Col·lecció
«Treballs d'Informàtica i Tecnologia»
Núm. 29

APRENDE SQL

G. Quintana, M. Marqués,
J. I. Aliaga, M. J. Aramburu

UNIVERSITAT
JAUME·I

BIBLIOTECA DE LA UNIVERSITAT JAUME I. Dades catalogràfiques

APRENDE SQL / G. Quintana ... [et al.]. — Castelló de la Plana :Publicacions de la Universitat Jaume I, D.L. 2008
p. : il. ; cm. — (Treballs d'informàtica i tecnologia ; 29)
Bibliografia.
ISBN 978-84-8021-661-6
1. SQL (Llenguatge de programació). I. Quintana, Gregorio. II. Universitat Jaume I. Publicacions. III. Sèrie.
681.3.06

Cap part d'aquesta publicació, incloent-hi el disseny de la coberta, no pot ser reproduïda, emmagatzemada, ni transmesa de cap manera, ni per cap mitjà (elèctric, químic, mecànic, òptic, de gravació o bé de fotocòpia) sense autorització prèvia de la marca editorial.

© Del text: els autors, 2008

© De la present edició: Publicacions de la Universitat Jaume I, 2008

Edita: Publicacions de la Universitat Jaume I. Servei de Comunicació i Publicacions
Campus del Riu Sec. Edifici Rectorat i Serveis Centrals. 12071 Castelló de la Plana
Fax 964 72 88 32
www.tenda.uji.es e-mail: publicacions@uji.es

ISBN: 978-84-8021-661-6

Dipòsit legal: B-27.869-2008

Imprimeix: Book Print Digital, SA

Índice

PRÓLOGO 1

1 INTRODUCCIÓN 5

1.1 ORIGEN Y DIFUSIÓN DE LAS BASES DE DATOS 5
1.2 CONCEPTOS BÁSICOS DE LOS SISTEMAS RELACIONALES 7
1.3 EL LENGUAJE SQL 8
1.3.1 Partes de SQL 9
1.3.2 Sentencias del Lenguaje de Manipulación de Datos 10
1.3.3 Orígenes y estándares 10
1.4 BASE DE DATOS DE EJEMPLO 11

2 INICIACIÓN A SQL 17

2.1 INICIACIÓN A LA SENTENCIA SELECT 17
2.2 MODIFICADOR DISTINCT 18
2.3 RESTRICCIÓN EN LAS FILAS 19
2.4 EJECUCIÓN DE SENTENCIAS 20
2.5 EJERCICIOS 20
2.6 AUTOEVALUACIÓN 21

3 FUNCIONES Y OPERADORES ESCALARES 23

3.1 EXPRESIONES Y TIPOS 23
3.2 OPERADORES DE COMPARACIÓN 24
3.2.1 Operador between 24
3.2.2 Operador in 25
3.2.3 Operador like 26
3.3 OPERADORES Y FUNCIONES DE ELECCIÓN 26
3.3.1 Operador estándar case 27
3.3.2 Funciones no estándares de elección: decode, iif, switch y choose 28
3.4 MANEJO DE LOS VALORES NULOS 29
3.4.1 Tablas de verdad de los valores nulos 30
3.4.2 Detección de valores nulos 30
3.4.3 Conversión de valores nulos 32
3.5 CONVERSIONES DE DATOS 33
3.5.1 Función to_char 33
3.5.2 Función to_number 33
3.6 PROCESAMIENTO DE FECHAS 34
3.6.1 Fecha actual 34
3.6.2 Extracción de una parte de una fecha 34
3.6.3 Conversión de una fecha al formato deseado 36

3.6.4 Conversión a fecha desde un formato determinado 36
3.7 PROCESAMIENTO DE CADENAS DE CARACTERES 37
3.7.1 Delimitación de cadenas 37
3.7.2 Concatenación de cadenas 37
3.7.3 Longitud de una cadena 37
3.7.4 Extracción de una parte de una cadena 38
3.7.5 Conversiones a mayúsculas y minúsculas 38
3.8 FUNCIONES MATEMÁTICAS 39
3.8.1 Función round 39
3.8.2 Otras funciones 39
3.9 ALIAS DE COLUMNAS 40
3.10 PRUEBAS DE FUNCIONES Y EXPRESIONES 40
3.11 EJERCICIOS 41
3.12 AUTOEVALUACIÓN 43

4 FUNCIONES DE COLUMNA 45

4.1 FUNCIONES ESCALARES FRENTE A FUNCIONES DE COLUMNA 45
4.2 FUNCIONES DE COLUMNA HABITUALES 46
4.3 FUNCIONES DE COLUMNA CON RESTRICCIONES 47
4.4 FUNCIONES DE COLUMNA CON VALORES NULOS 48
4.5 ERRORES HABITUALES 49
4.6 EJERCICIOS 50
4.7 AUTOEVALUACIÓN 52

5 AGRUPACIÓN 53

5.1 MOTIVACIÓN. INTRODUCCIÓN 53
5.2 FUNCIONES DE GRUPO 56
5.3 TIPOS DE AGRUPACIONES 56
5.4 AGRUPACIONES POR MÚLTIPLES FACTORES 57
5.5 AGRUPACIONES INCORRECTAS 58
5.6 RESTRICCIONES DE FILA Y RESTRICCIONES DE GRUPO 59
5.7 EJECUCIÓN DE UNA CONSULTA CON AGRUPACIÓN 61
5.8 COMBINACIÓN DE FUNCIONES DE GRUPO Y FUNCIONES DE COLUMNA 62
5.9 REGLAS NEMOTÉCNICAS 63
5.10 EJERCICIOS 63
5.11 AUTOEVALUACIÓN 66

6 CONCATENACIÓN INTERNA DE TABLAS 67

6.1 CONCATENACIÓN INTERNA DE DOS TABLAS 67
6.2 ALIAS DE TABLAS 69
6.3 SINTAXIS ESTÁNDAR 70
6.3.1 Operador A natural join B 70
6.3.2 Operador A [inner] join B using (lista_columnas) 71

6.3.3 Operador A [inner] join B on expresión_booleana 71
6.3.4 Operador A cross join B 72
6.3.5 Sintaxis tradicional frente a sintaxis estándar 72
6.3.6 Método de trabajo con la sintaxis estándar 72
6.4 CONCATENACIÓN INTERNA DE TRES O MÁS TABLAS 74
6.5 CONCATENACIÓN DE UNA TABLA CONSIGO MISMA 75
6.6 CONCATENACIÓN Y AGRUPACIÓN 75
6.7 CONSIDERACIONES SOBRE LAS PRESTACIONES 76
6.8 EJERCICIOS 77
6.9 AUTOEVALUACIÓN 80

7 ORDENACIÓN Y OPERACIONES ALGEBRAICAS 83

7.1 ORDENACIÓN DEL RESULTADO 83
7.2 OPERACIONES ALGEBRAICAS 84
7.2.1 Operador de unión 84
7.2.2 Operador de intersección 86
7.2.3 Operador de diferencia 86
7.2.4 Uso incorrecto de los operadores algebraicos 87
7.2.5 Variantes de SQL y operadores algebraicos 88
7.3 EJERCICIOS 88
7.4 AUTOEVALUACIÓN 91

8 CONCATENACIÓN EXTERNA DE TABLAS 93

8.1 PROBLEMAS DE LA CONCATENACIÓN INTERNA 93
8.2 CONCATENACIÓN EXTERNA DE DOS TABLAS 95
8.2.1 Concatenación externa por la izquierda: A left join B 95
8.2.2 Concatenación externa por la derecha: A right join B 96
8.2.3 Concatenación externa completa: A full join B 97
8.2.4 Equivalencias y Ejemplos 97
8.3 CONCATENACIÓN EXTERNA Y AGRUPACIÓN 98
8.4 CONCATENACIÓN EXTERNA Y UNIÓN 99
8.5 CONCATENACIÓN EXTERNA DE TRES O MÁS TABLAS 100
8.6 EJERCICIOS 101
8.7 AUTOEVALUACIÓN 104

9 SUBCONSULTAS 105

9.1 INTRODUCCIÓN 105
9.2 SUBCONSULTAS QUE DEVUELVEN UN ÚNICO VALOR 106
9.3 SUBCONSULTAS QUE DEVUELVEN UNA ÚNICA FILA 107
9.4 SUBCONSULTAS QUE DEVUELVEN UN CONJUNTO DE FILAS 110
9.4.1 Operador in 111
9.4.2 Operador not in 113
9.4.3 Operador any 115

9.4.4 Operador all 116
9.4.5 Referencias externas 119
9.4.6 Operador exists 120
9.4.7 Operador not exists 121
9.5 SUBCONSULTAS EN LA CLÁUSULA FROM 122
9.6 EQUIVALENCIA DE SUBCONSULTA Y CONCATENACIÓN INTERNA 124
9.7 EQUIVALENCIA DE SUBCONSULTA Y CONCATENACIÓN EXTERNA 124
9.8 OPERACIÓN "TODO" 125
9.9 EQUIVALENCIA DE SENTENCIAS 127
9.10 EJERCICIOS 127
9.11 AUTOEVALUACIÓN 133

10 CREACIÓN Y ACTUALIZACIÓN DE LOS DATOS 135

10.1 CREACIÓN DE TABLAS 135
10.2 BORRADO DE TABLAS 139
10.3 INSERCIÓN DE DATOS 140
10.4 MODIFICACIÓN DE DATOS 142
10.5 BORRADO DE DATOS 144

11 MANEJO AVANZADO DE LOS DATOS 145

11.1 MODIFICACIÓN DE TABLAS 145
11.2 CREACIÓN DE VISTAS 147
11.3 CREACIÓN DE ÍNDICES 149

12 SOLUCIÓN A LOS EJERCICIOS DE AUTOEVALUACIÓN 153

12.1 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 2 153
12.2 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 3 154
12.3 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 4 155
12.4 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 5 155
12.5 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 6 156
12.6 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 7 158
12.7 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 8 159
12.8 SOLUCIONES A LA AUTOEVALUACIÓN DEL CAPÍTULO 9 160

13 EJERCICIOS AVANZADOS 165

14 ANEXOS 183

14.1 FORMATOS PARA LAS FUNCIONES TO_NUMBER Y TO_CHAR 183
14.2 MÁSCARAS PARA LA FUNCIÓN EXTRACT 183
14.3 FORMATOS DE FECHA (TO_CHAR Y TO_DATE) 184

BIBLIOGRAFÍA 187

PRÓLOGO

El objetivo de este texto es permitir al lector introducirse y profundizar de forma práctica y aplicada en el lenguaje SQL (*Structured Query Language*), un lenguaje utilizado en la mayor parte de los sistemas de gestión de bases de datos actuales, tanto en los sistemas destinados a las pequeñas empresas como en los dedicados a las grandes corporaciones. De hecho, en pocas áreas de la informática un lenguaje predomina de forma tan clara y rotunda como el SQL en el campo de las bases de datos.

Este texto no está pensado para introducir conceptos teóricos de bases de datos. No obstante, se presentan los conceptos básicos necesarios sobre teoría de bases de datos para profundizar en este lenguaje.

El objetivo es introducir gradualmente al lector en el uso de este lenguaje mostrando sus diversos conceptos e ilustrándolos siempre con numerosos ejemplos y ejercicios aplicados, completos y sencillos. Todos los ejemplos giran en torno al sistema de facturación y control de *stocks* de una empresa, con lo cual se pretende atraer una mayor atención del lector a la par que se muestra un ejemplo con una utilidad práctica real en la empresa. La descripción de SQL será gradual, desde sentencias de dos líneas hasta las más avanzadas. Este texto no está pensado como un manual de referencia de SQL. Para ello existen otros muchos textos. La principal idea de este texo es introducir el lenguaje SQL desde un punto práctico y tutorial, pero sin olvidar los conceptos teóricos subyacentes.

La presentación del lenguaje SQL tiene en este texto un enfoque muy práctico y aplicado. Se han incluido numerosos ejemplos y ejercicios al lado de cada concepto nuevo introducido para que de esta forma el lector pueda practicar y ver en la práctica cada concepto que se describe. De hecho, la mayor parte de los capítulos incluyen más de 20 ejercicios. Los ejercicios se presentan siempre con solución para que el lector pueda contrastar ésta con la suya o estudiarla sin más. Las soluciones se dan inmediatamente tras la propuesta para evitar al lector la labor de ir al final del texto tras cada ejercicio a buscar la solución en un apéndice final. Únicamente se han llevado a un apartado final las soluciones de los ejercicios de autoevaluación dado su carácter más evaluador y menos didáctico. Estos ejercicios, que ya no incluyen ayuda alguna, deben realizarse cuando el lector haya resuelto satisfactoriamente todos los anteriores.

Este texto está casi principal, aunque no exclusivamente, dedicado a la sentencia de recuperación de datos **select**, dado que es con mucho la más compleja. A su lado, las otras resultan simples reducciones o simplificaciones suyas. Esta sentencia presenta tres conceptos muy importantes sobre los que se sustenta la potencia del lenguaje SQL: la agrupación, la concatenación de tablas y las subconsultas. Estos conceptos se introducirán poco a poco con numerosos ejemplos.

El contenido de este libro puede agruparse en varias partes, cada una de las cuales se descompone en uno o más capítulos. A continuación se presentan brevemente cada una de las partes y cada uno de los capítulos que las componen.

- La primera parte es, obviamente, una introducción general al mundo de las bases de datos y una introducción muy básica y rudimentaria al lenguaje SQL.
  - El capítulo 1 realiza una presentación general del mundo de las bases de datos relacionales, repasando su pasado, presente y futuro, y del lenguaje SQL. Finalmente presenta el ejemplo de base de datos que se va a emplear a lo largo de todo el texto.
  - El capítulo 2 inicia al lector en el lenguaje SQL y, concretamente, en la sentencia de recuperación de datos.
  - El capítulo 3 describe un conjunto de funciones y operadores que permiten realizar cálculos avanzados como por ejemplo las conversiones de tipos, extracciones de partes de una fecha, comparaciones avanzadas, diversas operaciones matemáticas, etc.
- La segunda parte comienza ya a profundizar en el lenguaje SQL presentando las funciones de columna y el concepto de agrupación.
  - El capítulo 4 introduce las funciones de columna que suele incluir SQL. Estas son funciones que resumen toda una columna de una tabla en un solo valor, como por ejemplo, la suma, el máximo, la media, etc.

- El capítulo 5 presenta un concepto fundamental de SQL: la agrupación. Esta es una técnica muy útil que permite calcular un valor para cada grupo de filas de una tabla

- La tercera parte se mete ya a fondo en el lenguaje SQL presentando diversos conceptos muy importantes: la concatenación interna, la ordenación, las operaciones algebraicas, la concatenación externa y las subconsultas.
  - El capítulo 6 describe una forma de extraer información almacenada entre varias tablas: la concatenación interna de tablas. A las sentencias que emplean este tipo de operación se les denomina consultas multitabla dado que acceden a varias tablas.
  - El capítulo 7 presenta las distintas operaciones algebraicas que ofrece este lenguaje y, además, la ordenación del resultado de una consulta.
  - El capítulo 8 presenta una variante de la concatenación interna de tablas: la concatenación externa de tablas, la cual permite procesar tablas con valores nulos de forma correcta y eficiente.
  - El capítulo 9 presenta un concepto muy importante de SQL: las subconsultas. Una subconsulta es una consulta metida dentro de otra consulta. Las subconsultas dotan de una gran potencia al lenguaje SQL.
- La cuarta parte describe dos aspectos fundamental: la parte de definición de datos del lenguaje SQL y la actualización (inserción, borrado y modificación) de los datos de las tablas. En esta parte se presentan conceptos imprescindibles como la creación de tablas, el borrado de tablas, la modificación del esquema de las tablas, la modificación del contenido de las tablas, etc.
  - El capítulo 10 realiza una introducción a la definición del esquema de las tablas y, además, a la actualización (inserción, borrado y modificación) del contenido de las tablas.
  - El capítulo 11 presenta los conceptos más avanzados de esta parte: modificación de los esquemas, vistas, índices, etc.
- La quinta parte se centra exclusivamente en presentar ejercicios y soluciones y algunos anexos. Su estudio no va a aportar nuevos conocimientos teóricos sobre el lenguaje SQL, pero sí va a permitir alcanzar unos mayores conocimientos aplicados.
  - El capítulo 12 contiene las soluciones a todos los ejercicios de autoevaluación de los capítulos anteriores. Estos ejercicios sirven para que el lector compruebe si ha alcanzado los objetivos previstos. Son ejercicios más complejos, sin ayuda ninguna, a los que el lector debe enfrentarse cuando haya resuelto sin problemas todos los ejercicios de cada capítulo.
  - El capítulo 13 presenta numerosos ejercicios de un nivel medio o alto que permiten practicar con todos los conceptos adquiridos en capítulos

anteriores. Dada su mayor dificultad, el lector debe abordar estos problemas cuando haya resuelto satisfactoriamente todos los anteriores.

- El capítulo 14 incluye algunos anexos muy útiles sobre formatos de fecha, etc.

Finalmente, con estas líneas deseamos agradecer su ayuda a todas aquellas personas que han colaborado en las numerosas revisiones de este texto, entre las cuales hay que destacar a Carlos Serra Toro.

1 INTRODUCCIÓN

Este capítulo introduce en primer lugar el mundo de las bases de datos y, a continuación, realiza una presentación general del lenguaje SQL, incluyendo la finalidad, motivación, orígenes y estándares. Asimismo, introduce los conceptos básicos necesarios sobre el modelo relacional. Finalmente describe la base de datos que se va a emplear a lo largo de todo el libro en los distintos ejemplos.

1.1 Origen y difusión de las bases de datos

Antes de la aparición de las bases de datos, en los comienzos de la informática, la información era procesada con aplicaciones construidas con lenguajes de programación tradicionales y era almacenada mediante un sistema de ficheros, habitualmente proporcionado por el sistema operativo. Más adelante, debido a las serias limitaciones y escasas posibilidades de los primeros sistemas de ficheros, éstos se fueron refinando y surgieron ciertas mejoras, proporcionando una mayor riqueza de características y operaciones (ficheros directos, ficheros secuencial-indexados, etc.).

La importancia del procesamiento automático de la información originó numerosos esfuerzos, trabajos e investigaciones que dieron sus frutos en el avance más importante de este campo (y uno de los más importantes en el mundo de la informática): la aparición de las primeras bases de datos en los años 1960. Las ventajas que aportaron frente a los sistemas de ficheros han sido determinantes en su auge actual: mayor flexibilidad, mayor independencia del sistema operativo y del *hardware*, mayor tolerancia a fallos, mayor facilidad en el uso concurrente, el concepto de transacción, etc. Sin embargo, todas estas ventajas no son gratuitas pues requieren una elevada potencia de cómputo. Afortunadamente, los ordenadores no han cesado de incrementar su velocidad, lo cual ha permitido reducir drásticamente los costes del equipamiento informático necesario para hacer funcionar una base de datos.

Mención especial merece el campo de las bases de datos sobre ordenadores personales. A principios de los 1980 los primeros ordenadores personales, dada su escasa potencia, no disponían de bases de datos. Por ello, la información debía guardarse trabajando con el sistema de ficheros del sistema operativo. Posteriormente surgieron para estos ordenadores algunos paquetes de *software* que aportaban un sistema de ficheros secuencial-indexados, los cuales permitían al programador trabajar de forma más cómoda.

Conforme los ordenadores personales ganaron potencia, a finales de los 1980, comenzaron a surgir las primeras bases de datos (p.e. dBASE III+) para este tipo de ordenadores. Aunque en realidad éstas no eran tales pues carecían de numerosas e importantes características presentes en las bases de datos de los ordenadores grandes, estos sistemas se extendieron enormemente dado que permitían reducir drásticamente el tiempo necesario para desarrollar una aplicación. Por ejemplo, aplicaciones que con Pascal y un paquete de ficheros secuencial-indexados costarían varios meses en ser desarrolladas, podían terminarse en un solo mes con una de estas bases de datos.

Posteriormente, el incremento casi exponencial de la potencia de los ordenadores personales ha resultado en la difusión de una nueva generación de bases de datos más modernas, complejas y poderosas que sus predecesoras. En la actualidad, las nuevas bases de datos para ordenadores personales incluyen numerosas características y tienen poco que envidiar, excepto su potencia, a las bases de datos de los grandes sistemas. Por ejemplo, aplicaciones que con dBASE costarían un mes en ser desarrolladas, con una base de datos actual pueden terminarse en unas pocas horas.

Actualmente, aunque las grandes y medianas empresas disponen todas ellas de bases de datos desde hace muchos años, el interés en este campo no ha menguado, sino todo lo contrario. Por ejemplo, en 2002 el crecimiento de este campo fue del 30%. La informatización con bases de datos de las grandes y medianas empresas ha continuado y, lo que es más importante, su uso crece vertiginosamente en el mundo de la informática personal y de la pequeña empresa.

Las bases de datos se pueden agrupar en varios tipos o modelos teóricos: modelo jerárquico, en red y relacional. Las bases de datos jerárquicas surgieron en la década de 1960, poco después aparecieron las bases de datos en red y, un poco más tarde, las bases de datos relacionales.

El modelo relacional fue propuesto por primera vez por el Dr. Edgar Codd, de IBM, en 1970 y el primer sistema comercial apareció en 1976. Debido a las importantes ventajas que aporta en comparación con los dos modelos anteriores, los ha desplazado completamente del mercado. Hoy en día la gran mayoría de bases de datos comerciales pertenecen al modelo relacional.

En realidad, comienzan ya a extenderse las bases de datos basadas en el modelo objeto-relacional. Su principal ventaja es que complementa al modelo relacional con el modelo orientado a objetos, permitiendo incorporar el manejo de tipos de datos complejos (XML, texto, multimedia, etc.).

1.2 Conceptos básicos de los sistemas relacionales

Una base de datos es un conjunto de datos relacionados entre sí. Un sistema de gestión de base de datos (SGBD) es un conjunto de programas que permiten almacenar y procesar la información contenida en una base de datos.

Una base de datos relacional es aquélla en la cual toda la información se almacena en tablas. Una tabla está formada por filas y columnas. Las bases de datos grandes pueden llegar a contener varias decenas de tablas, cada una con miles o millones de filas.

Cada tabla tiene un nombre único y un conjunto de filas y columnas. A continuación se presentan dos tablas de ejemplo. La primera contiene información relacionada con las facturas; la segunda, información relacionada con los clientes.

Tabla FACTURAS

CODFAC	CODCLI	FECHA	IVA	DTO
30	100	1-01-2011	16	5
31	101	2-01-2011		9
32	101		16	5
33	106	8-01-2011	16	5

Tabla CLIENTES

CODCLI	NOMBRE	DIRECCIÓN	CODPUE
101	Alberto	Cuesta, 5	1000
102	Carlos	Cervantes, 3	1000
103	Pedro	Colón, 4	1001

Cada fila contiene información sobre una única instancia de una entidad. Por ejemplo, cada fila de la tabla facturas contiene información sobre una factura.

Cada columna contiene información sobre una única propiedad o atributo de las entidades. Por ejemplo, la columna nombre de la tabla clientes contiene los nombres de los clientes.

Una celda es una intersección de una fila y de una columna. Cada celda puede contener bien un valor o bien ningún valor. Cuando no contiene ningún valor, se dice que tiene el valor nulo. El valor nulo puede tener dos orígenes: un valor desconocido o un valor no aplicable. Por ejemplo, la tabla de facturas tiene un valor nulo en la fecha de la factura con código 32.

Las filas de las tablas no están ordenadas ni tienen una posición fija. Por tanto, no se puede acceder a una determinada información de una tabla a través de su posición en la tabla (tercera fila, vigésima fila, etc.). Para acceder a una entidad hay que conocer alguna de sus propiedades o atributos.

La clave primaria es una columna o conjunto de columnas que identifican de forma única a cada una de las entidades (filas) que componen las tablas. En Access se la denomina clave principal a esta clave.

La clave ajena es una columna o conjunto de columnas cuyos valores coinciden con el valor de una clave primaria de una tabla.

Existen dos reglas que permiten mantener la integridad de la información de las bases de datos:

La *Regla de Integridad de Entidades* especifica que ninguna de las columnas que componen la clave primaria de una tabla puede contener valores nulos.

La *Regla de Integridad Referencial* especifica que las claves ajenas o bien son completamente nulos o bien contienen valores tales que coinciden con algún valor de la clave primaria a la que referencian. Por ejemplo, las anteriores tablas no cumplen esta regla dado que la factura con código 33 referencia a un cliente que no existe (código de cliente 106).

Resulta muy conveniente que la información contenida en las bases de datos cumpla ambas reglas. En caso contrario, la información se encuentra en un estado inconsistente que a la larga sólo puede producir errores. Pensemos, por ejemplo, en facturas para clientes que no existen, en ventas de artículos que no existen, etc.

No todos los SGBD hacen cumplir estas reglas a los datos que contienen. Algunos sistemas permiten hacer cumplir ambas reglas de forma opcional. En este caso se recomienda encarecidamente indicar al sistema su obligado cumplimiento pues así el diseñador se libera de tener que comprobar la integridad de los valores tras cada actualización, borrado o inserción.

1.3 El lenguaje SQL

SQL (*Structured Query Language*) es un lenguaje de programación diseñado específicamente para el acceso a Sistemas de Gestión de Bases de Datos Relacionales (SGBDR). Como la mayor parte de los sistemas actuales son de este tipo, y como el lenguaje SQL es el más ampliamente usado en éstos, se puede decir sin ningún género de dudas que este lenguaje es empleado mayoritariamente en los sistemas existentes hoy en día e indiscutiblemente no tiene rival alguno.

Este lenguaje es empleado en sistemas informáticos que van desde ordenadores personales muy básicos con apenas 64 MB de espacio en memoria central hasta los más potentes multiprocesadores y multicomputadores con decenas de procesadores superescalares de 64 bits.

Las principales ventajas que aportan SQL son dos:

- Su enorme difusión pues es empleado en la gran mayoría de los sistemas actuales.
- Su elevada expresividad. Por ejemplo, operaciones que costarían semanas de duro esfuerzo en ser desarrolladas en un lenguaje de programación tradicional pueden ser realizadas con SQL en tan sólo unos minutos.

El lenguaje SQL es un lenguaje de cuarta generación. Es decir, en este lenguaje se indica qué información se desea obtener o procesar, pero no cómo se debe hacer. Es labor interna del sistema elegir la forma más eficiente de llevar a cabo la operación ordenada por el usuario.

A la hora de describir el lenguaje SQL siempre hay que tomar una decisión difícil: decidir la variante de SQL que se va estudiar. Aunque los conceptos básicos son idénticos en todos los estándares y en todos los sistemas implementados, existen numerosas diferencias que dificultan la portabilidad. Desgraciadamente existen numerosos estándares de SQL (SQL-86, SQL-89, SQL-92, SQL-99, SQL-2003) y, lo que es peor, ninguno ha sido completamente aceptado. Además, las empresas fabricantes de SGBDR implementan el estándar que les da la gana y, lo que es peor, añaden y quitan características sin el menor rubor ni reparo.

Todo lo anterior hace bien difícil la elección de la variante de SQL en la que enfocar este texto. Así pues, para no centrar el libro en un estándar que nadie siga o centrarlo en un sistema comercial que deje de lado a otros sistemas y a los estándares, se va a procurar describir simultáneamente tanto un estándar como algunos de los sistemas comerciales más extendidos, procurando comentar las diferencias allí dónde las haya. Dado el carácter introductorio del texto este objetivo no va a resultar tan difícil pues las mayores divergencias surgen en los aspectos más avanzados. Las implementaciones comerciales del lenguaje SQL que se presentan son las de Microsoft Access, Oracle, PostgreSQL y MySQL al ser éstas de las más empleadas; tanto en el mundo de la pequeña empresa como entre las grandes compañías.

1.3.1 Partes de SQL

El lenguaje SQL consta de dos partes claramente diferenciadas:

- Lenguaje de Definición de Datos (en inglés *Data Definition Language* o DDL): Incluye aquellas sentencias que sirven para definir los datos o para modificar su definición, como por ejemplo la creación de tablas, índices, etc.
- Lenguaje de Manipulación de Datos (en inglés *Data Manipulation Language* o DML): Incluye aquellas sentencias que sirven para manipular o procesar los datos, como por ejemplo la inserción, borrado, modificación o actualización de datos en las tablas.

La primera parte, el DDL, se abordará en capítulos posteriores, mientras que ahora y en los capítulos inmediatos se van a estudiar las sentencias de manipulación de datos.

1.3.2 Sentencias del Lenguaje de Manipulación de Datos

SQL presenta cuatro sentencias de manipulación de datos:

- Sentencia **select**: Permite extraer información almacenada en la base de datos. Es una operación de sólo lectura.
- Sentencia **insert**: Permite insertar información en la base de datos.
- Sentencia **update**: Permite modificar información almacenada en la base de datos.
- Sentencia **delete**: Permite borrar información existente en la base de datos.

De estas cuatro sentencias, la más compleja y poderosa es sin duda la primera. De hecho, el funcionamiento y estructura de las tres últimas sentencias es un subconjunto de las posibilidades de la primera aplicadas a una tarea particular.

Por tanto, a continuación y en los temas siguientes se estudiará la sentencia **select**, dejando para el final las otras tres al ser su comportamiento mucho más sencillo y casi trivial en comparación con la primera.

1.3.3 Orígenes y estándares

El lenguaje SQL fue desarrollado por IBM dentro del proyecto System R a finales de 1970. Desde entonces ha ganando una gran aceptación y ha sido implementado por numerosos productos experimentales y, sobre todo, comerciales.

En 1986 y 1987 las organizaciones ANSI (*American National Standards Institute*) e ISO (*International Standards Organization*) publicaron el estándar SQL-86, oficialmente conocido como ANSI X3.135-1986 e ISO 9075:1987. Esta estandarización se concibió como un común denominador que debían poseer todas las implementaciones de SQL. Sin embargo, dada su limitación inicial, no incluyó muchas de las características entonces empleadas, que continuaron divergiendo. Este estándar ocupa alrededor de 100 páginas.

En 1989 las organizaciones ANSI e ISO mejoraron el estándar anterior añadiendo la integridad referencial y el soporte para otros lenguajes de programación. Este estándar fue oficialmente denominado ANSI X3.135-1989 e ISO/IEC 9075:1989 e informalmente conocido como SQL-89. Este estándar ocupa alrededor de 120 páginas.

Durante este tiempo X/Open publicó una especificación de SQL que no estaba ligada a los estándares anteriores, sino que reflejaba los productos existentes de sus participantes y accionistas. Finalmente, algunos años después la organización X/Open se alineó junto a los estandares ANSI e ISO.

En 1992 ANSI e ISO publicaron una nueva revisión del estándar SQL, conocido oficialmente como X3.135-1992 e ISO/IEC 9075:1992 e informalmente como SQL-92 o SQL2. Este estándar clasifica todas las características del lenguaje en varios niveles: *Entry SQL*, *Transitional SQL*, *Intermediate SQL* y *Full SQL*. X/Open aceptó el estándar de nivel más bajo, el *Entry SQL*, y diversas características del *Intermediate SQL*. Este estándar ocupa alrededor de 600 páginas.

El estándar SQL ha continuado evolucionando hasta hoy en día. Como una revisión y ampliación del estándar SQL-92 surgió el estándar SQL-99 o SQL3. Este ya no consta de niveles sino del núcleo (*Core SQL*) y de una parte no nuclear. Posee ciertos aspectos orientados a objetos. Este estándar ocupa alrededor de 2200 páginas.

El estándar SQL-2003 está ya más orientado hacia sistemas relacionales/orientados a objetos. Tanto las especificaciones de este estándar como las del anterior no están libremente disponibles, sino que se deben comprar a las organizaciones ANSI o ISO.

De todas formas, los distintos creadores de SGBD, tanto comerciales como no comerciales, que incluyen SQL no se han distinguido especialmente por su gran tradición en seguir los estándares, sino todo lo contrario. Más concretamente, dicen que siguen el estándar de turno, dicen que incluso aportan diversas características no incluidas en el estándar y, después, en letra más pequeña, dicen que algunas características del estándar no han sido incluidas. En resumen, han tomado del estándar de la fecha solamente lo que han querido y han añadido cuantas cosas han creído conveniente, incluso a veces sin respetar la sintaxis original propuesta en el estándar.

De hecho, a fecha de hoy algunos de los productos comerciales más vendidos y desarrollados por empresas consideradas muy serias aún no cumplen completamente el estándar SQL-92.

1.4 Base de datos de ejemplo

Siempre que se describe el manejo y programación de un sistema de gestión de base de datos, resulta muy conveniente la presentación de diversos ejemplos que muestren en la práctica los distintos conceptos teóricos.

Por ello, en este texto se ha elegido un ejemplo de base de datos, no muy complejo, pero sí lo suficiente para que se puedan estudiar en la práctica todos los conceptos más importantes. Este ejemplo es bastante real, aunque para su uso práctico en el mundo real podrían realizarse algunas simplificaciones y eliminarse algunas tablas, como la de pueblos y provincias. No obstante, se han dejado ambas dado que permiten practicar los distintos conceptos muy fácilmente con una escasa dificultad teórica y permiten proponer una gran diversidad de ejercicios aplicados.

La comprensión de este ejemplo es crucial para poder entender y asimilar mejor los capítulos siguientes. Por ello, se recomienda una lectura atenta de las distintas tablas que van a componer este ejemplo.

El ejemplo elegido va a ser el control de *stocks* y facturación de una determinada empresa. Su relativa sencillez, a la par que su posible uso profesional en la pequeña empresa, nos han llevado a elegirlo entre todos los posibles.

Seguidamente se presentan y describen las distintas tablas que lo componen. Para cada tabla se presenta su nombre y las columnas de que consta entre paréntesis. Las claves primarias aparecen subrayadas. Las claves ajenas están en cursiva.

- Tabla **provincias(codpro, nombre)**: Esta tabla almacena las provincias de España, cada una con su código de provincia (clave primaria) y su nombre.
- Tabla **pueblos(codpue, nombre, *codpro*)**: Almacena los pueblos de España o, por lo menos, aquéllos donde tenemos clientes. Para cada pueblo se dispone de su código de pueblo (clave primaria), su nombre y el código de la provincia a la que pertenece (clave ajena).
- Tabla **clientes(codcli, nombre, direccion, codpostal, *codpue*)**: Almacena información sobre los clientes de la empresa. Para cada cliente se dispone de su código de cliente (clave primaria), su nombre, su dirección, su código postal y el código de pueblo donde reside (clave ajena).
- Tabla **vendedores(codven, nombre, direccion, codpostal, *codpue, codjefe*)**: Almacena información sobre los vendedores de la empresa. Para cada vendedor se dispone de su código de vendedor (clave primaria), su nombre, su dirección, su código postal, el código de pueblo donde reside (clave ajena a la tabla pueblos) y el código de su jefe inmediato superior (clave ajena a la misma tabla de vendedores).
- Tabla **articulos(codart, descrip, precio, stock, stock_min)**: Almacena información sobre los artículos que ofrece la empresa y sus cantidades disponibles en el almacén (stocks). Para cada artículo se dispone de su código de artículo específico (clave primaria), su descripción, su precio actual, su stock y su stock mínimo, es decir, el valor umbral por debajo del cual se debe reponer.
- Tabla **facturas(codfac, fecha, *codcli*, *codven*, iva, dto)**: Almacena toda la información sobre las facturas, excepto sus líneas. Como en cada factura el número de líneas es variable, todas las líneas de todas las facturas se almacenan juntas en otra tabla. Para cada factura en esta tabla se guarda su código de factura (clave primaria), su fecha, el código del cliente que ha realizado la compra (clave ajena), el código del vendedor que ha realizado la venta (clave ajena), el iva aplicado y el descuento global de la factura.
- Tabla **lineas_fac(*codfac*, linea, cant, *codart*, precio, dto)**: Almacena información sobre las líneas de las facturas. Para cada línea se

dispone del código de factura a la que pertenece (clave ajena), su número de línea, la cantidad de la línea, el código del artículo vendido (clave ajena), el precio al que se vende el artículo y el descuento que se debe aplicar en la línea. No hay que confundir este descuento, cuyo ámbito de aplicación es la línea, con el descuento global de la factura, el cual se halla obviamente en la tabla de facturas. La clave primaria de esta tabla va a ser la combinación del código de factura y del número de línea pues, por ejemplo, sólo existirá una única tercera línea de la factura 15.

Los nombres de las tablas y de sus columnas se han escrito sin acentuar para evitar problemas con algunos sistemas de gestión de bases de datos.

A continuación se muestra diversa información sobre las columnas de las tablas: si aceptan nulos y su tipo de datos (o dominio). Si en la segunda columna aparece el texto **not null**, entonces la columna no acepta nulos. La tercera columna muestra el tipo de datos: **VARCHAR2(x)** significa una tira de hasta **x** caracteres de longitud.

Tabla Provincias

Columna	¿Nulo?	Tipo de Datos
codpro	not null	VARCHAR2(2)
nombre	not null	VARCHAR2(30)

Como se puede ver, el código de la provincia es una tira de dos caracteres. Tanto el código como el nombre no aceptan nulos.

Tabla Pueblos

Columna	¿Nulo?	Tipo de Datos
codpue	not null	VARCHAR2(5)
nombre	not null	VARCHAR2(40)
codpro	not null	VARCHAR2(2)

Se puede observar que el código de pueblo es una tira de 5 caracteres. Las tres columnas no aceptan nulos.

Tabla Clientes

Columna	¿Nulo?	Tipo de Datos
codcli	not null	NUMBER(5)
nombre	not null	VARCHAR2(50)
direccion	not null	VARCHAR2(50)
codpostal		VARCHAR2(5)
codpue	not null	VARCHAR2(5)

En la tabla de clientes el código de cliente es un número de hasta 5 dígitos. La única columna que acepta nulos es el código postal (éste puede ser desconocido).

Tabla Vendedores

Columna	¿Nulo?	Tipo de Datos
codven	not null	NUMBER(5)
nombre	not null	VARCHAR2(50)
direccion	not null	VARCHAR2(50)
codpostal		VARCHAR2(6)
codpue	not null	VARCHAR2(5)
codjefe	not null	NUMBER(5)

En la tabla de vendedores el código de cliente es también un número de hasta 5 dígitos. La única columna que acepta nulos es el código postal (desconocido).

Tabla Articulos

Columna	¿Nulo?	Tipo de Datos
codart	not null	VARCHAR2(8)
descrip	not null	VARCHAR2(40)
precio	not null	NUMBER(7,2)
stock		NUMBER(6)
stock_min		NUMBER(6)

En la tabla de artículos el código de artículo es una tira de hasta 8 caracteres. El precio es un número de hasta 7 dígitos, dos de los cuales son la parte fraccionaria (los céntimos de euros). Las columnas stock y stock_min son las únicas que aceptan nulos.

Tabla Facturas

Columna	¿Nulo?	Tipo de Datos
codfac	not null	NUMBER(6)
fecha	not null	DATE
codcli		NUMBER(5)
codven		NUMBER(5)
iva		NUMBER(2)
dto		NUMBER(2)

En la tabla de facturas el código de factura es un número de hasta 6 dígitos. La fecha es de tipo DATE. El resto de columnas aceptan valores nulos. El código de cliente o de vendedor puede ser nulo (valor desconocido). Si el descuento es nulo, se entiende que es cero. Lo mismo ocurre con el iva que también acepta nulos.

Tabla Lineas_fac

Columna	¿Nulo?	Tipo de Datos
codfac	not null	NUMBER(6)
linea	not null	NUMBER(2)
cant	not null	NUMBER(5)
codart	not null	VARCHAR2(8)
precio	not null	NUMBER(7,2)
dto		NUMBER(2)

En la tabla de líneas de facturas la cantidad, el precio y el descuento pueden ser nulos. Si el descuento es nulo, se entiende que es cero.

El resto de este texto va a estar basado, pues, en este ejemplo. Por ello, se recomienda encarecidamente un esfuerzo especial en la comprensión de las distintas tablas anteriores y sus implicaciones, lo cual no debe resultar por otro lado muy costoso.

2 INICIACIÓN A SQL

El objetivo de este capítulo es iniciar brevemente al lector en la sentencia de recuperación de datos del lenguaje SQL.

2.1 Iniciación a la sentencia select

La sentencia **select** más sencilla consta de dos cláusulas: la cláusula **select** y la cláusula **from**. Habitualmente cada una se escribe en una línea distinta para mejorar la legibilidad, aunque nada impide escribirlas en una única línea.

Su formato es el siguiente:

```
select * | columna1, columna2, columna3,...
from   tabla;
```

Toda sentencia debe terminar obligatoriamente con el carácter punto y coma.

La cláusula **select** permite especificar qué información se desea obtener. Se pueden poner tantas columnas de la tabla como se desee. Además, pueden ponerse expresiones de una o más columnas de la tabla. El carácter * indica que se muestren todas las columnas. En términos matemáticos, esta cláusula es similar a realizar una operación de proyección.

La cláusula **from** permite indicar de qué tabla se deben extraer los datos.

Cuando el SGBD ejecuta una sentencia con estas dos cláusulas, examina en primer lugar la cláusula **from** para saber qué tablas debe procesar y, a continuación, examina la cláusula **select** para determinar qué información se desea mostrar a partir de dichas tablas.

❖ **Ejercicio**: Mostrar el código y nombre de las provincias.

Solución: A continuación se muestras tres posibles soluciones que obtienen exactamente el mismo resultado.

```
select * from provincias;

select codpro, nombre from provincias;
```

```
select codpro, nombre
from   provincias;
```

❖ **Ejercicio**: Mostrar el nombre y después el código de las provincias.

Solución:

```
select nombre, codpro
from   provincias;
```

❖ **Ejercicio**: Mostrar el código de artículo y el doble del precio de cada artículo.

Solución:

```
select codart, precio * 2
from   articulos;
```

❖ **Ejercicio**: Mostrar el código de factura, número de línea e importe de cada línea (sin considerar impuestos ni descuentos).

Solución:

```
select codfac, linea, cant * precio
from   lineas_fac;
```

❖ **Ejercicio**: Mostrar el código de factura, número de línea e importe de cada línea.

Solución:

```
select codfac, linea,
       cant * precio * ( 1 - dto / 100 )
from   lineas_fac;
```

2.2 Modificador distinct

La sentencia **select** admite opcionalmente el uso del modificador **distinct** en su primera cláusula, tal como se muestra a continuación:

```
select [ distinct ] * | columna1, columna2, columna3,...
from   tabla;
```

Esta palabra es un modificador y no es una función por lo que no necesita paréntesis. Su función es obvia: eliminar del resultado aquellas filas o valores repetidos.

❖ **Ejercicio**: Mostrar los distintos tipos de ivas aplicados en las facturas.

Solución:

```
select distinct iva
from   facturas;
```

2.3 Restricción en las filas

En numerosas ocasiones resulta conveniente restringir o seleccionar sólo algunas filas que cumplen una determinada condición de entre todas las existentes en una tabla. Esta operación se lleva a cabo con la cláusula **where**, la cual es opcional y de aparecer, debe hacerlo siempre tras la cláusula **from**.

A esta palabra le debe seguir una expresión booleana o lógica que devuelva cierto o falso. Esta condición se evalúa para cada fila. Si para una fila da cierto, entonces la fila aparece en el resultado. En caso contrario (la condición devuelve falso o desconocido), la fila es ignorada y se descarta del resultado.

La expresión booleana puede incluir cualquier combinación correcta de otras expresiones lógicas conectadas con los operadores lógicos **and** y **or**.

Cuando el SGBD ejecuta una sentencia con las tres cláusulas **select**, **from** y **where**, examina en primer lugar la cláusula **from** para saber qué tablas debe procesar, a continuación realiza una criba dejando sólo aquellas filas que cumplan la condición o condiciones de la cláusula **where** y, finalmente, examina la cláusula **select** para determinar qué información se desea mostrar a partir de dichas filas.

❖ **Ejercicio**: Mostrar el código y nombre de aquellas provincias cuyo código es menor que '20'.

Solución:

```
select codpro, nombre
from   provincias
where  codpro < '20';
```

❖ **Ejercicio**: Mostrar los distintos tipos de descuentos aplicados por los vendedores cuyos códigos no superan el valor 50.

Solución:

```
select distinct dto
from   facturas
where  codven <= 50;
```

❖ **Ejercicio**: Mostrar el código y descripción de aquellos artículos cuyo stock iguala o supera las 50 unidades.

Solución:

```
select codart, descrip
from   articulos
where  stock >= 50;
```

❖ **Ejercicio**: Mostrar el código de factura y fecha de las facturas con iva 16 y del cliente 100.

Solución:

```
select codfac, fecha
from   facturas
```

```
where  iva = 16
and    codcli = 100;
```

❖ **Ejercicio**: Mostrar el código y fecha de las facturas del cliente 100 tales que tienen iva 16 o descuento 20.

Solución:

```
select codfac, fecha
from   facturas
where  codcli = 100
and    ( iva = 16
or       dto = 20 );
```

❖ **Ejercicio**: Mostrar el código de factura y el número de línea de aquellas líneas de facturas cuyo importe supera los 100 euros, sin considerar descuentos ni impuestos.

Solución:

```
select codfac, linea
from   lineas_fac
where  cant * precio > 100.0;
```

2.4 Ejecución de sentencias

Cuando el SGBD ejecuta una sentencia, lo hace siempre de la misma forma, aplicando el siguiente método:

1. Examina la cláusula **from** para saber con qué tablas va a trabajar.
2. Si existe la cláusula **where**, realiza una criba dejando sólo aquellas filas que cumplan la condición o condiciones de esta cláusula.
3. A partir del contenido de la cláusula **select** determina qué información se desea mostrar de las filas previamente seleccionadas.
4. Si existe el modificador **distinct**, elimina las filas repetidas del resultado anterior.

2.5 Ejercicios

❖ **Ejercicio 2.1**: Mostrar el código de artículo y la cantidad de las líneas de la factura cuyo código es 105.

Solución:

```
select codart, cant
from   lineas_fac
where  codfac = 105;
```

❖ **Ejercicio 2.2**: Mostrar el código de artículo y el precio de aquellos artículos cuyo precio supera los 2,05 euros y cuyo stock supera las 100 unidades.

Solución:

```
select codart, precio
from   articulos
where  precio > 2.05
and    stock > 100;
```

❖ **Ejercicio 2.3**: Mostrar el código de artículo y la cantidad de aquellas líneas cuyo descuento es igual a 10 o cuyo precio supera los 5,05 euros.

Solución:

```
select codart, cant
from   lineas_fac
where  dto = 10
or     precio > 5.05;
```

❖ **Ejercicio 2.4**: Tipos de descuentos aplicados en las facturas del cliente 222.

Solución:

```
select distinct dto
from   facturas
where  codcli = 222;
```

❖ **Ejercicio 2.5**: Mostrar el código de artículo, la cantidad, el precio unitario y el importe una vez aplicado el descuento de cada una de las líneas de la factura 325.

Solución:

```
select codart, cant, precio,
       cant * precio * ( 1 - dto/100 )
from   lineas_fac
where  codfac = 325;
```

2.6 Autoevaluación

❖ **Ejercicio 1**: Mostrar el código y nombre de aquellos vendedores cuyo jefe tiene el código 125.

❖ **Ejercicio 2**: Mostrar el código y descripción de aquellos artículos cuyo stock en el almacén supera los 100 euros.

❖ **Ejercicio 3**: Mostrar el código, sin que salgan repetidos, de los artículos vendidos en las facturas con código inferior a 100.

3 FUNCIONES Y OPERADORES ESCALARES

Este capítulo presenta las funciones y operadores escalares del lenguaje SQL. Son funciones que devuelven un valor modificado por cada valor inicial. Dado el elevado número de funciones y operadores que contienen tanto el estándar como las distintas implementaciones, en este capítulo se describen únicamente los más importantes. En Internet y en los manuales de referencia se pueden encontrar listas más exhaustivas.

En el caso de las funciones y operadores escalares la variación entre los distintos SQL es bastante alta. Afortunadamente, dado que estas funciones no son excesivamente complicadas, pasar de una implementación de SQL a otra no cuesta demasiado tiempo. Es más, en muchos casos se tratan de los mismos conceptos pero con nombres distintos. En el resto del capítulo se van a describir las funciones y operadores escalares más importantes tanto del estándar SQL-99 como de varias implementaciones comerciales de SQL.

3.1 Expresiones y Tipos

En la mayor parte de los casos SQL permite emplear una expresión en lugar de una columna. Las expresiones son una parte crucial dentro de una consulta. Las expresiones pueden construirse empleando los operadores matemáticos habituales y también las funciones escalares.

Toda expresión tiene un tipo determinado en su resultado. Es conveniente no mezclar los tipos para evitar problemas. Puede que algunos sistemas sean más laxos y permitan las mezclas obteniendo el resultado esperado, pero otros sistemas pueden ser más ortodoxos y el resultado puede ser muy distinto.

Un error muy habitual es la comparación de tiras de caracteres y números. Algunos sistemas realizan una conversión previa internamente y el resultado es el esperado. Sin embargo, en otros sistemas esto no es así y se puede producir un error o devolver un resultado no esperado.

Otro error frecuente es operar con valores enteros (sin decimales) cuando en realidad se desea trabajar con valores reales (con decimales). En algunos sistemas la operación 1/3 devuelve cero pues tanto el numerador como el denominador son enteros y, por tanto, se aplica la división entera. Si se desea trabajar con números reales puede realizarse con una operación de conversión explícita o bien añadiendo una parte fraccionaria. Si se desea realizar una división real, debería escribirse la instrucción 1.0/3.0.

❖ **Ejercicio**: Mostrar las provincias con código mayor que 20.

```
select codpro, nombre
from   provincias
where  codpro > 20;
```

Solución: Esta sentencia puede funcionar correctamente en algunos sistemas, pero no es aconsejable escribirla de esta forma pues en la cláusula **where** se compara la columna **codpro** (que es una tira de caracteres con dos caracteres) y un número entero. Sería mucho más conveniente reescribirla de la siguiente forma:

```
select codpro, nombre
from   provincias
where  codpro > '20';
```

3.2 Operadores de comparación

El lenguaje SQL incluye los operadores habituales de comparación: =, <>, >, >=, <, <=, etc. No obstante, además de éstos, suele incluir diversos operadores avanzados que simplifican y reducen bastante las expresiones si se usan adecuadamente, como son los operadores **between**, **in** y **like**.

3.2.1 Operador between

Este operador, al igual que el siguiente, no introduce ninguna operación que no se pueda realizar utilizando los operadores de comparación tradicionales, pero permite reducir significativamente las expresiones.

Su funcionamiento es muy sencillo: **a between b and c** devuelve cierto si **a** se halla entre **b** y **c** ambos inclusive, y falso en caso contrario. Es decir, devuelve cierto si **a** es mayor o igual que **b** y **a** es menor o igual que **c**. La siguiente línea resume su funcionamiento:

a between b and c equivale a **(b <= a) and (a <= c)**.

El funcionamiento del operador **not between** es, obviamente, justo el contrario que el anterior. La siguiente línea resume su funcionamiento:

a not between b and c equivale a **(a < b) or (c < a)**.

Su funcionamiento es completamente idéntico en SQL-99, Access, Oracle, PostgreSQL y MySQL.

❖ **Ejercicio**: Escribir una expresión que devuelva el código de cliente y el nombre de aquellos clientes cuyos códigos se encuentran comprendidos entre 100 y 200, sin incluir éstos.

Solución:

```
select codcli, nombre
from   clientes
where  codcli between 101 and 199;
```

❖ **Ejercicio**: Escribir una expresión que devuelva todas las columnas de los artículos cuyo stock no se halla entre el stock mínimo menos 500 unidades y el stock mínimo más 500 unidades.

Ayuda: Uso del operador *not between* y del carácter * en la cláusula *select*.

Solución:

```
select *
from   articulos
where  stock not between
           stock_min - 500 and stock_min + 500;
```

3.2.2 Operador in

Su funcionamiento es muy sencillo: **a in (b, c, d,..., z)** devuelve cierto si **a** es igual a alguno de los valores incluidos en la lista entre paréntesis (**b, c, d,..., z**) y devuelve falso en caso contrario. La siguiente línea resume su funcionamiento:

a in (b, c, d) equivale a **(a = b) or (a = c) or (a = d)**.

El funcionamiento del operador **not in** es justo el contrario: **a not in (b, c, d,..., z)** devuelve cierto si **a** no es igual a ninguno (o sea, es distinto de todos) de los valores incluidos en la lista entre paréntesis (**b, c, d,..., z**) y devuelve falso en caso contrario. La siguiente línea resume su funcionamiento:

a not in (b, c, d) equivale a **(a <> b) and (a <> c) and (a <> z)**.

Su funcionamiento es completamente idéntico en los distintos SQL de SQL-99, Access, Oracle, PostgreSQL y MySQL.

❖ **Ejercicio**: Escribir una expresión que devuelva el código y nombre de los pueblos pertenecientes a la comunidad valenciana (Alicante tiene el código de provincia '03'; Castellón, el '12' y Valencia, el '46').

Solución:

```
select codpue, nombre
from   pueblos
where  codpro in ( '03', '12', '46' );
```

3.2.3 Operador like

El operador **like** de SQL permite comparar cadenas de caracteres usando comodines. La principal diferencia en el uso de este operador en las distintas implementaciones de SQL radica en los comodines:

- Cualquier tira de caracteres de cualquier longitud se representa con el carácter comodín "%" en el estándar SQL-99, en Oracle y en PostgreSQL. En cambio, en Access se emplea el comodín "*".
- Cualquier carácter se representa con el carácter comodín "_" en el estándar SQL-99, en Oracle y en PostgreSQL. En cambio, en Access se emplea el comodín "?".

En resumen, la nomenclatura de comodines de Oracle es más estándar, pero la nomenclatura de Access es más popular e intuitiva.

Además, hay que recordar que el uso de mayúsculas y minúsculas difiere entre Access y Oracle: el primero no distingue entre mayúsculas y minúsculas, mientras que el segundo sí. Por tanto, en Access la expresión **like "*García*"** devolverá todos aquellos García, independientemente de cómo estén escritos (con cualquier combinación indistinta de mayúsculas y minúsculas).

El operador **not like** trabaja justo al revés que el operador **like**.

❖ **Ejercicio**: ¿Qué devuelven las siguientes sentencias?

```
nombre like 'a_b%'
nombre not like 'a_b%'
```

Solución: La primera devuelve cierto si el primer carácter de nombre es una 'a', el tercero es una 'b' y tras éste aparece cualquier tira de caracteres. La segunda sentencia devuelve justo lo contrario de la primera.

3.3 Operadores y funciones de elección

Seguidamente se muestran unos operadores denominados de elección pues permiten devolver un valor u otro en función de una expresión. Se presenta en primer lugar el operador **case**, perteneciente al estándar SQL-99 y que aparece también en Oracle 9i y en PostgreSQL. Sin embargo, no se halla en Oracle 8i ni en Access.

A continuación se presentan diversos operadores de Access y Oracle no estándares que permiten realizar funciones parecidas. Se incluyen estos operadores no estándares debido a que no todos los sistemas disponen del operador **case**.

3.3.1 Operador estándar case

El operador estándar de elección entre múltiples valores es el operador **case**. Tiene dos formas de funcionamiento algo distintas, pero muy intuitivas. A continuación se describen ambas.

La siguiente expresión devuelve el valor **ret1** si expresión se evalúa a **val1** y a continuación termina. Si no, devuelve el valor **ret2** si expresión se evalúa a **val2** y así sucesivamente. Si ninguna de las igualdades anteriores se ha cumplido, se devuelve **retn**. Si no existe cláusula **else** y ninguna de las igualdades anteriores se ha cumplido, entonces devuelve el valor **null**.

```
case expresión when val1 then ret1
               when val2 then ret2
               ...
                         else retn
end
```

La siguiente expresión devuelve el valor **ret1** si **expresion1** se evalúa a cierto y a continuación termina. Si no, devuelve el valor **ret2** si expresión se evalúa a cierto y así sucesivamente. Si ninguna de las expresiones anteriores se ha evaluado a cierto, se devuelve **retn**. Si no existe cláusula **else** y ninguna de las expresiones anteriores se ha evaluado a cierto, entonces devuelve el valor **null**.

```
case when expresión1 then ret1
     when expresión2 then ret2
     ...
                      else retn
end
```

❖ **Ejercicio**: Escribir una sentencia que devuelva el código de factura y el texto 'normal' cuando el iva vale 16, el texto 'reducido' cuando el iva vale 7 y el texto 'otros' en cualquier otro caso para las facturas con descuento del 20 %.

Solución: Seguidamente se presentan dos soluciones equivalentes.

```
select codfac, case iva when 16 then 'normal'
                        when  7 then 'reducido'
                                else 'otros'
               end
from   facturas
where  dto = 20;

select codfac, case when iva = 16 then 'normal'
                    when iva =  7 then 'reducido'
                                  else 'otros'
               end
from   facturas
where  dto = 20;
```

❖ **Ejercicio**: Escribir una sentencia que muestre el código de cada factura, su fecha y al lado el texto "descuento super" si el descuento supera el 20%. Si no supera dicho valor, debe aparecer el texto "descuento normal".

Solución: La expresión case sólo se puede escribir de una forma.

```
select codfac, fecha,
       case when dto > 20 then 'descuento super'
                          else 'descuento normal'
       end
from   facturas;
```

3.3.2 Funciones no estándares de elección: decode, iif, switch y choose

Siempre es conveniente utilizar operadores estándares como el **case**. Ahora bien, si el SBGD no soporta este operador se puede recurrir al uso de otras funciones no estándares.

Por ejemplo, aunque en versiones más modernas dicha carencia ya ha sido subsanada, en Oracle 8i no existe el operador **case**. No obstante, en la mencionada versión la función **decode** permite devolver un valor u otro según el resultado de una expresión. La llamada **decode(a, v1, ret1, v2, ret2,..., retn)** devuelve **ret1** si **a** es igual a **v1**; si no lo es y **a** es igual a **v2**, entonces devuelve **ret2** y así sucesivamente. Si el valor de **a** no iguala a ningún valor y el número de parámetros es impar, entonces devuelve el valor nulo. Si el valor de **a** no iguala a ningún valor y el número de parámetros es par, entonces devuelve el último parámetro.

En Access tampoco existe el operador **case**. En cambio, existen tres funciones que permiten elegir y devolver un valor de entre varios. Éstas son:

- Si Inmediato **Iif**. Su funcionamiento es muy sencillo e intuitivo. La llamada a la función **Iif(a, b, c)** devuelve **b** si **a** es cierto y **c** en caso contrario.
  Por ejemplo, si se desea mostrar la tira de caracteres "mayor" si el precio es superior a 100 y "normal" en cualquier otro caso, se debería hacer lo siguiente:
  - En Access:
    ```
    iif( precio>100, "mayor", "normal" )
    ```
  - En Oracle 8i:
    ```
    decode( sign(precio-100), 1, 'mayor', 'normal' )
    ```
- **Switch**. Esta función suele ser menos útil que la función **Iif**. Debe tener un número par de parámetros. Si el primer parámetro es cierto, se devuelve el segundo y se termina; en caso contrario, si el tercer parámetro es cierto, se devuelve el cuarto y se termina; y así sucesivamente. Para devolver un valor final si no se cumple ninguna de las condiciones anteriores, se pone como

penúltimo parámetro (última condición) el valor cierto, con lo cual esta condición se cumple siempre y se devolverá el último parámetro.
Por ejemplo, si se desea mostrar la tira "normal" cuando el iva vale 16, "reducido" cuando el iva vale 7 y "otro" en cualquier otro caso se debería teclear lo siguiente:

- En Access:

```
switch( iva=16, "normal", iva=7, "reducido",
        True, "otro" )
```

- En Oracle 8i:

```
decode( iva, 16, 'normal', 7, 'reducido', 'otro' )
```

- **Choose**. Esta función también tiene una aplicación parecida. Si el primer parámetro se evalúa a uno, devuelve el segundo parámetro; si el primer parámetro se evalúa a dos, devuelve el tercero; y así sucesivamente.
Por ejemplo, si se desea mostrar la tira "efectivo" cuando el modo de pago vale 1, "tarjeta" cuando modo de pago vale 2 y "cheque" cuando el modo de pago vale 3, se debería teclear lo siguiente:

- En Access:

```
choose( modoPago, "efectivo", "cheque", "tarjeta" )
```

- En Oracle 8i:

```
decode( modoPago, 1, 'efectivo', 2, 'cheque',
        3, 'tarjeta')
```

3.4 Manejo de los valores nulos

Muchas veces en la realidad ocurren excepciones que producen falta de información. En el contexto de una base de datos, esta situación se representa dentro de la columna correspondiente por medio de un nulo. Un valor nulo (**null**) dentro de una celda significa que esa información se desconoce, es irrelevante o que no es aplicable a la fila en cuestión. A este respecto es muy importante resaltar que un valor nulo tiene un significado muy diferente de un valor cero, de una cadena en blanco, o de un valor booleano igual a **false**. Un nulo se corresponde con una ausencia de valor que no puede ser comparado ni operado directamente junto con ningún otro valor de ningún tipo de datos.

El procesamiento de los valores nulos es una parte muy empleada en casi todos los entornos debido a que las bases de datos reales contienen numerosos valores nulos en sus celdas que hay que procesar debidamente.

Las funciones más habituales para el procesamiento de valores nulos son dos: la detección del valor nulo y la conversión del valor nulo en un valor más tratable.

3.4.1 Tablas de verdad de los valores nulos

La especificación del estándar SQL no indica cómo se deben manejar los nulos, simplemente propone su utilización y deja el resto de cuestiones abiertas para que las resuelva cada fabricante de software a su manera.

Lo que sí proporciona el estándar SQL son las siguientes tablas de verdad para los operadores AND, OR y NOT. La ausencia de valor de los nulos hace que cualquier comparación de un valor con un nulo devuelva un valor desconocido. Por esta razón, se consideran tres posibles valores (V de verdadero, F de falso y D de desconocido), dando lugar a una lógica de tres valores para SQL. A continuación se muestran las tablas de verdad para los tres operadores lógicos:

and	V	D	F
V	V	D	F
D	D	D	F
F	F	F	F

or	V	D	F
V	V	V	V
D	V	D	D
F	V	D	F

not	
V	F
D	D
F	V

Como ya hemos dicho anteriormente, la ausencia de información asociada a los nulos imposibilita su comparación con el resto de valores de la base de datos. Por ejemplo, un nulo nunca va a ser igual a otro valor, y un nulo tampoco va a ser nunca diferente de otro valor. Ni siquiera se puede decir que dos nulos sean iguales o diferentes entre sí. Únicamente se puede decir que la verdad se desconoce debido a la ausencia de información.

Además los nulos presentan el serio inconveniente de que se propagan, dado que cuando se opera (suma, resta, etc) un nulo con cualquier otro valor siempre da como resultado el valor nulo. Por ejemplo, si se desea calcular el número de artículos por encima del stock mínimo con la consulta **select (stock - stock_min) from articulos**; en la mayoría de sistemas de gestión de bases de datos la existencia de un artículo con un stock mínimo nulo provocará que el resultado de la resta sea igual a nulo, aunque el stock no lo sea.

En el proceso de creación de tablas los diseñadores suelen tener la opción de especificar qué columnas pueden aceptar nulos y cuáles no. Por esta razón, antes de formular sus consultas, los programadores de bases de datos deben considerar la posibilidad de que haya nulos y, cuando sea necesario, deben utilizar los operadores explicados para su detección y conversión.

3.4.2 Detección de valores nulos

La detección de valores nulos se suele realizar en la mayor parte de variantes de SQL (SQL-99, Oracle 9i, MySQL, PostgreSQL, etc.) con los operadores **is null** e **is not null**. En Access la detección de un valor nulo se realiza con la función **IsNull**.

Es importante destacar que los operadores de igualdad y de desigualdad no sirven para comparar con el valor nulo. Así pues, las expresiones **columna = null** y **columna <> null** siempre devolverán desconocido, valga lo que valga la parte izquierda.

Por ejemplo, para detectar si el descuento es nulo, se debería hacer lo siguiente:

- En el estándar y en la mayor parte de implementaciones:

```
dto is null
```

- En Access:

```
IsNull( dto )
```

Por ejemplo, para detectar si el descuento no es nulo, se debería hacer lo siguiente:

- En el estándar y en la mayor parte de implementaciones:

```
dto is not null
```

- En Access:

```
Not IsNull( dto )
```

❖ **Ejercicio**: Escribir una sentencia que muestre los códigos y fechas de aquellas facturas sin código de cliente o sin código de vendedor.

Solución:

```
select codfac, fecha
from   facturas
where  codcli is null
or     codven is null;
```

❖ **Ejercicio**: Se desea escribir una sentencia que muestre el código de los artículos con un stock superior a 50 sabiendo que este campo es nulo en algunas ocasiones. Dadas las tres sentencias siguientes, ¿qué realiza cada una de ellas?

```
select codart
from   articulos
where  stock > 50;
```

```
select codart
from   articulos
where  not(stock <= 50);
```

```
select codart
from   articulos
where  ( stock > 50 )
or     (stock is null);
```

Solución: La primera sentencia devuelve aquellos artículos que efectivamente tienen un stock superior a 50 unidades, haciendo caso omiso de los artículos con un stock nulo o con un stock inferior. La segunda sentencia recupera los mismos artículos que la anterior (pues la negación de desconocido es

desconocido). La tercera sentencia recupera los artículos cuyo stock supera las 50 unidades o es desconocido, es decir, devuelve aquellos artículos que podrían tener un stock superior a 50.

3.4.3 Conversión de valores nulos

La conversión de valores nulos se suele realizar en la mayor parte de variantes de SQL (SQL-99, Oracle 9i y superiores, MySQL, PostgreSQL, etc.) con la función **coalesce**. En cambio, en Oracle 8i se debe emplear la función **nvl** y en Access, la función **Nz**.

Por ejemplo, para devolver un cero si el descuento es nulo, se debería hacer lo siguiente:

- En el estándar y en la mayor parte de implementaciones (incluyendo Oracle 9i y superiores):

```
coalesce( dto, 0 )
```

  En realidad, el formato de esta función admite *n* parámetros. Su funcionamiento en el caso generalizado es muy sencillo: devuelve el primer parámetro no nulo comenzando desde la izquierda. El comportamiento con dos parámetros es una simplificación del caso general.

- En Oracle 8i:

```
nvl( dto, 0 )
```

- En Access:

```
Nz( dto, 0 )
```

  En este caso el segundo parámetro es opcional: si no está, devuelve cero o la tira vacía, dependiendo de ciertos criterios internos de Access. No obstante, se recomienda encarecidamente el uso explícito del segundo parámetro debido a que estos criterios pueden fallar, devolviendo por ejemplo la tira vacía cuando se espera un cero.

❖ **Ejercicio**: Escribir una sentencia que muestre los códigos de aquellos artículos (sin mostrar repetidos) vendidos alguna vez sin descuento en sus líneas de facturas. Se considera que un artículo no tiene descuento cuando éste es cero o nulo.

Solución:

```
select distinct codart
from   lineas_fac
where  coalesce( dto, 0 ) = 0;
```

❖ **Ejercicio**: Escribir una sentencia que muestre código, la fecha y el descuento de las facturas sin iva (iva nulo o cero), visualizando un cero en aquellas facturas cuyo descuento sea nulo.

Ayuda: Uso de la función *coalesce*.

Solución:

```
select codfac, fecha, coalesce( dto, 0 )
from   facturas
where  coalesce( iva, 0 ) = 0;
```

3.5 Conversiones de datos

La conversión de información desde unos tipos de datos a otros tipos de datos suele ser una labor muy frecuente en el procesamiento automático de la información. El estándar de SQL proporciona diversas funciones para convertir unos tipos a otros. Las más empleadas son las funciones **to_number**, **to_char** y **to_date**. En este apartado se describen las dos primeras, dejándose la tercera para el apartado específico de procesamiento de fechas.

3.5.1 Función to_char

La función **to_char(número, formato)** convierte el número recibido como primer argumento según el formato recibido en el segundo argumento en una tira de caracteres. Si se omite el formato, el número es convertido a una tira de caracteres lo suficientemente grande para representarlo.

En Oracle se usa muy frecuentemente la función **to_char** para extraer una o varias partes de una fecha, como se describe en el apartado siguiente.

Por ejemplo, la siguiente sentencia:

```
select to_char( 123, '99,999' ),
       to_char( 1234, '99,999' )
from   dual;
```

devuelve las tiras de caracteres '123' y '1,234'.

La tabla **dual** suele venir instalada por defecto con Oracle. Es una tabla con una única fila y una única columna que se suele emplear para examinar el funcionamiento de algunas funciones de SQL.

En los apéndices se puede encontrar una tabla con los caracteres más habituales que se pueden emplear en el formato.

3.5.2 Función to_number

La función **to_number(tira, formato)** convierte a número la tira de caracteres recibida en el primer argumento según el formato recibido en el segundo argumento. Se puede omitir el segundo parámetro si el formato sólo tiene dígitos.

Como se ve, la principal utilidad es convertir una tira de caracteres en un número. Esta conversión se suele emplear en aquellas funciones que devuelven una tira de caracteres y se necesita convertirlas a número para procesarlos (sumarlos, restarlos, etc.).