas, la primera contiene valores obtenidos de la tabla ARTÍCULO, la segunda contiene una cadena de 26 caracteres *´Diferencia de existencia´* y la tercera corresponde a una columna calculada.

Una expresión también puede ser una secuencia de expresiones separadas por paréntesis, la evaluación se hace desde los pares de paréntesis interiores hacia los exteriores. En la sección de consultas con multitablas que se verá más adelante, se revisarán ejemplos de expresiones aritméticas complejas que se utilizan en las columnas calculadas.

5.2.1.4 Selección de todas las columnas (SELECT *)

No existe un límite para el número de columnas que se pueden especificar en la cláusula SELECT. En efecto, es posible enlistar todas las columnas de una tabla fuente. SQL permite utilizar un asterisco (*) que representa la abreviatura de "*todas las columnas*" de la tabla, evitando tener que enlistar explícitamente los nombres de cada una. Por ejemplo, si se quiere mostrar todos los datos de los vendedores.

```
SELECT *
FROM   VENDEDOR
```

Vnombre	Apellido	CódigoVend	Sexo	FechaContrato	ObjVentas	Teléfono
Diego	Loja	1	M	03-mar-12	3000	2472713
Antonio	Calle	2	M	12-dic-11	2500	2876549
Lucía	Serrano	3	F	03-may-11	2000	Null
Arturo	Salto	4	M	15-jul-12	4000	2887643
Maricela	Vera	5	F	18-ago-08	3500	4657890

Como resultado de la consulta se obtiene siete columnas en el mismo orden de izquierda a derecha que la tabla fuente VENDEDOR.

Existen ciertos SGBD que consideran al (*) como un elemento más de la lista de selección, pudiendo escribirse consultas como la que se muestra a continuación, en la que se incluye una columna calculada.

```
SELECT      *, (ObjVentas * 1.2)
FROM   VENDEDOR
```

5.2.1.5 Eliminar filas duplicadas (DISTINCT)

La opción por omisión de la instrucción SELECT es ALL, que indica explícitamente que las líneas duplicadas no se eliminan. Al mantenerse por defecto todos los datos, la palabra clave ALL es innecesaria en la instrucción, es por este motivo que en los ejemplos no se incluye.

Es posible eliminar las filas duplicadas de los resultados de una consulta, para ello se incluye la palabra clave DISTINCT después de la palabra clave SELECT y justo antes de las lista de selección.

En resumen, una consulta con DISTINCT elimina del resultado las filas duplicadas; por el contrario, una consulta con ALL no lo hace. Por ejemplo, para enlistar los límites de crédito de los clientes, la consulta se puede escribir como:

```
SELECT LímCrédito
FROM CLIENTE
```

LímCrédito
600
1100
1500
Null
1000
1200
2000
1200
600

El resultado de la consulta genera 9 filas, con filas repetidas para los valores de 600 y 1200; sin embargo, si se necesita un listado de todos los límites de crédito diferentes asignados a los clientes, la instrucción se escribe como:

```
SELECT DISTINCT LímCrédito

FROM CLIENTE
```

LímCrédito
600
1100
1500
Null
1000
1200
2000

El resultado corresponde a una versión de la consulta anterior, en la que se incluye la palabra clave `DISTINCT`. La instrucción genera 7 filas que corresponden al resultado esperado. SQL en primer lugar genera una lista de todo el conjunto de resultados (incluidos los repetidos), para luego eliminar las filas duplicadas y formar la respuesta. Los valores Null que se incluyen serán analizados posteriormente cuando se trate el tema de valores nulos.

5.2.1.6 Selección de filas (`WHERE`)

Las opciones de la instrucción `SELECT` que se han revisado recuperan todas las filas de una tabla y los utiliza en el resultado; pero si se requiere encontrar solo la fila o las filas que se aplican a una persona en especial, a un lugar específico, a un valor numérico en particular o un rango de fechas, SQL incorpora la cláusula `WHERE` a la instrucción `SELECT`, que se utiliza para especificar la condición de búsqueda de las filas que se desean recuperar.

Por ejemplo, la siguiente instrucción recupera el nombre y el apellido de los clientes que tienen un límite de crédito mayor a 1.400 dólares.

```
SELECT Nombre, Apellido1, LímCrédito

FROM  CLIENTE

WHERE LímCrédito > 1400
```

Nombre	Apellido1	LímCrédito
Elena	Tapia	1500
Humberto	Pons	2000

Para resolver la consulta SQL analiza cada una de las filas de la tabla, y a cada valor correspondiente de la columna especificada en la condición de búsqueda (en nuestro caso *LímCrédito*) le aplica la condición, que puede producir los siguientes tres resultados: verdadero (`TRUE`), falso (`FALSE`) o desconocido (`NULL`).

Si la condición de búsqueda es `TRUE`, la fila correspondiente se incluye en el resultado de la consulta. En nuestro ejemplo, los límites de crédito de Elena Tapia y Humberto Pons son mayores a 1.400 dólares, en consecuencia se incluyen en el resultado.

Cuando la condición de búsqueda es `FALSE`, la fila se descarta del resultado, por ejemplo, en la consulta que se analiza, se excluyen del resultado todas las filas de los clientes que tienen un límite de crédito menor a 1.400 dólares.

Si la condición de búsqueda produce un valor `NULL`, como el caso del cliente Pablo Brito, la fila es excluida del resultado.

En la Figura 5.5 se muestra la función que realiza la cláusula `WHERE` para procesar la consulta. La condición de búsqueda opera como un filtro para cada una de las filas, si la condición es verdadera pasa el filtro y la fila forma parte del resultado, si la condición de búsqueda es falsa o desconocida la fila se excluye del resultado.

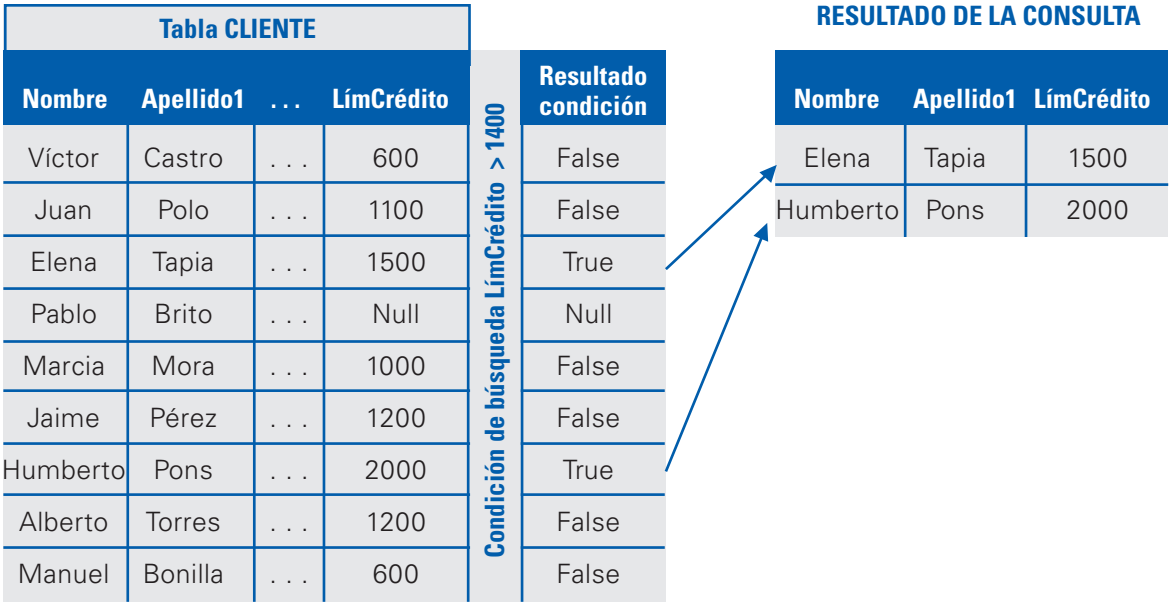


Figura 5.5. Función de la condición de búsqueda de la cláusula `WHERE`.

5.2.2 Predicados de la cláusula WHERE

En este apartado se revisa las cinco condiciones básicas de una consulta denominadas predicados en el estándar ANSI/ISO y son los siguientes: de comparación, de rango, de pertenencia a conjunto, de coincidencia de patrón y de valores nulos.

5.2.2.1 Comparación

La condición de búsqueda más común es la que usa un predicado de comparación y está formada por dos expresiones compatibles, separadas por un operador de comparación (=, <=, >=, <, >, <>), como se indica a continuación.

```
Expresión 1  (operador de comparación)  expresión 2
```

La comparación de las dos expresiones genera uno de los tres resultados: `TRUE` si la comparación es cierta, `FALSE` si la comparación es falsa y `NULL` si alguna de las dos expresiones devuelve un valor nulo. La condición que se usa con mayor frecuencia es la que compara el valor de una columna con una constante. Por ejemplo, enlistar los vendedores que fueron contratados antes del año 2012.

```
SELECT  Vnombre, Apellido

FROM  VENDEDOR

WHERE FechaContrato < '01-ENE-2012'
```

Cuando la columna de comparación es una clave primaria, la instrucción genera una sola fila de resultado, como en este ejemplo. Hallar el nombre, apellido y número telefónico del vendedor con código 2

```
SELECT Vnombre, Apellido, Teléfono

FROM  VENDEDOR

WHERE Código = 2
```

Vnombre	Apellido	Teléfono
Antonio	Calle	2876549

Es posible comparar fácilmente los datos numéricos o de fecha y hora, pero hay que prestar atención al comparar cadenas de caracteres. Por ejemplo, puede ser que no se obtengan los resultados esperados al comparar dos cadenas aparentemente similares como "Juan" y "JUAN". Un factor determinante para todas las comparaciones de cadenas de caracteres es el orden de clasificación utilizado por el sistema de bases de datos.

Muchos sistemas de bases de datos utilizan el orden de clasificación ASCII, que coloca números antes de las letras y todas las letras mayúsculas antes de letras minúsculas. Si la base de datos soporta la secuencia de clasificación ASCII, los caracteres están en la siguiente secuencia de valores de menor a mayor valor:

. . .0123456789 . . . ABC. . . XYZ . . . abc . . . xyz . . .

Otros sistemas, sin embargo, ofrecen en ciertos casos opciones insensibles. Por ejemplo, una *a* minúscula se considera igual a una *A* mayúscula. Los caracteres están en la siguiente secuencia desde el valor más bajo al valor más alto:

. . . 1234456789 . . . {Aa} {Bb} {Cc} . . . {Xx}{Yy}{Zz}. . .

Nótese que los caracteres encerrados entre llaves {} se consideran iguales, porque no se hace distinción entre mayúsculas y minúsculas.

Otros gestores de bases de datos que se ejecutan en sistemas mainframe de IBM utilizan la secuencia EBCDIC registrada por IBM. En este caso el orden de clasificación coloca todas las letras minúsculas al principio, luego todas las letras en mayúsculas y, finalmente los números. Si la base de datos soporta EBCDIC, los caracteres están en la siguiente secuencia desde el valor más bajo al valor más alto:

. . . abc . . . xyz . . . ABC . . . XYZ . . . 0123456789 . . .

Debido a que los proveedores de software han implementado SQL en máquinas con diferentes arquitecturas y para muchos otros idiomas aparte del inglés, el estándar SQL no define ninguna secuencia de clasificación por omisión, ésta dependerá del software de bases de datos que se esté utilizando. (Viescas J., Hernández M., 2014)

En conclusión, cuando se trata de realizar operaciones de comparación con cadenas de caracteres, se recomienda revisar la documentación del sistema de base de datos para determinar cómo se comparan las letras mayúsculas, minúsculas y números.

5.2.2.2 Rango (BETWEEN)

SQL incluye el operador de comparación **BETWEEN**, que comprueba si el valor de una expresión se encuentra dentro de un rango especificado de valores. El operador requiere de tres expresiones, la primera "*expresión de evaluación*" corresponde al valor a comprobar, la segunda el límite inferior "*expresión inferior*" y la tercera el límite superior "*expresión superior*", del rango a comprobar.

Expresión de evaluación **BETWEEN** *expresión inferior* **AND** *expresión superior*

En otras palabras, el predicado **BETWEEN** permite buscar un valor a partir de dos expresiones enmarcadas en la palabra clave **AND**.

`X BETWEEN Y AND Z` es equivalente a `(X >= Y)` AND `(X <= Z)`

En el siguiente ejemplo se muestra el uso de la comparación **BETWEEN**. Recuperar el nombre y el apellido de los clientes que tienen un límite de crédito entre 1.000 y 1.500 dólares.

```
SELECT Nombre, LímCrédito
FROM  CLIENTE
WHERE LímCrédito BETWEEN 1000 AND 1500
```

Esta instrucción es equivalente a:

```
SELECT Nombre, LímCrédito
FROM CLIENTE
WHERE LímCrédito >= 1000 AND LímCrédito <= 1500
```

Nombre	Apellido1	LímCrédito
Juan	Polo	1100
Elena	Tapia	1500
Marcia	Mora	1000
Jaime	Pérez	1200
Manuel	Castro	1200

La negación `NOT BETWEEN` comprueba los valores que se encuentran fuera del rango especificado de valores. Por ejemplo, si se requiere un listado con los nombres y apellidos de los clientes cuyos límites de crédito no se encuentran entre 1.000 y 1.500 dólares, la instrucción se puede escribir como:

```
SELECT Nombre, LímCrédito
FROM  CLIENTE
WHERE LímCrédito NOT BETWEEN 1000 AND 1500
```

Nombre	Apellido1	LímCrédito
Víctor	Castro	600
Manuel	Castro	600
Humberto	Pons	2000

5.2.2.3 Pertenencia a conjunto (IN)

El predicado de pertenencia a conjunto **IN** examina si un valor coincide con otro de una lista de valores determinados explícitamente por enumeración. Los valores también pueden ser determinados implícitamente a través de una subconsulta que se revisará posteriormente. La condición de búsqueda está definida por:

Expresión de evaluación **IN** *(lista de valores)*

`A IN (X, Y, Z)` y es equivalente a `(A = X) OR (A = Y) OR (A = Z)`

El siguiente ejemplo ilustra el uso del predicado **IN**. Enlistar el nombre y el apellido de todos los clientes representados por los vendedores con código 2, 3 o 4.

```
SELECT Nombre, Apellido1, CódRepre
FROM CLIENTE
WHERE CódRepre IN (2,3,4)
```

Nombre	Apellido1	CódRepre
Pablo	Brito	3
Jaime	Pérez	2
Humberto	Pons	3
Alberto	Torres	4
Manuel	Castro	2

Si la expresión de evaluación contiene un valor **NULL**, la condición de pertenencia a conjunto devuelve **NULL** y la fila no formará parte del resultado de la consulta.

La negación **NOT IN** comprueba los valores que no coinciden con los definidos en la lista de valores. Por ejemplo, listar el nombre y el apellido de todos los clientes que no son representados por los vendedores 2, 3 o 4.

```
SELECT Nombre, Apellido1, CódRepre
FROM CLIENTE
WHERE CódRepre NOT IN (2,3,4)
```

Nombre	Apellido1	CódRepre
Víctor	Castro	1
Juan	Polo	1
Marcia	Mora	1

La mayoría de SGBD y el estándar ANSI de SQL, no especifica el límite máximo de elementos que se incluyen en la lista de valores.

5.2.2.4 Coincidencia de patrón (LIKE)

El operador **LIKE** compara el valor de una columna si coincide con un patrón especificado. Los patrones se escriben utilizando uno o más caracteres "comodines" que se detallan continuación:

- Tanto por ciento (%). Coincide con cualquier subcadena de varios o ningún caracteres.

- Subrayado (_). El comodín "_" remplaza a cualquier carácter.

Es posible la combinación de los dos comodines "%" y "_", éstos pueden aparecer en cualquier lugar de la cadena y repetirse indeterminado número de veces. Se debe considerar que algunas bases de datos no diferencian entre mayúsculas y minúsculas, es decir, el carácter "x" no es diferente al carácter "X". En la Figura 5.6 se presentan ejemplos que ilustran el uso de los comodines.

Patrones	Ilustración de criterios	Muestra de valores
´Per%´	La cadena de caracteres puede tener cualquier longitud, pero inicia con "Per"	**Per**alta, **Per**domo, **Per**eira
´%ardo´	La cadena de caracteres puede tener cualquier longitud, pero termina con "ardo"	Bern**ardo**, Ric**ardo**, Leon**ardo**
´%nan%´	La cadena de caracteres puede tener cualquier longitud pero debe contener "nan"	Fer**nan**do, Her**nan**do, Ve**nan**cio
´re_´	La cadena tiene tres caracteres y empieza con "re"	**re**o, **re**y, **re**d
´_an´	La cadena tiene tres caracteres y termina con "an"	P**an**, d**an**, v**an**
´_ar_´	La cadena tiene cuatro caracteres y debe tener "a" como segundo y "r" como tercer carácter	C**ar**a, v**ar**a, c**ar**o
´_al%´	La cadena tiene cualquier longitud, pero el segundo y tercer carácter tiene las letras "a" y "l" respectivamente	M**al**ena, P**al**oma, B**al**tazar
´%os_´	La cadena tiene cualquier longitud, pero la antepenúltima y penúltima letra tienen que ser "o" y "s" respectivamente.	R**os**a, Ambr**os**i, J**os**é

Figura 5.6. Ejemplos de uso de patrones para el operador `LIKE`.

Por ejemplo, si se requiere un listado de todos los clientes cuyo primer apellido empiece con la letra "P"

```
SELECT Apellido1
FROM  CLIENTE
WHERE Apellido1 LIKE ´P%´
```

Apellido1
Polo
Pérez
Pons

El siguiente ejemplo muestra el uso del comodín "_". Se pide enlistar todos los tipos de perfiles que se tienen registrados en la tabla `ARTíCULO`. Esta consulta se puede escribir como:

```
SELECT Descripción
FROM  ARTÍCULO
WHERE Descripción LIKE ´Perfil _´
```

Descripción
Perfil T
Perfil L
Perfil G

SQL considera la posibilidad de buscar cadenas que no coincidan en lugar de cadenas que coincidan, utilizando el operador de comparación **NOT LIKE**.

SQL:1999 también ofrece la operación **SIMILAR TO**, que proporciona una comparación de patrones más potente que la operación **LIKE**. (Silberschatz , A. Korth, H. Sudarshan, S., 2014).

En ciertos casos es posible que un carácter comodín (% o _) forme parte de un patrón. El estándar SQL ANSI/ISO especifica mediante la palabra clave **ESCAPE** una forma de diferenciar si los caracteres son comodines o normales. Cuando el carácter de escape aparece en el patrón, indica que el carácter que lo sigue inmediatamente va a ser tratado como un carácter normal[1].

En el siguiente ejemplo se utiliza el signo (+) como carácter de escape:

```
LIKE ´A+%B%´ ESCAPE ´+´
```

El primer signo de porcentaje (%) del patrón corresponde a un carácter normal, debido a que está precedido del carácter escape (+) y el segundo carácter porcentaje (%) representa un comodín.

5.2.2.5 Valores nulos (IS NULL)

Si se recuerda que el valor **NULL** no representa un cero, ni una cadena de caracteres de uno o más espacios en blanco, un valor **NULL** representa un valor desconocido **UNKNOWN**. Si se realiza una consulta a una fila determinada, el resultado de una condición de búsqueda puede ser **TRUE**, **FALSE** o **NULL**. Un valor desconocido correspondería a este último resultado, cuando el valor de una fila de la columna evaluada, contenga un valor **NULL**. En ciertos casos es preferible evaluar una condición de búsqueda explícitamente con los valores **NULL** contenidos en una columna. Para cumplir con este propósito, SQL utiliza la palabra clave IS **NULL** para comprobar si un valor es **NULL** la instrucción devuelve siempre un resultado **TRUE** o **FALSE**. Por ejemplo, si se quiere saber el nombre de los clientes que no tienen representante, la consulta se puede escribe como:

[1] Un carácter normal suele recibir a veces el nombre de carácter literal (Groff J., Weinberg P., 2003)

```
SELECT Nombre, Apellido1
FROM CLIENTE
WHERE CódRepre IS NULL
```

Nombre	Apellido1	CódRepre
Elena	Tapia	Null

El uso de los valores `NULL` en las operaciones de comparación presenta un problema como el que se detalla a continuación. La comparación 5 = `NULL`, representa un error, porque no es posible comparar el valor de 5 con algo desconocido (`UNKNOWN`).

La forma negativa de la condición de búsqueda valor nulo es `IS NOT NULL`, que busca las filas que no contienen un valor `NULL`.

Algunos SGBD también tienen la posibilidad de comprobar si el resultado de una consulta es desconocido en lugar de `TRUE` o `FALSE`, mediante las cláusulas `IS UNKNOWN` o `IS NOT UNKNOWN`.

5.2.3 Uso de múltiples condiciones

Las consultas que se han realizado hasta el momento han requerido de una condición simple de búsqueda para obtener el resultado. A continuación se verá cómo se puede responder a solicitudes complejas utilizando la combinación de estas condiciones simples de búsqueda. SQL permite formar predicados complejos utilizando los operadores lógicos `AND`, `OR` y `NOT`.

5.2.3.1 Uso del operador `AND`

Una primera forma de combinar dos o más condiciones se realiza mediante el operador lógico `AND`. Este operador se utiliza cuando todas las condiciones de búsqueda que se combinan, cumplan simultáneamente para que una fila se incluya en el resultado. Por ejemplo, si se requiere el nombre y el apellido de los clientes que tienen como representante al vendedor de código 1 y tienen un límite de crédito mayor a igual a 1.000 dólares. La instrucción en SQL puede escribirse como:

```
SELECT Nombre, Apellido1
FROM CLIENTE
WHERE CódRepre = 1 AND LímCrédito >= 1000
```

Nombre	Apellido1
Juan	Polo
Marcia	Mora

En la Figura 5.7 se muestra la tabla de verdad para el operador lógico AND. Es importante recordar que todas las condiciones de búsqueda deben evaluar verdadero para que una fila aparezca en el resultado.

		Segunda expresión	
Primera expresión	AND	True	False
	True	True Fila seleccionada	False Fila rechazada
	False	False Fila rechazada	False Fila rechazada

Figura 5.7. Resultado de la combinación de dos expresiones con el operador AND.

5.2.3.2 Uso del operador OR

La segunda forma de combinar dos o más condiciones de búsqueda es con el uso del operador lógico OR. Este operador incluye una fila en el resultado, cuando una cualquiera o las dos condiciones de búsqueda sean ciertas. Por ejemplo: enlistar el nombre y apellido de los clientes que tienen como representante al vendedor con código 3 o tienen un crédito mayor a 1.200 dólares.

```
SELECT Nombre, Apellido1
FROM CLIENTE
WHERE CódRepre = 3 OR LímCrédito > 1200
```

Nombre	Apellido1
Elena	Tapia
Pablo	Brito
Humberto	Pons

Para esta consulta el cliente Humberto Pons cumple las dos condiciones de búsqueda, en tanto que, Elena Tapia cumple únicamente la condición de límite de crédito y el cliente Pablo Brito sólo la de su representante.

En la Figura 5.8 se muestra la tabla de verdad del operador OR. Para que una fila aparezca en el resultado, al menos una condición de búsqueda tiene que ser verdadera.

		Segunda expresión	
Primera expresión	AND	True	False
	True	True Fila seleccionada	True Fila seleccionada
	False	True Fila seleccionada	False Fila rechazada

Figura 5.8. Resultado de la combinación de dos expresiones con el operador OR.

5.2.3.3 Uso del operador NOT

En los apartados anteriores se revisó que el operador NOT es una opción que se utiliza con los predicados BETWEEN, IN, LIKE y IS NULL; sin embargo, el NOT se utiliza también como una palabra clave de una condición de búsqueda y permite seleccionar la fila donde la condición de búsqueda es falsa. La Figura 5.9 muestra la tabla de verdad del operador NOT.

Expresión	NOT (Expresión)
True	False
False	True

Figura 5.9. Tabla de verdad de NOT.

Por ejemplo, hallar los vendedores de sexo masculino, cuyos objetivos de ventas no estén por debajo de 3.000 dólares.

```
SELECT Vnombre, Apellido

FROM VENDEDOR

WHERE Sexo = ´M´ AND NOT ObjVentas < 3000
```

Vnombre	Apellido	Sexo	ObjVentas
Diego	Loja	M	3000
Arturo	Salto	M	4000

5.2.3.4 Uso de AND, OR y NOT juntos

Para agrupar los criterios de búsqueda y generar consultas complejas pueden utilizarse los operadores lógicos AND, OR y NOT. Es recomendable en este tipo de consultas utilizar siempre paréntesis para mostrar el orden de evaluación y eliminar cualquier posible ambigüedad. A continuación se muestran las reglas de precedencia para evaluar estas expresiones:

- Por defecto la expresión se evalúa de izquierda a derecha, que resulta verdadera para el caso de condiciones simples.
- Se evalúan primero las expresiones agrupadas en corchetes.
- Se evalúa el operador NOT antes que los operadores AND y OR.
- Se evalúa el operador AND antes que el operador OR.

En las Figuras 5.7, 5.8 y 5.9 se especifican las tablas de verdad para AND, OR y NOT respectivamente; sin embargo, en éstas no se considera el impacto que representa la presencia de valores nulos. Como se revisó en las secciones anteriores, el estándar SQL define el resultado de cualquier condición que evalúa un valor NULL como desconocido, además, una condición debe ser cierta para que una fila sea seleccionada, por lo que un resultado falso o desconocido rechaza la fila.

En las Figuras 5.10(a), (b) y (c) se muestra la tabla de verdad en la que se incluye un resultado desconocido NULL.

Segunda expresión

Primera expresión: AND	True	False	Null
True	True Fila seleccionada	False Fila rechazada	Null Fila rechazada
False	False Fila rechazada	False Fila rechazada	False Fila rechazada
Null	Null Fila rechazada	False Fila rechazada	Null Fila rechazada

(a)

Segunda expresión

Primera expresión: OR	True	False	Null
True	True Fila seleccionada	True Fila seleccionada	True Fila seleccionada
False	True Fila seleccionada	False Fila rechazada	Null Fila rechazada
Null	True Fila seleccionada	Null Fila rechazada	Null Fila rechazada

(b)

Expresión	NOT (Expresión)
True	False
False	True
Null	Null

(c)

Figura 5.10. Tablas de verdad:

a) Para el operador AND que incluye el resultado de una expresión desconocida.

b) Para el operador OR que incluye el resultado de una expresión desconocida.

c) Resultado de aplicar NOT a un valor verdadero/falso/ desconocido.

5.2.4 Orden en la presentación de resultados (ORDER BY)

La palabra clave **ORDER BY** seguida de una lista de especificaciones de orden separadas por comas, hace que las filas del resultado de una consulta se presenten en un cierto orden. La cláusula **ORDER BY** ordena de forma ascendente o descendente con las palabras claves **ASC** y **DESC** respectivamente, no obstante, de manera predeterminada las filas se ordenan de forma ascendente, pudiendo omitirse la palabra clave **ASC**. Siempre la cláusula **ORDER BY** debe ser la última cláusula de una instrucción **SELECT**. Por ejemplo, la siguiente consulta ordena los resultados sobre la base de la columna *Apellido1* y dentro de cada apellido ordenar por la columna *Nombre*.

```
SELECT Nombre, Apellido1
FROM  CLIENTE
ORDER BY Apellido1, Nombre
```

Nombre	Apellido1
Pablo	Brito
Manuel	Castro
Víctor	Castro
Marcia	Mora
Jaime	Pérez
Juan	Polo
Humberto	Pons
Elena	Tapia
Alberto	Torres

En este caso se incluyen dos columnas de ordenamiento. La clave de ordenamiento principal es la columna *Apellido1*; si los valores de esta columna son unívocos, no es necesario incluir una clave adicional; por el contrario, si no son unívocas (en nuestro caso el apellido "Castro" se repite dos veces) se incluye una clave de ordenamiento secundaria, que servirá para ordenar las filas que contienen el mismo valor de la clave de ordenamiento principal. En nuestro ejemplo se adiciona la columna *Nombre*.

La siguiente instrucción incluye la palabra clave **DESC** para ordenar las filas de forma descendente de acuerdo al contenido de la columna *ObjVentas*.

```
SELECT Vnombre, ObjVentas
FROM  VENDEDOR
ORDER BY ObjVentas DESC
```

Vnombre	ObjVentas
Arturo	4000
Maricela	3500
Diego	3000
Antonio	2500
Lucía	2000

Una columna de ordenamiento también se puede especificar mediante un número, que hace referencia a la ubicación de la columna en la lista de la cláusula `SELECT`; esta posibilidad se utiliza básicamente para ordenar columnas calculadas, que por lo general no tienen nombre. Por ejemplo, para ordenar los artículos en forma descendente de acuerdo a la diferencia entre la existencia máxima y mínima, la instrucción puede escribirse como:

```
SELECT Descripción, (ExiMáx – ExiMín)
FROM  ARTÍCULO
ORDER BY 2 DESC
```

Descripción	(ExiMáx - ExiMín)
Clavos	65
Cemento	40
Lija	20
Alambre	15
Pintura	11
Perfil L	10
Perfil T	7
Perfil G	7
Suelda	5

Para el caso de los valores nulos contenido en una columna, el estándar SQL especifica que la cláusula `ORDER BY` debe tratar estos valores como si fueran menores que cualquier valor no nulo o mayores que cualquier valor no nulo. Sera el SGBD que implemente cualquiera de las dos opciones.

5.2.5 Operaciones sobre conjuntos

En SQL, las operaciones de `UNION`, `INTERSECT` y `EXCEPT` corresponden a las operaciones matemáticas de la teoría de conjuntos unión (∪), intersección (∩) y diferencia (-) respectivamente, combinan los resultados de dos o más consultas en una única tabla de resultados.

Existen restricciones que deben considerarse al momento de ejecutar estas operaciones:

- Ambas tablas deben tener el mismo número de columnas.
- El tipo de dato y la longitud de cada columna en la primera tabla deben ser los mismos que su correspondiente columna en la segunda tabla.

- En las dos instrucciones `SELECT` de las operaciones sobre conjuntos, no puede aparecer la cláusula `ORDER BY`; no tiene sentido ordenar datos que en ningún momento serán visibles para el usuario. Si se requiere ordenar el resultado producido por las operaciones sobre conjuntos, se especifica la cláusula `ORDER BY` después de la segunda instrucción `SELECT`.

Para ejemplificar estas tres operaciones se consideran los conjuntos formados por las siguientes consultas:

En la primera se calcula el conjunto de todos los clientes que tienen como representante al vendedor de código 1.

```
SELECT Nombre, Apellido1
FROM   CLIENTE
WHERE CódRepre = 1
```

Nombre	Apellido1
Víctor	Castro
Juan	Polo
Marcia	Mora

En la segunda se determina el conjunto de todos los clientes que tienen un límite de crédito mayor a 1.000 dólares.

```
SELECT Nombre, Apellido1
FROM   CLIENTE
WHERE LímCrédito > 1000
```

Nombre	Apellido1
Juan	Polo
Elena	Tapia
Jaime	Pérez
Humberto	Pons
Alberto	Torres

5.2.5.1 La operación `UNION`

La operación `UNION` de dos tablas `R` y `S` define una tabla que incluye todas las filas que están en `R` o en `S` o en ambas. Por ejemplo, para determinar todos los clientes que tienen como representante al vendedor de código 1 o un límite de crédito mayor a 1.000 dólares. La instrucción se escribe como sigue:

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE CódRepre = 1
UNION
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE LímCrédito > 1000
```

El proceso de esta consulta con la operación `UNION` se muestra en la Figura 5.11.

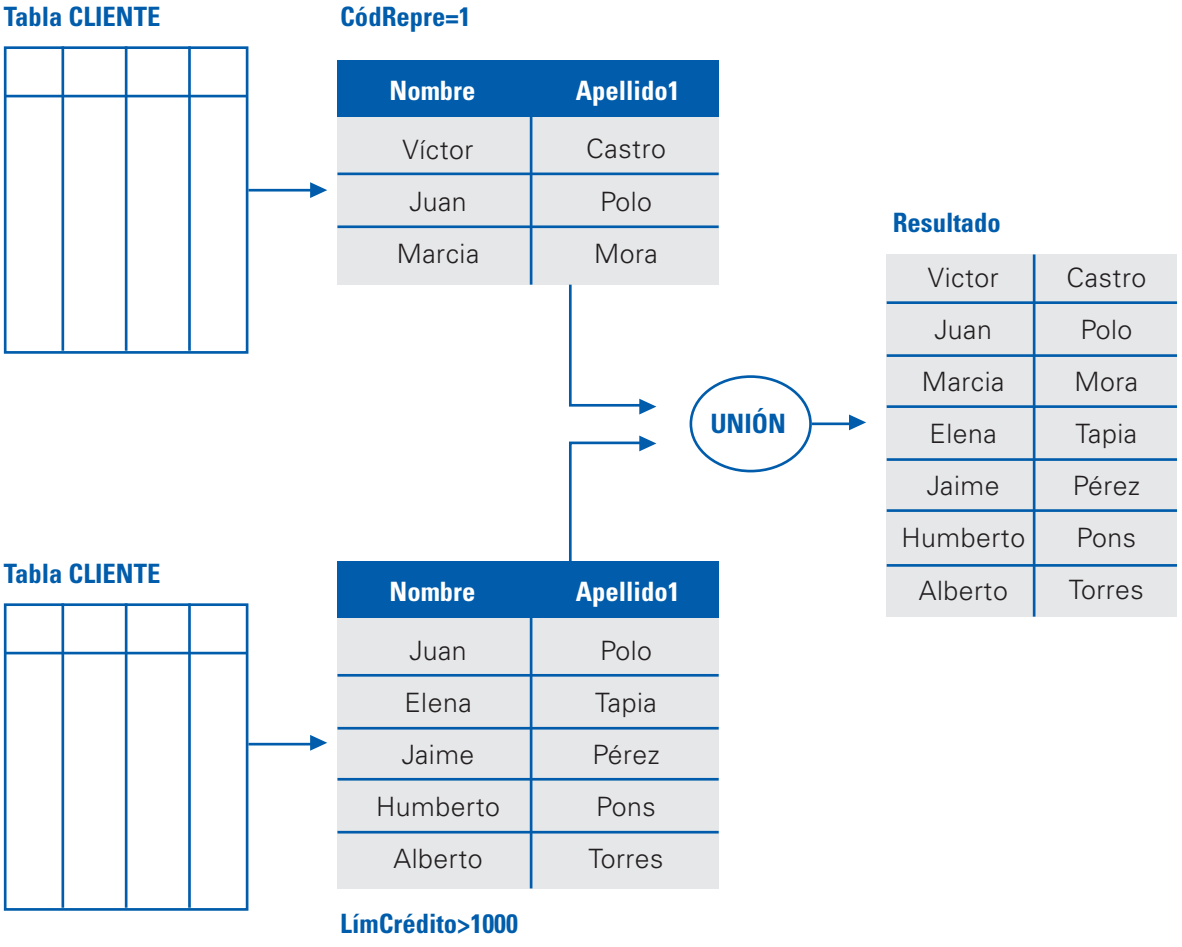


Figura 5.11. Combinar resultados con la operación `UNION`.

La operación UNION elimina las filas duplicadas. Si se requiere conservar las filas duplicadas, se remplaza UNION por UNION ALL.

Las columnas que forman parte del resultado de las operaciones sobre conjuntos no tiene nombre; esto implica que, para ordenar el resultado, la cláusula ORDER BY que se escribe después del segundo SELECT, debe especificar la columna de ordenamiento por su número. Por ejemplo, si se requiere al resultado de la consulta anterior ordenarle por el apellido, la instrucción se escribe como:

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE CódRepre = 1
UNION
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE LímCrédito > 1000
ORDER BY 2
```

5.2.5.2 La operación INTERSECT

La operación INTERSECT de dos tablas R y S, define una tabla que incluye todas las filas que están en R y en S. Por ejemplo, para encontrar todos los clientes que tienen como representante al vendedor de código 1, así como un límite de crédito mayor a 1.000 dólares, se escribe:

```
SELECT      Nombre, Apellido1
FROM        CLIENTE
WHERE       CódRepre = 1
INTERSECT
SELECT      Nombre, Apellido1
FROM        CLIENTE
WHERE       LímCrédito > 1000
```

Juan	Polo

La tabla resultante sólo contiene una fila con el nombre del cliente "Juan Polo", que cumple las dos condiciones: tiene como representante al vendedor de código 1 y también su límite de crédito es mayor a 1.000 dólares. La operación de intersección elimina automáticamente los valores duplicados. Si se necesita mantener las filas duplicadas, se remplaza `INTERSECT` por `INTERSECT ALL`.

5.2.5.3 La operación `EXCEPT`

La operación `EXCEPT` (deferencia) de dos tablas `R` y `S`, define una relación que incluye las filas que están en `R`, pero no en `S`. Por ejemplo, para hallar todos los clientes que tienen como representante al vendedor de código 1, pero no forman parte de los clientes que tienen un límite de crédito mayor a 1.000 dólares, se escribe:

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE  CódRepre = 1
EXCEPT
SELECT Nombre, Apellido1
FROM CLIENTE
WHERE LímCrédito > 1000
```

Victor	Castro
Marcia	Mora

El resultado de esta consulta tiene dos filas que corresponden a todas las filas de la primera consulta; pero no existen en la segunda. De igual manera que las dos operaciones anteriores, si se quiere mantener las filas duplicadas se escribe `EXCEPT` por `EXCEPT ALL`.

Las operaciones matemáticas de la teoría de conjuntos unión e intersección son conmutativas; no obstante, la diferencia no lo es. En consecuencia, si se intercambia el orden de las dos instrucciones `SELECT`, el resultado es diferente, según se observa a continuación:

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE LímCrédito > 1000
EXCEPT
SELECT Nombre, Apellido1
FROM CLIENTE
WHERE CódRepre = 1
```

Elena	Tapia
Jaime	Pérez
Humberto	Pons
Alberto	Torres

Algunas implementaciones de SGBD, específicamente Oracle, utilizan la palabra clave `MINUS` en lugar de `EXCEPT`.

5.2.6 Consultas multitabla

Los ejemplos de consultas estudiados hasta ahora han sido realizados basándose en una tabla; sin embargo, es usual que las consultas accedan a varias tablas en la misma instrucción. Por ejemplo, para generar un listado de todos los vendedores que tienen cargas familiares, en el que se muestre el nombre, el apellido del vendedor; el nombre y la relación de sus cargas familiares.

Los datos necesarios para esta consulta se encuentran almacenados en las tablas `VENDEDOR` y `CARGAFAMILIAR`, de ahí que, a continuación se revisarán sus características:

- De las columnas requeridas en la consulta, la tabla `VENDEDOR` contiene el nombre y apellido del vendedor, pero no la información referente a sus cargas familiares.
- La tabla `CARGAFAMILIAR` almacena el nombre y la relación de la carga familiar, pero no contiene el nombre y apellido del vendedor.
- La tabla `VENDEDOR` tiene como columna el *CódigoVend*, que se corresponde con el valor de la columna *CódVendedor* en la tabla `CARGAFAMILIAR`. Estas dos columnas servirán como enlace entre las dos tablas para que la instrucción `SELECT` genere el resultado.

Bajo estos lineamientos la consulta podría escribirse de la siguiente forma:

```
SELECT Vnombre, Apellido, NomDep, Relación
FROM VENDEDOR, CARGAFAMILIAR
WHERE CódigoVend = CódVendedor
```

Con el objeto de ejemplificar el proceso de la consulta en SQL, a continuación se detallan los pasos que podrían considerarse para generar el resultado:

1. La cláusula **FROM** forma un producto cartesiano de las dos tablas, como se muestra en la Figura 5.12. (Página 209)

2. El predicado de la cláusula **WHERE** se aplica al resultado del paso 1, como se muestra en la Figura 5.13.

Vnombre	Apellido	CódigoVend	...	NomDep	...	Relación	CódVendedor
Diego	Loja	1	...	María	...	Hija	1
Diego	Loja	1	...	Isabel	...	Hija	1
Antonio	Calle	2	...	Antonio	...	Hijo	2
Antonio	Calle	2	...	Maricela	...	Cónyuge	2
Lucía	Serrano	3	...	Teodoro	...	Cónyuge	3

Figura 5.13. Resultado de la cláusula **WHERE**.

3. Para cada fila que se obtiene en el resultado del paso 2 se extraen las columnas especificadas en la cláusula **SELECT**, según se detalla en la Figura 5.14. Se observa que los vendedores Arturo Salto y Marcia Vera, al no tener cargas familiares, no aparecen en el resultado.

Vnombre	Apellido	NomDep	Relación
Diego	Loja	María	Hija
Diego	Loja	Isabel	Hija
Antonio	Calle	Antonio	Hijo
Antonio	Calle	Maricela	Cónyuge
Lucía	Serrano	Teodoro	Cónyuge

Figura 5.14. Resultado de la cláusula **SELECT** y de toda la consulta.

Vnombre	Apellido	CódigoVend	...	NomDep	...	Relación	CódVendedor
Diego	Loja	1	...	María	...	Hija	1
Antonio	Calle	2	...	María	...	Hija	1
Lucía	Serrano	3	...	María	...	Hija	1
Arturo	Salto	4	...	María	...	Hija	1
Maricela	Vera	5	...	María	...	Hija	1
Diego	Loja	1	...	Isabel	...	Hija	1
Antonio	Calle	2	...	Isabel	...	Hija	1
Lucía	Serrano	3	...	Isabel	...	Hija	1
Arturo	Salto	4	...	Isabel	...	Hija	1
Maricela	Vera	5	...	Isabel	...	Hija	1
Diego	Loja	1	...	Antonio	...	Hijo	2
Antonio	Calle	2	...	Antonio	...	Hijo	2
Lucía	Serrano	3	...	Antonio	...	Hijo	2
Arturo	Salto	4	...	Antonio	...	Hijo	2
Maricela	Vera	5	...	Antonio	...	Hijo	2
Diego	Loja	1	...	Maricela	...	Cónyuge	2
Antonio	Calle	2	...	Maricela	...	Cónyuge	2
Lucía	Serrano	3	...	Maricela	...	Cónyuge	2
Arturo	Salto	4	...	Maricela	...	Cónyuge	2
Maricela	Vera	5	...	Maricela	...	Cónyuge	2
Diego	Loja	1	...	Teodoro	...	Cónyuge	3
Antonio	Calle	2	...	Teodoro	...	Cónyuge	3
Lucía	Serrano	3	...	Teodoro	...	Cónyuge	3
Arturo	Salto	4	...	Teodoro	...	Cónyuge	3
Maricela	Vera	5	...	Teodoro	...	Cónyuge	3

Figura 5.12 Resultado de la cláusula FROM.

En realidad los SGBD no ejecutan las consultas de esta forma, lo que hacen es optimizar la evaluación, restringiendo la combinación que crea el producto cartesiano, generando únicamente filas que satisfagan los predicados de la cláusula **WHERE**.

La consulta que se termina de analizar recibe también el nombre de consultas padre/hijo (Groff J., Weinberg P., 2003), en la que se especifica una condición de búsqueda entre las tablas que compara la clave primaria y la clave foránea, siendo la tabla padre la que contiene la clave primaria y la tabla hijo la que contiene la clave foránea.

Si una tabla T1 (padre) contiene una clave primaria compuesta p1 y p2, la tabla T2 (hijo) contendrá sus respectivas claves foráneas f1 y f2. Para reunir las dos tablas en términos de una consulta padre/hijo se debe especificar ambas parejas de columnas coincidentes, como se muestra a continuación:

```
SELECT A1, A1 . . . An
FROM   T1, T2
WHERE  p1 = f1 AND
       P2 = f2
```

SQL no restringe el número de columnas para una clave primaria compuesta que forme parte de una reunión multicolumna entre dos tablas, las condiciones del número de columnas vienen definidas por los requerimientos del mundo real. Si bien es cierto estas son menos comunes que las reuniones de una única columna.

En ciertas consultas es posible restringir aún más los contenidos del resultado, adicionando uno o más predicados a la cláusula **WHERE**. Por ejemplo, para enlistar el nombre y la relación de las cargas familiares únicamente del vendedor Diego Loja, la consulta se podría escribir de la siguiente forma:

```
SELECT NomDep, Relación
FROM  VENDEDOR, CARGAFAMILIAR
WHERE  Vnombre = ´Diego´ AND
       Apellido = ´Loja´
CódigoVend = CódVendedor;
```

NomDep	Relación
María	Hija
Isabel	Hija

Con el objeto de interpretar el proceso de la consulta se considera que el SGBD ejecuta los predicados de la cláusula `WHERE` en el orden descrito en la instrucción: con las dos primeras condiciones `Vnombre = ´Diego´ AND Apellido = ´Loja´,` se selecciona solo una fila como resultado, y con la tercera se compara el valor de la única clave primaria *CódigoVend* con los valores de la clave foránea *CódVendedor* de la tabla `CARGAFAMILIAR`.

Si a la consulta anterior se le modifica para expresarla en los siguientes términos: para todos los vendedores que tienen un límite de crédito menor o igual a 2.000 dólares, enlistar el nombre y apellido del vendedor, el nombre y la relación de sus cargas familiares. La instrucción se podría escribir de la siguiente forma:

```
SELECT Vnombre, Apellido, NomDep, Relación
FROM  VENDEDOR, CARGAFAMILIAR
WHERE  CódigoVend = CódVendedor AND
       ObjVentas < = 2500;
```

Vnombre	Apellido	NomDep	Relación
Antonio	Calle	Antonio	Hijo
Antonio	Calle	Maricela	Cónyuge
Lucía	Serrano	Teodoro	Cónyuge

Esta consulta se diferencia de la anterior en el número de filas (5) que genera como resultado, la condición de búsqueda `CódigoVend` = `CódVendedor`. La segunda condición restringe el resultado eliminando las filas que no cumplen con la condición `ObjVentas < = 2500.`

En los ejercicios anteriores, con el objeto de mostrar el proceso de una consulta, arbitrariamente se ha definido un orden para la ejecución de cada una de las cláusulas de la instrucción `SELECT`; no obstante, el orden de *cómo* generar la consulta es determinado por el SGBD.

La mayoría de las consultas entre múltiples tablas se realizan en relaciones de padre/hijo; sin embargo SQL no exige que las columnas de coincidencia sean necesariamente claves primarias y claves foráneas. Cualquier par de columnas puede servir como columnas de coincidencias, siempre que los tipos de datos sean compatibles.

Usando la misma técnica de las consultas de dos tablas, SQL puede combinar tres o más tablas. Por ejemplo, para obtener un listado con la descripción de los artículos que fueron vendidos en las diferentes facturas al cliente con cédula 0122334455, la instrucción se escribe de la siguiente manera:

```
SELECT Descripción
FROM  ARTÍCULO, DETALLE, FACTURA
WHERE  CcCliente = 0122334455 AND
       Número = FactNúmero AND
       ArtCódigo = Código
```

La Figura 5.15 (Página 213) muestra el proceso para generar el resultado de esta consulta. Primero se restringe el número de filas de la tabla **FACTURA** mediante la condición *CcCliente* = **0122334455**, luego se relaciona la clave primaria *Número* de la tabla **FACTURA** con la clave foránea *FactNúmero* de la tabla **DETALLE**. La última condición hace coincidir la clave foránea *ArtCódigo* de la tabla **DETALLE** con la clave primaria *Código* de la tabla **ARTÍCULO**. La instrucción termina recuperando los datos de la columna *Descripción* especificada en el **SELECT**.

A continuación se presenta una consulta que incluye 4 tablas. A partir del ejercicio anterior en el que se modificó la condición de búsqueda relacionada con el cliente, ahora se busca al cliente no por su cédula, sino por su nombre y apellido. Por ejemplo, enlistar la descripción de los artículos que fueron vendidos en las diferentes facturas al cliente Juan Polo. La consulta se puede escribir como:

```
SELECT Descripción
FROM  ARTÍCULO, DETALLE, FACTURA, CLIENTE
WHERE  Nombre = ´Juan´ AND Apellido = ´Polo´ AND
       Cédula = CcCliente AND
       Número = FactNúmero AND
       ArtCódigo = Código
```

La instrucción incluye una tabla adicional **CLIENTE**, en donde se registra el nombre y apellido "Juan Polo", que servirá para determinar la clave primaria *Cédula* para luego relacionarla con la clave foránea *CcCliente* de la tabla **FACTURA**.

FACTURA

Número	Fecha	CódVendedor	CcCliente
100	6 - Ene - 14	1	0102030405
101	13 - Ene - 14	3	0122334455
102	28 - Feb - 14	1	0123456789
103	5 - Feb - 14	4	0122334455
104	19 - Feb - 14	2	0987654321
.			
.			
.			

DETALLE

FactNúmero	ArtCódigo	Cantidad
100	2	5
100	9	1
100	10	5
101	6	4
101	8	5
102	5	6
103	11	1
103	9	4
103	1	4
.		
.		
.		

ARTÍCULO

Código	Descripción	Precio	. . .
1	Cemento	6,0	. . .
2	Clavos	4,8	. . .
5	Alambre	3,0	. . .
6	Perfil T	15,0	. . .
7	Perfil L	15,5	. . .
8	Perfil G	16,0	. . .
9	Pintura	19,5	. . .
10	Lija	0,6	. . .
11	Suelda	20,0	. . .

RESULTADO

Descripción
Cemento
Perfil G
Pintura
Suelda

Figura 5.15. Consulta de tres tablas.

5.2.7 Operación de renombrado

En este apartado se revisa el mecanismo que proporciona SQL para el renombrado de columnas y tablas. Por ejemplo, si se analiza la siguiente instrucción:

```
SELECT Vnombre, Apellido, NomDep
FROM  VENDEDOR, CARGAFAMILIAR
WHERE  CódigoVend = CódVendedor
```

Los nombres de las columnas, en el resultado se obtienen de las tablas `VENDEDOR` y `CARGAFAMILIAR` especificadas en la cláusula `FROM` no obstante, existen casos en los que no es posible referirse a ellos de una manera directa, por las siguientes razones:

- Cuando existen tablas en la cláusula `FROM`; que contengan columnas con el mismo nombre y generen ambigüedad en una consulta.

  Para evitar este problema se utiliza el concepto de nombres de columna cualificado (Groff J., Weinberg P., 2003), esto significa que, se debe **calificar** al nombre de la columna con el nombre de la tabla, con el propósito de eliminar las ambigüedades. En SQL para calificar una columna, se antepone al nombre de la columna, el nombre de la tabla separados por un punto (.). Por ejemplo, para la columna Sexo de la tabla `VENDEDOR`, su nombre cualificado es:

  ```
  VENDEDOR.SEXO
  ```

  En una sentencia SQL los nombres de columna cualificados se pueden utilizar en cualquier lugar en donde pueda aparecer un nombre de columna.

  Para ilustrar estos conceptos, si en el ejemplo de la consulta anterior, se requiere que en el reporte se incluya el sexo del vendedor y el sexo de la carga familiar. La sentencia se escribe de la siguiente forma:

```
SELECT Vnombre, Apellido, VENDEDOR.Sexo, NomDep,
       CARGAFAMILIAR.Sexo
FROM  VENDEDOR, CARGAFAMILIAR
WHERE CódigoVend = CódVendedor
```

Vnombre	Apellido	VENDEDOR.Sexo	NomDep	CARGAFAMILIAR.Sexo
Diego	Loja	M	María	F
Diego	Loja	M	Isabel	F
Antonio	Calle	M	Antonio	M
Antonio	Calle	M	Maricela	F
Lucía	Serrano	F	Teodoro	M

Es un error si en la cláusula `SELECT` de la anterior sentencia se colocan los nombres de las columnas como: `Vnombre, Apellido, Sexo, NomDep, Sexo`, porque *Sexo* es un nombre ambiguo de la columna y el SGBD no podrá diferenciar si corresponde a la tabla `VENDEDOR` o `CARGAFAMILIAR`. Para evitar este problema se utiliza el nombre de columnas cualificadas.

- SQL proporciona la cláusula `AS`, que permite cambiar el nombre de las columnas que se tienen registradas en la base de datos. Su formato es:

```
nombre_anterior AS nombre_actual
```

Por ejemplo, para sustituir los nombres de las columnas: *Vnombre* por *Nombre_vendedor*, *Apellido* por *Apellido_vendedor* y *NomDep* por *Nombre_carga*, la instrucción se escribe como:

```
SELECT Vnombre AS Nombre_vendedor, Apellido AS Apellido_vendedor,
       VENDEDOR.Sexo, NomDep AS Nombre_carga, CARGAFAMILIAR.Sexo
FROM  VENDEDOR, CARGAFAMILIAR
WHERE  CódigoVend = CódVendedor
```

También es posible mediante la cláusula **AS** asignar un nombre a una columna que se presenta en la cláusula **SELECT** como una expresión aritmética. Por ejemplo, a los clientes que tienen un límite de crédito mayor a 1.200 dólares, incrementar su límite en un 10 por ciento.

```
SELECT Nombre, Apellido1, (límCrédito + LímCrédito * 0.1)
       AS Crédito_Incr
FROM  CLIENTE
WHERE  LímCrédito > 1200
```

Nombre	Apellido1	Crédito_Incr
Elena	Tapia	1650
Humberto	Pons	2200

- Otra posibilidad que SQL proporciona con la cláusula **AS** es el renombramiento de tablas. Una razón para renombrar una tabla es sustituir un nombre largo por uno corto que facilita la escritura de la sentencia. Por ejemplo, enlistar el nombre y apellido de los vendedores que tienen al menos una carga familiar con el mismo sexo. Esta consulta podría escribirse de dos maneras:

La primera sin el uso de la opción de renombrado de tablas:

```
SELECT Nombre, Apellido
FROM  VENDEDOR, CARGAFAMILIAR
WHERE  CódigoVend = CódVendedor AND
       VENDEDOR.Sexo = CARGAFAMILIAR.Sexo
```

En la segunda alternativa que se escribe a continuación, se incluye la cláusula **AS** para renombrar las tablas especificadas en la cláusula **FROM.**

```
SELECT Nombre, Apellido
FROM  VENDEDOR AS V, CARGAFAMILIAR AS CF
WHERE  CódigoVend = CódVendedor AND
       V.Sexo = CF.Sexo
```

Vnombre	Apellido
Antonio	Calle

Otra razón para renombrar una tabla se presenta cuando una consulta compara columnas de una misma tabla. Por ejemplo, para cada cliente, recuperar el nombre y primer apellido; y el nombre y primer apellido de quien los recomendó.

```
SELECT C.Nombre, C.Apellido1, R.Nombre, R.Apellido1
FROM  CLIENTE AS C, CLIENTE AS R
WHERE  C.CédulaReco = R.Cédula
```

C.Nombre	C.Apellido1	R.Nombre	R.Apellido1
Víctor	Castro	Juan	Polo
Jaime	Pérez	Juan	Polo
Manuel	Bonilla	Juan	Polo
Alberto	Torres	Juan	Polo
Elena	Tapia	Jaime	Pérez
Marcia	Mora	Jaime	Pérez
Juan	Polo	Humberto	Pons
Pablo	Brito	Humberto	Pons

La cláusula `AS` permite crear dos copias idénticas `C` y `R` de la tabla `CLIENTE`, que en la norma SQL se denominan **nombre de correlación**; pero regularmente se llama alias. La primera copia C representa a los clientes en el rol de "recomendados" y la segunda copia `R` representa a los clientes con el rol de "recomendar". En realidad sólo hay una tabla `CLIENTE`, los nombres `C` y `R` son nombres alternativos para la tabla `CLIENTES`. Los alias ayudan a generar copias para poder concatenar la tabla consigo misma, haciendo coincidir las filas que cumplan la condición de búsqueda *C.CédulaReco = R.Cédula*. Este tipo de consultas reciben el nombre de recursivas.

Un alias se declara escribiendo el nombre de la tabla alternativa a continuación de la cláusula `AS`. Por ejemplo, `CLIENTE AS C`, como en la sentencia anterior. Es posible también declarar un alias, especificando el nombre de la tabla alterna inmediatamente después del nombre real de la tabla (por ejemplo, si se escribe `CLIENTE C`). El uso o no de la cláusula `AS` dependerá del SGBD.

SQL permite también renombrar las columnas dentro de la consulta en la cláusula `FROM`. Por ejemplo, si se requiere renombrar todos los nombres de las columnas de la tabla `CLIENTE`, a continuación de la cláusula `FROM` se escribe:

```
CLIENTE AS C(N, A1, A2, CC, Dir, Crédito, CR, Fecha)
```

En este caso, N será el alias de *Nombre*, A1 el alias de *Apellido1*, A2 de *Apellido2*, etcétera.

5.2.8 Tablas concatenadas o combinadas

5.2.8.1 Concatenación interna

El concepto de tabla concatenada en SQL fue creado con el objeto de especificar una tabla como resultado de una operación de concatenación que se declare en la cláusula `FROM`. Por ejemplo, si se analiza la siguiente consulta:

```
SELECT      *
FROM ARTÍCULO JOIN DETALLE ON ArtCódigo = Código
WHERE FactNúmero = 103
```

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín	FactNúmero	ArtCódigo	Cantidad
11	Suelda	20,0	Kilo	10	5	103	11	1
9	Pintura	19,5	Galón	15	4	103	9	4
1	Cemento	6,0	Saco	50	10	103	1	4

En este ejemplo la cláusula `FROM` contiene una tabla concatenada, formada por todas las columnas de la tabla `ARTÍCULO`, seguidas por todas las columnas de la tabla `DETALLE`. La palabra reservada `ON` permite la condición de concatenación entre las dos tablas, especificando que las filas de la tabla `ARTÍCULO` correspondan con las filas de la tabla `DETALLE`, si los valores de las columnas *Código* y *ArtCódigo* son iguales.

Este tipo de operaciones recibe el nombre de **concatenación interna**, y la cláusula se especifica como `INNER JOIN`; sin embargo, la palabra clave `INNER` es opcional.

La condición ON opera de igual manera que la cláusula WHERE. Por ejemplo, si a la consulta anterior se le aplica la cláusula WHERE, ésta produce el mismo resultado:

```
SELECT      *
FROM  ARTÍCULO, DETALLE
WHERE ArtCódigo = Código AND
      FactNúmero = 103
```

El estándar SQL define la operación NATURAL JOIN a aquella que vincula las dos tablas especificadas, haciendo coincidir todas las columnas con el mismo nombre. En otras palabras, NATURAL JOIN opera sobre dos tablas R y S, sin especificar condición de concatenación, sino que se crea una condición EQUJOIN implícita para cada par de columnas con el mismo nombre en R y S (consultar la Sección 4.4.2). Se debe tomar en cuenta que, de cada par de columnas, en la tabla resultante se incluye solo uno de ellos. Para ilustrar este concepto, en la tabla DETALLE, a la columna *ArtCódigo* se la renombra como *Código*, con el propósito de contar en la base de datos, con el mismo nombre para las columnas que almacenan el código del artículo en las tablas ARTÍCULO y DETALLE. Con estas consideraciones, la consulta anterior se puede escribir en los siguientes términos:

```
SELECT      *
FROM  ARTÍCULO NATURAL JOIN DETALLE
WHERE FactNúmero = 103
```

Código	Descripción	Precio	Unidad	ExiMáx	ExiMín	FactNúmero	Cantidad
11	Suelda	20,0	Kilo	10	5	103	1
9	Pintura	19,5	Galón	15	4	103	4
1	Cemento	6,0	Saco	50	10	103	4

En esta instrucción al tener las columnas de concatenación el mismo nombre (*Código*), no es necesario especificar una condición de concatenación. El resultado de la consulta, solo tiene ocho columnas, las que dan información de los artículos vendidos en la factura número 103. En este resultado no se repite la columna (*Código*) que coincide en ambas tablas, sino que aparece una sola vez.

La cláusula `FROM` de este ejemplo puede ser sustituida por la siguiente alternativa que proporciona el estándar SQL.

```
FROM ARTÍCULO JOIN DETALLE USING Código
```

La operación `JOIN...USING` es una forma de reunión natural que solo requiere que los valores sobre determinadas columnas correspondan. Al igual que la condición `ON` la especificación de la lista de atributos aparecen al final de la expresión de concatenación. Algunos sistemas de gestión de bases de datos todavía no admiten el uso de `USING`; no obstante se puede obtener el mismo resultado con la cláusula `ON`.

En el ejemplo anterior, para mostrar el uso de la operación `NATURAL JOIN`, se supuso que los nombres de las columnas de concatenación serían iguales; sin embargo, si los nombres de las columnas no coinciden, es posible previamente renombrarlos para su posterior aplicación. A partir del ejemplo anterior se puede generar un par de columnas con el mismo nombre, renombrando las columnas de la tabla `DETALLE`. La consulta se puede escribir en SQL como:

```
SELECT      *
FROM (ARTÍCULO NATURAL JOIN
      (DETALLE AS DETA (Factura, Código, Cantidad)))
WHERE FactNúmero = 103
```

En una consulta SQL una cláusula `FROM` puede tener varias relaciones combinadas, utilizando la operación `NATURAL JOIN`, como se describe a continuación:

```
SELECT C1, C2,...Cn
FROM t1 NATURAL JOIN t2, NATURAL JOIN … NATURAL JOIN tm
WHERE P
```

5.2.8.2 Concatenación externa

Para especificar la concatenación entre dos tablas, a la opción `INNER JOIN` (igual que `JOIN`), SQL dispone de las siguientes alternativas denominadas concatenaciones externas: `LEFT OUTER JOIN`, `RIGHT OUTER JOIN` y `FULL OUTER JOIN`, siendo opcional la palabra `OUTER`. La definición de estas operaciones fueron analizadas en la Sección 4.4.4.

El cálculo de una operación de concatenación externa entre las tablas `R` y `S`, se puede realizar de la siguiente forma: primero se calcula el resultado de una concatenación interna como se analizó anteriormente; luego, para cada fila de la parte izquierda (`R`) de la concatenación que no corresponda con ninguna fila de la parte derecha (`S`), se añade una fila al resultado, considerando el siguiente análisis: Los valores de las filas de la parte izquierda se rellenan con los correspondientes de la tabla S; el resto de valores se rellenan con `NULL`.

A continuación el siguiente ejemplo ilustra el uso de la concatenación externa izquierda:

```
SELECT Nombre, Apellido1, Apellido2, Vnombre, Apellido
FROM CLIENTE LEFT OUTER JOIN VENDEDOR
ON  CódRepre = CódigoVend
```

Nombre	Apellido1	Apellido2	Vnombre	Apellido
Víctor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Marcia	Mora	Durazno	Diego	Loja
Manuel	Castro	León	Antonio	Calle
Jaime	Pérez	Guerrero	Antonio	Calle
Pablo	Brito	Parra	Lucía	Serrano
Humberto	Pons	Coronel	Lucía	Serrano
Alberto	Torres	Viteri	Arturo	Salto
Elena	Tapia	Barrera	Null	Null

Para el caso de la concatenación externa derecha, el procedimiento es similar, puesto que se considera que la concatenación externa derecha es simétrica a la concatenación externa izquierda (Silberschatz , A. Korth, H. Sudarshan, S., 2014). Por ejemplo:

```
SELECT Nombre, Apellido1, Apellido2, Vnombre, Apellido
FROM CLIENTE RIGHT OUTER JOIN VENDEDOR
ON  CódRepre = CódigoVend
```

Nombre	Apellido1	Apellido2	Vnombre	Apellido
Víctor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Marcia	Mora	Durazno	Diego	Loja
Manuel	Castro	León	Antonio	Calle
Jaime	Pérez	Guerrero	Antonio	Calle
Pablo	Brito	Parra	Lucía	Serrano
Humberto	Pons	Coronel	Lucía	Serrano
Alberto	Torres	Viteri	Arturo	Salto
Null	Null	Null	Maricela	Vera

La reunión externa completa o `FULL OUTER JOIN` es una combinación de los dos tipos de reunión externa izquierda y derecha. La operación primero calcula el resultado de la concatenación interna, luego le asigna un valor null a las filas de la parte izquierda de la concatenación que no corresponden con ninguna de la parte derecha, y le añaden al resultado. De igual manera se realiza el proceso para las filas de la parte derecha. Por ejemplo:

```
SELECT Nombre, Apellido1, Apellido2, Vnombre, Apellido
FROM CLIENTE FULL OUTER JOIN VENDEDOR
ON  CódRepre = CódigoVend
```

Nombre	Apellido1	Apellido2	Vnombre	Apellido
Víctor	Castro	Torres	Diego	Loja
Juan	Polo	Ávila	Diego	Loja
Marcia	Mora	Durazno	Diego	Loja
Manuel	Castro	León	Antonio	Calle
Jaime	Pérez	Guerrero	Antonio	Calle
Pablo	Brito	Parra	Lucía	Serrano
Humberto	Pons	Coronel	Lucía	Serrano
Alberto	Torres	Viteri	Null	Null
Null	Null	Null	Maricela	Vera

Si las columnas de concatenación tienen el mismo nombre se puede especificar la concatenación natural utilizando la palabra clave `NATURAL`. Por ejemplo, `NATURAL LEFT OUTER JOIN`

5.2.9 Consulta de resumen

SQL a más de extraer filas y columnas de una base de datos, permite resumir los datos mediante un conjunto de funciones de columna o de agregación. Estas funciones operan sobre una única columna de una tabla y devuelven un único valor. El estándar define cinco funciones de agregación:

- `COUNT( )` cuenta el número de valores de una columna especificada.
- `SUM ( )` calcula la suma de los valores contenidos en una columna.
- `AVG ( )` permite calcular el promedio de los valores contenidos en una columna.
- `MIN ( )` encuentra el valor más pequeño contenido en una columna.
- `MAX ( )` encuentra el máximo valor contenido en una columna.

Las funciones `MIN, MAX y COUNT` se aplican a datos numéricos o no numéricos como cadenas de caracteres; por el contrario, las funciones `SUM` y `AVG` se aplican únicamente a datos numéricos.

Dentro del paréntesis de una función de agregación se coloca el argumento que puede ser un nombre de columna o una expresión aritmética, por ejemplo: `SUM(Salario)` o `AVG(Salario * 1,2)`. Las funciones de agregación solo pueden utilizarse con las cláusulas `SELECT` y `HAVING` que se analizarán más adelante.

La función `COUNT(*)` es un caso especial de `COUNT` y sirve para contar el número de filas de una tabla, sin considerar si éstas contienen valores nulos o duplicados.

5.2.9.1 Función AVG( )

La función de agregación `AVG()` suma los valores de una columna y luego divide para el número de valores. El resultado de la función no necesariamente puede tener el mismo tipo de datos que el de la columna. Por ejemplo, si los datos de la columna son enteros, al dividirlos el resultado puede ser un número decimal.

Por ejemplo, en la siguiente consulta: determinar el promedio del límite de crédito de los clientes que tienen como representante al vendedor de código 1.

```
SELECT AVG(LímCrédito)
FROM  CLIENTE
WHERE  CódRepre = 1
```

AVG(LímCrédito)
900

El resultado de esta consulta es una tabla que contiene una fila y una columna. Al nombre de la columna del resultado es posible renombrarlo utilizando la cláusula `AS`, como se indica a continuación:

```
SELECT AVG(LímCrédito) AS Crédito_Medio
FROM  CLIENTE
WHERE  CódRepre = 1
```

5.2.9.2 Función SUM( )

Mediante la función **SUM()** se puede calcular la suma de los valores numéricos de una columna. Esta función procesa todos los valores no nulos y devuelve como resultado un sólo valor. Por ejemplo, para calcular el importe total de las ventas realizadas en la factura número 100.

```
SELECT SUM(Precio * Cantidad) AS Importe
FROM  ARTÍCULO, DETALLE
WHERE  Código = ArtCódigo AND
       FactNúmero = 100
```

Importe
46,5

5.2.9.3 Función MAX( ), MIN( )

Las funciones **MAX()** y **MIN()** devuelven respectivamente el valor máximo y mínimo contenidos en una columna. Los valores pueden ser numéricos, de cadena o de fecha. Por ejemplo, determinar el mayor y menor límite de crédito de los clientes.

```
SELECT MAX(LímCrédito), MIN(LímCrédito)
FROM  CLIENTE
```

MAX (LímCrédito)	MIN(LímCrédito)
2000	600

Un ejemplo con datos tipo fecha podría ser el siguiente: ¿Cuál es la fecha del contrato del vendedor más antiguo?

```
SELECT MIN(FechaContrato)
FROM  VENDEDOR
```

FechaContrato
18-ago-08

5.2.9.4 Función COUNT( )

La función de agregación COUNT() cuenta el número de valores de una columna, estos valores pueden ser de cualquier tipo. El resultado de esta función siempre es un número entero, independiente del tipo de dato de la columna referida. En los siguientes ejemplos se muestran consultas con la función COUNT().

¿Cuántas cargas familiares de sexo femenino hay?

```
SELECT COUNT(SEXO)
FROM CARGAFAMILIAR
WHERE SEXO = ´F´
```

COUNT (Sexo)
3

Determinar el número de vendedores que tienen un objetivo de ventas mayor a 2.500 dólares.

```
SELECT COUNT(ObjVentas)
FROM  VENDEDORES
WHERE ObjVentas > 2500
```

COUNT (ObjVentas)
3

La función COUNT()cuenta cuántos datos hay en la columna sin tomar en consideración el valor del dato, de ahí que no afecta realmente en el resultado la columna que se declare en el argumento de la función. Por ejemplo, en el ejercicio anterior el resultado es el mismo al escribirse de la siguiente manera:

```
SELECT COUNT(CódigoVend)
FROM  VENDEDORES
WHERE ObjVentas > 2500
```

Nótese que en estos ejercicios no se ha considerado que las columnas tengan datos repetidos o con valores nulos, temas que serán tratados en las siguientes secciones.

La función `COUNT(*)` es un caso especial de la función `COUNT()`, que sirve para contar el número de filas de una tabla, independientemente de si existen duplicados o valores nulos. Por ejemplo, para contar el número total de facturas, la instrucción podría escribirse como:

```
SELECT COUNT(*)
FROM   FACTURA
```

COUNT(*)
12

Varias funciones pueden ser el resultado de un `SELECT`, como por ejemplo la siguiente instrucción que contiene cinco funciones: Determinar el número de vendedores y, para los objetivos de ventas hallar: la suma, el promedio, el valor máximo y el valor mínimo.

```
SELECT    COUNT(*), SUM(ObjVentas), AVG(ObjVentas),
          MAX(ObjVentas), MIN(ObjVentas)
FROM      VENDEDOR
```

COUNT(*)	SUM(ObjVentas)	AVG(ObjVentas)	MAX(ObjVentas)	MIN(ObjVentas)
5	15000	3000	4000	2000

Cuando se trabaja con dos o más funciones de agregación se debe tener en cuenta que no se puede incluir una función de agregación dentro de otra. Esta restricción hace que la siguiente expresión no sea correcta.

```
SUM(AVG(ObjVentas))
```

5.2.9.5 Funciones de agregación con la palabra clave DISTINCT

SQL permite eliminar valores duplicados de una columna antes de emplear una función de agregación mediante la palabra clave DISTINCT, colocándola inmediatamente después del paréntesis izquierdo. Para aplicar DISTINCT el argumento de la función debe ser un simple nombre de columna.

El estándar de SQL permite utilizar la palabra clave DISTINCT para las funciones de SUM(), AVG() y COUNT(). No se permite usar DISTINCT para funciones MAX() y MIN() puesto que, no tiene ningún efecto en el resultado. De la misma manera no se utiliza con la función COUNT(*), ya que ésta cuenta filas en lugar de valores de datos de una columna. Solo se puede especificar DISTINCT una vez dentro de una misma consulta. A continuación se muestran varios ejemplos de consultas en las que se aplica la palabra clave DISTINCT.

Determinar el número de vendedores que tienen cargas familiares.

```
SELECT COUNT(DISTINCT CódVendedor)AS Número
FROM  CARGAFAMILIAR
```

Número
3

En esta consulta se debe tomar en consideración que un mismo vendedor puede tener varias cargas familiares, por lo que es necesario emplear la palabra clave DISTINCT para eliminar los vendedores duplicados.

Determinar el número de clientes que recomendaron a otros clientes.

```
SELECT COUNT(DISTINCT CédulaReco) AS Recomiendan
FROM  CLIENTE
```

Recomiendan
3

Calcular el promedio de todos los límites de crédito distintos de los clientes.

```
SELECT AVG(DISTINCT LímCrédito) AS Promedio
FROM  CLIENTE
```

Promedio
1233,33

5.2.9.6 Funciones de agregación con valores NULL

El proceso de las operaciones de una consulta se dificulta cuando la columna contiene uno a más valores nulos. El estándar de SQL y como regla general para la operación de las funciones de agregación con presencia de valores `NULL` especifica que todas la funciones excepto `COUNT(*)`, deben ignorar los valores nulos que aparezcan como entrada. "Como resultado de esta regla de ignorar los valores nulos, la colección de entrada puede estar vacía. Se define que `COUNT()` de una colección vacía debe valer 0, y el resto de operaciones de agregación devuelve un valor `NULL` cuando se aplica a una colección vacía" (Silberschatz , A. Korth, H. Sudarshan, S., 2014, pág. 40).

Para ejemplificar estos conceptos se utiliza la tabla `T` de la Figura 5.16, que contiene las columnas `A` y `B`, y cinco filas.

T

A	B
3	1
5	NULL
2	2
1	1
4	1

Figura 5.16 Tabla que incluye valores `NULL`.

Si se aplica la función `COUNT()` y su caso especial `COUNT(*)` a la tabla `T`, se puede observar cómo los valores `NULL` son ignorados. Por ejemplo, si se quiere contar el número de filas, el número de valores de la columna `A` y el número de valores de la columna `B`, la instrucción se podría escribir de la siguiente manera:

```
SELECT COUNT(*), COUNT(A), COUNT(B)
FROM  T
```

COUNT(*)	COUNT(A)	COUNT(B)
5	5	4

En esta tabla de resultados la función `COUNT(*)` devuelve un valor de cinco, puesto que la función cuenta filas independientemente de si existen o no valores nulos. Para el caso de `COUNT(A)` el resultado también es cinco, porque la columna no contiene valores nulos; sin embargo, para el caso de la función `COUNT(B)`, cuenta y devuelve como resultado cuatro, ignorando el valor `NULL`.

La siguiente consulta sirve para analizar el problema que puede presentarse con el uso de las funciones de agregación `SUM()` y `AVG()`, cuando una columna, contiene valores nulos.

```
SELECT SUM(A), SUM(B), SUM(A)-SUM(B), SUM(A-B)
FROM  T
```

SUM(A)	SUM(B)	SUM(A)-SUM(B)	SUM(A-B)
15	5	10	5

La función `SUM(A)`, suma todos los cinco valores de la columna, en tanto que, la función `SUM(B)` suma únicamente los cuatro valores de la columna, que no son nulos.

Aparentemente las expresiones `SUM(A)-SUM(B)` y `SUM(A-B)` producen el mismo efecto, sin embargo, al observar el resultado de la consulta, se muestra que éstas son diferentes. Las dos expresiones son correctas, la primera calcula la suma de los valores de la columna `A` menos la suma de los valores de la columna `B`. La segunda calcula la suma de los valores de `A` menos los valores de `B`. En este caso la presencia del valor `NULL` hace que los resultados cambien, la resta entre un valor no nulo y un valor nulo genera un resultado nulo, en consecuencia la expresión `SUM(A-B)` suma únicamente cuatro valores.

A pesar de que el estándar SQL especifica claramente las reglas que operan sobre los valores `NULL`, los SGBD comerciales pueden producir resultados diferentes a los del estándar; de ahí que, se debería determinar el comportamiento del sistema de gestión de bases de datos que se use ante la presencia de valores nulos.

5.2.9.7 Funciones de agregación con la cláusula `GROUP BY`

Las consultas de agregación revisadas hasta el momento aplican una función de agregación a un único conjunto de filas de una columna y generan como resultado un único valor. A veces es necesario aplicar la función a un grupo de un conjunto de filas para resumir los resultados por niveles "subtotales".

La cláusula **GROUP BY** agrupa filas con valores idénticos de la lista de columnas especificadas. Para cada grupo de filas se crea un único resultado si se incluye una función de agregación.

Los valores nulos de la columna especificada se agrupan y no se omiten, pero éstos no se evalúan en ninguna de las funciones de agregación. A las consultas que incluyen una cláusula **GROUP BY** se las denomina consultas de agrupamiento y al listado de columnas después de la cláusula se denomina columnas de agrupación (Connolly T., Begg C., 2005) (Groff J., Weinberg P., 2003).

Partiendo de una consulta que no incluye una cláusula **GROUP BY**, a continuación se presenta una serie de ejemplos que ilustran el uso de la cláusula:

¿Cuántas cargas familiares hay?

```
SELECT COUNT(*)
FROM  CARGAFAMILIAR
```

COUNT(*)
5

Para obtener el resultado, SQL cuenta todas las filas de la tabla **CARGAFAMILIAR**; en cambio, si se requiere un reporte del número de cargas familiares que tiene cada uno de los vendedores, la instrucción requiere de la cláusula **GROUP BY** y se escribe de la siguiente manera:

```
SELECT CódVendedor, COUNT(*)
FROM  CARGAFAMILIAR
GROUP BY CódVendedor
```

CódVendedor	COUNT(*)
1	2
2	2
3	1

SQL agrupa las filas de acuerdo al código del vendedor, luego para cada grupo cuenta el número de filas. En la tabla resultante cada fila contiene la columna con el código del vendedor *CódVendedor* y su número de cargas familiares **COUNT(*).**

¿Calcular la suma de los límites de crédito de los clientes, agrupados por su representante?

```
SELECT CódRepre, SUM(LímCrédito)
FROM  CLIENTE
GROUP BY CódRepre
```

En esta consulta SQL agrupa los clientes por cada representante, luego para cada grupo suma los valores de la columna *LímCrédito*, generando una única fila de resultados por valor de la columna *CódRepre*. En la Figura 5.17 se muestra el detalle del proceso de esta consulta.

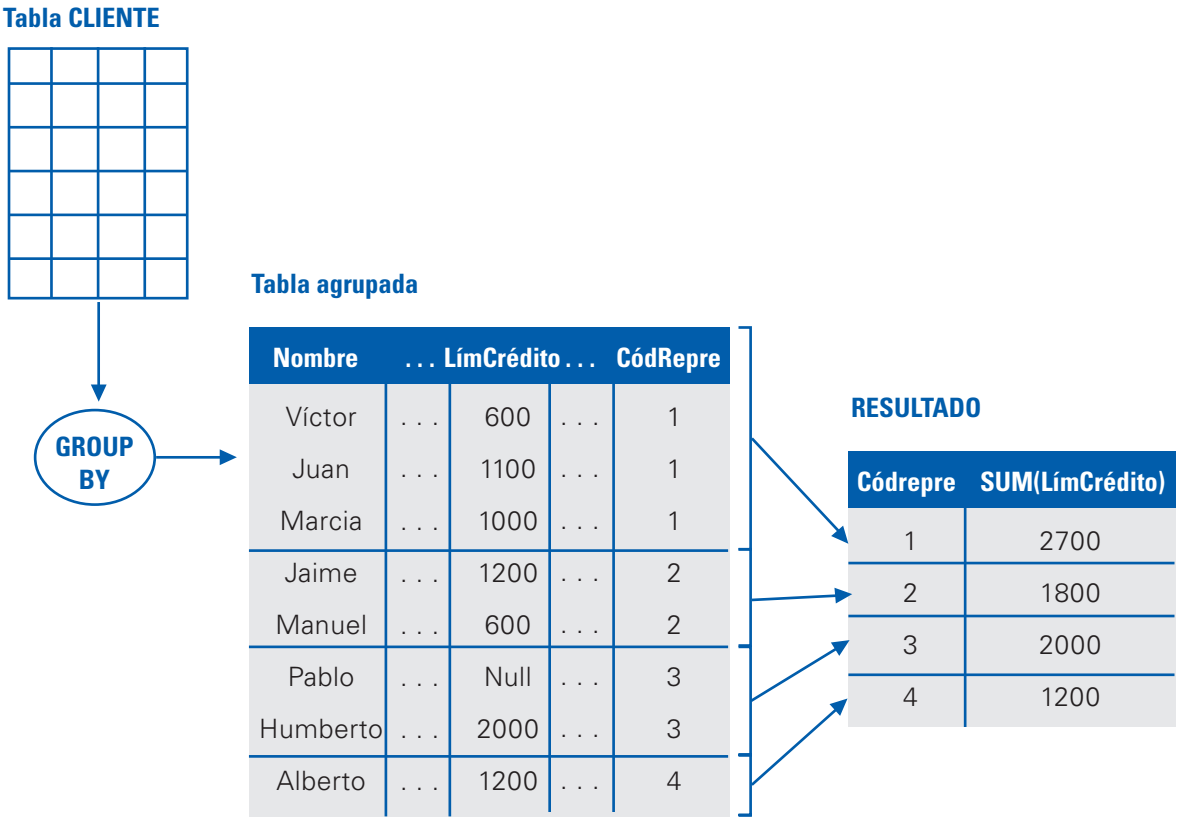


Figura 5.17. Proceso de una consulta con la cláusula `GROUP BY` y función de agregación.

La siguiente consulta determina el monto total de cada factura.

```
SELECT FactNúmero, SUM(Cantidad * Precio) AS Total
FROM  ARTÍCULO, DETALLE
WHERE  Código = ArtCódigo
GROUP BY FactNúmero
```

FactNúmero	Total
100	46,5
101	140
102	18
103	122
104	63
105	30
106	38,4
107	26,4
108	84
109	20
110	6,6
111	114

Cuando una consulta utiliza agrupamiento es necesario garantizar que solo los nombres de las columnas que aparecen en la sentencia `SELECT` sin agregarse se encuentren en la cláusula `GROUP BY`. Por ejemplo, la siguiente consulta no es correcta, puesto que *Código* no aparece en la cláusula `GROUP` BY y sí aparece en la cláusula `SELECT`, sin haberse agregado:

```
/* consulta errónea */
SELECT Código, FactNúmero, SUM(Cantidad*Precio) AS Total
FROM  ARTÍCULO, DETALLE
WHERE  Código = ArtCódigo
GROUP BY FactNúmero
```

5.2.9.8 Consultas agrupadas con la cláusula `HAVING`

De igual manera que la cláusula `WHERE` que se usa para seleccionar filas individuales, la cláusula `HAVING` es útil para indicar una condición de búsqueda que se aplica a grupos de filas. SQL aplica las condiciones de la cláusula `HAVING` después de haber realizado los agrupamientos. Por

ejemplo, para obtener un listado con el nombre de los vendedores que tienen más de una carga familiar, la consulta se puede escribir de la siguiente manera:

```
SELECT Vnombre, COUNT(CódVendedor) AS NúmeroCargas
FROM  CARGAFAMILIAR, VENDEDOR
WHERE  CódVendedor = CódigoVend
GROUP BY CódVendedor
HAVING COUNT(CódVendedor)> 1
```

Vnombre	NúmeroCargas
Diego	2
Antonio	2

El proceso de esta consulta se define en función de la siguiente secuencia de operaciones:

- En primer lugar se evalúa la cláusula `FROM` para obtener una tabla.
- Si existe la cláusula `WHERE`, como en el caso de este ejemplo, al resultado de la cláusula `FROM` se le aplica la condición de búsqueda de la cláusula `WHERE`.
- Las filas del resultado de aplicar la cláusula `WHERE` se agrupan de acuerdo con la cláusula `GROUP BY`.
- A cada grupo de filas generadas en la operación anterior se les aplica la función de agregación y los resultados obtenidos son evaluados mediante la condición de búsqueda de la cláusula `HAVING`.

Como ejemplo adicional del uso de la cláusula `HAVING`, a continuación se desarrolla la siguiente consulta: Enlistar los clientes que registran más de una factura en La Ferretería. La consulta se puede escribir como:

```
SELECT        Nombre, Apellido1, COUNT(CcCliente) AS Facturas
FROM          CLIENTE, FACTURAS
WHERE         Cédula = CcCliente
GROUP BY      CcCliente
HAVING COUNT  (CcCliente)> 1
```

Nombre	Apellido1	Facturas
Juan	Polo	2
Manuel	Castro	3

5.2.10 Subconsultas

SQL permite anidar una consulta, dentro de otra (consulta principal), es decir, utiliza los resultados de una consulta para formar parte de otra. Las subconsultas aparecen siempre como parte de una cláusula `WHERE`, `HAVING` o la cláusula `FROM` de una consulta principal. Se encuentra encerrada entre paréntesis y tiene un formato idéntico al de una sentencia `SELECT`, que incluye la cláusula `FROM` y las cláusulas opcionales `WHERE`, `GROUP BY` y `HAVING`. Una subconsulta cumple las funciones normales que el `SELECT` de la consulta principal; sin embargo, existen algunas diferencias que se detallan a continuación:

- Por lo general una subconsulta produce una única columna de datos, esto quiere decir que en la cláusula `SELECT` se incluirá solo un elemento de selección; sin embargo, en la condición de búsqueda de pertenencia a un conjunto, es posible incluir un número de elementos arbitrarios de selección (Silberschatz , A. Korth, H. Sudarshan, S., 2014), como se verá más adelante.

- En la subconsulta no está permitido el uso de la cláusula `ORDER BY,` debido a que los resultados de la subconsulta serán usados internamente por la consulta principal y no serán visibles para el usuario.

- En una subconsulta solo se permite una sentencia `SELECT` y no puede ser la `UNIÓN` de varias sentencias `SELECT`.

- No se puede utilizar una subconsulta como una expresión para el cálculo de una función de agregación.

- Los nombres de las columnas de una subconsulta pueden referirse a nombres de columnas de las tablas especificadas en la consulta principal. Este concepto recibe el nombre de **Referencia externa**, que se revisará en la sección 5.2.10.4.

A continuación se estudian las diferentes condiciones de búsqueda en subconsultas que aparecen en la cláusula `WHERE` o `HAVING`:

5.2.10.1 Comparación

La condición de búsqueda de comparación en una subconsulta compara el valor de una expresión con el único valor (una única fila y una única columna) calculado por una subconsulta; si devuelve más de una fila y una columna, SQL informará del error.

Igual que en la comparación simple, los seis operadores (=, <>, <, >, ≤, ≥) están disponibles para las subconsultas. Si la comparación es cierta SQL devuelve un resultado `TRUE`. A continuación se presentan ejemplos de esta opción:

Enlistar todos los clientes que tienen como representante al vendedor Antonio Calle.

```
SELECT Nombre, Apellido1
FROM  VENDEDOR, CLIENTE
WHERE CódRepre = (SELECT CódigoVend
                  WHERE Nombre = ´Antonio´ AND
                  Apellido1 = ´Calle´)
```

Nombre	Apellido1
Jaime	Pérez
Manuek	Castro

Enlistar el nombre de los vendedores que tienen un objetivo de ventas superior al promedio del objetivo de ventas de todos los vendedores.

```
SELECT Vnombre, Apellido
FROM   VENDEDOR
WHERE  ObjVentas > (SELECT AVG(ObjVentas)
                    FROM   VENDEDOR)
```

Nombre	Apellido
Arturo	Salto
Maricela	Vera

5.2.10.2 Pertenencia a conjuntos (IN)

Esta condición de búsqueda se usa cuando se quiere comparar un valor de la fila de la consulta principal, con un conjunto de valores producido por una subconsulta. Esta opción funciona exactamente igual a la de una consulta simple `IN`, con la diferencia que el conjunto de valores es el resultado de una subconsulta. Por ejemplo, si se requiere un listado de todos los artículos que se adquirieron con la factura número 103. La consulta podría escribirse de la siguiente manera:

```
SELECT Descripción
FROM   ARTÍCULO
WHERE  Código IN(SELECT ArtCódigo
                 FROM DETALLE
                 WHERE FactNúmero = 103)
```

Descripción
Suelda
Pintura
Cemento

En este ejemplo la subconsulta genera un conjunto de tres valores (11, 9, 1) correspondientes a los códigos de cada uno de los artículos que fueron adquiridos en la factura 103. En la consulta principal se comprueba la pertenencia de cada valor de la columna *Código* de la tabla `ARTÍCULO` con el conjunto que produjo la subconsulta.

Para comprobar la no pertenencia a un conjunto se utiliza la palabra reservada `NOT IN`. Por ejemplo, si se quiere un listado de todos los vendedores que no tienen cargas familiares. La consulta puede escribirse del modo siguiente:

```
SELECT Vnombre, Apellido
FROM VENDEDOR
WHERE CódigoVend NOT
      IN (SELECT CódVendedor
      FROM CARGAFAMILIAR)
```

Nombre	Apellido
Arturo	Salto
Maricela	Vera

En los ejemplos anteriores se aplica la condición de pertenencia a un conjunto, usando una columna; sin embargo, es posible comprobar esta condición para un número arbitrario de condiciones. Por ejemplo, enlistar el nombre del vendedor que tiene una carga familiar con el mismo nombre.

```
SELECT Vnombre
FROM  VENDEDOR
WHERE (CódigoVend, Vnombre)
       IN (SELECT CódVendedor, NomDep
       FROM CARGAFAMILIAR)
```

Vnombre
Antonio

5.2.10.3 Cuantificados (SOME y ALL)

SQL incluye las palabras claves SOME y ALL para utilizar en aquellas subconsultas que producen una única columna de números. Una condición de búsqueda cuantificada permite realizar una comparación entre una expresión, un operador de comparación y el resultado de una subconsulta, con una de las opciones SOME o ALL.

Opción SOME

Si una subconsulta está precedida por la palabra clave SOME, la evaluación solo será verdadera si alguna de las comparaciones individuales es TRUE. En este caso la opción SOME devuelve un resultado TRUE.

La condición de búsqueda con SOME expresa "**al menos una**" y se usa en conjunción con uno de los seis operadores, permitiendo realizar las comparaciones < SOME "**menor que**

al menos una", `> SOME, <= SOME, >= SOME, = SOME` y `<> SOME.` La comparación = `SOME` es idéntica a `IN`, sin embargo, `<> SOME` no es el equivalente a `NOT IN`.

La instrucción `WHERE X < SOME (SELECT Y ...)`, se puede leer de la siguiente manera:

"en donde para al menos un `Y, X` es menor que `Y`".

Por ejemplo, si X = 1000 y la subconsulta genera una columna Y con los valores (600, 500, 1200, 800), el resultado de la opción `SOME` es `TRUE`, debido a que, existe al menos un valor (1200) que cumple la condición, como se muestra resaltada en la Figura 5.18.

WHERE	X	<SOME	Y
1000		<	600
			500
			1200
			800

Figura 5.18. Condición de búsqueda `SOME`.

En algunos SQL se utiliza la palabra `ANY` como sinónimo de `SOME`. En un inicio los gestores solo permitían la palabra `ANY`; pero posteriormente se incluyó la alternativa `SOME`, con el objeto de evitar ambigüedades lingüísticas de la palabra `ANY` (cualquiera) en lenguaje natural. (Silberschatz , A. Korth, H. Sudarshan, S., 2014).

Los siguientes ejemplos de consultas ilustran la condición de búsqueda con la palabra clave `SOME`:

Recuperar el nombre de los clientes que tienen al menos un límite de crédito mayor a los límites de crédito de los clientes representados por el vendedor de código 1.

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE LímCrédito > SOME (SELECT LímCrédito
                         FROM CLIENTE
                         WHERE CódRepre = 1)
```

Enlistar el nombre de los vendedores que tienen cargas familiares

```
SELECT Vnombre, Apellido
FROM  VENDEDOR
WHERE CódigoVend = SOME (SELECT CódVendedor
                         FROM CARGAFAMILIAR)
```

Vnombre	Apellido
Diego	Loja
Antonio	Calle
Lucía	Serrano

Opción ALL

Si una subconsulta está precedida por la palabra clave **ALL**, la evaluación solo será verdadera si todas las comparaciones individuales dan un resultado verdadero. En ese caso la opción **ALL** devuelve un resultado **TRUE**.

La condición de búsqueda con **ALL** expresa "**a todos**" y se usa en conjunción con uno de los seis operadores, permitiendo realizar la comparaciones **< ALL** "**menor a todos**", **> ALL**, **<= ALL, >= ALL, = ALL y <> ALL**. La comparación **<> ALL** es idéntica a **NOT IN**, sin embargo, **= ALL** no es el equivalente a IN.

La instrucción **WHERE X < ALL (SELECT Y ...)**, se puede leer de la siguiente manera:

"en donde para todo **Y, X** es menor que **Y**".

Si a los datos del ejemplo anterior que fueron utilizados para analizar la opción **SOME**, se les aplica la opción **ALL**, el resultado es **FALSE**, ya que el valor de **X** no es menor a todos los valores de **Y**, como se resalta en la Figura 5.19.

WHERE	X	<ALL	Y
1000		<	**600**
			500
			1200
			800

Figura 5.19. Condición de búsqueda ALL.

El siguiente ejemplo ilustra el uso de las condiciones de búsqueda mediante la palabra clave ALL:

Recuperar el nombre de los clientes que tienen un límite de crédito mayor a todos los límites de crédito de los clientes representados por el vendedor Diego Loja.

```
SELECT Nombre, Apellido1
FROM  CLIENTE
WHERE LímCrédito > ALL (SELECT LímCrédito
                        FROM CLIENTE, VENDEDOR
                        WHERE Vnombre = ´Diego´ AND
                               Apellido = ´Loja´ AND
                               CódigoVend = CódRepre)
```

5.2.10.4 Existencia (EXISTS)

La condición de búsqueda con las palabras claves EXISTS y NOT EXISTS se utiliza en comparaciones de verdadero /falso, para determinar si la subconsulta devuelve alguna fila como resultado. La evaluación EXISTS es TRUE, si y solo si existe una fila en la tabla de resultados generada por la subconsulta; y es FALSE si devuelve una tabla de resultados vacía. La opción EXISTS no produce valores NULL y se utiliza únicamente con subconsultas.

La palabra clave NOT EXISTS corresponde a la lógica inversa del EXISTS, es decir, la evaluación es FALSE si la subconsulta produce filas, y TRUE en caso de que no produzca.

SQL permite utilizar la forma SELECT * en la subconsulta a continuación de las palabras claves, en razón de que la condición de búsqueda no usa realmente los resultados de la subconsulta. Por ejemplo, para recuperar el nombre de los vendedores que tiene cargas familiares, la consulta podría escribirse como:

```
SELECT Vnombre, Apellido
FROM VENDEDOR
WHERE EXISTS (SELECT *
              FROM CARGAFAMILIAR
              WHERE CódigoVend = CódVendedor)
```

En este ejemplo la subconsulta incluye una **referencia externa** a una columna de la tabla de la consulta principal. En la subconsulta la columna *CódigoVend* hace referencia a la tabla VENDEDOR de la consulta principal. Una referencia externa es un nombre de columna que no se refiere a ninguna de las tablas especificadas en la cláusula FROM de la subconsulta, sino que se refiere a una columna de una tabla especificada en el FROM de la consulta principal.

En la práctica, la subconsulta con la palabra clave EXISTS siempre contiene una referencia externa, que enlaza la subconsulta a la fila que está siendo examinada por la consulta principal.

5.2.11 Subconsultas en la cláusula HAVING

La mayor parte de subconsultas se presentan en la cláusula WHERE; sin embargo, es posible también utilizar en la cláusula HAVING, como se muestra en el siguiente ejemplo:

Enlistar el nombre del vendedor cuyo promedio del límite de crédito de sus clientes a quienes representa, es mayor al promedio del límite de crédito de todos los clientes.

```
SELECT Vnombre, Apellido
FROM  VENDEDOR, CLIENTE
WHERE CódigoVend = CódRepre
GROUP BY CódRepre
HAVING AVG(LímCrédito) > (SELECT AVG(LímCrédito)
                          FROM CLIENTE)
```

5.3 ACTUALIZACIÓN DE LA BASE DE DATOS

El componente del lenguaje SQL para la manipulación de datos (DML), adicionalmente a la opción de consulta, incluye comandos para modificar los datos de una base de datos. A estas opciones se las denomina consultas de acción y son las encargadas de borrar, añadir y modificar una fila. Las instrucciones que corresponden a estas acciones son: `DELETE, INSERT y UPDATE` respectivamente.

5.3.1 BORRADO (`DELETE`)

La instrucción `DELETE` borra las filas completas seleccionadas de una tabla. El borrado de filas se expresa mediante el siguiente formato general:

`DELETE FROM` tabla `WHERE` criterio

Donde la cláusula `FROM` especifica la tabla destino que contiene las filas a ser borradas. La cláusula `WHERE` contiene una condición de búsqueda de las filas que van a ser eliminadas. Esta instrucción borra las filas completas. No es posible eliminar el contenido de alguna columna en concreto. Se puede prescindir de la cláusula `WHERE`, en este caso, se borran todas las filas de la tabla.

Los siguientes ejemplos ilustran el uso de la instrucción para el borrado de filas:

- Si el vendedor Arturo Salto decide renunciar a La Ferretería, la instrucción `DELETE` que elimina la fila correspondiente se puede escribir de la siguiente manera:

```
DELETE FROM VENDEDOR
WHERE  Vnombre = ´Arturo´ AND Apellido = ´Salto´
```

- Si se quiere eliminar todas las filas de la tabla **CLIENTE**; se escribe la instrucción sin la cláusula **WHERE**, de la siguiente manera:

```
DELETE FROM VENDEDOR
```

Esta instrucción suprime los datos de la tabla **CLIENTE**, la tabla sigue existiendo, pero vacía.

- Borrar las cargas familiares del vendedor Diego Loja.

```
/* consulta errónea */
DELETE FROM CARGAFAMILIAR, VENDEDOR
WHERE CódigoVend = CódVendedor AND
                   Vnombre = ´Diego´ AND
                   Apellido = ´Loja´
```

En esta instrucción las filas seleccionadas, que serán eliminadas, dependen de los datos contenidos en otra tabla, razón por la cual se tiene que referir a más de una tabla. Esta instrucción presenta un error al especificar más de una tabla en la cláusula **FROM**.

Para evitar estos inconvenientes se puede referir a cualquier número de tablas en una **SELECT – FROM — WHERE** anidado mediante una subconsulta a la cláusula **WHERE** de un **DELETE**, como se indica a continuación.

```
DELETE FROM CARGAFAMILIAR
WHERE CódVendedor = (SELECT CódigoVend
                     FROM   VENDEDOR
                     WHERE  Vnombre = ´Diego´ AND
                            Apellido = ´Loja´
```

5.3.2 INSERTAR (INSERT)

Esta instrucción, en términos generales, agrega una fila a una tabla o formula una consulta, cuyo resultado es el conjunto de filas que serán insertadas en la tabla. Los valores de las columnas de las filas que se insertan deben tener el mismo dominio que el de la tabla destino. A una fila se la considera como la unidad de datos más pequeña que puede añadirse a una base de datos relacional.

SQL facilita tres opciones para insertar filas de datos en una base de datos:

- Inserción de una fila.

La instrucción INSERT más sencilla es una solicitud de inserción de una fila y puede expresarse de la siguiente manera:

```
INSERT INTO tabla (columna 1, columna 2, … , columna n)
VALUES (valor 1, valor 2, … , valor 3)
```

Donde la expresión graba en la tabla el valor 1 en la columna 1, el valor 2 en la columna 2 y así sucesivamente. Por ejemplo:

Si se quiere ingresar un nuevo artículo con los siguientes datos:

Código:	12
Descripción:	Martillo
Precio:	7
Unidad:	unidad
Existencia máxima:	10
Existencia mínima:	3

La instrucción podría escribirse de la siguiente manera:

```
INSERT INTO ARTíCULO (Código, Descripción, Precio, Unidad, ExiMáx, ExiMín)
VALUES (12, ´Martillo´, 7, ´unidad´, 10, 3)
```

Si se insertan todas las columnas de la tabla se puede omitir el listado de las columnas en la cláusula `INSERT`. Al utilizar esta alternativa, SQL genera automáticamente todas las columnas de la tabla en secuencia de izquierda a derecha. De una manera más sencilla la instrucción anterior se puede escribir como:

```
INSERT INTO ARTíCULO
VALUES (12, ´Martillo´, 7, ´unidad´, 10, 3)
```

En ocasiones se presentan casos con tablas que tienen muchas columnas y sólo se asignarán valores a unas cuantas de ellas en la nueva fila. Las columnas que no fueron establecidas con ningún valor serán omitidas y se registran con `NULL`. Para utilizar esta alternativa se debe considerar que es posible omitir las columnas siempre y cuando éstas permitan valores `NULL` o `DEFAULT`, condición que fue establecida al momento de crear la tabla. Por ejemplo, para introducir una nueva fila en la tabla `ARTÍCULO`, de la que se conoce únicamente el código, la descripción, el precio y la unidad, la instrucción se escribe como:

```
INSERT INTO ARTíCULO (Código, Descripción, Precio, Unidad)
VALUES (12, ´Martillo´, 7, ´unidad´)
```

En este ejemplo las columnas no especificadas se establecen a `NULL` o `DEFAULT`.

También es posible insertar varias filas con una sola instrucción `INSERT`, separando las filas con comas y los valores de las columnas que corresponden a cada fila encerrarlos entre paréntesis.

Inserción de multifilas

Una segunda variación de la instrucción `INSERT` corresponde a la inserción de varias filas en una tabla, en donde el origen de las nuevas filas proviene del resultado de una consulta. SQL expresa la inserción de varias filas mediante el siguiente formato:

```
INSERT INTO Tabla (columna1, columna2, ... , columnaN)
SELECT TablaOrigen.columna1, TablaOrigen.columna2, ... TablaOrigen.columnaN
FROM  TablaOrigen
```

En este caso se seleccionarán las columnas 1, 2, ... , N de la `TablaOrigen` y se graban en los columnas 1, 2, ... , N de la `Tabla` (destino). La condición `SELECT` puede incluir la cláusula `WHERE` para filtrar las filas a copiar. Si la `Tabla` (destino) y la `TablaOrigen` poseen la misma estructura, la sintaxis se puede simplificar de la siguiente manera:

```
INSERT INTO Tabla SELECT TablaOrigen * FROM TablaOrigen
```

Por ejemplo, copiar en la tabla `CLIENTE2` el nombre y los dos apellidos de los clientes que tienen un límite de crédito mayor a 1.000 dólares.

```
INSERT INTO CLIENTE2 (Nombre, Apellido1, Apellido2)
SELECT Nombre, Apellido1, Apellido2
FROM  CLIENTE
WHERE LímCrédito > 1000
```

La representación gráfica de este ejemplo se muestra a continuación en la Figura 5.20

CLIENTE

Nombre	Apellido1	Apellido2	...	LímCrédito	...
Víctor	Castro	Torres	...	600	...
Juan	Polo	Ávila	...	1100	...
Elena	Tapia	Barrera	...	1500	...
Pablo	Brito	Parra	...	Null	...
Marcia	Mora	Durazno	...	1000	...
Jaime	Pérez	Guerrero	...	1200	...
Humberto	Pons	Coronel	...	2000	...
Alberto	Torres	Viteri	...	1200	...
Manuel	Castro	León	...	600	...

```
SELECT Nombre, Apellido1, Apellido2
FROM CLIENTE
WHERE LímCrédito > 1000
```

CLIENTE2

Nombre	Apellido1	Apellido2
Juan	Polo	Ávila
Elena	Tapia	Barrera
Jaime	Pérez	Guerrero
Humberto	Pons	Coronel
Alberto	Torres	Viteri

RESULTADO DE LA CONSULTA

Nombre	Apellido1	Apellido2
Juan	Polo	Ávila
Elena	Tapia	Barrera
Jaime	Pérez	Guerrero
Humberto	Pons	Coronel
Alberto	Torres	Viteri

Figura 5.20. Proceso de inserción de varias filas.

Carga masiva

La mayoría de productos comerciales de SGBD incluyen utilidades especiales de carga masiva para insertar en las tablas grandes conjuntos de filas. El objetivo de esta característica es ofrecer una técnica más eficiente para insertar datos, que la secuencia equivalente de instrucciones `INSERT`.

5.3.3 Actualizaciones (UPDATE)

La instrucción UPDATE permite modificar un valor dentro de una fila sin cambiar todos los valores de la misma. SQL expresa las actualizaciones a través de:

```
UPTADE tabla SET columna1 = valor1, columna2 = valor2,… , columnaN =
ValorN
WHERE <condición>
```

La cláusula SET especifica y calcula el nuevo valor de las columnas que se van a actualizar, y la cláusula WHERE selecciona las filas de la tabla que van a ser actualizadas. Esta cláusula se utiliza de igual manera que en una instrucción INSERT o DELETE. Por ejemplo, se requiere incrementar en un 10 por ciento el objetivo de ventas, a los vendedores que tienen un objetivo de ventas inferior al promedio total; la instrucción se puede escribir como:

```
UPDATE    VENDEDOR
SET       ObjVentas = ObjVentas * 1.1
WHERE     ObjVentas < (SELECT AVG (ObjVentas)
          FROM VENDEDOR)
```

La cláusula WHERE es opcional y si ésta se omite, la instrucción UPDATE actualiza todas las filas. Por ejemplo, si se desea incrementar el precio de los artículos un cinco por ciento, la instrucción se escribe:

```
UPDATE    ARTÍCULO
SET       Precio = Precio * 1.05
```

El siguiente ejemplo muestra el uso de la cláusula WHERE para seleccionar filas que se quieren actualizar, en función de datos contenidos en otra tabla. Incrementar el límite de crédito en 200 dólares a todos los clientes representados por el vendedor Antonio Calle.

```
UPDATE CLIENTE
SET    LímCrédito = LímCrédito + 200
WHERE  CódRepre = (SELECT CódigoVend)
       FROM VENDEDOR
       WHERE Vnombre = ´Antonio´ AND
             Apellido = ´Calle´)
```

CLIENTE

Nombre	Apellido1	Cédula	...	LímCrédito	CódRepre	...
Víctor	Castro	0102030405	...	600	1	...
Juan	Polo	0122334455	...	1100	1	...
Elena	Tapia	0111555666	...	1500	Null	...
Pablo	Brito	0123456789	...	Null	3	...
Marcia	Mora	0987654321	...	1000	1	...
Jaime	Pérez	0777888999	...	**1400**	2	...
Humberto	Pons	0456456456	...	2000	3	...
Alberto	Torres	0999998888	...	1200	4	...
Manuel	Castro	0123123123	...	**800**	2	...

En este ejemplo dos filas han sido actualizadas, las correspondientes a Jaime Pérez y Manuel Castro.

Cuando se realizan instrucciones `UPDATE` consecutivas es importante analizar el orden de ejecución. En ciertos casos pueden presentarse problemas de inconsistencia en los resultados obtenidos. Por ejemplo, si se requiere que todos los clientes que tengan un límite de crédito superior a 1.300 dólares reciban un incremento del 10 por ciento, en tanto que, los restantes reciban un 15 por ciento de aumento. Las dos siguientes instrucciones resuelven esta solicitud:

```
UPDATE    CLIENTE
SET       LímCrédito = LímCrédito * 1.1
WHERE      LímCrédito  > 1300;

UPDATE    CLIENTE
SET       LímCrédito = LímCrédito * 1.15
WHERE     LímCrédito <= 1300;
```

Si se cambia el orden de las dos instrucciones, los clientes con un límite de crédito justo por debajo de los 1.300 dólares, recibirán un incremento del 25 por ciento, como es el caso de los clientes que tienen un límite de crédito de 1.200 dólares.

Para evitar estos problemas del orden de actualización, SQL cuenta con las instrucción **CASE**, cuya estructura básica es similar a la instrucción **IF … THEN … ELSE** de los lenguajes de programación y se puede utilizar para ejecutar las dos instrucciones de actualización anteriores en una sola instrucción **UPDATE**, impidiendo la presencia de inconsistencia en los resultados.

```
UPDATE CLIENTE
SET LímCrédito = CASE
                  WHEN LímCrédito <= 1300
                  THEN LímCrédito * 1.15
                  ELSE LímCrédito * 1.1
             END;
```

5.4 RESTRICCIONES DE INTEGRIDAD DE DATOS

Cuando los contendidos de una base de datos se modifican con las instrucciones **INSERT**, **DELETE** o **UPDATE**, la consistencia e integridad de los datos puede perderse. Con el objeto de preservar estas condiciones de los datos almacenados, un SGBD relacional impone una serie de restricciones de integridad que limitan los valores de los datos. A continuación se presentan algunos ejemplos de integridad:

- El límite de crédito de un cliente debe ser menor a 3.000 dólares.
- El nombre de un cliente no puede ser nulo.
- No puede haber dos facturas con el mismo número.
- Todas las cargas familiares deben estar relacionadas con un vendedor.

En una base de datos se encuentran diferentes tipos de restricciones de integridad, que generalmente se identifican como parte del proceso de diseño del esquema de la base de datos y se declaran al momento de crear la tabla con el comando `CREATE TABLE`. En este apartado se revisan las restricciones de integridad que se pueden especificar como parte de la creación de una tabla: `NOT NULL`, `UNIQUE`, `DEFAULT`, `CHECK` y la integridad referencial.

5.4.1 Restricción `NOT NULL`

Ciertas columnas de una base de datos deben contener un valor de datos válido en cada fila, es decir, no se permite ausencia de valores o que contenga valores `NULL`. Esta restricción de integridad se declara mediante la especificación `NOT NULL` al momento de crear la tabla con el comando `CREATE TABLE`. Cualquier modificación de la base de datos que inserte un valor nulo en una columna declarada como `NOT NULL`, generará un error.

Para el caso de la clave primaria, que no permite valores nulos, no hace falta declarar `NOT NULL` de manera explícita.

Por ejemplo, si se quiere especificar la descripción del artículo y el precio como un dato requerido, se declara de la siguiente manera:

```
Descripción varchar(20) NOT NULL

Precio numeric(10,2) NOT NULL
```

5.4.2 Integridad de entidad

La restricción de integridad de entidad establece que ninguna clave primaria debe contener un valor `NULL`, debido a que esta clave sirve para identificar filas individuales en una tabla. Permitir el ingreso de un valor nulo significa que no será posible diferenciar dos filas que tengan registrado `NULL` en su clave primaria.

El SGBD comprueba automáticamente la unicidad del valor de la clave primaria con cada sentencia `INSERT` y `UPDATE` que se aplica a una tabla. Pretender ingresar un valor duplicado en una clave primaria generará un mensaje de error.

En ciertas ocasiones es necesario que una columna que no es clave primaria contenga valores únicos en cada fila (clave candidata). Este requerimiento se obtiene con la restricción de unicidad `UNIQUE`, que se declara al momento de crear la tabla. En esta restricción se permite que la clave candidata tenga valores `NULL`, a menos que explícitamente se haya declarado como `NOT NULL`.

5.4.3 Cláusula `DEFAULT`

Otro tipo de restricción que permite SQL es la de definir un valor predeterminado para una columna, con el uso de la cláusula `DEFAULT (valor)`. Si a una fila nueva no se le proporciona un valor explícito de una columna, el valor predeterminado será incluido en esa fila.

5.4.4 Chequeo de validez

El SGBD está preparado para controlar el ingreso de datos válidos para cada columna de acuerdo a su dominio y tipo. La cláusula `CHECK` garantiza que los valores de las columnas cumplan con las condiciones especificadas. Por ejemplo, para expresar la siguiente condición: que el límite de crédito de los vendedores no sea menor a 2.000 dólares, la restricción se escribe como:

```
CHECK (LímCrédito => 2000)
```

La cláusula `CHECK` se utiliza también para representar un tipo de enumeración. Por ejemplo, si se requiere restringir a "hijo", "hija" y "cónyuge" los valores de la columna Relación de la tabla `CARGAFAMILIAR`, la cláusula de escribe como:

```
CHECK (Relación IN (´Hijo´, ´Hija´, Cónyuge´))
```

Para especificar que los códigos de los vendedores estén restringidos a números entre 1 y 10, la expresión se escribe de la siguiente forma:

```
CHECK (CódigoVend > 0 AND CódigoVend < 11)
```

5.4.5 Integridad referencial

La restricción de integridad referencial en forma general expresa que una fila de una tabla **A**, que hace referencia a otra tabla **B**, debe hacerlo refiriéndose a una fila existente de esa tabla.

Por ejemplo, si la tabla **CLIENTE** tiene una clave foránea (**FOREIGN KEY**) CódRepre correspondiente a la clave primaria **(PRIMARY KEY)** CódigoVend de la tabla **VENDEDOR**, todo valor de dicha clave foránea debe concordar con un valor de la clave primaria de **VENDEDOR**. En términos generales, una clave foránea enlaza cada fila de la tabla hijo que contiene la clave foránea, con la correspondiente fila de la tabla padre que contiene la clave primaria.

Las columnas *CódigoVend* y *CódRepre* crea una relación padre/hijo entre las tablas **VENDEDOR** y **CLIENTE**. Cada fila de la tabla **VENDEDOR** (padre) tiene según el caso, cero o más filas **CLIENTE** (hijo), con número de vendedor coincidente. Análogamente cada fila **CLIENTE** (hijo), tiene exactamente una fila **VENDEDOR** (padre), con un número de vendedor coincidente. La Figura 5.21 muestra este ejemplo de integridad referencial. (Página 255)

En la siguiente instrucción se analiza la integridad referencial cuando se ingresa una nueva fila a la tabla **CLIENTE** con la instrucción **INSERT**:

```
INSERT INTO CLIENTE (Nombre, Apellido1, Apellido2, Cédula, Dirección,
LímCrédito, CédulaRec, CódRepre, FechaAsig)

VALUES (María, López, García, 1111222233, Amazonas 3-78, 1300, 0456456456,
7, 12-feb-15)
```

La fila que se desea insertar con esta instrucción rompe la relación padre/hijo entre las tablas **VENDEDOR** y **CLIENTE**, al no existir correspondencia entre el valor de la clave primaria *CódigoVend* y el

CLIENTE

Nombre	Apellido1	Apellido2	Cédula	. . .	CódRepre	FechaAsig
Víctor	Castro	Torres	0102030405	. . .	1	13-oct-12
Juan	Polo	Ávila	0122334455	. . .	1	02-mar-13
Elena	Tapia	Barrera	0111555666	. . .	Null	04-jun-12
Pablo	Brito	Parra	0123456789	. . .	3	15-dic-11
Marcia	Mora	Durazno	0987654321	. . .	1	23-dic-12
Jaime	Pérez	Guerrero	0777888999	. . .	2	02-oct-12
Humberto	Pons	Coronel	0456456456	. . .	3	10-sep-13
Alberto	Torres	Viteri	0999998888	. . .	4	13-ene-13
Manuel	Castro	León	0123123123	. . .	2	12-feb-12

Clave foránea

Referencia

VENDEDOR

Clave primaria

Nombre	Apellido	CódigoVend	. . .	Teléfono
Víctor	Castro	Torres	. . .	2472713
Juan	Polo	Ávila	. . .	2876549
Elena	Tapia	Barrera	. . .	Null
Pablo	Brito	Parra	. . .	2887643
Marcia	Mora	Durazno	. . .	4657890

Figura 5.21. Referencia entre clave primaria y clave foránea.

de la clave foránea *CódRepre*, debido a que se está intentando ingresar el valor de 7 como clave foránea en la tabla `CLIENTE`, sin que exista su correspondiente en la tabla `VENDEDOR`. El SGBD controla este quebrantamiento de la integridad referencial, impidiendo el ingreso de la nueva fila, con lo que se garantiza la relación padre/hijo entre la clave primaria y foránea.

La restricción de integridad referencial puede vulnerarse cuando se insertan o eliminan filas o cuando se modifica el valor de una columna de la clave foránea o de la clave primaria.

En una base de datos se presentan cuatro tipos de actualizaciones que afectan a la integridad referencial de las relaciones padre/hijo.

Cuando se i**nserta una nueva fila en la tabla hijo**, el valor de la clave foránea debe coincidir con uno de los valores de la clave primaria de la tabla padre. Si el valor no coincide, se está infringiendo la integridad referencial. Por ejemplo, si se ingresa una nueva fila en la tabla `CARGAFAMILIAR`, el valor de la clave foránea *CódVendedor*, debe corresponder a uno de los valores de la clave primaria *CódigoVend* de la tabla `VENDEDOR`. El insertar una nueva fila en la tabla padres no presenta problema; simplemente se crea un padre sin hijos.

Si se **actualiza una clave foránea de la tabla hijo**, el nuevo valor debe corresponder con uno de los valores de la clave primaria de la tabla padre, caso contrario, la fila que fue actualizada quedará sin padres.

Si mediante la instrucción `DELETE`, se s**uprime una fila de la tabla padre**, referida a una o varias filas de la tabla hijo, estas filas quedarán huérfanas sin padre. Por el contrario, si se suprime una fila de la tabla hijo, no representa problema puesto que, el padre de esta fila tendrá un hijo menos. Un análisis similar se podría considerar para el caso en el que se **actualiza una clave primaria** de una tabla padre; sin embargo, por lo general los valores de una clave primaria casi nunca se modifican.

Si renuncia el vendedor Diego Loja, se tiene que eliminar (`DELETE`) la fila correspondiente (se suprime una fila padre) de la tabla `VENDEDOR`. Surge el interrogante, ¿qué sucedería con sus tres clientes a quienes representa? (filas hijo). Según el caso y básicamente dependiendo de las restricciones del mundo real, se podrían considerar las siguientes opciones:

- Impedir que el vendedor se elimine hasta reasignar a los clientes.
- Suprimir automáticamente a los tres clientes asignados.
- A los tres clientes colocar en la columna *CódRepre* el valor de `NULL`.
- A los tres clientes colocar en la columna *CódRepre* un valor por defecto.

Iguales opciones se presentan cuando se intenta actualizar (`UPDATE`) una clave primaria.

En el supuesto de que se quiera alterar la integridad referencial se presentan dos alternativas de ejecución; en la primera el SGBD toma la acción de rechazar la operación y en la segunda el diseñador de la base de datos puede especificar una de las siguientes opciones: `CASCADE`, `SET NULL` y `SET DEFAULT`, denominadas **acción de activación referencial** a cualquier restricción de la clave foránea (Elmasri R., Navathe S., 2016).

Operación RESTRICT: Borrar una fila o modificar la clave primaria de una tabla que está referida en otra tabla sólo se permite si no existen filas con dicha clave en la tabla que contiene la clave foránea. La operación RESTRICT impide suprimir una fila o actualizar una clave primaria de la tabla padre, si esa fila tiene hijos. Por ejemplo, para borrar un vendedor, no tendría que haber ninguna carga familiar asociada a dicho vendedor. El estándar SQL2 a la operación RESTRICT la denomina NO ACTION.

Operación CASCADE: El borrar o modificar una fila de la tabla que contiene la clave primaria referida, lleva consigo el borrado o modificado automático en cascada de las filas de la tabla que contiene la clave foránea. La opción CASCADE especifica que cuando una fila en la tabla padre es borrada o modificada, todas las filas referidas en la tabla hijo son también borradas o modificadas automáticamente. Por ejemplo, al modificar la clave primaria *CódigoVend* de la tabla VENDEDOR, se modifica automáticamente la clave foránea *CódVendedor* de todas las cargas familiares de ese vendedor. Para el caso de eliminación de un vendedor, se elimina automáticamente todas las cargas familiares de ese vendedor.

Operación SET NULL: Si se borra una fila o se modifica el valor de una clave primaria de una tabla padre, los valores de la clave foránea de todas las filas hijo deben automáticamente pasar a NULL. Por ejemplo, cuando se borra un vendedor, a los clientes asignados como representantes de ese vendedor, se les asignará un vendedor desconocido (NULL).

Operación SET DEFAULT: Especifica que, cuando la fila padre sea suprimida o el valor de la clave primaria de una fila padre sea modificado, el valor de la clave foránea en todas las filas hijo, deben automáticamente pasar al valor por defecto para esa columna particular. Por ejemplo, si se modifica la clave primaria de un vendedor o se elimina un vendedor, automáticamente cambia la clave foránea de las filas de los clientes representados, por el valor de defecto especificado en la definición de la tabla.

Si no se especifica ninguna opción, el valor por omisión es RESTRICT.

5.5 DEFINICIÓN DE DATOS Y TIPOS DE DATOS

En el siguiente apartado se analizarán los tipos básicos de datos disponibles y la definición de un esquema.

5.5.1 Tipos de datos

SQL especifica diferentes tipos de datos que pueden almacenarse en una base de datos; entre ellos están los de tipo: caracteres, numéricos, cadena de bits, booleano, de fecha y de hora.

- El tipo de dato cadena de caracteres puede ser de longitud fija `CHAR(n)` o `CHARACTER(n)`, donde n es la longitud definida por el usuario. El tipo `VARCHAR(n)` permite almacenar cadenas de caracteres de longitud variable con un tamaño máximo de (n) especificado por el usuario. Su forma equivalente completa es `CHARACTER VARYING`.

  Por ejemplo, si a una columna A se le asigna un tamaño fijo de 10 caracteres `CHAR(10)` y se almacena la cadena ´Juan´, en este caso, se añade seis espacios en blanco para completar el tamaño especificado de 10 caracteres. Por otra parte, si a la columna `B` se le especifica un tamaño variable de 10 caracteres `VARCHAR(10)`, y se registra el valor de ´Juan´, a diferencia del caso anterior no se añaden espacios extras.

  Algunos gestores de bases de datos tienen la posibilidad de almacenar en una columna, largas cadenas de texto (entre 32.000 a 65.000 caracteres o más), lo que permite a la base de datos el manejo de documentos completos, resúmenes, descripciones, etcétera.

- Los tipos de datos numéricos pueden ser enteros `INT` o en su forma completa `INTEGER`, que generalmente son utilizados para el registro de números de facturas, contadores, edades, etcétera. Otro tipo de dato numérico corresponde al de coma fija, cuya precisión la especifica el usuario, su formato es `NUMERIC(p,d)` donde `p` representa el número de dígitos (más el signo), y de estos `p` dígitos, `d` representa la parte decimal.

  Incluyen también en este tipo de datos los números en coma flotante `FLOAT(n)` cuya precisión es al menos de **n** dígitos. Los números en coma flotante de baja precisión están representados por `REAL` y los números en coma flotante de alta precisión por `DOUBLE PRECISION`.

  SQL permite realizar operaciones aritméticas y de comparación sobre todos los tipos de datos numéricos que se han mencionado.

- El tipo de datos booleano almacena valores lógicos `TRUE` o `FALSE`, a menos que lo prohíba una restricción `NOT NULL`. Los datos booleanos utilizan una lógica de un tercer valor de verdad `UNKNOWN` como valor `NULL`. Todos los valores de tipo booleanos son comparables. El valor `TRUE` es mayor que el valor `FALSE`.

- El tipo de datos bit se usa para definir secuencias de dígitos binarios (bits), cada uno de los cuales puede tener valores 0 y 1, son de longitud fija `BIT(n)` o longitud variable `BIT VARYING(n)`.

- SQL también proporciona tipos de datos relacionados con fecha y hora. `DATE` que contiene el año con cuatro dígitos, el mes y el día. `TIME` que almacena la hora del día, en horas minutos y segundos. `WITH TIMEZONE` permite almacenar la información sobre el huso horario.

Un tipo de datos `TIMESTAMP` combina los campos de `DATE` y `TIME`, tiene su variante `TIMESTAMP`(p) que se usa para especificar el número de cifras decimales para los segundos. SQL tiene diferentes funciones útiles: con `CURRENT_DATE` se obtiene la fecha actual y `CURRENT_TIME` devuelve la hora actual. La función `INTERVAL` permite realizar cálculos basados en fechas, hora e intervalos.

5.5.2 Definición del esquema (`CREATE TABLE`)

El **esquema SQL** define la estructura de una base de datos, sus tablas, las relaciones, los dominios, las vistas y las restricciones; adicionalmente especifica la información relativa a seguridades y autorizaciones de cada tabla y la estructura de almacenamiento de cada una de éstas en el disco. El esquema se especifica mediante un lenguaje especial denominado **lenguaje de definición de datos**, o DDL (*Data Definition Language*, por sus siglas en inglés).

El comando `CREATE TABLE` se utiliza para definir una nueva tabla vacía en la base de datos, asignándole un nombre, sus columnas y las restricciones, preparándole para recibir datos mediante la instrucción `INSERT`.

En el cuerpo del comando `CREATE TABLE`, se especifica las columnas en una lista separada por comas y encerrada entre paréntesis. A cada definición de una columna se le asigna un **nombre** que debe ser único, un **tipo de dato** que especifica el dominio de valores que serán almacenados en la columna, acompañado de la longitud o posición de los decimales dependiendo del tipo. Se incluye también las restricciones como `NOT NULL`, restricciones de identidad, de clave, integridad de entidad e integridad referencial. Para ilustrar el uso del comando `CREATE TABLE`, a continuación se definen las siete tablas del esquema de la Figura 2.43(a) de nuestro caso de estudio **La Ferretería.**

La siguiente instrucción crea la tabla `VENDEDOR` formada por siete columnas que se pueden especificar de la siguiente manera: *Vnombre* y *Apellido*. Son cadenas de caracteres de longitud variables máximo de 20, *CódigoVend* definido como número entero, *Sexo* se define como una cadena de longitud fija de 1, *FechaContrato* especificada como un dato tipo fecha, *ObjVentas* es un dato numérico con un total de 8 dígitos, de los cuales 2 corresponden a los decimales, y por último la columna *Teléfono* definida como una cadena de longitud fija de 10 caracteres.

La clave primaria de la tabla se especifica como `PRIMARY KEY` y corresponde a la columna *CódigoVend*, que de acuerdo con las restricciones de integridad, los valores tienen que ser no nulos y únicos, en otras palabras, no se aceptan valores `NULL` y ningún par de filas de la tabla puede tener valores iguales en la columna de la clave primaria.

La restricción **NOT NULL** en *Vnombre*, *Apellido*, *Sexo* y *FechaContrato* garantiza que no se registren valores nulos para estas columnas, de modo que, la restricción excluye el valor nulo del dominio del atributo.

Por otro lado la restricción del chequeo de validez **CHECK** (ObjVentas = > 2000) garantiza que los valores del objetivo de ventas no sean menores a 2.000 dólares.

En función de este análisis, la tabla **VENDEDOR** se define de la siguiente manera:

```
CREATE TABLE VENDEDOR
(Vnombre          VARCHAR (20)      NOT NULL,
Apellido          VARCHAR (20)      NOT NULL,
CódigoVend        INT,
Sexo              CHAR (1)          NOT NULL,
FechaContrato     DATE              NOT NULL,
ObjVentas         NUMERIC (8,2) CHECK (ObjVentas = > 2000),
Teléfono          CHAR(10),
PRIMARY KEY       (CódigoVend))
```

En la siguiente instrucción se crea la tabla **CARGAFAMILIAR**, cuya integridad referencial se especifica mediante la cláusula **FOREIGN KEY** en la columna *CódVendedor*, que guarda correspondencia con los valores de la clave primaria *CódigoVend* de la tabla **VENDEDOR**.

Adicionalmente en la definición de la **FOREIGN KEY** se determina la acción que hay que realizar en caso de que se elimine una clave primaria. Para este ejemplo se considera que si se elimina un vendedor, sus cargas familiares también serán eliminadas.

En las tablas **CLIENTE**, **FACTURA**, **DETALLE** y **CLIENTE_TELÉF**, que se definen a continuación, también se muestran restricciones de clave foránea.

```
CREATE TABLE CARGAFAMILIAR
(NomDep          VARCHAR (20)      NOT NULL,
Sexo             CHAR(1) CHECK (Sexo in(´M´,´F´)),
FechaNac         DATE              NOT NULL,
Relación         VARCHAR(10) CHECK (Relación IN   (´Hija´,´Hijo´,´Cónyuge´)),
CódVendedor      INT,
FOREIGN KEY      (CódVendedor) REFERENCES VENDEDOR
                 ON DELETE CASCADE))
```

En la definición de la tabla `CLIENTE`, que se detalla a continuación, se incluye `NULL` como valor predeterminado para el límite de crédito del cliente, especificado mediante la cláusula `DEFAULT NULL`.

Adicionalmente si a cada uno de los nuevos clientes de La Ferretería se les asigna como representante de manera predeterminada a la vendedora ´Maricela Vera´ cuyo código es 5. Para especificar esta restricción, en la declaración de la columna *CódRepre* se incluye la cláusula `DEFAULT` 5.

```
CREATE TABLE CLIENTE
(Nombre          VARCHAR(20)       NOT NULL,
Apellido1        VARCHAR(20)       NOT NULL,
Apellido2        VARCHAR(20)       NOT NULL,
Cédula           CHAR(10),
Dirección        VARCHAR(30),
LímCrédito       NUMERIC(8,2)      DEFAULT NULL,
CédulaReco       CHAR(10),
CódRepre         INT,
FechaAsig        DATE              NOT NULL,
PRIMARY KEY      (Cédula),
FOREIGN KEY      (CédulaReco) REFERENCES CLIENTE
```

```
                ON DELETE SET NULL,
FOREIGN KEY     (CódRepre) REFERENCES VENDEDOR
                ON DELETE SET NULL)

CREATE TABLE ARTÍCULO
(Código         INT,
Descripción     VARCHAR(30)      NOT NULL,
Precio          NUMERIC (8,2)    NOT NULL,
Unidad          VARCHAR(20)      NOT NULL,
ExiMáx          INT              NOT NULL,
ExiMín          INT              NOT NULL,
PRIMARY KEY     (Código))

CREATE TABLE FACTURA
(Número         INT,
Fecha           DATE       NOT NULL,
CódVendedor     INT,
CcCliente       CHAR(10),
PRIMARY KEY     (Número),
FOREIGN KEY     (CódVendedor) REFERENCES VENDEDOR
                ON DELETE SET NULL,
FOREIGN KEY     (CcCliente) REFERENCES CLIENTE
                ON DELETE RESTRICT)
```

```
CREATE TABLE DETALLE
(FactNúmero       INT              NOT NULL,
ArtCódigo         INT              NOT NULL,
Cantidad          Numeric (7,2) CHECK (Cantidad > 0),
FOREIGN KEY       (FactNúmero) REFERENCES FACTURA
                  ON DELETE RESTRICT),
FOREIGN KEY       (ArtCódigo) REFERENCES ARTÍCULO
                  ON DELETE RESTRICT))

CREATE TABLE CLIENTE_TELEF
CédulaCli         CHAR(10),
NúmeroTelef       CHAR(10),
FOREIGN KEY       (CédulaCli) REFERENCES CLIENTE
                  ON DELETE CASCADE))
```

En las instrucciones anteriores hay algunas claves foráneas (`FOREIGN KEY`) que pueden generar errores, porque ya sea que se especifique a través de referencias circulares como el caso de la relación `RECOMIENDA` de la tabla `CLIENTE`, o porque se refiere a una tabla que no existe aún, como el caso de la `FOREIGN KEY` *CódRepre* de la tabla `CLIENTE` que se refiere a la tabla `VENDEDOR` que todavía no se ha creado. El SGBD no acepta la instrucción `CREATE TABLE` y genera un error indicando que la definición de tabla hace referencia a una tabla que aún no existe. Para solucionar este problema se debe crear la primera tabla, omitiendo la restricción de clave foránea, y añadirse más tarde usando la instrucción `ALTER TABLE` que se verá más adelante.

5.5.3 Eliminar una tabla (`DROP TABLE`)

Si se considera que una tabla no es necesaria en la base de datos relacional se puede eliminar utilizando el comando `DROP TABLE`. Por ejemplo, para eliminar la tabla que almacena los datos de las cargas familiares, la instrucción se escribe como:

```
DROP TABLE CARGAFAMILIAR
```

Una vez ejecutada la instrucción `DROP TABLE` se pierde la definición de la tabla y todo su contenido, sin que exista una manera de recuperar los datos. Adicionalmente no se puede volver a insertar una fila, a no ser que se vuelva a crear la tabla con el comando `CREATE TABLE`.

5.5.4 Modificar la definición de una tabla (`ALTER TABLE`)

SQL permite modificar la estructura de una tabla después de haberla creado mediante el comando `ALTER TABLE`. Las posibles opciones de modificación incluyen la adición o eliminación de una columna, la eliminación de una restricción o adición de una nueva, la asignación o eliminación de un valor predeterminado a una columna. Por ejemplo, se requiere añadir a la tabla `VENDEDOR` una nueva columna que registre el salario de los vendedores, la instrucción se escribe de la siguiente manera:

```
ALTER TABLE VENDEDOR ADD Salario NUMERIC (10,2)
```

A la nueva columna el SGBD le asigna un valor `NULL` en todas las filas ya existentes de la tabla, razón por la cual, la restricción `NOT NULL` no está permitida. Para introducir un valor en la columna se puede especificar un valor predeterminado o utilizar la instrucción `UPDATE`.

5.6 CUESTIONARIO Y EJERCICIOS

Preguntas de repaso

1. ¿En qué se diferencia la cláusula `WHERE` de la cláusula `HAVING`?

2. ¿Cuáles son las restricciones que deben considerarse para ejecutar las operaciones de conjuntos `UNION`, `INTERSECT` y `EXCEPT`?

3. Explique el término integridad de entidad.

4. Explique cómo se tratan los valores `NULL` cuando se aplican funciones de agregación en una consulta.

5. Explique mediante un ejemplo el concepto de integridad referencial.

6. Explique cada una de las cláusulas de la instrucción `CREATE TABLE`.

Ejercicios resueltos

Las consultas que se presentan a continuación se basan en la instancia de la base de datos del caso La Ferretería que se detalla en la Figura 2.43(b).

7. Recuperar el nombre y apellido de los vendedores de sexo masculino que tienen un objetivo de ventas mayor a 2.500 dólares.

```
Select      Vnombre, Apellido
From        VENDEDOR
Where       Sexo = "M" and
            ObjVentas > 2500
```

8. Al precio de todos los artículos incrementar el 15%.

```
Select      Código, Descripción, (precio*1.15) AS Incremento, Unidad,
            ExiMáx, ExiMín
From        ARTÍCULO
```

9. Seleccionar los clientes que tienen como representante al vendedor de código 1.

```
Select      Nombre, Apellido1
From        CLIENTE
Where       CódRepre = "1"
```

10 Seleccionar el nombre y los apellidos de los clientes que tienen como representante al vendedor Diego Loja.

```
Select      Nombre, Apellido1, Apellido2
From        VENDEDOR, CLIENTE
Where       Vnombre = "Diego" and
            Apellido = "Loja" and
            CódigoVend = CódRepre
```

11. Enlistar la cédula de identidad y el nombre de los clientes que tienen un límite de crédito mayor a 1.000 dólares.

```
Select      Cédula, Nombre
From        CLIENTE
Where       LímCrédito > 1000
```

12. Seleccionar el nombre y el primer apellido de los clientes que tienen como representante al vendedor Diego Loja y tienen un límite de crédito mayor o igual a 1.000 dólares.

```
Select      Nombre, Apellido1
From        VENDEDOR, CLIENTE
Where       Vnombre = "Diego" and
            Apellido = "Loja" and
            CódigoVend = CódRepre and
            LímCrédito ≥ 1000
```

13. Para todos los vendedores de sexo masculino, recuperar el nombre y apellido de los clientes a quienes representan y tienen un límite de crédito mayor a 1.000 dólares.

```
Select      Nombre, Apellido1, Apellido2
From        VENDEDOR, CLIENTE
Where       Sexo = "M" and    CódigoVend = CódRepre and
            LímCrédito > 1000
```

14 Para cada cliente, recuperar su nombre y primer apellido, el nombre y primer apellido del cliente que lo recomendó.

```
Select      C. Nombre, C. Apellido1, R. Nombre, R. Apellido1
From        CLIENTE C, CLIENTE R
Where       C.CédulaReco = R.Cédula
```

15. Enlistar el número de cada una de las facturas de compra realizadas por el cliente Pablo Brito.

```
Select      Número
From        CLIENTE, FACTURA
Where       Cédula = CcCliente and Nombre = ”Pablo” and
            Apellido1 = “Brito”
```

16. Determinar cuántas facturas tiene el cliente Pablo Brito.

```
Select      count(CcCliente)
From        CLIENTE, FACTURA
Where       Nombre = “Pablo” and Apellido1 = “Brito” and
            Cédula = CcCliente
```

17. Enlistar el nombre de todos los artículos comprados con las diferentes facturas por el cliente Manuel Bonilla.

```
Select      Distinct (Descripción)
From        FACTURA, DETALLE, ARTÍCULO
Where       Nombre = “Manuel” and Apellido1 = “Bonilla” and
            Cédula = CcCliente and Número = FactNúmero and
            ArtCódigo = Código
```

18. Enlistar el nombre de los clientes a quienes facturó el vendedor Arturo Salto.

```
Select      Nombre
From        VENDEDOR, FACTURA, CLIENTE
Where       Vnombre = “Arturo” and Apellido = “Salto” and
            CódigoVend = CódVendedor and CcCliente = Cédula
```

19. Recuperar el nombre de las cargas familiares del vendedor Diego Loja.

```
Select      NomDep
From        VENDEDOR, CARGAFAMILIAR
Where       Vnombre = "Diego" and Apellido = "Loja" and
            CódigoVend = CódVendedor
```

20. Recuperar el nombre y el apellido de los vendedores que tienen cargas familiares.

```
Select      Distinct Vnombre, Apellido
From        VENDEDOR, CARGAFAMILIAR
Where       CódigoVend = CódVendedor
```

21. Para cada vendedor enlistar el nombre y el apellido, y el número de sus cargas familiares.

```
Select      Vnombre, Apellido, count(CódVendedor)
From        VENDEDOR, CARGAFAMILIAR
Where       CódigoVend = CódVendedor
Group by    CódVendedor
```

22. Enlistar el nombre y el apellido de los vendedores que tienen más de una carga familiar.

```
Select      Vnombre, Apellido, count(CódVendedor)
From        VENDEDOR, CARGAFAMILIAR
Where       CódVendedor = CódigoVend
Group by    CódVendedor
Having      count(CódVendedor) > 1
```

23. Enlistar el nombre, el apellido y el sexo del vendedor e incluir el nombre y sexo de su carga familiar.

```
Select      V.Vnombre, V.Apellido, V.Sexo, C.NomDep, C.Sexo
From        VENDEDOR V, CARGAFAMILIAR C
Where       C.CódVendedor = V.CódigoVend
```

24. Recuperar el nombre de las cargas familiares ordenadas alfabéticamente, primero las de sexo femenino, seguidas por las de sexo masculino.

```
Select      NomDep, Sexo
From        CARGAFAMILIAR
Order by    Sexo, NomDep
```

25. Recuperar el nombre y el apellido de los vendedores que tienen cargas familiares con el mismo primer nombre y el mismo sexo.

```
Select      Vnombre, Apellido
From        VENDEDOR V, CARGAFAMILIAR C
Where       CódVendedor = CódigoVend and Vnombre = NomDep and
            V.Sexo = C.Sexo
```

26. Seleccionar el nombre y el apellido de los vendedores que tienen únicamente cargas familiares de sexo femenino.

```
Select      Distinct Vnombre, Apellido
From        CARGAFAMILIAR, VENDEDOR
Where       CódVendedor = CódigoVend and
            CódigoVend not in (Select CódVendedor
                               From CARGAFAMILIAR
                               Where Sexo = "M")
```

27. Recuperar el nombre y el apellido de los vendedores que no tienen cargas familiares.

```
Select      Vnombre, Apellido
From        VENDEDOR
Where       CódigoVend not in (Select CódVendedor
                               From CARGAFAMILIAR)
```

28 Recuperar el nombre y el apellido de los vendedores que tienen al menos una carga familiar mujer.

```
Select      Vnombre, Apellido
From        VENDEDOR
Where       CódigoVend in (Select CódVendedor
                           From CARGAFAMILIAR
                           Where Sexo = "F")
```

29. Enlistar el nombre y el apellido de los vendedores que tienen como carga familiar a su esposa.

```
Select      Vnombre, Apellido
From        VENDEDOR
Where       CódigoVend in (Select CódVendedor
                           From CARGAFAMILIAR
                           Where Sexo = "F" and
                           Relación = "Cónyuge")
```

30. Enlistar el nombre de los vendedores que suscribieron el contrato con fecha posterior al 1 de enero de 2008.

```
Select      Vnombre, Apellido
From        VENDEDOR
Where       FechaContrato >= "2008-01-01"
```

31. Recuperar el nombre, el apellido y el número de facturas realizadas por cada vendedor.

```
Select          Vnombre, Apellido, count(Número)
From            FACTURA, VENDEDOR
Where           CódVendedor = CódigoVend
group by        CódVendedor
```

32. Determinar cuántos artículos diferentes se vendieron en la factura número 100.

```
Select          count(ArtCódigo)
From            DETALLE
where           FactNúmero = 100
```

33. Para cada uno de los artículos vendidos en la factura número 100, recuperar la descripción y el monto total.

```
Select          Descripción, (Cantidad * Precio) AS MontoTotal
From            DETALLE, ARTÍCULO
Where           ArtCódigo = Código and FactNúmero = 100
```

34. Determinar el monto total de la factura número 100.

```
Select          FactNúmero, sum(Cantidad * Precio) AS MontoTotal
From            DETALLE, ARTÍCULO
Where           ArtCódigo = Código and FactNúmero = 100
```

35. Para todos los artículos vendidos, recuperar el código, la descripción y las cantidades vendidas.

```
Select          Código, Descripción, sum(Cantidad)
From            DETALLE, ARTÍCULO
Where           ArtCódigo = Código
Group by        Código
```

36. Enlistar los diferentes límites de crédito de los clientes.

```
Select Distinct LímCrédito
From CLIENTE
```

37. Recuperar el nombre, la cédula de identidad de los clientes y su monto total de compras realizadas en La Ferretería.

```
Select      Nombre, Cédula, sum(Cantidad * Precio) AS Total
From        CLIENTE, FACTURA, DETALLE, ARTÍCULO
Where       Cédula = CcCliente and Número = FactNúmero and
            ArtCódigo = Código
Group by    Cédula
```

38. Determinar el nombre del vendedor que más ha vendido.

```
Select      Vnombre, Apellido, sum(Cantidad * Precio) AS Total
From        FACTURA, DETALLE, ARTÍCULO, VENDEDOR
Where       Número = FactNúmero and ArtCódigo = Código
            and CódVendedor = CódigoVend
Group by    CódVendedor
Having      Total >= all (Select sum(Cantidad * Precio)
                          From FACTURA, DETALLE, ARTÍCULO
                          Where Número = FactNúmero and
                          ArtCódigo = Código
                          Group by CódVendedor)
```

39. Recuperar el número total de facturas realizadas entre el 1 y 28 de febrero del 2014.

```
Select      count(Número)
From F      ACTURA
Where       Fecha between “2014-02-01” and “2014-02-28”
```

40. Recuperar el monto total de cada factura y ordenarlas de menor a mayor.

```
Select      Número, sum(Cantidad * Precio) AS Monto
From        FACTURA, DETALLE, ARTÍCULO
Where       Número = FactNúmero and ArtCódigo = Código
Group by    Número
Order by    Monto
```

41. Recuperar el monto total de cada una de las facturas del cliente Víctor Castro.

```
Select      Número, sum(Cantidad * Precio) AS Monto
From        FACTURA, DETALLE, ARTÍCULO, CLIENTE
Where       Nombre = "Víctor" and Apellido1 = "Castro" and
            Cédula =  CcCliente  and Número = FactNúmero and
            ArtCódigo = Código
Group       by Número
```

42. Recuperar el nombre de los clientes que compraron cemento y el nombre de quién los vendió.

```
Select      Nombre as Cliente, Vnombre as Vendedor
From        ARTÍCULO, DETALLE, FACTURA, CLIENTE, VENDEDOR
Where       Descripción= "Cemento"  and
            Código = ArtCódigo and
            FactNúmero = Número and
            CcCliente = Cédula and
CódVendedor = CódigoVend
```

43. Recuperar el nombre y apellido1 de los clientes que tienen un límite de crédito mayor a 500 dólares y han comprado más de dos facturas.

```
Select Nombre, Apellido1
From CLIENTE
Where LímCrédito > 500 and Cédula in (Select CcCliente
                                      From FACTURA
                                      Group by CcCliente
                                      Having count(CcCliente) > 2)
```

44. Recuperar el nombre y la cédula de los clientes que al menos en una de sus facturas, han comprado un monto que supera el 10% de su límite de crédito.

```
Select      Nombre, Apellido1, Número
From        CLIENTE, FACTURA
Where       Cédula = CcCliente and
            Número in (Select FactNúmero
                       From DETALLE, ARTÍCULO
                       Where Código = ArtCódigo
                       Group by FactNúmero
                       Having sum(Precio * Cantidad) > (LímCrédito * 0.10))
```

45. Determinar el nombre del vendedor que más facturas ha realizado.

```
Select      Vnombre, count(CódVendedor) as Vend
From        VENDEDOR, FACTURA
Where       CódigoVend = CódVendedor
Group by    CódVendedor
Having      Vend >= ALL (SELECT count(CódVendedor)
                         From FACTURA
                         Group by CódVendedor)
```

46. Para la factura con el mayor monto de ventas, enlistar el nombre y apellido del cliente y el nombre y apellido del vendedor.

```
Select      Vnombre, Apellido, Nombre, Apellido1,
            sum(Cantidad * Precio) AS MontoTotal
From        FACTURA, VENDEDOR, CLIENTE, DETALLE, ARTÍCULO
Where       Cédula=CcCliente and CódVendedor=CódigoVend and
            Número = FactNúmero and ArtCódigo = Código
Group by    Número
Having      MontoTotal >= ALL (Select sum(Cantidad * Precio)
      From FACTURA, VENDEDOR, CLIENTE, DETALLE, ARTÍCULO
      Where       Cédula=CcCliente and
                  CódVendedor = CódigoVend and
                  Número = FactNúmero and
                  ArtCódigo = Código
                  Group by Número)
```

47. Enlistar el nombre y el apellido1 de los clientes que tienen un límite de crédito mayor a 1.000 dólares o tienen como representante a la vendedora Lucía Serrano.

```
Alternativa 1

Select      Nombre, Apellido1
From        CLIENTE, VENDEDOR
Where       CódRepre = CódigoVend AND
            Vnombre = “Lucía” and Apellido = “Serrano” or
            LímCrédito > 1000
```

Alternativa 2 (uso de la operación `UNION`).

```
Select      Nombre, Apellido1
From        CLIENTE
Where       LímCrédito > 1000
UNION
Select      Nombre, Apellido1
From        CLIENTE, VENDEDOR
WHERE       Vnombre = “Lucía” and Apellido = “Serrano” and
            CódRepre = CódigoVend
```

48. Enlistar el número de cédula de los clientes recomendados directamente por el cliente Humberto Pons.

```
SELECT      Cédula
FROM        CLIENTE
WHERE       CédulaReco = (SELECT Cédula
                          FROM CLIENTE
                          WHERE  Nombre = “Humberto” and
                                 Apellido1 = “Pons”)
```

49.Enlistar el nombre de los clientes que tienen como representante a los vendedores de código 1, 3 o 4.

Alternativa 1

```
Select      Nombre, Apellido1
From        CLIENTE, VENDEDOR
Where       CódRepre = CódigoVend and
            CódigoVend in (1,3,4)
```

Altenativa 2

```
Select      Nombre, Apellido1
From        CLIENTE join VENDEDOR on CódRepre = CódigoVend
Where       CódigoVend in (1,3,4)
```

50. Enlistar el nombre, el apellido y la cédula de ciudadanía de todos los clientes cuyo primer apellido están comprendidos entre los apellidos Mora y Tapia.

```
Select      Nombre, Apellido1, Apellido2, Cédula
From        CLIENTE
Where       Apellido1 BETWEEN 'Mora' and 'Tapia'
```

51. Para cada factura, recuperar el número, el nombre y apellido del cliente y del vendedor, el monto total; ordenados por el primer apellido del cliente.

```
Select      Número, Nombre, Apellido1, Vnombre, Apellido,
            sum(Cantidad*Precio) as MontoTotal
From        FACTURA, ARTÍCULO, DETALLE, CLIENTE, VENDEDOR
Where       Cédula = CcCliente and
            CódVendedor = CódigoVend and
            FactNúmero = Número and Código = ArtCódigo
Group by Número
Order By Apellido1
```

Número	Nombre	Apellido1	Vnombre	Apellido	MontoTotal
109	Manuel	Bonilla	Antonio	Calle	20
111	Manuel	Bonilla	Diego	Loja	114
108	Manuel	Bonilla	Lucía	Serrano	84
102	Pablo	Brito	Diego	Loja	18
100	Víctor	Castro	Diego	Loja	46,5
104	Marcia	Mora	Antonio	Calle	63
106	Jaime	Pérez	Arturo	Salto	38,4
101	Juan	Polo	Lucía	Serrano	140
103	Juan	Polo	Arturo	Salto	122
110	Humberto	Pons	Arturo	Salto	6,6
105	Elena	Tapia	Antonio	Calle	30
107	Alberto	Torres	Lucía	Serrano	26,4

52. Para cada vendedor que tenga más de dos clientes a quien representar, recuperar el límite de crédito máximo y mínimo; el límite de crédito total y el número de clientes a quienes representa.

```
Select      Vnombre, count(Cédula) as Representados,
max(LímCrédito),min(LímCrédito),sum(LímCrédito)
From        CLIENTE join VENDEDOR on CódigoVend = CódRepre
Group       by CódRepre
Having count(Cédula) > 2
```

53. Recuperar el monto total de las compras realizadas por cada cliente.

```
Select      Nombre, Apellido1, Cédula, sum(Cantidad * Precio) as Total
From        CLIENTE, FACTURA, DETALLE, ARTÍCULO
Where       Cédula = CcCliente and Número = FactNúmero
            and ArtCódigo = Código
Group by Cédula
```

Nombre	Apellido1	Cédula	Total
Juan	Polo	122334455	262
Manuel	Bonilla	123123123	218
Pablo	Brito	123456789	18
Humberto	Pons	456456456	6,6
Jaime	Pérez	777888999	38,4
Marcia	Mora	987654321	63
Alberto	Torres	999998888	26,4

54. Determinar el número de límites de crédito diferentes de los clientes.

```
Select count(distinct(LímCrédito))
from CLIENTE
```

55. Para cada factura con más de dos artículos vendidos, recuperar el nombre del cliente y el nombre del vendedor.

```
Select      Número, count(ArtCódigo) as NúmArtículo, Nombre,
            Apellido1, Vnombre, Apellido
From        FACTURA, DETALLE, VENDEDOR, CLIENTE
Where       Número = FactNúmero and CódVendedor = CódigoVend
            and CcCliente = Cédula
Group by FactNúmero
Having count(ArtCódigo) > 2
```

56. Determinar cuántos clientes de La Ferretería fueron recomendados.

```
Alternativa 1

SELECT      count(CédulaReco)
FROM        CLIENTE
WHERE       CédulaReco is not null
```

Alternativa 2

```
SELECT      count(CédulaReco)
FROM        CLIENTE
```

57. Enlistar los nombres de los clientes que tienen un límite de crédito mayor al promedio del límite de crédito de los clientes representados por el vendedor Antonio Calle.

```
Select      Nombre, Apellido1, Apellido2
From        CLIENTE
Where       LímCrédito > (Select avg(LímCrédito)
                    From VENDEDOR, CLIENTE
                    Where Vnombre = "Antonio" and
                    Apellido = "Calle" and
                    CódigoVend = CódRepre)
```

58.Determinar el mayor límite de crédito promedio de los clientes agrupados por su representante.

```
Select      Avg (LímCrédito)
From        CLIENTE
Group by CódRepre
Having Avg (LímCrédito) ≥ ALL (Select Avg(LímCrédito)
                               From CLIENTE
                               Group by CódRepre)
```

59. Recuperar el nombre de los vendedores cuyos representados (todos) tienen un límite de crédito mayor a 1.000 dólares.

```
Select distinct Vnombre, Apellido
From  CLIENTE, VENDEDOR
Where CódRepre = CódigoVend and
      CódigoVend not in (Select CódRepre
                         From CLIENTE
                         Where LímCrédito < 1000)
```

Ejercicios propuestos

Para los siguientes ejercicios propuestos de consultas en SQL se plantean tres casos de estudio que fueron revisados en los capítulos anteriores:

- Se recomienda realizar las consultas en SQL del caso de estudio La Ferretería, de los ejercicios resueltos y propuestos en el Capítulo 3.

- Para los siguientes ejercicios considerar la base de datos del caso La Empresa que se detalla en el ejercicio resuelto número 16 del Capítulo 2.

60. Enlistar todos los números de proyecto que son controlados por el departamento en donde el empleado Tapia es jefe, o todos los proyectos en donde trabaja el empleado Tapia.

61. Recuperar los nombres de los empleados que trabajan en los proyectos controlados por el departamento de investigación.

62. Determinar el número total de horas trabajadas por los empleados en cada proyecto.

63. Enlistar el nombre de los departamentos en donde todos los empleados ganan más de 3.000 dólares.

64. Enlistar el nombre de los empleados cuyo salario es mayor al promedio de los salarios de todos los empleados.

65. Enlistar el nombre de los empleados que tienen un salario mayor al promedio de los salarios de los empleados del Departamento Administrativo.

■ Partiendo del esquema de la base de datos del caso Registro de calificaciones del ejercicio 14 del Capítulo 2, realizar las siguientes consultas en SQL.

66. Determinar el número de estudiantes que cursan la materia de Sistemas Expertos.

67. Calcular el promedio de las notas (aporte1, aporte2 y final) de cada una de las materias.

68. Calcule el promedio de notas (aporte1, aporte2 y final) de cada uno de los estudiantes.

69. Para que un estudiante apruebe una materia debe tener una nota total igual o mayor a 14 puntos. Se pide enlistar los nombres de los estudiantes, sus materias reprobadas y la nota total. La nota total es igual al promedio de los dos aportes más la nota del examen final.

70. Enlistar el total de materias aprobadas por el estudiante Barrera, que tiene un número de cédula de identidad 110775849

NORMALIZACIÓN

En los Capítulos 2 y 3 se revisaron las técnicas para el diseño de base de datos denominadas modelo Entidad - Relación y modelo relacional, cuyo objetivo principal es crear un esquema que represente de manera precisa los datos de una organización, considerando sus relaciones y restricciones. En este capítulo se explica parte de la teoría desarrollada para examinar las relaciones que existen entre los atributos, mediante una serie de pruebas (formas normales) que sirven para identificar el agrupamiento óptimo de estos atributos, y comprobar la validez y pertinencia de las relaciones, mediante un conjunto de técnicas de diseño de datos denominada normalización.

Los objetivos principales de este capítulo son abordar los conceptos fundamentales de la normalización como un proceso de análisis de relaciones que se revisarán en la Sección 6.1. En la Sección 6.2 se examinan los problemas potenciales asociados con la redundancia de datos que generan anomalías de actualización. En la Sección 6.3 se analizará el concepto de dependencia funcional que describe las relaciones entre uno o más atributos de una tabla. En la Sección 6.4 se describe el proceso de normalización y la identificación de las formas normales más comúnmente utilizadas, como son la primera forma normal (1FN), segunda forma normal (2FN) y tercera forma normal (3FN). Estas formas normales con excepción de la primera, están definidas a través de dependencias funcionales y su aplicación en una relación paso a paso, en donde cada paso corresponde a una forma normal, permiten obtener relaciones con requisitos cada vez más restringidos y menos sensibles a las anomalías de actualización. Por último en la Sección 6.5, sobre la base de una relación no normalizada se realiza un ejemplo práctico del proceso de normalización.

6.1 TEORÍA DE LA NORMALIZACIÓN

El objetivo del diseño de una base de datos es crear un esquema relacional que represente de manera precisa los datos, sus relaciones, restricciones, y que éstos sean pertinentes para la organización. Para cumplir con este propósito se puede utilizar una o más técnicas de modelado de diseño de base de datos, como las que se revisaron en los Capítulos 2 y 3. La técnica que se adopte probablemente esté definida por el tamaño y complejidad de la base de datos o por la preferencia o experiencia del diseñador. En síntesis, el esquema relacional es posible obtener de dos maneras:

- Realizar primero el diseño conceptual (modelo Entidad - Relación) obteniendo el esquema conceptual, para posteriormente mediante el mapeo, generar el esquema relacional.

- A partir de los requerimientos del mundo real, especificar directamente un conjunto de atributos, relaciones y restricciones que definan el esquema relacional.

Al diseñar una base de datos mediante el modelo relacional o con cualquiera de los otros modelos, es posible obtener diferentes esquemas relacionales, de los cuales, unos representarán el mundo real mejor que otros. La labor del diseñador es evaluar los esquemas para que éstos se encaminen a la calidad del diseño, determinando por qué un conjunto de atributos agrupados en una relación es mejor que otro.

Es importante conocer los problemas que se pueden generar a partir de un diseño inadecuado, tal es el caso, en las relaciones que no tienen una estructura eficaz o apropiada; la modificación de datos puede traer consigo consecuencias indeseables originadas por anomalías de actualización, que se pueden eliminar mediante un método formal de análisis denominado normalización, que permite examinar los errores y generar esquemas correctos.

Otro problema que se presenta cuando los diseños son inadecuados es la redundancia de datos. Uno de los objetivos del diseño de base de datos es agrupar los atributos en relaciones de tal manera que se minimice la redundancia de datos, con lo cual se obtiene ventajas al momento de almacenar los datos, reduciendo el espacio de almacenamiento, facilitando el acceso para la actualización por parte del usuario y reduciendo la presencia de incoherencias en los datos almacenados. En la siguiente sección se plantean ciertos lineamientos de diseño para los esquemas de relación.

6.2 CRITERIOS DE DISEÑO PARA EL ESQUEMA RELACIONAL

Elmasri (2007) propone cuatro directrices de calidad para el diseño de un esquema de relación:

1. **Significado o semántica de los atributos.** La interpretación de los valores de un atributo en una tupla recibe el nombre de semántica y sirve para identificar cómo se relacionan entre sí los atributos. Por ejemplo, en el esquema de la base de datos de nuestro caso de estudio La Ferretería que se muestra en la Figura 6.1. El significado de la relación `ARTÍCULO` es simple, cada una de las tuplas almacena información correspondiente a un artículo determinado, en la que se registra su código definido como clave primaria (*Código*), la manera como se representa o detalla el artículo (*Descripción*), el precio de venta (*Precio*), la unidad

de medida (*Unidad*) y las existencia máxima y mínima (*ExiMáx* y *ExiMín*). De manera similar la semántica de las otras relaciones es simple y clara. Por ejemplo, en la relación `CLIENTE` el atributo *CódRepre* es una clave foránea y significa la asociación entre un cliente y su vendedor como representante.

La relación `DETALLE` contiene dos claves foráneas, la primera *FacNúmero* que representa el número de la factura y la segunda *ArtCód* en la que se registra el código del artículo, adicionalmente tiene el atributo *Cantidad* que expresa el número de unidades de un artículo compradas en una factura. Este significado o semántica simple y directa de cada uno de los atributos representa una medida informal de lo bien que está diseñada la relación.

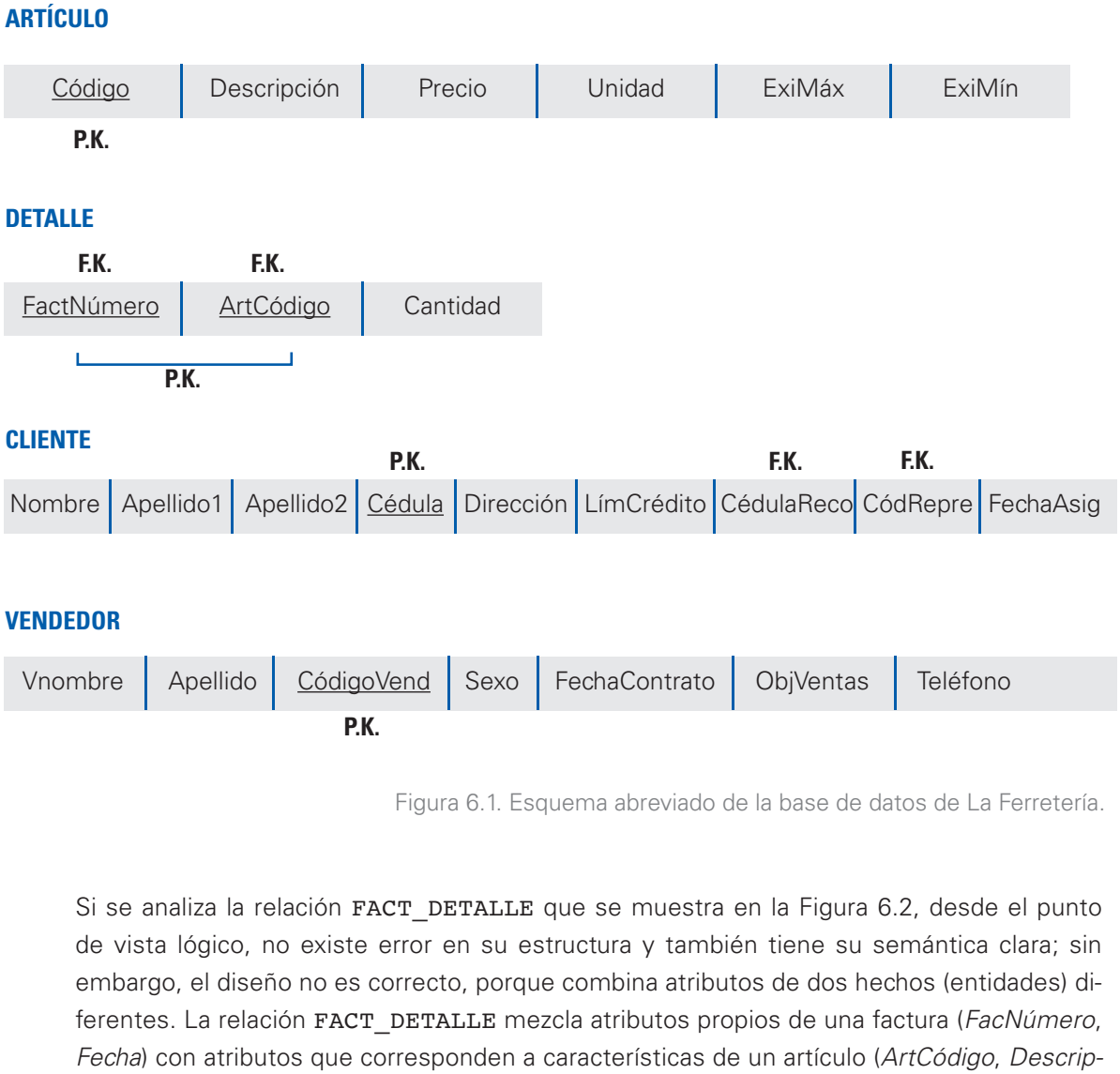


Figura 6.1. Esquema abreviado de la base de datos de La Ferretería.

Si se analiza la relación `FACT_DETALLE` que se muestra en la Figura 6.2, desde el punto de vista lógico, no existe error en su estructura y también tiene su semántica clara; sin embargo, el diseño no es correcto, porque combina atributos de dos hechos (entidades) diferentes. La relación `FACT_DETALLE` mezcla atributos propios de una factura (*FacNúmero*, *Fecha*) con atributos que corresponden a características de un artículo (*ArtCódigo*, *Descripción*, *Precio*, *Unidad*).

FACT_DETALLE

FactNúmero	ArtCódigo	Fecha	Cantidad	Descripción	Precio	Unidad

Figura 6.2. Esquema de relación con anomalías.

Sobre la base de lo antes expuesto, con relación a la semántica de los atributos, Elmasri (2007) propone la siguiente regla de diseño de un esquema de relación: *"diseñar un esquema de relación para que sea fácil explicar su significado. No combine atributos de varios tipos de entidad y de relación en una única relación."*

2. Redundancia de datos y anomalía de actualización. La técnica de diseño de base de datos tiene como objetivo tratar de eliminar la redundancia de datos integrando las relaciones de tal manera que no se almacenen varias copias de los mismos datos; no obstante, la técnica no elimina por completo la redundancia, siendo necesario en ciertos casos duplicar elementos de datos como claves primarias, que copiadas representan claves foráneas y permiten la relación entre los datos. Por ejemplo, en la relación `DETALLE` de la Figura 6.1, se duplican los atributos *FactNúmero* y *ArtCódigo* como claves foráneas de las claves primarias *Número* y *Código* de las relaciones `FACTURA` y `ARTÍCULO` respectivamente, redundancia necesaria y controlada que sirve para expresar los artículos comprados en determinada factura.

Los problemas de redundancia de datos no controlados pueden generar problemas denominados **anomalías de actualización**, las que se clasifican en anomalías de inserción, modificación y borrado.

- Anomalía de inserción: existen dos tipos de anomalías de inserción, que a continuación se ilustran mediante ejemplos basados en la relación `CLIE_VEND` que se muestra en la Figura 6.3 y contiene la información de los clientes y sus vendedores asignados como representantes. En esta relación existe información redundante de los vendedores en los atributos *Vnombre*, *Apellido* y *Sexo*.

  Para insertar un nuevo cliente en la relación `CLIE_VEND`, se debe incluir los datos del vendedor asignado como representante. Por ejemplo, si a un nuevo cliente se le

asigna el vendedor de código 2, además de ingresar el código del vendedor se tiene que ingresar la información correcta de su nombre, el apellido y su objetivo de ventas, de tal manera que sea coherente con los datos registrados en otras tuplas.

De otra parte, si se quiere ingresar un nuevo vendedor, de la manera como está concebida la relación, es necesario contar con un nuevo cliente, de lo contrario se tendría que ingresar valores nulos en los atributos correspondientes al cliente.

CLIE_VEND

Nombre	Apellido1	Apellido2	Cédula	Código	Vnombre	Apellido	ObjVentas
Victor	Castro	Torres	0102030405	1	Diego	Loja	3000
Juan	Polo	Ávila	0122334455	1	Diego	Loja	3000
Elena	Tapia	Barrera	0111555666	1	Diego	Loja	3000
Pablo	Brito	Parra	0123456789	3	Lucía	Serrano	2000
Marcia	Mora	Durazno	0987654321	1	Diego	Loja	3000
Jaime	Pérez	Guerrero	0777888999	2	Antonio	Calle	2500
Humberto	Pons	Coronel	0456456456	3	Lucía	Serrano	2000
Alberto	Torres	Viteri	0999998888	4	Arturo	Salto	4000
Manuel	Bonilla	León	0123123123	2	Antonio	Calle	2500

Figura 6.3. Relación con anomalías.

- Anomalía de borrado: para ilustrar esta anomalía se supone que la vendedora "Lucía Serrano" renuncia a La Ferretería, al borrar las tuplas correspondientes en la relación `CLIE_VEND`, se pierde la información de todos los clientes con quienes estuvo asociada como representante.

- Anomalía de modificación: si se requiere cambiar el valor del objetivo de ventas del vendedor de código 1, se tiene que modificar en todas las tuplas en donde aparece el código 1; en caso de no hacerlo, se generaría incoherencia en la base de dados.

3. Valores nulos. En lo referente a los valores nulos se recomienda evitar que se incluya en una relación atributos cuyos valores sean nulos (`NULL`). Por ejemplo, si en la relación `ARTÍCULO`, únicamente al 5% de los artículos se les asigna un porcentaje de descuento,

este requerimiento no sería una condición para incluir un atributo en la relación, puesto que al 95% de la tuplas de los artículos que no tienen descuento se les tendrá que asignar un valor `NULL`.

La presencia de valores nulos en una relación puede causar tres inconvenientes:

- Desperdicio de espacio de almacenamiento.
- Problemas al momento de ejecutar la instrucción `JOIN`, debido a que las operaciones de concatenación externa e interna producen resultados diferentes cuando los valores nulos forman parte ésta.
- Inconvenientes al momento de procesar los valores nulos, cuando se aplica sobre ellos operaciones de agrupamiento como `SUM` o `COUNT`.

4. Tuplas falsas. Se puede considerar como tupla falsa, aquella que representa información que no es válida. Para impedir la generación de este tipo de tuplas se recomienda *"evitar las relaciones que contienen atributos coincidentes que no son combinaciones de foreign key y clave primaria, porque la concatenación de estos atributos puede producir tuplas falsas*· (Elmasri R., Navathe S., 2016). La reunión no aditiva es una condición formal que garantiza que ciertas concatenaciones no produzcan tuplas falsas. Si el lector se interesa en profundizar el concepto de reunión no aditiva, se recomienda la bibliografía (Elmasri R., Navathe S., 2007), en el Capítulo 11.

Como recomendación general, para evitar estos problemas analizados en esta sección se sugiere crear la base de datos mediante un riguroso diseño conceptual (modelo Entidad Relación) y posteriormente su traspaso al modelo relacional (mapeo).

6.3 DEPENDENCIA FUNCIONAL

Uno de los conceptos más importantes relacionados con la teoría de la normalización es la dependencia funcional, considerada como una restricción que describe la relación entre uno o más atributos.

Definición: sea el esquema relacional `R` definido sobre el conjunto de atributos `A` y sea `X` e `Y` subconjuntos de `A`. Se dice que `Y` depende funcionalmente de `X` o que `X` determina a `Y,` (expresado

como X ⟶ Y), si y solo si cada valor de X tiene asociado en todo momento un único valor de Y. (X e Y pueden estar formados por uno o más atributos).

En términos generales, el atributo Y es funcionalmente dependiente del atributo X, si el valor de X determina Y, esto significa que si se conoce el valor de X se puede obtener el valor de Y. La abreviatura de dependencia funcional es DF y al conjunto de atributos de X se denomina lado izquierdo de DF, mientras que Y es el lado derecho. Por ejemplo, en la relación ARTÍCULO, el atributo *Código* determina funcionalmente al atributo *Unidad*. Esto significa que si se conoce el código de un artículo se puede saber cuál es su unidad.

Código ⟶ *Unidad*

La expresión se lee "*Código* determina funcionalmente a *Unidad*" o "Unidad es dependiente de *Código*". Los atributos del lado izquierdo de la flecha se denominan **determinantes**.

De acuerdo con lo establecido, si el atributo *Código* determina al atributo *Unidad*, un valor particular de código formará un par con un solo valor de unidad; en este caso la relación es de tipo uno a uno (1:1). Por el contrario, un valor de unidad podrá asociarse con uno o más valores diferentes de código; en este caso la relación es de tipo uno a muchos (1:N), es decir, la dependencia *Código* ⟶ *Unidad* en la relación ARTÍCULO es cierta, pero la inversa no es verdadera *Unidad* ⟶ *Código*. En términos generales se expresa de la siguiente manera:

Si X ⟶ Y en la relación R, esto no supone que Y ⟶ X en R.

Por ejemplo, en el esquema de base de datos de nuestro caso de estudio La Ferretería, se pueden definir las siguientes dependencias funcionales:

CódigoVend ⟶ *{Vnombre, Apellido, Sexo, FechaContrato, ObjVentas, Teléfono}*

Código ⟶ *{Descripción, Precio, Unidad, ExiMáx, ExiMín}*

La dependencia funcional es una propiedad de la semántica o significado de los atributos (Elmasri R., Navathe S., 2016), y sirve para determinar cómo se relacionan los atributos. Por ejemplo, si a un estudiante universitario se le identifica con su cédula de identidad *Ci* y se supone que puede cur

sar solo una carrera como se describe en la Figura 6.4(a), en este caso, el valor de la *Ci* determina la *Carrera* que cursa el estudiante, sin embargo, si se cambia el significado de la relación entre los dos atributos y se supone que un estudiante puede cursar varias carreras como se muestra en la Figura 6.4(b), entonces no existiría dependencia funcional entre *Ci* y *Carrera*.

ESTUDIANTE

Ci	Carrera
0101010101	Contabilidad
0123451234	Economía
0112222222	Informática
0133388809	Informática
0133556678	Economía
0198765432	Informática

(a)

ESTUDIANTE

0101010101	Contabilidad
0101010101	Economía
0112222222	Informática
0133388809	Informática
0133556678	Economía
0133556678	Informática

(b)

Figura 6.4. Relación `ESTUDIANTE`.

La dependencia funcional puede considerar grupos de atributos. Por ejemplo, la relación `DETALLE` (*FactNúmero*, *ArtCódigo*, *Cantidad*) sirve para registrar la cantidad de artículos comprados en una determinada factura, en donde, la combinación entre el número de la factura y el código del artículo determinan la cantidad.

{FactNúmero, ArtCódigo ⟶ *Cantidad*

Nótese que tanto *FactNúmero* como *ArtCódigo* son necesarios para determinar *Cantidad*, y no es posible dividir la dependencia funcional, puesto que, ni *FactNúmero* ni *ArtCódigo* determinan por sí mismo *Cantidad*.

En función de este ejemplo se precisa el concepto de **dependencia funcional completa**; "*Si* `X` *e* `Y` *son atributos de una relación,* `Y` *depende funcionalmente de manera completa de* `X`, *si* `Y` *depende funcionalmente de X pero no de ningún subconjunto de* `X`." (Connolly T., Begg C., 2005). En otras palabras, una dependencia funcional `X` ⟶ `Y` es completa, si al eliminar cualquier atributo de `X` hace que se pierda la dependencia funcional. Del ejemplo anterior se deduce:

{FactNúmero, ArtCódigo} ⟶ *Cantidad* (DF completa)

{ArtCódigo} ⟶ *Cantidad* (la dependencia deja de existir)

{FactNúmero} ⟶ *Cantidad* (la dependencia deja de existir)

Otro ejemplo podría ser el registro de la nota recibida en un examen por un estudiante en una materia. Si se considera la relación `CALIFICACIÓN` *(Ci, Materia, Nota)*, en donde la combinación entre la cédula de identidad del estudiante y una materia, determina una nota.

{*Ci*, *Materia*} ⟶ *Nota*

Es importante considerar la diferencia que se presenta en los siguientes patrones basados en la regla de inferencia[2] de descomposición:

si `X` ⟶ `(Y,Z)`, entonces `X` ⟶ `Y` `X` ⟶ `Z`, sin embargo,

si `(X,Y)` ⟶ `Z` entonces en general no es verdad `X` ⟶ `Z` o `Y` ⟶ `Z`

Una dependencia funcional `X` ⟶ `Y` es una **dependencia parcial**, si al eliminar cualquier atributo de `X` continua existiendo la dependencia.

Desde el punto de vista de la normalización, las dependencias funcionales que interesan son: aquellas que tenga una relación uno a uno 1:1, entre el o los atributos del lado izquierdo (determinante) y el o los atributos del lado derecho de la dependencia; y las dependencias funcionales completas.

Hasta el momento se han revisado las dependencias funcionales que son necesarias para el proceso de normalización; sin embargo, existe un tipo de dependencia funcional cuya presencia en la

[2] Los axiomas o reglas de inferencia se pueden usar para inferir o deducir nuevas dependencias funcionales a partir de un conjunto de dependencias funcionales concretas u obvias. Las reglas de inferencia se explican en (Silberschatz , A. Korth, H. Sudarshan, S., 2014, pág. 153).

relación podría causar anomalías de actualización. Este tipo de dependencia recibe el nombre de **dependencia transitiva.**

Una dependencia transitiva se define en los siguientes términos: si `X, Y, Z` son atributos de una relación `R`, en la que existen las dependencias funcionales `X` ⟶ `Y` y `Y` ⟶ `Z`, se dice que `Z` tiene dependencia transitiva respecto de `X` a través de `Y`.

Para ilustrar este concepto, a la relación simplificada `CLIENTE` se le incluyen los atributos *Ciudad* y *Provincia*, que representan el lugar en donde está ubicado un cliente, como se muestra en la Figura 6.5. Para el análisis de la dependencia transitiva se determinan las siguientes dependencias funcionales:

Cédula ⟶ *{Nombre, Apellido1, Dirección, Ciudad, Provincia}*

Ciudad ⟶ *{Provincia}*

Si de estas dependencias se considera *Cédula* ⟶ *Ciudad* y *Ciudad* ⟶ *Provincia*, entonces se dice que *Provincia* tiene dependencia transitiva respecto de *Cédula* a través de *Ciudad*. En otras palabras, el atributo *Cédula* determina funcionalmente a *Provincia* a través del atributo *Ciudad*, pero ni *Ciudad* ni *Provincia* determinan funcionalmente *Cédula*. Esta dependencia se presenta porque cada cliente vive en una sola ciudad y esta ciudad pertenece a una sola provincia.

CLIENTE

Cédula	Nombre	Apellido1	Dirección	Ciudad	Provincia
0102030405	Víctor	Castro	Sucre 1-12	Cuenca	Azuay
0122334455	Juan	Polo	Bolívar 5-67	Cuenca	Azuay
0111555666	Elena	Tapia	Amazonas 3-45	Quito	Pichincha
0123456789	Pablo	Brito	Alfaro 1-23	Guayaquil	Guayas
0987654321	Marcia	Mora	Olmedo 1-10	Riobamba	Chimborazo
0777888999	Jaime	Pérez	Colón 4-98	Quito	Pichincha
0456456456	Humberto	Pons	Borrero 3-45	Cuenca	Azuay
0999998888	Alberto	Torres	Luque 10-10	Guayaquil	Guayas
0123123123	Manuel	Bonilla	Montalvo 5-98	Quito	Pichincha

Figura 6.5. Relación `CLIENTE` con dependencia transitiva.

6.4 PROCESO DE NORMALIZACIÓN Y FORMAS NORMALES

Crear un buen esquema relacional que refleje las necesidades de información del mundo real es el objetivo del diseño de una base de datos. Uno de los inconvenientes en el diseño es el crear directamente un esquema relacional, sin pasar por el diseño conceptual, generando en las relaciones problemas de ambigüedad, redundancia y actualización que se analizaron en las secciones anteriores.

Cualquiera que sea la metodología utilizada de diseño, una vez obtenido el esquema relacional y si éste presenta problemas como los enumerados, es necesario y recomendable evaluar la relación mediante los procesos de normalización.

La normalización es una técnica fundamentada en una serie de reglas que pueden aplicarse a las relaciones individuales, para que satisfagan ciertos requisitos basados en su clave primaria y en las dependencias funcionales. Si una relación no cumple con los requisitos, ésta tiene que ser descompuesta en una serie de relaciones que individualmente cumplan con las condiciones determinadas a través de las formas normales.

Se encuentran definidas seis formas normales en una relación (NF por sus siglas en inglés *normal first*), como se muestran en la Figura 6.6. Si una relación está en primera forma normal (1FN) tiene más redundancia de datos que una relación en una forma normal superior; en consecuencia, presenta más anomalías de actualización de datos. El proceso de normalización se puede asumir como una secuencia de pasos, en donde cada paso corresponde a una forma normal.

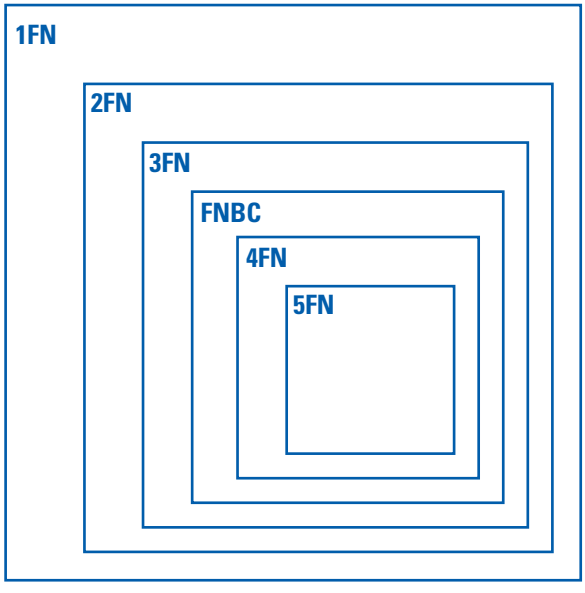


Figura 6.6. Formas normales.

Dependiendo de su estructura, una relación puede estar en 1FN, 2FN, 3FN o en cualquier otra forma normal. Conforme se avanza en el proceso de normalización, la relación adquiere una estructura más restringida y menos sensible a las anomalías. Para evitar las anomalías de actualización se recomienda en general alcanzar hasta la tercera forma normal (3NF).

6.4.1 Primera forma normal (1NF)

Se dice que una relación está en primera forma normal cuando los valores para cada atributo son atómicos, es decir, en cada intersección de una columna y una fila se registra un solo valor. No se permite grupos, ni arreglos repetidos como valor. Por ejemplo, la relación de la Figura 6.7(a), no está en primera forma normal, debido a que en varias intersecciones de una fila y una columna no se registran valores atómicos. Por el contrario, la relación de la Figura 6.7(b), sí está en primera forma normal (1FN), puesto que para cada intersección de una fila y una columna, se registra un sólo valor.

ESTUDIANTE

Ci	Carrera
0101010101	Contabilidad
0123451234, 0133556678	Economía
0112222222, 0133388809, 0198765432	Informática

(a)

ESTUDIANTE

Ci	Carrera
0101010101	Contabilidad
0123451234	Economía
1112222222	Informática
0133388809	Informática
0133556678	Economía
0198765432	Informática

(b)

Figura 6.7. (a) Relación no normalizada. (b) Relación en primera forma normal.

6.4.2 Segunda forma normal (2FN)

Una relación está en segunda forma normal, si y solamente si está en primera forma normal y si todos los atributos que no son clave primaria dependen funcionalmente de manera completa respecto de la clave principal.

6.4.3 Tercera forma normal (3FN)

Una relación está en tercera forma normal, si y solamente si está en segunda forma normal y no tiene dependencias transitivas. La tercera forma normal es la mínima requerida para asegurar la coherencia lógica de los datos al momento de estructurar una relación.

6.5 EJEMPLO PRÁCTICO DEL PROCESO DE NORMALIZACIÓN

En esta sección se presenta un ejemplo práctico de normalización, que alcanza el proceso hasta la tercera forma normal, que es la recomendada para evitar anomalías (Connolly T., Begg C., 2005).

Con el propósito de evitar ambigüedades entre los términos "relación" y "relación entre dos entidades", en esta sección se utiliza el término "tabla" para referirnos a una relación, de acuerdo con la alternativa de la terminología propuesta en el modelo relacional.

Si bien en este ejemplo práctico se pasa de una forma normal a la siguiente de orden superior; sin embargo, este proceso no es una regla. En otros casos una tabla en primera forma normal podrá descomponerse en una serie de tablas en segunda forma normal, o en otros casos, pasará directamente a una serie de tablas en tercera forma normal.

El ejemplo inicia poniendo en evidencia las anomalías que se presentan en el mantenimiento de una tabla, cuando se realizan operaciones de actualización con las instrucciones `INSERT, DELETE` y `UPDATE`, para luego, mediante la técnica de normalización, identificar un conjunto adecuado de tablas, sin la presencia de anomalías y que se garantice actualizaciones coherentes en la base de datos.

Para ilustrar estos procesos se define la tabla `ARTÍCULO`, a través de la instrucción `CREATE TABLE` de SQL.

Create Table `ARTíCULO`

```
(Fabricante        char 15 not null whit default,
 Descripción       char 20 not null whit default,
 Nro_Casilla        numeric 4,
 Bodega            int,
 Estantería        int);
```

Esta tabla contiene la información referente a los artículos que se almacenan en las bodegas de La Ferretería, en la que se registra el nombre y el número de casilla de los fabricantes, la descripción de los artículos, así como los números de la bodega y de la estantería en donde se almacena el artículo, organizado de acuerdo con su fabricante.

El mundo real especifica que cada bodega contará con dos estanterías y sus números serán secuenciales, como se ilustra en la Figura 6.8.

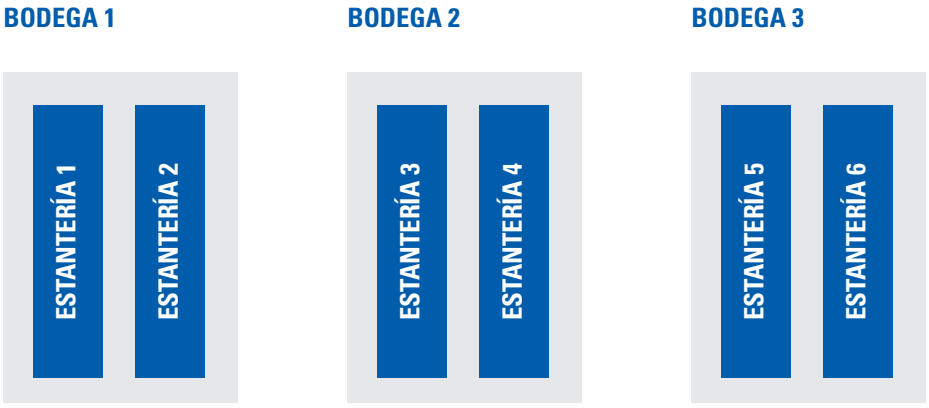


Figura 6.8. Distribución de bodegas y estantería.

Adicionalmente se trabajará con la instancia de la tabla `ARTÍCULOS` que se muestra en la Figura 6.9. (Página 297)

6.5.1 Primera forma normal (1FN) y sus inconvenientes

Revisando los conceptos que se analizaron al inicio del presente capítulo, una tabla está en primera forma normal cuando ésta posee únicamente datos elementales para cada columna en cada línea, es decir, debe contener datos atómicos o indivisibles.

De acuerdo con lo revisado en la Sección 3.4, una relación debe cumplir con las siguientes condiciones:

- Las celdas de una tabla (fila, columna) deben ser atómicas, es decir, deben tener un sólo valor. No está permitido grupos de valores.

- Todos los valores ingresados para un atributo (columna) deben ser del mismo tipo.

ARTÍCULO

Fabricante	Descripción	Nro_Casilla	Bodega	Estantería
Concretos S.A.	Cemento	8765	1	1
Concretos S.A.	Cementina	8765	1	1
JMHierro	Malla	2659	1	2
JMHierro	Hierro	2659	1	2
JMHierro	Alambre	2659	1	2
Maderesa	Madera cerezo	4170	2	3
Maderesa	Madera roble	4170	2	3
Granito	Cerámica	1234	2	4
Granito	Porcelanato	1234	2	4
Grupo TPVC	Tubería PVC	9876	3	5
Accelec	Accesorios eléctricos	1357	3	5
PlastiMex	Tubería PVC	2468	3	6
Aluminio S.A.	Perfil de aluminio	9483	2	3

Figura 6.9. Instancia de la tabla ARTÍCULO.

- Cada atributo (columna) debe tener un nombre único.
- El orden de las columnas en la tabla no es importante.
- El orden de las tuplas (filas) no es significativo.

Con estas condiciones la tabla **ARTÍCULO** se encuentra en primera forma normal, sin embargo, esta tabla no ofrece todas las garantías de integridad cuando se realizan diversas operaciones de actualización, como las que se ilustran en los casos de anomalías que se detallan a continuación:

Anomalía de eliminación (DELETE)

Por ejemplo, si se requiere eliminar de la tabla **ARTÍCULO** (Figura 6.9) las filas correspondientes a los artículos fabricados por la empresa "Concretos S.A.", la instrucción en SQL se escribe de la siguiente manera:

```
Delete from ARTÍCULO where Fabricante = "Concretos S.A."
```

Los resultados de esta instrucción en SQL se muestran en la Figura 6.10.

ARTÍCULO

Fabricante	Descripción	Nro_Casilla	Bodega	Estantería
JMHierro	Malla	2659	1	2
JMHierro	Hierro	2659	1	2
JMHierro	Alambre	2659	1	2
Maderesa	Madera cerezo	4170	2	3
Maderesa	Madera roble	4170	2	3
Granito	Cerámica	1234	2	4
Granito	Porcelanato	1234	2	4
Grupo TPVC	Tubería PVC	9876	3	5
Accelec	Accesorios eléctricos	1357	3	5
PlastiMex	Tubería PVC	2468	3	6
Aluminio S.A.	Perfil de aluminio	9483	2	3

Figura 6.10. Resultado de la operación `DELETE`.

Al anular las filas correspondientes a los artículos fabricados por la empresa "Concretos S.A", se pierde además el nombre y número de casilla del fabricante, información que es independiente de los artículos. Esta pérdida se denomina **anomalía de eliminación**, y sucede cuando al suprimir los datos de una entidad, se pierden los datos de otra entidad. Este inconveniente se presenta debido a que en la tabla `ARTÍCULO` se almacenan datos de dos entidades diferentes: fabricante y artículo.

Anomalía de inserción (`INSERT`)

Se ilustra esta anomalía de inserción a partir de los datos de la tabla de la Figura 6.10. Por ejemplo, para introducir en la tabla `ARTÍCULO`, el artículo "Madera pino" del fabricante "Maderesa", se ejecuta la operación en SQL que se presenta a continuación y el resultado de la instrucción se muestra en la Figura 6.11.

```
Insert into ARTÍCULO (Fabricante, Descripción)
Values ("Maderesa", "Madera pino")
```

ARTÍCULO

Fabricante	Descripción	Nro_Casilla	Bodega	Estantería
JMHierro	Malla	2659	1	2
JMHierro	Hierro	2659	1	2
JMHierro	Alambre	2659	1	2
Maderesa	Madera cerezo	4170	2	3
Maderesa	Madera roble	4170	2	3
Granito	Cerámica	1234	2	4
Granito	Porcelanato	1234	2	4
Grupo TPVC	Tubería PVC	9876	3	5
Accelec	Accesorios eléctricos	1357	3	5
PlastiMex	Tubería PVC	2468	3	6
Aluminio S.A.	Perfil de aluminio	9483	2	3
Maderesa	Madera pino	0	--------	--------

Figura 6.11. Resultado de la operación `INSERT`.

La operación `INSERT` no obliga a ingresar todos los valores concernientes a las columnas de la tabla `ARTÍCULO`, situación que puede ocasionar que se ingrese incoherencias. Por ejemplo, el número de casilla del fabricante "Maderesa" es "4170"; no obstante, se ingresó un valor de "0". De igual manera puede suceder con los valores correspondientes a la ubicación de los artículos de este fabricante en las columnas *Bodega* y *Estantería*.

Anomalía de modificación (`UPDATE`)

Con el objeto de dar mantenimiento a la tabla `ARTÍCULO` de la Figura 6.11 del ejemplo anterior, se ingresan los valores pendientes: número de casilla, bodega y estantería en la fila relativa al fabricante "Maderesa" y al artículo "Madera pino", a través de la siguiente operación `UPDATE` y se obtienen los resultados que se muestran en la Figura 6.12.

```
Update ARTÍCULO

Set Nro_Casilla = 3333, Bodega = 2, Estantería = 2

Where Fabricante = "Maderesa" and Descripción = "Madera pino"
```

ARTÍCULO

Fabricante	Descripción	Nro_Casilla	Bodega	Estantería
JMHierro	Malla	2659	1	2
JMHierro	Hierro	2659	1	2
JMHierro	Alambre	2659	1	2
Maderesa	Madera cerezo	4170	2	3
Maderesa	Madera roble	4170	2	3
Granito	Cerámica	1234	2	4
Granito	Porcelanato	1234	2	4
Grupo TPVC	Tubería PVC	9876	3	5
Accelec	Accesorios eléctricos	1357	3	5
PlastiMex	Tubería PVC	2468	3	6
Aluminio S.A.	Perfil de aluminio	9483	2	3
Maderesa	Madera pino	**3333**	**2**	**2**

Figura 6.12. Anomalía al ingresar una nueva fila.

En este ejemplo se introducen voluntariamente valores erróneos en la línea correspondiente; el número de casilla y la estantería no corresponden a los valores del fabricante "Maderesa", puesto que la instrucción `UPDATE` no controla la presencia de datos referentes a "Maderesa", que se encuentran en otra línea de la tabla.

Para corregir el error se puede realizar la siguiente instrucción:

```
Update ARTÍCULO
Set Nro_Casilla = 4170, Bodega = 1, Estantería = 1
Where Fabricante = "Maderesa" and Descripción = "Madera pino"
```

Si bien con la instrucción `UPDATE` las inconsistencias se corrigen; sin embargo, para determinar los valores correctos, fue necesario recorrer todas las apariciones del fabricante "Maderesa" de la tabla `ARTÍCULO`, lo que puede generar problemas de rendimiento cuando se trata de tablas grandes que contienen muchas líneas de información.

Las anomalías en la tabla `ARTÍCULO` de la Figura 6.9 se pueden definir de la siguiente forma intuitiva:

- La tabla tal como está definida no es más que una yuxtaposición de columnas, que contienen datos que están relacionados a dos temas diferentes.

- Existen datos que corresponden únicamente a los artículos y se registran en las columnas *Fabricante* y *Descripción*.

- Existen también datos relativos al fabricante *(Fabricante, Nro_Casilla, Bodega, Estantería)*. Estos dos últimos atributos corresponden al fabricante, debido a que los artículos están almacenados en las bodegas y estanterías, organizados de acuerdo al fabricante.

Cuando se ingresa una nueva fila se debe agregar datos de dos temas al mismo tiempo (anomalía de inserción), y cuando se suprime una fila, se eliminan por fuerza datos de dos temas al mismo tiempo (anomalía de eliminación).

Este análisis tiene su analogía con la gramática, cuando se dice que un párrafo puede tener un sólo tema o idea. En el caso de tener más de un tema, se tiene que dividir en dos o más párrafos, de tal manera que cada uno se refiera a un sólo tema. Un proceso similar que es la esencia de la normalización se utiliza para el diseño de bases de datos. Cuando una tabla contiene datos de varios temas, se divide en dos o más tablas, de tal manera que cada una corresponda a un sólo tema.

De otra parte, si se analiza ¿qué pasa con la clave y dependencias de esta tabla? Ninguna columna sola, de la tabla `ARTÍCULO` de la Figura 6.9, es una clave, razón por la cual, ésta debe formarse por la combinación de dos o más columnas. Se determina como clave a la combinación de los atributos (*Fabricante*, *Descripción*) que especifican una fila única. El problema con esta tabla es que tiene una dependencia que implica sólo una parte de la clave. Por ejemplo, si se analiza la dependencia *Fabricante* $\longrightarrow$ *Bodega*, el determinante de esta dependencia (*Fabricante*) es una parte de la clave (*Fabricante*, *Descripción*). En estas condiciones se dice que en la tabla `ARTÍCULO`, el atributo *Bodega* es **parcialmente dependiente** de la clave, en consecuencia, la relación presenta

anomalías de actualización por la presencia de una dependencia funcional parcial, sabiendo que la normalización requiere de dependencias funcionales completas en una relación. Para suprimir las anomalías se divide la relación en dos.

Es importante considerar que cada vez que se divide una tabla se pueden crear restricciones de integridad referencial, razón por la cual, es necesario verificar estas condiciones cada vez que se divide una tabla en dos o más.

6.5.2 Relación en segunda forma normal

Para solucionar las anomalías de la tabla ARTÍCULO se crean dos nuevas tablas, como se muestran en las siguientes operaciones:

```
Create Table ARTíCULO1
      (Fabricante       char 15 not null,
       Descripción      char 20 not null)

Create table FABRICANTE1
      (Fabricante       char 15 not null,
      Nro_Casilla       numeric 4,
      Bodega            int,
      Estantería  int);
```

La tabla ARTÍCULO1 obtenida a partir de la tabla ARTÍCULO contiene información relacionada con el fabricante y la descripción de sus artículos. En la Figura 6.13 se describe la instancia de esta tabla.

A partir de la tabla ARTÍCULO se crea la segunda tabla FABRICANTE1, que contiene datos concernientes únicamente al fabricante, como su casilla, la bodega y la estantería en donde se almacenan. Al atributo *Fabricante* se le define como clave. La instancia de esta tabla se describe en la Figura 6.14.

ARTÍCULO 1

Fabricante	Descripción
Concretos S.A.	Cemento
Concretos S.A.	Cementina
JMHierro	Malla
JMHierro	Hierro
JMHierro	Alambre
Maderesa	Madera cerezo
Maderesa	Madera roble
Granito	Cerámica
Granito	Porcelanato
Grupo TPVC	Tubería PVC
Accelec	Accesorios eléctricos
PlastiMex	Tubería PVC
Aluminio S.A.	Perfil de aluminio

Figura 6.13. Instancia de la tabla `ARTÍCULO1`.

FABRICANTE1

Fabricante	Descripción	Bodega	Estantería
Concretos S.A.	8765	1	1
JMHierro	2659	1	2
Maderesa	4170	2	3
Granito	1234	2	4
Grupo TPVC	9876	3	5
Accelec	1357	3	5
PlastiMex	2468	3	6
Aluminio S.A.	9483	2	3

Figura 6.14. Instancia de la tabla `FABRICANTE1`.

Una vez normalizada la tabla `ARTÍCULO` y alcanzada la segunda forma normal en las tablas `ARTÍCULO1` y `FABRICANTE1`, a continuación se analiza el comportamiento de las dos tablas frente a las anomalías que se presentaron antes del proceso de normalización:

- Se descarta la anomalía de eliminación debido a que cada una de las dos tablas generadas a partir de la tabla `ARTÍCULO`, contienen hechos acerca de un solo tema. Es por esta razón que al eliminar los artículos fabricados por la empresa "Concretos S.A.", en la instrucción SQL se hace referencia únicamente a la tabla `ARTÍCULO1` que almacena información del nombre del fabricante y sus artículos, sin considerar la tabla `FABRICANTE1` que registra información concerniente únicamente al fabricante. La instrucción en SQL se podría escribir de la siguiente manera y su resultado se muestra en la Figura 6.15.

```
Delete from ARTÍCULO1 where Fabricante = "Concretos S.A."
```

ARTÍCULO 1

Fabricante	Descripción
JMHierro	Malla
JMHierro	Hierro
JMHierro	Alambre
Maderesa	Madera cerezo
Maderesa	Madera roble
Granito	Cerámica
Granito	Porcelanato
Grupo TPVC	Tubería PVC
Accelec	Accesorios eléctricos
PlastiMex	Tubería PVC
Aluminio S.A.	Perfil de aluminio

Figura 6.15. Tabla `ARTÍCULO1` en segunda forma normal.

- De igual manera se descarta la anomalía de inserción. Por ejemplo, al insertar el artículo "Madera pino" del fabricante "Maderesa" en la tabla `FABRICANTE1`, no se generan incoherencias de datos para los valores de la casilla, bodega y estantería como los que se presentaban cuando la tabla estaba en primera forma (ver tabla de la Figura 6.11)

- Para el caso de la anomalía que se manifestó al actualizar los datos del número de la casilla, la bodega y la estantería en la tabla `ARTÍCULO` (ver Figura 6.12), los errores se presentaron en los tres atributos; sin embargo, en la tabla `FABRICANTE1` al estar en segunda forma normal, no se presenta el error, debido a la existencia de dependencia funcional *Fabricante* ⟶ *{Nro_Casilla, Estantería, Bodega}*. Si bien se corrigió este inconveniente, no obstante, aún se presentan errores en la tabla, los que se revisarán cuando se analice la tercera forma normal.

Una relación está en segunda forma normal, si y solamente si está en primera forma normal y si todo los atributos que no son claves son dependientes de todos los atributos de la clave. Sobre la base de esta definición, cada tabla que tiene un atributo único como clave, está en segunda forma normal; no puede haber en la relación dependencia parcial.

La tabla ARTÍCULO1 no contiene más redundancia, es un conjunto de datos referidos únicamente a los artículos.

Los atributos *Nro_Casilla, Bodega y Estantería* **dependen funcionalmente** de los datos del *Fabricante* en la tabla FABRICANTE1, puesto que, a cada dato de *Fabricante*, corresponde un dato de *Nro_Casilla* y uno sólo, un dato de *Bodega* y uno sólo y un dato de *Estantería* y uno sólo.

Bajo estas consideraciones las tablas ARTÍCULO1 y FABRICANTE1 de las Figuras 6.13 y 6.14 respectivamente, están en segunda forma normal; pero las anomalías pueden todavía ocurrir al momento de realizar el mantenimiento de la tabla FABRICANTE1, ya que también existe dependencia funcional en el interior mismo de la tabla FABRICANTE1, debido a que una estantería pertenece a una y sólo una bodega, según se definió en las condiciones iniciales del mundo real de La Ferretería (ver Figura 6.8). Esto implica que el conocimiento de una estantería define el conocimiento de su bodega (lo contrario no, ya que con el conocimiento de una bodega no se puede determinar la estantería).

Las dependencias funcionales en la tabla FABRICANTE1 están definidas de la siguiente manera: *Fabricante* ⟶ *Estantería* y *Estantería* ⟶ *Bodega*. Esta dependencia se genera porque cada fabricante tiene asignado una estantería y esa estantería pertenece a una sola bodega.

Puesto que, *Fabricante* determina *Estantería* y *Estantería* determina *Bodega*, entonces indirectamente *Fabricante* determina *Bodega*. A este arreglo de dependencia funcional se lo denominó **dependencia transitiva**. En otras palabras, existe una dependencia transitiva entre *Fabricante* y *Bodega* a través de *Estantería*.

La presencia de dependencia transitiva en la tabla FABRICANTE1 genera anomalías que se analizan a continuación:

Anomalía de eliminación (DELETE)

Si se desea eliminar al fabricante "Concretos S.A." de la tabla FABRICANTE1, la instrucción en SQL se escribe de la siguiente manera:

```
Delete from FABRICANTE1 where Fabricante = "Concretos S.A."
```

Además de eliminar la información concerniente al fabricante "Concretos S.A." se produce también la pérdida de los datos de la bodega 1 y la estantería 1, información independiente del fabricante. En la Figura 6.16 se muestra resaltada la fila que se elimina con la instrucción `DELETE`.

FABRICANTE1

Fabricante	Nro_Casilla	Bodega	Estantería
~~Concretos S.A.~~	~~8765~~	~~1~~	~~1~~
JMHierro	2659	1	2
Maderesa	4170	2	3
Granito	1234	2	4
Grupo TPVC	9876	3	5
Accelec	1357	3	5
PlastiMex	2468	3	6
Aluminio S.A.	9483	2	3

Figura 6.16. Anomalía de eliminación en la tabla `FABRICANTE1`.

Anomalía de inserción (`INSERT`)

No es posible crear una nueva bodega y una nueva estantería si no se tiene un fabricante para registrar.

Anomalía de modificación (`UPDATE`)

Si se ejecuta la siguiente operación en SQL

```
Updade FABRICANTE1 Set Bodega = 4
       where Fabricante = "Accelec"
```

Esta instrucción introduce incoherencias en los datos debido a que la estantería 5 pertenece a la bodega 4 para el fabricante "Accelec", y a la bodega 3 para el fabricante "Grupo TPVC". Además, de acuerdo con la distribución de las bodegas y las estanterías que se especificó en el mundo real, la estantería 5 no puede pertenecer a la bodega 4. La tabla resultante luego de ejecutar la instrucción SQL se muestra en la Figura 6.17.

FABRICANTE1

Fabricante	Nro_Casilla	Bodega	Estantería
Concretos S.A.	8765	1	1
JMHierro	2659	1	2
Maderesa	4170	2	3
Granito	1234	2	4
Grupo TPVC	9876	3	5
Accelec	1357	**4**	5
PlastiMex	2468	3	6
Aluminio S.A.	9483	2	3

Figura 6.17. Anomalía de modificación de la tabla `FABRICANTE1`.

6.5.3 Relación en tercera forma normal 3FN

La solución a los problemas de la tabla `FABRICANTE1` que se encuentra en segunda forma normal es crear dos tablas a partir de `FABRICANTE1`, eliminado la dependencia transitiva:

- La primera tabla `FABRICANTE2` contiene los atributos *Fabricante*, *Nro_Casilla* y *Estantería*, con las siguientes dependencias funcionales.

  Fabricante ⟶ *Nro_Casilla*

  Fabricante ⟶ *Estantería*

- La segunda tabla se denomina **LOCAL** y contiene los atributos *Estantería* y *Bodega*, con la siguiente dependencia funcional:

Estantería ⟶ *Bodega*

Las dos tablas obtenidas se encuentran en tercera forma normal y se muestran en la Figura 6.18.

FABRICANTE2

Fabricante	Nro_Casilla	Estantería
Concretos S.A.	8765	1
JMHierro	2659	2
Maderesa	4170	3
Granito	1234	4
Grupo TPVC	9876	5
Accelec	1357	5
PlastiMex	2468	6
Aluminio S.A.	9483	3

LOCAL

Estantería	Bodega
1	1
2	1
3	2
4	2
5	3
6	3

Figura 6.18. Tablas en tercera forma normal.

Una tabla está en tercera forma normal, si y solamente si está en segunda forma normal y no tiene dependencia transitiva. Por lo tanto, para eliminar las anomalías de una tabla en segunda forma normal, es necesario quitar las dependencias transitivas.

La tercera forma normal es la mínima requerida para asegurar la coherencia lógica de los datos al momento de la concepción de una tabla.

Una vez normalizada la tabla **FABRICANTE1**, y alcanzada la tercera forma normal en las tablas **FABRICANTE2** y **LOCAL**, a continuación se analiza el comportamiento de las dos nuevas tablas, frente a las anomalías que se presentaron antes del proceso de normalización:

- Se descarta la anomalía de eliminación, puesto que ahora es posible eliminar al fabricante "Concretos S.A.", sin ocasionar la pérdida de los datos de la bodega 1 y estantería 1. Por ejemplo, la siguiente instrucción en SQL ejecuta dicha acción en la tabla `FABRICANTE2`, y el resultado se muestra en la Figura 6.19. Si bien se elimina el valor del atributo Estantería; sin embargo, éste corresponde a una clave foránea que se refiere a la tabla `LOCAL`.

FABRICANTE2

Fabricante	Nro_Casilla	Estantería
JMHierro	2659	2
Maderesa	4170	3
Granito	1234	4
Grupo TPVC	9876	5
Accelec	1357	5
PlastiMex	2468	6
Aluminio S.A.	9483	3

Figura 6.19. Resultado de la operación `DELETE` sin que genere anomalía.

- Se descarta la anomalía de inserción, ya que es posible crear en la tabla `LOCAL` una nueva bodega y una nueva estantería, independientemente del registro de un fabricante. Por ejemplo, para insertar la bodega 4 y la estantería 7, se ejecuta la siguiente instrucción en SQL. Los resultados de esta operación se muestran en la Figura 6.20.

```
Insert into LOCAL (Estantería, Bodega)

Values (7,4)
```

LOCAL

Estantería	Bodega
1	1
2	1
3	2
4	2
5	3
6	3
7	4

Figura 6.20 Resultado de la operación `INSERT` sin que genere anomalía.

- Se descarta la anomalía de modificación, puesto que tal como están definidas las tablas cada fabricante está asignado a una estantería específica en la tabla **FABRICANTE2** (dependencia funcional entre *Fabricante* ⟶ *Estantería*).

  Como resultado del proceso de normalización realizado a la tabla **ARTÍCULO** de la Figura 6.9, se han obtenido tres tablas normalizadas, con las que se ha conseguido eliminar las anomalías de actualización. Esta separación en tres tablas ha generado restricciones de integridad referencial, las mismas que han sido consideradas y se muestran en el esquema de la Figura 6.21.

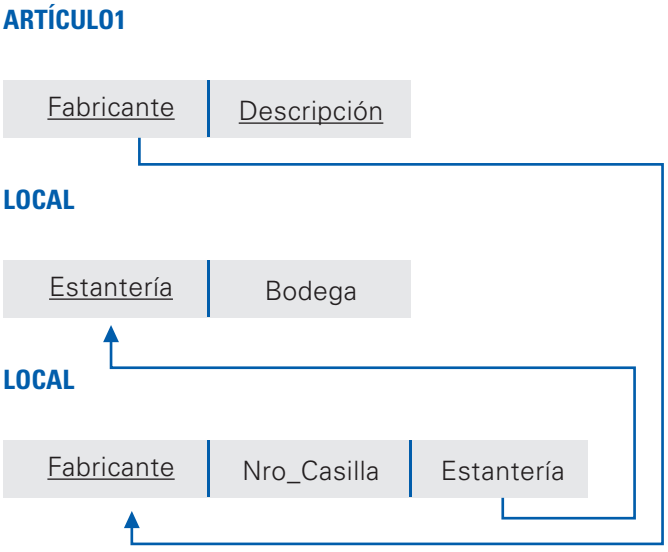


Figura 6.21. Esquema relacional normalizado que incluye integridad referencial.

Con criterios similares con los que se analizó la cardinalidad en el Capítulo 2, la normalización es un proceso que considera las relaciones entre los atributos, relaciones que pueden usarse al momento de construir una tabla y están definidas de la siguiente manera:

- Relación uno a uno. Cuando dos atributos se determinan funcionalmente entre sí.

- Relación uno a muchos. Si uno determina funcionalmente al otro, pero no en el sentido inverso.

- Relación muchos a muchos. Si ninguno de los atributos determina a otro.

En términos generales la normalización es un proceso para convertir una relación que presenta ciertos problemas, en dos o más relaciones libres de anomalías de actualización. La normalización se puede usar como mecanismo para comprobar la validez y pertinencia de las relaciones.

Hablar de la pertinencia o no de la normalización en algunos casos de relaciones, va a depender del costo del procesamiento adicional que involucra el contar con dos o más tablas y sus restricciones de integridad referencial, comparado con el beneficio de evitar anomalías de modificación.

Las anomalías en una relación se producen en procesos de actualización (`INSERT`, `DELETE`, `UPDATE`) y no en procesos de consulta (`SELECT`) a la base de datos. La normalización "castiga" a las consultas, al disminuir la eficiencia en el procesamiento, debido a que aumenta el número de relaciones en la base de datos, lo que implica que, para realizar una consulta se tiene que acceder a varias tablas, aumentando el costo de la misma.

6.6 CUESTIONARIO Y EJERCICIOS

1. Defina lo que es la dependencia funcional y explique a través de un ejemplo.

2. Mediante un ejemplo explique el concepto de primera forma normal (1FN).

3. Describa el concepto de dependencia funcional completa y explique su relación con la segunda forma normal (2FN).

4. Describa el concepto de dependencia transitiva y explique su relación con la tercera forma normal (3FN).

5. Explique los inconvenientes que se presentan en un esquema relacional con la presencia de valores nulos.

6. ¿Qué es la anomalía de eliminación? Explique con un ejemplo.

7. ¿Qué es la anomalía de inserción? Explique con un ejemplo.

8. ¿Qué es la anomalía de modificación? Explique mediante un ejemplo.

9. Defina la dependencia parcial.

10. ¿Por qué el significado o la semántica de los atributos es importante para el diseño de un esquema de base de datos?

11. Describa el objetivo del proceso de normalización.

12. Defina qué es un determinante.

Ejercicios resueltos

13. Normalizar la relación ALMACÉN

ALMACÉN

Departamento	Artículo	Jefe	Cantidad
1	Computadora	Pérez	15
1	Monitor	Pérez	20
7	Teclado	Aguilar	13
7	Monitor	Aguilar	25
2	Ratón	Torres	40
3	Impresora	Suárez	35
3	Computadora	Suárez	10
3	Tarjeta	Suárez	50

ALMACÉN1

Departamento	Jefe
1	Pérez
2	Torres
3	Suárez
7	Aguilar

ALMACÉN2

Departamento	Artículo	Cantidad
1	Computadora	15
1	Monitor	20
7	Teclado	13
7	Monitor	25
2	Ratón	40
3	Impresora	35
3	Computadora	10
3	Tarjeta	50

14. Normalizar la relación VUELO

VUELO

Nro_Vuelo	Fecha	Cód_piloto	Nom_piloto	Total_H_Vuelo
304	4-Ene	1227	Torres	7250
304	8-Ene	3084	Pérez	15412
503	12-Ene	1227	Torres	7250
503	16-Ene	5324	Brito	10284

VUELO_PILOTO

Nro_Vuelo	Cód_piloto	Fecha
304	1227	4-Ene
304	3084	8-Ene
503	1227	12-Ene
503	5324	16-Ene

PILOTO

Cód_piloto	Nom_piloto	Total_H_Vuelo
1227	Torres	7250
3084	Pérez	15412
5324	Brito	10284

1. Normalizar la relación LIBRO_AUTOR

LIBRO_AUTOR

NomAutor	Nacionalidad	Cód_libro	Título	Editorial	Año
Date, C.	Norteamericana	98987	Database	Addison, W.	1990
Date, C.	Norteamericana	97777	SQL Stan	Addison, W.	1986
Date, C.	Norteamericana	98999	Guide for SQL	Addison, W.	1988
Cood, E	Norteamericana	7890	Relational	Addison, W.	1990
Gardain	Francesa	12345	Basi Dati	Paraninfo	1986
Gardain	Francesa	67890	Comp DB	Eyrolles	1984
Valduriez	Francesa	67890	Comp DB	Eyrolles	1984
Kim, W.	Norteamericana	11223	BD OO	ACM	1989
Lochovsky	Canadiense	11223	BD OO	ACM	1989

LIBRO

Cód_Libro	Título	Editorial	Año
98987	Database	Addison, W.	1990
97777	SQL Stan	Addison, W.	1986
98999	Guide for SQL	Addison, W.	1988
7890	Relational	Addison, W.	1990
12345	Basi Dati	Paraninfo	1986
67890	Comp DB	Eyrolles	1984
11223	BD OO	ACM	1989

AUTOR

NomAutor	Nacionalidad
Date, C.	Norteamericana
Cood, E	Norteamericana
Gardain	Francesa
Valduriez	Francesa
Kim, W.	Norteamericana
Lochovsky	Canadiense

ESCRIBE

NomAutor	Cód_Libro
Date, C.	98987
Date, C.	97777
Date, C.	98999
Cood, E	7890
Gardain	12345
Gardain	67890
Valduriez	67890
Kim, W.	11223
Lochovsky	11223

Ejercicios propuestos

16. Normalizar la relación que contiene datos referentes a las instituciones, según se detallan a continuación:

Nombre. Conjunto de palabras para identificar a una institución de educación.

Siglas. Abreviatura formada por el conjunto de letras iniciales del nombre de la institución.

Dirección. Calle y número del lugar en donde se ubica la institución.

Ciudad. Ciudad en donde se ubica la institución.

Provincia. Provincia a la que pertenece la ciudad.

Teléfonos. Números telefónicos de la institución.

Fecha de fundación. Año, mes y día en el que se fundó la institución.

17. Normalizar la siguiente relación.

Código	Nombre	Apellido	Escuela	Facultad	Materia	Nota
3412	Juan	Calle	Contabilidad	Administración	Auditoría	10
3412	Juan	Calle	Contabilidad	Administración	Finanzas	7
3789	Ernesto	López	Economía	Administración	Producción	9
3789	Ernesto	López	Economía	Administración	Finanzas	8
3512	Paulo	Núñez	Sistemas	Ingeniería	Cálculo	9
3512	Paulo	Núñez	Sistemas	Ingeniería	Programación	6
3512	Paulo	Núñez	Sistemas	Ingeniería	Redes	8

18. Normalice la siguiente relación:

VEHÍCULO *(Marca, Modelo, Placa, Propietario, AñoDeFabricación, Colores, TipoDeLicencia, TeléfonoPropietario)*

BIBLIOGRAFÍA

Abad, R., Medina, M., Careaga, A. (1993). *Fundamentos de las estructuras de datos relacionales*. Grupo Noriega Editores, Limusa, México.

Camuña, J. (2014). *Lenguaje de definición y modificación de datos SQL* [Versión electrónica]. (1ra. ed.). IC Editorial, Málaga.

Connolly, T., Begg, C. (2005). *Sistemas de bases de datos. Un enfoque práctico para diseño, implementación y gestión* (4ta. ed.). Pearson, Madrid.

De Miguel, A., Martínez, P., Castro, E., Cavero, J., Cuadra, D., Iglesias, A., Nieto, C. (2005). *Diseño de bases de datos, Problemas resueltos.* Alfaomega Ra-Ma, México.

De Miguel, A., Piattini, M., (1993). *Concepción y diseño de bases de datos. Del modelo E/R al modelo relacional.* Ra-Ma, Madrid.

Elmasri, R., Navathe, S. (2016). *Fundamentals of database systems* (7ma. ed.). Pearson, Hoboken.

Elmasri, R., Navathe, S. (2007). *Fundamentos de sistemas de bases de datos* (5ta. ed.). Pearson, Madrid.

Groff, J., Weinberg, P. (2003). *SQL manual de referencia.* McGraw-Hill, Madrid.

Ibáñez, I. (2014). *Gestión de bases de datos* (2da. ed.). Ra-Ma, Madrid.

Jiménez, M. (2014). *Bases de datos relacionales y modelado de datos,* [Versión electrónica]. IC Editorial.

Koutchouk, M. (1992). *SQL et DB2 le relationnel et sa practique* (2da. ed.). Masson, Paris.

Kroenke, D. (2003). *Procesamiento de bases de datos.* (8va. ed.). Pearson, Mexico.

Martin, J. (1995). *Organización de las bases de d* (Martin, 1995)atos. Prentice Hall Hispanoamericana, México.

Moratalla, J. (2001). *Bases de datos en SQL Server 2000. Transact SQL*. Grupo EIDOS, Madrid. [Versión electrónica].

Recuparado de:

https://issuu.com/dianacarolinapauca/docs/bases_de_datos_con_sql_-_j.moratall

Piattini, M., Clavo, J., Cervera, J., Fernández, L. (1996) *Análisis y diseño detallado de aplicaciones informáticas de gestión.* Ra-Ma, Madrid.

Silberschatz, A., Korth, H., Sudarshan, S. (2014). *Fundamentos de bases de datos (*6ta. ed.). McGraw-Hill, Madrid.

Viescas, J., Hernández, M. (2014). *SQL Queries form mere mortals* (3ra. ed.). Addison - Wesley.

INDICE DE FIGURAS