

FUNDAMENTOS DE

**Mc
Graw
Hill**

Tercera edición

SQL

Cubre:
SQL:2006 ANSI/ISO standard • SQL/XML •
La última versión de los programas RDBMS.

Andy Oppel
Robert Sheldon



Fundamentos de **SQL**

Tercera edición

Andy Oppel
Robert Sheldon

Traducción

Carlos Fabián Jiménez Castillo

Traductor profesional



MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • MADRID • NUEVA YORK
SAN JUAN • SANTIAGO • SÃO PAULO • AUCKLAND • LONDRES • MILÁN • MONTREAL
NUEVA DELHI • SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

Director editorial: Fernando Castellanos Rodríguez
Editor: Miguel Ángel Luna Ponce
Supervisor de producción: Zeferino García García

FUNDAMENTOS DE SQL
Tercera edición

Prohibida la reproducción total o parcial de esta obra,
por cualquier medio, sin la autorización escrita del editor.



Educación

DERECHOS RESERVADOS © 2010, respecto a la primera edición en español por
McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V.

A Subsidiary of The McGraw-Hill Companies, Inc.

Corporativo Punta Santa Fe,
Prolongación Paseo de la Reforma 1015, Torre A
Piso 17, Colonia Desarrollo Santa Fe,
Delegación Álvaro Obregón
C.P. 01376, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

ISBN: 978-607-15-0251-3

Translated from the 3rd English edition of

SQL: A Beginner's Guide

By: Andrew Oppel and Robert Sheldon

Copyright © 2009 by The McGraw-Hill Companies. All rights reserved.

ISBN: 978-0-07-154864-9

1234567890

109876543210

Impreso en México

Printed in Mexico

Acerca de los autores

Andrew (Andy) J. Oppel es graduado de la Latin School de Maryland y la Transylvania University (Lexington, Kentucky) donde obtuvo un grado en ciencias de la computación en 1974. Desde entonces ha sido empleado continuamente en una amplia variedad de puestos sobre tecnologías de la información, incluyendo programador, programador/analista, arquitecto en sistemas, administrador de proyectos, administrador senior de base de datos, administrador del grupo de bases de datos, asesor, diseñador de bases de datos, modelador de datos y arquitecto de datos. Además, ha sido instructor de medio tiempo para un campus de la Universidad de California (Berkeley) durante más de 20 años, y recibió el premio de Instructor Honorario durante el año 2000. Su labor en la enseñanza incluye el desarrollo de tres cursos para el campus de la Universidad de California, “Conceptos de Sistemas de Administración de Bases de Datos”, “Introducción a los Sistemas de Administración de Bases de Datos Relacionales”, y “Modelado de Datos y Diseño de Bases de Datos”. También obtuvo la certificación Oracle 9i Database Associate en 2003. Actualmente se encuentra empleado como modelador de datos senior para Blue Shield de California. Muy aparte de los sistemas de cómputo, Andy disfruta la música (la guitarra y el canto), el radio amateur (vicedirector de la Pacific Division, American Radio Relay League) y el fútbol (instructor de árbitros, U.S. Soccer).

Andy ha diseñado e implementado cientos de bases de datos para un amplio rango de aplicaciones, incluyendo investigación médica, para la banca, seguros, fabricación de instrumentaria, telecomunicaciones, comunicaciones inalámbricas y recursos humanos. Es el autor de *Databases Demystified* (McGraw-Hill/Osborne, 2004) y *SQL Demystified* (McGraw-Hill/Osborne, 2005). Su experiencia en productos de bases de datos incluye IMS, DB2, Sybase, Microsoft SQL Server, Microsoft Access, MySQL y Oracle (versiones 7, 8, 8i, 9i y 10g).

Robert Sheldon ha trabajado como asesor y escritor técnico durante muchos años. Como asesor, ha administrado el desarrollo y mantenimiento de aplicaciones basadas en Web y aplicaciones servidor-cliente y las bases de datos que soportan a esas aplicaciones. Ha diseñado e implementado diferentes bases de datos de Access y SQL y ha utilizado SQL para construir bases de datos, crear y modificar objetos de las bases de datos, consultar y modificar los datos, y para solucionar problemas relacionados con los datos y con los sistemas. Robert también ha escrito o coescrito ocho libros sobre diferentes tecnologías de red y de servidor, uno de los cuales recibió un certificado al mérito del Puget Sound Chapter de la Society for Technical Communication. Además, dos de los libros que Robert ha escrito se enfocan ex-

clusivamente al diseño e implementación de SQL Server. Robert también ha escrito y editado una gran cantidad de material relacionado con las bases de datos SQL y otras tecnologías de computación. Su trabajo en escritura no sólo incluye material sobre la industria de la computación (también ha escrito un poco de todo, desde artículos de noticias hasta trabajos en documentación legal) y ha recibido dos premios de la Colorado Press Association.

Acerca del revisor técnico

James Seymour es graduado de la universidad de North Carolina en Chapel Hill con grado en historia y ciencias políticas, y de la universidad de Kentucky con una maestría en historia. Se involucró por primera vez con tecnología computacional en 1965 con el entorno de la computadora central en North Carolina. En el ejército de Estados Unidos durante la guerra de Vietnam perteneció al pequeño equipo que trabajó con la puesta en marcha de la computadora central del Pentágono para diferentes escenarios de estrategia militar.

Desde 1972 está involucrado en diferentes entornos de computación con el segundo proyecto punto de venta y control de inventarios en la industria de venta al público, programas analíticos e iniciativas de bases de datos para industrias de seguros y ayuda social, la puesta en marcha de controles de pérdida, y otros proyectos de control de inventario y rastreo de ventas a través de diferentes industrias.

De 1987 a 1995, James fue instructor de manejo de bases de datos en el sistema de universidades comunitarias en el estado de Kentucky. Con esta capacidad creó los primeros cursos de manejo de bases de datos y de programación en C en el estado de Kentucky y ayudó a las entidades tanto públicas como privadas con sus necesidades urgentes de entrenamiento, incluyendo la programación de sistemas guía para los misiles crucero para Tormenta del Desierto.

Antes de 1985 fue administrador de sistemas, administrador de redes, programador y administrador de bases de datos. A partir de 1985, James ha sido administrador senior de bases de datos y ha trabajado principalmente con DBMS de DB2 y Oracle en múltiples plataformas, incluyendo SQL Server desde la versión 7.0. Actualmente es el administrador senior de bases de datos y arquitecto de datos para una compañía del Fortune 100 supervisando proyectos mayores en Estados Unidos, Canadá y el Reino Unido.

Contenido

ACERCA DE LOS AUTORES	iii
AGRADECIMIENTOS	xi
INTRODUCCIÓN	xi

Parte 1 Bases de datos relacionales y SQL

1 Introducción a las bases de datos relacionales y a SQL	3
Entienda las bases de datos relacionales	4
El modelo relacional	5
Aprenda acerca de SQL	15
La evolución de SQL	15
Tipos de instrucciones de SQL	18
Tipos de ejecución	19
El estándar SQL frente a las implementaciones de producto	21
2 Trabajo con el entorno SQL	29
Entienda el entorno SQL	30
Entienda los catálogos SQL	32
Esquemas	34
Objetos de esquema	35
¿Qué es una base de datos?	37
Nombrado de objetos en un entorno SQL	40
Nombres calificados	41
Creación de un esquema	42
Creación de una base de datos	44

3 Creación y modificación de tablas.....	49
Creación de tablas en SQL.....	50
Especificación de los tipos de datos en una columna.....	54
Tipos de datos de cadena.....	55
Tipos de datos numéricos.....	57
Tipos de datos de fecha y hora.....	58
Tipo de datos de intervalo.....	60
Tipo de datos booleanos.....	61
Utilice tipos de datos SQL.....	62
Creación de tipos definidos por el usuario.....	63
Especificación de los valores predeterminados en una columna.....	64
Eliminación de tablas en SQL.....	69
4 Implementación de la integridad de datos.....	73
Entienda las restricciones de integridad.....	74
Utilice restricciones NOT NULL.....	76
Añada restricciones UNIQUE.....	77
Añada restricciones PRIMARY KEY.....	79
Añada restricciones FOREIGN KEY.....	83
La cláusula MATCH.....	88
La cláusula <acción referencial desencadenada>.....	89
Defina restricciones CHECK.....	95
Defina afirmaciones.....	97
Creación de dominios y restricciones de dominio.....	98
5 Creación de vistas en SQL.....	103
Añada vistas a la base de datos.....	104
Definición de vistas de SQL.....	108
Creación de vistas actualizables.....	114
Utilice la cláusula WITH CHECK OPTION.....	116
Eliminación de vistas de la base de datos.....	117
6 Gestión de seguridad en la base de datos.....	123
Entienda el modelo de seguridad de SQL.....	124
Sesiones SQL.....	126
Acceda a objetos de base de datos.....	128
Creación y eliminación de roles.....	130
Otorgue y revoque privilegios.....	131
Revoque privilegios.....	135
Otorgue y revoque roles.....	137
Revoque roles.....	138

Parte II Acceso y modificación de datos

7 Consulta de datos de SQL	145
Utilice la instrucción SELECT para la recuperación de datos.	146
La cláusula SELECT y la cláusula FROM	147
Utilice la cláusula WHERE para definir condiciones de búsqueda	152
Defina la cláusula WHERE	156
Utilice la cláusula GROUP BY para agrupar los resultados de una consulta	159
Utilice la cláusula HAVING para especificar un grupo de condiciones de búsqueda	164
Utilice la cláusula ORDER BY para ordenar los resultados de una consulta	166
8 Modificar datos SQL	175
Insertar datos SQL	176
Insertar valores desde una instrucción SELECT	180
Actualizar datos SQL	182
Actualizar valores desde una instrucción SELECT	185
Eliminar datos SQL	186
9 Utilizar predicados	193
Comparar datos SQL	194
Utilizar el predicado BETWEEN	199
Arrojar valores nulos	200
Arrojar valores similares	203
Hacer referencia a fuentes adicionales de datos	209
Utilizar el predicado IN	209
Utilizar el predicado EXISTS	213
Determinar la cantidad de predicados de comparación	216
Utilizar los predicados SOME y ANY	216
Utilizar el predicado ALL	218
10 Trabajar con funciones y expresiones de valor	225
Utilizar funciones Set	226
Utilizar la función COUNT	227
Utilizar las funciones MAX y MIN	229
Utilizar la función SUM	231
Utilizar la función AVG	232
Utilizar funciones de valor	232
Trabajar con funciones de valor de cadena	233
Trabajar con funciones de valor de fecha y hora	236
Utilizar expresiones de valor	238
Trabajar con expresiones de valor numéricas	238

Utilizar la expresión de valor CASE.....	241
Utilizar la expresión de valor CAST.....	244
Utilizar valores especiales	245
11 Acceder a múltiples tablas.....	253
Realizar operaciones básicas join	254
Utilizar nombres de correlación	257
Crear operaciones join con más de dos tablas	258
Crear la operación cross join	259
Crear la operación self-join.....	260
Unir tablas con nombres de columna compartidos	261
Crear el método join natural	262
Crear el método join de columna nombrada.....	263
Utilizar el método join de condición	263
Crear la inner join	264
Crear la outer join	266
Realizar operaciones de unión	269
12 Utilizar subconsultas para acceder y modificar datos.....	277
Crear subconsultas que arrojen múltiples filas	278
Utilizar el predicado IN	279
Utilizar el predicado EXISTS.....	281
Utilizar predicados de comparación cuantificados.....	282
Crear subconsultas que arrojen un solo valor	283
Trabajar con subconsultas correlacionadas	284
Utilizar subconsultas anidadas.....	286
Utilizar subconsultas para modificar datos	288
Utilizar subconsultas para insertar datos	288
Utilizar subconsultas para actualizar datos.....	290
Utilizar subconsultas para eliminar datos.....	291
 Parte III Acceso avanzado a los datos	
13 Crear rutinas invocadas por SQL.....	299
Entender las rutinas invocadas por SQL	300
Procedimientos y funciones invocadas por SQL	301
Trabajar con la sintaxis básica	302
Crear procedimientos invocados por SQL.....	303
Invocar procedimientos invocados por SQL.....	305
Agregar parámetros de entrada a sus procedimientos	306
Utilizar procedimientos para modificar datos.....	309
Agregar variables locales a sus procedimientos.....	311

Trabajar con instrucciones de control	313
Crear instrucciones compuestas	313
Crear instrucciones condicionales	314
Crear instrucciones de repetición	316
Agregar parámetros de salida a sus procedimientos	320
Crear funciones invocadas por SQL	321
14 Crear activadores SQL.....	329
Entender los activadores SQL	330
Contexto de ejecución del activador	331
Crear activadores SQL	333
Referenciar valores antiguos y nuevos	334
Quitar activadores SQL	335
Crear activadores de inserción	336
Crear activadores de actualización	338
Crear activadores de eliminación	343
15 Utilizar cursores SQL.....	351
Entender los cursores SQL	352
Declarar y abrir cursores SQL	353
Declarar un cursor	355
Trabajar con elementos opcionales de la sintaxis	356
Crear una instrucción de cursor	360
Abrir y cerrar un cursor	363
Recuperar datos desde un cursor	363
Utilizar instrucciones UPDATE y DELETE posicionadas	368
Utilizar la instrucción UPDATE posicionada	369
Utilizar la instrucción DELETE posicionada	370
16 Manejar transacciones SQL	377
Entender las transacciones SQL	378
Configurar las propiedades de la transacción	381
Especificar un nivel de aislamiento	382
Especificar un tamaño de diagnóstico	387
Crear una instrucción SET TRANSACTION	388
Iniciar una transacción	389
Determinar el aplazamiento de una restricción	390
Crear puntos de recuperación en una transacción	392
Liberar un punto de recuperación	394
Finalizar una transacción	395
Completar una transacción	395
Reinvertir una transacción	396

17 Acceder a datos SQL desde un programa host	403
Invocar SQL directamente	404
Incrustar instrucciones SQL en el programa	406
Crear una instrucción SQL incrustada	407
Utilizar variables host en las instrucciones SQL	408
Recuperar datos SQL	411
Manejo de errores	413
Crear módulos cliente de SQL	417
Definir módulos cliente de SQL	418
Utilizar una interfaz de nivel de llamada de SQL	419
Asignar indicadores	421
Ejecutar instrucciones SQL	423
Trabajar con variables host	424
Recuperar datos SQL	426
18 Trabajar con datos XML	433
Aprender los conceptos básicos de XML	434
Aprender acerca de SQL/XML	437
El tipo de datos XML	437
Funciones SQL/XML	439
Reglas de trazado de SQL/XML	441
 Parte IV Apéndices	
A Respuestas a los autoexámenes	449
B Palabras clave de SQL:2006	491
Palabras clave reservadas de SQL	492
Palabras clave no reservadas de SQL	494
C Código SQL utilizado en los ejercicios Pruebe esto.	497
Código SQL por cada ejercicio	498
La base de datos INVENTARIO	514
Índice	519

Agradecimientos

Hubo mucha gente involucrada en el desarrollo de *Fundamentos de SQL, tercera edición*, muchos de los cuales ni siquiera conozco por nombre. Primero que nada, a los editores y el personal en McGraw-Hill Professional que proporcionaron incontables horas de soporte para este proyecto. Me gustaría agradecer especialmente a la editora Jane K. Brownlow, a la coordinadora de compras Jennifer Housh y a Wilson Drozdowski quien cubrió brevemente a Jennifer, y a todas las personas con quienes individualmente tuve contacto directo a través de todo el proceso de escritura y edición. Sus comentarios y sugerencias, así como sus rápidas y exactas respuestas a muchas de mis preguntas, hicieron que las tareas de escritura se realizaran sin ningún problema; su trabajo tras bambalinas mantuvo al proyecto completo avanzando suavemente. También me gustaría agradecer al corrector de estilo y todos los demás editores, correctores de pruebas, indícers, diseñadores, ilustradores y otros participantes cuyos nombres desconozco. Un agradecimiento muy especial va dedicado para mi amigo y ex colega Jim Seymour, revisor técnico, por su atención a los detalles y sus aportaciones de mucha ayuda a través de todo el proceso de edición. También me gustaría agradecer el trabajo de Robert Sheldon, autor de las primeras dos ediciones, cuyo excelente trabajo en escritura hizo que las revisiones necesarias para esta edición fueran mucho más fáciles de lograr. Finalmente, quiero agradecer a mi familia por su apoyo y comprensión para poder organizar un horario para la escritura y creación de este libro en mi ya muy ocupada vida.

Andy Oppel

Introducción

Las bases de datos relacionales se han convertido en el mecanismo de almacenamiento de datos más común para las aplicaciones computacionales modernas. Los lenguajes de programación como Java, C y COBOL, y los lenguajes interpretados de programación como Perl, VBScript y JavaScript muy a menudo acceden a las fuentes de datos para poder recuperar o modificar los datos. Muchas de estas fuentes de datos son administradas a través de un sistema de administración de bases de datos relacionales (RDBMS), como Oracle, Microsoft SQL Server, MySQL y DB2, que tiene como base el Lenguaje de Consulta Estructurado (SQL) para crear y alterar los objetos de la base de datos, agregar datos y eliminarlos de la base de datos, modificar datos que han sido agregados a esa base de datos y, por supuesto, recuperar datos almacenados en la base de datos para su desplegado y procesamiento.

SQL es el lenguaje más ampliamente implementado para las bases de datos relacionales. De la misma manera que las matemáticas son el lenguaje de la ciencia, SQL es el lenguaje de las bases de datos relacionales. SQL no solamente permite administrar los datos dentro de la base de datos, sino también manejar la base de datos en sí.

Utilizando las instrucciones SQL, es posible acceder a una base de datos SQL directamente al utilizar una aplicación cliente interactiva o a través de un lenguaje de programación de aplicación o

lenguaje interpretado. Sin importar cuál sea el método que se utilice para acceder a una fuente de datos, es obligatoria una buena base acerca de cómo escribir instrucciones SQL para poder acceder a los datos relacionales. *Fundamentos de SQL, tercera edición*, le proporciona esa base. Se describen todos los tipos de instrucciones que soporta SQL y se explica cómo son utilizadas para administrar bases de datos y sus datos. A través del trabajo con este libro, usted construirá fuertes cimientos en SQL básico y obtendrá un profundo entendimiento de cómo utilizar SQL para acceder a los datos en su base de datos relacional.

Esta tercera edición ha sido actualizada para incluir las disposiciones del estándar ISO SQL:2006 además de todos sus errores corregidos que se publicaron en 2007. El capítulo 18 ha sido incluido para cubrir SQL/XML, que fue agregado al estándar SQL en 2006. Además, las instrucciones SQL han sido reformateadas y todos los nombres de objeto de la base de datos han sido convertidos a mayúsculas para hacer más fácil su lectura y conversión, a través de la amplia variedad de los productos RDBMS disponibles comercialmente.

Quién deberá leer este libro

Fundamentos de SQL está recomendado para cualquiera que busque construir una base en programación de SQL basado en el estándar ISO SQL:2006. El libro está diseñado específicamente para aquellos que son nuevos, o relativamente nuevos en SQL; sin embargo, aquellos que necesitan refrescar sus conocimientos en SQL también encontrarán beneficios en este libro. Ya sea que usted sea un programador experimentado, tenga cierta experiencia en el desarrollo Web, sea un administrador de bases de datos o sea completamente nuevo en programación y bases de datos *Fundamentos de SQL* le proporcionará fuertes bases que serán útiles para cualquiera que desee aprender más acerca de SQL. De hecho, cualquiera de los siguientes ejemplos de personas encontrará útil este libro cuando intente comprender y utilizar SQL:

- El novato que no tenga conocimiento acerca del diseño de bases de datos y programación SQL
- El analista o administrador que quiera una mejor comprensión de cómo implementar y acceder a las bases de datos SQL
- El administrador de bases de datos que quiera aprender más acerca de programación
- El profesional de soporte técnico o ingeniero de pruebas/control de calidad quien realiza consultas *ad hoc* en alguna fuente de datos SQL
- El desarrollador Web que escribe aplicaciones que deberán acceder a bases de datos SQL
- El programador del lenguaje de tercera generación (3GL) que incrusta SQL dentro del código fuente de una aplicación
- Cualquier otro individuo que desee aprender acerca de cómo escribir código SQL que pueda ser utilizado para crear y acceder a bases de datos dentro de algún RDBMS

Cualquiera que sea la categoría en la que se encuentre, un punto importante a recordar es que el libro está enfocado a cualquiera que quiera aprender SQL estándar, y no una versión específica de un producto o de este lenguaje. La ventaja de esto es que es posible tomar las habilidades aprendidas en este libro y aplicarlas a situaciones del mundo real, sin estar limitado a los estándares de un solo producto.

Aún necesitará, por supuesto, conocer cómo implementa SQL el producto en el que está trabajando; pero con la base proporcionada por este libro, será capaz de moverse entre un RDBMS y otro

y aún tener una comprensión básica de cómo se utiliza SQL. Como resultado, este libro es una herramienta útil para cualquier persona que no sepa utilizar las bases de datos basadas en SQL, sin importar el producto que utilice. Los programadores de SQL necesitarán solamente adaptar su conocimiento al RDBMS específico.

Qué contenido cubre el libro

Fundamentos de SQL está dividido en tres partes. La parte I introduce al lector a los conceptos básicos de SQL y explica cómo crear objetos dentro de la base de datos. La parte II proporciona una base acerca de cómo recuperar datos desde una base de datos y modificar (agregar, cambiar y eliminar) esos datos que se encuentran almacenados en la base de datos. La parte III proporciona información acerca de técnicas avanzadas de acceso de datos que permiten expandir lo que se aprendió en las partes I y II. Adicional a estas tres partes, *Fundamentos de SQL* contiene apéndices que incluyen material de referencia para la información presentada en las tres partes.

Descripción de los contenidos del libro

El siguiente resumen describe los contenidos del libro y muestra cómo el libro se encuentra separado en capítulos que se enfocan en determinadas tareas.

PARTE I: Bases de datos relacionales y SQL

Capítulo 1: Introducción a las bases de datos relacionales y a SQL

En este capítulo se encuentra la introducción a las bases de datos relacionales y al modelo relacional, que forma la base para SQL. También se proporcionará un vistazo general a SQL y cómo se relaciona con los RDBMS.

Capítulo 2: Trabajo con el entorno SQL

Este capítulo describe los componentes que conforman el entorno SQL. También se encontrará una introducción a los objetos que conforman un esquema, y se aprenderá cómo crear un esquema dentro del entorno SQL. Se presentará el concepto de crear un objeto de la base de datos en una implementación SQL que soporte la creación de objetos de base de datos.

Capítulo 3: Creación y modificación de tablas

En este capítulo se aprenderá cómo crear tablas SQL, especificar tipos de datos de columna, crear tipos definidos por el usuario y especificar valores de columna por defecto. Se aprenderá cómo alterar una definición de tabla y eliminar esa definición de la base de datos.

Capítulo 4: Implementación de la integridad de datos

Este capítulo explica cómo se utilizan las restricciones de integridad para reforzar la integridad de los datos en las tablas SQL. El capítulo incluye información sobre restricciones relacionadas con las tablas, afirmaciones y restricciones de dominio. Se aprenderá cómo crear restricciones NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY y CHECK.

Capítulo 5: Creación de vistas en SQL

En este capítulo se aprenderá cómo agregar vistas a las bases de datos SQL. También se aprenderá cómo crear vistas actualizables y cómo abandonar vistas de la base de datos.

Capítulo 6: Gestión de seguridad en la base de datos

En este capítulo se introducirá el modelo de seguridad de SQL y se aprenderá cómo se definen los identificadores de autorización dentro del contexto de una sesión. Luego, se aprenderá cómo crear y eliminar roles, otorgar y revocar privilegios, y otorgar y revocar roles.

PARTE II Acceso y modificación de datos

La parte II explica cómo acceder y modificar los datos en una base de datos SQL. También se aprenderá cómo utilizar predicados, funciones y valores de expresión para manipular esos datos. Adicionalmente, la parte II describe cómo unir tablas y utilizar subconsultas para acceder a los datos en múltiples tablas.

Capítulo 7: Consulta de datos de SQL

Este capítulo describe los componentes básicos de la instrucción `SELECT` y cómo la instrucción es utilizada para recuperar datos desde una base de datos SQL. Se aprenderá cómo definir cada cláusula que pueda ser incluida en la instrucción `SELECT` y cómo esas cláusulas son procesadas cuando se consulta una base de datos.

Capítulo 8: Modificar datos SQL

En este capítulo se aprenderá cómo modificar los datos en una base de datos SQL. Específicamente, se aprenderá cómo insertar datos, actualizar datos y eliminar datos. El capítulo repasa cada componente de las instrucciones SQL que permiten realizar esas modificaciones a los datos.

Capítulo 9: Utilizar predicados

En este capítulo se aprenderá cómo utilizar predicados para comparar datos SQL, arrojar valores nulos, arrojar valores similares, hacer referencia a fuentes adicionales de datos y cuantificar predicados de comparación. El capítulo describe los diferentes tipos de predicados y muestra cómo son utilizados para recuperar datos específicos de una base de datos SQL.

Capítulo 10: Trabajar con funciones y expresiones de valor

Este capítulo explica cómo utilizar los diferentes tipos de funciones y expresiones de valor en las instrucciones SQL. Se aprenderá cómo utilizar funciones set, funciones de valor, expresiones de valor y valores especiales en diferentes cláusulas dentro de la instrucción SQL.

Capítulo 11: Acceder a múltiples tablas

Este capítulo describe cómo unir tablas para poder recuperar datos desde esas tablas. Se aprenderá cómo realizar operaciones join básicas, unir tablas con nombres de columna compartidos, utilizar la función join de condición y realizar operaciones de unión.

Capítulo 12: Utilizar subconsultas para acceder y modificar datos

En este capítulo se aprenderá cómo crear subconsultas que arrojen múltiples filas y otras que arrojen solamente un valor. También se aprenderá cómo utilizar subconsultas correlacionadas y subconsultas anidadas. Adicionalmente, se aprenderá cómo utilizar subconsultas para modificar datos.

PARTE III Acceso avanzado a los datos

La parte III presenta una introducción a las técnicas avanzadas de acceso de datos como las rutinas invocadas por SQL, activadores y cursores. También se aprenderá cómo administrar transacciones, cómo acceder a los datos SQL desde el programa host y cómo incorporar datos XML a la base de datos.

Capítulo 13: Crear rutinas invocadas por SQL

Este capítulo describe los procedimientos y las funciones invocadas por SQL y cómo pueden crearse en la base de datos SQL. Se aprenderá cómo definir parámetros de entrada, agregar variables locales a la rutina, trabajar con instrucciones de control y utilizar parámetros de salida.

Capítulo 14: Crear activadores SQL

Este capítulo presenta los activadores SQL y explica cómo crear activadores de inserción, de actualización y de eliminación en la base de datos SQL. Se aprenderá cómo los activadores son invocados automáticamente y qué tipos de acciones se pueden tomar.

Capítulo 15: Utilizar cursores SQL

En este capítulo se aprenderá cómo los cursores SQL se utilizan para recuperar una fila de datos a la vez desde un conjunto de resultados. El capítulo explica cómo declarar un cursor, abrir y cerrar un cursor, y cómo recuperar datos desde un cursor. También se aprenderá cómo utilizar instrucciones UPDATE y DELETE posicionadas después de buscar una fila utilizando un cursor.

Capítulo 16: Manejar transacciones SQL

En este capítulo se aprenderá cómo se utilizan las transacciones para asegurar la integridad de los datos SQL. El capítulo describe cómo establecer las propiedades de la transacción, iniciar una transacción, establecer el desplazamiento de las restricciones, crear puntos de recuperación en una transacción y finalizar una transacción.

Capítulo 17: Acceder a datos SQL desde un programa host

Este capítulo describe los cuatro métodos soportados por el estándar SQL para acceder a una base de datos SQL. Se aprenderá cómo invocar SQL directamente desde una aplicación cliente, incrustar instrucciones SQL en un programa, crear módulos cliente de SQL y utilizar una interfaz de nivel de llamada de SQL para acceder a los datos.

Capítulo 18: Trabajar con datos XML

Este capítulo describe cómo los datos XML pueden incorporarse a una base SQL. Aprenderá los aspectos básicos de XML, cómo usar los tipos de datos de XML para almacenar XML en los valores de columna de la tabla, escribir funciones SQL/XML para arrojar datos desde las bases formateadas como XML, y las reglas de trazado de SQL/XML que describen cómo los valores SQL son transformados a XML y viceversa.

PARTE IV Apéndices

Apéndice A: Respuestas a los autoexámenes

Este apéndice proporciona las respuestas para las preguntas de las autoevaluaciones enlistadas al final de cada capítulo.

Apéndice B: Palabras clave de SQL:2006

Este apéndice enumera las palabras clave reservadas y no reservadas de la forma en que se utilizan en las instrucciones SQL, como se define en el estándar SQL:2006.

Apéndice C: Código SQL utilizado en los ejercicios Pruebe esto

Este apéndice enumera todo el código SQL utilizado en los ejercicios de este libro, consolidados en un solo lugar para una referencia más fácil. Este código también puede ser descargado desde www.mcgraw-hill-educacion.com. Haga una búsqueda por autor, ISBN o título.

Contenido de cada capítulo

Como se puede observar en el resumen, *Fundamentos de SQL* está organizado en capítulos. Cada capítulo se enfoca en un conjunto de tareas relacionadas. El capítulo contiene la información de importancia que se necesita para comprender los diferentes conceptos relacionados con esas tareas, explica cómo crear las instrucciones SQL necesarias para realizar las tareas y proporciona ejemplos de cómo se crean esas instrucciones. Suplementariamente, cada capítulo contiene elementos adicionales para ayudar a comprender mejor la información cubierta en ese capítulo:

- **Pregunte al experto** Cada capítulo contiene una o dos de estas secciones; proporcionan información acerca de preguntas que pudieran surgir referentes a la información presentada en el capítulo.
- **Autoexamen** Cada capítulo termina con un Autoexamen, que es un conjunto de preguntas que examinan al usuario acerca de la información y habilidades que se aprendieron en este capítulo. Las respuestas a los autoexámenes se encuentran en el apéndice A.

Sintaxis SQL

La sintaxis de una instrucción SQL se refiere a la estructura y a las reglas utilizadas para esa instrucción, como se especifica en SQL:2006. La mayoría de los capítulos incluirán la sintaxis para una o más instrucciones para que se pueda tener un entendimiento de los elementos básicos contenidos en esas instrucciones. Por ejemplo, la siguiente sintaxis representa la información necesaria cuando se define una instrucción CREATE TABLE:

```
<definición de la tabla> ::=  
CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <nombre de la tabla>  
(<elemento de la tabla> [ { , <elemento de la tabla> } ... ] )  
[ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

NOTA

Por el momento no tiene que preocuparse acerca del significado de este código SQL. Se presenta este ejemplo solamente para mostrar cómo se representan las instrucciones SQL en este libro.

Como se puede ver, la sintaxis de una instrucción puede contener muchos elementos. Nótese que la mayoría de las palabras utilizadas dentro de la instrucción se muestran en mayúsculas. Las palabras en mayúsculas son palabras clave SQL que se utilizan para formular la instrucción SQL. (Para una lista completa de las palabras clave SQL:2006, vea el apéndice B.) A pesar de que SQL no requiere que las palabras clave sean escritas en mayúsculas, se utiliza esa convención en este libro para que pueda identificar fácilmente las palabras clave dentro de una instrucción. Adicionalmente a las palabras clave, la sintaxis para una instrucción SQL incluye otros muchos elementos que ayudan a definir cómo deberá ser creada una instrucción en particular:

- **Corchetes** Los corchetes indican que la sintaxis encerrada en esos corchetes es opcional. Por ejemplo, la cláusula ON COMMIT en la instrucción CREATE TABLE es opcional.
- **Corchetes angulares** Los corchetes angulares encierran información que representa un marcador de posición. Cuando la instrucción ha sido creada, el marcador de posición es reemplazado por los

elementos o identificadores SQL apropiados. Por ejemplo, deberá reemplazarse el marcador de posición <nombre de la tabla> con un nombre para la tabla cuando se defina la instrucción CREATE TABLE.

- **Llaves** Las llaves se utilizan para agrupar elementos. Las llaves indican que primero se deberá decidir cómo manejar los contenidos dentro de las llaves y luego determinar cómo encajan dentro de la instrucción. Por ejemplo, el conjunto de palabras clave PRESERVE | DELETE está encerrado entre llaves. Primero se deberá elegir entre PRESERVE o DELETE y luego tratar con la línea completa del código. Como resultado, la cláusula puede leerse como ON COMMIT PRESERVE ROWS, o como ON COMMIT DELETE ROWS.
- **Barras verticales** La | Puede leerse como “o”, lo que significa que deberá utilizar ya sea la opción PRESERVE o la opción DELETE.
- **Tres puntos** Los tres puntos indican que se puede repetir la cláusula tantas veces como sea necesario. Por ejemplo, se pueden incluir tantos elementos de la tabla como sean necesarios (representados por <elemento de la tabla>).
- **Dos puntos y signo de igual** El símbolo ::= (dos veces consecutivas los dos puntos y el signo de igual) indica que el marcador de posición a la izquierda del símbolo está definido por la sintaxis que sigue al símbolo. En el ejemplo de la sintaxis, el marcador de posición <definición de la tabla> iguala a la sintaxis que conforma una instrucción CREATE TABLE.

Al referirse a la sintaxis deberá ser capaz de construir una instrucción SQL que cree objetos de la base de datos o modifique datos SQL según sea necesario. Sin embargo, para poder demostrar mejor cómo se aplica la sintaxis, cada capítulo contiene también ejemplos de instrucciones SQL reales.

Ejemplos de instrucciones SQL

Cada capítulo proporciona ejemplos de cómo se implementan las instrucciones SQL cuando se accede a una base de datos SQL. Por ejemplo, podría observarse una instrucción SQL similar a la siguiente:

```
CREATE TABLE ARTISTAS
( ID_ARTISTA          INT,
  NOMBRE_ARTISTA      VARCHAR(60) ,
  FDN_ARTISTA         DATE,
  POSTER_EN_EXISTENCIA  BOOLEAN );
```

Observe que la instrucción está escrita en un tipo especial para mostrar que se trata de código SQL. Observe también que las palabras clave y los nombres de objeto están todos en mayúsculas. (Hasta este punto, no es necesario preocuparse por ninguno de los demás detalles.)

Los ejemplos utilizados en este libro son de SQL puro, lo que significa que están basados en el estándar SQL:2006. Se encontrará, sin embargo, que en algunos casos las implementaciones SQL no soportan una instrucción SQL exactamente de la misma forma en la que está definida en el estándar. Por esta razón, también pudiera ser necesario referirse a la documentación para un producto en particular con tal de asegurarse de que sus instrucciones SQL acaten la implementación de ese producto de SQL.

Pudiera ser a veces solamente una variación ligera, pero habrá ocasiones en las que la instrucción del producto será sustancialmente diferente de la instrucción SQL estándar.

Los ejemplos en cada capítulo están basados en una base de datos relacionada a un inventario de discos compactos. Sin embargo, los ejemplos no son necesariamente consistentes en términos de los nombres utilizados para los objetos de la base de datos y cómo esos objetos están definidos. Por ejemplo, dos capítulos diferentes pudieran contener ejemplos que hacen referencia a una tabla llamada INVENTARIO_CD. Sin embargo, no se puede asumir que las tablas utilizadas en diferentes ejemplos estén conformadas por las mismas columnas o contengan los mismos datos. Debido a que cada ejemplo se enfoca en un aspecto único de SQL, las tablas utilizadas en los ejemplos están definidas en una forma específica para las necesidades de ese ejemplo, como se verá mientras se avanza en los capítulos. Éste no es el caso para los ejercicios, que utilizan una estructura de base de datos consistente a través de todo el libro.

Ejercicios Pruebe esto

Cada capítulo contiene uno o dos ejercicios que permiten aplicar la información que se aprendió durante el capítulo. Cada ejercicio está dividido en pasos que lo llevarán a través del proceso de completar una tarea en particular. Muchos de los proyectos incluyen archivos relacionados que pueden ser descargados desde nuestro sitio Web en www.mcgraw-hill-educacion.com. Los archivos a menudo incluyen las instrucciones SQL utilizadas dentro de los ejercicios Pruebe esto (**en inglés**). Adicionalmente, se incluye una consolidación (**en español**) de estas instrucciones SQL en el apéndice C.

Los ejercicios están basados en la base de datos INVENTARIO. Usted creará la base de datos y las tablas en otros objetos en la base de datos, agregará datos a esas tablas, y luego manipulará esos datos. Debido a que los proyectos se construyen uno sobre el otro, es mejor que los complete en el orden en que están presentados en el libro. Esto resulta de especial importancia para los capítulos en la parte I, en la cual se crean los objetos de la base de datos, y en el capítulo 7, en el cual se insertan los datos en las tablas. Sin embargo, si de todas maneras usted prefiere hacerlos de forma aleatoria, puede referirse al apéndice C, que proporciona todo el código necesario para crear los objetos de la base de datos y poblar esas tablas con datos.

Para completar la mayoría de los ejercicios de este libro, usted necesitará tener acceso a un RDBMS que le permita ingresar y ejecutar instrucciones SQL interactivamente. Si está accediendo a un RDBMS a través de una red, verifique que el administrador de la base de datos se asegure de que usted está ingresando con las credenciales necesarias para crear una base de datos y un esquema. Podría necesitar permisos especiales para crear esos objetos. También verifique si es que existen ciertos parámetros que deberá incluir cuando cree la base de datos (por ejemplo, el tamaño de archivo de ingreso), restricciones acerca de los nombres que puedan utilizarse o restricciones de cualquier otro tipo. Asegúrese de revisar la documentación del producto antes de trabajar con cualquier producto de base de datos.

Parte I

Bases de datos relacionales y SQL

Capítulo 1

Introducción a las
bases de datos
relacionales y a SQL

Habilidades y conceptos clave

- Entienda las bases de datos relacionales
 - Aprenda acerca de SQL
 - Use un sistema de gestión de base de datos relacional
-

En el 2006, la International Organization for Standardization (ISO) y el American National Standards Institute (ANSI) publicaron revisiones a su estándar SQL, al cual llamaré SQL:2006. Como verá después, el estándar está dividido en partes, y cada parte es aprobada y publicada en su propia línea de tiempo; por lo tanto, diferentes partes tienen distintos años de publicación. Es común usar el último año como nombre colectivo para el conjunto de todas las partes publicadas durante ese año. El estándar SQL 2006, como sus predecesores SQL:2003, SQL:1999 (también conocido como SQL3) y SQL-92, se basa en el modelo de datos relacional, el cual define cómo los datos pueden ser almacenados y manipulados dentro de una base de datos relacional. Los sistemas de gestión de base de datos relacional (RDBMS) como Oracle, Sybase, DB2, MySQL y Microsoft SQL Server (o sólo SQL Server) usan el estándar SQL como base de su tecnología, proporcionando entornos de base de datos que apoyan tanto a SQL como al modelo de datos relacional. Hay más información sobre el estándar SQL más adelante en este capítulo.

Entienda las bases de datos relacionales

El lenguaje estructurado de consultas (SQL, Structured Query Language) apoya la creación y mantenimiento de la base de datos relacional y la gestión de los datos dentro de la base de datos. Sin embargo, antes de entrar en discusión acerca de las bases de datos relacionales, quiero explicar el significado del término *base de datos*. El término ha sido utilizado para referirse a cualquier cosa, desde una colección de nombres y direcciones hasta un complejo sistema de recuperación y almacenamiento de datos que se basa en interfaces de usuarios y una red de computadoras y servidores. Hay tantas definiciones para la palabra *base de datos* como libros sobre éstas. Por otra parte, los distintos proveedores de DBMS han desarrollado diferentes arquitecturas, por lo que no todas las bases de datos están diseñadas de la misma manera. A pesar de la falta de una definición absoluta, la mayoría de las fuentes coinciden en que una base de datos, por lo menos, es una colección de datos organizada en un formato estructurado que es definido como *metadatos* que describe esa estructura. Puede pensar en los metadatos como información sobre los datos almacenados, que define cómo se almacenan éstos en una base de datos.

A lo largo de los años se ha implementado una serie de modelos de base de datos para almacenar y administrar la información. Varios de los modelos más comunes incluyen los siguientes:

- **Jerárquico** Este modelo tiene una estructura primario y secundario que es similar a un árbol invertido, que es lo que constituye la jerarquía. Los datos están organizados en *nodos*, el equivalente lógico de tablas en una base de datos relacional. Un nodo principal puede tener muchos nodos secundarios, pero un nodo secundario sólo puede tener un nodo principal. Aunque

el modelo ha sido muy utilizado, a menudo se considera inadecuado para muchas aplicaciones debido a su estructura inflexible y la falta de apoyo de relaciones complejas. Aun algunas implementaciones como IMS de IBM han introducido características que trabajan en torno a estas limitaciones.

- **Red** Este modelo aborda algunas de las limitaciones del modelo jerárquico. Los datos están organizados en *tipos de registro*, el equivalente lógico de tablas en una base de datos relacional. Al igual que el modelo jerárquico, el modelo de red usa la estructura de un árbol invertido, pero los tipos de registro se organizan en una estructura que relaciona pares de tipos de registro en propietarios y miembros. Cualquier tipo de registro puede participar en cualquier conjunto con otros tipos de registro en la base de datos, que apoya a las consultas y relaciones más complejas de lo que es posible en el modelo jerárquico. Hasta el modelo de red tiene sus limitaciones, y la más seria es la complejidad. Al acceder a la base de datos, el usuario debe estar familiarizado con la estructura y mantener un seguimiento cuidadoso de dónde está y cómo llegó ahí. También es difícil cambiar la estructura sin afectar las aplicaciones que interactúan con la base de datos.
- **Relacional** Este modelo aborda algunas de las limitaciones de los modelos jerárquicos y de red. En una base de datos de modelo jerárquico o de red, la aplicación se basa en una implementación determinada de esa base de datos, que luego es codificada en la aplicación. Si agrega un nuevo atributo (elemento de datos) a la base de datos, debe modificar la aplicación, aun cuando no se use ese atributo. Sin embargo, una base de datos relacional es independiente de la aplicación; puede hacer modificaciones no destructivas a la estructura sin afectar la aplicación. Además, la estructura de la base de datos relacional se basa en la relación, o tabla, junto con la habilidad de definir relaciones complejas entre ellas. Se puede acceder directamente a cada relación sin la lentitud de las limitaciones de los modelos jerárquicos o propietario/miembro que requiere de una navegación a través de una estructura compleja de datos. En la siguiente sección, “El modelo relacional”, se verá con mayor detalle este modelo.

Aunque aún se usan en muchas organizaciones, las bases de datos de modelo jerárquico y de red ahora se consideran como soluciones heredadas. El modelo relacional es el más ampliamente aplicado en los sistemas de negocios modernos, y es el modelo relacional el que proporciona la base para SQL.

El modelo relacional

Si alguna vez tiene la oportunidad de ver un libro acerca de base de datos relacionales, es muy posible que vea el nombre de E. F. (Ted) Codd, a quien se hace referencia en el contexto del modelo relacional. En 1970, Codd publicó su trabajo más importante “A Relational Model Of Data For Large Shared Data Banks” (Un modelo relacional de datos para grandes bancos de datos compartidos), en el diario *Communications of the ACM*, volumen 13, número 6 (junio de 1970). Codd define una estructura de datos relacional que protege los datos y permite que sean manipulados de manera que es previsible y resistente al error. El modelo relacional, el cual se basa principalmente en los principios matemáticos de la teoría de conjuntos y lógica de predicados, apoya la recuperación de datos sencilla, aplica la *integración de datos* (la precisión y coherencia de los datos), y proporciona una estructura de base de datos independiente de las aplicaciones al acceder a los datos almacenados.

El núcleo del modelo relacional es la relación. Una *relación* es un conjunto de columnas y filas reunidas en una estructura en forma de tabla que representa una entidad única formada por los datos relacionados. Una *entidad* es una persona, lugar, cosa, evento o concepto sobre el cual los datos son recolectados, como un artista, un libro o una transacción de ventas. Cada relación comprende uno o más atributos (columnas). Un *atributo* es un hecho simple que describe o caracteriza una entidad de alguna manera. Por ejemplo, en la figura 1-1, la entidad es un disco compacto (CD) con atributos de NOMBRE_CD (el título del CD), NOMBRE_ARTISTA (el nombre del artista) y AÑO_DERECHOSDEAUTOR (el año de los derechos de autor del disco).

Como se ve en la figura 1-1, cada atributo tiene un dominio asociado. Un *dominio* define el tipo de datos que son almacenados en un atributo particular; sin embargo, un dominio no es lo mismo que un tipo de datos. Un *tipo de datos*, el cual se tratará con mayor detalle en el capítulo 3, es un tipo específico de *restricción* (un control usado para hacer cumplir la integridad de los datos) asociados con una columna, mientras que un dominio, tal como se utiliza en el modelo relacional, tiene un significado más amplio y describe exactamente qué datos pueden ser incluidos en un atributo asociado con ese dominio. Por ejemplo, el atributo AÑO_DERECHOSDEAUTOR es asociado con el dominio de Año. Como se ve en este ejemplo, es común la práctica de incluir una clase de palabra que describe los nombres de dominio en el atributo, pero esto no es obligatorio. El dominio puede ser definido de manera que el atributo sólo incluya datos cuyos valores y formato sean limitados a años, a diferencia de días o meses. El dominio también puede limitar los datos a un rango específico de años. Un tipo de datos, por otro lado, restringe el formato de los datos, como el permitir únicamente dígitos numéricos, pero no los valores, a menos que esos valores violen de alguna manera el formato.

Los datos se almacenan en una relación en tuplas (filas). Una *tupla* es un conjunto de datos cuyos valores hacen una instancia de cada atributo definido por esa relación. Cada tupla representa un registro de datos relacionados. (De hecho, el conjunto de datos se conoce en ocasiones como *registro*.) Por ejemplo, en la figura 1-1 la segunda tupla de arriba hacia abajo contiene el valor “Joni Mitchell” para el atributo NOMBRE_ARTISTA, el valor “Blue” para el atributo NOMBRE_CD y el valor “1971” para el atributo AÑO_DERECHOSDEAUTOR. Estos tres valores juntos forman una tupla.

Atributo	Nombre del atributo (NOMBRE_CD)	Nombre del dominio (Año)
NOMBRE_ARTISTA:NombreCompleto	NOMBRE_CD:Título	AÑO_DERECHOSDEAUTOR:Año
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue	1971
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988

Figura 1-1 Relación con los atributos NOMBRE_CD, NOMBRE_ARTISTA y AÑO_DERECHOSDEAUTOR.

NOTA

Los términos lógicos relación, atributo y tupla se usan principalmente para referirse al modelo relacional. SQL usa los términos físicos tabla, columna y fila para describir esos elementos. Debido a que el modelo relacional se basa en modelos matemáticos (un modelo lógico) y SQL se enfoca más a la implementación física del modelo, los significados para los términos del modelo y los términos de lenguaje de SQL son ligeramente distintos, pero los principios básicos son los mismos. Los términos de SQL se analizan con más detalle en el capítulo 2.

El modelo relacional es, por supuesto, más complejo que los meros atributos y tuplas que hacen la relación. Dos consideraciones importantes en el diseño e implementación de cualquier base de datos relacional son la normalización de los datos y las asociaciones de relaciones entre los distintos tipos de datos.

Normalización de datos

La parte central de los principios del modelo relacional es el concepto de *normalización*, una técnica para producir un conjunto de relaciones que poseen un conjunto de ciertas propiedades que minimizan los datos redundantes y preservan la integridad de los datos almacenados tal como se mantienen (añadidos, actualizados y eliminados). El proceso fue desarrollado por E. F. Codd en 1972, y el nombre es un chiste político debido a que el presidente Nixon estaba “normalizando” relaciones con China en ese momento. Codd imaginó que si las relaciones con un país pueden normalizarse, entonces seguramente podría normalizar las relaciones de la base de datos. La normalización se define por un conjunto de normas, que se conocen como *formas normales*, que proporcionan una directriz específica de cómo los datos son organizados para evitar anomalías que den lugar a inconsistencias y pérdida de los datos tal como se mantienen almacenados en la base de datos.

Cuando Codd presentó por primera vez la normalización, incluía tres formas normales. A pesar de que formas normales adicionales se han agregado desde entonces, las tres primeras cubren la mayoría de las situaciones que se encontrarán en las dos bases de datos personales y empresariales, y ya que la intención principal es presentar el proceso de normalización, sólo se analizarán esas tres formas.

Elección de un identificador único Un identificador único es un atributo o conjunto de atributos que únicamente identifican cada fila de datos en una relación. El identificador único eventualmente se convertirá en la clave principal de la tabla creada en la base de datos física desde la relación de normalización, pero muchos usan los términos *identificador único* y *clave principal* de manera intercambiable. A cada identificador potencial único se le denomina *candidato clave*, y cuando hay varios candidatos, el diseñador elegirá el mejor, el cual es el menos probable de cambiar valores o el más simple y/o el más corto. En muchos casos, un solo atributo se identifica únicamente en los datos en cada tupla de la relación. Sin embargo, cuando ningún atributo que es único se encuentra, el diseñador busca varios atributos que puedan ser unidos (puestos juntos) para formar un identificador único. En los pocos casos donde ningún candidato clave razonable es encontrado, el diseñador debe inventar un identificador único denominado *sustituto clave*, frecuentemente con valores asignados al azar o secuencialmente cuando las tuplas sean agregadas a la relación.

Mientras no sea absolutamente necesario hasta la segunda forma normal, es habitual seleccionar un identificador único como primer paso en la normalización. Es más fácil de esa manera.

Primera forma normal La primera forma normal, que proporciona la fundación para la segunda y tercera forma normal, incluye las siguientes directrices:

- Cada atributo de una tupla contiene sólo un valor.
- Cada tupla en una relación contiene el mismo número de atributos.
- Cada tupla es diferente, lo que significa que la combinación de los valores de todos los atributos de una tupla dada no puede ser como ninguna otra tupla en la misma relación.

Como se ve en la figura 1-2, la segunda tupla y la última tupla violan la primera forma normal. En la segunda tupla, el atributo NOMBRE_CD y el atributo AÑO_DERECHOSDEAUTOR contienen cada una dos valores. En la última tupla, el atributo NOMBRE_ARTISTA contiene tres valores. También debe vigilar la repetición de valores en forma de repetición de columnas. Por ejemplo, dividir el atributo NOMBRE_ARTISTA en tres atributos llamados NOMBRE_ARTISTA_1, NOMBRE_ARTISTA_2 y NOMBRE_ARTISTA_3 no es una solución adecuada porque es muy probable que necesite un cuarto nombre, luego un quinto, y así sucesivamente. Además, repetir columnas hace que las consultas sean más difíciles, ya que debe recordar buscar en todas las columnas cuando busque un valor específico.

Para normalizar la relación mostrada en la figura 1-2, debe crear relaciones adicionales que separen los datos de modo que cada atributo contenga un solo valor, cada tupla contenga el mismo número de atributos y cada tupla sea diferente, como se muestra en la figura 1-3. Ahora los datos se ajustan a la primera forma normal.

Observe que hay valores duplicados en la segunda relación; el valor de 10002 en el ID_ARTISTA se repite y el valor de 99308 en el ID_CD también se repite. Sin embargo, cuando el valor de los dos atributos en cada tupla son puestos juntos, la tupla como un todo forma una combinación única, lo cual significa que, a pesar de la aparente duplicidad, cada tupla en la relación es diferente.

NOMBRE_ARTISTA	NOMBRE_CD	AÑO_DERECHOSDEAUTOR
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue; Court and Spark	1971; 1974
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988
Jose Carreras; Plácido Domingo; Luciano Pavarotti	Carreras Domingo Pavarotti in Concert	1990

Figura 1-2 Relación de la violación de la primera forma normal.

ID_ARTISTA	NOMBRE_ARTISTA	ID_ARTISTA	ID_CD	ID_CD	NOMBRE_CD	AÑO_DERECHOSDEAUTOR
10001	Jennifer Warnes	10001	99301	99301	Famous Blue Raincoat	1991
10002	Joni Mitchell	10002	99302	99302	Blue	1971
10003	William Ackerman	10002	99303	99303	Court and Spark	1974
10004	Kitaro	10003	99304	99304	Past Light	1983
10005	Bing Crosby	10004	99305	99305	Kojiki	1990
10006	Patsy Cline	10005	99306	99306	That Christmas Feeling	1993
10007	Jose Carreras	10006	99307	99307	Patsy Cline: 12 Greatest Hits	1988
10008	Plácido Domingo	10007	99308	99308	Carreras Domingo Pavarotti in Concert	1990
10009	Luciano Pavarotti	10008	99308			
		10009	99308			

Figura 1-3 Relación que se ajusta a la primera forma normal.

¿Observó que los atributos ID_ARTISTA y ID_CD fueron agregados? Se hizo esto porque no había otros candidatos clave. NOMBRE_ARTISTA no es único (dos personas con el mismo nombre pueden ser artistas), y tampoco es NOMBRE_CD (dos CD pueden terminar con el mismo nombre, aunque es probable que sean de diferentes sellos discográficos). ID_ARTISTA es la clave principal de la primera relación, e ID_CD es la clave principal en la tercera. La clave principal en la segunda relación es la combinación de ID_ARTISTA y ID_CD.

Segunda forma normal Para comprender la segunda forma normal, primero debe entender el concepto de *dependencia funcional*. Para esta definición se usarán dos atributos arbitrarios, hábilmente llamados A y B. El atributo B es *funcionalmente dependiente* (*dependiente* para abreviar) del atributo A si en cualquier momento no hay más que un valor del atributo B asociado con el valor dado al atributo A. Para que no te preguntes de qué planeta vengo, tratemos de hacer la definición más entendible. Si se dice que el atributo B es funcionalmente dependiente del atributo A, también estaremos diciendo que el atributo A *determina* al atributo B, o que A es un factor *determinante* (identificador único) del atributo B. En la figura 1-4, AÑO_DERECHOSDEAUTOR depende de ID_CD, ya que sólo puede haber un valor de AÑO_DERECHOSDEAUTOR para cualquier CD. Dicho de otra manera, ID_CD es un factor determinante de AÑO_DERECHOSDEAUTOR.

La segunda forma normal expone que una relación debe estar en la primera forma normal y que todos los atributos en la relación dependen del identificador único *completo*. En la figura 1-4, si la combinación de ID_ARTISTA y ID_CD es seleccionada como identificador único, entonces AÑO_DERECHOSDEAUTOR violaría la segunda forma normal porque sólo dependería de ID_CD en lugar de la combinación ID_CD y ID_ARTISTA. A pesar de que la relación se ajusta a la primera forma normal, se violaría la segunda forma normal. De nuevo, la solución sería separar los datos en relaciones diferentes, como se vio en la figura 1-3.

Tercera forma normal La tercera forma normal, como la segunda forma normal, depende de la relación del identificador único. Para adherir a las directrices de la tercera forma normal, una

← Identificador único →		
ID_ARTISTA	ID_CD	AÑO_DERECHOSDEAUTOR
10001	99301	1991
10002	99302	1971
10002	99303	1974
10003	99304	1983
10004	99305	1990
10005	99306	1993
10006	99307	1988

Figura 1-4 Relación con un identificador único concatenado.

relación debe estar en la segunda forma normal y sin un atributo clave (atributos que no sean parte de algún candidato clave) deben ser independiente el uno del otro y depender del identificador único. Por ejemplo, el identificador único en la relación mostrada en la figura 1-5 es el atributo ID_ARTISTA.

Identificador único			
ID_ARTISTA	NOMBRE_ARTISTA	ID_AGENCIA	ESTADO_AGENCIA
10001	Jennifer Warnes	2305	NY
10002	Joni Mitchell	2306	CA
10003	William Ackerman	2306	CA
10004	Kitaro	2345	NY
10005	Bing Crosby	2367	VT
10006	Patsy Cline	2049	TN
10007	Jose Carreras	2876	CA
10008	Placido Domingo	2305	NY
10009	Luciano Pavarotti	2345	NY

Figura 1-5 Relación con un atributo que viola la tercera forma normal.

Los atributos NOMBRE_ARTISTA e ID_AGENCIA dependen del identificador único y son independientes uno del otro. Sin embargo, el atributo ESTADO_AGENCIA depende del atributo ID_AGENCIA, y por lo tanto viola las condiciones de la tercera forma normal. Este atributo se adapta mejor en una relación que incluye datos sobre las agencias.

NOTA

En el mundo teórico del diseño relacional, el objetivo es almacenar los datos de acuerdo con las reglas de normalización. Sin embargo, en el mundo real de aplicación de bases de datos, ocasionalmente se desnormalizan los datos, lo que significa que se violen deliberadamente las reglas de normalización, particularmente la segunda y la tercera forma normal. La desnormalización se usa principalmente para mejorar el rendimiento o reducir la complejidad en los casos en donde una estructura demasiado normalizada complica la implementación. Con todo, el objetivo de la normalización es asegurar la integridad de los datos; por lo tanto, la desnormalización se realizaría con sumo cuidado y como último recurso.

Relaciones

Hasta ahora el enfoque en este capítulo se ha centrado en la relación y la manera de normalizar los datos. Sin embargo, un componente importante de cualquier base de datos relacional es de qué forma esas relaciones se asocian entre sí. Esas asociaciones, o *relaciones*, se vinculan en forma significativa, lo que contribuye a garantizar la integridad de los datos de modo que una acción realizada en una relación no repercuta negativamente en los datos de otra relación.

Hay tres tipos principales de relaciones:

- **Una a una** Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con al menos una tupla en la segunda relación, y una tupla en la segunda relación esté relacionada con al menos una tupla en la primera relación.
- **Una a varias** Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con ninguna, una o más tuplas en la segunda relación, pero una tupla en la segunda relación esté relacionada con al menos una tupla en la primera relación.
- **Varias a varias** Una relación entre dos relaciones en la cual una tupla en la primera relación esté relacionada con ninguna, una o más tuplas en la segunda relación, y una tupla en la segunda relación esté relacionada con ninguna, una o más tuplas en la primera relación.

La mejor manera de ilustrar estas relaciones es ver un modelo de datos de varias relaciones (mostrado en la figura 1-6). Las relaciones se nombraron para hacer referencia a ellas con mayor facilidad.

- Una relación una a una existe entre las relaciones AGENCIAS_ARTISTA y NOMBRES_ARTISTA. Para cada uno de los artistas que se listan en la relación AGENCIAS_ARTISTA, sólo puede haber una tupla correspondiente en la relación NOMBRES_ARTISTA, y viceversa. Esto implica una regla de negocio que un artista puede trabajar con sólo una agencia a la vez.

- Una relación una a varias existe entre las relaciones NOMBRE_ARTISTA y CD_ARTISTA. Para cada uno de los artistas en la relación NOMBRES_ARTISTA, ninguna, una o más tuplas para ese artista pueden estar en la lista en la relación CD_ARTISTA. En otras palabras, cada artista pudo haber hecho ninguno, uno o más CD. Sin embargo, para cada uno de los artistas que figuran en la relación CD_ARTISTA, sólo puede haber una tupla correspondiente para cada artista en la relación NOMBRES_ARTISTA debido a que cada artista puede tener sólo una tupla en la relación NOMBRES_ARTISTA.
- Una relación una a varias existe entre las relaciones CD_ARTISTA y DISCOS_COMPACTOS. Para cada CD, puede haber uno o más artistas; sin embargo, cada tupla en CD_ARTISTA puede corresponder en sólo una tupla en DISCOS_COMPACTOS debido a que cada CD sólo aparece una vez en la relación DISCOS_COMPACTOS.
- Una relación varias a varias existe entre las relaciones NOMBRE_ARTISTA y DISCOS_COMPACTOS. Para cada artista, puede haber ninguno, uno o más CD, y para cada CD, puede haber uno o más artistas.

NOTA

Las bases de datos relacionales sólo apoyan una relación una a varias directamente. Una relación varias a varias se implementa físicamente agregando una tercera relación entre la primera y la segunda para crear dos relaciones una a varias. En la figura 1-6, la relación CD_ARTISTA se agregó entre las relaciones NOMBRES_ARTISTA y DISCOS_COMPACTOS. Una relación una a una se aplica físicamente al igual que una relación una a varias, excepto que se añade una limitación para evitar duplicar los registros que coinciden en los “muchos” lados de la relación. En la figura 1-6 se añadió una limitación única en el atributo ID_ARTISTA para evitar que un artista aparezca en más de una agencia.

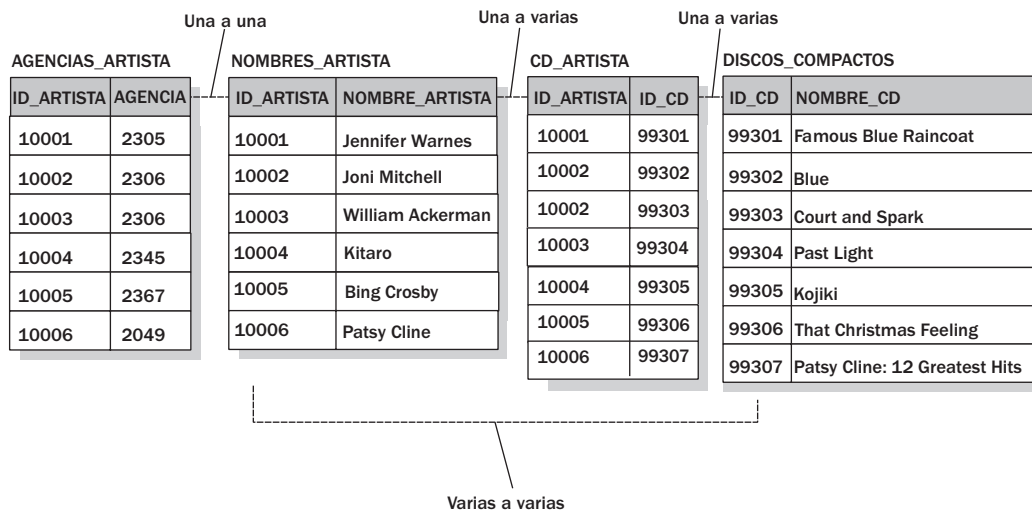


Figura 1-6 Tipos de relaciones entre relaciones.

Las relaciones se clasifican también por *cardinalidad mínima* (el número mínimo de tuplas que participan en una relación). Si cada tupla en una relación *debe* tener una tupla correspondiente en la otra, la relación se dice que es *obligatoria* en esa dirección. Del mismo modo, si cada tupla en una relación no requiere una tupla correspondiente en la otra, la relación se dice que es *opcional* en esa dirección. Por ejemplo, la relación entre NOMBRES_ARTISTA y AGENCIAS_ARTISTA es obligatoria-obligatoria porque cada artista debe tener una agencia y cada tupla de AGENCIAS_ARTISTA debe referirse a un único artista. Las reglas de negocio deben entenderse antes que la cardinalidad mínima se determine con certeza. Por ejemplo, ¿podemos tener un artista en la base de datos que en algún momento no tenga CD en la base de datos (es decir, ninguna tupla correspondiente en CD_ARTISTA)? Si es así, entonces la relación entre NOMBRES_ARTISTA y CD_ARTISTA es obligatoria-opcional; de lo contrario, es obligatoria-obligatoria.

Pregunta al experto

P: Se mencionó que las relaciones entre relaciones contribuyen a garantizar la integridad de los datos. ¿Cómo pueden las relaciones hacerlo posible?

R: Suponga que su modelo de datos incluye una relación (llamada NOMBRE_ARTISTA) que muestre en su inventario todos los artistas que han grabado CD. Su modelo también incluye una relación (CD_ARTISTA) donde coinciden los ID de los artistas con los ID de los discos compactos. Si existe una relación entre las dos relaciones, las tuplas de una relación siempre corresponderán a las tuplas de la otra relación. Como resultado, se podrán evitar ciertas acciones que puedan comprometer los datos. Por ejemplo, no sería posible añadir el ID de un artista a la relación CD_ARTISTA si ese ID no se muestra en la lista en la relación NOMBRE_ARTISTA. Tampoco sería posible borrar un artista de la relación NOMBRE_ARTISTA si el ID del artista hace referencia a la relación CD_ARTISTA.

P: ¿A qué se refiere con el término *modelo de datos*?

R: Con modelo de datos me refiero al diseño, presentado a menudo usando diagramas, que representan la estructura de la base de datos. El modelo identifica las relaciones, atributos, claves, dominios y relaciones dentro de la base de datos. Algunos diseñadores de bases de datos crean un modelo lógico y un modelo físico. El modelo lógico se basa más en la teoría relacional y aplica los principios de normalización apropiados a los datos. El modelo físico, por otro lado, se refiere a la aplicación real, ya que los datos se almacenan en un RDBMS. Basado en el diseño lógico, el diseño físico trae la estructura de datos al mundo real de la implementación.

Pruebe esto 1-1

Normalización de datos e identificación de relaciones

Como programador principiante de SQL, es poco probable que sea responsable de la normalización de la base de datos. Sin embargo, es importante que entienda estos conceptos, al igual que es importante que entienda las clases de tratos que existen entre relaciones. La normalización y las relaciones, como las relaciones mismas, ayudan a proporcionar la base sobre la que se construye SQL. Como resultado, el ejercicio se enfoca en el proceso de normalización de datos e identificación de las relaciones entre relaciones. Para completar el ejercicio, sólo necesitará papel y lápiz en el cual esbozar el modelo de datos.

Paso a paso

1. Revise la relación en la siguiente ilustración:

ID_CD	NOMBRE_CD	CATEGORIA
99301	Famous Blue Raincoat	Folk, Pop
99302	Blue	Folk, Pop
99304	Past Light	New Age
99305	Kojiki	New Age, Classical
99306	That Christmas Feeling	Christmas, Classics
99307	Patsy Cline: 12 Greatest Hits	Country, Pop, Classics

2. Identifique cualquier elemento que no se ajuste a las tres formas normales. Encontrará que el atributo CATEGORIA contiene más de un valor por tupla, que viola la primera forma normal.
3. Normalice los datos de acuerdo a las formas normales. Projete el modelo de datos que incluya las relaciones, atributos y tuplas apropiadas. Su modelo incluirá tres tablas, una para la lista de CD, otra para la lista de las categorías de música (por ejemplo, Pop), y otra que asocie los CD con las apropiadas categorías de música. Vea el archivo en línea Try_This_01-1a.jpg para un ejemplo de cómo el modelo de datos puede verse (en inglés).
4. En la ilustración que dibujó, identifique las relaciones entre la relaciones. Recuerde que cada CD se puede asociar con una o más categorías, y cada categoría se puede asociar con cero, uno o más CD. Vea el archivo en línea Try_This_01-1b.jpg para ver las relaciones entre relaciones (en inglés).

Resumen de Pruebe esto

Los modelos de datos suelen ser más específicos que las ilustraciones mostradas en el ejercicio. Las relaciones y claves se marcan claramente con símbolos que se ajustan a un tipo particular de sistema de modelado de datos, y las relaciones muestran sólo los atributos, pero no las tuplas. Sin embargo, para efectos de este capítulo, es suficiente con que tenga los conocimientos básicos de normalización y relaciones entre relaciones. El ejercicio se entiende como un forma para que entienda mejor esos conceptos y cómo se aplican al modelo relacional.

Aprenda acerca de SQL

Ahora que tiene un conocimiento fundamental del modelo relacional, es el momento para introducirlo a SQL y sus características básicas. Como recordará de la sección “Entienda las bases de datos relacionales” vista anteriormente en este capítulo, SQL se basa en el modelo relacional, aunque no se trate de una aplicación exacta. Mientras el modelo relacional proporciona las bases teóricas de la base de datos relacional, es el lenguaje SQL el que apoya la aplicación física de esa base de datos.

SQL, el lenguaje relacional casi universalmente aplicado, es diferente de otros lenguajes computacionales como C, COBOL y Java, los cuales son de procedimiento. Un lenguaje de procedimiento define *cómo* las operaciones de una aplicación deben realizarse y el orden en el cual se realizan. Un lenguaje de no procedimiento, por otro lado, se refiere a los resultados de una operación; el entorno fundamental del software determina cómo se procesan las operaciones. Esto no quiere decir que SQL respalda a la funcionalidad de no procedimiento. Por ejemplo, los procedimientos almacenados, agregados a varios productos RDBMS hace algunos años, son parte del estándar SQL:2006 y proporciona capacidades parecidas a procedimiento. (Los procedimientos almacenados se analizan en el capítulo 13.) Muchos de los proveedores de RDBMS añadieron extensiones a SQL para proporcionar esas capacidades de procedimiento, como Transact-SQL encontrado en Sybase y Microsoft SQL Server y PL/SQL encontrado en Oracle.

SQL aún carece de muchas de las capacidades básicas de programación de la mayoría de los lenguajes computacionales. Por esta razón, a menudo SQL se considera como un *sublenguaje* de datos porque se utiliza con frecuencia en asociación con la aplicación de lenguajes de programación como C y Java, lenguajes que no fueron diseñados para la manipulación de datos almacenados en una base de datos. Como resultado, SQL se utiliza en conjunto con la aplicación del lenguaje para proporcionar un medio eficaz de acceder a los datos, razón por la cual se considera a SQL como un sublenguaje.

La evolución de SQL

A principios de la década de 1970, después que se publicó el artículo de E. F. Codd, IBM comenzó a desarrollar un lenguaje y un sistema de base de datos que podría usarse para la aplicación de ese modelo. Cuando se definió por primera vez, el lenguaje fue denominado Lenguaje de consulta estructurado (en inglés, SEQUEL, Structured English Query Language). Cuando se descubrió que SEQUEL era propiedad de una marca comercial de Hawker-Siddeley Aircraft Company en el Reino Unido, el nombre se cambió a SQL. Cuando se pasó la voz de que IBM estaba desarrollando

un sistema de base de datos relacional basado en SQL, otras compañías comenzaron a desarrollar sus propios productos basados en SQL. De hecho, Relational Software, Inc., ahora Oracle Corporation, lanzó el sistema de base de datos antes de que IBM lanzara el suyo al mercado. Conforme más proveedores lanzaron sus productos, SQL comenzó a surgir como el lenguaje estándar de base de datos relacional.

En 1986, el American National Standards Institute (ANSI) dio a conocer el primer estándar publicado para el lenguaje (SQL-86), el cual fue adoptado por la International Organization for Standardization (ISO) en 1987. El estándar se actualizó en 1989, 1992, 2003, 2006, y el trabajo continúa. Ha crecido con el tiempo (el estándar original estaba muy por debajo de 1 000 páginas, mientras que la versión de SQL:2006 tiene más de 3 700 páginas). El estándar se escribió en partes para permitir la publicación programada de revisiones y facilitar el trabajo paralelo por diferentes comités. La tabla 1-1 proporciona una descripción general de las partes y el estado actual de cada una.

Partes	Tema	Estado
1	SQL/Framework	Completado en 1999; revisado en el 2003, correcciones publicadas en el 2007
2	SQL/Foundation	Completado en 1986, revisado en 1999 y 2003, correcciones publicadas en el 2007
3	SQL/CLI	Completado en 1995, revisado en 1999 y en el 2003, correcciones publicadas en el 2005
4	SQL/PSM	Completado en 1996, revisado en 1999 y en el 2003, correcciones publicadas en el 2007
5	SQL/Bindings	Establecido como una parte separada en 1999, fusionado de nuevo en la parte 2 en el 2003; actualmente no existe la parte 5
6	SQL/Transaction	Proyecto cancelado; actualmente no existe la parte 6
7	SQL/Temporal	Retirado; actualmente no existe la parte 7
8	SQL/Objects and Extended Objects	Fusionado en la parte 2; no existe la parte 8
9	SQL/MED	Comenzado después de 1999, completado en el 2003, correcciones publicadas en el 2005
10	SQL/OLB	Completado como el estándar ANSI en 1998, versión ISO completada en 1999, revisión en el 2003, correcciones publicadas en el 2007
11	SQL/Schemata	Extraído de una parte separada en el 2003, correcciones publicadas en el 2007
12	SQL/Replication	Proyecto empezado en el 2000, pero posteriormente abandonado; actualmente no existe la parte 12
13	SQL/JRT	Completado como el estándar ANSI en 1999, revisión completada en el 2003, correcciones publicadas en el 2005
14	SQL/XML	Completado en el 2003, expandido en el 2006, correcciones publicadas en el 2007

Tabla 1-1 Partes del estándar SQL.

Los proveedores RDBMS lanzaron productos al mercado antes que hubiera un estándar, y muchas de las características de esos productos se aplicaron bastante diferentes, así que el estándar no pudo acomodarlos a todos cuando se desarrollaron. A menudo se llaman *proveedores por extensión*. Esto explica por qué no hay un estándar para una base de datos. Y cuando cada versión estándar de SQL se lanza, los proveedores RDBMS tienen que trabajar para incorporar el nuevo estándar en sus productos. Por ejemplo, los procedimientos almacenados y activadores son nuevos en el estándar SQL:1999, pero fueron implementados en RDBMS por muchos años. SQL:1999 simplemente estandarizó el lenguaje utilizado para implementar funciones que ya existían.

NOTA

Aunque se analizarán los procedimientos almacenados en el capítulo 13 y los activadores en el capítulo 14, daré una definición rápida de cada uno. Un procedimiento almacenado es un conjunto de instrucciones de SQL almacenadas como un objeto en el servidor de una base de datos al que un cliente puede recurrir simplemente llamando al procedimiento. Un activador es similar a un procedimiento almacenado en el sentido que es un conjunto de instrucciones de SQL almacenadas como un objeto en el servidor de una base de datos. Sin embargo, en lugar de que un cliente recurra al activador, éste es invocado automáticamente cuando se produce un evento predefinido, tales como la inserción o la actualización de datos.

Objeto de modelo relacional

El lenguaje SQL se basa en el modelo relacional, y hasta SQL-92, también el estándar SQL. Sin embargo, comenzando con SQL:1999, el estándar SQL se extendió más allá del modelo relacional puro para incluir construcciones orientadas a objetos en el lenguaje. Estas construcciones se basan en los conceptos inherentes de *programación orientada a objetos*, una programación metodológica que define colecciones autónomas de estructura de datos y rutinas (llamadas *objetos*). En los lenguajes orientados a objetos como Java y C++, los objetos interactúan entre sí de manera que permiten al lenguaje abordar problemas complejos que no serían fáciles de resolver en lenguajes tradicionales.

Con la llegada de la programación orientada a objetos (junto con los avances tecnológicos en el hardware y software y la creciente complejidad de aplicaciones) se hizo cada vez más evidente que un lenguaje puramente relacional era insuficiente para satisfacer las demandas del mundo real. De preocupación específica fue el hecho que SQL no podía respaldar tipos de datos complejos y definidos por el usuario ni la extensibilidad requerida para aplicaciones más complejas.

Impulsados por la competencia natural de la industria, los proveedores RDBMS se encargaron de aumentar sus productos e incorporar la funcionalidad orientada a objetos en sus sistemas. El estándar SQL:2006 sigue el ejemplo y extiende el modelo relacional con capacidades orientadas a objetos, como métodos, encapsulación, y tipos de datos complejos y definidos por el usuario, lo que hace a SQL un lenguaje de base de datos *relacional a objeto*. Como se muestra en la tabla 1-1, la parte 14 (SQL/XML) se amplió considerablemente y se reeditó con SQL:2006, y todas las demás partes se tomaron de SQL:2003.

La conformidad con SQL:2006

Una vez que SQL se estandarizó, se desprende que el estándar también define lo que se tomó para una aplicación de SQL (un producto RDBMS) considerado en conformidad al estándar. Por ejem-

plo, el estándar SQL-92 proporcionó tres niveles de conformidad: entrada, intermedio y pleno. Los RDBMS más populares alcanzaron sólo el nivel de entrada de conformidad. Debido a esto, SQL:2006 tomó un enfoque diferente al establecer las normas de conformidad. Para que un producto esté en conformidad con SQL:2006, debe apoyar el nivel de conformidad Core SQL. Core SQL en el estándar SQL:2006 se define en conformidad a la parte 2 (SQL/Foundation) y la parte 11 (SQL/Schemata) del estándar.

Además del nivel de conformidad Core SQL, los proveedores demandan la conformidad en cualquier parte cumpliendo con los requisitos mínimos de conformidad por su parte.

NOTA

Puede encontrar información sobre el estándar SQL:2006 comprando una copia del documento(s) estándar apropiado publicado por ANSI e ISO. El estándar se divide en nueve documentos (una parte por documento). El primer documento (ANSI/ISO/IEC 9075-1:2003) incluye la descripción general de las nueve partes. El sufijo del nombre de cada documento contiene el año de publicación, y las diferentes partes tienen distintos años de publicación debido a que las partes fueron actualizadas y publicadas independientemente por distintos comités. Como puede ver en la tabla 1-1, la parte 1 fue publicada en el 2003 y, de hecho, sólo la parte 14 lleva una fecha de publicación del 2006 (todas las demás partes se publicaron en el 2003). Puede adquirir estos documentos en línea en ANSI Electronic Standards Store (<http://webstore.ansi.org/>), NCITS Standards Store (<http://www.techstreet.com/ncitsgate.html>) o ISO Store (<http://www.iso.org/iso/store.htm>). En el sitio de la ANSI observe que hay dos variantes de cada documento con un contenido esencialmente idéntico, llamado INCITS/ISO/IEC 9075 e ISO/IEC 9075. Las variantes ISO/IEC cuestan entre 139 y 289 dólares por documento, mientras que las variantes INCITS/ISO/IEC cuestan entre 30 dólares por documento. ISO Store tiene el conjunto de documento disponible en un cómodo CD por 356 francos suizos (alrededor de 350 dólares). Obviamente, los precios están sujetos a cambio en cualquier momento. También están disponibles las correcciones sin cargo extra, llamadas "Correcciones técnicas". Como se muestra en la tabla 1-1, se corrigieron tres partes publicadas en 2005, y también se corrigieron las otras seis partes publicadas en 2007.

Tipos de instrucciones de SQL

Aunque SQL se considera un sublenguaje debido a su naturaleza de no procesamiento, aun así es un lenguaje completo que le permite crear y mantener objetos en una base de datos, asegurar esos objetos y manipular la información dentro de los objetos. Un método común usado para categorizar las instrucciones SQL es dividir las de acuerdo con las funciones que realizan. Basado en este método, SQL se separa en tres tipos de instrucciones:

- **Lenguaje de definición de datos (DDL, Data Definition Language)** Las instrucciones DDL se usan para crear, modificar o borrar objetos en una base de datos como tablas, vistas, esquemas, dominios, activadores, y almacenar procedimientos. Las palabras clave en SQL más frecuentemente asociadas con las instrucciones DDL son CREATE, ALTER y DROP. Por ejemplo, se usa la instrucción CREATE TABLE para crear una tabla, la instrucción ALTER TABLE para modificar las características de una tabla, y la instrucción DROP TABLE para borrar la definición de la tabla de la base de datos.

- **Lenguaje de control de datos (DCL, Data Control Language)** Las instrucciones DCL permiten controlar quién o qué (un usuario en una base de datos puede ser una persona o un programa de aplicación) tiene acceso a objetos específicos en la base de datos. Con DCL, puede otorgar o restringir el acceso usando las instrucciones GRANT o REVOKE, los dos comandos principales en DCL. Las instrucciones DCL también permiten controlar el tipo de acceso que cada usuario tiene a los objetos de una base de datos. Por ejemplo, puede determinar cuáles usuarios pueden ver un conjunto de datos específico y cuáles usuarios pueden manipular esos datos.
- **Lenguaje de manipulación de datos (DML, Data Manipulation Language)** Las instrucciones DML se usan para recuperar, agregar, modificar o borrar datos almacenados en los objetos de una base de datos. Las palabras clave asociadas con las instrucciones DML son SELECT, INSERT, UPDATE y DELETE, las cuales representan los tipos de instrucciones que probablemente son más usadas. Por ejemplo, puede usar la instrucción SELECT para recuperar datos de una tabla y la instrucción INSERT para agregar datos a una tabla.

La mayoría de las instrucciones SQL que se utilizan caen perfectamente en una de estas categorías, y estas instrucciones se analizarán durante el resto del libro.

NOTA

Hay varias formas de clasificar las instrucciones además de la manera en que se clasifican en la lista anterior. Por ejemplo, se pueden clasificar de acuerdo a cómo se ejecutan o si pueden o no ser incrustadas en un lenguaje de programación estándar. El estándar SQL proporciona diez categorías amplias basadas en funciones. Sin embargo, se usa el método anterior ya que se utiliza comúnmente en la documentación relacionada con SQL, y porque es una manera simple de proporcionar una buena visión general de la funciones inherentes a SQL.

Tipos de ejecución

Además de definir cómo se usa el lenguaje, el estándar SQL proporciona detalles de cómo las instrucciones SQL son ejecutadas. Este método de ejecución, conocido como *estilos de unión*, no sólo afecta la naturaleza de la ejecución, sino también determina cuáles instrucciones, como mínimo, deben ser soportadas por un estilo de unión particular. El estándar define cuatro métodos de ejecución:

- **Invocación directa** Mediante el uso de este método, puede comunicarse directamente desde una aplicación de usuario, como iSQL*Plus en Oracle o Management Studio en Microsoft SQL Server, en la base de datos. (La aplicación de usuario y la base de datos pueden estar en la misma computadora, pero a menudo no lo están.) Simplemente introduzca su consulta en la ventana de la aplicación y ejecute la instrucción SQL. Los resultados de su consulta se le devolverán tan rápido como el poder del procesador y las limitaciones de la base de datos lo permitan. Ésta es una forma rápida de comprobar datos, verificar conexiones y ver los objetos en una base de datos. Sin embargo, las directrices del estándar SQL sobre la invocación directa son bastante mínimas; por lo tanto, los métodos utilizados y los estándares SQL respaldados pueden variar ampliamente de un producto a otro.

- **SQL incrustado** En este método, las instrucciones SQL están codificadas (incrustadas) directamente en el lenguaje de programación anfitrión. Por ejemplo, las instrucciones SQL se pueden incrustar en el código C de la aplicación. Antes que el código se compile, un pre-procesador analiza las instrucciones SQL y las desglosa desde el código C. El código SQL se convierte en una forma que RDBMS puede entender, y el código C restante se compila como lo haría normalmente.
- **Unión de módulo** Este método permite crear bloques de instrucciones SQL (módulos) que están separados del lenguaje de programación anfitrión. Una vez que el módulo es creado, es una combinación entre una aplicación y un vinculador. Un módulo contiene, entre otras cosas, procedimientos, y son los procedimientos los que contienen las instrucciones SQL reales.
- **Interfaz convocatoria a nivel (CLI, Call-level interface)** Una CLI permite invocar instrucciones SQL a través de una interfaz mediante la aprobación de instrucciones SQL como valores argumentativos para las subrutinas. Las instrucciones no están precompiladas como en el SQL incrustado y la Unión de módulo. En lugar de eso, son ejecutadas directamente por los RDBMS.

La invocación directa, aunque no es el método utilizado más comúnmente, es el que será usado principalmente para los ejemplos y ejercicios en este libro, ya que apoya la presentación de las consultas *ad hoc* en la base de datos y genera resultados inmediatos. Sin embargo, SQL incrustado es actualmente el método utilizado más comúnmente en las aplicaciones de negocios. Se analizará este método, así como el unión de módulo y CLI, con mayor detalle en el capítulo 17.

Pregunta al experto

- P:** Se afirma que, para que un RDBMS esté en conformidad con el estándar SQL:2006, debe cumplir con Core SQL. ¿Hay algún requisito adicional que un producto debe cumplir?
- R:** Sí. Además de Core SQL, un RDBMS debe respaldar a ambos el SQL incrustado o la unión de módulo. La mayoría de los productos respaldan sólo al SQL incrustado, algunos respaldan a ambos. El estándar SQL no requiere productos RDBMS para respaldar la invocación directa o CLI, aunque las mayoría lo hace.
- P:** ¿Cuáles son las diez categorías usadas por el estándar SQL:2006 para clasificar las instrucciones SQL?
- R:** El estándar SQL clasifica las instrucciones en las siguientes categorías: esquema, datos, cambio de datos, operación, conexión, control, sesión, diagnósticos, dinámico e instrucción incrustada de excepción. Tenga en cuenta que estas clasificaciones no son más que una herramienta que puede utilizar para comprender mejor el alcance del lenguaje y los conceptos básicos. En última instancia, son las instrucciones SQL (y lo que pueden hacer) lo que es importante.

Utilice un sistema de gestión de base de datos relacional

A lo largo de este capítulo, al examinar el modelo relacional y SQL, se mencionaron a menudo los RDBMS y la forma en que utilizan el estándar SQL como base de sus productos. Un *sistema de gestión de base de datos relacional* es un programa o conjunto de programas que almacenan, administran, recuperan, modifican y manipulan información en una o más bases de datos relacionales. Oracle, Microsoft SQL Server, IBM's DB2 y el producto de prueba MySQL son todos ejemplos de RDBMS. Estos productos, como otros RDBMS, permiten interactuar con la información almacenada en el sistema. Aunque no es necesario que un RDBMS se base en SQL, la mayoría de los productos en el mercado están basados en SQL y tratan de ajustarse al estándar SQL. Como mínimo, estos productos demandan el nivel de entrada de conformidad con el estándar SQL-92, y ahora se trabaja para Core SQL en conformidad con SQL:2006.

Además del cumplimiento con los estándares SQL, la mayoría de los RDBMS apoyan otras características, como instrucciones SQL adicionales, productos basados en herramientas administrativas, y aplicaciones de interfaz gráfica de usuario (GUI) que permiten la consulta y manipulación de información, la gestión de objetos en una base de datos, y la administración del sistema y su estructura. Los tipos de funcionalidad aplicados y los métodos utilizados para proporcionar la funcionalidad pueden variar ampliamente de un producto a otro. Conforme las bases de datos crecen, se vuelven más complicadas y se distribuyen en más áreas, los productos RDBMS utilizados para la gestión de las bases de datos se hacen más complejos y robustos, satisfaciendo las demandas del mercado, así como aplicación de nuevas y más sofisticadas tecnologías.

El estándar SQL frente a las implementaciones de producto

El centro de cualquier RDBMS basado en SQL es, por supuesto, el propio SQL. Sin embargo, el lenguaje utilizado no es SQL puro. Cada producto extiende su lenguaje con el fin de implementar las características definidas por el proveedor y mejorar la funcionalidad basada en SQL. Además, una serie de productos RDBMS lo fabricaron para el mercado antes de que hubiera un estándar. En consecuencia, cada proveedor respalda una variación ligeramente diferente de SQL, lo que significa que el lenguaje utilizado en cada producto tiene una aplicación específica. Por ejemplo, SQL Server utiliza Transact-SQL, que incluye tanto a SQL como a las extensiones del proveedor para proporcionar las instrucciones de procedimiento necesarias para los activadores y los procedimientos almacenados. Por otro lado, Oracle proporciona instrucciones de procedimiento en un componente de producto separado llamado PL/SQL. Como resultado, las instrucciones SQL que se proporcionan en el libro pueden ser ligeramente diferentes en la aplicación del producto que utiliza.

A lo largo del libro se utilizará SQL puro en la mayoría de los ejemplos y ejercicios. Sin embargo, me doy cuenta de que, como un programador principiante de SQL, su interés principal es la aplicación de SQL en el mundo real. Por esa razón, a veces usaré SQL Server (con Transact-SQL) u Oracle (con PL/SQL) para demostrar o aclarar un concepto particular que no puede explicarse totalmente por medio de SQL puro únicamente.

Una de las ventajas de utilizar un producto como Oracle o SQL Server es que ambos respaldan la invocación directa a través de una aplicación GUI de usuario. SQL Server usa la interfaz Management Studio, que se muestra en la figura 1-7. La interfaz GUI hace posible crear una consulta específica en SQL, presentarla a DBMS para la transformación y ver los resultados, lo que le permite aplicar lo que se aprende en el libro en un entorno real de SQL. Oracle tiene varias soluciones para un GUI de usuario, incluyendo el interfaz basado en la web *iSQL** Plus, mostrado en la figura 1-8.

Además de las interfaces GUI, la mayoría de los productos incluyen una interfaz de línea de comandos que se utiliza en las terminales más antiguas que no tienen capacidad gráfica. Estas interfaces también son útiles para la ejecución de instrucciones SQL y por las conexiones de acceso telefónico donde las interfaces gráficas son demasiado lentas. En la figura 1-9 se muestra la interfaz de línea de comandos por el cual MySQL se ejecuta en Microsoft Windows.

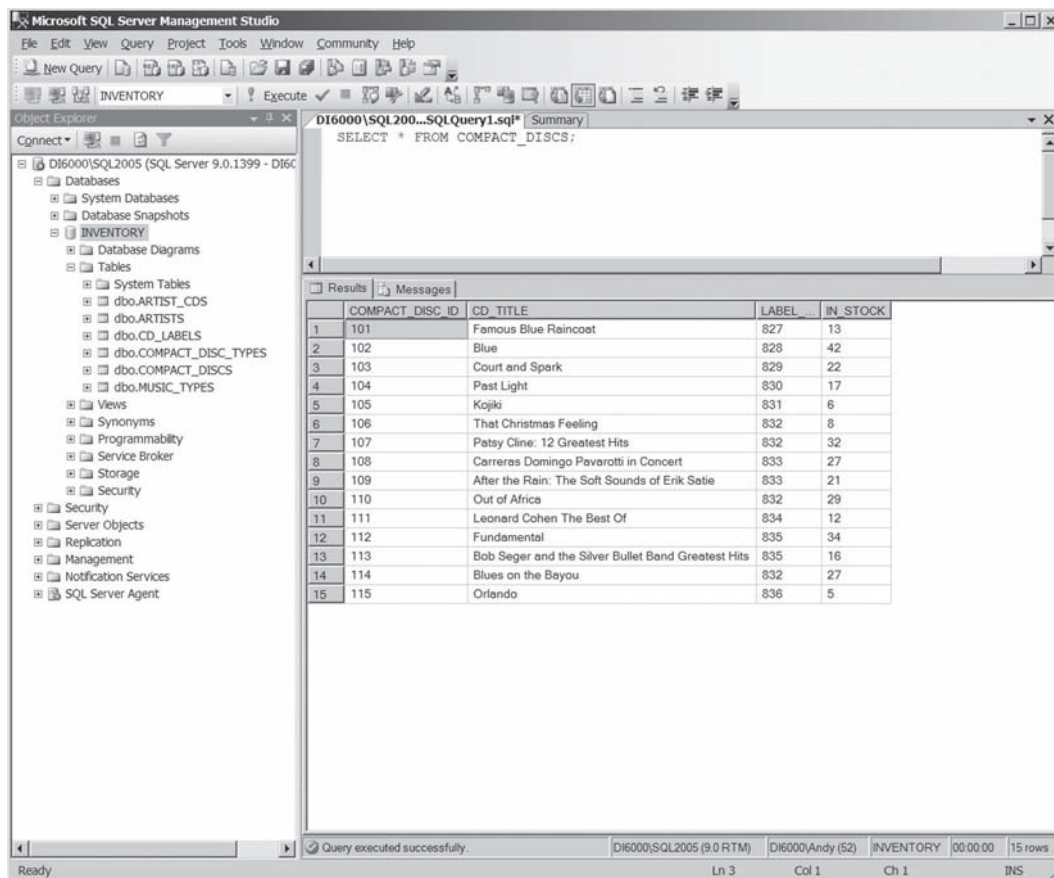


Figura 1-7 Uso de SQL Server Management Studio.

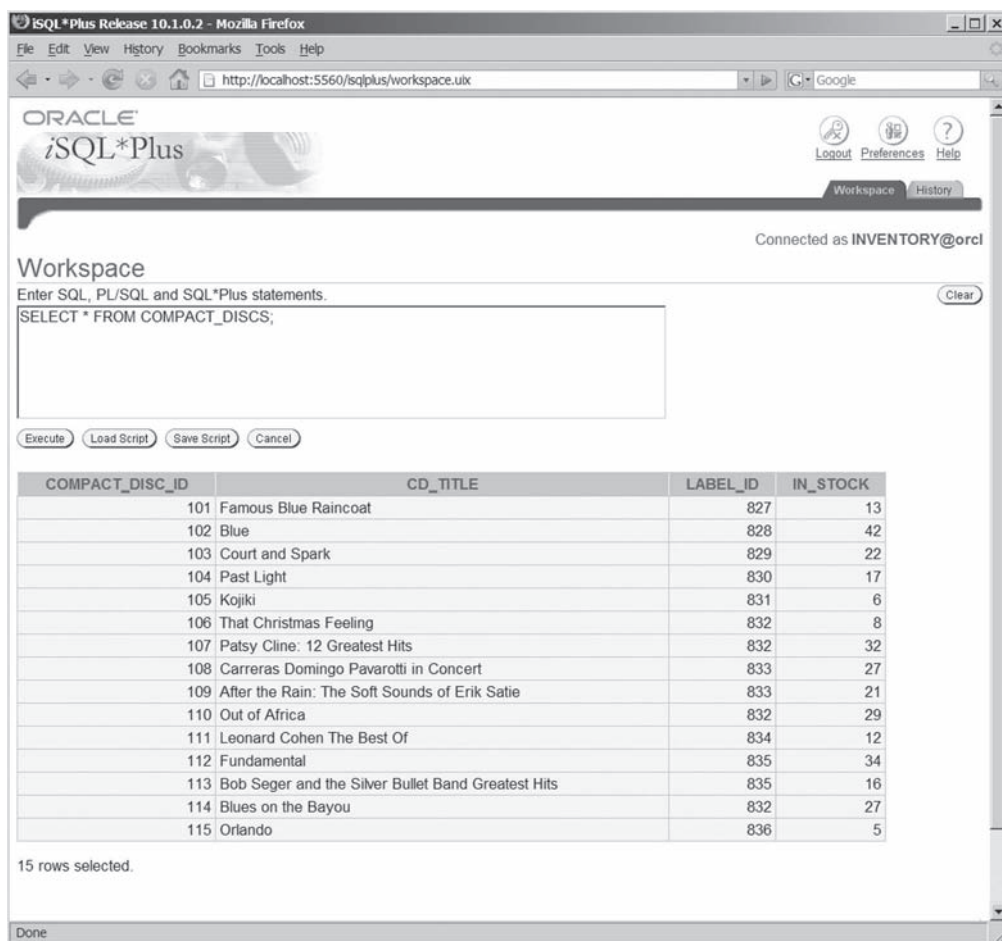
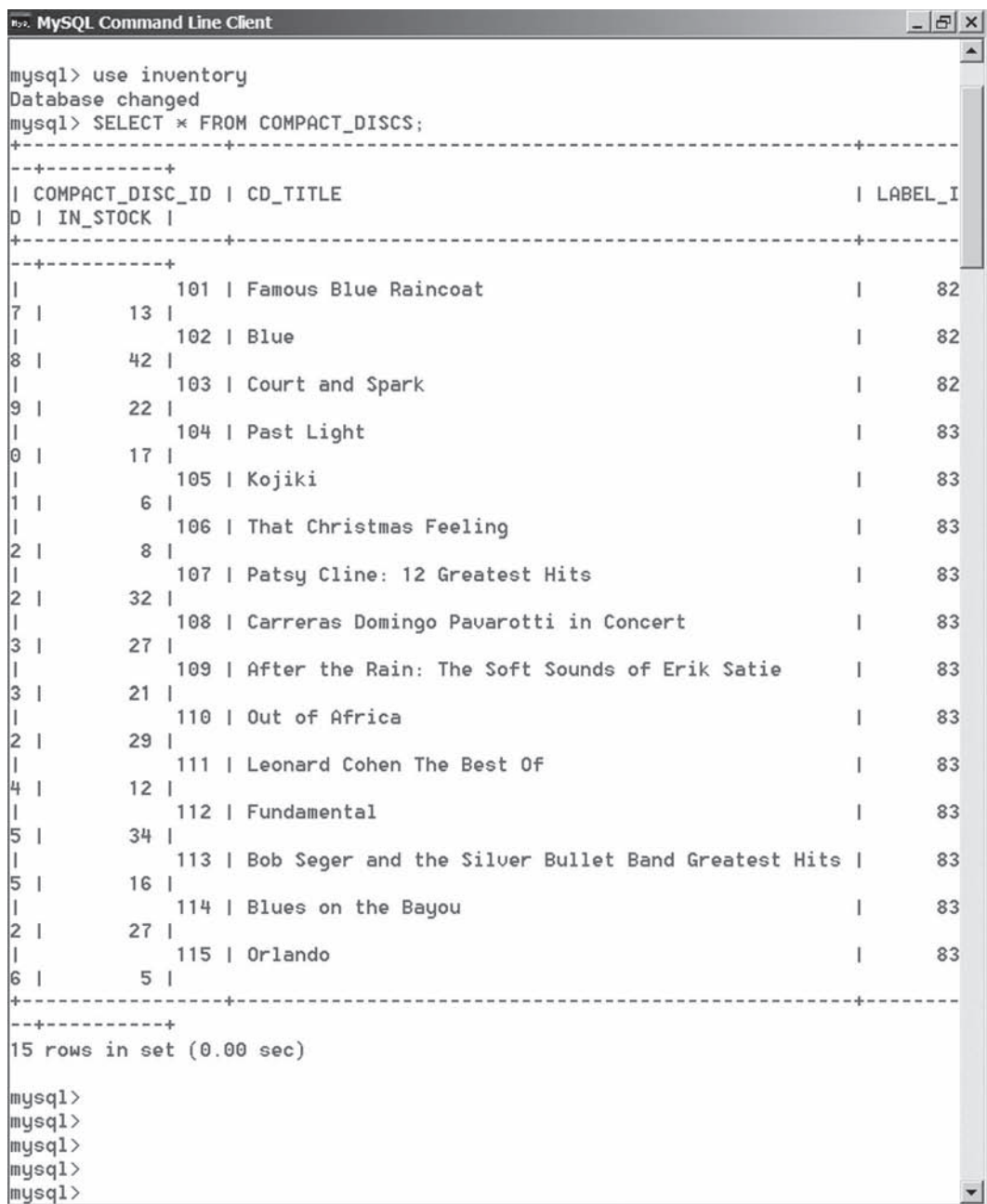


Figura 1-8 Uso de Oracle iSQL*Plus.

El uso de estos productos no implica que se apoye a alguno de ellos por encima de cualquier otro producto comercial (como Sybase o Informix) o los productos de prueba (como PostgreSQL), y de hecho, le animamos a utilizar cualquier RDBMS que tenga disponible, suponiendo que da soporte a gran parte de la funcionalidad que se analiza en este libro. Sin embargo, se eligen SQL Server y Oracle para la mayoría de los ejemplos en el libro, ya que se demostrará cómo se aplica SQL en el mundo real y cómo SQL puede diferir de una aplicación específica de una versión del lenguaje, y esos dos productos suministran el medio para llevarlo a cabo. Tenga en cuenta que, para que pueda tener una plena comprensión de SQL y pueda utilizarlo en varios productos RDBMS, necesitará entender tanto el estándar SQL como el lenguaje que se aplica a los productos que van a utilizar.



```
mysql> use inventory
Database changed
mysql> SELECT * FROM COMPACT_DISCS;
```

COMPACT_DISC_ID	CD_TITLE	LABEL_I
101	Famous Blue Raincoat	82
102	Blue	82
103	Court and Spark	82
104	Past Light	83
105	Kojiki	83
106	That Christmas Feeling	83
107	Patsy Cline: 12 Greatest Hits	83
108	Carreras Domingo Pavarotti in Concert	83
109	After the Rain: The Soft Sounds of Erik Satie	83
110	Out of Africa	83
111	Leonard Cohen The Best Of	83
112	Fundamental	83
113	Bob Seger and the Silver Bullet Band Greatest Hits	83
114	Blues on the Bayou	83
115	Orlando	83

```
15 rows in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>
```

Figura 1-9 Uso de la interfaz de línea de comandos MySQL.

Pruebe esto 1-2 Conexión a una base de datos

Aunque este libro se centra principalmente en SQL puro, para probar los ejemplos y hacer la mayoría de los ejercicios, necesitará acceder a un RDBMS para ejecutar las instrucciones SQL. Como resultado, una de las primeras cosas que debe hacer es asegurarse que pueda acceder a un entorno SQL. Este ejercicio le ayudará a hacerlo; sin embargo, a diferencia de la mayoría de los otros ejercicios en el libro, éste requerirá un mayor esfuerzo de su parte para buscar recursos fuera de la obra para establecerse con un RDBMS que le permita invocar las instrucciones SQL directamente. Con este fin, el ejercicio intenta que empiece, pero debe utilizar su propia iniciativa para garantizar que tiene un entorno en el que se sienta cómodo trabajando.

Paso a paso

1. Identifique el RDBMS que va a utilizar para los ejercicios del libro. Quizás haya un sistema con el cual ya está familiarizado o uno que esté disponible en su entorno de trabajo. Si no tiene ninguno disponible en su trabajo y no está listo para comprar un producto, revise en línea para ver cuál está disponible. La mayoría de los proveedores RDBMS ofrecen una versión gratuita básica de uso restringido de sus productos, a menudo denominada “Express Edition” (la edición sencilla); otros ofrecen una versión de prueba que puede usarse gratis por un periodo limitado. Por ejemplo, puede descargar Oracle Express Edition para Linux o Windows en <http://www.microsoft.com/sql/editions/express/>. (Necesitará una conexión a Internet de alta velocidad para descargar archivos de gran tamaño.)

Si prefiere un producto de prueba, MySQL es una opción popular. De hecho, tanto Yahoo como Google utilizan MySQL ampliamente en una configuración conocida como LAMP (Linux, Apache, MySQL y PHP). Puede descargar MySQL Community Server gratis de <http://dev.mysql.com/downloads/mysql>. Por otro lado, MySQL Enterprise Server es una edición con una tarifa respaldada por el proveedor.

Antes que se decida por un producto en particular, haga las investigaciones necesarias para asegurarse de que admite la invocación directa, preferiblemente a través de una aplicación GUI, y pueda ejecutarse en su computadora. También compruebe qué tanto del estándar SQL:2006 respalda y revise los acuerdos de licencia para asegurarse de que está en el cumplimiento. Si un sistema está disponible en su trabajo, asegúrese de hablar con los administradores de base de datos y de red para determinar qué servidor puede usar, si puede y cómo debe descargar una copia, y cómo hacer la conexión con el servidor SQL. A menudo se necesita una cuenta para conectarse a la RDBMS, por lo que si éste es el caso, averigüe qué nombre de usuario y contraseña debe usar.

2. Una vez que establezca cuál RDBMS usará, instálelo en su equipo. Si se conecta a un sistema a través de la red, tendrá que instalar sólo las herramientas de clientes en su equipo local.
3. Abra el cliente GUI que le permita invocar directamente las instrucciones SQL. Cuando abra el GUI, es posible que se le pida un nombre de usuario y contraseña. Cuándo y si se le pide puede variar dependiendo del producto que utiliza, si se conecta a través de la red, si el RDBMS se configura como un sistema autónomo, y otras variables específicas del producto. Además, un producto como SQL Server ofrece seguridad integrada con el sistema operativo, por lo que es posible que sólo se le pida un nombre de servidor.

(continúa)

4. Ejecute la instrucción `SELECT` en la aplicación de entrada de la ventana. Me doy cuenta que aún no se cubren las instrucciones `SELECT`, pero la sintaxis básica es relativamente fácil:

```
SELECT * FROM <tabla>
```

El marcador de posición `<tabla>` debe sustituirse con el nombre de una tabla en una base de datos existente.

El propósito de este ejercicio es simplemente comprobar que tiene conectividad con los datos almacenados en el RDBMS. La mayoría de los productos incluyen información de muestra, y esa información es con la que intentará conectarse. Compruebe la documentación del producto o consulte con el administrador de bases de datos para verificar si existe una base de datos que pueda acceder.

Si trabaja en Oracle y los esquemas de muestra están instalados, puede ejecutar las siguientes instrucciones:

```
SELECT * FROM scott.emp;
```

Para ejecutar las instrucciones, escríbalas en la ventana de entrada de `iSQL*Plus` y luego dé un clic en `EXECUTE`.

Si trabaja en SQL Server con la información muestra instalada, puede ejecutar la siguiente instrucción:

```
USE pubs
SELECT * FROM employee
```

Para ejecutar la instrucción, escríbala en la ventana de entrada de SQL Server Management Studio y luego dé un clic en `EXECUTE`.

Si trabaja en el interfaz de línea de comandos MySQL, simplemente escriba la instrucción SQL y oprima `ENTER`.

Una vez que ejecute la instrucción, los resultados de la consulta aparecerán en la ventana de salida. En este momento, no se preocupe por el significado de cada palabra en la instrucción SQL o con los resultados de la consulta. Su única preocupación es asegurarse que todo funcione. Si no puede ejecutar la instrucción, consulte con el administrador de bases de datos o la documentación del producto.

5. Cierre la aplicación GUI sin guardar la consulta.

Resumen de Pruebe esto

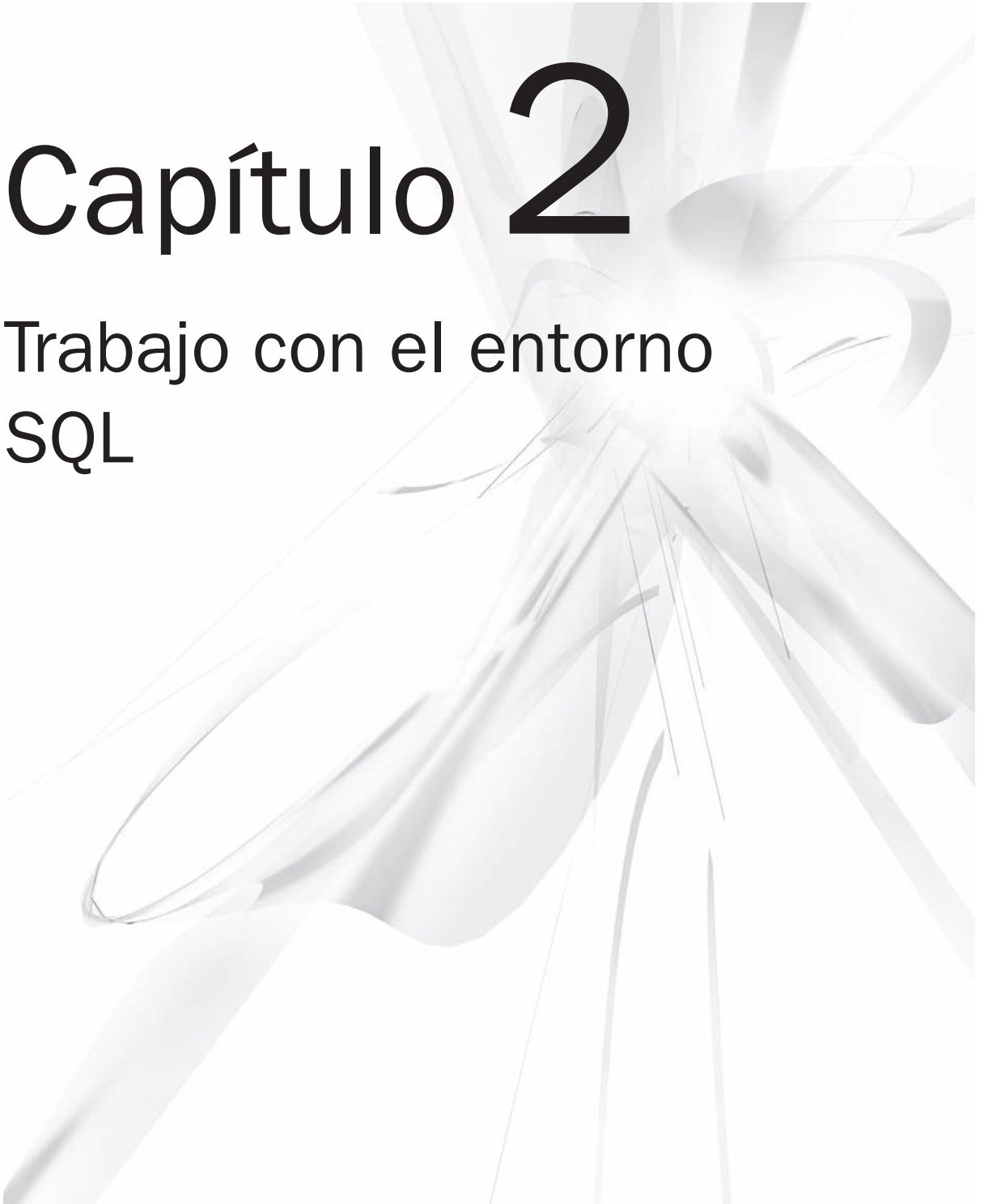
Como se dijo al principio del ejercicio, es diferente de la mayoría de los demás en el libro debido a que está básicamente por su propia cuenta para establecer la conectividad con el RDBMS. De nuevo, esto se debe a que SQL es un lenguaje estándar, independiente de las implementaciones RDBMS, y las cuestiones específicas de los proveedores están, en su mayor parte, más allá del alcance de este libro. Además, los métodos utilizados para conectarse a una base de datos, las herramientas disponibles para hacer esas conexiones y la forma en la que un RDBMS está configurado varía de un producto a otro, de un entorno a otro e incluso de un sistema operativo a otro. Sin embargo, el tiempo que tome ahora para investigar qué producto utilizar y para asegurarse de que puede conectar la información existente en una base de datos será de valor incalculable conforme aplique información analizada en el resto del libro.

✓ Autoexamen Capítulo 1

1. ¿Qué es una base de datos?
2. ¿Cuál de los siguientes objetos conforma una relación?
 - A Tipos de datos
 - B Tuplas
 - C Atributos
 - D Formas
3. Un(a) _____ es un conjunto de datos cuyos valores conforman una instancia de cada uno de los atributos definidos para esa relación.
4. ¿Cuáles son las diferencias entre la primera forma normal y la segunda forma normal?
5. Una relación está en la tercera forma normal si está en la segunda forma normal y si cumple con las demás directrices de esa forma. ¿Cuáles son esas directrices?
6. ¿Cuáles son los tres principales tipos de relaciones soportados por una base de datos relacional?
7. En el modelo de datos, hay dos relaciones asociadas cada una entre sí por una relación varias a varias. ¿Cómo se implementa físicamente esta relación en una base de datos relacional?
8. ¿Cómo se diferencia SQL de los lenguajes de programación como C, COBOL y Java?
9. ¿Qué factores contribuyeron a que el estándar SQL:2006 incorporara capacidades orientadas a objetos?
10. ¿Qué nivel de conformidad debe soportar un RDBMS para poder cumplir con SQL:2006?
 - A Entrada
 - B Core
 - C Pleno
 - D Intermedio
11. ¿Cuáles son las diferencias entre las instrucciones DDL y DML?
12. ¿Qué método de ejecución de instrucciones SQL usaría si deseara comunicarse directamente con una base de datos SQL desde una aplicación de usuario?
13. ¿Cuáles son los cuatro métodos que soporta el estándar SQL:2006 para la ejecución de las instrucciones SQL?
14. ¿Qué es un sistema de gestión de base de datos relacional?
15. ¿Cuál es un ejemplo de un RDBMS?

Capítulo 2

Trabajo con el entorno SQL



Habilidades y conceptos clave

- Entienda el entorno SQL
 - Entienda los catálogos SQL
 - Nombrado de objetos en un entorno SQL
 - Creación de un esquema
 - Creación de una base de datos
-

En el capítulo 1 se analizó la teoría relacional, SQL, y los sistemas de gestión de base de datos relacional (RDBMS). En este capítulo deseo llevar este análisis un paso más allá e introducir el entorno SQL, como se define en el estándar SQL:2006. El entorno SQL proporciona la estructura en la que se ejecuta SQL. Dentro de esta estructura puede utilizar las instrucciones SQL para definir los objetos en una base de datos y almacenar información en dichos objetos. Sin embargo, antes de comenzar a escribir instrucciones SQL debe tener un entendimiento básico de los fundamentos sobre los cuales el entorno SQL se construye para que pueda aplicar esta información en todo el resto del libro. De hecho, puede resultar útil hacer referencia a este capítulo a menudo como ayuda para adquirir una comprensión conceptual del entorno SQL y cómo se relaciona con los elementos de SQL acerca de los cuales aprenderá en los siguientes capítulos.

Entienda el entorno SQL

El *entorno SQL* es, simplemente, la suma de todas las partes que conforman ese entorno. Cada parte, o componente, trabaja en conjunto con otros componentes para respaldar las operaciones de SQL tales como la creación y modificación de objetos, almacenamiento y consulta de información, o modificación y eliminación de datos. En conjunto, estos componentes forman un modelo en el que un RDBMS puede basarse. Esto no significa, sin embargo, que los proveedores de RDBMS se adhieren estrictamente a este modelo; cuáles componentes implementan, y cómo lo hacen se deja, en su mayor parte, a la discreción de esos proveedores. Aun así, deseo darle una visión general de la forma en que el entorno SQL se define, en términos de los distintos componentes, tal como se describen en el estándar SQL:2006.

El entorno SQL se compone de seis tipos de componentes, como se muestra en la figura 2-1. Los clientes SQL y los servidores SQL son parte de la aplicación de SQL y son, por lo tanto, subtipos de ese componente.

Observe que sólo hay un agente SQL y una aplicación de SQL, pero hay varios componentes de otros tipos, tales como catálogos y sitios. De acuerdo con SQL:2006, debe haber exactamente un agente SQL y una aplicación de SQL y cero o más módulos de cliente SQL, identificador de autorización y catálogos. El estándar no especifica cuántos sitios respalda, pero implica varios sitios.

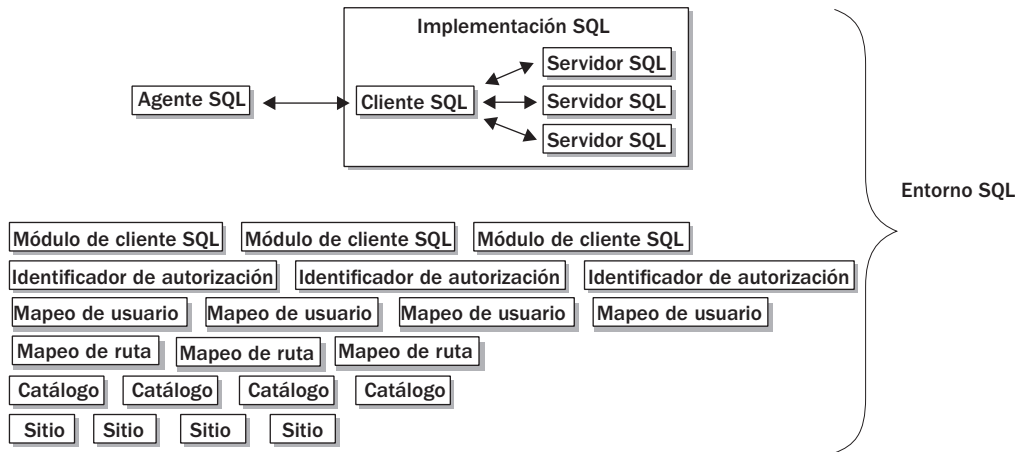


Figura 2-1 Componentes del entorno SQL.

Cada tipo de componente realiza una función específica dentro del entorno SQL. La tabla 2-1 describe los ocho tipos.

En su mayor parte, es necesario que tenga sólo una comprensión básica de los componentes que forman el entorno SQL (en términos de principios de programación de SQL). Sin embargo, uno de estos componentes (el catálogo) desempeña un papel más crítico que los otros; respecto a esto es lo que aprenderá en este libro. Por lo tanto, se cubrirá este tema con más detalle y se explicará cómo se relaciona con la gestión de datos y los objetos que sostienen dichos datos.

Tipo de componente	Descripción
Agente SQL	Cualquier estructura que haga que las instrucciones SQL se ejecuten. El agente SQL está ligado al cliente SQL dentro de la implementación de SQL.
Aplicación de SQL	Un procesador que ejecuta las instrucciones SQL en función de las necesidades del agente SQL. La aplicación de SQL incluye un cliente SQL y uno o más servidores SQL. El cliente SQL establece conexiones SQL con los servidores SQL y mantiene los datos relativos a las interacciones con el agente SQL y los servidores SQL. Un servidor SQL gestiona la sesión SQL que tiene lugar durante la conexión SQL y ejecutará instrucciones SQL recibidas del cliente SQL.

Tabla 2-1 Tipos de componentes respaldados en un entorno SQL.

Tipo de componente	Descripción
Módulo de cliente SQL	Una colección de instrucciones SQL que se escriben por separado de su lenguaje de programación de aplicación pero que pueden ser llamados dentro de ese lenguaje. Un módulo de cliente SQL contiene cero o más procedimientos invocados externamente; cada procedimiento consiste en una única instrucción SQL. Un módulo de cliente SQL reside dentro del entorno SQL y se procesa por la aplicación SQL, a diferencia de SQL incrustado, el cual se escribe dentro de la aplicación de lenguaje de programación y precompila antes de que el lenguaje de programación se elabore. Los módulos de cliente SQL se analizarán con mayor detalle en el capítulo 17.
Identificador de autorización	Un identificador representa a un usuario o un rol al que se le concede privilegios de acceso a objetos o información dentro del entorno SQL. Un usuario es una cuenta individual de seguridad que puede representar a un individuo, una aplicación o un servicio del sistema. Un <i>rol</i> es un conjunto de privilegios predefinidos que se asignan a un usuario o a otro rol. Se analizarán los identificadores autorizados, usuarios y roles en el capítulo 6.
Mapeo de usuario	Un mapeo de usuario equipara a un identificador de autorización con un descriptor de servidor exterior.
Mapeo de rutas	Un mapeo de rutas equipara a una ruta invocada en SQL con un descriptor de servidor exterior.
Catálogo	Un grupo de esquemas reunidos en un nombre definido. Cada catálogo contiene un esquema de información, que incluye los descriptores de una serie de objetos de esquema. El propio catálogo proporciona una estructura jerárquica para organizar los datos dentro de los esquemas. (Un <i>esquema</i> es básicamente un contenedor de objetos como tablas, vistas y dominios, los cuales se analizarán con mayor detalle en la siguiente sección, “Entienda los catálogos SQL.”)
Sitio	Un grupo de tablas base que contienen datos SQL, tal como lo describe el contenido de los esquemas. Se puede pensar en estos datos como “la base de datos”, pero tenga en cuenta que el estándar SQL no incluye una definición del término “base de datos”, ya que tiene muchos significados diferentes.

Tabla 2-1 Tipos de componentes respaldados en un entorno SQL (*continuación*).

Entienda los catálogos SQL

En la sección anterior, “Entienda el entorno SQL”, se expuso que un entorno SQL es la suma de todas las partes que lo componen. Se puede utilizar esa misma lógica para describir un *catálogo*, en el sentido de que un catálogo es una colección de esquemas, y esos esquemas, en conjunto, definen un nombre dentro del entorno SQL.

NOTA

Un espacio de nombre es una estructura de nombres que identifica los componentes relacionados en un determinado entorno. Un nombre es a menudo representado por una configuración de un árbol invertido para representar la relación jerárquica de objetos. Por ejemplo, suponga que su nombre incluye dos objetos: OBJETO_1 y OBJETO_2. Si el nombre es NOMBRE_1, los nombres completos de los objetos son NOMBRE_1.OBJETO_1 y NOMBRE_1.OBJETO_2 (o algunos como nomenclatura de configuración), lo que indica que comparten el mismo nombre.

Otra manera de ver a un catálogo es como una estructura jerárquica con el catálogo como el objeto primario y los esquemas como los objetos secundarios, como se muestra en la figura 2-2. En la parte superior de la jerarquía está el entorno SQL, que puede contener cero o más catálogos (aunque un entorno con cero catálogos no es bueno, ya que en el catálogo es donde se encuentran las definiciones de la información y los datos SQL). Los esquemas se localizan en el tercer nivel, debajo del catálogo, y los objetos de esquema están en el cuarto nivel.

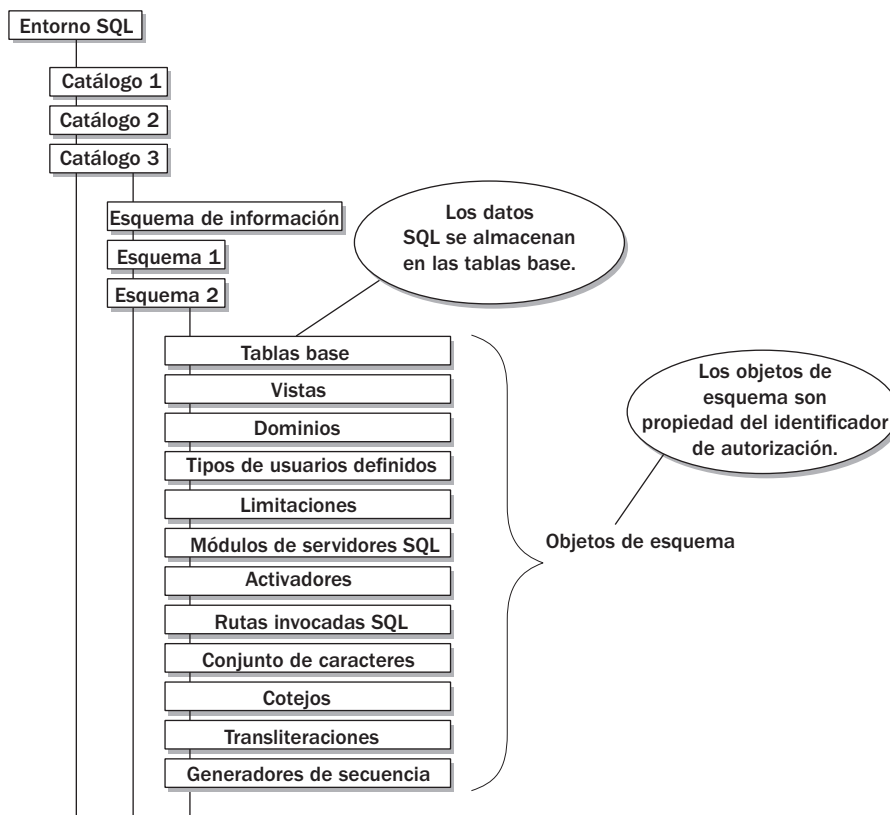


Figura 2-2 Los componentes de un catálogo.

Puede comparar las relaciones entre los objetos en un catálogo a las relaciones entre los archivos y directorios en el sistema operativo de su equipo. El catálogo se representa por un directorio raíz; los esquemas, por subdirectorios, y los objetos de esquema, por archivos dentro de subdirectorios.

Al igual que la estructura jerárquica de un sistema de archivo, la estructura de un catálogo es *lógica* por naturaleza; esto es, un sistema de archivos se presenta en una forma jerárquica (al igual que la de Windows Explorer), pero eso no significa que los archivos son almacenados jerárquicamente en el disco duro. En el mismo sentido, el catálogo de jerarquía no es más que una representación de la relación entre objetos en el entorno SQL, por lo que no implica ninguna contención física o una organización. Cómo se implementan estos objetos realmente con relación a la estructura del catálogo, y cuáles se implementan, se deja a la discreción de los proveedores de RDBMS. De hecho, el estándar SQL:2006 no define un lenguaje para la creación o eliminación de catálogos; esto también se deja a los proveedores, e incluso pocos sistemas respaldan a los catálogos.

Esquemas

Cada catálogo contiene uno o más esquemas. Un *esquema* es un conjunto de objetos relacionados que se reúnen bajo un nombre común. El esquema actúa como un contenedor de esos objetos, los que a su vez almacenan los datos SQL o realizan otras funciones con datos relacionados. Cada esquema, los objetos contenidos en el esquema y los datos SQL dentro de esos objetos son propiedad del identificador de autorización asociado con ese esquema.

A diferencia de los catálogos, los esquemas son ampliamente implementados en los productos RDBMS. Sin embargo, como con los catálogos, SQL deja la mayor parte de los detalles de implementación a los proveedores, aun cuando el estándar proporciona un lenguaje para la creación y eliminación de esquemas. Para crear un esquema se usa la instrucción `CREATE SCHEMA`, y para eliminar un esquema se usa la instrucción `DROP SCHEMA`. La creación y eliminación de esquemas se analiza con mayor detalle en la sección “Creación de un esquema”.

El tratamiento de los esquemas en un RDBMS puede variar ampliamente del estándar, y por lo tanto es importante que lea cuidadosamente la documentación del producto si desea crear un esquema en el entorno SQL. Por ejemplo, la arquitectura de la base de datos en Oracle combina el concepto de un esquema y la propiedad del identificador de autorización (cuando se crea a un usuario en Oracle, también está implícita la creación de un esquema para ese usuario). Mientras que Oracle 11g contiene la instrucción `CREATE SCHEMA` de compatibilidad con el estándar SQL, simplemente permite ejecutar un lote específico de instrucciones SQL para crear tablas y vistas y la concesión de privilegios dentro de un esquema que ya existe (es decir, uno que ya fue implícitamente creado usando la instrucción `CREATE USER`).

Esquema de información

Cada catálogo contiene un esquema especial llamado `INFORMATION_SCHEMA`. Este esquema contiene las definiciones de una serie de objetos de esquema, la mayoría de vista. Una *vista* es una tabla virtual que permite observar los datos reunidos de tablas reales. Mediante el uso de esas vistas, puede mostrar las definiciones de los objetos en ese catálogo como si se tratara de datos SQL. No puede cambiar ninguno de los datos (si lo hiciera cambiaría las definiciones objeto), pero puede mostrar la información simplemente consultando la vista correspondiente.

Como en la mayoría de características de SQL, la aplicación del esquema de información y qué funciones respalda varía de un producto a otro, aunque estas implementaciones son generalmente bastante sencillas. Por ejemplo, SQL Server 2008 incluye una vista del esquema de información llamado `INFORMATION_SCHEMA.COLUMNS`. Si consulta esta vista, los resultados incluirán una lista que contiene información acerca de cada columna accesible para el usuario actual dentro de la actual base de datos. Los resultados incluyen tanto la información como el nombre de la columna, el tipo de datos asignados a esa columna y el propietario (identificador de autorización) a quien pertenece esa columna.

Objetos de esquema

En la parte inferior del nivel del catálogo jerárquico se ubican los objetos de esquema. Los *objetos de esquema* son un conjunto de componentes relacionados que están contenidos dentro de un esquema. Éste es el nivel en el que se almacenan los datos SQL y, en consecuencia, el nivel que más preocupa a los programadores de SQL. Mediante el uso de SQL, será capaz de definir los objetos de SQL, modificar esas definiciones, y almacenar y manipular los datos de SQL dentro de los objetos. De hecho, mucho de lo que se hará en este libro de aquí en adelante tiene un impacto directo o está conectado directamente con los objetos de esquema.

El estándar SQL:2006 define 12 tipos de objetos de esquema. Estos objetos, descritos en la tabla 2-2, proporcionan las bases para el entorno SQL y la estructura de la forma en que se almacenan los datos dentro de ese entorno. Se discutirá sobre estos objetos con mayor detalle más adelante en el libro; como resultado de ello, se incluyen referencias, en su caso, a las disposiciones aplicables a los capítulos.

Esquema de objeto	Descripción
Tabla base	La unidad básica de gestión de datos en el entorno SQL. Una tabla se compone de columnas y filas y es análoga a una relación (con sus atributos y tuplas) en una teoría relacional. Cada columna se asocia con un tipo de datos y mantiene los valores que están de algún modo relacionados entre sí. Por ejemplo, una tabla de clientes contiene columnas que tienen información acerca de esos clientes, como sus nombres y direcciones. (Ver el capítulo 3.)
Vista	Una tabla virtual que se crea cuando se invoca la vista (al llamar su nombre). La tabla no existe realmente (sólo la instrucción SQL que define la tabla se almacena en la base de datos). Cuando se invoca esa instrucción, la vista toma los datos de las tablas base y muestra los resultados como si los viera de una tabla base de consulta. (Ver el capítulo 5.)
Dominio	Un objeto definido por un usuario que puede especificarse en lugar de un tipo de datos en el momento de especificar una columna (una parte del proceso de crear o alterar una definición de tabla). (Ver el capítulo 4.)

Tabla 2-2 Tipos de objetos que se definen en cada esquema.

Esquema de objeto	Descripción
Tipo definido por el usuario (UDT)	Un objeto definido por el usuario se puede especificar en lugar de un tipo de datos cuando se define una columna. SQL respalda dos tipos de UDT: distinto y estructurado. Los tipos <i>distintos</i> se basan en los tipos de datos en SQL y sus valores definidos. Los tipos <i>estructurados</i> se componen de valores de atributos, cada uno de los cuales se basan en el tipo de datos de SQL. (Ver el capítulo 3.)
Restricción	Una restricción definida en una tabla, columna o dominio que limita el tipo de datos que se inserta en el objeto de aplicación. Por ejemplo, se puede crear una restricción en una columna que restrinja los valores que se insertan en esa columna a un rango específico o una lista de números. (Ver el capítulo 4.)
Módulo de servidor SQL	Un módulo que contiene rutinas invocadas en SQL. Un <i>módulo</i> es un objeto que contiene instrucciones SQL, rutas y procedimientos. Una ruta invocada en SQL es una función o procedimiento que se invoca desde SQL. Ambas funciones y procedimientos son tipos de instrucciones de SQL que pueden manejar parámetros (valores pasados a una instrucción cuando se invoca esa instrucción). Una <i>función</i> puede recibir parámetros de entrada y devolver un valor basado en la expresión incluida en la instrucción de función. Un <i>procedimiento</i> puede recibir y devolver parámetros de entrada y salida. (Ver el capítulo 13.)
Activadores	Un objeto asociado con una tabla base que define una acción que debe tomarse cuando un evento se produce en relación con esa tabla. La acción que causa la activación del activador (ejecución) puede ser un insertar en, eliminar de, o la actualización de una tabla base. Por ejemplo, la eliminación de una fila en una tabla puede causar la activación de un activador que luego borra los datos de otra tabla. (Ver el capítulo 14.)
Rutina invocada por SQL	Una función o procedimiento que se invoca desde SQL. Una ruta invocada en SQL puede estar en un esquema de objeto o incrustado en un módulo, que también es un objeto de esquema. (Ver el capítulo 13.)
Conjunto de caracteres	Una colección de atributos de caracteres que definen cómo se representan. Un conjunto de caracteres tiene tres atributos: el repertorio, la forma de uso y el cotejo predeterminado. El <i>repertorio</i> determina qué caracteres se expresan (por ejemplo, A, B, C, y así sucesivamente). La <i>forma de uso</i> determina cómo se representan los caracteres como cadenas de hardware y software (por ejemplo, A viene antes que B, B viene antes que C). El <i>cotejo predeterminado</i> determina cómo esas cadenas se comparan unas con otras.
Cotejo	Un conjunto de reglas que controlan cómo las cadenas de caracteres se comparan unas con otras dentro de un repertorio en particular. Esta información se usa para ordenar los caracteres (por ejemplo, A viene antes que B, B viene antes que C). Un cotejo predeterminado se define por cada conjunto de caracteres.
Transliteración	Una operación que mapea caracteres de un conjunto de caracteres a otro conjunto de caracteres. Las transliteraciones incluyen operaciones tales como la traducción de caracteres de mayúsculas a minúsculas o de un alfabeto en otro.
Generador de secuencia	Un mecanismo para generar valores sucesivos de datos numéricos (enteros), uno a la vez. El generador de secuencia mantiene un <i>valor de base actual</i> , que se usa como la base para generar el siguiente valor secuencial.

Tabla 2-2 Tipos de objetos que se definen en cada esquema (*continuación*).

Como se comentó, se analizarán la mayoría de los temas de la tabla con mayor detalle más adelante en el libro. Sin embargo, los últimos tres temas, los cuales están relacionados con los conjuntos de caracteres, se tratan brevemente. Los conjuntos de caracteres, cotejos y transliteraciones respaldados por los RDBMS pueden variar de un producto a otro, y también la implementación de esas características. A lo largo de este libro, todos los ejemplos y proyectos que se darán cuentan con cualquier conjunto de caracteres predeterminados para el producto que utiliza. Si se quiere cambiar ese conjunto de caracteres, ya sea en un nivel predeterminado o en la base de datos o en la tabla nivel, primero debe revisar cuidadosamente la documentación del producto para averiguar qué se respalda y cómo se implementan esas características.

Pregunta al experto

- P:** Se describe un dominio como un objeto definido por el usuario que se basa en un tipo de datos pero puede incluir un valor predeterminado y una restricción. ¿Cómo difiere este tipo de dominio de un dominio como el que se describió en el modelo relacional?
- R:** En muchos casos los dos son lo mismo, y para todos los efectos prácticos se puede pensar en un dominio de SQL como contraparte a un dominio en el modelo relacional. Hay una sutil diferencia, no obstante [un dominio en el modelo relacional es meramente una descripción de los datos que se incluyen en un atributo (columna) asociado con ese dominio en particular]. Por otro lado, un dominio de SQL restringe los datos que se pueden insertar en una columna. Un dominio de SQL hace esto a través del uso de restricciones, las cuales son reglas de validación que son parte del sistema de integridad de los datos. La idea principal a tener en cuenta es que un dominio en el modelo relacional es un concepto lógico, mientras que un dominio de SQL es un concepto físico.
- P:** Cuando se habló acerca de objetos de esquema, se mencionaron las tablas base. ¿Respalda SQL cualquier otro tipo de tablas?
- R:** El estándar SQL:2006 respalda cuatro tipos de tablas: tablas base, tablas transitorias, tablas derivadas y tablas vistas. La tabla base es un tipo de tabla cuyos datos se almacenan en alguna parte. En otras palabras, los datos de SQL se almacenan en una tabla base. Una tabla transitoria es una tabla nombrada que implícitamente se crea durante la evaluación de una expresión de consulta o la ejecución de un activador. Una tabla derivada es aquella que contiene los resultados de una consulta (el conjunto de datos especificados en la consulta). Una tabla vista es otro nombre que se le da a una vista, la cual es una tabla virtual cuya definición se almacena, pero cuyos datos se derivan de tablas base en el momento en que se llama la vista.

¿Qué es una base de datos?

Como habrá notado, en ninguna parte de la estructura del entorno de SQL o de un catálogo se hace mención de una base de datos. La razón es que en ninguna parte del estándar SQL se define el término “base de datos”. De hecho, la única mención de una base de datos, en términos de cómo

podría encajar en la estructura del entorno de SQL, es que pueda considerar a los sitios como la base de datos, aunque esto se ofrece como una sugerencia más que como una definición absoluta. Si bien el estándar utiliza la palabra para referirse a SQL como el lenguaje de una base de datos, en realidad nunca se define el término.

Este enfoque puede estar bien para el estándar, pero en el mundo real puede ser difícil para un RDBMS crear un entorno SQL sin crear algún tipo de componentes que los usuarios puedan señalar y decir: “Sí, es la base de datos.” Y, de hecho, la mayoría de los productos permiten crear, alterar y eliminar objetos que se llaman bases de datos. En SQL Server, por ejemplo, una instancia del software de DBMS puede gestionar cualquier número de bases de datos, siendo cada base de datos una colección lógica de base de datos objetos que el diseñador escoge para administrar juntos. Sybase, MySQL y DB2 de IBM tienen arquitecturas similares. SQL Server proporciona una consola de administración llamada Microsoft SQL Server Management Studio para ver y manipular la base de datos objetos. El panel Object Explorer a lo largo del margen izquierdo proporciona una estructura de directorio jerárquica que incluye una base de datos nodo, con cada base de datos mostrada debajo, y los objetos tales como tablas figuran abajo de cada base de datos. La figura 2-3 muestra el SQL Server Management Studio con la base de datos INVENTARIO expandida hacia abajo de las columnas de la tabla CDS_ARTISTA.

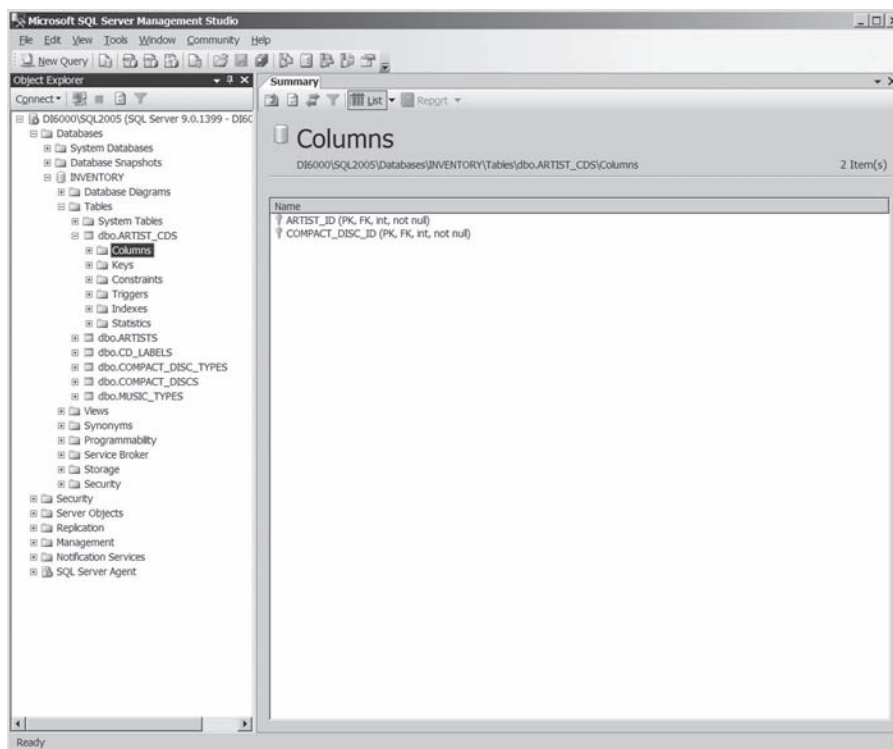


Figura 2-3 Microsoft SQL Server Management Studio con la base de datos INVENTARIO expandida.

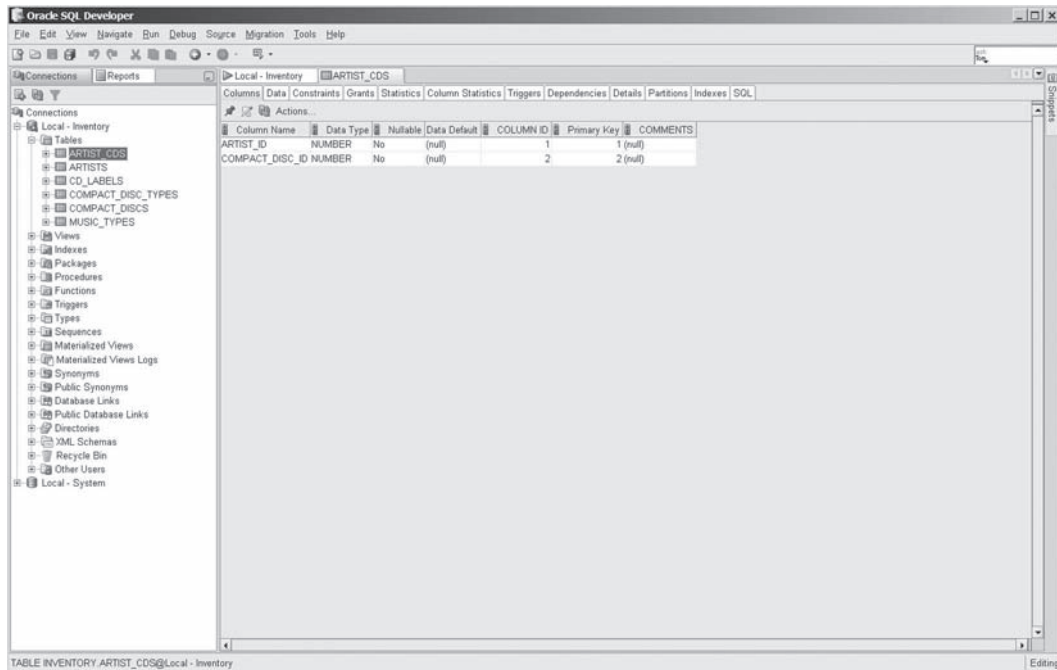


Figura 2-4 Oracle SQL Developer con el esquema INVENTARIO expandido.

El DBMS Oracle tiene una arquitectura diferente. Cada instancia del software de DBMS administra sólo una base de datos. Sin embargo, el usuario de cada base de datos obtiene un esquema distinto para almacenar una base de datos objetos que pertenece a ese usuario. Por lo tanto, en Oracle un esquema es muy similar a lo que SQL Server llama una base de datos. Oracle tiene una herramienta llamada SQL Developer que es funcionalmente similar al SQL Server Management Studio. La figura 2-4 muestra SQL Developer con el esquema INVENTARIO expandido hacia abajo de las columnas en la tabla CDS_ARTISTA. Oracle fue el primer RDBMS comercialmente disponible, y como fue creado mucho antes de que existiera un estándar SQL, no debe ser sorpresa que sea arquitectónicamente diferente.

En el capítulo 1 se expuso que una base de datos es una colección de datos organizados en un formato estructurado que se define por los metadatos que describen esa estructura. En ambos, SQL Server y Oracle, se puede observar cómo se aplica esta definición. Los dos sistemas (y cualquier RDBMS con el que trabaje) reúnen los datos en un formato estructurado y definen esos datos a través del uso de esquemas, los cuales contienen los metadatos. Esta definición también se puede aplicar al estándar SQL y a la construcción de su entorno y a los catálogos. Los datos de SQL se almacenan en un formato organizado dentro de tablas base. Estas tablas base figuran dentro de un esquema, que define esas tablas, y de este modo define los datos. Así que aunque el estándar SQL:2006 no define en realidad el término “base de datos”, no obstante apoya su concepto, al igual que los productos RDBMS que aplican SQL.

Nombrado de objetos en un entorno SQL

Hasta este punto en el libro se ha proporcionado gran cantidad de información conceptual y antecedentes. La razón de esto es que quiero que haya un fundamento básico en SQL antes de empezar a escribir instrucciones SQL. Creo que con esta información estará en mejores condiciones de comprender la lógica detrás del código SQL que creará y la razón para crearlo, y no cabe duda de que está más que listo para comenzar a escribir esas instrucciones.

Sin embargo, antes de empezar a entrar de lleno a SQL, hay un tema más que se necesita cubrir brevemente: los identificadores de objeto. Un *identificador* es un nombre dado a un objeto de SQL. El nombre puede ser de hasta (pero no incluir) 128 caracteres, y debe seguir los convenios definidos. Un identificador se puede asignar a cualquier objeto que se crea con instrucciones SQL, tales como dominios, tablas, columnas, vistas o esquemas. El estándar SQL:2006 define dos tipos de identificadores: identificadores regulares e identificadores delimitados.

Los identificadores regulares son bastante restrictivos y deben seguir convenios específicos:

- Los nombres no se distinguen entre mayúsculas y minúsculas. Por ejemplo, Nombres_Artista es lo mismo que NOMBRES_ARTISTA y nombres_artista.
- Sólo se permiten letras, dígitos y guiones. Por ejemplo, se pueden crear identificadores tales como Primer_Nombre, 1erNombre o PRIMER_NOMBRE. Observe que el guión bajo es el único carácter válido que se usa como separador entre palabras. Los espacios no son aceptables ni tampoco guiones (los guiones se interpretan como operaciones de sustracción).
- No se puede utilizar palabras clave reservadas en SQL.

NOTA

Una palabra clave es una palabra que es parte del léxico de SQL. Hay dos tipos de palabras clave en SQL: reservadas y no reservadas. Como el nombre sugiere, las palabras clave reservadas no se pueden usar para cualquier propósito, ya que están destinadas a utilizarse dentro de una instrucción SQL. Las palabras no reservadas no tienen dichas restricciones. Para un listado completo de palabras clave de SQL, vea el apéndice B.

SQL no distingue mayúsculas y minúsculas, por lo que respecta a los identificadores regulares. Todos los nombres se cambian a mayúsculas cuando se almacenan en SQL, que es la razón por la que 1erNombre y 1ERNOMBRE se leen como valores idénticos. Como ya se mencionó, el caso de falta de distinción a utilizar mayúsculas y minúsculas es el comportamiento predefinido en la mayoría de los RDBMS, y mientras que la predefinición se puede modificar en algunos productos, se recomienda que no se cambie, ya que no sería coherente con el estándar SQL y daría lugar a problemas de compatibilidad en caso de que utilice otros productos para acceder a sus datos.

Los identificadores delimitados no son tan restrictivos como los identificadores regulares, pero aún deben seguir convenios específicos:

- El identificador debe estar incluido en un conjunto de comillas dobles, como el identificador "NombresArtista".
- Las comillas no se almacenan en la base de datos, pero todos los demás caracteres se almacenan como aparecen en la instrucción SQL.

- Los nombres son sensibles a mayúsculas y minúsculas. Por ejemplo, “Nombres_Artista” *no es* lo mismo que “nombres_artista” o “NOMBRES_ARTISTA”, pero “NOMBRES_ARTISTA” *es* lo mismo que NOMBRES_ARTISTA y Nombres_Artista (porque los identificadores regulares se cambian a mayúsculas).
- La mayoría de los caracteres están permitidos, incluyendo espacios.
- Se pueden utilizar palabras clave reservadas a SQL.

Cuando se decide cómo nombrar los objetos SQL, hay una serie de sistemas que se pueden seguir. La primera elección que tendrá que hacer es si desea utilizar los identificadores regulares o delimitados. También decidirá otras cuestiones, tales como utilizar mayúsculas o minúsculas, y el uso del guión bajo, y si los identificadores estarán en singular o plural. Por ejemplo, se puede nombrar una tabla como TitulosDiscoCompacto, titulos_disco_compacto, TITULOS_DISCO_COMPACTO, “Titulos Disco Compacto”, o alguna otra forma de ese nombre. La parte importante para recordar es que debe elegir un nombre convencional y adherirse a él a lo largo de la codificación para una base de datos en particular. También debe tomar en cuenta el RDBMS que utilizará. Versiones anteriores del estándar SQL sólo permitían letras mayúsculas en los nombres de objetos (como consecuencia de los sistemas centrales que usaban una colección de caracteres llamados EBCDIC, que en los primeros días sólo contenían letras mayúsculas). Como resultado, muchos de los productos RDBMS más antiguos, incluyendo Oracle y DB2, automáticamente convertían los nombres de minúsculas a mayúsculas. Por lo tanto, se podía crear una tabla llamada TitulosDiscoCompacto, pero aparecía en el catálogo como TITULOSDISCOCOMPACTO, que obviamente no es un nombre fácil de usar. Por cierto, MySQL sigue la sensibilidad a mayúsculas y minúsculas del sistema operativo, así que en Windows no es sensible a mayúsculas y minúsculas, pero Linux y Unix son sensibles a mayúsculas y minúsculas. Y finalmente, tenga en cuenta la extensión máxima del identificador en el producto RDBMS que utilice. Mientras que SQL Server permite 128 caracteres para los nombres, Oracle admite hasta 30 caracteres (8 para los nombres de base de datos), y muchos otros tienen una extensión máxima que es inferior a 128.

NOTA

Para los ejemplos y proyectos en este libro, se usan identificadores regulares con letras mayúsculas y los guiones bajos se utilizan para separar las palabras (por ejemplo, TITULOS_DISCO_COMPACTO). Se hace esto porque dichos identificadores son compatibles con (y forman nombres de objeto fáciles de utilizar) todos los productos RDBMS. Sin embargo, reconozco que a los usuarios de SQL Server les agrada en particular utilizar identificadores con mayúsculas y minúsculas.

Nombres calificados

Todos los identificadores de esquema de objeto se califican por la forma lógica en la que encajan en la estructura jerárquica del entorno SQL. Un nombre completo calificado incluye el nombre del catálogo, el nombre del esquema y el nombre del objeto de esquema, cada uno separado por un punto. Por ejemplo, suponga que tiene una tabla llamada CD_ARTISTAS. La tabla está en el esquema DISCOS_COMPACTOS, que está en el catálogo MUSICA. El nombre completo calificado para esa tabla sería MUSICA.DISCOS_COMPACTOS.CD_ARTISTAS.

La forma en que se desempeñan esos convenios de nomenclatura en diferentes productos RDBMS depende de cómo ese producto aplica la estructura del entorno SQL. Por ejemplo, un

nombre completo calificado en SQL Server se basa en el nombre del servidor, nombre de la base de datos, nombre del propietario y nombre del objeto. En este caso, una tabla llamada ARTISTAS puede tener el nombre completo calificado de SERVIDOR01.BD_MUSICA.PBD.ARTISTAS, donde SERVIDOR01 es el nombre del servidor, BD_MUSICA es el nombre de la base de datos, y PBD (que se refiere al propietario de la base de datos) es el nombre del propietario del objeto. Para determinar cómo se manejan los nombres completos calificados para un RDBMS en particular, verifique la documentación del producto.

Creación de un esquema

Ahora que tiene una comprensión fundamental de cómo se utilizan los identificadores para nombrar los objetos SQL, está listo para empezar a escribir instrucciones SQL. Se empezará con la instrucción CREATE SCHEMA, ya que los esquemas están en la parte superior de la jerarquía de SQL, en términos de los objetos que el estándar SQL:2006 le permite crear. (Recuerde, el estándar SQL no proporciona ninguna clase de instrucción como CREATE CATALOG o CREATE DATABASE. Se deja a los proveedores de RDBMS determinar cómo y si se aplica a estos objetos.) Y como ya se mencionó, Oracle crea automáticamente un esquema para cada usuario; así, mientras tiene una instrucción CREATE SCHEMA, es sólo por compatibilidad con el estándar SQL. En la siguiente sección, “Creación de una base de datos”, saldré del modo SQL y analizaré la creación de base de datos, ya que la mayoría de los productos RDBMS respaldan la creación de objetos de base de datos, y es probable que encuentre que desea crear una base de datos con el fin de probar los ejemplos y proyectos en este libro.

El lugar para empezar con cualquier tipo de instrucción SQL es la sintaxis que define las instrucciones. La siguiente sintaxis muestra los componentes básicos de la instrucción CREATE SCHEMA:

```
CREATE SCHEMA <nombre de la cláusula>  
[ <conjunto de caracteres o ruta> ]  
[ <elementos del esquema> ]
```

NOTA

Los corchetes angulados contienen información que sirve como un marcador de posición para un valor o una cláusula en relación con esa información. Por ejemplo, <nombre de la cláusula> es el marcador de posición para palabras clave y valores relacionados con el nombramiento del esquema. Por otro lado, las llaves significan que la cláusula es opcional. No es necesario especificar un conjunto de caracteres, ruta o el elemento del esquema.

Echemos un vistazo a la sintaxis de la instrucción CREATE SCHEMA pieza por pieza. Las palabras clave CREATE SCHEMA activan la aplicación de SQL con el tipo de instrucción que se ejecuta. Esto continúa con el marcador de posición <nombre de la cláusula>, el cual puede incluir un nombre para el esquema, un identificador de autorización (precedido por la palabra clave AUTHORIZATION), o ambos. Como resultado, el nombre de la cláusula puede tener cualquiera de las siguientes formas:

- <nombre del esquema>
- AUTHORIZATION <identificador de autorización>
- <nombre del esquema> AUTHORIZATION <identificador de autorización>

El valor <identificador de autorización> especifica quién es el propietario del esquema y sus objetos. Si ninguno se especifica, el valor predetermina el del usuario actual. Si no se especifica el valor <nombre del esquema>, se crea un nombre que se base en el identificador de autorización.

La siguiente cláusula, <conjunto de caracteres o ruta>, le permite definir un conjunto de caracteres predeterminados, una ruta predeterminada, o ambos. El nombre del conjunto de caracteres se precede por las palabras clave DEFAULT CHARACTER SET y especifican un conjunto de caracteres predeterminados para el nuevo esquema. La ruta especifica una orden para buscar rutinas invocadas por SQL (procedimientos y funciones) que se crean como parte de la instrucción CREATE SCHEMA. (Las rutinas invocadas por SQL se analizan en el capítulo 13.)

La cláusula <elementos del esquema> se compone de varios tipos de instrucciones de SQL que se pueden incluir en la instrucción CREATE SCHEMA. En su mayor parte, esta cláusula permite crear objetos de esquema tales como tablas, vistas, dominios y activadores. La ventaja de esto es que los objetos se añaden correctamente al esquema cuando se crea, todo en un solo paso.

Ahora que ha visto la sintaxis para la instrucción CREATE SCHEMA, veamos un ejemplo. El siguiente código crea un esquema llamado INVENTARIO. La instrucción también especifica un nombre de identificador de autorización MNGR y un conjunto de caracteres llamado Latino1.

```
CREATE SCHEMA INVENTARIO AUTHORIZATION MNGR
DEFAULT CHARACTER SET Latino1
CREATE TABLE ARTISTAS
( ID_ARTISTA INTEGER, NOMBRE_ARTISTA CHARACTER (20) );
```

Observe que el código de ejemplo incluye la instrucción CREATE TABLE. Éste es uno de los elementos que se puede especificar como parte de la cláusula <elementos del esquema>. Se pueden incluir tantas instrucciones como se quiera. Esta instrucción en particular crea tablas llamadas ARTISTAS que contienen la columna de ID_ARTISTA y la columna NOMBRE_ARTISTA. (Se analizará la instrucción CREATE TABLE con mayor detalle en el capítulo 3.)

Además de definir la instrucción CREATE SCHEMA, SQL:2006 también define la instrucción DROP SCHEMA, como se muestra en la siguiente sintaxis:

```
DROP SCHEMA <nombre del esquema>
CASCADE | RESTRICT
```

La primera línea es bastante sencilla: el esquema nombrado será eliminado del sistema. La segunda línea tiene dos opciones: CASCADE y RESTRICT.

NOTA

El símbolo de barra vertical (|) se puede leer como “o”, lo que significa que se debe utilizar la opción CASCADE o la opción RESTRICT, pero no ambas.

Si se especifica la opción CASCADE, todos los objetos de esquema y los datos de SQL dentro de esos objetos se eliminan del sistema. Si se usa la opción RESTRICT, el esquema se elimina sólo si no existen objetos de esquema. Este método se utiliza como protección contra la eliminación de cualquier objeto que no se desea borrar. Es una forma de comprobar que los objetos que se eliminan son los que se desean borrar antes de que realmente se elimine el esquema.

Ahora veamos un ejemplo de la instrucción DROP SCHEMA. El siguiente código elimina el esquema INVENTARIO:

```
DROP SCHEMA INVENTARIO CASCADE;
```

Observe que se utiliza la opción CASCADE, lo que significa que todos los objetos de esquema y los datos de SQL se eliminarán.

Creación de una base de datos

A pesar del hecho de que el estándar SQL no define qué es una base de datos, y mucho menos proporciona una instrucción para crear cualquier tipo de base de datos objeto, hay una gran posibilidad de que trabaje con un RDBMS que no sólo respalde la creación de una base de datos objeto, sino que también utilice a ese objeto como base para su estructura jerárquica en la gestión de datos objetos. Por consiguiente, se puede encontrar que, con el fin de trabajar a través de los ejemplos y proyectos de este libro, deseará crear una base de datos de prueba para tener un entorno en el cual pueda crear, modificar o eliminar los datos objetos o los datos según sea necesario, sin correr el riesgo de la pérdida de las definiciones de datos o información de la base de datos actual. (Idealmente, se trabaja con un RDBMS que está limpio de instalaciones, sin ningún tipo de base de datos existentes, excepto el sistema preinstalado y base de datos de prueba.)

Si ya ha trabajado con un RDBMS, puede ser que esté familiarizado con la forma en que los objetos de base de datos están organizados dentro de un sistema. Por ejemplo, si echa un vistazo de nuevo a la figura 2-3, puede ver que SQL Server organiza las bases de datos del servidor en una estructura lógica debajo del nodo Bases de datos. Cada nodo Bases de datos (por ejemplo, INVENTARIO) contiene nodos secundarios que representan los diferentes tipos de objetos asociados con esa base de datos en particular. Como se puede ver, la base de datos INVENTARIO actualmente enumera ocho categorías de objetos: diagramas de base de datos, tablas, vistas, sinónimos, programación, agente de servicio, almacenamiento y seguridad. Y debajo de la tabla CDS_ARTISTA, las categorías son columnas, claves, restricciones, desencadenadores, índices y estadísticas. Para una definición de cómo SQL Server define cada uno de esos tipos de objetos, tiene que examinar la documentación del producto, que debe hacer para cualquier RDBMS. Compare y contraste con las categorías de objetos de Oracle mostradas en la figura 2-4.

La mayoría de los productos que respaldan objetos de base de datos también apoyan el lenguaje para crear esos objetos. Por ejemplo, Oracle, MySQL y SQL Server incluyen la instrucción CREATE DATABASE en los lenguajes basados en SQL. Sin embargo, qué parámetros se pueden definir cuando se construyen esas instrucciones, qué permisos se necesitan con el fin de ejecutar esas instrucciones y cómo un sistema implementa los objetos de base de datos varía de un producto a otro. Afortunadamente, la mayoría de los productos utilizan la misma sintaxis básica para crear una base de datos objeto:

```
CREATE DATABASE <nombre de la base de datos>  
<parámetros adicionales>
```

Antes de crear una base de datos en cualquier sistema, asegúrese de leer primero la documentación del producto, y si es apropiado, consulte con el administrador de base de datos para tener la certeza que es seguro agregar una base de datos objeto en el entorno de SQL. Una vez que se crea la base de datos, se pueden crear esquemas, tablas, vistas y otros objetos dentro de esa base de datos, y desde allí llenar las tablas con los datos necesarios.

Pruebe esto 2-1 La creación de una base de datos y un esquema

En “Pruebe esto 1-2: Conexión a una base de datos” se estableció la forma de acceder a un RDBMS. En ese proyecto se usó una aplicación de usuario que permite invocar directamente instrucciones SQL. Se utilizará esa aplicación para este proyecto (y el resto de los proyectos en el libro) para crear una base de datos y un esquema, o cualquiera de esas funciones que el sistema respalda. Una vez que se crea la base de datos, se trabajará dentro del contexto de esa base de datos para futuros ejemplos y proyectos. Si el sistema respalda la creación de esquemas pero no la creación de base de datos, debe trabajar dentro del contexto de ese esquema para los otros proyectos.

Paso a paso

1. Abra la aplicación de cliente que le permita invocar directamente las instrucciones SQL. Si es aplicable, consulte con el administrador de base de datos para asegurarse de que está entrando con las autorizaciones necesarias para crear una base de datos y un esquema. Puede que necesite permisos especiales para crear esos objetos. También verifique si hay algunos parámetros que debe incluir cuando se crea la base de datos (por ejemplo, el tamaño del archivo de registro), restricciones en el nombre que utilizará o restricciones de cualquier otro tipo. Asegúrese de comprobar la documentación del producto antes de seguir adelante.
2. Cree una base de datos llamada INVENTARIO (si el RDBMS respalda esta funcionalidad, en Oracle se crea un nombre de usuario llamado INVENTARIO, que implícitamente crea un esquema con el mismo nombre). Dependiendo del producto que se use, se ejecutará una instrucción que sea similar a la siguiente:

```
CREATE DATABASE INVENTARIO;
```

Si se requiere incluir parámetros adicionales en la instrucción, lo más probable es que se incluyan en las líneas siguientes a la cláusula CREATE DATABASE. Una vez que se ejecute la instrucción, debe recibir algún tipo de mensaje que indique que la instrucción se ejecutó con éxito.

3. Conexión a la nueva base de datos. El método para hacer esto varía de un producto a otro. En Oracle se puede conectar a la base de datos introduciendo la información de inicio de sesión en cualquiera de las muchas herramientas tales como SQL*Plus, iSQL*Plus y SQL Developer. En SQL Server, es simplemente cuestión de seleccionar la base de datos apropiada de la lista de conexión desplegable de las bases de datos en la barra de herramientas del SQL Server Management Studio, o se puede ejecutar la siguiente instrucción (MySQL utiliza la misma sintaxis):

```
USE Inventario
```

4. Cree un esquema llamado CD_INVENTARIO (si el RDBMS respalda esta funcionalidad). Cree un esquema bajo su actual identificador de autorización. No incluya ninguno de los elementos del esquema en este momento. En la mayoría de los casos, se ejecutará una instrucción que sea similar a la siguiente:

```
CREATE SCHEMA CD_INVENTARIO;
```

(continúa)

Resumen de Pruebe esto

Los ejercicios paso a paso de este tipo pueden ser complicados porque dependen de la forma en que los productos RDBMS implementan diversas características. Como resultado, debe confiar en gran medida en la documentación del producto (que debe utilizar de todos modos) y, en su caso, a los administradores de base de datos. Sin embargo, ahora que consiguió pasar por este ejercicio y se creó la base de datos necesaria y/o el esquema del entorno, debe estar listo para continuar con los ejemplos y proyectos del resto del libro. Ya que se han sentado las bases, estará listo para crear, alterar, suprimir los objetos de datos, e insertar, modificar y borrar los datos almacenados en esos objetos.

Autoexamen Capítulo 2

1. ¿Cuáles son las diferencias entre un agente SQL y una implementación de SQL?
2. ¿Qué componente del entorno SQL representa a un usuario o rol que concede privilegios específicos para acceder a los objetos y datos?
 - A Catálogo
 - B Identificador de autorización
 - C Módulo de cliente SQL
 - D Agente SQL
3. Un(a) _____ es una colección de esquemas que forman un nombre dentro del entorno SQL.
4. ¿Qué es un esquema?
5. ¿Qué instrucción se utiliza para agregar un esquema en el entorno SQL?
 - A ADD SCHEMA
 - B INSERT SCHEMA
 - C CREATE SCHEMA
6. ¿Cuál es el nombre del esquema que contiene las definiciones para los objetos de esquema en un catálogo?
7. ¿Cuáles son los 11 tipos de objetos de esquema que pueden estar contenidos en un esquema?
8. ¿Qué es una vista?

- 9.** ¿Cuáles objetos de esquema proporcionan la unidad básica de gestión de datos en el entorno SQL?
 - A** Vistas
 - B** Dominios
 - C** Tablas base
 - D** Conjunto de caracteres
- 10.** ¿Cómo define el estándar SQL:2006 a una base de datos?
- 11.** Un(a) _____ es el nombre dado a un objeto SQL.
- 12.** ¿Cómo se distingue un identificador regular de un identificador delimitado en una instrucción SQL?
- 13.** ¿Qué tipo de identificador permite que se utilicen espacios como parte del nombre de un objeto?
- 14.** El entorno SQL incluye un catálogo llamado INVENTARIO. En ese catálogo está el esquema llamado DISCOS_COMPACTOS, y en ese esquema se encuentra una tabla denominada ARTISTAS. ¿Cuál es el nombre cualificado de esa tabla?
- 15.** ¿Cuáles son las tres formas que puede tomar el componente <cláusula de nombre> de una instrucción CREATE SCHEMA?
- 16.** ¿Cuáles son las diferencias entre la opción CASCADE y la opción RESTRICT en la instrucción DROP SCHEMA?
- 17.** Dentro de la jerarquía del entorno SQL, ¿cómo está relacionado un dominio con un catálogo?
- 18.** ¿Qué tipo de identificador permite utilizar una palabra clave reservada?

Capítulo 3

Creación y modificación de tablas



Habilidades y conceptos clave

- Creación de tablas en SQL
 - Especificación de los tipos de datos en una columna
 - Creación de tipos definidos por el usuario
 - Especificación de los valores predeterminados en una columna
 - Modificación de tablas en SQL
 - Eliminación de tablas en SQL
-

En el entorno SQL, las tablas son la unidad básica de gestión de datos. La mayoría de la programación que se hace en SQL se relaciona directa o indirectamente con esas tablas. Como resultado, antes de insertar la información en la base de datos o modificar esa información, las tablas apropiadas deben haberse creado o debe crearlas. El estándar SQL:2006 proporciona tres instrucciones que permiten definir, cambiar o eliminar las definiciones de las tablas en el entorno SQL. Se puede utilizar la instrucción `CREATE TABLE` para añadir una tabla, la instrucción `ALTER TABLE` para modificar esa definición, o la instrucción `DROP TABLE` para eliminar la tabla y sus datos de la base de datos. De esas tres instrucciones, la instrucción `CREATE TABLE` tiene la sintaxis más compleja. No sólo es esto por los distintos tipos de tablas respaldadas por SQL, sino también porque la definición de la tabla puede incluir muchos elementos. Sin embargo, a pesar de esas complejidades, la creación de una tabla es un proceso bastante sencillo, una vez que se comprende la sintaxis básica.

Creación de tablas en SQL

Como puede que recuerde del capítulo 2, SQL respalda tres tipos de tablas: tablas base, tablas derivadas y tablas vistas. La mayoría de las tablas base son objetos de esquema que tienen los datos de SQL. Las tablas derivadas son los resultados que se observan cuando se solicitan (consultan) datos de una base de datos. Las tablas vistas son otro nombre para las vistas, que se analizarán en el capítulo 5. Se puede pensar en las tablas vistas como un tipo de nombre derivado de una tabla, con una definición de la vista almacenado en el esquema.

En este capítulo se trabajará con tablas base. De hecho, la mayoría de lo que estará trabajando directamente a lo largo de este libro (como también a lo largo de su carrera como programador) son tablas base; sin embargo, no todas las tablas base son lo mismo. Algunas son persistentes (permanentes) y otras son temporales. Algunas están en objetos de esquema y otras están contenidas en módulos. Todos los módulos de tablas base son también tablas temporales. SQL respalda cuatro tipos de tablas base:

- **Tablas base persistentes** Un objeto de esquema nombrado definido por la definición de una tabla en la instrucción `CREATE TABLE`. Las tablas base persistentes tienen los datos de SQL

que se almacenan en la base de datos. Éste es el tipo más común de tabla base y es a menudo a lo que se refiere la gente cuando menciona tablas base o tablas. Una tabla base persistente existe desde que la definición de tabla existe, y se puede llamar desde cualquier sesión de SQL.

- **Tablas temporales globales** Un objeto de esquema nombrado definido por una definición de tabla en la instrucción `CREATE GLOBAL TEMPORARY TABLE`. Aunque la definición de la tabla es parte del esquema, la tabla actual existe sólo cuando se hace referencia dentro del contexto de la sesión SQL en la cual se creó. Cuando la sesión termina, la tabla ya no existe. No se puede acceder a una tabla temporal global creada en una sesión desde otra sesión de SQL. Los contenidos son distintos en cada sesión de SQL.
- **Tablas temporales locales creadas** Un objeto de esquema nombrado definido por una definición de tabla en la instrucción `CREATE LOCAL TEMPORARY TABLE`. Al igual que una tabla temporal global, sólo se puede hacer referencia a una tabla temporal local creada dentro del contexto de la sesión de SQL en la cual se creó, y no se puede acceder desde otra sesión de SQL. Sin embargo, se puede acceder a una tabla global desde cualquier lugar dentro de una sesión asociada de SQL, mientras que en una tabla temporal local sólo se podrá acceder dentro del módulo asociado. Los contenidos son distintos dentro de ese módulo.
- **Tablas temporales locales declaradas** Una tabla declarada como parte de un procedimiento en un módulo. La definición de la tabla no se incluye en el esquema y no existe hasta que ese procedimiento se ejecuta. Al igual que otras tablas temporales, sólo se hace referencia a una tabla temporal local declarada dentro del contexto de la sesión SQL en la cual se creó.

NOTA

Una sesión SQL se refiere a la conexión entre un usuario y un agente SQL. Durante esta conexión, una secuencia de instrucciones SQL consecutivas se invocan por ese usuario y luego se ejecutan. Un módulo es un objeto que contiene instrucciones SQL, rutas o procedimientos. Los módulos se analizarán en el capítulo 13 y en el capítulo 17.

Como puede ver, se puede utilizar una forma de la instrucción `CREATE TABLE` para crear todos los tipos de tablas, excepto las tablas temporales locales declaradas. A lo largo del resto del capítulo, se analizarán en primer lugar las tablas temporales locales persistentes, aunque se tocará el tema de las tablas temporales en los capítulos subsecuentes. Mientras tanto, veamos la sintaxis en la instrucción `CREATE TABLE`:

```
CREATE [ {GLOBAL | LOCAL} TEMPORARY ] TABLE <nombre de la tabla>
( <elemento de la tabla> [ {, <elemento de la tabla> }... ] )
[ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

NOTA

Las llaves se usan para agrupar elementos. Por ejemplo, en la primera línea de la sintaxis, las palabras clave `GLOBAL | LOCAL` se agrupan juntas. Los corchetes indican que primero debe decidir cómo manejar los contenidos dentro de los corchetes y luego determinar cómo encajan en la cláusula. En la primera línea, debe utilizar ya sea `GLOBAL` o `LOCAL` junto con `TEMPORARY`. Sin embargo, toda la cláusula es opcional. Los tres puntos (en la segunda línea) indican que puede repetir la cláusula tantas veces como sea necesario. En este caso, puede añadir tantas cláusulas `<elemento de la tabla>` como la definición requiera.

La sintaxis que se muestra proporciona sólo los fundamentos de la instrucción CREATE TABLE, que es en realidad mucho más compleja. (La sintaxis y sus explicaciones toman alrededor de 38 páginas del estándar SQL:2006.) Aun así, la sintaxis proporcionada aquí es un fundamento suficiente para crear la mayoría de las tablas que es probable que utilice.

En la primera línea de la sintaxis se designa si la tabla es temporal y se proporciona un nombre a la tabla; por lo tanto, tiene tres opciones:

- CREATE TABLE <nombre de la tabla>
- CREATE GLOBAL TEMPORARY TABLE <nombre de la tabla>
- CREATE LOCAL TEMPORARY TABLE <nombre de la tabla>

Dependiendo del RDBMS en que trabaje, puede que tenga que calificar el nombre de la tabla incluyendo un nombre de esquema, identificador autorizado, o el nombre de la base de datos (por ejemplo, INVENTARIO.ARTISTAS).

NOTA

Hay una serie de variaciones específicas en la aplicación con respecto a las tablas temporales que vale la pena mencionar. Oracle (desde 11g) no tiene una opción LOCAL para crear una tabla temporal; los datos en una tabla temporal se privan de la sesión expresada. IBM DB2 UDB desde 9.1 utiliza el comando, DECLARE GLOBAL TEMPORARY TABLE, para la creación de una tabla temporal global; no parece haber ninguna designación para crear/declarar una tabla temporal local. En SQL Server (desde 2008), las tablas temporales se crean con el comando típico CREATE TABLE, pero los nombres de las tablas temporales locales llevan un prefijo con un signo numérico único (#nombre_tabla), y los nombres de las tablas temporales globales llevan un prefijo con un signo numérico doble (##nombre_tabla).

La segunda línea de la sintaxis permite especificar las partes que componen la tabla, tales como columnas. (Se retomará esto en un momento.) La tercera línea de la sintaxis aplica sólo si se crea una tabla temporal. La cláusula permite especificar si la tabla se debe vaciar o no cuando la instrucción COMMIT se ejecuta. La instrucción COMMIT se usa en una transacción al hacer cambios en la base de datos. Se analizarán las transacciones en el capítulo 16.

Se puede pensar de las cláusulas <elemento de la tabla> como el punto central de la instrucción CREATE TABLE. Es aquí donde se definen las columnas, limitaciones y otros elementos específicos de la tabla que se crea. Se pueden definir una o más cláusulas <elemento de la tabla>. Si se define más de una, se deben separar por comas. De los elementos que puede crear, nos centraremos principalmente en las columnas (en este capítulo) y las restricciones (en el capítulo 4). Echemos un vistazo más de cerca a la sintaxis que se utiliza para definir una columna:

```
<nombre de columna> { <tipo de datos> | <dominio> }  
[ <cláusula predeterminada> ] [ <restricción de columna> ] [ COLLATE  
<nombre de cotejo> ]
```

En la primera línea de la sintaxis debe proporcionar el nombre de la columna y declarar el tipo de datos o el dominio definido por el usuario. Se analizarán los tipos de datos en la sección “Especificación de los tipos de datos en una columna”, más adelante en este capítulo, y los dominios en el capítulo 4.

En la segunda línea de la sintaxis se tiene la opción de proporcionar un valor predeterminado (ver la sección “Especificación de los valores predeterminados en una columna”), las restricciones de la columna (ver el capítulo 4) o el cotejo (ver el capítulo 2).

En su forma más básica, la instrucción CREATE TABLE podría verse como la siguiente instrucción:

```
CREATE TABLE ARTISTAS
( ID_ARTISTA      INTEGER,
  NOMBRE_ARTISTA  CHARACTER (60) );
```

En esta instrucción se creó una tabla llamada ARTISTAS, una columna llamada ID_ARTISTA y una columna llamada NOMBRE_ARTISTA. La columna ID_ARTISTA se asocia con el tipo de dato INTEGER, y la columna NOMBREA_ARTISTA se asocia con el tipo de datos CHARACTER. Observe que las definiciones de las dos columnas se separan por una coma. También observe que se colocaron las definiciones de las dos columnas en líneas separadas y se alinearon los tipos de datos agregando espacios extra (todo esto es para mejorar la legibilidad, pero aparte de eso es innecesario) (cuando se procesan las instrucciones SQL, los espacios extra y las líneas nuevas simplemente se ignoran). Si se ejecuta la instrucción CREATE TABLE, la tabla será similar a la mostrada en la figura 3-1.

NOTA

Las filas de datos mostradas no estarán en una tabla hasta que se añadan esos datos. Las filas se muestran simplemente para darle una idea del tipo de tabla que esta instrucción podría crear.

Antes de ir más lejos con la discusión de la creación de una tabla, echemos un vistazo más de cerca a los tipos de datos, que juegan un rol integral en la definición de cualquier columna.

ID_ARTISTA: INTEGER	NOMBRE_ARTISTA: CHARACTER(60)
10001	Jennifer Warnes
10002	Joni Mitchell
10003	William Ackerman
10004	Kitaro
10005	Bing Crosby
10006	Patsy Cline
10007	Jose Carreras
10008	Placido Domingo
10009	Luciano Pavarotti

Figura 3-1 Las columnas ID_ARTISTA y NOMBRE_ARTISTA de la tabla ARTISTAS.

Pregunta al experto

P: Cuando se analizaron los diversos tipos de tablas respaldadas por SQL, se habló brevemente de las tablas temporales. ¿Cuál es el propósito de las tablas temporales?

R: Las tablas temporales proporcionan una forma de almacenar resultados temporales dentro del contexto de su sesión. Puede encontrar que necesita un lugar para almacenar los datos con el fin de tomar un cierto curso de acción. Puede crear explícitamente una tabla base persistente, almacenar datos en ella y después eliminar la tabla cuando termine, pero la tabla temporal permite hacer lo mismo sin tener que destruir explícitamente la tabla cada vez que la use. En otras palabras, una tabla temporal es una herramienta útil cuando se requiere almacenar datos por sólo un periodo específico. Por ejemplo, suponga que tiene una aplicación que permita generar un informe trimestral basado en el inventario al final del periodo que se examina. La aplicación necesitará reunir la información en una colección significativa para generar el reporte. Sin embargo, una vez que el reporte se genera, la aplicación ya no necesita almacenar esa información; por lo tanto, la tabla se puede eliminar. Una de las ventajas de utilizar una tabla temporal es que, debido a que es única en una sesión, la tabla no interactúa con otros usuarios o sesiones. Como resultado, el RDBMS no tiene que tomar medidas especiales para bloquear los datos para evitar que otros usuarios apliquen actualizaciones conflictivas a las tablas temporales, y evitar el bloqueo puede resultar en un mejor rendimiento.

Especificación de los tipos de datos en una columna

Cada vez que se define una columna en la instrucción CREATE TABLE, al menos debe proporcionar un nombre para la columna y un tipo de dato asociado o un dominio. El tipo de datos o dominio (que se analiza en el capítulo 4) limita los valores que pueden introducirse en esa columna. Por ejemplo, algunos tipos de datos limitan los valores de una columna a números, mientras otros tipos de datos permiten que se introduzca cualquier carácter. SQL respalda tres formas de tipos de datos:

- **Predefinido** Los tipos de datos predefinidos son los más comunes. Cada tipo de datos predefinido es un elemento nombrado (utilizando una palabra clave en SQL) que limita valores a las restricciones definidas por esa base de datos. SQL incluye cinco formas de tipos de datos predefinidos: cadena, numérico, fecha y hora, intervalo y booleano.
- **Construido** Los tipos de datos contruidos también se denominan elementos, pero tienden a ser más complejos que los tipos de datos predefinidos ya que pueden contener múltiples valores. Los tipos contruidos permiten *construir* estructuras más complicadas que la mayoría de los tipos de datos tradicionales. Una discusión detallada de estos tipos está fuera del alcance de este libro, pero se mencionan para que sepa que existen.

- **Definido por el usuario** Los tipos de datos definidos por el usuario se basan en los tipos predefinidos o definiciones de atributos, y se agregan como objetos de esquema al entorno SQL. SQL respalda dos formas de tipos de datos definidos por el usuario: distinto y estructurado. El tipo distinto se basa en el tipo de dato predefinido, y el tipo estructurado se basa en las definiciones de atributo. Se analizarán los tipos definidos por el usuario en la sección “Creación de tipos definidos por el usuario”, más adelante en este capítulo.

A pesar de que todas las implementaciones de SQL soportan los tipos de datos, qué tipos de datos soportan varían de un producto a otro. Sin embargo, como un programador principiante de SQL, encontrará que la mayoría de las aplicaciones respaldan la forma básica (más tradicional) de tipos de datos, que son los que se usarán en los ejemplos y ejercicios a lo largo del libro. Estos tipos de datos más tradicionales, algunas veces conocidos como tipos primitivos, son parte de los tipos de datos predefinidos de SQL, que se describen en la sección siguiente. No trate de memorizar cada uno de estos tipos, pero comience a familiarizarse con las diferencias entre ellos. Encontrará que, cuando empiece a utilizar tipos de datos específicos, se sentirá más cómodo con éstos. Mientras tanto, remítase a las secciones siguientes tanto como sea necesario siempre que trabaje con definición de tablas o datos SQL.

Tipos de datos de cadena

Los tipos de datos de cadena se componen por tipos que permiten valores basados en los conjuntos de caracteres o en bits de datos. Los valores permitidos por los tipos en cadena pueden ser de longitud fija o variable, dependiendo del tipo específico. SQL define cuatro formas de tipos de datos en cadena:

- **Cadenas de caracteres** Los valores permitidos se deben extraer de un conjunto de caracteres, ya sea de un conjunto predeterminado o de un conjunto definido en el momento que la columna se define. Los tipos de datos en la cadena de caracteres incluyen CHARACTER, CHARACTER VARYING y CHARACTER LARGE OBJECT.
- **Cadenas de carácter nacional** Los valores permitidos son similares a los de las cadenas de caracteres, salvo que el conjunto de caracteres asociados con estos tipos de datos se definen por la aplicación. Como resultado, cuando una cadena de carácter nacional se especifica, los valores asociados con ese tipo de datos deben basarse en el conjunto de caracteres especificado por el sistema de gestión de base de datos relacional (RDBMS) para las cadenas de carácter nacional. Éstas son útiles para almacenar cadenas de caracteres en varios lenguajes en la misma base de datos. Los tipos de datos en la cadena de carácter nacional incluyen NATIONAL CHARACTER, NATIONAL CHARACTER VARYING y NATIONAL CHARACTER LARGE OBJECT.
- **Cadenas de bits** Los valores permitidos se basan en bits de datos (dígitos binarios), en lugar de conjuntos de caracteres o cotejos, lo que significa que estos tipos de datos permiten sólo valores de 0 o 1. SQL respalda dos formas de tipos de datos de cadena de bits: BIT y BIT VARYING.
- **Cadenas binarias** Los valores permitidos son similares a las cadenas de bits, excepto que se basan en bytes (denominado como *octetos* en SQL:2006), en lugar de bits. Como resultado, ningún conjunto de caracteres o cotejos se relacionan con ellas. (Un byte es igual a 8 bis, la

razón por la cual el estándar SQL utiliza el término octeto.) SQL sólo respalda un tipo de datos en una cadena binaria: **BINARY LARGE OBJECT**. Este tipo es útil para almacenar datos binarios puros tales como clips de sonido o imágenes en la base de datos.

Ahora que tiene una descripción general de las formas de tipos de datos en cadena, echemos un vistazo más de cerca a cada uno. La tabla 3-1 describe cada uno de estos tipos de datos y proporciona un ejemplo de una definición de columna que utilice el tipo específico.

Tipo de dato	Descripción/ejemplo
CHARACTER	Especifica el número exacto de caracteres (que debe ser de un conjunto de caracteres) que se almacenará por cada valor. Por ejemplo, si se define el número de caracteres como 10, pero el valor contiene sólo seis caracteres, los cuatro caracteres restantes serán espacios. El tipo de dato puede abreviarse como CHAR . Ejemplo: <code>NOMBRE_ARTISTA CHAR(60)</code>
CHARACTER VARYING	Especifica el mayor número de caracteres (que debe ser de un conjunto de caracteres) que se incluyen en un valor. El número de caracteres almacenados es exactamente el mismo número que el valor introducido; por lo tanto, no se agregan espacios al valor. El tipo de dato puede abreviarse como CHAR VARYING o VARCHAR . Ejemplo: <code>NOMBRE_ARTISTA VARCHAR(60)</code>
CHARACTER LARGE OBJECT	Almacena grandes grupos de caracteres, hasta la cantidad especificada. El número de caracteres almacenados es exactamente el mismo número que el valor introducido; por lo tanto, no se agregan espacios al valor. El tipo de dato puede abreviarse como CLOB . Ejemplo: <code>BIO_ARTISTA CLOB(200K)</code>
NATIONAL CHARACTER	Funciona igual que el tipo de dato CHARACTER , excepto que se basa en una aplicación definida de un conjunto de caracteres. El tipo de dato puede abreviarse como NATIONAL CHAR y NCHAR . Ejemplo: <code>NOMBRE_ARTISTA NCHAR(60)</code>
NATIONAL CHARACTER VARYING	Funciona igual que el tipo de dato CHARACTER VARYING , excepto que se basa en una aplicación definida de un conjunto de caracteres. El tipo de dato puede abreviarse como NATIONAL CHAR VARYING o NCHAR VARYING . Ejemplo: <code>NOMBRE_ARTISTA NCHAR VARYING(60)</code>
NATIONAL CHARACTER LARGE OBJECT	Funciona igual que el tipo de dato CHARACTER LARGE OBJECT , excepto que se basa en una aplicación definida de un conjunto de caracteres. El tipo de dato puede abreviarse como NCHAR LARGE OBJECT o NCLOB . Ejemplo: <code>BIO_ARTISTA NCLOB(200K)</code>
BIT	Especifica el número exacto de bits que pueden almacenarse para cada carácter. Por ejemplo, si se define el número de bits como 2, pero el valor contiene sólo 1 bit, el bit restante será un espacio. Si el número de bits no se especifica, 1 bit se almacena. Ejemplo: <code>EN_EXISTENCIA BIT</code>

Tabla 3-1 Tipos de datos en cadena con ejemplos de definiciones de columna.

Tipo de dato	Descripción/ejemplo
BIT VARYING	Especifica el mayor número de bits que pueden incluirse en un valor. El número de bits almacenados es exactamente el mismo número que el valor introducido; por lo tanto, no se agregan espacios al valor. Ejemplo: <code>EN_EXISTENCIA BIT VARYING (2)</code>
BINARY LARGE OBJECT	Almacena grandes grupos de bytes hasta la cantidad especificada. El número de bytes almacenados es exactamente el mismo número que el valor introducido; por lo tanto, no se agregan espacios al valor. El tipo de dato también puede remitirse como BLOB. Ejemplo: <code>IMG_ARTISTA BLOB (1M)</code>
XML	El lenguaje de marcado extensible (XML) es un lenguaje de marcado para fines generales utilizado para describir documentos en un formato que es conveniente para la visualización de páginas web y para intercambiar datos entre diferentes partes. Las especificaciones para almacenar datos XML en bases de datos SQL se añaden al estándar SQL en SQL:2003 y se abordan en el capítulo 18. Ejemplo: <code>BIO_ARTISTA XML(DOCUMENT(UNTYPED))</code>

Tabla 3-1 Tipos de datos en cadena con un ejemplo de definiciones de columna (*continuación*).

Tipos de datos numéricos

Como probablemente podrá suponer por el nombre, los valores especificados por los tipos de datos numéricos son números. Todos los tipos de datos numéricos tienen una precisión, y algunos tienen una escala. La *precisión* se refiere al número de dígitos (dentro de un valor numérico específico) que se pueden almacenar. La *escala* se refiere al número de dígitos en la parte fraccional de ese valor (los dígitos a la derecha del punto decimal). Por ejemplo, el número 435.27 tiene una precisión de 5 y una escala de 2. La escala no puede ser un número negativo o ser más larga que la precisión. Una escala de 0 indica que el número es un número entero y no contiene ningún componente fraccional. SQL define dos formas de tipos de datos numéricos:

- **Numéricos exactos** Los valores permitidos tienen precisión y escala, que, para algunos tipos de datos numéricos, se definen por la implementación. Los tipos de datos numéricos exactos incluyen NUMERIC, DECIMAL, INTEGER y SMALLINT.
- **Numéricos aproximados** Los valores permitidos tienen precisión pero no escala. Como resultado, el punto decimal puede flotar. Un número de *punto flotante* es aquel que contiene un punto decimal, pero el punto decimal se puede localizar en cualquier lugar dentro del número, por lo que se dice que un numérico aproximado no tiene escala. Los tipos de datos numéricos aproximados incluyen REAL, DOUBLE PRECISION y FLOAT.

La tabla 3-2 describe cada uno de los tipos de datos numéricos y proporciona un ejemplo de una definición de columna que utiliza un tipo específico.

Tipo de dato	Descripción/ejemplo
NUMERIC	Especifica la precisión y la escala de un valor numérico. Se puede especificar sólo la precisión y utilizar la escala definida (predeterminada) por la aplicación, o se puede especificar la precisión y la escala. Si no se especifica ni la precisión ni la escala, la aplicación proporcionará los valores predeterminados para ambas. Ejemplo: <code>TASA_ARTISTAS NUMERIC (5, 2)</code>
DECIMAL	Especifica valores similares al del tipo de datos NUMERIC. Sin embargo, si la precisión definida por la aplicación es superior a la precisión especificada, los valores con la precisión superior se aceptarán, pero la escala siempre será la que se especifique. Ejemplo: <code>REGALIAS_ARTISTAS DECIMAL (5, 2)</code>
INTEGER	Especifica un valor con una precisión definida por la aplicación y una escala de 0, lo que significa que sólo se aceptan enteros y no se especifica ningún parámetro con este tipo de datos. El tipo de dato puede abreviarse como INT. Ejemplo: <code>ID_ARTISTA INT</code>
SMALLINT	Especifica un valor similar al del tipo de datos INTEGER. Sin embargo, la precisión definida por la aplicación debe ser menor que la precisión de INTEGER. Ejemplo: <code>ID_ARTISTA SMALLINT</code>
FLOAT	Especifica la precisión de un valor numérico, pero no la escala. Ejemplo: <code>REGALIAS_ARTISTAS FLOAT (6)</code>
REAL	Especifica un valor con una precisión definida por la aplicación, pero sin una escala. La precisión debe ser menor a la precisión definida por un tipo de datos DOUBLE PRECISION. Ejemplo: <code>REGALIAS_ARTISTAS REAL</code>
DOUBLE PRECISION	Especifica un valor con una precisión definida por la aplicación, pero sin una escala. La precisión debe ser superior a la precisión definida por el tipo de datos REAL. La implicación es que el valor de la precisión debe ser doble que en el tipo de datos REAL, pero cada implementación define <i>doble</i> de diferentes maneras. Ejemplo: <code>REGALIAS_ARTISTAS DOUBLE PRECISION</code>

Tabla 3-2 Tipos de datos numéricos con un ejemplo de definiciones de columna.

Tipos de datos de fecha y hora

Como su nombre lo indica, los tipos de datos de fecha y hora se refieren a las formas de seguimiento de fechas y horas. SQL define tres formas de tipos de datos de fecha y hora (DATE, TIME y TIMESTAMP) y las variaciones en estos tipos. Estas variaciones están relacionadas con el Tiempo Universal Coordinado (UTC), que solía llamarse el Tiempo Medio de Greenwich (GMT), y las distintas zonas horarias. La tabla 3-3 describe cada uno de los tipos de datos de fecha y hora en el estándar SQL:2006 y proporciona un ejemplo de una definición de columna que utiliza un tipo específico.

Tipo de dato	Descripción/ejemplo
DATE	Especifica el año, mes y día del valor de una fecha. El año tiene cuatro dígitos y respalda valores desde 0001 hasta 9999; el mes tiene dos dígitos y respalda valores desde 01 hasta 12, y el día tiene dos dígitos y respalda valores desde 01 hasta 31. Ejemplo: <code>FECHA_CONTRATACION DATE</code>
TIME	Especifica la hora, minuto y segundo del valor de una hora. La hora tiene dos dígitos y respalda valores desde 00 hasta 23; el minuto tiene dos dígitos y respalda valores desde 00 hasta 59, y el segundo tiene, al menos, dos dígitos y respalda valores desde 00 hasta 61.999 (para dar cabida al salto de segundos). El tipo de datos no incluye dígitos fraccionados a menos que se especifiquen. Por ejemplo, <code>TIME(3)</code> daría tres dígitos fraccionales. El tipo de dato también se puede denominar como <code>TIME WITHOUT TIME ZONE</code> . Ejemplo: <code>HORA_EVENTO TIME(2)</code>
TIME WITH TIME ZONE	Especifica la misma información que el tipo de datos <code>TIME</code> salvo que el valor también incluye información específica del UTC y las zonas horarias. Los valores agregados al tipo de datos tienen un rango desde -11:59 a +12:00. Ejemplo: <code>HORA_EVENTO TIME WITH TIME ZONE(2)</code>
TIMESTAMP	Combina los valores de <code>TIME</code> y <code>DATE</code> . La única diferencia es que con el tipo de datos <code>TIME</code> , el número predeterminado de dígitos fraccionarios es 0, pero con el tipo de datos <code>TIMESTAMP</code> , el número predeterminado es 6. Se puede especificar un número diferente de dígitos fraccionarios incluyendo un parámetro, tal como <code>TIMESTAMP(4)</code> . El tipo de datos se puede también denominar como <code>TIMESTAMP WITHOUT TIME ZONE</code> . Ejemplo: <code>FECHA_COMPRA TIMESTAMP(3)</code>
TIMESTAMP WITH TIME ZONE	Especifica la misma información que el tipo de datos <code>TIMESTAMP</code> , salvo que el valor también incluye información específica del UTC y las zonas horarias. Los valores añadidos al tipo de datos tienen un rango desde -11:59 a +12:00. Ejemplo: <code>FECHA_COMPRA TIMESTAMP WITH TIME ZONE(2)</code>

Tabla 3-3 Tipos de datos de fecha y hora con un ejemplo de definiciones de columna.

Encontrará variaciones de implementación considerables para los tipos de datos de fecha y hora, ya que en un principio las bases de datos no las tenían en absoluto. Esto parece extraño hasta que se da cuenta que se construyeron por informáticos que no sabían cuántas aplicaciones comerciales se basan en fechas y horas. Conforme los usuarios de negocios solicitaron las capacidades de fecha y hora, los proveedores comerciales de RDBMS de hoy en día (mucho antes de que hubiera un estándar SQL) se apresuraron a ofrecer las nuevas características y, por lo tanto, se implementaron en formas muy diferentes. Por ejemplo, el tipo de datos `DATE` de Oracle siempre incluye un componente de hora, y SQL Server utiliza un tipo de datos `TIMESTAMP` para un fin completamente diferente, con un tipo de datos llamado `DATETIME` que funciona como el tipo de datos `TIMESTAMP` de SQL:2006. Por lo tanto, como siempre, consulte la documentación del proveedor.

Tipo de datos de intervalo

El tipo de datos de intervalo está estrechamente relacionado con los tipos de datos de fecha y hora. El valor de un tipo de datos de intervalo representa la diferencia entre dos valores de fecha y hora. SQL respalda dos tipos de intervalos:

- **Intervalos de año-mes** El tipo de datos de intervalo especifica intervalos entre años, meses, o ambos. Se pueden utilizar sólo los campos de AÑO y MES en un intervalo de año-mes.
- **Intervalos de día-hora** El tipo de datos de intervalo especifica intervalos entre cualquiera de los siguientes valores: días, horas, minutos o segundos. Se pueden utilizar sólo los campos de DAY, HOUR, MINUTE y SECOND en un intervalo de día-tiempo.

No se puede mezclar un tipo de intervalo con los demás. Por ejemplo, no se puede definir un tipo de datos de intervalo que utilice el campo YEAR y el campo HOUR.

El tipo de datos de intervalo utiliza la palabra clave INTERVAL seguido por una cláusula de <calificador de intervalo>. La cláusula es una serie de reglas complejas que describen cómo se puede definir el tipo de datos de INTERVAL para expresar la participación de los intervalos de años, meses, días, horas, minutos o segundos. Además, el campo principal (la primera palabra) en la cláusula se puede definir con una precisión (*p*). La precisión es el número de dígitos que se utilizan en el campo principal. Si la precisión no se especifica, la precisión predeterminada es 2. Para los intervalos de año-mes, se puede especificar uno de los siguientes tipos de datos de intervalo:

- INTERVAL YEAR
- INTERVAL YEAR(*p*)
- INTERVAL MONTH
- INTERVAL MONTH(*p*)
- INTERVAL YEAR TO MONTH
- INTERVAL YEAR(*p*) TO MONTH

Hay muchas más opciones para los intervalos día-hora, ya que hay más campos de donde escoger. Por ejemplo, se pueden especificar cualesquiera de los siguientes tipos de intervalos utilizando el campo DAY como campo principal o independientemente del campo:

- INTERVAL DAY
- INTERVAL DAY(*p*)
- INTERVAL DAY TO HOUR
- INTERVAL DAY(*p*) TO HOUR
- INTERVAL DAY TO MINUTE
- INTERVAL DAY(*p*) TO MINUTE
- INTERVAL DAY TO SECOND
- INTERVAL DAY(*p*) TO SECOND
- INTERVAL DAY TO SECOND(*x*)
- INTERVAL DAY(*p*) TO SECOND(*x*)

Cuando el campo de rastreo (la última palabra) es `SECOND`, se puede especificar una precisión adicional (*x*), que define el número de dígitos después del punto decimal. Como se puede observar de estos ejemplos, hay muchos más tipos de datos de intervalos de día-hora que se pueden definir. Sin embargo, tenga en cuenta que el campo principal debe tener siempre una unidad de tiempo superior que el campo de rastreo. Por ejemplo, el campo `YEAR` es mayor que `MONTH`, y `hour` es mayor que `minute`.

Si fuera a utilizarse un tipo de datos de intervalo en una definición de columna, se vería algo como lo siguiente:

```
RANGO_FECHA INTERVAL YEAR(4) TO MONTH
```

En este ejemplo, un valor en esa columna incluiría cuatro dígitos para el año, un guión, y luego dos dígitos para el mes, tal como 1999-08. Si la precisión no se especifica para el año, el rango de año puede incluir sólo dos dígitos (00 hasta 99).

Tipo de datos booleanos

El tipo de datos booleano (a diferencia de los tipos de datos de intervalo) es muy sencillo y fácil de aplicar. El tipo de dato respalda una construcción de verdadero/falso que permite sólo tres valores: verdadero, falso o desconocido. Un valor nulo se evalúa como desconocido. (En SQL, un valor *nulo* se utiliza para indicar que un valor no está definido o no se conoce. Se analizarán los valores nulos en el capítulo 4.)

Los valores en el tipo de datos booleano se pueden utilizar en consultas de SQL y expresiones con fines de comparación. (Se analizarán las comparaciones en el capítulo 9.) Las comparaciones en el tipo de datos booleano siguen una lógica específica:

- Verdadero es mayor que falso.
- Una comparación con un valor desconocido (nulo) devolverá un resultado desconocido.
- Un valor desconocido se puede asignar a una columna sólo si admite valores nulos.

Para utilizar el tipo de datos booleano, debe utilizar la palabra clave `BOOLEAN` sin parámetros, como se muestra en el siguiente ejemplo:

```
ARTISTAS_CON_AGENTE BOOLEAN
```

La columna `ARTISTAS_CON_AGENTE` sólo aceptará valores de verdadero, falso y desconocido.

NOTA

El tipo de datos booleano se basa en un tipo específico de lógica del ordenador conocido como booleana (nombrado por el matemático del siglo XIX George Boole), que evalúa las condiciones de verdadero o falso en una operación o expresión determinada. Muchos lenguajes de programación respaldan la lógica booleana mediante el uso de operadores lógicos como `AND`, `OR` y `NOT`, por ejemplo, "`ARTÍCULO_A IS NOT FALSE`" o "`ARTÍCULO_A AND ARTÍCULO_B OR ARTÍCULO_C IS TRUE`". En SQL, la lógica booleana se aplica mediante el uso de operadores de comparación para equiparar valores dentro de varios tipos de datos. Estos operadores se analizarán en el capítulo 9.

Pregunta al experto

P: ¿Cómo se pueden comparar los tipos de datos predefinidos en SQL con los tipos de datos que se encuentran en otros lenguajes de programación?

R: En su mayor parte, es poco probable que los tipos de datos de dos diferentes lenguajes sean los mismos. Un conjunto de tipos de datos en un lenguaje pueden variar en estructura y semántica de un conjunto de tipos de datos en otro lenguaje. Estas diferencias, a menudo llamadas desajuste de impedancia, pueden conducir a la pérdida de información cuando una aplicación toma datos de una base de datos de SQL. De hecho, a menudo es una buena idea saber qué lenguaje se usará para las aplicaciones cuando se diseña la base de datos. En algunos casos, el diseño de la base de datos puede afectar el lenguaje de aplicación que más fácilmente se utiliza para manipular los datos en una base de datos de SQL. Sin embargo, SQL incluye una expresión de conversión de datos llamada CAST. La expresión CAST permite convertir datos de un tipo de datos a otro tipo de datos, permitiendo al lenguaje anfitrión acceder valores que no podrían manejarse en su forma original. La expresión CAST se analizará a mayor detalle en el capítulo 10.

P: ¿Se pueden asignar los tipos de datos de SQL a objetos que no sean columnas?

R: Cada valor de datos de SQL, o literal, pertenece a un tipo de datos. Por ejemplo, los tipos de datos se pueden asignar a los parámetros de los procedimientos invocados externamente. Los *procedimientos invocados externamente* son procedimientos que están contenidos dentro de un módulo de clientes de SQL. Un procedimiento es una instrucción SQL (o series de instrucciones) que pueden convocarse desde otro elemento en el código, que en el caso de procedimientos invocados externamente es un código externo. Un parámetro, que es el literal que pertenece a un tipo de datos, es un valor que se pasa al procedimiento y se utiliza cuando el procedimiento se procesa. El parámetro actúa como un marcador de posición para ese valor. Los módulos de clientes SQL se analizan en el capítulo 17.

Utilice tipos de datos SQL

Ahora que ha echado un vistazo a los diferentes tipos de datos predefinidos, veamos la instrucción CREATE TABLE, que define a una tabla con columnas que usan diferentes tipos de datos. En el siguiente ejemplo, la instrucción crea una tabla llamada ARTISTAS que incluye cuatro columnas:

```
CREATE TABLE ARTISTAS
( ID_ARTISTA          INT,
  NOMBRE_ARTISTA      VARCHAR(60) ,
  FDN_ARTISTA         DATE,
  POSTER_EN_EXISTENCIA  BOOLEAN );
```

ID_ARTISTA: INT	NOMBRE_ARTISTA: VARCHAR(60)	FDN_ARTISTA: DATE	POSTER_EN_EXISTENCIA: BOOLEAN
10001	Jennifer Warnes	1947-03-03	Falso
10002	Joni Mitchell	1943-11-07	Desconocido
10005	Bing Crosby	1904-05-02	Verdadero
10006	Patsy Cline	1932-09-08	Verdadero
10008	Placido Domingo	1941-01-21	Falso
10009	Luciano Pavarotti	1935-10-12	Desconocido

Figura 3-2 La tabla ARTISTAS definida con diferentes tipos de datos.

Como puede ver, la columna ID_ARTISTA tiene un tipo de datos numérico, la columna NOMBRE_ARTISTA tiene un tipo de datos de cadena, la columna FDN_ARTISTA tiene un tipo de datos de fecha y hora, y la columna POSTER_EN_EXISTENCIA tiene un tipo de datos booleano. La figura 3-2 muestra cómo puede verse esta tabla.

Creación de tipos definidos por el usuario

En el capítulo 1 se mencionó que el estándar SQL:2006 incorporó algunos de los principios de programación orientados a objetos (OOP) a su lenguaje. Un ejemplo de esto es el tipo definido por el usuario, a veces denominado como el tipo de datos definido por el usuario. El *tipo definido por el usuario* es una forma de tipo de datos (almacenada como un objeto de esquema) que está en parte definida por el programador y en parte basada en uno o más tipos de datos. SQL apoya dos formas de tipos definidos por el usuario:

- **Tipos estructurados** Estos tipos se forman por uno o más atributos, cada uno de los cuales está basado en otro tipo de datos, incluyendo tipos predefinidos, tipos construidos, y otros tipos estructurados. Además de asociarse con un tipo de datos, cada atributo puede incluir una cláusula predeterminada y puede especificar un cotejo. Un tipo estructurado puede incluir métodos en su definición. Un *método* es un tipo de función que se asocia con un tipo definido por el usuario. Una *función* es el nombre de una operación que realiza tareas predefinidas que no se podrían realizar normalmente mediante el uso de sólo instrucciones SQL. Es un tipo de ruta que toma parámetros de entrada (que a menudo son opcionales) y devuelve un valor único basado en esos parámetros.
- **Tipos distintos** Estos tipos se basan simplemente en tipos de datos predefinidos, y cualquier tipo de parámetro se define para ese tipo de datos si los parámetros son necesarios o deseados.

SQL proporciona la instrucción `CREATE TYPE` para definir los tipos definidos por el usuario. Sin embargo, el lenguaje utilizado para crear un tipo definido por el usuario puede variar de un producto a otro. Las características que se respaldan en el tipo definido por el usuario también varían ampliamente. Por ejemplo, SQL Server 2000 no respalda la instrucción `CREATE TYPE`, pero SQL Server 2005 sí lo hace.

A pesar de las diferencias y limitaciones en las aplicaciones de los productos, se trata por lo menos de proporcionar un ejemplo de cómo la instrucción `CREATE TYPE` se utiliza para crear un tipo diferente. En la siguiente instrucción se creó un tipo definido por el usuario que se basa en el tipo de datos `NUMERIC`:

```
CREATE TYPE SALARIO AS NUMERIC(8,2)
FINAL;
```

Este sencillo ejemplo es bastante claro: la creación de un tipo llamado `SALARIO` con un tipo de datos `NUMERIC(8,2)`. Sin embargo, la palabra clave `FINAL` es probablemente nueva. Cuando se especifica `FINAL`, le dice a SQL que no se definen subtipos para este tipo. La alternativa es especificar `NOT FINAL`, lo que significa que se pueden definir los subtipos para este tipo. Una vez creado el tipo, se puede utilizar en la definición de una columna como si fuera un tipo de datos predefinido:

```
CREATE TABLE EMPLEADO
( ID_EMPLEADO          INTEGER,
  SALARIO_EMPLEADO     SALARIO );
```

Cualquier valor que se agregue a la columna `SALARIO_EMPLEADO` se ajustará a las especificaciones del tipo de datos `NUMERIC` con una precisión de 8 y una escala de 2. Como resultado, el valor puede ser cualquiera desde -999999.99 hasta 999999.99. La mejor parte es que se puede utilizar el tipo definido por el usuario `SALARIO` en cualquier otro tipo de tablas que requieren valores similares.

NOTA

Como se observa, los tipos numéricos permiten números negativos, así como el número cero y números positivos. Si no se desean números negativos (que sería evidentemente el caso del salario de alguien), el tipo de datos por sí solo no haría el trabajo, pero se puede utilizar la restricción `CHECK` para ese fin. Las restricciones `CHECK` se analizarán en el capítulo 4.

Especificación de los valores predeterminados en una columna

Otra característica valiosa que SQL respalda es la habilidad de especificar un valor predeterminado para una columna cuando se utilice la instrucción `CREATE TABLE` para crear una tabla. La sintaxis para la definición de una columna con un valor predeterminado tendría este aspecto:

```
<nombre de columna> <tipo de datos> DEFAULT <valor predeterminado>
```

Los marcadores de posición <nombre de columna> y <tipo de datos>, con los cuales ahora debe estar familiarizado, son seguidos por la palabra clave DEFAULT. Después de la palabra clave DEFAULT, debe especificarse un valor para el marcador de posición <valor predeterminado>. Este valor puede ser un literal, que es un valor de datos de SQL (tal como la cadena 'A determinar' o el número 0); una función de valor de fecha y hora, que es una función que permite realizar operaciones relacionadas con fechas y tiempo (discutidas en el capítulo 10), o una función de usuario relacionada con una sesión, que es una función que devuelve la información relacionada con el usuario (discutida en el capítulo 10).

Cualquier tipo de valor utilizado para el marcador de posición <valor predeterminado> debe ajustarse a los requisitos de información del tipo de datos especificados en la definición de una columna. Por ejemplo, si se define una columna con un tipo de datos INT o un tipo de datos CHAR(4), no se puede especificar un valor predeterminado 'Desconocido'. En el primer caso, INT requiere un valor numérico, y en el segundo caso, CHAR(4) requiere que el valor contenga no más de cuatro caracteres.

En el siguiente ejemplo se utiliza la instrucción CREATE TABLE para definir a una tabla llamada ARTISTAS, que contiene tres columnas:

```
CREATE TABLE ARTISTAS
( ID_ARTISTA          INT,
  NOMBRE_ARTISTA      VARCHAR(60),
  LUGAR_DE_NACIMIENTO VARCHAR(60) DEFAULT 'Desconocido' );
```

Observe que la columna LUGAR_DE_NACIMIENTO incluye el valor predeterminado 'Desconocido'. El valor es aceptable, ya que se ajusta a los requisitos del tipo de datos VARCHAR(60). También observe que el valor predeterminado se encierra en comillas simples. Debe utilizar comillas simples para valores de cadena de caracteres. La figura 3-3 muestra cómo puede verse la tabla si se llena con filas de datos.

Si se insertan nuevas filas en esta tabla y no se sabe el lugar de nacimiento de un artista, el sistema automáticamente insertará el valor 'Desconocido'.

ID_ARTISTA: INT	NOMBRE_ARTISTA: VARCHAR(60)	LUGAR_DE_NACIMIENTO: VARCHAR(60)
10001	Jennifer Warnes	Desconocido
10002	Joni Mitchell	Fort MacLeod, Alberta, Canada
10005	Bing Crosby	Tacoma, Washington, Estados Unidos
10006	Patsy Cline	Winchester, Virginia, Estados Unidos
10008	Placido Domingo	Madrid, España
10009	Luciano Pavarotti	Desconocido

Figura 3-3 Un valor predeterminado 'Desconocido' para la columna LUGAR_DE_NACIMIENTO.

Pruebe esto 3-1 Creación de tablas en SQL

Probablemente notará que se ha utilizado información relacionada con CD para los ejemplos que se han mostrado hasta ahora. Se tratará este tema a lo largo del libro cuando se comience a construir la base de datos que registre el inventario de CD de una empresa pequeña. En este ejercicio se crean tres tablas que están relacionadas con la base de datos INVENTARIO, que se creó en el capítulo 2, Pruebe esto 2-1. Antes de empezar, eche un vistazo al modelo de datos simple (figura 3-4) que muestra las tres tablas que se crean. Cada tabla se representa por un rectángulo, con el nombre de la tabla en la parte superior del rectángulo y el nombre de las columnas, junto con los tipos de datos, que figuran dentro del rectángulo.

Se utilizará el modelo de datos a lo largo del libro (mientras evoluciona para convertirse en una estructura más compleja) para definir los objetos en la base de datos. También puede descargar el archivo Try_This_03.txt, que contiene instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de clientes de su RDBMS y conecte la base de datos INVENTARIO. Todos los objetos se crean dentro de esa base de datos. (Si el RDBMS no respalda la creación de una base de datos y en su lugar crea el esquema INVENTARIO_CD, debe crear todos los objetos dentro de ese esquema.)
2. La primera tabla que se crea es la tabla DISCOS_COMPACTOS. Observe que incluye tres columnas, dos de las cuales tienen un tipo de datos INT y una que tiene un tipo de datos VARCHAR(60). Esta tabla tiene información acerca de los discos compactos en el inventario. La columna ID_DISCO_COMPACTO contiene números que identifican únicamente a cada CD. La columna TITULO_CD contiene los nombres actuales de los CD. La columna ID_DISQUERA contiene números que identifican a las compañías que editan los CD. Introduzca la siguiente instrucción SQL en la ventana de entrada de la aplicación clientes:

```
CREATE TABLE DISCOS_COMPACTOS
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) ,
  ID_DISQUERA        INT );
```

3. Verifique que se introdujo la información correcta y ejecute la instrucción. Debe recibir un mensaje de confirmación de que la instrucción se ejecutó exitosamente.

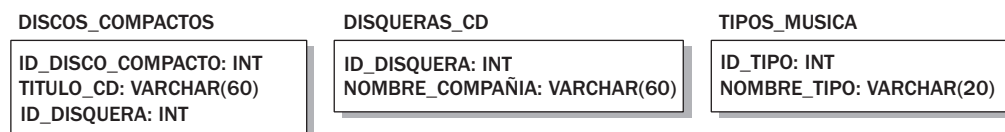


Figura 3-4 Modelo de datos simple de la base de datos INVENTARIO.

4. La próxima tabla que se crea es la tabla `DISQUERAS_CD`. La tabla incluye la columna `ID_DISQUERA`, que identifica únicamente a cada compañía que edita los CD, y la columna `NOMBRE_COMPañIA`, que enumera los nombres actuales de las compañías. Introduzca y ejecute el siguiente código:

```
CREATE TABLE DISQUERAS_CD
( ID_DISQUERA          INT,
  NOMBRE_COMPañIA      VARCHAR(60) );
```

5. La última tabla que se crea es la tabla `TIPOS_MUSICA`. La tabla incluye la columna `ID_TIPO`, que identifica únicamente cada categoría de música, y la columna `NOMBRE_TIPO`, que enumera los nombres actuales de las categorías de música (por ejemplo, blues o jazz). Introduzca y ejecute el siguiente código:

```
CREATE TABLE TIPOS_MUSICA
( ID_TIPO              INT,
  NOMBRE_TIPO          VARCHAR(20) );
```

6. Cierre la aplicación clientes.

Resumen de Pruebe esto

La base de datos ahora contiene tres tablas nuevas. Estas tablas sirven como base para otros ejercicios del libro. A medida que progrese mediante estos ejercicios, podrá modificar esas tablas, crear tablas adicionales, insertar datos en las tablas, y después hacer consultas y manipular esos datos. Para cuando complete todos los ejercicios, creará y completará una pequeña base de datos que almacene datos sobre un inventario de discos compactos.

Modificación de tablas en SQL

Tomando lo que aprendió acerca de la creación de tablas, puede utilizar la instrucción `ALTER TABLE` para modificar las definiciones de tablas base almacenadas en la base de datos (como objetos de esquema). En su forma más básica, la sintaxis para la instrucción `ALTER TABLE` se representa:

```
ALTER TABLE <nombre de la tabla>
  ADD [COLUMN] <definición de columna>
| ALTER [COLUMN] <nombre de columna>
  { SET DEFAULT <valor predeterminado> | DROP DEFAULT }
| DROP [COLUMN] <nombre columna> { CASCADE | RESTRICT }
```

La instrucción permite tomar tres diferentes acciones: añadir columnas, modificar columnas o eliminar columnas.

NOTA

La instrucción `ALTER TABLE` también permite añadir o eliminar limitaciones en una tabla. Una limitación en una tabla es una regla que restringe qué datos se introducen en la tabla. La limitación forma parte de la definición de la tabla, pero no forma parte de cualquiera de las definiciones de una columna específica. Las limitaciones se discuten con detalle en el capítulo 4.

El marcador de posición <definición de columna> en la cláusula `ADD [COLUMN]` es similar a la sección de definición de columna de la instrucción `CREATE TABLE`. Se proporciona un nombre de columna y un tipo de datos o dominio. También se tiene la opción de añadir una cláusula predeterminada, una limitación de columna o un cotejo. Por ejemplo, se puede utilizar la siguiente instrucción para modificar la tabla `ARTISTAS` de modo que incluya la columna `FDN_ARTISTA`:

```
ALTER TABLE ARTISTAS
  ADD COLUMN FDN_ARTISTA DATE;
```

A diferencia de la cláusula `ADD [COLUMN]`, la cláusula `ALTER [COLUMN]` se limita a dos acciones: establecer una predeterminación o eliminar una predeterminación (aunque existen aplicaciones de productos que permiten cambiar otras propiedades, tal como el tipo de datos o la precisión y la escala). Por ejemplo, suponga que la tabla `ARTISTS` incluye la columna `LUGAR_DE_NACIMIENTO`, pero no se define ninguna predeterminación para esa columna. Puede agregar una predeterminación utilizando la siguiente instrucción:

```
ALTER TABLE ARTISTAS
  ALTER COLUMN LUGAR_DE_NACIMIENTO SET DEFAULT 'Desconocido';
```

También puede eliminar la predeterminación utilizando la siguiente instrucción:

```
ALTER TABLE ARTISTAS
  ALTER COLUMN LUGAR_DE_NACIMIENTO DROP DEFAULT;
```

La cláusula final en la sintaxis (`DROP[COLUMN]`) proporciona dos opciones para eliminar una columna y los datos de la tabla: las palabras clave `CASCADE` y `RESTRICT`. Es posible que recuerde estas palabras clave de la discusión sobre la instrucción `DROP SCHEMA` en el capítulo 2. Si se especifica la opción `CASCADE`, la columna y los datos dentro de la columna se eliminan independientemente de si otros objetos hacen referencia a esa columna. Todas las vistas, restricciones, rutas o activadores que hacen referencia a la columna también se eliminan. Si se utiliza la opción `RESTRICT`, la columna se elimina sólo si no hay vistas, restricciones, rutas o activadores que hagan referencia a la columna. Por ejemplo, la siguiente instrucción elimina la columna `LUGAR_DE_NACIMIENTO` y los datos almacenados en la columna, independientemente de las dependencias:

```
ALTER TABLE ARTISTAS
  DROP COLUMN LUGAR_DE_NACIMIENTO CASCADE;
```

En general, es práctico conocer la instrucción `ALTER TABLE`, ya que las definiciones de las tablas cambian invariablemente, así como los tipos de datos almacenados en esas tablas. Sin embargo, esta instrucción, como la mayoría de las instrucciones de SQL, puede variar ampliamente de una aplicación a otra en términos de cómo los detalles de una instrucción se aplican. Por ejemplo, SQL Server no respalda las palabras clave `CASCADE` y `RESTRICT`. En Oracle, `CASCADE` debe escribirse como `CASCADE CONSTRAINTS`, y `RESTRICT` (que es el comportamiento predeterminado) no se respalda explícitamente. Como siempre, asegúrese de comprobar la documentación del producto.

Eliminación de tablas en SQL

Como puede imaginar, el proceso de eliminación de una tabla y sus datos almacenados es muy sencillo. La siguiente sintaxis muestra lo fácil que resulta este proceso:

```
DROP TABLE <nombre de la tabla>{ CASCADE | RESTRICT }
```

La única decisión que necesita tomar cuando se elimina una tabla es si debe escoger la opción CASCADE o RESTRICT. Como en los anteriores ejemplos de sintaxis, las dos opciones determinan si debe eliminar la tabla y sus datos aun cuando la tabla hace referencia a otros objetos. Si se utiliza CASCADE, la tabla y sus datos se eliminan, junto con todas las vistas, restricciones, rutas o activadores que hacen referencia a la tabla. Si se utiliza RESTRICT, la tabla se elimina sólo si no existen dichas dependencias. (Como con la cláusula DROP COLUMN, SQL Server no respalda CASCADE o RESTRICT, y Oracle permite sólo CASCADE CONSTRAINTS.) Por ejemplo, la siguiente instrucción elimina la tabla ARTISTAS y los datos almacenados en la columna, independientemente de las dependencias:

```
DROP TABLE ARTISTAS CASCADE;
```

Pregunta al experto

P: ¿Cómo se pueden eliminar los datos en una tabla, pero no la propia definición de la tabla?

R: En lugar de usar la instrucción DROP TABLE, tiene que utilizar la instrucción DELETE. La instrucción DELETE elimina todas las filas de una tabla o elimina sólo filas específicas, tal como se defina en la instrucción. Esto no es lo mismo que la instrucción DROP TABLE, que elimina la definición de la tabla y los datos. Se analizará la instrucción DELETE con más detalle en el capítulo 8. Muchas aplicaciones de productos también proporcionan la instrucción TRUNCATE, que suministra una forma rápida y eficaz para borrar todos los datos de una tabla. Sin embargo, la instrucción TRUNCATE no se incluye en el estándar SQL:2006.

P: Se expuso que cuando un valor predeterminado se define por una columna, el valor se inserta automáticamente en la columna cuando se agrega una fila a la tabla, pero no se especifica un valor para esa columna en particular. ¿Qué pasa si la definición de columna no incluye una predeterminación y se trata de insertar esa fila?

R: La acción tomada depende de si se permiten valores nulos dentro de la columna. Un valor nulo significa que el valor no se conoce. Esto no es lo mismo que cero, en blanco o predeterminado. Si un valor nulo se presenta, entonces los datos no están disponibles. Por predeterminación, todas las columnas permiten valores nulos, aunque se puede ignorar dicha predeterminación (discutido en el capítulo 4). Si trata de insertar una fila sin definir un valor específico, un valor nulo se inserta en esa columna si la columna permite valores nulos. Si la columna no permite valores nulos, no podrá insertar una fila sin definir un valor específico para esa columna.

(continúa)

P: A menudo se escucha el término “índices” en relación con la creación de tablas SQL. ¿Cómo se crean los índices?

R: Curiosamente, el estándar SQL:2006 no respalda la creación y mantenimiento de índices y tampoco proporciona una definición o los menciona de ninguna otra forma. Para los que no estén familiarizados con ellos, un índice es un conjunto de valores y referencias de búsqueda (en una tabla auxiliar) que corresponden a las filas de una tabla. Los índices aceleran las consultas y mejoran el rendimiento, haciendo el acceso a los datos mucho más eficiente, al igual que utilizar el índice de un libro ayuda a encontrar las cosas más rápidamente que la búsqueda secuencial de páginas. Como resultado, casi todos los RDBMS respaldan alguna forma de indexación y, de hecho, son una parte importante de ese producto. Sin embargo, el método utilizado para aplicar la indexación es muy variable; por lo tanto, cada producto proporciona su propio sistema para crear y mantener los índices. Por ejemplo, la instrucción CREATE INDEX está disponible en la mayoría de los productos; sin embargo, la sintaxis para la instrucción puede variar considerablemente. Como siempre, asegúrese de revisar la documentación del producto.

Pruebe esto 3-2 Modifique y elimine tablas en SQL

A lo largo del ciclo de vida de casi cualquier base de datos, la probabilidad de que los requerimientos del negocio cambien y la modificación de las bases de datos es casi una conclusión inevitable. Como resultado, no cabe duda que se pasará por situaciones en que las definiciones de tablas se modifican o eliminan. En este ejercicio se creará una tabla, se eliminará, se volverá a crear, y luego cambiará por la eliminación de una columna. En el momento en que termine, se añadirá una tabla más a la base de datos INVENTARIO y utilizará esa tabla en el último ejercicio. Puede descargar el archivo Try_This_03.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de cliente del RDBMS y conéctese a la base de datos INVENTARIO (o el esquema INVENTARIO_CD).
2. Cree una tabla llamada TIPOS_DISCO_COMPACTO. La tabla incluye la columna ID_DISCO_COMPACTO y la columna ID_TIPO. Ambas columnas se asignarán a un tipo de datos INT. Introduzca y ejecute el siguiente código:

```
CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  ID_TIPO            INT );
```

3. Ahora elimine la tabla de la base de datos. Introduzca y ejecute el siguiente código:

```
DROP TABLE TIPOS_DISCO_COMPACTO CASCADE;
```

4. Ahora se volverá a crear la tabla que se hizo en el paso 2, sólo que esta vez se incluirá una tercera columna llamada TITULO_CD con un tipo de datos VARCHAR(60). Introduzca y ejecute el siguiente código:

```
CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60),
  ID_TIPO            INT );
```

5. El siguiente paso es eliminar la columna TITULO_CD. Introduzca y ejecute el siguiente código:

```
ALTER TABLE TIPOS_DISCO_COMPACTO
  DROP COLUMN TITULO_CD CASCADE ;
```

6. La tabla TIPOS_DISCO_COMPACTO debe ahora contener sólo las columnas ID_DISCO_COMPACTO y la columna ID_TIPO. Cierre la aplicación clientes.

Resumen de Pruebe esto

La base de datos INVENTARIO ahora debe contener cuatro tablas: DISCOS_COMPACTOS, DISQUERAS_CD, TIPOS_MUSICA y TIPOS_DISCO_COMPACTO. La tabla TIPOS_DISCO_COMPACTO, que se acaba de crear, contiene dos columnas, ID_DISCO_COMPACTO y ID_TIPO, ambas definidas con un tipo de datos INT. Estos ejercicios continuarán la construcción de esta base de datos agregando tablas nuevas y modificando las existentes.

Autoexamen Capítulo 3

1. ¿Qué tipos de tablas base se pueden crear utilizando una instrucción CREATE TABLE?
 - A Tablas base persistentes
 - B Tablas base temporales globales
 - C Tablas temporales locales creadas
 - D Tablas temporales locales declaradas
2. ¿Cuál es la principal diferencia entre una tabla temporal global y una tabla temporal local creada?
3. Está creando una tabla llamada AGENTES. La tabla incluye la columna ID_AGENTE, que tiene un tipo de datos INT, y la columna NOMBRE_AGENTE, que tiene un tipo de datos CHAR(60). ¿Qué instrucción SQL utilizaría?
4. ¿Cuáles son los tres tipos de datos que soporta SQL?
5. ¿Cuáles son los cuatro tipos de datos de cadena?

6. Un(a) _____ es un tipo de datos que permite valores que se basan en bits de datos, en lugar de conjuntos de caracteres o cotejos. Este tipo de datos permite sólo valores de 0 y 1.
7. ¿Cuál es la precisión y la escala del número 5293.472?
8. ¿Cuáles son las diferencias entre los tipos de datos numéricos exactos y los tipos de datos numéricos aproximados?
9. ¿Cuáles tipos de datos son tipos de datos numéricos exactos?
 - A DOUBLE PRECISION
 - B DECIMAL
 - C REAL
 - D SMALLINT
10. Un tipo de datos _____ especifica los valores de una fecha por año, mes y día.
11. ¿Cuáles son las dos formas de tipos de datos de intervalo que soporta SQL?
12. ¿Qué tipo de datos debe utilizarse para soportar una construcción verdadero/falso que pueda ser utilizada para comparar valores?
13. Está creando un tipo definido por el usuario distinto llamado CIUDAD. El tipo de usuario se basa en el tipo de datos CHAR(40). ¿Qué instrucción SQL utilizaría?
14. Se crea una tabla llamada CLIENTES. La tabla incluye la columna NOMBRE_CLIENTE y la columna CIUDAD_CLIENTE. Ambas columnas tienen un tipo de datos VARCHAR(60). La columna CIUDAD_CLIENTE también tiene el valor predeterminado *Seattle*. ¿Qué instrucción SQL utilizaría?
15. ¿Qué instrucción SQL deberá utilizarse para eliminar una columna de una tabla existente?
16. ¿Qué instrucción SQL deberá utilizarse para eliminar la definición de una tabla y todos los datos de SQL de una base de datos?
17. Una base de datos incluye una tabla llamada CANTANTES_OPERA. Se quiere agregar una columna llamada NACIONALIDAD a esa tabla. La columna debe tener el tipo de datos VARCHAR(40). ¿Qué instrucción SQL utilizaría?
18. Se desea eliminar la definición de la tabla CANTANTES_OPERA de la base de datos. También se quieren eliminar todos los datos y cualquier dependencia de la tabla. ¿Qué instrucción SQL utilizaría?

Capítulo 4

Implementación de la integridad de datos



Habilidades y conceptos clave

- Entienda las restricciones de integridad
 - Utilice restricciones NOT NULL
 - Añada restricciones UNIQUE
 - Añada restricciones PRIMARY KEY
 - Añada restricciones FOREIGN KEY
 - Defina restricciones CHECK
-

Una base de datos SQL debe hacer más que sólo almacenar datos. Se debe asegurar que el almacenamiento de los datos es el correcto. Si la integridad de los datos se compromete, los datos pueden ser inexactos o inconsistentes, poniendo en cuestionamiento la fiabilidad de la base de datos. Con el fin de asegurar la integridad de los datos, SQL proporciona una serie de *restricciones de integridad*, reglas que se aplican a la base de datos para restringir los valores que se pueden colocar en esas tablas. Se pueden aplicar restricciones a columnas individuales, a tablas individuales o a múltiples tablas. En este capítulo se discute cada tipo de restricción y se explica cómo se pueden aplicar a la base de datos SQL.

Entienda las restricciones de integridad

Las restricciones de integridad de SQL, a las que se les conoce simplemente como restricciones, pueden dividirse en tres categorías:

- **Restricciones relacionadas con tablas** Un tipo de restricción que se precisa dentro de la definición de una tabla. Las restricciones definidas a nivel de tabla pueden aplicar a una o más columnas.
- **Afirmaciones** Un tipo de restricción que se precisa dentro de una definición de afirmación (separado de la definición de una tabla). Una afirmación puede relacionarse con una o más tablas.
- **Restricciones de dominio** Un tipo de restricción que se precisa dentro de una definición de dominio (separado de la definición de una tabla). Una restricción de dominio se asocia con cualquier columna que se define dentro del dominio específico.

De estas tres categorías de restricciones, las restricciones relacionadas con tablas son las más comunes e incluyen el mayor número de opciones de restricción. Las restricciones relacionadas con tablas se dividen en dos subcategorías: restricciones de tabla y restricciones de columna. Las restricciones en ambas subcategorías se precisan en la definición de la tabla. Una restricción de columna se incluye con la definición de la columna, y una restricción de tabla se incluye como un elemento de la tabla, similar a la manera en que las columnas se definen como elementos de la tabla. (En el capítulo 3 se analizaron los elementos de tablas y las definiciones de columnas.) Tanto

las restricciones de columna como las restricciones de tabla soportan una serie de diferentes tipos de restricciones. Éste no es el caso para las restricciones de afirmación y dominio, que se limitan sólo a un tipo de restricción. La figura 4-1 proporciona una descripción general de los tipos de restricciones que se pueden crear.

En la parte superior de la ilustración se pueden observar las tres categorías de restricciones. Debajo de la categoría de restricciones relacionadas con tablas están las subcategorías restricciones de columna y restricciones de tabla, cada una de las cuales contienen tipos específicos de restricciones. Por ejemplo, las restricciones de tabla pueden incluir restricciones únicas (restricciones UNIQUE y de PRIMARY KEY), referenciales (restricciones FOREIGN KEY) y CHECK, mientras que las restricciones de columna pueden incluir la restricción NOT NULL, así como restricciones únicas, referenciales y CHECK. Sin embargo, las de dominios y afirmaciones sólo soportan las restricciones CHECK.

NOTA

En algunos lugares, el estándar SQL:2006 utiliza el término “restricción de tabla” para referirse a ambos tipos de restricciones relacionadas con tablas. Utilizaré el término “relacionado con tablas” para evitar confusión.

Como se muestra en la figura 4-1 existen cinco diferentes tipos de restricciones: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY y CHECK. En SQL, las restricciones UNIQUE y PRIMARY KEY se consideran restricciones únicas, y las restricciones de FOREIGN KEY se consideran como restricciones referenciales. El resto del capítulo se dedica a explicar lo que significan cada una de estas restricciones y cómo aplicarlas.

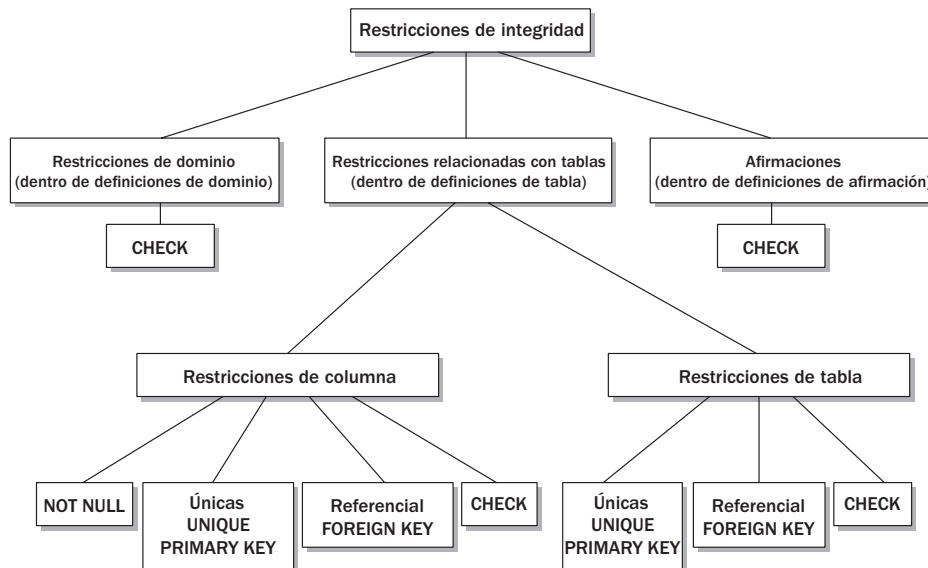


Figura 4-1 Tipos de restricciones de integridad de SQL.

Utilice restricciones NOT NULL

En el capítulo 3 se dijo que *nulo* significa que un valor no está definido o no se conoce. Esto no es lo mismo que cero, en blanco, una cadena vacía o un valor predeterminado. En lugar de ello, indica que un valor de dato está ausente. Se puede pensar en un valor nulo como una marca. (Una *marca* es un carácter, número o bit que indica un cierto hecho sobre una columna. La marca sirve como un marcador que designa una condición particular o la existencia de algo.) En el caso nulo, si una columna no proporciona ningún valor, se coloca la marca, indicando que el valor es desconocido, o nulo. Cada columna tiene una característica *de nulo* que indica si la columna acepta valores nulos. Por predeterminación, todas las columnas aceptan valores nulos. Sin embargo, se puede anular la característica de nulo predeterminada utilizando una restricción NOT NULL, que indica que la columna *no* acepta valores nulos.

NOTA

Algunos RDBMS permiten modificar la característica de nulo predeterminada de cualquier columna nueva que se crea. Además, algunos sistemas respaldan una restricción NULL, que se utiliza para designar que una columna acepta valores nulos.

La restricción NOT NULL sólo puede utilizarse como una restricción de columna. No se soporta para restricciones de tablas, afirmaciones o restricciones de dominio. La aplicación de una restricción NOT NULL es un proceso muy sencillo. Simplemente utilice la siguiente sintaxis cuando se crea una definición de columna:

```
<nombre de columna> { <tipo de datos> | <dominio> } NOT NULL
```

Por ejemplo, suponga que quiere crear una tabla llamada ARTISTAS_DISCO_COMPACTO que requiere tres columnas: ID_ARTISTA, NOMBRE_ARTISTA y LUGAR_DE_NACIMIENTO. Quiere asegurarse de que cualquier fila que se agregue a la tabla incluya un valor para la columna ID_ARTISTA y un valor para la columna NOMBRE_ARTISTA. Para hacer esto, debe añadir una restricción NOT NULL en ambas definiciones de columna, como se muestra en la siguiente instrucción de SQL:

```
CREATE TABLE ARTISTAS_DISCO_COMPACTO
( ID_ARTISTA          INT          NOT NULL,
  NOMBRE_ARTISTA      VARCHAR(60)  NOT NULL,
  LUGAR_DE_NACIMIENTO VARCHAR(60) );
```

Observe que la columna LUGAR_DE_NACIMIENTO no incluye una restricción NOT NULL. Como resultado, si no se suministra un valor para esa columna (cuando se inserta una fila), un valor nulo se insertará. (La marca se coloca.) La figura 4-2 muestra el aspecto que puede tener la tabla si se insertan filas que no contienen valores para la columna LUGAR_DE_NACIMIENTO.

Como puede verse, las columnas ID_ARTISTA, NOMBRE_ARTISTA no contienen (y no pueden contener) valores nulos. La columna LUGAR_DE_NACIMIENTO, por otro lado, contiene dos valores nulos.

ID_ARTISTA: INT:	NOMBRE_ARTISTA: VARCHAR(60)	LUGAR_DE_NACIMIENTO: VARCHAR(60)
10001	Jennifer Warnes	NULL
10002	Joni Mitchell	Fort MacLeod, Alberta, Canada
10005	Bing Crosby	Tacoma, Washington, Estados Unidos
10006	Patsy Cline	Winchester, Virginia, Estados Unidos
10008	Placido Domingo	Madrid, España
10009	Luciano Pavarotti	NULL

Figura 4-2 Valores nulos que aparecen en la columna LUGAR_DE_NACIMIENTO en la tabla ARTISTAS_DISCO_COMPACTO.

Añada restricciones UNIQUE

Si se remite de nuevo a la figura 4-1, notará que tanto las restricciones de columna como las restricciones de tabla respaldan restricciones únicas. También se observa que hay dos tipos de restricciones únicas: UNIQUE y de PRIMARY KEY. Esta sección se enfoca en la restricción UNIQUE. La restricción de PRIMARY KEY se analizará en la sección “Añada restricciones PRIMARY KEY”, más adelante en este capítulo.

La restricción UNIQUE permite exigir que una columna o conjunto de columnas contengan valores únicos, valores significativos que sean diferentes de todas las demás filas en la misma tabla. Por ejemplo, eche un vistazo a la figura 4-3, que muestra la tabla CD_INVENTARIO. La tabla contiene tres columnas: NOMBRE_ARTISTA, NOMBRE_CD y DERECHOSDEAUTOR.

NOMBRE_ARTISTA: VARCHAR(40)	NOMBRE_CD: VARCHAR(60)	AÑO_DERECHOSDEAUTOR: INT
Jennifer Warnes	Famous Blue Raincoat	1991
Joni Mitchell	Blue	1971
William Ackerman	Past Light	1983
Kitaro	Kojiki	1990
Bing Crosby	That Christmas Feeling	1993
Patsy Cline	Patsy Cline: 12 Greatest Hits	1988

Figura 4-3 Tabla CD_INVENTARIO con las columnas NOMBRE_ARTISTA, NOMBRE_CD y AÑO_DERECHOSDEAUTOR.

Puede decidir que desea que los valores en la columna `NOMBRE_CD` sean únicos, de modo que no haya dos nombres iguales de CD. Si se aplica una restricción `UNIQUE` a la columna, no será posible insertar una fila que contenga un valor de `NOMBRE_CD` que ya exista en la tabla. Suponga que ahora se da cuenta que hacer los valores `NOMBRE_CD` únicos no es una buena idea, ya que es posible para más de un CD compartir el mismo nombre. Decide adoptar otro enfoque y utilizar una restricción `UNIQUE` en las columnas `NOMBRE_ARTISTA` y `NOMBRE_CD`. De esa manera, ningún par `NOMBRE_ARTISTA/NOMBRE_CD` se puede repetir. Se puede repetir el valor `NOMBRE_ARTISTA` o el valor `NOMBRE_CD`, pero no se puede repetir la misma combinación de los dos. Por ejemplo, la tabla ya contiene la fila con el valor `NOMBRE_ARTISTA` de Joni Mitchell y el valor `NOMBRE_CD` de Blue. Si una restricción `UNIQUE` se aplica a estas dos columnas, no se podrá añadir otra fila que contenga *ambos* valores.

NOTA

Cabe señalar que las tablas utilizadas para ilustrar los conceptos en este capítulo no son necesariamente buenos diseños. Por ejemplo, los nombres de las personas y cosas rara vez son buenas opciones para identificar únicamente las filas de los datos porque son bastante largos (comparado con números), tienden a cambiar, y son propensos a problemas con valores duplicados. Sin embargo, estas tablas se escogieron ya que ilustran bien los conceptos.

Ahora que se tiene un entendimiento básico de cómo las restricciones `UNIQUE` se aplican, echemos un vistazo a la sintaxis que se utiliza para crearlas. Recuerde que se dijo que se puede crear una restricción `UNIQUE` que sea una restricción de columna o una restricción de tabla. Para crear una restricción de columna, añádala como parte de la definición de una columna, como se muestra en la siguiente sintaxis:

```
<nombre de columna> { <tipo de datos> | <dominio> } UNIQUE
```

Si se quiere añadir una restricción única como una restricción de tabla, se debe agregar como un elemento de definición de tabla, como se muestra en la siguiente sintaxis:

```
[ CONSTRAINT <nombre de la restricción> ]  
UNIQUE ( <nombre de columna> [{, <nombre de columna>}... ] )
```

Como se puede observar, aplicar una restricción `UNIQUE` como una restricción de columna es un poco más simple que aplicarla como una restricción de tabla. Sin embargo, si se aplica la restricción al nivel de columna, se puede aplicar sólo a una columna. A pesar de si se utiliza una restricción de columna o de tabla, se pueden definir tantas restricciones `UNIQUE` como sean necesarias en una sola definición de tabla.

Ahora volvamos a la tabla de la figura 4-3 que se usará para crear ejemplos de códigos para aplicar restricciones `UNIQUE`. En el primer ejemplo, se aplicó una restricción `UNIQUE` a la columna `NOMBRE_CD`:

```
CREATE TABLE CD_INVENTARIO  
( NOMBRE_ARTISTA    VARCHAR(40) ,  
  NOMBRE_CD         VARCHAR(60) UNIQUE ,  
  DERECHOSDEAUTOR   INT );
```

También se pueden aplicar restricciones UNIQUE a otras columnas, pero no tendrían el mismo efecto que la combinación de dos columnas en una restricción de tabla, como se muestra en el siguiente ejemplo:

```
CREATE TABLE CD_INVENTARIO
( NOMBRE_ARTISTA VARCHAR(40),
  NOMBRE_CD       VARCHAR(60),
  DERECHOSDEAUTOR INT,
  CONSTRAINT UN_ARTISTA_CD UNIQUE ( NOMBRE_ARTISTA, NOMBRE_CD ) );
```

Las columnas NOMBRE_ARTISTA y NOMBRE_CD ahora deben contener combinaciones únicas de valores con el fin de que una fila se añada a la tabla CD_INVENTARIO.

Hasta ahora se ha dicho que una restricción UNIQUE evita la duplicación de valores que se introducen en una columna o columnas definidas con esa restricción. Sin embargo, hay una excepción para esto (el valor nulo). Una restricción UNIQUE permite múltiples valores nulos en una columna. Como con otras columnas, los valores nulos se permiten por predeterminación. Sin embargo, se puede anular la predeterminación utilizando una restricción NOT NULL conjuntamente con una restricción UNIQUE. Por ejemplo, se puede añadir una restricción NOT NULL a la definición de columna NOMBRE_CD:

```
CREATE TABLE CD_INVENTARIO
( NOMBRE_ARTISTA VARCHAR(40),
  NOMBRE_CD       VARCHAR(60) NOT NULL UNIQUE,
  DERECHOSDEAUTOR INT );
```

También se puede añadir una restricción NOT NULL a la definición de una columna que hace referencia a una restricción de tabla:

```
CREATE TABLE CD_INVENTARIO
( NOMBRE_ARTISTA VARCHAR(40),
  NOMBRE_CD       VARCHAR(60) NOT NULL,
  DERECHOSDEAUTOR INT,
  CONSTRAINT UN_ARTISTA_CD UNIQUE (NOMBRE_CD) );
```

En cada caso, tanto la restricción NOT NULL como la restricción UNIQUE se aplican en la columna NOMBRE_CD, lo que significa que los valores NOMBRE_CD deben ser únicos y sin valores nulos.

Añada restricciones PRIMARY KEY

Como se mencionó en la sección “Añada restricciones UNIQUE”, una restricción PRIMARY KEY, como una restricción UNIQUE, es un tipo de restricción única de SQL. Ambos tipos de restricciones permiten sólo valores en columnas específicas, ambos tipos pueden aplicarse a una o más columnas, y ambos tipos se pueden definir como restricciones de columna o restricciones de tabla. Sin embargo, las restricciones PRIMARY KEY tienen dos restricciones que aplican sólo a éstas:

- Una columna que se define con una restricción PRIMARY KEY no puede contener valores nulos. No importa si la definición de columna especifica NOT NULL (la columna no puede contener valores nulos, ya que es una restricción PRIMARY KEY).
- Sólo una restricción PRIMARY KEY puede definirse para cada tabla.

El motivo de estas restricciones es el rol que una restricción de clave primaria (identificador único) juega en la tabla. Como podría recordar del capítulo 1, cada fila en una tabla debe ser única. Esto es importante, ya que SQL no puede diferenciar entre dos filas que son completamente idénticas; por lo tanto, no se puede actualizar o eliminar una fila duplicada sin hacer lo mismo con la otra. La clave primaria para una tabla se escoge por el diseñador de la base de datos de claves de candidato disponibles. Una *clave de candidato* es un conjunto de una o más columnas que identifican de forma exclusiva a cada fila. Por ejemplo, en la figura 4-4, la única clave de candidato razonable en la tabla CD_ARTISTAS es la columna ID_ARTISTA. Cada valor en la columna es único. De esa manera, aun si los valores NOMBRE_ARTISTA y AGENCIAS se duplican, la fila sigue siendo única ya que el valor ID_ARTISTA siempre es único.

La singularidad de una clave de candidato se puede ejecutar con una restricción UNIQUE o con una restricción PRIMARY KEY. Sin embargo, cada tabla debe incluir una clave primaria aun si no se define ninguna restricción UNIQUE. Esto se considera como una de las mejores prácticas en la industria, ya que una clave primaria no puede aceptar valores nulos, lo que la hace la medida definitiva por la cual la singularidad de una fila se puede asegurar. Las claves primarias también son útiles cuando una tabla hace referencia de otra a través del uso de claves foráneas. (Ver la sección “Añada restricciones FOREIGN KEY” más adelante en este capítulo.) Además, algunos RDBMS requieren la definición de claves primarias en determinadas circunstancias, como cuando una columna de tabla se incluye en un índice de texto completo.

Para definir una clave primaria, se debe utilizar una restricción de PRIMARY KEY para especificar qué columna o columnas servirán como la clave primaria de una tabla. El proceso de definición de una restricción de PRIMARY KEY es muy similar a la de la definición de la restricción



ID_ARTISTA: INT	NOMBRE_ARTISTA: VARCHAR(60)	ID_AGENCIA: INT
10001	Jennifer Warnes	2305
10002	Joni Mitchell	2306
10003	William Ackerman	2306
10004	Kitaro	2345
10005	Bing Crosby	2367
10006	Patsy Cline	2049
10007	Jose Carreras	2876
10008	Placido Domingo	2305
10009	Luciano Pavarotti	2345

Figura 4-4 La clave de candidato en la tabla CD_ARTISTAS.

UNIQUE. Si se quiere añadir una restricción PRIMARY KEY a una definición de columna, utilice la siguiente sintaxis:

<nombre de columna> { <tipo de datos> | <dominio> } PRIMARY KEY

Si se quiere añadir una restricción PRIMARY KEY como una restricción de tabla, se debe agregar como un elemento de tabla en la definición de la tabla, como se muestra en la siguiente sintaxis:

[CONSTRAINT <nombre de la restricción>]
PRIMARY KEY (<nombre de columna> [{, <nombre de columna> }...])

Como con la restricción UNIQUE, se puede utilizar una restricción de columna para definir una clave primaria si se incluye sólo una columna en la definición. Por ejemplo, si se define una restricción de PRIMARY KEY para la tabla mostrada en la figura 4-4, se utiliza la siguiente instrucción SQL:

```
CREATE TABLE CD_ARTISTAS
( ID_ARTISTA      INT              PRIMARY KEY,
  NOMBRE_ARTISTA  VARCHAR(60) ,
  ID_AGENCIA      INT ) ;
```

Si se quiere aplicar la restricción a varias columnas (o si simplemente se quiere mantener como una definición separada, entonces se debe utilizar la restricción de tabla:

```
CREATE TABLE CD_ARTISTAS
( ID_ARTISTA      INT,
  NOMBRE_ARTISTA  VARCHAR(60) ,
  ID_AGENCIA      INT,
  CONSTRAINT PK_ID_ARTISTA PRIMARY KEY ( ID_ARTISTA, NOMBRE_ARTISTA ) ) ;
```

Este método crea una clave primaria en las columnas ID_ARTISTA y NOMBRE_ARTISTA, de modo que los valores combinados de ambas columnas deben ser únicos, aunque puede existir duplicación dentro de una columna individual. Un experimentado diseñador de bases de datos señalará rápidamente que ésta es una *superclave*, lo que significa que tiene más columnas que el mínimo necesario para formar una clave primaria. Y esto es cierto (ID_ARTISTA por sí misma es única, y realmente no es necesario añadirle NOMBRE_ARTISTA con el fin de formar una clave primaria, y si se quiere estar seguro de que los valores duplicados de ID_ARTISTA no se incluyen en la tabla, lo que significa que se debe tener una clave primaria con sólo un ID_ARTISTA en ella). Se hace aquí para ilustrar que una clave primaria puede contener múltiples columnas y para definir una forma que una restricción de tabla debe utilizar.

Puede encontrar que desea definir tanto una restricción PRIMARY KEY como una restricción UNIQUE en una tabla. Para hacerlo, simplemente defina las restricciones como normalmente lo hace. Por ejemplo, la siguiente instrucción de SQL define una restricción PRIMARY KEY en la columna ID_ARTISTA y una restricción UNIQUE en la columna NOMBRE_ARTISTA:

```
CREATE TABLE CD_ARTISTAS
( ID_ARTISTA      INT              PRIMARY KEY,
  NOMBRE_ARTISTA  VARCHAR(60) ,
  ID_AGENCIA      INT,
  CONSTRAINT UN_NOMBRE_ARTISTA UNIQUE (NOMBRE_ARTISTA) ) ;
```

Se alcanza el mismo resultado con el siguiente código:

```
CREATE TABLE CD_ARTISTAS
( ID_ARTISTA      INT,
  NOMBRE_ARTISTA  VARCHAR(60) UNIQUE,
  ID_AGENCIA      INT,
  CONSTRAINT PK_ID_ARTISTA PRIMARY KEY (ID_ARTISTA) );
```

NOTA

Se utiliza una restricción **UNIQUE** en estas instrucciones de SQL sólo como una forma de demostrar cómo se puede usar la restricción en una tabla con una clave primaria. Lo más probable es que no se quiera utilizar una restricción **UNIQUE** para la columna **NOMBRE_ARTISTA**, ya que es posible para dos artistas compartir el mismo nombre. (Por ejemplo, dos diferentes artistas de blues, ambos de los cuales vivieron en la primera parte del siglo pasado, con el nombre de Sonny Boy Williamson.)

Pregunta al experto

- P:** ¿Pueden las columnas en una tabla pertenecer tanto a una restricción **UNIQUE** como a una restricción **PRIMARY KEY**?
- R:** Sí, siempre y cuando no sean exactamente las mismas columnas. Por ejemplo, suponga que tiene una tabla que incluye tres columnas: **ID_ARTISTA**, **NOMBRE_ARTISTA** y **LUGAR_DE_NACIMIENTO**. Se puede definir una restricción **PRIMARY KEY** que incluya las columnas **ID_ARTISTA** y **NOMBRE_ARTISTA**, que garantice pares de valores únicos en esas dos columnas, pero los valores dentro de columnas individuales pueden aún estar duplicados. Sin embargo, se puede entonces definir una restricción **UNIQUE** que incluya sólo la columna **NOMBRE_ARTISTA** para garantizar que esos valores también son únicos. (Ciertamente este no es el mejor diseño, pero ilustra el punto.) También se puede crear una restricción **UNIQUE** que incluya las columnas **NOMBRE_ARTISTA** y **LUGAR_DE_NACIMIENTO** para garantizar pares de valores únicos en esas dos columnas. Lo único que se puede hacer es crear una construcción **UNIQUE** que incluya exactamente las mismas columnas como en la restricción **PRIMARY KEY** y viceversa.
- P:** Se expuso que una columna que se incluye en una restricción **PRIMARY KEY** no acepta valores nulos. ¿Qué sucede si esa columna se configura también con una restricción **NOT NULL**?
- R:** Nada diferente sucede. La tabla se crea de la misma forma. Una definición de columna que incluye una **PRIMARY KEY** está diciendo lo mismo que una definición de columna que incluye **NOT NULL PRIMARY KEY**. De hecho, antes de SQL-92, las palabras clave **NOT NULL** se requerían en todas las columnas incluidas en una restricción **PRIMARY KEY**.

Cabe mencionar lo mismo para las restricciones UNIQUE. No fue sino hasta SQL-92 que se permitieron los valores nulos en columnas incluidas en una restricción UNIQUE, que las diferenciaban claramente de las restricciones PRIMARY KEY. También desconfíe de las diferentes implementaciones entre los proveedores. Por ejemplo, Oracle añade automáticamente restricciones NOT NULL a columnas incluidas en una restricción PRIMARY KEY, mientras que SQL Server (o por lo menos algunas versiones) expone un error si se intenta crear una restricción de PRIMARY KEY utilizando columnas que no tienen especificada una restricción NOT NULL.

Añada restricciones FOREIGN KEY

Hasta este punto, los tipos de restricciones que se analizaron tienen que ver principalmente con la garantía de la integridad de los datos dentro de una tabla. La restricción NOT NULL evita el uso de valores nulos dentro de una columna, y las restricciones UNIQUE y PRIMARY KEY garantizan la singularidad de los valores dentro de una columna o conjunto de columnas. Sin embargo, la restricción FOREIGN KEY es diferente en el sentido de que se ocupa de cómo los datos en una tabla hacen referencia a los datos en otra tabla, que es la razón por la que se conoce como una *restricción referencial* (en relación con otra tabla). (De hecho, hay una excepción, llamada *relación recursiva*, donde la clave foránea hace referencia a otra fila en la misma tabla, pero por el momento se ignorará este caso especial para centrarse en los fundamentos.)

Posiblemente recuerde del capítulo 1 que las tablas en una base de datos relacional están unidas entre sí de una manera significativa con el fin de garantizar la integridad de los datos. Esta asociación entre tablas forma una relación que proporciona una *integridad referencial* entre las tablas. La integridad referencial evita la manipulación de los datos en una tabla que afecte negativamente los datos en otra tabla. Echemos un vistazo a un ejemplo que ilustra este punto. La figura 4-5 muestra dos tablas (TITULOS_CD y EDITORES_CD) que están definidas con una clave primaria.

TITULOS_CD			EDITORES_CD	
ID_TITULO_CD: INT	TITULO_CD: VARCHAR(60)	ID_EDITOR: INT	ID_EDITOR: INT	NOMBRE_COMPAÑIA: VARCHAR(60)
11001	Famous Blue Raincoat	5422	5403	MCA Records
11002	Blue	5402	5402	Reprise Records
11003	Past Light	5412	5409	Geffen
11004	Kojiki	5409	5412	Windham Hill Records
11005	That Christmas Feeling	5403	5422	Private Music
11006	Patsy Cline: 12 Greatest Hits	5403		

Figura 4-5 Relación entre las tablas TITULOS_CD y EDITORES_CD.

La columna `ID_TITULO_CD` en la tabla `TITULOS_CD` se configuró con una restricción `PRIMARY KEY`, así como la columna `ID_EDITOR` en la tabla `EDITORES_CD`. Ambas columnas están sombreadas en la ilustración.

Observe que la tabla `TITULOS_CD` contiene una columna llamada `ID_EDITOR`. Esta columna incluye valores de la columna `ID_EDITOR` de la tabla `EDITORES_CD`. De hecho, los valores de `ID_EDITOR` en la tabla `TITULOS_CD` incluyen sólo valores que vienen de la columna `ID_EDITOR` en la tabla `EDITORES_CD`. No será posible insertar una fila en `TITULOS_CD` si el valor `ID_EDITOR` no figura en la tabla `EDITORES_CD`. Al mismo tiempo, si se modifica o elimina el valor `ID_EDITOR` en la tabla `EDITORES_CD`, se puede predecir el resultado de la acción si esos mismos valores existen en la tabla `TITULOS_CD`. Bajo ninguna circunstancia se querrá eliminar a un editor y dejar los valores `ID_EDITOR` en la tabla `TITULOS_CD` que hacen referencia a un editor que ya no existe. Estos resultados se pueden lograr utilizando una restricción de `FOREIGN KEY`. Una restricción `FOREIGN KEY` aplica la integridad referencial entre dos tablas para garantizar que no se tome ninguna acción en cualquiera de las tablas que afecte negativamente los datos protegidos por la restricción.

En las tablas mostradas en la figura 4-5, la restricción `FOREIGN KEY` debe configurarse en la columna `ID_EDITOR` de la tabla `TITULOS_CD`. La restricción `FOREIGN KEY` restringe los valores en esa columna de los valores de un clave de candidato (a menudo la clave primaria) en la tabla relacionada. Sólo se permiten los valores de datos válidos en la columna o columnas `FOREIGN KEY`.

NOTA

La tabla que contiene la clave foránea es la tabla de referencia. La tabla a la que se hace referencia en la clave foránea es la tabla referenciada. Asimismo, la columna o columnas que componen la clave foránea en la tabla referenciada hacen referencia a éstas como columnas referenciadas. Al hacer referencia a las columnas por la clave foránea, éstas son columnas referenciadas.

Cuando se crea una restricción `FOREIGN KEY`, se deben seguir varias directrices:

- Las columnas referenciadas se deben definir con una restricción `UNIQUE` o una `PRIMARY KEY`. Como podrá adivinar, la de clave primaria es la más utilizada por las columnas referenciadas.
- Una restricción `FOREIGN KEY` se puede crear como una restricción de tabla o una restricción de columna. Si se crea la clave foránea como una restricción de columna, se puede incluir sólo una columna. Si se crea la clave foránea como una restricción de tabla, se pueden incluir una o más columnas.
- La clave foránea en la tabla de referencia debe incluir el mismo número de columnas que sean referenciadas, y las columnas de referencia deben cada una configurarse con los mismos tipos de datos que su contraparte de referencia. Sin embargo, las columnas de referencia no deben tener necesariamente los mismos nombres que las columnas referenciadas.
- Si no se especifican las columnas referenciadas cuando se define una restricción `FOREIGN KEY`, entonces las columnas definidas en la clave primaria de la tabla referenciada se utilizan como columnas referenciadas.

Estas directrices estarán más claras cuando explique cómo implementar una clave foránea. En primer lugar, echemos un vistazo a la sintaxis básica utilizada para crear esa restricción. Si se quiere añadir una restricción FOREIGN KEY como una restricción de columna, se debe agregar la restricción a la definición de una columna, como se muestra en la siguiente sintaxis:

```
<nombre de columna> { <tipo de datos> | <dominio> } [NOT NULL]
REFERENCES <tabla referenciada> [( <columnas referenciadas> )]
[MATCH { FULL | PARTIAL | SIMPLE }]
[ < acción referencial desencadenada > ]
```

Si desea añadir una restricción FOREIGN KEY como restricción de tabla debe agregarla como elemento de tabla en la definición, como en la siguiente sintaxis:

```
[CONSTRAINT <nombre de la restricción>]
FOREIGN KEY (<columna referenciada> [{, <columna referenciada> }...])
REFERENCES <tabla referida> [(columnas de referencia>)]
[MATCH { FULL | PARTIAL | SIMPLE }]
[ < acción referencial desencadenada > ]
```

Como puede observarse, la restricción FOREIGN KEY es un poco más compleja que la sintaxis de la restricción que se había analizado hasta ahora. Sin embargo, crear la restricción básica FOREIGN KEY es un proceso relativamente simple. Echemos un vistazo a uno primero, para luego continuar con los elementos del lenguaje más complejos.

En el siguiente ejemplo se utiliza la instrucción CREATE TABLE para crear la tabla TITULOS_CD (mostrada en la figura 4-5) y definir una restricción de columna:

```
CREATE TABLE TITULOS_CD
( ID_TITULO_CD INT,
  TITULO_CD VARCHAR(60),
  ID_EDITOR INT REFERENCES EDITORES_CD );
```

Esta instrucción define la restricción FOREIGN KEY en la columna ID_EDITOR. Observe que, con el fin de añadir una restricción de columna, todo lo que se tiene que hacer es agregar la palabra clave REFERENCES y el nombre de la tabla referenciada. También note que la clave foránea contiene el mismo número de columnas que la clave primaria en la tabla referenciada, y las columnas referenciadas y de referencia son del mismo tipo de datos. Recuerde: si no se hace referencia de la clave primaria en la tabla referenciada, se debe también incluir el nombre de la columna o columnas (por ejemplo, REFERENCES EDITORES_CD (ID_EDITOR)).

NOTA

Antes de poder crear una clave foránea en una tabla, la tabla referenciada ya debe existir y se debe definir una restricción UNIQUE o PRIMARY KEY para esa tabla.

En el siguiente ejemplo se crea una clave foránea, que es una restricción de tabla. A diferencia del ejemplo anterior, se incluye el nombre de una columna referenciada en esta definición de restricción, aun cuando no es necesario:

```
CREATE TABLE TITULOS_CD
( ID_TITULO_CD INT,
  TITULO_CD VARCHAR(60) ,
  ID_EDITOR INT,
  CONSTRAINT FK_ID_EDITOR FOREIGN KEY (ID_EDITOR)
    REFERENCES EDITORES_CD (ID_EDITOR) );
```

Las últimas dos líneas del código son la definición de restricción. El nombre de la restricción, `FK_ID_EDITOR`, viene después de la palabra clave `CONSTRAINT`. Los nombres de las restricciones no son necesarios porque el RDBMS asigna un nombre generado por el sistema si no se proporciona uno. Sin embargo, es una buena práctica suministrar uno propio, ya que los nombres de las restricciones a menudo muestran mensajes de error cuando una instrucción SQL intenta violar una restricción, y los nombres suministrados serán fáciles de reconocer de aquellos proporcionados por el DBMS. Después del nombre de la restricción, las palabras clave `FOREIGN KEY` indican el tipo de restricción, que es seguido por el nombre de la columna de referencia, `ID_EDITOR`. Éste es el nombre de la columna en la que la restricción tendrá lugar. Si hubiera varios nombres de columnas, deben separarse por comas. El nombre de la columna de referencia es entonces seguido por la palabra clave `REFERENCES`, que es seguido por el nombre de la tabla referenciada, `EDITORES_CD`. El nombre de la columna referenciada viene después del nombre de la tabla referenciada.

Eso es todo lo que hay que hacer. Una vez que la restricción se define, no podrá poner valores en la columna `ID_EDITOR` de la tabla `TITULOS_CD` a menos que esos valores ya existan en la clave primaria de la tabla `EDITORES_CD`. Debe observar, sin embargo, que los valores en la clave foránea no tienen que ser únicos, como deben ser en la clave primaria `EDITORES_CD`. Los valores en la clave foránea pueden repetirse cualquier número de veces, a menos que se restrinja la columna por una restricción única.

Antes de pasar a discutir los otros elementos de la sintaxis de `FOREIGN KEY`, echemos un vistazo rápido a la clave foránea que incluye varias columnas. En la figura 4-6 hay dos tablas: `ARTISTAS_INTERPRETES` y `TIPOS_MUSICA_ARTISTAS`.

La clave primaria en la tabla `ARTISTAS_INTERPRETES` se define en las columnas `NOMBRE_ARTISTA` y `FDN_ARTISTA`. La siguiente instrucción de SQL crea la tabla `TIPOS_MUSICA_ARTISTAS`, que incluye una clave foránea formada por las columnas `NOMBRE_ARTISTA` y `FDN`:

```
CREATE TABLE TIPOS_MUSICA_ARTISTAS
( NOMBRE_ARTISTA VARCHAR(60) ,
  FDN DATE,
  ID_TIPO INT,
  CONSTRAINT FK_ARTISTAS_CD FOREIGN KEY ( NOMBRE_ARTISTA, FDN )
    REFERENCES ARTISTAS_INTERPRETES (NOMBRE_ARTISTA, FDN_ARTISTA) );
```



Figura 4-6 Una clave foránea formada por varias columnas.

En esta instrucción hay dos columnas de referencia (NOMBRE_ARTISTA y FDN) y dos columnas referenciadas (NOMBRE_ARTISTA y FDN_ARTISTA). Las columnas NOMBRE_ARTISTA en las dos tablas tienen el mismo tipo de datos, y la columna FDN tiene el mismo tipo de datos que la columna FDN_ARTISTA. Como se puede observar, una de las columnas de referencia (FDN) tiene un nombre diferente que su contraparte de referencia (FDN_ARTISTA).

Pregunta al experto

P: En la figura 4-6 y en los ejemplos anteriores, se creó una restricción FOREIGN KEY en las columnas NOMBRE_ARTISTA y FDN en la tabla TIPOS_MUSICA_ARTISTAS. ¿Cuál podría ser la clave primaria para esta tabla?

R: Recuerde que una clave primaria debe identificar de manera exclusiva cada fila en una tabla. Sin embargo, ya que los pares de valores en las columnas NOMBRE_ARTISTA y FDN se pueden repetir (lo que significa que también pueden repetirse en las columnas individuales), esas dos columnas no se pueden utilizar como una clave primaria para esta tabla. Por otro lado, la columna ID_TIPO también puede repetir valores; por lo tanto, esa columna no se puede utilizar. Además, probablemente no desee combinar la columna ID_TIPO con una de las

(continúa)

otras dos columnas porque es posible que se repitan filas (por ejemplo, dos artistas con el mismo nombre interpretando el mismo tipo de música, tal como los dos músicos de blues llamados Sonny Boy Williamson, o dos artistas con la misma fecha de nacimiento interpretando el mismo tipo de música). Como resultado, la mejor solución (además de añadir otra columna a la tabla) es mover las tres columnas a una clave primaria. Juntas, las tres columnas identifican de forma exclusiva cada fila, ya que es poco probable que alguien comparta el mismo nombre, fecha de nacimiento y tipo de música (aunque cualquier cosa es posible, razón por la cual, en última instancia, añadir otra columna que garantice que los datos son únicos es la mejor manera de hacerse).

La cláusula MATCH

Ahora que tiene una comprensión de cómo definir una restricción básica de clave foránea, echemos un vistazo de otra línea de sintaxis de FOREIGN KEY:

```
[ MATCH { FULL | PARTIAL | SIMPLE } ]
```

Se puede decir por los paréntesis que se trata de una cláusula opcional. Y de hecho, muy pocos proveedores de productos respaldan actualmente esta cláusula (no se respalda por SQL Server 2005, Oracle 11g o MySQL 5.0, por ejemplo), por lo que verá que no se utiliza mucho. Sin embargo, se describe en el estándar SQL, lo que significa que se espera que más proveedores de productos la respalden en el futuro. Su propósito es permitirle decidir cómo tratar los valores nulos en las columnas de clave foránea con respecto a los valores que permiten que se inserten en las columnas de referencia. Si las columnas no permiten valores nulos, entonces la cláusula MATCH no se aplica. Se tienen tres opciones que se pueden utilizar en la cláusula MATCH:

- Si se especifica MATCH FULL, todas las columnas de referencia deben tener un valor nulo o ninguna de esas columnas puede tener un valor nulo.
- Si se especifica MATCH PARTIAL y una o más columnas de referencia pueden tener valores nulos siempre y cuando el resto de las columnas de referencia tengan valores que igualen a las columnas de referencia correspondientes.
- Si se especifica MATCH SIMPLE y una o más columnas de referencia tienen valores nulos, entonces el resto de las columnas de referencia pueden tener valores que no están contenidos en las columnas de referencia correspondientes. La opción SIMPLE está implícita si no se incluye la cláusula MATCH en la definición de la restricción de FOREIGN KEY.

La mejor manera de ilustrar cada una de esas opciones MATCH es mediante ejemplos de datos válidos e inválidos que pueden insertarse en las columnas de referencia. Volviendo al ejemplo mostrado en la figura 4-6, se puede observar que una clave foránea en la tabla TIPOS_MUSICA_ARTISTAS se forma por dos columnas de referencia: NOMBRE_ARTISTA y FDN. La tabla 4-1 proporciona ejemplos de datos que se pueden o no insertar en las columnas de clave foránea. Los ejemplos se basan en los datos de las columnas de clave primaria de la tabla ARTISTAS_INTERPRETES.

Opción MATCH	Ejemplos de datos válidos	Ejemplos de datos inválidos
FULL	Joni Mitchell, 1943-11-07 NULL, NULL	NULL, 1943-11-07 Joni Mitchell, NULL Joni Mitchell, 1802-08-03
PARTIAL	Patsy Cline, 1932-09-08 NULL, 1932-09-08 Patsy Cline, NULL NULL, NULL	NULL, 1802-08-03 Henryk Górecki, NULL Patsy Cline, 1947-03-03
SIMPLE	Bing Crosby, 1904-05-02 NULL, 1904-05-02 Bing Crosby, NULL NULL, 1802-08-03 Henryk Górecki, NULL NULL, NULL	Bing Crosby, 1802-08-03 Bing Crosby, 1947-03-03 Henryk Górecki, 1947-03-03

Tabla 4-1 Ejemplos válidos e inválidos de las opciones de una cláusula MATCH.

NOTA

Probablemente no desea permitir que haya valores nulos en las columnas de referencia en la tabla TIPOS_MUSICA_ARTISTAS, particularmente en la columna NOMBRE_ARTISTA. Y si cualquiera de esas columnas se utiliza en la clave primaria, no sería posible permitir valores nulos. Sin embargo, con el fin de demostrar cómo trabajan las opciones MATCH, asumamos que se permiten los valores nulos.

Si decide utilizar una cláusula MATCH, simplemente añádala al final de la definición de la restricción FOREIGN KEY, como se muestra en la siguiente instrucción de SQL (asumiendo que la aplicación de SQL la respalda):

```
CREATE TABLE TIPOS_MUSICA_ARTISTAS
( NOMBRE_ARTISTA VARCHAR(60),
  FDN DATE,
  ID_TIPO INT,
  CONSTRAINT FK_ARTISTAS_CD FOREIGN KEY ( NOMBRE_ARTISTA, FDN )
    REFERENCES ARTISTAS_INTERPRETES MATCH FULL );
```

Para insertar datos en las columnas de referencia (NOMBRE_ARTISTA y FDN), ambos valores tienen que ser nulos o deben ser valores de datos válidos de las columnas referenciadas en la tabla ARTISTAS_INTERPRETES.

La cláusula <acción referencial desencadenada>

La cláusula final en la sintaxis de la restricción FOREIGN KEY es la cláusula opcional <acción referencial desencadenada>. La cláusula permite definir qué tipos de acciones se deben tomar cuando se intenta actualizar o eliminar datos desde columnas referenciadas (si ese intento causa una violación de los datos en las columnas de referencia). Por ejemplo, suponga que trata de eliminar datos de una tabla con clave primaria. Si esa clave primaria hace referencia a una clave foránea

y si los datos a eliminar se almacenan en una clave foránea, entonces eliminar los datos desde una clave primaria causaría una violación a la restricción FOREIGN KEY. Los datos en las columnas de referencia siempre deben incluirse en las columnas referenciadas.

El punto que debe recordarse acerca de la cláusula <acción referencial desencadenada> es que se incluye en la definición de la tabla de referencia (a través de una clave foránea) una acción que se debe tomar como resultado de algo que se realiza a la tabla referenciada. Esto se puede aclarar echando un vistazo a la sintaxis para la cláusula de <acción referencial desencadenada>:

```
ON UPDATE <acción referencial> [ ON DELETE <acción referencial> ]
| ON DELETE <acción referencial> [ ON UPDATE <acción referencial> ]
<acción referencial>::=
CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION
```

NOTA

El símbolo ::= (dos puntos consecutivos y un signo de igual) se utiliza en el estándar SQL:2006 para separar a un marcador de posición de las llaves de la definición. En la sintaxis anterior se define el marcador de posición <acción referencial>. El marcador de posición se utiliza en el código que precede a la definición. Se tomará entonces la definición (las cinco palabras clave) y se usarán en lugar del marcador de posición <acción referencial> como se utiliza en las cláusulas ON UPDATE y ON DELETE.

Como se observa de la sintaxis, se puede definir una cláusula ON UPDATE, y una cláusula ON DELETE, o ambas, y se pueden definir en cualquier orden. Para cada una de estas cláusulas se puede escoger una de cinco acciones referenciales:

- Si se utiliza CASCADE y los datos se actualizan o eliminan en las columnas referenciadas, los datos en las columnas de referencia se actualizan o eliminan.
- Si se utiliza SET NULL y los datos se actualizan o eliminan en las columnas referenciadas, los valores en las columnas de referencia correspondientes se establecen como nulos. Los valores nulos tienen que respaldarse en las columnas de referencia para que esta opción funcione.
- Si se utiliza SET DEFAULT y los datos se actualizan o eliminan en las columnas referenciadas, los valores en las columnas de referencia correspondientes se establecen a sus valores predeterminados. Se deben asignar valores predeterminados a las columnas de referencia para que esta opción funcione.
- Si se utiliza RESTRICT y se tratan de actualizar o eliminar datos en las columnas referenciadas que causan una violación de clave foránea, se previene la ejecución de esta acción. Los datos en las columnas de referencia nunca pueden violar la restricción FOREIGN KEY, ni siquiera temporalmente.
- Si se utiliza NO ACTION y se tratan de actualizar o eliminar datos en las columnas referenciadas que causan una violación de clave foránea, se previene la ejecución de esta acción. Sin embargo, las violaciones de datos pueden ocurrir temporalmente bajo ciertas condiciones durante la ejecución de una instrucción SQL, pero los datos en la clave foránea nunca se violan en su estado final (al final de la ejecución). La opción NO ACTION es predeterminada y se utiliza tanto para actualizar como para eliminar, si no se especifica una acción referencial desencadenada.

Si decide utilizar la cláusula <acción referencial desencadenada>, simplemente añádala al final de la definición de la restricción de FOREIGN KEY, como se muestra en la siguiente instrucción SQL:

```
CREATE TABLE TIPOS_MUSICA_ARTISTAS
( NOMBRE_ARTISTA VARCHAR(60),
  FDN DATE,
  ID_TIPO INT,
  CONSTRAINT FK_ARTISTAS_CD FOREIGN KEY ( NOMBRE_ARTISTA, FDN )
    REFERENCES ARTISTAS_INTERPRETES ON UPDATE CASCADE ON DELETE CASCADE );
```

Si actualiza o elimina datos de las columnas referenciadas en ARTISTAS_INTERPRETES, esos cambios se harán en las columnas de referencia en la tabla TIPOS_MUSICA_ARTISTAS.

Pruebe esto 4-1

Añada restricciones NOT NULL, únicas y referenciales

En el capítulo 3, Pruebe esto 3-1 y 3-2, se crearon varias tablas que se agregaron a la base de datos INVENTARIO (o el esquema CD_INVENTARIO). En este ejercicio se añadirán una serie de restricciones a las tablas y se crearán nuevas tablas que también se definirán con restricciones. Sin embargo, en lugar de utilizar la instrucción ALTER TABLE para modificar las tablas que ya se crearon, se volverán a crear esas tablas. La ventaja de esto es que podrá observar la definición de la tabla por completo en lo que se refiere a la actualización del modelo de datos, mostrado en la figura 4-7.

El modelo de datos incorpora un par de elementos más, que se han visto antes. Incorpora tablas, columnas dentro de esas tablas, tipos de datos para esas columnas, restricciones y relaciones entre tablas. Ya debe estar familiarizado con la forma en que se representan las tablas, columnas y tipos de datos, así que echemos un vistazo a las restricciones y relaciones:

- Las columnas incluidas en la clave primaria están en la sección superior de la tabla, y las otras columnas se encuentran en la sección inferior. Por ejemplo, en la tabla DISCOS_COMPACTOS, la columna ID_DISCO_COMPACTO es la clave primaria. En algunos casos, como en la tabla TIPOS_DISCOS_COMPACTOS, todas las columnas se incluyen en la clave primaria.
- Cada clave foránea se representa por [FK].
- Las predeterminaciones, las restricciones UNIQUE y las restricciones NOT NULL se identifican con cada columna aplicable.
- Las relaciones, tal como las claves foráneas las definen, se representan por líneas que conectan la clave foránea en una tabla a la clave de candidato (suele ser la clave primaria) en otra tabla.

Encontrará útil este modelo de datos no sólo en este ejercicio, sino para otros ejercicios en el libro, los cuales continúan aprovechando o utilizando la base de datos INVENTARIO. Puede también descargar el archivo Try_This_04.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

(continúa)

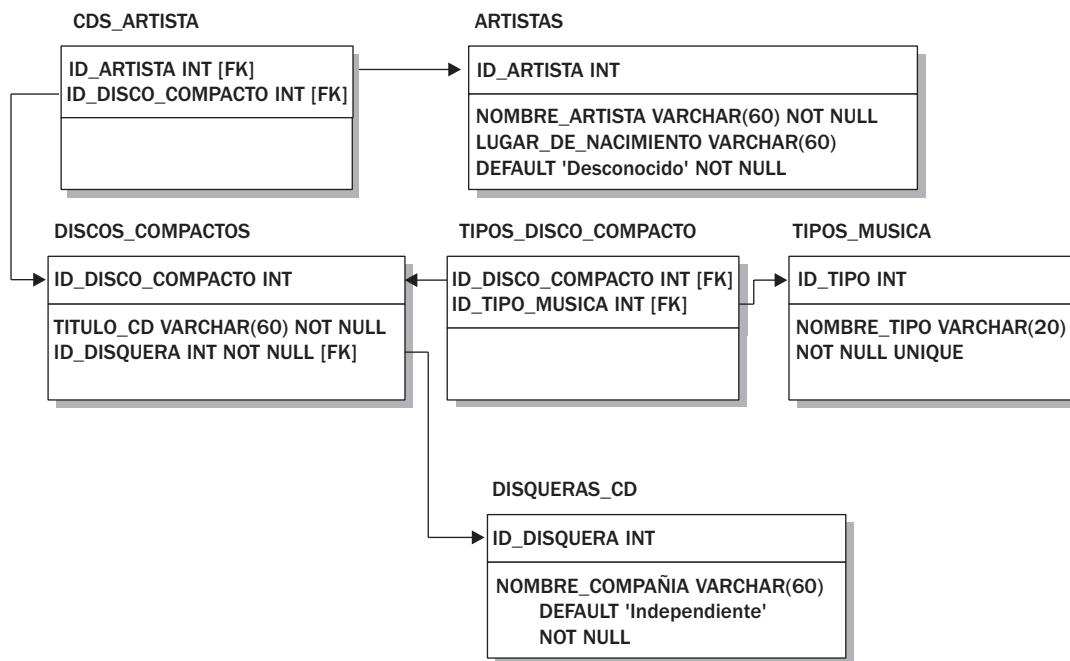


Figura 4-7 Modelo de datos para la base de datos INVENTARIO.

NOTA

Los modelos de datos son muy variados. El modelo que aquí se utiliza es específico para las necesidades del libro. Encontrará en el mundo real que los modelos son diferentes de lo que se ve aquí. Por ejemplo, las relaciones entre tablas se representan de manera diferente, y la información de la definición de columna podría no ser tan extensa.

Paso a paso

1. Abra la aplicación de clientes de su RDBMS y conecte la base de datos INVENTARIO.
2. Primero necesita eliminar las cuatro tablas (DISCOS_COMPACTOS, TIPOS_DISCO_COMPACTO, TIPOS_MUSICA y DISQUERAS_CD) que ya se crearon. Introduzca y ejecute las siguientes instrucciones:

```

DROP TABLE DISCOS_COMPACTOS      CASCADE;
DROP TABLE TIPOS_DISCO_COMPACTO  CASCADE;
DROP TABLE TIPOS_MUSICA          CASCADE;
DROP TABLE DISQUERAS_CD          CASCADE;
  
```

NOTA

Si creó la tabla ARTISTAS o la tabla CD_ARTISTA cuando probó con ejemplos o experimentó con las instrucciones CREATE TABLE, asegúrese de eliminar ésas también.

NOTA

SQL Server no soporta la opción CASCADE, y en Oracle se debe escribir como CASCADE CONSTRAINTS.

Ahora puede volver a crear estas tablas y crear otras nuevas. Debe crear las tablas en el orden indicado en este ejercicio, ya que las tablas referidas en las claves foráneas tienen que existir (con las claves primarias creadas) antes de que pueda crear las claves foráneas. Asegúrese de hacer referencia al modelo de datos de la figura 4-7 para obtener más detalles acerca de cada una de las tablas que se crean.

3. La primera tabla que se crea es la tabla TIPOS_MUSICA. Contiene dos columnas: ID_TIPO y NOMBRE_TIPO. Se configura la columna ID_TIPO como la clave primaria, y se configura la restricción UNIQUE y la restricción NOT NULL en la columna NOMBRE_TIPO. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE TIPOS_MUSICA
( ID_TIPO          INT,
  NOMBRE_TIPO      VARCHAR(20) NOT NULL,
  CONSTRAINT UN_NOMBRE_TIPO UNIQUE (NOMBRE_TIPO),
  CONSTRAINT PK_TIPOS_MUSICA PRIMARY KEY (ID_TIPO) );
```

4. La siguiente tabla que se crea es la tabla DISQUERAS_CD. La tabla incluye la columna ID_DISQUERA, que se define como la clave primaria, y la columna NOMBRE_COMPañIA, que se define con un valor predeterminado y una restricción NOT NULL. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE DISQUERAS_CD
( ID_DISQUERA      INT,
  NOMBRE_COMPañIA  VARCHAR(60) DEFAULT 'Independiente' NOT NULL,
  CONSTRAINT PK_DISQUERAS_CD PRIMARY KEY (ID_DISQUERA) );
```

5. Ahora que se creó la tabla DISQUERAS_CD, se puede crear la tabla DISCOS_COMPACTOS. La tabla DISCOS_COMPACTOS contiene una clave foránea que hace referencia a la tabla DISQUERAS_CD. Ésta es la razón por la que se creó primero la tabla DISQUERAS_CD. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE DISCOS_COMPACTOS
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) NOT NULL,
  ID_DISQUERA        INT NOT NULL,
  CONSTRAINT PK_DISCOS_COMPACTOS PRIMARY KEY (ID_DISCO_COMPACTO),
  CONSTRAINT FK_ID_DISQUERA FOREIGN KEY (ID_DISQUERA) REFERENCES
DISQUERAS_CD );
```

(continúa)

6. La siguiente tabla, TIPOS_DISCO_COMPACTO, incluye dos claves foráneas, junto con su clave primaria. Las claves foráneas hacen referencia a las tablas DISCOS_COMPACTO y TIPOS_MUSICA, las cuales ya se habían creado. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  ID_TIPO_MUSICA     INT,
  CONSTRAINT PK_TIPOS_DISCO_COMPACTO
    PRIMARY KEY ( ID_DISCO_COMPACTO, ID_TIPO_MUSICA),
  CONSTRAINT FK_ID_DISCO_COMPACTO_01
    FOREIGN KEY (ID_DISCO_COMPACTO) REFERENCES DISCOS_COMPACTOS,
  CONSTRAINT FK_ID_TIPO_MUSICA
    FOREIGN KEY (ID_TIPO_MUSICA) REFERENCES TIPOS_MUSICA );
```

7. Ahora ya se puede crear la tabla ARTISTAS. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE ARTISTAS
( ID_ARTISTA INT,
  NOMBRE_ARTISTA VARCHAR(60) NOT NULL,
  LUGAR_DE_NACIMIENTO VARCHAR(60) DEFAULT 'Desconocido' NOT NULL,
  CONSTRAINT PK_ARTISTAS PRIMARY KEY (ID_ARTISTA) );
```

8. La última tabla que se crea (al menos por ahora) es la tabla CDS_ARTISTA. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE TABLE CDS_ARTISTA
( ID_ARTISTA INT,
  ID_DISCO_COMPACTO INT,
  CONSTRAINT PK_CDS_ARTISTA PRIMARY KEY ( ID_ARTISTA, ID_DISCO_
    COMPACTO ),
  CONSTRAINT FK_ID_ARTISTA FOREIGN KEY (ID_ARTISTA) REFERENCES
    ARTISTAS,
  CONSTRAINT FK_ID_DISCO_COMPACTO_02 FOREIGN KEY (ID_DISCO_COMPACTO)
    REFERENCES DISCOS_COMPACTOS );
```

9. Cierre la aplicación de cliente.

Resumen de Pruebe esto

La base de datos cuenta ahora con seis tablas, cada una configurada con los valores predeterminados y las restricciones necesarias. En este ejercicio, se seguirá un orden específico para crear las tablas con el fin de aplicar más fácilmente las claves foráneas. Sin embargo, se pudieron haber creado las tablas en cualquier orden, sin las claves foráneas (a menos que las tablas referenciadas ya se hayan creado) y luego agregarse en la clave foránea, pero esto agrega pasos extra. De hecho, se hubieran podido modificar las tablas que existían antes de este ejercicio (en lugar de eliminarlas y después volver a crearlas), siempre y cuando se crearan las claves primarias (o restricciones UNIQUE) en las tablas referenciadas antes de crear las claves foráneas en las tablas de referencia. Independientemente del enfoque que tome, el resultado final debe ser que ahora la base de datos cuenta con las tablas necesarias para empezar a pasar a otros componentes de SQL.

Defina restricciones CHECK

Anteriormente en este capítulo, en la sección “Entienda las restricciones de integridad”, se analizaron varias categorías de restricciones y los tipos de restricciones que se soportan. (Remítase a la figura 4-1 para una descripción general de esas categorías.) Un tipo de restricción (la restricción CHECK) se puede definir como restricciones de tabla, restricciones de columna, restricciones de dominio, o en afirmaciones. Una restricción CHECK permite especificar qué valores se pueden incluir en una columna. Se puede definir un rango de valores (por ejemplo, entre 10 y 100), una lista de valores (por ejemplo, blues, jazz, pop, country), o una serie de otras condiciones que restringen exactamente qué valores se permiten en una columna.

Las restricciones CHECK son las más flexibles de todas las restricciones y suelen ser las más complicadas. A pesar de esto, la sintaxis básica para una restricción CHECK es relativamente sencilla. Para crear una restricción CHECK de columna, utilice la siguiente sintaxis en la definición de columna:

```
<nombre de columna> { <tipo de datos> | <dominio> } CHECK ( <condición de búsqueda> )
```

Para crear una restricción CHECK de tabla, utilice la siguiente sintaxis en la definición de la tabla:

```
[ CONSTRAINT <nombre de restricción> ] CHECK ( <condición de búsqueda> )
```

Se analizarán las restricciones de dominio y afirmaciones después en esta sección.

Como se puede observar por la sintaxis, una restricción CHECK es relativamente sencilla. Sin embargo, los valores utilizados para la cláusula <condición de búsqueda> pueden ser muy amplios y, por consiguiente, bastante complejos. El concepto principal es que la <condición de búsqueda> se pruebe (se puede decir “compruebe”) para cualquier instrucción SQL que intente modificar los datos en una columna cubierta por la restricción de CHECK, y si se evalúa como TRUE, la instrucción SQL se completa; si se evalúa como FALSE, la instrucción SQL se suspende y despliega un mensaje de error. La mejor manera de aprender acerca de la cláusula es examinar los ejemplos. Sin embargo, la mayoría de los componentes de la <condición de búsqueda> se basan en el uso de predicados con el fin de crear la condición de búsqueda. Un *predicado* es una expresión que opera en valores. Por ejemplo, un predicado se puede utilizar para comparar valores (por ejemplo, COLUMNA_1 > 10). El símbolo mayor que (>) es un predicado de comparación, a menudo denominado como *operador de comparación*. En este caso, el predicado verifica que cualquier valor insertado en la COLUMNA_1 sea mayor que 10.

Muchos de los componentes de la <condición de búsqueda> también se basan en el uso de subconsultas. Una *subconsulta* es una expresión que se utiliza como un componente dentro de otra expresión. Las subconsultas se usan cuando una expresión debe acceder o calcular varias capas de datos, tal como tener que buscar una segunda tabla para proporcionar datos para una primera tabla.

Tanto los predicados como las subconsultas son temas bastante complicados que van más allá del alcance de un análisis acerca de las restricciones CHECK, y de hecho, cada uno de los temas se tratan por separado en su propio capítulo. (Vea el capítulo 9 para información acerca de predicados y el capítulo 12 para información acerca de subconsultas.) A pesar del hecho de que ambos temas se analizan después en el libro, quiero proporcionar al menos unos ejemplos de las restricciones CHECK para dar una idea de cómo se implementan en el entorno SQL.

El primer ejemplo que se verá es una restricción CHECK que define valores mínimos y máximos que se pueden insertar en una columna. La siguiente definición de tabla en este ejemplo crea

tres columnas y una restricción CHECK (como una restricción de tabla) que restringe los valores de una de las columnas a una serie de números entre 0 y 30:

```
CREATE TABLE TITULOS_CD
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) NOT NULL,
  EN_EXISTENCIA      INT          NOT NULL,
  CONSTRAINT CK_EN_EXISTENCIA CHECK ( EN_EXISTENCIA > 0 AND EN_
EXISTENCIA < 30 ) );
```

Si se trata de introducir un valor en la columna EN_EXISTENCIA distinto de 1 a 29, se recibe un error. Se pueden lograr los mismos resultados definiendo una restricción de columna:

```
CREATE TABLE TITULOS_CD
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) NOT NULL,
  EN_EXISTENCIA      INT          NOT NULL
  CHECK ( EN_EXISTENCIA > 0 AND EN_EXISTENCIA < 30 ) );
```

Echemos un vistazo más de cerca a la cláusula <condición de búsqueda> en la instrucción, que en este caso es (EN_EXISTENCIA > 0 AND EN_EXISTENCIA < 30). La cláusula primero nos dice que cualquier valor introducido en la columna EN_EXISTENCIA debe ser mayor que 0 (EN_EXISTENCIA > 0). La palabra clave AND nos dice que las condiciones definidas en ambos lados de AND deben aplicarse. Finalmente, la cláusula nos dice que el valor debe ser menor que 30 (EN_EXISTENCIA < 30). Debido a que se usa la palabra clave AND, el valor debe ser mayor que 0 y menor que 30.

Otra manera en la que se puede utilizar una restricción CHECK es explícitamente listar los valores que se pueden introducir en la columna. Ésta es una opción práctica si se tiene un número limitado de valores y no son susceptibles de cambio (o cambian con poca frecuencia). La siguiente instrucción SQL crea una tabla que incluye una restricción CHECK que define a qué década pertenece la música:

```
CREATE TABLE TITULOS_CD
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) NOT NULL,
  ERA                CHAR(5),
  CONSTRAINT CK_ERA CHECK ( ERA IN ( '1940s', '1950s',
'1960s', '1970s', '1980s', '1990s', '2000s' ) ) );
```

El valor introducido en la columna ERA debe ser una de las siete décadas representadas en la condición de búsqueda. Si se trata de introducir un valor que no sea un valor nulo o uno de estos siete, se recibirá un error. Observe que se utiliza el operador IN para designar que los valores en la columna ERA deben ser uno del conjunto de valores delimitados entre paréntesis después de la palabra clave IN.

Si el número del paréntesis empieza a confundirlo, se puede separar el código en líneas que siguen a la incrustación de esos paréntesis. Por ejemplo, la instrucción anterior se puede escribir como sigue:

```
CREATE TABLE TITULOS_CD
(
```

```

ID_DISCO_COMPACTO  INT,
TITULO_CD          VARCHAR(60)  NOT NULL,
ERA                CHAR(5),
CONSTRAINT CK_ERA  CHECK
(
    ERA IN
    (
        '1940s', '1950s', '1960s', '1970s', '1980s', '1990s', '2000s'
    )
)
);

```

Cada conjunto de paréntesis y su contenido se escriben a un nivel que corresponde al nivel de incrustación para esa cláusula en particular, al igual que un esquema. Utilizando este método nos dice exactamente qué cláusulas se incluyen en qué conjunto de paréntesis, y la instrucción se ejecuta de la misma forma como si no se hubiera separado las líneas. El inconveniente es que ocupa mucho espacio (que es la razón por la que no se utiliza este método en el libro), aunque podría ser una herramienta útil para esas instrucciones que son un poco más complejas.

Ahora veamos otro ejemplo de una restricción CHECK. Este ejemplo es similar al primero que examinamos, sólo que éste se refiere a valores entre determinados números:

```

CREATE TABLE TITULOS_CD
( ID_DISCO_COMPACTO  INT,
  TITULO_CD          VARCHAR(60)  NOT NULL,
  EN_EXISTENCIA      INT          NOT NULL,
  CONSTRAINT CK_EN_EXISTENCIA CHECK
    ( ( EN_EXISTENCIA BETWEEN 0 AND 30 ) OR
      ( EN_EXISTENCIA BETWEEN 49 AND 60 ) ) ) ;

```

En esta instrucción se utiliza el operador BETWEEN para especificar un rango que *incluye* los criterios de valoración. Ya que se crean dos diferentes rangos, se encierra la especificación de cada rango en paréntesis: (EN_EXISTENCIA BETWEEN 0 AND 30) y (EN_EXISTENCIA BETWEEN 49 AND 60). Estas dos especificaciones de rango se conectan por la palabra clave OR, lo que indica que una o la otra condición debe cumplirse. Como resultado, cualquier valor introducido en la columna EN_EXISTENCIA debe ser de 0 a 30 o de 49 a 60.

Como se mencionó antes, se aprenderá más acerca de condiciones de búsqueda en el capítulo 9. En ese momento se verá cuán flexible es la restricción CHECK. Y cuando se utiliza con subconsultas (vea capítulo 12), proporciona una herramienta poderosa para definir explícitamente qué valores se permiten en una columna en particular.

Defina afirmaciones

Una afirmación es simplemente un tipo de restricción CHECK que se puede aplicar a varias tablas. Por esta razón, una afirmación se crea por separado de la definición de una tabla. Desafortunadamente, la mayoría de los proveedores de productos, incluyendo Oracle 11g, SQL Server 2005 y MySQL 5.0, aún no respaldan las afirmaciones. Para crear una afirmación, utilice la siguiente sintaxis:

```
CREATE ASSERTION <nombre de la restricción> CHECK <condiciones de búsqueda>
```

Crear una afirmación es muy similar a crear una restricción CHECK. Después de la palabra clave CHECK, se debe proporcionar la(s) condición(es) de búsqueda necesaria(s). Ahora veamos un ejemplo. Suponga que la tabla TITULOS_CD incluye una columna para el número de discos compactos en existencia. Se desea que el total para esa tabla sea siempre inferior que el máximo de inventario que desea llevar. En el siguiente ejemplo, se crea una afirmación que totalice los valores en la columna EN_EXISTENCIA y verifique que el total sea menor que 5 000:

```
CREATE ASSERTION LIMITE_EN_EXISTENCIA CHECK
  ( ( SELECT SUM (EN_EXISTENCIA) FROM TITULOS_CD ) < 5000 );
```

En esta instrucción se utiliza la subconsulta (SELECT SUM (EN_EXISTENCIA) FROM TITULOS_CD) y se compara con 5 000. La subconsulta comienza con la palabra clave SELECT, que se usa para consultar datos en una tabla. La función SUM suma los valores en la columna EN_EXISTENCIA, y la palabra clave FROM especifica en qué tabla se encuentra la columna. Los resultados de esta subconsulta se comparan (utilizando el operador de comparación menor que) a 5 000. Si se intenta agregar un valor en la columna EN_EXISTENCIA que cause que el total sea superior a 5 000, se recibirá un error.

Creación de dominios y restricciones de dominio

El último tipo de restricción CHECK es del tipo que se inserta en una definición de dominio. En su mayor parte, la definición de restricción es similar a lo que ha visto antes, excepto que ésta no se vincula a una columna o tabla específica. De hecho, las restricciones de dominio utilizan la palabra clave VALUE cuando se refieren al valor en una columna definida con esa restricción en particular. Echemos un vistazo a la sintaxis para crear un dominio:

```
CREATE DOMAIN <nombre del dominio> [AS] <tipo de datos>
[DEFAULT <valor predeterminado>]
[CONSTRAINT <nombre de la restricción>] CHECK (<condición de búsqueda>)
```

Ya debe estar familiarizado con la mayoría de los elementos en esta sintaxis. Se discutieron los tipos de datos y las cláusulas predeterminadas en el capítulo 3, y la definición de la restricción es similar a lo que se ha visto hasta ahora en este capítulo.

En el siguiente ejemplo se crea un dominio que se basa en el tipo de datos INT y que requiere que todos los valores sean entre 0 y 30:

```
CREATE DOMAIN CANTIDAD_EN_EXISTENCIA AS INT
  CONSTRAINT CK_CANTIDAD_EN_EXISTENCIA CHECK (VALUE BETWEEN 0 AND 30);
```

El único elemento realmente nuevo aquí (distinto de la cláusula CREATE DOMAIN) es la palabra clave VALUE, la cual, como se dijo, se refiere al valor de la columna definida con el dominio CANTIDAD_EN_EXISTENCIA. Como resultado, si se trata de insertar un valor (en una de esas columnas) que no sea entre 0 y 30, se recibirá un error.

Pruebe esto 4-2 Añada una restricción CHECK

En este ejercicio, que es relativamente corto, se utiliza la instrucción `ALTER TABLE` para modificar la tabla `DISCOS_COMPACTOS`. Se añade una columna a la tabla y luego se define una restricción `CHECK` que restrinja los valores que se pueden introducir en la columna. La columna adicional y la restricción no tienen un impacto en las otras tablas de la base de datos `INVENTARIO` o en la relación entre las tablas. Puede descargar el archivo `Try_This_04.txt` (en inglés), que contiene las instrucciones `SQL` que se usan en este ejercicio.

Paso a paso

1. Abra la aplicación de clientes de su RDBMS y conéctese a la base de datos `INVENTARIO`.
2. Se modifica la tabla `DISCOS_COMPACTOS` añadiendo la columna `EN_EXISTENCIA`. Introduzca y ejecute la siguiente instrucción `SQL`:

```
ALTER TABLE DISCOS_COMPACTOS  
ADD COLUMN EN_EXISTENCIA INT NOT NULL;
```

NOTA

Para Oracle y SQL Server, omita la palabra clave `COLUMN`.

3. Ahora que la columna existe, se puede agregar una restricción `CHECK` a la definición de la tabla. Se puede introducir la restricción como una restricción de columna, pero añadiéndola por separado como una restricción de tabla le permitirá hacer cada paso por separado para que pueda ver los resultados de sus acciones. La restricción `CHECK` limita los valores que se pueden introducir en la columna `EN_EXISTENCIA`. Cada valor debe ser mayor que 0, pero menor que 50. Introduzca y ejecute la siguiente instrucción `SQL`:

```
ALTER TABLE DISCOS_COMPACTOS  
ADD CONSTRAINT CK_EN_EXISTENCIA CHECK ( EN_EXISTENCIA > 0 AND  
EN_EXISTENCIA < 50 );
```

4. Cierre la aplicación de cliente.

Resumen de Pruebe esto

La nueva columna, `EN_EXISTENCIA`, sigue el número de cada disco compacto que figura en la tabla `DISCOS_COMPACTOS`. La restricción `CK_EN_EXISTENCIA` restringe el número por fila a una cantidad entre 0 y 50. Ahora que la tabla se ha actualizado, no se puede añadir ningún valor que viole la restricción.

✓ Autoexamen Capítulo 4

1. ¿Cuáles son las tres categorías de las restricciones de integridad?
2. ¿Cuáles son las diferencias entre una restricción de columna y una restricción de tabla?
3. ¿Qué tipos de restricciones se pueden incluir en una definición de columna?
4. ¿Cuál es la diferencia entre una restricción de tabla y una afirmación?
5. ¿Qué significa un valor nulo?
6. ¿Cuál de los siguientes tipos de restricciones soporta restricciones NOT NULL?
 - A Restricciones de tabla
 - B Restricciones de columna
 - C Restricciones de dominio
 - D Afirmaciones
7. Se crea una tabla que incluye una columna que acepta valores nulos pero cuyos valores no nulos deben ser únicos. ¿Qué tipo de restricción se debe utilizar?
8. Se crea una tabla que incluye la columna NOMBRE_TIPO. La columna se define con el tipo de datos CHAR(10) y requiere una restricción UNIQUE, que se define como una restricción de columna. ¿Qué código SQL se debe utilizar para la definición de columna?
9. ¿Cuáles dos restricciones se aplican a las restricciones PRIMARY KEY pero no aplican a las restricciones UNIQUE?
10. Se crea una restricción de PRIMARY KEY llamada PK_TIPOS_MUSICA_ARTISTA en la tabla TIPOS_MUSICA_ARTISTA. La clave primaria incluye las columnas NOMBRE_ARTISTA y FDN_ARTISTA. ¿Qué código SQL se debe utilizar para la restricción de la tabla?
11. ¿Cómo difiere una restricción referencial de una restricción única?
12. Una restricción _____ impone la integridad referencial entre dos tablas garantizando que no se lleve a cabo ninguna acción en ninguna tabla que pueda afectar a los datos protegidos por la restricción.
13. Se crea una tabla que incluye la columna llamada ID_TIPO_NEGOCIO, con un tipo de datos INT. La columna se define con una restricción FOREIGN KEY que hace referencia a la clave primaria en la tabla llamada TIPOS_NEGOCIO. La clave foránea se añade como una restricción de columna. ¿Qué código SQL se debe usar para la definición de columna?
14. ¿Cuáles tres opciones se pueden utilizar en la cláusula MATCH de una restricción FOREIGN KEY?
15. ¿Cuáles son los dos tipos de acciones referenciales desencadenadas que se pueden definir en una restricción FOREIGN KEY?

- 16.** Se crea una restricción FOREIGN KEY y se desea que los valores en la columna de referencia se actualicen si los valores en la columna referenciada se actualizan. ¿Qué cláusula <acción referencial desencadenada> utilizaría?
- A** ON UPDATE RESTRICT
 - B** ON UPDATE NO ACTION
 - C** ON UPDATE CASCADE
 - D** ON UPDATE SET DEFAULT
- 17.** ¿Qué sintaxis debe utilizar para una restricción CHECK que se define como una restricción de tabla?
- 18.** ¿Qué tipo de restricciones se pueden definir dentro en una afirmación?
- 19.** Se crea una restricción CHECK en la columna NUMERO_EN_EXISTENCIA. Se desea limitar los valores que se pueden introducir en la columna en un rango de 11 a 29. ¿Qué debe utilizar para la cláusula <condición de búsqueda> de la restricción?

Capítulo 5

Creación de vistas en SQL



Habilidades y conceptos clave

- Añada vistas a la base de datos
 - Creación de vistas actualizables
 - Eliminación de vistas de la base de datos
-

Como aprendió en el capítulo 3, las tablas base persistentes almacenan los datos de SQL en su base de datos. Sin embargo, esas tablas no son siempre una forma útil si sólo desea ver datos específicos de una tabla o datos de varias tablas. Por esta razón, el estándar SQL:2006 respalda el uso de tablas vistas, o vistas. Una *vista* es una tabla virtual cuya definición existe como un objeto de esquema. A diferencia de las tablas base persistentes, en la vista no hay datos almacenados. De hecho, las tablas vistas en realidad no existen (sólo existe la definición que las define). Esta definición es la que permite seleccionar información específica de una o más tablas, basada en las instrucciones de consulta en esa definición. Una vez que se crea una vista, simplemente se invoca llamándola por su nombre en una consulta como en una tabla base. Los datos entonces se presentan como si se buscaran en una tabla base.

Añada vistas a la base de datos

Antes de entrar demasiado profundo en las características específicas de las vistas, quiero revisar rápidamente algunos puntos que se discutieron en los capítulos 2 y 3. Una vista, como posiblemente recuerde, es uno de los tres tipos de tablas respaldadas por SQL, junto con las tablas base y las tablas derivadas. La mayoría de las tablas base son objetos de esquema y vienen en cuatro tipos: tablas base persistentes, tablas temporales globales, tablas temporales locales creadas y tablas temporales locales declaradas. De estos cuatro tipos, las tablas base persistentes son las que manejan los datos reales de SQL. Las tablas derivadas, por otro lado, son simplemente el resultado que se observa cuando se consultan datos de una base de datos. Por ejemplo, si se solicitan datos de la tabla DISCOS_COMPACTOS, los resultados de la solicitud se despliegan en formato parecido a una tabla, conocida como tabla derivada.

En algunos aspectos, una vista se encuentra entre una tabla base persistente y una tabla derivada. Es como una tabla base persistente, ya que la definición de la vista se almacena como un objeto de esquema utilizando un nombre único (dentro del esquema) que se puede acceder como en una tabla base. Sin embargo, una vista es como una tabla derivada, ya que ningún dato se almacena en asociación con la vista. Tanto las tablas derivadas como las vistas son tipos de tablas virtuales. Los datos se seleccionan de una o más tablas base cuando se invoca la vista. De hecho, se puede pensar en una vista como simplemente una tabla derivada nombrada, con la definición de la vista almacenada en el esquema. Los resultados de los datos que se ven cuando se llama a una vista se almacenan en cualquier lugar, pero se derivan de tablas base existentes.

Las vistas pueden ser herramientas útiles cuando se accede a los diferentes tipos de datos. Una de las principales ventajas de utilizar las vistas es que se pueden definir consultas complejas

y almacenarlas dentro de la definición de la vista. En lugar de volver a crear las consultas cada vez que se necesiten, puede simplemente invocar la vista. Además, las vistas pueden ser una manera práctica de presentar información a los usuarios sin suministrar más información que la que necesitan o información que no deben ver. Por ejemplo, puede ser que desee que los usuarios de su organización puedan acceder al registro de determinados empleados, pero puede que no quiera que la información tal como los números de seguro social o los sueldos estén disponibles para esos usuarios, así que se puede crear una vista que proporcione sólo la información que los usuarios deben ver. Las vistas también se pueden utilizar para sintetizar estructuras complejas y presentar información de una manera que sea más fácil de comprender para algunos usuarios, que en efecto oculta la estructura fundamental y la complejidad de la base de datos de los usuarios.

Ahora que tiene una descripción general de qué son las vistas, veamos algunos ejemplos que ilustren cómo se extraen los datos de tablas base en el tipo de tabla derivada que se presenta por la definición de la vista. El primer ejemplo que veremos, mostrado en la figura 5-1, se basa en la tabla `INVENTARIO_DISCO_COMPACTO`, que incluye seis columnas. Suponga que desea ver sólo las columnas `TITULO_CD`, `DERECHOSDEAUTOR` y `EN_EXISTENCIA`. Se puede crear una vista que extraiga estas tres columnas de la tabla y las organice como si los datos existieran en su propia tabla, como se muestra en la figura 5-1. La vista `DISCOS_COMPACTOS_EN_EXISTENCIA` contiene una consulta que define exactamente qué datos debe devolver la vista.

Se puede observar que los nombres de las columnas en la vista son diferentes de los nombres de las columnas de la tabla `INVENTARIO_DISCO_COMPACTO`, aun cuando los datos dentro de las columnas son los mismos. Esto se debe a que, si se desea, se puede asignar que los nombres de las columnas de las vistas sean diferentes a los de la tabla original. Lo mismo ocurre con los tipos de datos. Las columnas de la vista heredan los tipos de datos a partir de sus respectivas columnas de la tabla. Por ejemplo, la columna `DISCO_COMPACTO` en la vista `DISCOS_COMPACTOS_EN_EXISTENCIA` heredó el tipo de datos `VARCHAR(60)` de la columna `TITULO_CD` de la tabla `INVENTARIO_DISCO_COMPACTO`. No se especificó el tipo de datos `VARCHAR(60)` en ningún lugar dentro de la definición de la vista.

Como se puede observar, una vista permite definir qué columnas se devuelven cuando se invoca la vista. La definición para la vista `DISCOS_COMPACTOS_EN_EXISTENCIA` especifica tres columnas; sin embargo, se pudo haber especificado cualquiera de las columnas de la tabla `INVENTARIO_DISCO_COMPACTO`. Además de las columnas, la definición de una vista puede especificar qué filas se devuelven. Por ejemplo, la figura 5-2 muestra la vista `CDS_EN_EXISTENCIA_1990S`. Observe que contiene las mismas columnas que la vista `DISCOS_COMPACTOS_EN_EXISTENCIA` (mostrada en la figura 5-1), pero tiene menos filas. En este caso, la definición de la vista no sólo especifica las mismas tres columnas de la tabla `INVENTARIO_DISCO_COMPACTO`, sino que también especifica que sólo las filas con valores entre 1990 y 1999 (inclusive) en la columna `DERECHOSDEAUTOR` se devuelven.

En los dos ejemplos anteriores se examinaron las vistas que se derivan de datos de sólo una tabla; sin embargo, se pueden crear vistas basadas en varias tablas. Esto es particularmente útil si se quiere desplegar información relacionada que abarque más de una tabla. Echemos un vistazo a la figura 5-3, que incluye las tablas `INVENTARIO_CD` y `DISQUERAS`. La tabla `INVENTARIO_CD` contiene un listado de los CD en el inventario, y la tabla `DISQUERAS` contiene un listado de las compañías que editan CD. Se dará aquí una breve introducción del acceso a múltiples tablas, ya que es una gran manera de demostrar qué tan bien pueden las vistas ocultar la complejidad de una consulta. Este tema se trata en detalle en el capítulo 11.

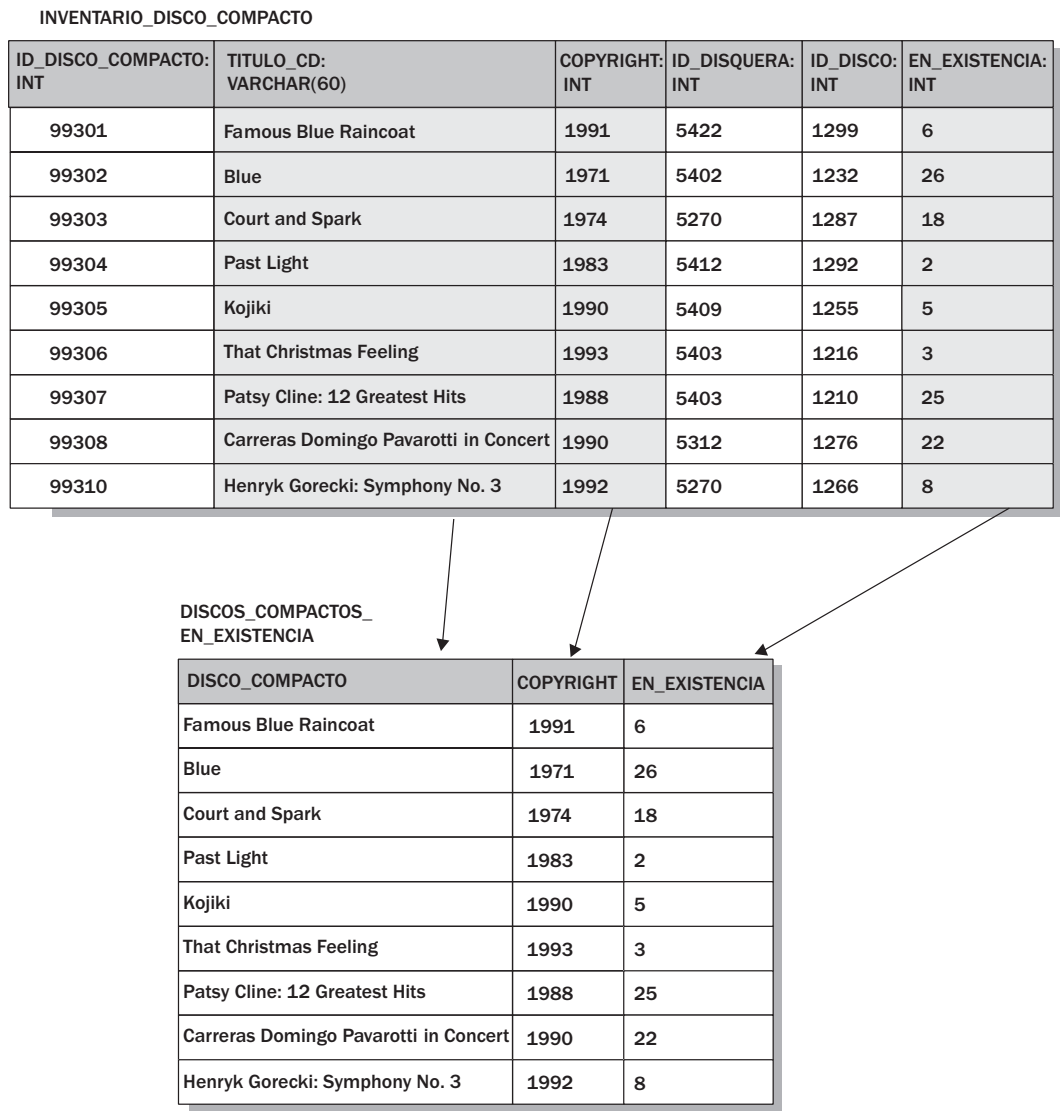


Figura 5-1 La vista DISCOS_COMPACTOS_EN_EXISTENCIA, basada en la tabla INVENTARIO_DISCO_COMPACTO.

Suponga que tiene usuarios que desean poder ver los nombres del CD y del editor, pero que no les interesan los valores ID_DISCO_COMPACTO e ID_DISQUERA. Y ciertamente no les interesa tener que ver en dos lugares diferentes para comparar los valores ID_DISQUERA con el fin de igualar los CD con los nombres de la compañía. Una solución es crear una vista que iguale esta información, mientras que al mismo tiempo se despliegue sólo la información que les sea útil.

CDS_EN_EXISTENCIA_1990S

DISCO_COMPACTO	COPYRIGHT	EN_EXISTENCIA
Famous Blue Raincoat	1991	6
Kojiki	1990	5
That Christmas Feeling	1993	3
Carreras Domingo Pavarotti in Concert	1990	22
Henryk Gorecki: Symphony No. 3	1992	8

Figura 5-2 La vista CDS_EN_EXISTENCIA_1990S, basada en la tabla INVENTARIO_DISCO_COMPACTO.

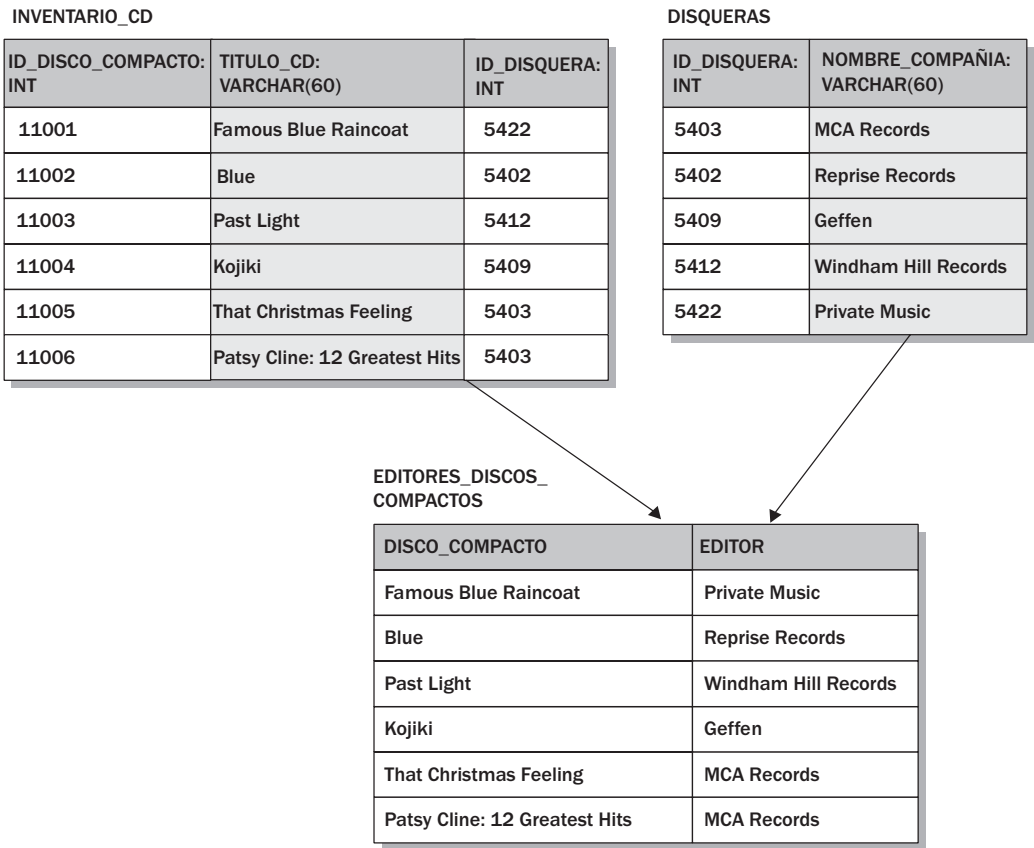


Figura 5-3 La vista EDITORES_DISCOS_COMPACTOS, basada en las tablas DISQUERAS e INVENTARIO_CD.

En el caso de las tablas INVENTARIO_CD y DISQUERAS, se puede crear una vista (llamada EDITORES_DISCO_COMPACTO en la figura 5-3) que vincule (*una*) los datos para los usuarios, al tiempo que oculte la estructura fundamental y datos extraños.

Una vista de este tipo es posible que tome ventaja de las relaciones entre tablas. En el caso de las tablas INVENTARIO_CD y DISQUERAS, una clave foránea se definió en la columna ID_DISQUERA de la tabla INVENTARIO_CD que hace referencia a la columna ID_DISQUERA de la tabla DISQUERAS. La consulta contenida en la definición de la vista EDITORES_DISCO_COMPACTO coincide con los valores en la columna ID_DISQUERA de la tabla INVENTARIO_CD con los valores en la columna ID_DISQUERA de la tabla DISQUERAS. Para cada coincidencia que se encuentra, se devuelve una fila. Por ejemplo, la fila Famous Blue Raincoat incluye el valor ID_DISQUERA de 5422. En la tabla DISQUERAS se puede ver que ese valor coincide con la fila Private Music. Como resultado, la vista contiene una fila con los valores Famous Blue Raincoat y Private Music.

NOTA

No necesariamente se tiene que utilizar una relación de clave foránea para unir las tablas. Se pueden utilizar dos columnas de diferentes tablas que almacenen la misma información. Esto podría significar utilizar todas las columnas en una clave foránea (si la clave foránea incluye múltiples columnas), usando sólo una de las columnas, o sin utilizar una clave foránea. Se analizará la unión de múltiples tablas en el capítulo 11.

Además de unir información de tablas diferentes, también se pueden utilizar las vistas para modificar los datos que se extraen de la columna de una tabla y se presentan en una columna de vista. Esto permite tomar acciones como la realización de cálculos, encontrar promedios, determinar valores mínimos y máximos, y completar un sinnúmero de otras operaciones. Entonces se pueden tomar los resultados de estas operaciones e incluirlas en una columna dentro de una vista. En la figura 5-4, por ejemplo, la vista DESCUENTOS_CD deduce un 10 por ciento de descuento del precio al menudeo y presenta el resultado en la columna PRECIO_DESCUENTO.

La vista DESCUENTOS_CD incluye tres columnas. La columna DISCO_COMPACTO extrae los datos directamente de la columna TITULO_CD. Las columnas PRECIO_MENUDEO y PRECIO_DESCUENTO en la vista extraen los datos de la columna PRECIO_MENUDEO en la tabla INVENTARIO. La columna PRECIO_MENUDEO en la vista copia los valores justo como son. Sin embargo, para la columna PRECIO_DESCUENTO, los valores extraídos de la columna PRECIO_MENUDEO en la tabla INVENTARIO se multiplican por 0.9.

Como puede observar, se pueden especificar varios tipos de operaciones en una vista y luego simplemente invocar la vista cuando necesite la información. La mayor parte de lo que puede incluirse en una consulta se puede incluir en una vista. De hecho, es la consulta, o la expresión de consulta, la que constituye el núcleo de la vista. Sin embargo, antes de examinar las expresiones de consulta, deseo en primer lugar analizar la sintaxis utilizada para la creación de las vistas.

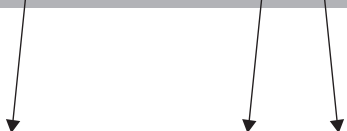
Definición de vistas de SQL

El tipo más simple de vista a crear es uno que hace referencia sólo a una tabla y recupera los datos de columnas dentro de una tabla sin modificar esos datos. Entre más complicada es la vista, más

INVENTARIO

ID_DISCO_COMPACTO: INT	TITULO_CD: VARCHAR(60)	COPYRIGHT: INT	PRECIO_MENUEO: NUMÉRICO(5,2)	EN_EXISTENCIA: INT
99301	Famous Blue Raincoat	1991	16.99	6
99302	Blue	1971	14.99	26
99303	Court and Spark	1974	14.99	18
99304	Past Light	1983	15.99	2
99305	Kojiki	1990	15.99	5
99306	That Christmas Feeling	1993	10.99	3
99307	Patsy Cline: 12 Greatest Hits	1988	16.99	25

DESCUENTOS_CD



DISCO_COMPACTO	PRECIO_MENUEO	PRECIO_DESCUENTO
Famous Blue Raincoat	16.99	15.29
Blue	14.99	13.49
Court and Spark	14.99	13.49
Past Light	15.99	14.39
Kojiki	15.99	14.39
That Christmas Feeling	10.99	9.89
Patsy Cline: 12 Greatest Hits	16.99	15.29

Figura 5-4 La vista DESCUENTOS_CD, basada en la tabla INVENTARIO.

complicada es la expresión de consulta en la que se basa la vista. En su forma más básica, la sintaxis para una vista es la siguiente:

```
CREATE VIEW <nombre de la vista> [(<nombres de las columnas de la vista>)]
AS <expresión de consulta>
[ WITH CHECK OPTION ]
```

Por ahora nos centraremos sólo en las primeras dos líneas de la sintaxis y se dejará WITH CHECK OPTION para después, en la sección “Creación de vistas actualizables”. Como se mues-

tra en la primera línea de la sintaxis, se debe proporcionar un nombre para la vista. Además, se deben proporcionar los nombres para las columnas en las siguientes circunstancias:

- Si alguno de los valores de columna se basa en algún tipo de operación que calcule el valor que se insertará en la columna, en lugar de que el valor se copie directamente de la tabla. (Ver la figura 5-4.)
- Si los nombres de las columnas de la tabla se duplican, lo cual puede suceder cuando las tablas se unen.

Incluso si no se requiere proporcionar los nombres de las columnas, se pueden proporcionar si así lo desea. Por ejemplo, puede encontrar que desea cambiar los nombres para que sean más lógicos para usuarios en particular. Sin embargo, si proporciona los nombres de las columnas utilizando la sintaxis <nombres de las columnas de la vista>, se deben proporcionar los nombres para todas las columnas.

NOTA

También hay una forma alternativa de proporcionar los nombres de las columnas utilizando la palabra clave AS dentro de la expresión de consulta, que se analizará en el capítulo 7.

La segunda línea de la sintaxis incluye la palabra clave AS, que es requerida, y el marcador de posición <expresión de consulta>. El marcador de posición <expresión de consulta>, aunque parece sencillo, puede implicar una estructura compleja de instrucciones de consulta que pueden realizar una serie de operaciones, incluyendo la recuperación de datos de varias tablas, el cálculo de datos, limitar el tipo de datos devueltos, y la realización virtual de cualquier otro tipo de operación respaldada por una expresión de consulta. Debido a la complejidad de las expresiones de consulta, pasaré la mayoría de la segunda parte de este libro discutiendo diversas formas de consulta de datos. Lo que implica, entonces, que será muy difícil resumir un debate a fondo de las expresiones de consulta dentro del tema de las vistas. A pesar de todo, quiero proporcionar una serie de ejemplos que ilustren cómo se pueden crear vistas que realicen varias funciones. Con cada ejemplo, se incluirá una breve explicación de la expresión de consulta utilizada en la definición de la vista. Esté al tanto, sin embargo, que se tratarán los detalles de las expresiones de consulta con mayor profundidad, más adelante en el libro, empezando en el capítulo 7. También observe que las tablas base utilizadas en estos ejemplos no se han creado en los ejercicios anteriores; por lo tanto, si quiere probar los ejemplos, se tendrán que crear primero las tablas base. Las figuras muestran la información que necesita para hacer esto.

El primer ejemplo que se considera se basa en la vista mostrada en la figura 5-1. La vista DISCOS_COMPACTOS_EN_EXISTENCIA obtiene los datos de la tabla INVENTARIO_DISCO_COMPACTO e incluye tres columnas de esa tabla. Para crear la vista, utilice la siguiente instrucción CREATE VIEW:

```
CREATE VIEW DISCOS_COMPACTOS_EN_EXISTENCIA
( DISCO_COMPACTO, DERECHOSDEAUTOR, EN_EXISTENCIA ) AS
  SELECT TITULO_CD, DERECHOSDEAUTOR, EN_EXISTENCIA
  FROM INVENTARIO_DISCO_COMPACTO;
```

Esta vista es la más simple de todos los tipos de vistas a crear. Se basa en una tabla y extrae tres de las seis columnas de esa tabla. Tenga en cuenta que mientras SQL normalmente requiere

cláusulas para estar en un orden particular, no hay restricciones en cuanto a espacios y saltos de línea. Por ejemplo, cuando se crean las vistas, se prefiere poner la lista de nombres de columna (cuando se presente) en una línea nueva y colocar la palabra clave AS al final de la línea que precede la expresión de consulta. Otros prefieren colocar la palabra clave AS en una línea, y aun otros prefieren colocarla al comienzo de la primera línea de la expresión de consulta. El RDBMS no tomará en cuenta la forma en que lo hace, pero la adopción de un estilo y el adherirse a él hará que SQL sea más fácil de leer, entender y mantener.

Analizando la instrucción un poco, la primera línea proporciona un nombre para la vista, DISCOS_COMPACTOS_EN_EXISTENCIA. La segunda línea proporciona un nombre para cada una de las tres columnas: DISCO_COMPACTO, DERECHOSDEAUTOR y EN_EXISTENCIA, y termina con la palabra clave AS. Si se omitieran los nombres de las columnas, las columnas de la vista tomarían los nombres de las columnas de la tabla. La tercera y cuarta líneas de la instrucción CREATE VIEW contienen la expresión de consulta, que en este caso es la siguiente instrucción SELECT:

```
SELECT DISCO_COMPACTO, DERECHOSDEAUTOR, EN_EXISTENCIA
FROM INVENTARIO_DISCO_COMPACTO
```

La instrucción SELECT es una de las instrucciones más comunes (si no *la* instrucción más común) que utilizará como programador de SQL. También es una de las instrucciones más extensas y flexibles que usará, permitiendo formar consultas intrincadas que pueden devolver exactamente el tipo de datos que se desee recuperar de la base de datos.

La instrucción SELECT utilizada en la definición de la vista DISCOS_COMPACTOS_EN_EXISTENCIA es la instrucción SELECT en su forma más básica. La instrucción se divide en dos cláusulas: la cláusula SELECT y la cláusula FROM. La cláusula SELECT identifica qué columnas se devuelven (TITULO_CD, DERECHOSDEAUTOR y EN_EXISTENCIA), y la cláusula FROM identifica la tabla de la que se extraen los datos (INVENTARIO_DISCO_COMPACTO). Cuando se invoca la vista DISCOS_COMPACTOS_EN_EXISTENCIA, se está esencialmente invocando la instrucción SELECT que se incrusta en la definición de la vista, que a su vez toma los datos de la(s) tabla(s) base aplicable(s).

En el siguiente ejemplo, basado en la vista de la figura 5-2, la instrucción CREATE VIEW es casi la misma que la del ejemplo anterior, excepto que una cláusula adicional se agrega a la instrucción:

```
CREATE VIEW CDS_EN_EXISTENCIA_1990S
( DISCO_COMPACTO, DERECHOSDEAUTOR, EN_EXISTENCIA ) AS
SELECT TITULO_CD, DERECHOSDEAUTOR, EN_EXISTENCIA
FROM INVENTARIO_DISCO_COMPACTO
WHERE DERECHOSDEAUTOR > 1989 Y DERECHOSDEAUTOR < 2000;
```

La cláusula WHERE define una condición que debe cumplirse para que los datos sean devueltos. Como en el ejemplo anterior, se extraen los datos de las columnas TITULO_CD, DERECHOSDEAUTOR y EN_EXISTENCIA de la tabla INVENTARIO_DISCO_COMPACTO, sólo que esta vez se limitan los datos a las filas cuyos valores DERECHOSDEAUTOR son mayores que 1989 pero menores que 2000 (DERECHOSDEAUTOR > 1989 Y DERECHOSDEAUTOR < 2000). Puede que reconozca los operadores de comparación mayor que (>) y menor que (<) del capítulo 4 del análisis acerca de las restricciones CHECK. Se utilizan para limitar los valores que se incluyen en la vista.

NOTA

Los operadores utilizados en la cláusula WHERE (o cualquier condición definida en la cláusula) no tienen efecto en los datos almacenados en las tablas base. Sólo afectan a los datos devueltos cuando se invoca la vista. Se analizarán estos tipos de operadores a mayor detalle en el capítulo 9.

Se puede utilizar la cláusula WHERE en la instrucción SELECT para definir una amplia variedad de condiciones. Por ejemplo, la cláusula WHERE se puede usar para ayudar a unir las tablas, como se muestra en la siguiente instrucción CREATE VIEW:

```
CREATE VIEW EDITORES_DISCO_COMPACTO
( DISCO_COMPACTO, EDITOR ) AS
SELECT INVENTARIO_CD.TITULO_CD, DISQUERAS.NOMBRE_COMPANÍA
FROM INVENTARIO_CD, DISQUERAS
WHERE INVENTARIO_CD.ID_DISQUERA = DISQUERAS.ID_DISQUERA;
```

Esta instrucción crea la vista que se observa en la figura 5-3. El nombre de la vista es EDITORES_DISCO_COMPACTO e incluye las columnas DISCO_COMPACTO y EDITOR. La vista extrae información de dos fuentes: la columna TITULO_CD en la tabla INVENTARIO_CD y la columna NOMBRE_COMPANÍA en la tabla DISQUERAS.

Primero echemos un vistazo a la cláusula SELECT. Observe que el nombre de cada columna se califica por el nombre de la tabla respectiva (por ejemplo, INVENTARIO_CD.TITULO_CD). Cuando se unen dos o más tablas, se deben calificar los nombres de las columnas si hay alguna posibilidad de que los nombres de las columnas se puedan confundir, que sería el caso si se incluyen columnas con el mismo nombre. Sin embargo, si no hay posibilidad de que los nombres de las columnas se puedan confundir, entonces se pueden omitir los nombres de la tabla. Por ejemplo, la cláusula SELECT puede leerse como se muestra a continuación:

```
SELECT TITULO_CD, NOMBRE_COMPANÍA
```

A pesar de que los nombres calificados no siempre son necesarios, muchos programadores prefieren utilizarlos en todos los casos, ya que es más fácil saber a qué tabla se hace referencia si alguna vez necesita modificar la estructura de la base de datos o la definición de la vista en un momento posterior.

La siguiente cláusula en la instrucción SELECT es la cláusula FROM. Cuando se unen las tablas, se deben incluir los nombres de todas las tablas participantes, separadas por comas. Además de la cuestión de los diversos nombres, la cláusula FROM es similar a lo que se ha visto en otros ejemplos.

La cláusula WHERE, que es la cláusula final en la instrucción SELECT, es la que une las filas. La cláusula WHERE es necesaria, ya que sin ella no habría forma de saber cómo igualar los valores de las diferentes tablas. La cláusula WHERE especifica cómo hacer esto. En la definición de la vista EDITORES_DISCO_COMPACTO, el valor en la columna ID_DISQUERA de la tabla INVENTARIO_CD debe ser igual al valor en la columna ID_DISQUERA de la tabla DISQUERAS para que una fila se devuelva. Por ejemplo, si se refiere de nuevo a la figura 5-3, observará que la fila Past Light en la tabla INVENTARIO_CD tiene un valor de 5412 en la columna ID_DISQUERA, que corresponde a la fila Windham Hill Records en la tabla DISQUERAS. Observe que, una vez más, los nombres de las columnas se califican por los nombres de la tabla, que es esencial en este caso, ya que las columnas comparten el mismo nombre. Sin los nombres de la tabla, SQL no sabría si se comparan valores por sí mismos o con la otra tabla.

También se puede expandir la cláusula WHERE para mayor calidad de consulta. En el siguiente ejemplo, la cláusula WHERE limita las filas devueltas a sólo aquellas que contienen el valor de 5403 en la columna ID_DISQUERA de la tabla INVENTARIO_CD:

```
CREATE VIEW EDITORES_DISCO_COMPACTO
( DISCO_COMPACTO, EDITOR ) AS
  SELECT INVENTARIO_CD.TITULO_CD, DISQUERAS.NOMBRE_COMPANIA
    FROM INVENTARIO_CD, DISQUERAS
   WHERE INVENTARIO_CD.ID_DISQUERA = DISQUERAS.ID_DISQUERA
        AND INVENTARIO_CD.ID_DISQUERA = 5403;
```

Entonces si se invocara la vista EDITORES_DISCO_COMPACTO, se verían sólo los CD que se producen por MCA Records.

Ahora veamos otro ejemplo, que se basa en la vista de la figura 5-4. Del mismo modo que los primeros dos ejemplos que se consideraron, esta vista obtiene los datos de sólo una tabla. Sin embargo, esta vista, de hecho, realiza cálculos que devuelven datos que fueron modificados. La instrucción CREATE VIEW tiene este aspecto:

```
CREATE VIEW DESCUENTOS_CD
( DISCO_COMPACTO, PRECIO_MENUDEO, PRECIO_DESCUENTO ) AS
  SELECT TITULO_CD, PRECIO_MENUDEO, PRECIO_MENUDEO * 0.9
    FROM INVENTARIO;
```

La vista DESCUENTOS_CD incluye tres columnas: DISCO_COMPACTO, PRECIO_MENUDEO y PRECIO_DESCUENTO. La columna PRECIO_DESCUENTO contiene los valores calculados. La cláusula SELECT identifica las columnas de la tabla que contienen los datos fuente. Las dos primeras columnas se definen de la misma manera como se vio en los ejemplos anteriores. Los datos se copian de las columnas TITULO_CD y PRECIO_MENUDEO en la tabla INVENTARIO a las columnas DISCO_COMPACTO y PRECIO_MENUDEO de la vista DESCUENTOS_CD. Sin embargo, la definición de la tercera columna ($\text{PRECIO_MENUDEO} * 0.9$) es un poco diferente. Los valores se toman nuevamente de la columna PRECIO_MENUDEO, sólo que esta vez los valores se multiplican por 0.9 (90 por ciento) para llegar al precio de descuento que aparece en la columna PRECIO_DESCUENTO de la vista.

También se puede añadir la cláusula WHERE a la instrucción SELECT utilizada en la definición de la vista DESCUENTOS_CD:

```
CREATE VIEW DESCUENTOS_CD
( DISCO_COMPACTO, PRECIO_MENUDEO, PRECIO_DESCUENTO ) AS
  SELECT TITULO_CD, PRECIO_MENUDEO, PRECIO_MENUDEO * 0.9
    FROM INVENTARIO
   WHERE EN_EXISTENCIA > 10;
```

La cláusula WHERE restringe la consulta a sólo aquellas filas cuyo valor EN_EXISTENCIA es mayor que 10. Observe que puede utilizar un operador de comparación en una columna de tabla (EN_EXISTENCIA) cuyos valores no son ni siquiera los devueltos por la vista.

Como se puede observar de todos estos ejemplos de definiciones de vista, hay muchas cosas que se pueden hacer con las vistas como resultado de la flexibilidad y extensibilidad de la instrucción SELECT.

Más adelante en el libro, cuando se familiarice más con los distintos tipos de instrucciones `SELECT` que puede crear y las operaciones que puede realizar, será capaz de crear vistas que son mucho más complejas que las que se han considerado hasta ahora.

Creación de vistas actualizables

En SQL, algunos tipos de vistas son actualizables. En otras palabras, se puede utilizar una vista para modificar los datos (cambiar los datos existentes y/o insertar nuevas filas) en la tabla fundamental. Si una vista es actualizable depende de la instrucción `SELECT` que se define dentro de la definición de la vista. Normalmente, entre más compleja es la instrucción, es menos probable que la vista sea actualizable. No hay una sintaxis dentro de la instrucción `CREATE VIEW` que designe explícitamente a una vista como actualizable. En cambio, se determina estrictamente por la naturaleza de la instrucción `SELECT`, que se debe adherir a los estándares específicos para que la vista sea actualizable.

Hasta este punto en el capítulo, se dijo que el marcador de posición <expresión de consulta> en la sintaxis `CREATE VIEW` está compuesto de la instrucción `SELECT`. Para ser más preciso, una expresión de consulta puede ser uno de los varios tipos de expresiones. La más común de éstas, y a la que se refiere este libro, es la especificación de consulta. Una *especificación de consulta* es una expresión de SQL que comienza con la palabra clave `SELECT` e incluye una serie de elementos que forman esa expresión, como se puede observar en los ejemplos de vistas que se examinan. Una especificación de consulta es actualizable si cumple con las numerosas directrices que se exponen en el estándar SQL:2006. En aras de la simplicidad, me refiero a la especificación de consulta como la instrucción `SELECT`, que a menudo es la forma en que se hace referencia en los distintos tipos relacionados con SQL y la documentación relacionada con productos.

Aparte de la cuestión de las especificaciones de consulta y la complejidad de los estándares SQL, el punto que se intenta tratar es que las normas de sintaxis que determinan la actualización de una vista no son simples, directrices claras, en particular teniendo en cuenta el hecho de que aún se tiene que cubrir la instrucción `SELECT` en profundidad (que se hará al comienzo del capítulo 7). Sin embargo, hay bases lógicas que pueden deducirse de estas directrices:

- Los datos sin una vista no pueden ser resumidos, agrupados o eliminados automáticamente.
- Por lo menos una columna de la tabla fuente debe ser actualizable.
- Cada columna en la vista debe ser trazable exactamente a una columna fuente en una tabla.
- Cada fila en la vista debe ser trazable exactamente a una fila fuente en una tabla. Sin embargo, note que muchos proveedores de productos permiten modificaciones (pero no insertan) a las vistas creadas a partir de varias tablas, siempre y cuando la actualización sólo haga referencia a columnas que tracen a sólo una tabla base.

En muchos casos se podrá determinar la actualización de una vista simplemente aplicando sentido común. Veamos un ejemplo. Suponga que decide agregar información acerca de los empleados a la base de datos, ya que quiere rastrear las comisiones ganadas de la venta de CD. Decide crear la tabla `COMISIONES_EMPLEADO`, mostrada en la figura 5-5, que enumera la cantidad total de comisiones por cada empleado hecha durante un periodo de 3 años.

COMISIONES_EMPLEADO

ID_EMPLEADO: INT	AÑO_1999: NUMERICO(7,2)	AÑO_2000: NUMERICO(7,2)	AÑO_2001: NUMERICO(7,2)
99301	126.32	11.28	16.86
99302	16.27	90.20	198.14
99303	354.34	16.32	1237.56
99304	112.11	87.56	14.14

Figura 5-5 Ingresos anuales por comisión en la tabla COMISIONES_EMPLEADO.

Ahora suponga que quiere saber el promedio de la comisión por cada año para todos los empleados. Se puede crear una vista que determine el promedio por cada año y muestre esos promedios en tres columnas separadas. Para hacerlo se utiliza la siguiente instrucción CREATE VIEW:

```
CREATE VIEW COM_EMP ( PROM_1999, PROM_2000, PROM_2001 ) AS
  SELECT PROM(AÑO_1999), PROM(AÑO_2000), PROM(AÑO_2001)
  FROM COMISIONES_EMPLEADO;
```

Como se puede ver de la instrucción, la vista COM_EMP contiene tres columnas: PROM_1999, PROM_2000 y PROM_2001. La cláusula SELECT extrae la información de las tres columnas en la tabla COMISIONES_EMPLEADO (AÑO_1999, AÑO_2000 y AÑO_2001) y utiliza la función PROM para encontrar el promedio para todos los valores en cada columna, como se muestra en la figura 5-6. Por ejemplo, la función PROM primero promedia los cuatro valores en la columna AÑO_1999 y luego introduce ese promedio en la columna PROM_1999 de la vista COM_EMP.

Ahora suponga que desea actualizar los importes de la comisión en la tabla COMISIONES_EMPLEADO. No se podría hacer a través de la vista, ya que los valores en la vista se basan en cálculos realizados sobre los valores en la tabla. Por ejemplo, si se trata de actualizar los valores en la columna PROM_1999, los RDBMS no sabrían cómo se afectan muchas de las filas o cómo se distribuyen los valores dentro de esas filas. En otras palabras, la fila en la vista no se puede rastrear de nuevo a exactamente la fila fuente.

COM_EMP

PROM_1999	PROM_2000	PROM_2001
152.26	51.34	366.68

Figura 5-6 La vista COM_EMP, basada en el promedio trimestral de ingresos.

COM_EMP

ID_EMPLEADO	AÑO_1999	AÑO_2000
99301	126.32	11.28
99302	16.27	90.20
99303	354.34	16.32
99304	112.11	87.56

Figura 5-7 Vista COM_EMP, basada en el primero y segundo trimestres de ingresos.

Se puede, sin embargo, crear una vista que simplemente extraiga información de la tabla COMISIONES_EMPLEADO:

```
CREATE VIEW COM_EMP AS
SELECT ID_EMPLEADO, AÑO_1999, AÑO_2000
FROM COMISIONES_EMPLEADO;
```

En esta instrucción se crea una vista que muestra sólo tres de las cuatro columnas de la tabla. No se realizan cálculos y sólo se utiliza una tabla. La figura 5-7 muestra cómo se ve la vista.

Esta vista, a diferencia de la última, es actualizable. Se pueden modificar e insertar datos, ya que éstos no se resumieron o agruparon; cada columna se puede rastrear exactamente a una columna fuente en una tabla; cada fila se rastrea exactamente a una fila fuente en una tabla. Además, ningún dato se resume o se agrupa. Por supuesto, si se fueran a actualizar o insertar datos mediante la vista, son los datos en la tabla fundamental los que en realidad se modifican. Eso significa que cualquier modificación de datos debe aún adherirse a las limitaciones que tienen lugar en la tabla. Por ejemplo, no se podría insertar una fila a través de la vista COM_EMP si los valores nulos no fueran permitidos en la columna AÑO_2001 de la tabla. La vista no tendría la capacidad de aceptar un valor para esa columna, y la tabla no permitiría insertar una fila sin suministrar ese valor.

A menudo se puede determinar si una tabla es actualizable sólo observando el resultado de cualquier intento de modificación. Si su objetivo es crear vistas que permitan a los usuarios actualizar datos en las tablas fundamentales, entonces debe considerar la complejidad de esas vistas y las funciones que vayan a realizar. También considere que las limitaciones que tengan lugar en las tablas fundamentales afectan la habilidad para modificar e insertar datos mediante una vista.

Utilice la cláusula WITH CHECK OPTION

Ahora volvamos a la sintaxis de CREATE VIEW que ya se presentó en la sección “Definición de las vistas de SQL”. La última línea de la sintaxis incluye la siguiente cláusula:

[WITH CHECK OPTION]

La cláusula `WITH CHECK OPTION` aplica a vistas actualizables que incluyen la cláusula `WHERE` en la instrucción `SELECT`. La mejor manera de ilustrar cómo funciona esto es por medio de un ejemplo. Modifiquemos la definición de la última vista que examinamos:

```
CREATE VIEW COM_EMP AS
  SELECT ID_EMPLEADO, AÑO_1999, AÑO_2000
  FROM COMISIONES_EMPLEADO
  WHERE AÑO_1999 > 100;
```

La cláusula `WHERE` especifica que sólo las filas `AÑO_1999` con valores mayores que 100 se devuelven. Esto en sí es bastante claro. Sin embargo, suponga que quiere actualizar esta vista estableciendo un valor `AÑO_1999` que sea menor o igual que 100. Ya que la vista es actualizable, le permitirá hacer eso. Sin embargo, si se invoca la vista, la fila que se actualizó ya no sería visible ni se podría actualizar más.

Para evitar este problema se puede agregar la cláusula `WITH CHECK OPTION` a la definición de la vista, como en el siguiente ejemplo:

```
CREATE VIEW COM_EMP AS
  SELECT ID_EMPLEADO, AÑO_1999, AÑO_2000
  FROM COMISIONES_EMPLEADO
  WHERE AÑO_1999 > 100
  WITH CHECK OPTION;
```

Ahora si se trata de actualizar un valor `AÑO_1999` a una cantidad menor o igual que 100, se recibe un mensaje de error diciendo que el cambio no se puede hacer. Como se observa, la cláusula `WITH CHECK OPTION` es una manera práctica de garantizar que los usuarios no realicen actualizaciones que les impida la utilización eficaz de las vistas que se crean.

Eliminación de vistas de la base de datos

No cabe duda que se enfrentará a situaciones en las que desee quitar la definición de una vista de su base de datos. La sintaxis para hacerlo es bastante simple:

```
DROP VIEW <nombre de la vista>
```

Cuando se ejecuta la instrucción `DROP VIEW`, la definición de la vista se quita; sin embargo, ninguno de los datos fundamentales (que se almacenan en las tablas base) se afectan. Una vez que la vista se elimina, se puede volver a crear la vista o crear una vista diferente con el mismo nombre. Ahora veamos un ejemplo rápido:

```
DROP VIEW COM_EMP;
```

Esta instrucción elimina la vista `COM_EMP` de la base de datos, pero deja los datos fundamentales intactos.

Pregunta al experto

P: Se analizó la creación y eliminación de vistas, pero no se mencionó su modificación. ¿Respalda SQL cualquier tipo de instrucción ALTER VIEW?

R: No, el estándar SQL:2006 no soporta la modificación de las vistas. Sin embargo, algunos RDBMS soportan la instrucción ALTER VIEW. Sin embargo, esté consciente de que la funcionalidad respaldada por estas instrucciones puede variar de un producto a otro. Por ejemplo, SQL Server y MySQL tienen las instrucciones ALTER VIEW, que son bastante robustas y permiten cambiar muchos aspectos de la definición de una vista, incluyendo la instrucción SELECT. Por otra parte, la instrucción ALTER VIEW en Oracle se utiliza para recompilar manualmente, a fin de evitar la sobrecarga en el tiempo de ejecución o modificar algunas limitaciones que Oracle respalda en las vistas. De hecho, para modificar una vista en Oracle tiene primero que eliminarse y después volver a crearse, como es el caso en el estándar SQL. Sin embargo, Oracle tiene la instrucción CREATE OR REPLACE VIEW que esencialmente permite eliminar y volver a crear una vista en un solo paso.

P: En los ejemplos que se usaron para mostrar cómo se crean las vistas, se utilizaron una o dos tablas para los datos fuente. ¿Se pueden basar las vistas en más de dos tablas?

R: Sí, una vista puede basarse en tantas tablas como pueda ser lógicamente consultada en la instrucción SELECT. Por ejemplo, suponga que desea crear una vista en la base de datos INVENTARIO. (La base de datos INVENTARIO es con la que se ha trabajado para los ejercicios en el libro.) La vista muestra los nombres de los artistas junto con los títulos de CD. Sin embargo, para hacer esto, la instrucción SELECT tiene que unir tres tablas. Tendrían que coincidir los valores de ID_ARTISTA en la tabla ARTISTAS y la tabla CD_ARTISTA, y tendrían que coincidir los valores ID_DISCO_COMPACTO en la tabla DISCOS_COMPACTOS y la tabla CD_ARTISTA. El resultado sería una vista que muestra una lista de artistas y sus CD. (En el capítulo 11 se analizará cómo se pueden unir estas tablas en la instrucción SELECT.)

P: En los ejemplos que se utilizaron para mostrar cómo se crean las vistas, todas las vistas hacen referencia a tablas base. ¿Se crean todas las vistas a partir de tablas base?

R: No, las vistas se pueden crear utilizando expresiones de consulta que extraen datos de otras vistas. También es posible crear una vista que contenga sólo datos calculados y, por lo tanto, no disponga de datos que dirijan a una tabla base.

Pruebe esto 5-1 **Añada vistas a su base de datos**

En este ejercicio se crearán dos vistas en la base de datos INVENTARIO. Las vistas se basan en las tablas que se crearon en los ejercicios anteriores. La primera vista se basa en una sola tabla, y la segunda vista en dos tablas. La segunda vista se creará dos veces. Se creará una vez, luego se eliminará la definición de la vista de la base de datos, y a continuación se volverá a crear una versión modificada de la vista. Se puede descargar el archivo Try_This_05.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de cliente de su RDBMS y conéctese a la base de datos INVENTARIO.
2. La primera vista que se crea se llama CD_EN_EXISTENCIA. La vista se basa en las columnas TITULO_CD y EN_EXISTENCIA en la tabla DISCOS_COMPACTOS. Se desea que la vista incluya sólo las filas cuyos valores en la columna EN_EXISTENCIA sean mayores que 10. La vista utilizará los mismos nombres de columna que la tabla e incluirá la cláusula WITH CHECK OPTION para evitar que los valores inferiores o iguales a 10 se añadan a la columna EN_EXISTENCIA. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE VIEW CDS_EN_EXISTENCIA AS
  SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
   WHERE EN_EXISTENCIA > 10 WITH CHECK OPTION;
```

3. Después se crea la vista llamada EDITORES_CD, que contiene las columnas TITULO_CD y EDITOR. La vista se basa en la columna TITULO_CD de la tabla DISCOS_COMPACTOS y en la columna NOMBRE_COMPañIA de la tabla DISQUERAS_CD. Tendrá que utilizar la cláusula WHERE para que coincidan las filas en las dos tablas. La cláusula WHERE también va a limitar las filas incluidas en la vista a aquellas cuyo valor ID_DISQUERA en la tabla DISQUERAS_CD sea 5403 o 5402. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE VIEW EDITORES_CD
  ( TITULO_CD, EDITOR ) AS
  SELECT DISCOS_COMPACTOS.TITULO_CD, DISQUERAS_CD.NOMBRE_COMPañIA
    FROM DISCOS_COMPACTOS, DISQUERAS_CD
   WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA
     AND DISQUERAS_CD.ID_DISQUERA = 5403 OR DISQUERAS_CD.ID_DISQUERA =
5402;
```

4. Decide que no desea limitar las filas a valores específicos en la columna ID_DISQUERA, así que debe eliminar la definición de la vista de la base de datos y volver a crear la vista sin las restricciones en los valores. Introduzca y ejecute la siguiente instrucción SQL:

```
DROP VIEW EDITORES_CD;
```

(continúa)

5. Ahora se puede volver a crear la vista EDITORES_CD. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE VIEW EDITORES_CD
( TITULO_CD, EDITOR ) AS
SELECT DISCOS_COMPACTOS.TITULO_CD, DISQUERAS_CD.NOMBRE_COMPANÍA
FROM DISCOS_COMPACTOS, DISQUERAS_CD
WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA;
```

6. Cierre la aplicación de cliente.

Resumen de Pruebe esto

Además de las seis tablas creadas en los ejercicios anteriores, la base de datos ahora incluye las vistas CD_EN_EXISTENCIA y EDITORES_CD. Más adelante en el libro, se utilizarán estas vistas para consultar datos de las tablas base y actualizar esos datos. Una vez que tenga una mejor comprensión de cómo crear las instrucciones SELECT, será capaz de definir vistas que son aún más amplias y proporcionan un mayor nivel de detalle que las vistas que se crearon hasta ahora.

Autoexamen Capítulo 5

1. ¿Cuáles son las dos ventajas de utilizar vistas?
2. ¿Cuáles son los tres tipos de tablas almacenadas, soportadas por SQL?
3. ¿Qué sucede si no se asignan nombres de columna a una vista?
4. ¿Cómo se asignan los tipos de datos a las columnas de una vista?
5. ¿En qué circunstancias se deben proporcionar nombres a las columnas de una vista en una definición de vista?
6. Se crea la vista llamada EMP_CUMPLEAÑOS. La vista se basa en las columnas EMP_NOMBRE y CUMPLEAÑOS de la tabla EMPLEADOS. Los nombres de columna de la vista serán los mismos que los nombres de columna de la tabla. ¿Qué código SQL se utilizaría para crear la vista?
7. Se crea una vista basada en la tabla DISCOS_COMPACTOS en la base de datos INVENTARIO. Se desea que la vista incluya sólo las filas cuyo valor en la columna ID_DISQUERA sea 546. ¿Qué cláusula (además de las cláusulas SELECT y FROM) deberá incluirse en la instrucción SELECT para la vista?
8. Se crea una vista que hace referencia a las tablas EMPLEADOS y TITULO_TRABAJO. Los datos en las dos tablas coinciden a través de la columna ID_TITULO_TRABAJO en cada tabla. ¿Cómo se debe escribir la cláusula WHERE en la instrucción SELECT de la vista?

9. Se crea una vista que hace referencia a las tablas EMPLEADOS y TITULO_TRABAJO. Los datos en las dos tablas coinciden por la columna ID_TITULO_TRABAJO en cada tabla. Se desea que la vista muestre sólo las filas cuyo valor en la columna ID_TITULO_TRABAJO de la tabla TITULO_TRABAJO sea 109. ¿Cómo se debe escribir la cláusula WHERE en la instrucción SELECT de la vista?
10. ¿Qué es una especificación de consulta?
11. ¿Qué directrices deben seguirse si desea crear una vista actualizable?
 - A Los datos dentro de la vista no se pueden resumir, agrupar o eliminar automáticamente.
 - B Por lo menos una columna en la tabla fuente debe ser actualizable.
 - C Cada columna en la vista debe rastrearse exactamente a una columna fuente en una tabla.
 - D Cada fila en la vista debe rastrearse exactamente a una fila fuente en una tabla.
12. Se crea la siguiente vista basada en la tabla DISCOS_COMPACTOS en la base de datos INVENTARIO:

```
CREATE VIEW PROMEDIO_EN_EXISTENCIA AS
  SELECT PROM(EN_EXISTENCIA)
    FROM DISCOS_COMPACTOS;
```

¿Cómo se insertan datos a través de esta vista?

13. ¿A qué tipo de vista aplica la cláusula WITH CHECK OPTION?
14. Se crea la siguiente definición de vista:

```
CREATE VIEW COM_EMP AS
  SELECT ID_EMPLEADO, AÑO_1999, AÑO_2000
    FROM COMISION_EMPLEADO
   WHERE AÑO_1999 > 100;
```

Se quiere utilizar la vista para actualizar los datos. ¿Qué sucede si se cambia el valor AÑO_1999 a una cantidad inferior o igual que 100?

15. Se desea modificar la definición de la vista COM_EMP en la base de datos. ¿Cómo se modifica esa definición?
16. Se desea eliminar la definición de la vista EMP_CUMPLEAÑOS de la base de datos. ¿Qué instrucción SQL deberá utilizarse?
17. ¿Qué les sucede a los datos de SQL cuando se elimina una vista de la base de datos?

Capítulo 6

Gestión de seguridad en la base de datos



Habilidades y conceptos clave

- Entienda el modelo de seguridad de SQL
 - Creación y eliminación de roles
 - Otorgue y revoque privilegios
 - Otorgue y revoque roles
-

Un componente crítico de cualquier base de datos es la habilidad para proteger los datos contra accesos no autorizados y ataques maliciosos. Una base de datos debe garantizar que los usuarios no autorizados no puedan ver o cambiar datos que no deban ver o cambiar. Al mismo tiempo, a los usuarios autorizados no se les debe impedir acceder a cualquier información que está disponible para ellos. El equilibrio ideal es dar con exactitud a cada uno de los usuarios de base de datos los privilegios que necesitan para hacer su trabajo, nada más y nada menos. Con el fin de respaldar estas capacidades, SQL define un modelo de seguridad que permite determinar qué usuarios pueden acceder a datos específicos y lo que pueden hacer con esos datos. La parte esencial de este modelo es el identificador de autorización. Un *identificador de autorización*, como se aprendió en el capítulo 2, es un objeto en el entorno SQL que representa a un usuario o grupo de usuarios a los que se les otorgan privilegios específicos para acceder a objetos y datos dentro del entorno SQL. Los privilegios a los objetos de esquema se les conceden a los identificadores de autorización. El tipo de privilegio otorgado determina el tipo de acceso. En este capítulo se examina el modelo de seguridad de SQL, cómo utiliza a los identificadores de autorización y cómo configura los privilegios sobre objetos en la base de datos SQL.

Entienda el modelo de seguridad de SQL

Los identificadores de autorización proporcionan la base para la seguridad en la base de datos. Se permite acceder a todos los objetos a través de estos identificadores. Si el identificador de autorización no tiene los privilegios apropiados para acceder a un objeto específico, como una tabla, los datos dentro de esa tabla no están disponibles para ese usuario. Además, cada identificador de autorización se puede configurar con diferentes tipos de privilegios. Por ejemplo, puede permitir que algunos identificadores de autorización consulten los datos dentro de una tabla específica, mientras que permita a otros identificadores de autorización tanto ver como modificar esos datos.

SQL respalda dos tipos de identificadores de autorización: identificadores de usuario (o usuarios) y nombres de rol (o roles). Un *identificador de usuario* es una cuenta de seguridad individual que puede representar a una persona, una aplicación o un servicio del sistema (de los cuales todos se consideran como usuarios de la base de datos). El estándar SQL no especifica cómo una aplicación de SQL puede crear a un identificador de usuario. El identificador puede estar vinculado al sistema operativo en el que se ejecuta el sistema de gestión de base de datos relacional (RDBMS), o puede estar creado explícitamente en el entorno RDBMS.

Un *nombre de rol* es un conjunto de privilegios definidos que se pueden asignar a un usuario o a otro rol. Si a un nombre de rol se le concede acceder a un objeto de esquema, entonces todos los identificadores de usuario y los nombres de rol que se asignaron al nombre del rol específico se les concede acceder a ese objeto siempre y cuando el nombre de rol sea el del identificador de autorización actual. Por ejemplo, en la figura 6-1 el nombre de rol DEPTO_MRKT se asignó al nombre de rol DEPTO_ACCT y los cuatro identificadores de usuario: Ethan, Max, Linda y Emma. Si el nombre de rol DEPTO_ACCT es el identificador de autorización actual y se le concedió acceder a la tabla INTERPRETES, el nombre de rol DEPTO_ACCT y los cuatro identificadores de usuario tienen acceso a la tabla INTERPRETES. Observe que, a diferencia de un identificador de usuario, SQL *sí* especifica cómo crear un nombre de rol, que se analizará en la sección “Creación y eliminación de roles”, más adelante en este capítulo.

Los nombres de rol se utilizan comúnmente como un mecanismo para la concesión de un conjunto uniforme de privilegios a los identificadores de autorización que deben tener los mismos privilegios, como las personas que trabajan en el mismo departamento. También tienen la ventaja de la existencia independiente de los identificadores de usuario, lo que significa que se pueden crear antes que los identificadores de usuario, y persisten incluso después de que los identificadores de usuario suprimen las referencias. Esto es muy útil a la hora de administrar los privilegios para un trabajo fluido.

Además de los identificadores de usuario y los nombres de rol, SQL respalda a un identificador de autorización especial incorporado llamado PUBLIC, que incluye a todos los que utilizan la base de datos. Al igual que con cualquier otro identificador de autorización, se pueden conceder privilegios de acceso a la cuenta PUBLIC. Por ejemplo, suponga que quiere que todos los clientes potenciales puedan ver su lista de CD.

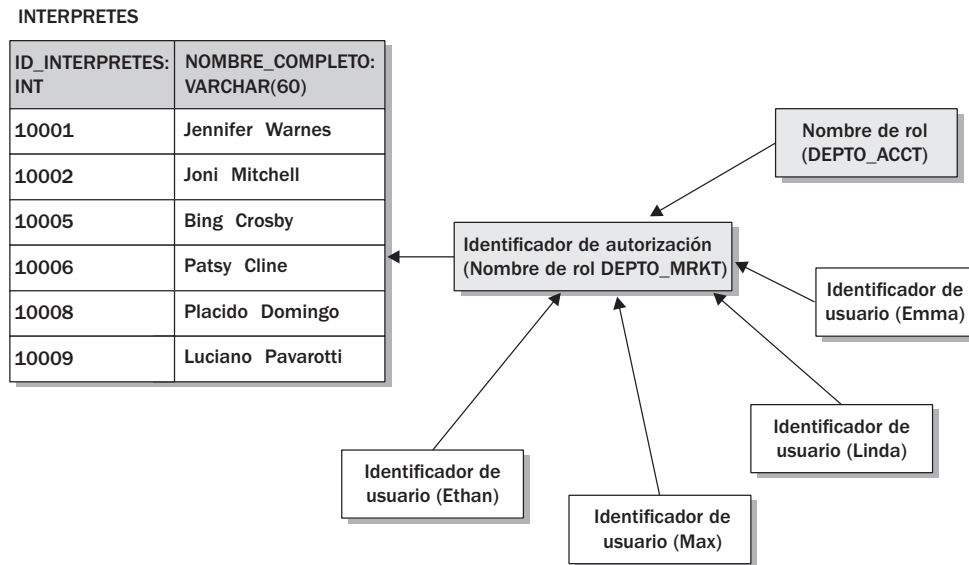


Figura 6-1 Rol DEPTO_MRKT asignado a cuatro identificadores de usuario y a un rol.

Puede conceder los privilegios necesarios a la cuenta PUBLIC para las tablas y columnas correspondientes. Evidentemente, la cuenta PUBLIC debe utilizarse con sumo cuidado, ya que puede abrir la oportunidad para personas con malas intenciones. De hecho, muchas organizaciones prohibieron su uso por completo.

Sesiones SQL

Cada sesión SQL se asocia con un identificador de usuario y un nombre de rol. Una *sesión SQL* es la conexión entre algún tipo de aplicación de cliente y la base de datos. La sesión proporciona el contexto en el que el identificador de autorización ejecuta las instrucciones SQL durante una sola conexión. A través de esta conexión, la sesión SQL mantiene la asociación con el par identificador de usuario / nombre de rol.

Echemos un vistazo a la figura 6-2, que muestra al par identificador de usuario/nombre de rol asociado con una sesión. Cuando una sesión se establece por primera vez, el identificador de usuario es siempre el *identificador de usuario de la sesión SQL*, que es un tipo especial de identificador de usuario que permanece asociado con la sesión a través de la conexión. Corresponde a la aplicación SQL determinar cómo una cuenta específica se convierte en el identificador de usuario de la sesión SQL, aunque puede ser una cuenta de usuario del sistema operativo o una cuenta específica para el RDBMS. Cualquier método que se utilice para asociar una cuenta con el identificador de usuario de la sesión SQL, esta cuenta es la que actúa como identificador de usuario actual.

Como también se puede observar en la figura 6-2, el nombre de rol es un valor nulo. El nombre de rol es siempre nulo cuando una sesión se establece por primera vez. En otras palabras, cada vez que la base de datos de SQL inicie y establezca una sesión, el identificador de usuario inicial siempre será el identificador de usuario de la sesión SQL y el nombre de rol siempre será un valor nulo.

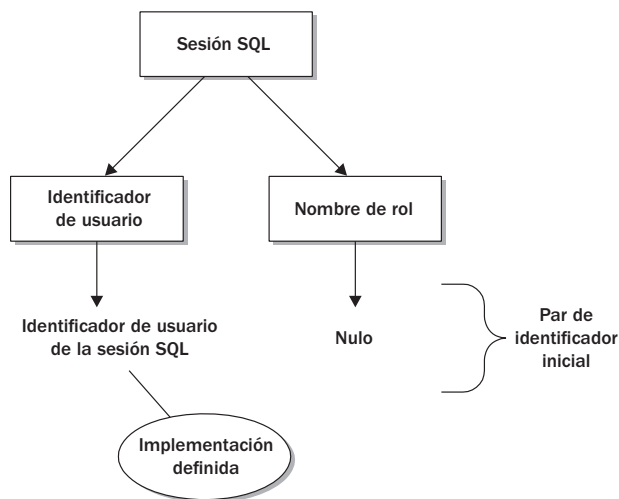


Figura 6-2 Sesión SQL con un identificador de usuario y un nombre de rol.

En cualquier momento durante una conexión, la sesión se asocia con un par identificador de usuario/nombre de rol; sin embargo, no es siempre el mismo par a lo largo del periodo de la sesión. Por ejemplo, las instrucciones SQL incrustadas, los módulos de clientes SQL y las rutinas invocadas SQL pueden especificar un identificador de autorización. Si se especifica un identificador nuevo, se convierte en el identificador de autorización actual hasta que la transacción se complete, y el acceso a los objetos se concede sobre la base del par actual identificador de usuario/nombre de rol.

Para cualquier par actual de identificador de usuario/nombre de rol, uno de los dos valores casi siempre es nulo. En otras palabras, si se especifica un identificador de autorización, entonces el nombre de rol debe ser nulo; si un nombre de rol se especifica, entonces el identificador de usuario debe ser nulo. Cualquier valor que no sea nulo es el identificador de autorización.

Cuando más de un par identificador de usuario/nombre del rol se utiliza durante una sesión, se crea una pila de autorización que refleja al identificador de usuario actual. El par en la parte superior de la pila es el identificador de autorización actual. La figura 6-3 muestra un ejemplo de una pila de autorización que se puede crear durante una sesión.

En este ejemplo, el par identificador de usuario/nombre del rol inicial está en la parte inferior de la pila. Como cabría esperar, el identificador de usuario es el identificador de usuario de la sesión SQL y el nombre de rol es un valor nulo. Acceder a un objeto de base de datos se basa en los privilegios concedidos al identificador de usuario de la sesión SQL cuando es actual.

Durante la sesión, una instrucción SQL incrustada especifica un identificador de autorización Usuario_App, que es un identificador de usuario. Cuando la instrucción incrustada se ejecuta, el Usuario_App se convierte en el identificador de autorización actual y los privilegios de acceso se basan en esa cuenta.

Suponga una de las instrucciones SQL incrustadas, luego llame una rutina SQL invocada que especifique una autorización de ROL_RUTINA, que es un nombre de rol. ROL_RUTINA se convierte en el identificador de autorización actual y está en la parte superior de la pila de autorización. Una vez que se ejecuta la ruta, el identificador de autorización actual revierte al Usua-

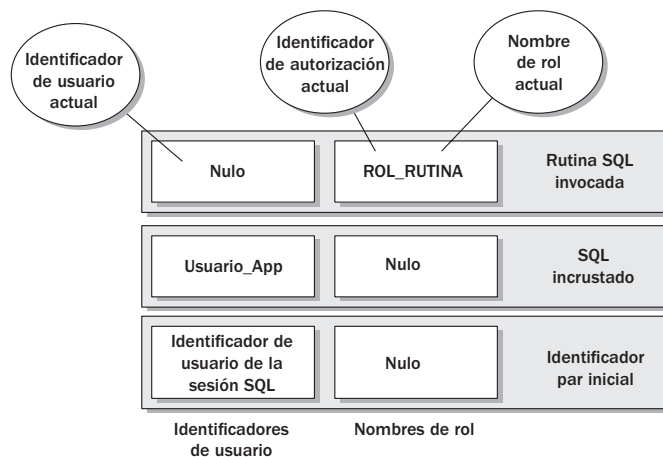


Figura 6-3 Pila de autorización creada durante una sesión SQL.

rio_App, hasta que las instrucciones incrustadas se ejecuten, con lo que después el identificador de autorización se revierte al identificador de usuario de la sesión SQL.

Observe que en cada par identificador de usuario/nombre de rol mostrado en la figura 6-3 hay exactamente un valor nulo. El otro valor, el que no es nulo, es el identificador de autorización.

Pregunta al experto

- P:** Se expuso que el identificador de autorización actual puede cambiar. ¿Cómo se puede determinar la autorización del usuario actual y el nombre de rol en cualquier momento durante una sesión?
- R:** SQL soporta varios valores especiales que permiten determinar los valores actuales de varios tipos de usuarios. Los valores especiales actúan como marcadores de posición para el valor de usuario relacionado actual. Se pueden utilizar estos valores especiales en expresiones para devolver el valor del tipo específico de usuario. Por ejemplo, se puede utilizar el valor especial `USUARIO_ACTUAL` para devolver el valor del identificador de usuario actual. SQL respalda cinco de estos valores especiales: `USUARIO_ACTUAL`, `USUARIO`, `ROL_ACTUAL`, `USUARIO_SESION` y `USUARIO_SISTEMA`. Los valores `USUARIO_ACTUAL` y `USUARIO` significan lo mismo y devuelven un valor igual al del identificador de usuario actual. El `ROL_ACTUAL` devuelve el nombre de rol actual, y `USUARIO_SESION` devuelve el identificador de usuario de la sesión SQL. Si el identificador de usuario de la sesión SQL es el identificador de usuario actual, entonces `USUARIO_ACTUAL`, `USUARIO` y `USUARIO_SESION` tienen el mismo valor, que puede ocurrir si el par identificador inicial es el único par identificador de usuario/nombre de rol activo (el par en la parte superior de la pila de autorización). La última función, `USUARIO_SISTEMA`, devuelve el sistema operativo del usuario que invocó el módulo SQL. A medida que avancemos más en este capítulo, se verá cómo los valores especiales `USUARIO_ACTUAL` y `ROL_ACTUAL` se utilizan para reconocer al identificador de autenticación actual en el momento de crear los roles y conceder privilegios. (Ver las secciones “Creación y eliminación de roles”, “Otorgue y revoque privilegios” y “Otorgue y revoque roles”.) Además, se encuentra más información sobre valores especiales en el capítulo 10.

Acceda a objetos de base de datos

Ahora que tiene una mejor comprensión de lo que es un identificador de autorización (junto con los identificadores de usuario y los nombres de rol), echemos un vistazo de lo que puede hacer con estos identificadores. Acceder a los datos en una base de datos se basa en la posibilidad de acceder a los objetos que contienen los datos. Por ejemplo, puede conceder a algunos usuarios el acceso a un conjunto específico de tablas, mientras que otros usuarios pueden acceder sólo a columnas específicas dentro de una tabla. SQL permite definir los privilegios de acceso a los siguientes objetos de esquema:

- Tablas base
- Vistas

- Columnas
- Dominios
- Conjunto de caracteres
- Cotejos
- Traducciones
- Tipos de usuarios definidos
- Secuencias
- Activadores
- Rutas invocadas SQL

Para cada tipo de objeto se pueden asignar determinados tipos de privilegios que varían según el tipo de objeto. Estos privilegios asignados se asocian con identificadores de autorización específicos. En otras palabras, se pueden asignar uno o más de los privilegios de un objeto a uno o más identificadores de autorización. Por ejemplo, se puede asignar el privilegio SELECT para una tabla para el identificador de autorización PUBLIC. Esto permite a todos los usuarios de la base de datos ver los contenidos de esa tabla.

SQL define nueve tipos de privilegios que se pueden asignar a un objeto de esquema. La tabla 6-1 describe cada uno de estos privilegios y enumera los tipos de objetos a los cuales se puede asignar el privilegio.

Los privilegios se conceden en las bases de datos objetos utilizando la instrucción GRANT para especificar los objetos así como los identificadores de autorización que adquieren los privile-

Privilegio	Descripción	Objetos
SELECT	Permite que identificadores de autorización específicos consulten datos en el objeto. Por ejemplo, si al UsuarioA se le concede el privilegio SELECT en la tabla CD_ARTISTAS, el usuario puede ver los datos de esa tabla.	Tablas Vistas Columnas Métodos (en tipos estructurados)
INSERT	Permite que identificadores de autorización específicos inserten datos en el objeto. Por ejemplo, si al UsuarioA se le concede el privilegio INSERT en la tabla CD_ARTISTAS, el usuario puede añadir datos a esa tabla.	Tablas Vistas Columnas
UPDATE	Permite que identificadores de autorización específicos actualicen datos en el objeto. Por ejemplo, si al UsuarioA se le concede el privilegio UPDATE en la tabla CD_ARTISTAS, el usuario puede modificar datos a esa tabla. Sin embargo, este privilegio no le permite al usuario cambiar la definición de la tabla.	Tablas Vistas Columnas
DELETE	Permite que identificadores de autorización específicos eliminen datos del objeto. Por ejemplo, si al UsuarioA se le concede el privilegio DELETE en la tabla CD_ARTISTAS, el usuario puede eliminar datos de esa tabla. Sin embargo, este privilegio no le permite al usuario eliminar la definición de la tabla de la base de datos.	Tablas Vistas

Tabla 6-1 Privilegios de seguridad asignados a objetos de base de datos.

Privilegio	Descripción	Objetos
REFERENCES	Permite que identificadores de autorización específicos definan los objetos (como limitaciones referenciales) que hacen referencia a la tabla configurada con el privilegio REFERENCES. Por ejemplo, si al UsuarioA se le concede el privilegio REFERENCES en la tabla CD_ARTISTAS, el usuario puede crear otros objetos que hagan referencia a la tabla CD_ARTISTAS, como sería el caso con claves foráneas. (Note que el UsuarioA también debe tener la autorización para crear otros objetos.)	Tablas Vistas Columnas
TRIGGER	Permite que identificadores de autorización específicos generen activadores en la tabla. Por ejemplo, si al UsuarioA se le concede el privilegio TRIGGER en la tabla CD_ARTISTAS, el usuario puede crear activadores en esa tabla.	Tablas
USAGE	Permite que los identificadores de autorización específicos utilicen el objeto en una definición de columna. Por ejemplo, si al UsuarioA se le concede el privilegio USAGE en el dominio DINERO, el usuario puede incluir el dominio en la definición de columna cuando se crea una tabla. (Note que el UsuarioA también debe tener la autorización para crear una tabla.)	Dominios Conjunto de caracteres Cotejos Traducciones Tipos definidos por el usuario Secuencias
EXECUTE	Permite que los identificadores de autorización específicos invoquen una rutina SQL invocada. Por ejemplo, si al UsuarioA se le concede el privilegio EXECUTE en el procedimiento almacenado LISTA_CD_ACTUALIZADA, el usuario sería capaz de invocar ese procedimiento almacenado.	Rutinas invocadas SQL
UNDER	Permite que los identificadores de autorización específicos definan un subtipo directo en un tipo estructurado. Un <i>subtipo directo</i> es un tipo estructurado que se asocia con otro tipo estructurado como un objeto secundario de ese tipo. Por ejemplo, si al UsuarioA se le concede el privilegio UNDER en el tipo estructurado EMPLEADO, el usuario puede definir subtipos directos tales como ADMINISTRADOR o SUPERVISOR.	Tipos estructurados

Tabla 6-1 Privilegios de seguridad asignados a objetos de base de datos (continuación).

gios. También se pueden revocar privilegios usando la instrucción REVOKE. Se entrará en mayor detalle sobre ambas instrucciones a medida que avancemos por el capítulo. Sin embargo, antes de discutir cómo conceder o revocar privilegios, primero deseo discutir cómo crear un nombre de rol. (Recuerde, el estándar SQL no respalda la creación de un identificador de usuario, sólo nombres de rol. El proceso de creación de identificadores de usuario es la aplicación específica.)

Creación y eliminación de roles

En su mayor parte, la creación de un rol es un proceso muy sencillo. La instrucción incluye sólo una cláusula obligatoria y una cláusula opcional, como se muestra en la siguiente sintaxis:

```
CREATE ROLE <nombre del rol>
[ WITH ADMIN { CURRENT_USER | CURRENT_ROLE } ]
```

Observe que la única parte requerida de la sintaxis es la cláusula `CREATE ROLE`, lo que significa que todo lo que realmente se necesita hacer es especificar un nombre para el rol. La cláusula `WITH ADMIN` es opcional y rara vez necesitará utilizarla. Es necesario sólo si el par identificador de usuario/nombre de rol actual contiene valores no nulos. La cláusula permite diseñar tanto al identificador de usuario actual (`CURRENT_USER`) o el nombre de rol actual (`CURRENT_ROLE`) como el identificador de autenticación permite asignar el rol a los identificadores de usuario o los nombres de rol. Si la cláusula `WITH ADMIN` no se especifica, al identificador de autenticación actual, ya sea el identificador de usuario actual o el nombre de rol actual, se le permite asignar el rol.

NOTA

Probablemente encontrará que rara vez necesite utilizar la cláusula `WITH ADMIN` de la instrucción `CREATE ROLE`, particularmente como programador principiante de SQL, y no es ampliamente respaldado en los productos RDBMS. Como resultado, se mantiene una breve discusión de la cláusula.

Ahora veamos la creación de un rol. En el siguiente ejemplo se utiliza la instrucción `CREATE ROLE` para crear el rol `CLIENTES`:

```
CREATE ROLE CLIENTES;
```

Eso es todo lo que hay que hacer. Una vez que se crea el rol, se puede otorgar el rol a identificadores de usuario u otros nombres de roles. Se analizará la concesión y revocación de roles en la sección “Otorgue y revoque roles” después en este capítulo.

La eliminación de un rol es tan fácil como la creación de uno. La sintaxis que se utiliza es la siguiente:

```
DROP ROLE <nombre del rol>
```

En este caso, simplemente necesita identificar el nombre del rol, como en el siguiente ejemplo:

```
DROP ROLE CLIENTES;
```

El rol se elimina de la base de datos. Sin embargo, antes de eliminar un rol, asegúrese que se trata del rol que ya no es necesario o que es el que particularmente quiere eliminar (por razones de seguridad).

Como se puede observar, la creación y eliminación de roles es un proceso muy sencillo, y puede hacer la gestión de los usuarios mucho más fácil. Los roles esencialmente permiten agrupar los usuarios que requieren los mismos privilegios sobre los mismos objetos. Ahora echemos un vistazo a la concesión y revocación de privilegios para los identificadores de autenticación, incluyendo los identificadores de usuario y los nombres de rol.

NOTA

El respaldo para las instrucciones `CREATE ROLE` y `DROP ROLE` varía de una aplicación a otra. Por ejemplo, Oracle y SQL Server (2005 y 2008) respaldan ambas instrucciones, pero no la opción `WITH ADMIN`. MySQL 5.0 no parece respaldar el concepto de roles.

Otorgue y revoque privilegios

Cuando se conceden privilegios en un objeto, se asocian uno o más privilegios con uno o más identificadores de autorización. Este conjunto de privilegios e identificadores de autorización se

asignan al objeto, que permite al identificador de autorización poder acceder al objeto según el tipo de privilegios definidos. Para otorgar privilegios, se debe utilizar la instrucción `GRANT`, como se muestra en la siguiente sintaxis:

```
GRANT { ALL PRIVILEGES | <lista de privilegios> }  
ON <tipo de objeto> <nombre del objeto>  
TO { PUBLIC | <lista de identificador de autorización> } [WITH GRANT OPTION]  
[GRANTED BY { CURRENT_USER | CURRENT_ROLE }]
```

La instrucción, como se puede observar, incluye tres cláusulas requeridas —`GRANT`, `ON` y `TO`— y dos cláusulas opcionales —`WITH GRANT OPTION` y `GRANTED BY`—. Se analizará cada cláusula individualmente, excepto la cláusula `GRANTED BY`. La cláusula `GRANTED BY` es similar a la cláusula `WITH ADMIN` en la instrucción `CREATE ROLE`. Como esa cláusula, la cláusula `GRANTED BY` aplica sólo en esas situaciones donde el par identificador de usuario/nombre de rol actual contiene valores no nulos, y no se implementa ampliamente en los RDBMS. Como principiante en la programación de SQL, no necesita inquietarse con la cláusula `GRANTED BY`.

NOTA

Muchas aplicaciones de los proveedores contienen disposiciones para la asignación de privilegios del sistema tales como iniciar y detener la base de datos de los identificadores de autorización utilizando la instrucción `GRANT`. Ya que la sintaxis de estas variantes es totalmente específica a implementación, no se cubrirá aquí.

Se deben tener los privilegios necesarios en un objeto para otorgar privilegios en ese objeto. Si creó el objeto, entonces es el propietario, lo que significa que puede acceder por completo al objeto. (Todos los privilegios se le otorgan, incluyendo la habilidad para asignar privilegios a otros identificadores de autorización.)

Ahora echemos un vistazo a la cláusula `GRANT`. La cláusula incluye dos opciones: `ALL PRIVILEGES` y el marcador de posición `<lista de privilegios>`. Si utiliza las palabras clave `ALL PRIVILEGES`, se otorgan todos los privilegios disponibles a ese objeto de acuerdo con los privilegios que se otorgan sobre el objeto. Por ejemplo, asuma por un momento que se crea una tabla y es el propietario. Como resultado, automáticamente se le conceden los privilegios `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRIGGER` y `REFERENCES`. (Éstos son los únicos privilegios que aplican a una tabla. Remítase a la tabla 6-1 para ver la lista de privilegios y de los objetos a los que se aplican.) También se le concede de forma automática la habilidad de asignar estos privilegios. En esta situación, si se utilizan las palabras clave `ALL PRIVILEGES`, se le otorgan estos seis privilegios a los identificadores de autorización en la instrucción `GRANT`.

Si decide no utilizar la opción `ALL PRIVILEGES`, debe entonces listar cada privilegio que debería aplicarse a los identificadores de usuario. Sin embargo, puede listar sólo los privilegios que se pueden aplicar al objeto específico. Por ejemplo, no se puede listar el privilegio `DELETE` si se otorga un privilegio en un dominio. También note que si se lista más de un privilegio, debe separar los nombres de los privilegios con comas.

La siguiente cláusula que veremos es la cláusula `ON`, que incluye dos marcadores de posición: `<tipo de objeto>` y `<nombre del objeto>`. El marcador de posición `<tipo de objeto>` simplemente se

refiere al tipo de objeto en el que está la concesión de permisos. SQL respalda los siguientes valores para el marcador de posición <tipo de objeto>:

- TABLE (incluye vistas)
- DOMAIN
- COLLATION
- CHARACTER SET
- TRANSLATION
- TYPE
- SEQUENCE
- Designación especial para rutinas invocadas SQL

Se requiere un valor para la designación <tipo de objeto>, a menos que el valor sea TABLE, en cuyo caso se puede dejar fuera. Si se proporciona el nombre de un objeto sin especificar un tipo, SQL asume que el valor del <tipo de objeto> es TABLE. Como se señala en la lista, la palabra clave TABLE también incluye vistas. Y por supuesto, no todas las aplicaciones respaldan todos los tipos de objetos incluidos en el estándar SQL, y algunos incluyen tipos de objetos que el estándar no cubre (lo cual conduce a la aplicación específica de las variaciones de SQL).

NOTA

En Oracle, la palabra clave TABLE se debe omitir. Por otra parte, en DB2 es opcional, pero ampliamente recomendado por el proveedor. SQL Server no respalda ninguna de las palabras clave de <tipo de objeto> del estándar SQL en la instrucción GRANT, pero en lugar de eso permite una clase de palabra clave seguida por un separador <::>; por lo tanto, en lugar de TABLE, se puede escribir OBJECT:: (OBJECT es la palabra clave para las bases de datos objetos tales como tablas y vistas). Es evidente que la lección aquí es que siempre consulte la documentación para la aplicación específica de SQL.

El marcador de posición <nombre del objeto> en la cláusula ON se refiere al nombre de un objeto específico. Siempre se requiere este valor.

La siguiente cláusula es la cláusula TO. Así como la cláusula GRANT, la cláusula TO tiene dos opciones: PUBLIC y el marcador de posición <lista de identificador de autorización>. Si se utiliza PUBLIC, todos los usuarios de la base de datos pueden acceder al objeto. Si se utiliza la opción <lista de identificador de autorización>, entonces se debe proporcionar el nombre de uno o más identificadores de autorización. Si se proporciona más de uno, deben estar separados por comas.

La última cláusula que se va a discutir es la cláusula WITH GRANT OPTION. Esta cláusula otorga a los identificadores de autorización el permiso para conceder cualquiera de los privilegios que se les otorgan en la instrucción GRANT. Por ejemplo, suponga que al identificador de usuario EmmaW se le otorga el privilegio SELECT en una de las tablas. Si se utiliza WITH GRANT OPTION, EmmaW podrá otorgar el privilegio SELECT a otro usuario. Si no se utiliza WITH GRANT OPTION, EmmaW no podrá otorgar el privilegio a otro usuario. Por cierto, la mayoría

de los expertos en seguridad recomiendan que nunca se utilice esta opción, ya que rápidamente se pierde el control sobre quién tiene qué privilegios.

Ahora que echamos un vistazo a la sintaxis, veamos algunos ejemplos. En el primer ejemplo, veamos la instrucción GRANT que otorga el privilegio SELECT al identificador de autorización PUBLIC. El privilegio se otorga a la vista llamada CD_DISPONIBLES, que enumera los CD que actualmente se tienen en existencia. Para otorgar el privilegio, utilice la siguiente instrucción:

```
GRANT SELECT ON TABLE CD_DISPONIBLES TO PUBLIC;
```

El privilegio SELECT permite a todos los usuarios de la base de datos (PUBLIC) ver los datos en la vista CD_DISPONIBLES. Sin embargo, ya que a PUBLIC no se le otorga ningún otro privilegio, los usuarios pueden ver los datos, pero no tomar ninguna acción. Además, ya que la cláusula WITH GRANT OPTION no se incluye en la instrucción, los usuarios no pueden asignar el privilegio SELECT a ningún otro usuario (que es un punto discutible en este caso porque todos pueden ya acceder a la vista CD_DISPONIBLES).

Ahora veamos otro ejemplo. Esta vez se otorgan los privilegios SELECT, UPDATE e INSERT a los roles VENTAS y CONTABILIDAD para que puedan acceder a la tabla INVENTARIO_CD:

```
GRANT SELECT, UPDATE, INSERT
ON TABLE INVENTARIO_CD
TO VENTAS, CONTABILIDAD WITH GRANT OPTION;
```

Observe que los privilegios se separan por comas, así como los roles. Como resultado de esta instrucción, los usuarios asociados con los roles VENTAS y CONTABILIDAD pueden ver, actualizar e insertar información en la tabla CD_INVENTARIO. Además, estos usuarios pueden asignar los privilegios SELECT, UPDATE e INSERT a otros usuarios que necesiten acceder a la tabla CD_INVENTARIO.

El siguiente ejemplo que se examina tiene una ligera variación de este último. Todo es igual, excepto que esta vez se especifica qué columna se puede actualizar:

```
GRANT SELECT, UPDATE(TITULO_CD), INSERT
ON TABLE INVENTARIO_CD
TO VENTAS, CONTABILIDAD WITH GRANT OPTION;
```

Observe que se puede añadir un nombre de columna después de un privilegio específico. Se pueden agregar nombres de columnas sólo a los privilegios SELECT, UPDATE e INSERT. Si se añade más de un nombre de columna, se deben separar por comas.

La instrucción GRANT en este ejemplo aún permite a los usuarios de VENTAS y CONTABILIDAD ver e insertar información en la tabla INVENTARIO_CD, pero sólo pueden actualizar valores en la columna TITULO_CD. No pueden actualizar ningún otro valor de columna en la tabla. Además, aunque puede aún asignar privilegios a otros usuarios, sólo pueden asignar el privilegio UPDATE en la columna TITULO_CD.

Echemos un vistazo a un ejemplo más que otorgue el privilegio SELECT al identificador de autorización PUBLIC:

```
GRANT SELECT (TITULO_CD, EN_EXISTENCIA) ON INVENTARIO_CD TO PUBLIC;
```

El identificador de autorización PUBLIC permite a todos los usuarios ver los datos en las columnas TITULO_CD y EN_EXISTENCIA de la tabla INVENTARIO_CD, pero no pueden ver ninguna otra información de esa tabla y no pueden modificar los datos en modo alguno. Observe en esta instrucción que la palabra clave TABLE no está incluida. Como se dijo antes, TABLE no se requiere.

La instrucción GRANT, cuando se utiliza conjuntamente con los privilegios disponibles y los identificadores de autorización, proporciona una base sólida para la seguridad de la base de datos. Sin embargo, cada aplicación de SQL es diferente con respecto a cómo se implementa y mantiene la seguridad. Por lo tanto, cuando se trata de cuestiones de seguridad, es importante que trabaje en estrecha colaboración con los administradores de red y base de datos y lea cuidadosamente la documentación del producto.

Revoque privilegios

Ahora que sabe cómo conceder privilegios a los identificadores de autorización, es hora de aprender cómo revocar esos privilegios. La instrucción que se utiliza para revocar privilegios es la instrucción REVOKE, como se muestra en la siguiente sintaxis:

```
REVOKE [ GRANT OPTION FOR ] { ALL PRIVILEGES | <lista de privilegios> }
ON <tipo de objeto> <nombre del objeto>
FROM { PUBLIC | <lista de identificador de autorización>
[GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
{RESTRICT | CASCADE}
```

Probablemente reconozca muchos de los elementos de la sintaxis de la instrucción GRANT o de otras instrucciones. De hecho, el único componente nuevo, que no sea la palabra clave REVOKE, es la cláusula GRANT OPTION FOR. Echemos un vistazo a la primera, ya que está al comienzo de la instrucción REVOKE. Esta cláusula aplica sólo cuando la cláusula WITH GRANT OPTION se utiliza en la instrucción GRANT. Si un privilegio se otorga con esta cláusula, se puede utilizar la cláusula GRANT OPTION FOR para eliminar ese permiso en particular. En caso de hacer uso de ésta, los privilegios se conservan, pero el usuario ya no puede otorgar esos privilegios a otros usuarios. Sin embargo, muy pocos productos RDBMS respaldan esta cláusula.

Olvide por un momento la cláusula GRANT OPTION FOR y veamos la cláusula REVOKE, que se utiliza para revocar todos los privilegios de un objeto (ALL PRIVILEGES) o sólo los privilegios definidos (<lista de privilegios>). Ambas opciones tienen el mismo significado que tuvieron en la instrucción GRANT; puede utilizar ALL PRIVILEGES o puede listar cada privilegio separado por una coma.

Las cláusulas ON y GRANTED BY en la instrucción REVOKE son exactamente las mismas que las cláusulas ON y GRANTED BY en la instrucción GRANT. Para la cláusula ON, se deben especificar valores para los marcadores de posición <tipo de objeto> y <nombre del objeto>; sin embargo, si el valor del <tipo de objeto> es TABLE, entonces se puede dejar fuera (y como antes, se debe omitir en Oracle y SQL Server). Para la cláusula GRANTED BY, asumiendo que

el RDBMS lo respalda (la mayoría no lo hace), puede escoger una de dos opciones (CURRENT_USER o CURRENT_ROLE).

La cláusula FROM en la instrucción REVOKE se puede comparar también a la instrucción GRANT. La única diferencia es que en la instrucción GRANT se utiliza la palabra clave TO, pero en la instrucción REVOKE se usa la palabra clave FROM. En ambos casos, se debe utilizar PUBLIC como identificador de autorización, o se deben enumerar los identificadores de usuario y los nombres de rol.

Los últimos elementos de la instrucción a analizar son las palabras clave RESTRICT y CASCADE. Probablemente recuerde estas palabras clave de los capítulos 2, 3 y 4. Si se especifica RESTRICT, el privilegio no se revoca si se pasa a otros usuarios (en otras palabras, si hay algún privilegio dependiente). (Esto significaría que WITH GRANT OPTION se utilizó en la instrucción GRANT y que el identificador de autorización que otorgó el privilegio entonces concedió el privilegio a alguien más.) Si se especifica CASCADE, el privilegio se revoca así como cualquiera de los privilegios que se pasan a otros usuarios.

NOTA

Las aplicaciones de los proveedores varían. En Oracle, CASCADE debe especificarse como CASCADE CONSTRAINTS. En ambos, Oracle y SQL Server, no se puede especificar RESTRICT, sino que más bien es el comportamiento predeterminado cuando no se especifica CASCADE. MySQL no permite que se especifique ninguno.

Ahora veamos algunos ejemplos de revocación de privilegios. La siguiente instrucción revoca el privilegio SELECT que se otorga al identificador de autorización PUBLIC en la vista CD_DISPONIBLES:

```
REVOKE SELECT ON TABLE CD_DISPONIBLES FROM PUBLIC CASCADE;
```

Como se puede observar, esta instrucción es muy similar a la instrucción GRANT. Se deben identificar los privilegios, los identificadores de autorización y el objeto. Además, se debe especificar RESTRICT o CASCADE.

El siguiente ejemplo se basa en privilegios que se otorgan a la tabla llamada INVENTARIO_CD. A los roles VENTAS y CONTABILIDAD se les otorgan los siguientes privilegios en esta tabla: GRANT, SELECT e INSERT. Para revocar esos privilegios, utilice la siguiente instrucción REVOKE:

```
REVOKE SELECT, UPDATE, INSERT ON TABLE INVENTARIO_CD  
FROM VENTAS, CONTABILIDAD CASCADE;
```

Observe que sólo tiene que especificar los privilegios que quiere revocar, el nombre de los objetos y el nombre de los identificadores de autorización. Sin embargo, ya que se están revocando todos los privilegios que se otorgaron, se puede simplificar la instrucción utilizando las palabras clave ALL PRIVILEGES, como se muestra en el siguiente ejemplo:

```
REVOKE ALL PRIVILEGES ON TABLE INVENTARIO_CD  
FROM VENTAS, CONTABILIDAD CASCADE;
```

Si no desea revocar todos los privilegios, pero en vez de eso quiere revocar sólo los privilegios UPDATE e INSERT, se pueden especificar sólo estos privilegios, como se muestra en el siguiente ejemplo:

```
REVOKE UPDATE, INSERT ON TABLE INVENTARIO_CD
FROM VENTAS, CONTABILIDAD CASCADE;
```

También se puede elegir revocar privilegios para sólo uno de los nombres de rol, en lugar de ambos. Además, se puede utilizar la palabra clave RESTRICT en lugar de CASCADE.

Ahora suponga que se otorgan los mismos privilegios como en el ejemplo anterior, pero además de éstos, se especifica WITH GRANT OPTION cuando se conceden los privilegios. Si se quiere revocar sólo la habilidad de los roles Ventas y Contabilidad de otorgar privilegios a otros usuarios, se puede utilizar la siguiente instrucción:

```
REVOKE GRANT OPTION FOR ALL PRIVILEGES ON INVENTARIO_CD
FROM VENTAS, CONTABILIDAD CASCADE;
```

Esta instrucción revoca sólo la habilidad de conceder privilegios; los roles Ventas y Contabilidad aún pueden acceder a la tabla INVENTARIO_CD. Si se desea revocar todos los privilegios, se tiene que ejecutar esta instrucción sin la cláusula GRANT OPTION FOR. Observe en esta instrucción que la palabra clave TABLE no se usa antes del nombre de la tabla. La instrucción REVOKE, como la instrucción GRANT, no requiere la palabra clave TABLE cuando se especifica una tabla o vista.

NOTA

Las diferencias de implementación del proveedor indicadas con el uso de la palabra clave TABLE en la instrucción GRANT (anteriormente en este capítulo) aplican de la misma forma a la instrucción REVOKE. En general, los proveedores respaldan la sintaxis idéntica entre las instrucciones GRANT y REVOKE, exceptuando que TO en la instrucción GRANT se convierte en FROM en la instrucción REVOKE.

Otorgue y revoque roles

Ahora que sabe cómo crear y eliminar roles y conceder y revocar privilegios, veamos la concesión y revocación de roles. Empezaremos con la concesión de roles. Para otorgar un rol, se debe utilizar la instrucción GRANT para asignar uno o más nombres de rol o uno o más identificadores de autorización, como se muestra en la siguiente sintaxis:

```
GRANT <lista de nombres de rol>
TO { PUBLIC | <lista de identificador de autorización> } [ WITH ADMIN OPTION ]
[ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]
```

Por ahora, la mayor parte de esta sintaxis debe lucir bastante familiar, excepto por algunas variaciones. La cláusula GRANT permite especificar un listado de uno o más nombres de rol. Si se especifica más de un nombre, se deben separar por comas. La cláusula TO permite especificar uno o más identificadores de autorización. Una vez más, si hay más de uno, se deben separar por comas. También se puede especificar el identificador de autorización PUBLIC para otorgar un rol a todos los usuarios de la base de datos. La cláusula WITH ADMIN OPTION, que es opcional,

permite a los identificadores de autorización otorgar un rol a otros usuarios. Y la cláusula **GRANTED BY**, que también es opcional (y sólo se soporta en algunos productos RDBMS), se utiliza en los casos raros cuando el par identificador de usuario/nombre de rol no contiene un valor nulo.

Echemos un vistazo a otro ejemplo. Suponga que tiene que crear un rol llamado **ADMINISTRADORES** y desea asignar ese rol a un identificador de usuario llamado LindaN. Se utiliza la siguiente sintaxis:

```
GRANT ADMINISTRADORES TO LindaN;
```

Ahora suponga que quiere darle a LindaN la habilidad de otorgar el rol **ADMINISTRADORES** a otros usuarios. Para hacer esto, simplemente añada la cláusula **WITH ADMIN OPTION**, como en el siguiente ejemplo:

```
GRANT ADMINISTRADORES TO LindaN WITH ADMIN OPTION;
```

También se pueden conceder múltiples roles a múltiples identificadores de usuario. Los identificadores de usuario pueden ser identificadores de usuarios u otros nombres de rol. En el siguiente ejemplo se otorgan los roles **ADMINISTRADORES** y **CONTABILIDAD** al identificador de usuario LindaN y el nombre de rol **MARKETING**:

```
GRANT ADMINISTRADORES, CONTABILIDAD TO LindaN, MARKETING WITH ADMIN  
OPTION;
```

Ahora que sabe cómo otorgar roles a los identificadores de autorización, es hora de aprender cómo revocar esos roles.

Revoque roles

La revocación de roles es muy similar a la revocación de privilegios. La instrucción que se utiliza para revocar privilegios es la instrucción **REVOKE**, como se muestra en la siguiente sintaxis:

```
REVOKE [ ADMIN OPTION FOR ] <lista de nombres de rol>  
FROM { PUBLIC | <lista de identificador de autorización> }  
[ GRANTED BY { CURRENT_USER | CURRENT_ROLE } ]  
{ RESTRICT | CASCADE }
```

Como se puede observar, no hay nada nuevo en la sintaxis excepto por la cláusula **ADMIN OPTION FOR**, que es similar a la cláusula **GRANT OPTION FOR** utilizada cuando se revocan privilegios. La cláusula permite revocar la habilidad de asignar roles a otros usuarios, sin revocar el propio rol.

NOTA

Las mismas diferencias mencionadas para Oracle, SQL Server y MySQL en cuanto a la revocación de privilegios aplican al uso de la instrucción **REVOKE**.

Veamos un ejemplo de la revocación de un rol. Suponga que otorga el rol **ADMINISTRADORES** al identificador de autorización LindaN. Se puede revocar ese rol utilizando la siguiente instrucción **REVOKE**:

```
REVOKE ADMINISTRADORES FROM LindaN CASCADE;
```

Si se otorgan los roles ADMINISTRADORES y CONTABILIDAD a LindaN y el rol MARKETING, la instrucción REVOKE sería como la siguiente:

```
REVOKE ADMINISTRADORES, CONTABILIDAD FROM LindaN, MARKETING CASCADE;
```

Ahora que hemos visto cómo otorgar y revocar roles, se puede ver cómo esto es similar a la concesión y revocación de privilegios. Una vez más, debo destacar que no todas las aplicaciones son similares con respecto a cómo otorgan y revocan privilegios y roles, así que asegúrese de revisar la documentación del producto y trabaje en estrecha colaboración con el administrador de bases de datos.

Pruebe esto 6-1 Gestión de roles y privilegios

En este ejercicio se crearán dos roles en la base de datos INVENTARIO, se otorgarán privilegios al identificador de autorización PUBLIC y a uno de los roles que se crea, se concederá uno de los roles al otro rol, y luego se revocarán todos los privilegios y roles. Finalmente, se eliminarán los dos roles que se crearon. La habilidad de seguir todos los pasos en este ejercicio dependerá del tipo de instrucciones relacionadas con la seguridad respaldada en la aplicación de SQL que esté utilizando. Sin embargo, el ejercicio está diseñado para que cualquiera de los roles que se creen o cualquiera de los privilegios que se asignen sean eliminados al final del ejercicio. No se utilizarán estos roles para ninguno de los siguientes ejercicios. Si por cualquier motivo este ejercicio afectara la seguridad del sistema en el que está trabajando, debería discutir este ejercicio con el administrador de la base de datos u omitirlo por completo. Puede descargar el archivo Try_This_06.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de cliente de su RDBMS y conéctese a la base de datos INVENTARIO.
2. Lo primero que se hace es crear el rol MRKT. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE ROLE MRKT;
```
3. Después se crea el rol PERSONAL_VENTAS. Introduzca y ejecute la siguiente instrucción SQL:

```
CREATE ROLE PERSONAL_VENTAS;
```
4. Ahora se otorga el privilegio SELECT en la vista CD_EN_EXISTENCIA. El privilegio se asigna al identificador de autorización PUBLIC. Introduzca y ejecute la siguiente instrucción SQL:

```
GRANT SELECT ON CD_EN_EXISTENCIA TO PUBLIC;
```
5. El siguiente privilegio que se otorga es el rol PERSONAL_VENTAS que se creó en el paso 3. Se otorgan los privilegios SELECT, INSERT y UPDATE en la tabla DISCOS_COMPACTOS. Para el privilegio UPDATE se especifica la columna TITULO_CD. También se le permite al rol

(continúa)

STAFF_VENTAS otorgar estos privilegios a otros usuarios. Introduzca y ejecute la siguiente instrucción SQL:

```
GRANT SELECT, INSERT, UPDATE(TITULO_CD) ON DISCOS_COMPACTOS  
TO PERSONAL_VENTAS WITH GRANT OPTION;
```

- 6.** Ahora se otorga el rol PERSONAL_VENTAS al rol MRKT. Introduzca y ejecute la siguiente instrucción SQL:

```
GRANT PERSONAL_VENTAS TO MRKT;
```

- 7.** El siguiente paso es revocar el privilegio SELECT que se otorgó al identificador de autorización PUBLIC. Introduzca y ejecute la siguiente instrucción SQL:

```
REVOKE SELECT ON CD_EN_EXISTENCIA FROM PUBLIC CASCADE;
```

- 8.** Ahora se revocan los privilegios que se concedieron al rol PERSONAL_VENTAS. Ya que se revocan todos los privilegios, se puede utilizar la palabra clave ALL PRIVILEGES. También debe asegurarse de revocar cualquier privilegio dependiente; por lo tanto, se utiliza la palabra clave CASCADE. Introduzca y ejecute la siguiente instrucción SQL:

```
REVOKE ALL PRIVILEGES ON DISCOS_COMPACTOS FROM PERSONAL_VENTAS CASCADE;
```

- 9.** Ahora se puede revocar el rol PERSONAL_VENTAS del rol MRKT. Introduzca y ejecute la siguiente instrucción SQL:

```
REVOKE PERSONAL_VENTAS FROM MRKT CASCADE;
```

- 10.** El siguiente paso es eliminar el rol MRKT. Introduzca y ejecute la siguiente instrucción SQL:

```
DROP ROLE MRKT;
```

- 11.** Finalmente, se requiere eliminar el rol PERSONAL_VENTAS. Introduzca y ejecute la siguiente instrucción SQL:

```
DROP ROLE PERSONAL_VENTAS;
```

- 12.** Cierre la aplicación clientes.

Resumen de Pruebe esto

La base de datos INVENTARIO debe estar compuesta de la misma manera como antes de comenzar este ejercicio. Los permisos y roles concedidos se revocaron, y los roles que se crearon se eliminaron. De esta manera, no tendrá que preocuparse por las consideraciones de seguridad para otros ejercicios. Para el resto de los ejercicios en el libro, se continuará trabajando en el mismo contexto de seguridad en el que se ha trabajado para este ejercicio y para todos los ejercicios anteriores al presente.

✓ Autoexamen Capítulo 6

1. ¿Cuál es la diferencia entre un identificador de usuario y un nombre de rol?
2. ¿Cuál es el nombre del identificador de autorización especial que otorga el acceso a todos los usuarios de la base de datos?
3. Cada _____ se asocia con un identificador de usuario y un nombre de rol.
4. ¿Con cuál de los siguientes se asocia una sesión SQL?
 - A Privilegio
 - B Identificador de autorización
 - C PUBLIC
 - D Nombre de rol
5. Cuando se establece por primera vez una sesión SQL, el identificador de usuario siempre es el _____.
6. ¿Cuál es el valor del nombre de rol actual cuando se establece por primera vez una sesión SQL?
7. ¿Qué es un identificador de autorización?
8. ¿Cuáles son los dos tipos de identificadores de autorización que respalda SQL?
9. ¿Qué privilegios se le deben otorgar a un objeto si desea permitir que un identificador de autorización consulte los datos de ese objeto?
10. Se establece una sesión SQL con la base de datos. El identificador de usuario actual es EthanW. El nombre de rol actual es nulo. ¿Cuál es el identificador de autorización actual?
11. ¿En qué objetos de esquema se pueden definir privilegios de acceso?
12. ¿En qué tipos de objetos de base de datos se puede asignar el privilegio DELETE?
 - A Tablas
 - B Vistas
 - C Columnas
 - D Dominios
13. ¿En qué tipos de objetos de base de datos se puede asignar el privilegio TRIGGER?
 - A Tablas
 - B Vistas
 - C Columnas
 - D Dominios
14. Se crea un rol llamado CONTABILIDAD. ¿Qué instrucción SQL deberá utilizarse?

- 15.** Se otorgan todos los privilegios de la vista NOMBRES_CD a todos los que utilizan la base de datos. ¿Qué instrucción SQL deberá utilizarse?
- 16.** Se otorga el privilegio SELECT al rol EMPLEADO_VENTAS a una tabla de la base de datos. Se desea que el rol EMPLEADO_VENTAS sea capaz de asignar el privilegio SELECT a otros usuarios. ¿Qué cláusula deberá incluirse en la instrucción GRANT?
- 17.** Se desea conceder el rol ACCT al usuario autorizado MaxN. No se quiere que el usuario pueda otorgar el rol a otros usuarios. ¿Qué instrucción SQL deberá utilizarse para otorgar el rol?

Parte II

Acceso y modificación de datos



Capítulo 7

Consulta de datos de SQL



Habilidades y conceptos clave

- Utilice la instrucción SELECT para la recuperación de datos
 - Utilice la cláusula WHERE para definir condiciones de búsqueda
 - Utilice la cláusula GROUP BY para agrupar los resultados de una consulta
 - Utilice la cláusula HAVING para especificar un grupo de condiciones de búsqueda
 - Utilice la cláusula ORDER BY para ordenar los resultados de una consulta
-

Una vez que se crearon los objetos en una base de datos y las tablas base están pobladas de datos, se pueden presentar las consultas que permiten recuperar información específica de la base de datos. Estas consultas, que suelen adoptar la forma de instrucciones SELECT, pueden variar en complejidad desde una simple instrucción que devuelve todas las columnas de una tabla a una instrucción que una múltiples tablas, calcule valores y defina condiciones de búsqueda que restrinja exactamente qué filas de datos se deben devolver. La instrucción SELECT se compone de una serie de cláusulas flexibles que juntas determinan qué datos se recuperarán. En este capítulo aprenderá cómo utilizar cada una de estas cláusulas para realizar tareas básicas de recuperación de datos, definir condiciones de búsqueda, agrupar los resultados de una consulta, especificar un grupo de condiciones de búsqueda y ordenar los resultados de una búsqueda.

Utilice la instrucción SELECT para la recuperación de datos

En el capítulo 5, cuando se analizó acerca de las vistas, se presentó la instrucción SELECT. Como podrá recordar, la instrucción SELECT permite formar consultas intrincadas que pueden devolver exactamente el tipo de datos que se desea recuperar. Es una de las instrucciones más comunes que utilizará como programador de SQL, y también es una de las instrucciones más flexible y amplia en el estándar SQL.

La instrucción SELECT es una expresión de consulta que comienza con la palabra clave SELECT e incluye una serie de elementos que forman la expresión. La sintaxis básica para la instrucción SELECT puede dividirse en varias cláusulas específicas, cada una de las cuales ayuda a refinar la consulta para que sólo se devuelvan los datos requeridos. La sintaxis para la instrucción SELECT puede mostrarse como sigue:

```
SELECT [ DISTINCT | ALL ] { * | <selección de lista> }  
FROM <tabla de referencia> [ { , <tabla de referencia> }... ]  
[ WHERE <condición de búsqueda> ]  
[ GROUP BY <especificación de agrupación> ]  
[ HAVING <condición de búsqueda> ]  
[ ORDER BY <condición de orden> ]
```

Como se puede observar, las únicas cláusulas requeridas son la cláusula `SELECT` y la cláusula `FROM`. Todas las demás cláusulas son opcionales.

Las cláusulas `FROM`, `WHERE`, `GROUP BY` y `HAVING` hacen referencia como expresiones de tabla. Esta porción de la instrucción `SELECT` siempre se evalúa en el orden indicado en la sintaxis. El resultado de esa evaluación es una tabla virtual que se utiliza en la evaluación posterior. En otras palabras, los resultados de la primera cláusula evaluada se utilizan en la siguiente cláusula. Los resultados de esa cláusula se utilizan en la siguiente cláusula, hasta que cada cláusula en la expresión de la tabla se evalúa. Por ejemplo, la primera cláusula a evaluar en la instrucción `SELECT` es la cláusula `FROM`. Ya que esta cláusula es necesaria, siempre es la primera cláusula evaluada. Los resultados de la cláusula `FROM` se utilizan en la cláusula `WHERE`, si se especifica la cláusula `WHERE`. Si no se especifica la cláusula, entonces los resultados de la cláusula `FROM` se utilizan en la siguiente cláusula especificada, ya sea la cláusula `GROUP BY` o la cláusula `HAVING`. Una vez que se evalúa la cláusula final en la expresión de tabla, los resultados se utilizan en la cláusula `SELECT`. Después de evaluar la cláusula `SELECT`, se evalúa la cláusula `ORDER BY`.

Para resumir todo esto, las cláusulas de la instrucción `SELECT` se aplican en el siguiente orden:

- Cláusula `FROM`
- Cláusula `WHERE` (opcional)
- Cláusula `GROUP BY` (opcional)
- Cláusula `HAVING` (opcional)
- Cláusula `SELECT`
- Cláusula `ORDER BY` (opcional)

Es importante tener un entendimiento básico del orden de evaluación al crear instrucciones `SELECT` más complejas, especialmente cuando se trabaja con uniones y subconsultas (discutidas en los capítulos 11 y 12, respectivamente). Este entendimiento es también útil cuando se analiza cada cláusula individualmente, ya que explica cómo se relaciona una cláusula con otras cláusulas. Como resultado, es una buena idea mantener este orden de evaluación en mente a lo largo de este capítulo y en los capítulos siguientes que se basan en diversos aspectos de la instrucción `SELECT`.

La cláusula `SELECT` y la cláusula `FROM`

Ahora que tiene una visión general básica de cómo la instrucción `SELECT` se ejecuta, echemos un vistazo más de cerca a la cláusula `SELECT` y la cláusula `FROM`, las dos cláusulas requeridas en la instrucción. Se analizarán las otras cláusulas en secciones separadas durante el resto del capítulo.

Empecemos con la cláusula `SELECT`. La cláusula `SELECT` incluye las palabras clave `DISTINCT` y `ALL`. La palabra clave `DISTINCT` se utiliza si se desean eliminar filas duplicadas de los resultados de la consulta, y la palabra clave `ALL` se utiliza si se quieren devolver todas las filas de los resultados de una consulta. Por ejemplo, suponga que la base de datos incluye una tabla llamada `CD_INTERPRETE`. La tabla incluye las columnas `NOMBRE_INTERPRETE` y `NOMBRE_CD`. Ya que un CD puede incluir más de un intérprete, el nombre del CD puede aparecer más de una vez en la tabla. Ahora suponga que desea consultar la tabla sólo por el nombre de los CD, pero

no quiere los nombres repetidos. Puede utilizar la palabra clave `DISTINCT` para asegurar que la consulta devuelva el nombre de cada CD sólo una vez, o puede utilizar la palabra clave `ALL` para especificar que se devuelvan todas las filas, aun si hay filas duplicadas. Si no se especifica ninguna de las palabras clave, se toma la palabra clave `ALL`.

Pregunta al experto

P: Se afirma que se puede utilizar un asterisco para incluir todas las columnas en el resultado de una consulta. ¿Presenta esto algún problema si el número de columnas cambia?

R: Sí, esto puede presentar un problema. De hecho, generalmente se recomienda que se utilice el asterisco sólo cuando se accede a una base de datos SQL a través de una invocación directa. Si se utiliza el asterisco en SQL incrustado y el número de columnas cambia, puede encontrar que la aplicación ya no responda correctamente, ya que el programa de aplicación se codificó para esperar una respuesta específica. Si se utiliza un asterisco y se espera que columnas específicas se devuelvan, entonces puede encontrarse con una serie de sorpresas si la base de datos ha cambiado. Por este motivo, se debe evitar el asterisco a menos que se invoque directamente la instrucción `SELECT`. Sin embargo, en el caso de la invocación directa, el asterisco es una manera práctica de devolver todas las columnas sin tener que especificar el nombre de cada una. De hecho, muchos de los ejemplos en el capítulo utilizan el asterisco para evitar el tener que repetir los nombres de las columnas innecesariamente.

Además de las palabras clave `DISTINCT` y `ALL`, la cláusula `SELECT` incluye el asterisco (*) y el marcador de posición <selección de lista>. Se debe especificar una de estas opciones en la cláusula. Si se especifica el asterisco, todas las columnas aplicables se incluyen en el resultado de la consulta.

Si no se especifica el asterisco en la cláusula `SELECT`, se debe especificar cada columna tal como se deriva del origen. El marcador de posición <selección de lista> se puede desglosar en la siguiente sintaxis:

```
<columna derivada> [ [AS] <nombre de la columna> ]
[ { , <columna derivada> [ [AS] <nombre de la columna> } ... ]
```

Echemos un vistazo a la primera línea de la sintaxis. (La segunda línea no es más que una repetición, tantas veces como sea necesario, de la primera línea.) El marcador de posición <columna derivada> en la mayoría de los casos se refiere al nombre de la columna en la tabla origen. Si más de una columna se especifica, entonces se deben separar por comas. Sin embargo, el marcador de posición <columna derivada> también puede referirse a una columna o conjunto de columnas que son de alguna manera parte de una expresión. Por ejemplo, en el capítulo 5 se analizó la función `AVG`, que saca el promedio de los valores en una columna específica. El ejemplo que se muestra en este capítulo utiliza la instrucción `SELECT` para consultar datos de la tabla `COMISIONES_EMPLEADO`, que enumera la cantidad total de comisiones hechas por cada empleado durante un

periodo de tres años. La instrucción `SELECT` promedia los valores en tres diferentes columnas, como se muestra en la siguiente instrucción `SELECT`:

```
SELECT AVG(AÑO_1999), AVG(AÑO_2000), AVG(AÑO_2001)
FROM COMISIONES_EMPLEADO;
```

En este caso, hay tres expresiones que se utilizan para el marcador de posición <columna derivada>: `AVG(AÑO_1999)`, `AVG(AÑO_2000)` y `AVG(AÑO_2001)`. Observe que la expresión de cada columna derivada se separa por una coma, como sería el caso si cada valor fuera simplemente un nombre de columna. El siguiente ejemplo muestra la misma instrucción `SELECT` como en el ejemplo anterior, excepto que se utilizan sólo los nombres de las columnas como columnas derivadas:

```
SELECT AÑO_1999, AÑO_2000, AÑO_2001
FROM COMISIONES_EMPLEADO;
```

Si se fuera a ejecutar esta instrucción `SELECT`, la consulta devolvería todos los valores en las tres columnas, en lugar de promediar esos valores.

La cláusula `SELECT` también permite proporcionar un nombre de columna para cada columna derivada. Para hacer esto, se añade la palabra clave `AS` y el nuevo nombre de columna después de la columna derivada, como se muestra en el siguiente ejemplo:

```
SELECT AVG(AÑO_1999) AS PROMEDIO_1999
FROM COMISIONES_EMPLEADO;
```

En esta instrucción `SELECT`, el valor que se devuelve de la columna `AÑO_1999` se coloca en la columna llamada `PROMEDIO_1999`. Éste es el nombre de la columna que se devuelve como parte de una tabla virtual en los resultados de la consulta. Si no se especifica la subcláusula `AS`, el nombre de columna en la tabla virtual es el mismo que el nombre de columna en la tabla origen. Si un nombre de columna no se puede heredar naturalmente (por ejemplo, cuando se agregan dos valores de columna), se debe utilizar la subcláusula `AS`.

Observe que en los ejemplos anteriores la cláusula `FROM` se utiliza para especificar la tabla (`COMISIONES_EMPLEADO`) que contiene las columnas mencionadas en la cláusula `SELECT`. La cláusula `FROM` incluye la palabra clave `FROM` y una o más tablas de referencia. Si hay varias tablas de referencia, deben separarse utilizando comas. En la mayoría de los casos, la tabla de referencia es el nombre de una tabla o de tablas unidas, aunque puede también ser un tipo de subconsultas. Se discutirá la unión de tablas en el capítulo 11 y las subconsultas en el capítulo 12. Para este capítulo, la cláusula `FROM` se utiliza principalmente para hacer referencia a nombres de tablas, como se define la cláusula en los dos ejemplos anteriores (donde <tabla de referencia> es igual a `COMISIONES_EMPLEADO`).

Juntas las cláusulas `SELECT` y `FROM` forman las bases para la instrucción `SELECT`, que puede ser tan simple como consultar cada fila y cada columna de una tabla, como se muestra en el siguiente ejemplo:

```
SELECT * FROM INTERPRETES;
```

En esta instrucción, se especifica que cada columna de la tabla `INTERPRETES` se debe devolver. Además, cada fila será devuelta, ya que no se especificaron otras cláusulas.

ID_INTERPRETE: INT	NOMBRE_INTERPRETE: VARCHAR(60)	LUGAR_DE_NACIMIENTO: VARCHAR(60)
2001	Jennifer Warnes	Seattle, Washington, USA
2002	Joni Mitchell	Fort MacLeod, Alberta, Canada
2003	William Ackerman	Alemania
2004	Kitaro	Toyohashi, Japon
2005	Bing Crosby	Tacoma, Washington, Estados Unidos
2006	Patsy Cline	Winchester, Virginia, Estados Unidos
2007	Jose Carreras	Barcelona, España
2008	Luciano Pavarotti	Modena, Italia
2009	Placido Domingo	Madrid, España

Figura 7-1 Las columnas ID_INTERPRETE, NOMBRE_INTERPRETE y LUGAR_DE_NACIMIENTO de la tabla INTERPRETES.

Echemos un vistazo más de cerca de esto. La tabla INTERPRETES incluye las columnas ID_INTERPRETE, NOMBRE_INTERPRETE y LUGAR_DE_NACIMIENTO, como se muestra en la figura 7-1.

Si se ejecuta la instrucción SELECT mostrada en el ejemplo anterior, los resultados de la consulta serían similares a los siguientes:

ID_INTERPRETE	NOMBRE_INTERPRETE	LUGAR_DE_NACIMIENTO
-----	-----	-----
2001	Jennifer Warnes	Seattle, Washington, Estados Unidos
2002	Joni Mitchell	Fort MacLeod, Alberta, Canadá
2003	William Ackerman	Alemania
2004	Kitaro	Toyohashi, Japón
2005	Bing Crosby	Tacoma, Washington, Estados Unidos
2006	Patsy Cline	Winchester, Virginia, Estados Unidos
2007	Jose Carreras	Barcelona, España
2008	Luciano Pavarotti	Modena, Italia
2009	Placido Domingo	Madrid, España

Observe que cada fila y cada columna se devuelven en los resultados de la consulta. Si utiliza el asterisco en la cláusula SELECT, no se tienen que especificar los nombres de columnas.

Ahora suponga que desea devolver sólo las columnas NOMBRE_INTERPRETE y LUGAR_DE_NACIMIENTO. Se puede modificar la instrucción SELECT para verse como la siguiente:

```
SELECT NOMBRE_INTERPRETE AS NAME, LUGAR_DE_NACIMIENTO
FROM INTERPRETES;
```

Los resultados de la consulta ahora contienen sólo dos columnas, como se muestran a continuación:

NOMBRE	LUGAR_DE_NACIMIENTO
Jennifer Warnes	Seattle, Washington, Estados Unidos
Joni Mitchell	Fort MacLeod, Alberta, Canadá
William Ackerman	Alemania
Kitaro	Toyohashi, Japón
Bing Crosby	Tacoma, Washington, Estados Unidos
Patsy Cline	Winchester, Virginia, Estados Unidos
Jose Carreras	Barcelona, España
Luciano Pavarotti	Modena, Italia
Placido Domingo	Madrid, España

Observe que el nombre de la primera columna es NOMBRE, en lugar de NOMBRE_INTERPRETE. Esto es debido a que la subcláusula AS (especificada en NOMBRE) se define como parte de la columna derivada NOMBRE_INTERPRETE. Si se fuera a especificar la palabra clave DISTINCT en esta situación particular, aún recibirían el mismo número de filas, aunque podrían no ser devueltas en el mismo orden en que estaban cuando no se utilizaba la palabra clave, dependiendo de la aplicación SQL. La razón por la que la palabra clave DISTINCT no haría ninguna diferencia en los resultados de la consulta es porque no hay filas duplicadas en la tabla. Sin embargo, utilizar la palabra clave DISTINCT puede afectar la ejecución, particularmente si el RDBMS tiene que ordenar a través de un gran número de filas, así que asegúrese de utilizar la palabra clave sólo cuando sea necesario.

Ahora veamos un ejemplo que utilice la palabra clave DISTINCT. Suponga que la base de datos incluye una tabla que empareje a los intérpretes con los tipos de música, como se muestra en la figura 7-2.

NOMBRE_INTERPRETE: VARCHAR(60)	TIPO_INTERPRETE: VARCHAR(10)
Jennifer Warnes	Folk
Jennifer Warnes	Pop
Joni Mitchell	Pop
Joni Mitchell	Folk
Joni Mitchell	Jazz
William Ackerman	New Age
Kitaro	New Age
Kitaro	International

Figura 7-2 Columnas NOMBRE_INTERPRETE y TIPO_INTERPRETE de la tabla TIPO_INTERPRETE.

Si la instrucción `SELECT` incluye ambas (todas) columnas en la cláusula `SELECT`, como se muestra en el siguiente ejemplo, la consulta devolvería todas las filas:

```
SELECT * FROM TIPO_INTERPRETE;
```

No importa si se especifica la palabra clave `DISTINCT` en este caso, ya que los resultados de la consulta no incluyen filas duplicadas. Los resultados serían los mismos si se incluyera la palabra clave `ALL`, en lugar de `DISTINCT`, o si no se especifican ninguno de los dos calificadores. En ambos casos, los resultados de la consulta incluyen la misma información que se muestra en la tabla de la figura 7-2.

Ahora echemos un vistazo a la misma instrucción, sólo que esta vez se especifica la palabra clave `DISTINCT` y sólo una de las dos columnas:

```
SELECT DISTINCT NOMBRE_INTERPRETE  
FROM TIPO_INTERPRETE;
```

Observe que esta instrucción incluye sólo la columna `NOMBRE_INTERPRETE`, que incluye valores duplicados. Mediante el uso de la palabra clave `DISTINCT`, los resultados de la consulta incluirían sólo un ejemplo de cada uno de los valores. Si se ejecuta la instrucción `SELECT` en el ejemplo anterior, los resultados de la consulta serían similares a los siguientes:

```
NOMBRE_INTERPRETE  
-----  
Jennifer Warnes  
Joni Mitchell  
Kitaro  
William Ackerman
```

Aunque hay siete filas en la tabla `TIPO_INTERPRETE`, sólo se devuelven cuatro filas, ya que existen sólo cuatro valores únicos en la columna `NOMBRE_INTERPRETE` y los demás valores son duplicados.

Como se puede observar, las cláusulas `SELECT` y `FROM` son bastante sencillas, al menos en este nivel de codificación. Una vez que lleguemos a estructuras más complejas, encontrará que ambas cláusulas pueden ser más complicadas. Sin embargo, lo importante para recordar ahora es que estas cláusulas actúan como base para el resto de la instrucción `SELECT`. En términos de ejecución, la instrucción `SELECT`, para todos los efectos prácticos, comienza con la cláusula `FROM` y termina con la cláusula `SELECT`. (La cláusula `ORDER BY` se utiliza principalmente para mostrar los objetivos, y de hecho no afecta la información devuelta. La cláusula `ORDER BY` se discute con mayor detalle en la sección “Utilice la cláusula `ORDER BY` para ordenar los resultados de una consulta”, más adelante en este capítulo.)

Utilice la cláusula `WHERE` para definir condiciones de búsqueda

La siguiente cláusula en la instrucción `SELECT` es la cláusula `WHERE`. La cláusula `WHERE` toma los valores devueltos por la cláusula `FROM` (en la tabla virtual) y aplica la condición de búsqueda que se define en la cláusula `WHERE`. La cláusula `WHERE` actúa como filtro sobre los resultados devueltos por la cláusula `FROM`. Cada fila se evalúa contra las condiciones de búsqueda.

da. Las filas que se evalúan como verdaderas se devuelven como parte del resultado de la consulta. Aquellas que se evalúan como desconocidas o falsas no se incluyen en los resultados.

Para una mejor comprensión de cómo se evalúa cada fila, echemos un vistazo de cerca al marcador de posición <condición de búsqueda>. La condición de búsqueda se compone de uno o más predicados que se utilizan para poner a prueba el contenido devuelto por la cláusula FROM. Un *predicado* es una expresión de SQL que define un hecho acerca de cualquier fila devuelta por la instrucción SELECT. Ya se vieron ejemplos de predicados en los capítulos 4 y 5. Por ejemplo, un ejemplo de definición de una vista (en el capítulo 5) incluye la siguiente instrucción SELECT:

```
SELECT TITULO_CD, DERECHOSDEAUTOR, EN_EXISTENCIA
FROM INVENTARIO_DISCO_COMPACTO
WHERE DERECHOSDEAUTOR > 1989 Y DERECHOSDEAUTOR < 2000;
```

Esta instrucción consulta tres columnas en la tabla INVENTARIO_DISCO_COMPACTO. La cláusula SELECT especifica las columnas que se devuelven, y la cláusula FROM especifica la tabla origen. La cláusula WHERE determina qué filas (basado en la cláusula FROM) se incluyen en los resultados. En este caso, la cláusula WHERE contiene dos predicados que se conectan por la palabra clave AND. El primer predicado (DERECHOSDEAUTOR > 1989) especifica que todas las filas incluidas en los resultados de la consulta deben contener un valor mayor que 1989 en la columna DERECHOSDEAUTOR. El segundo predicado (DERECHOSDEAUTOR < 2000) especifica que todas las filas incluidas en los resultados de la consulta deben contener un valor menor que 2000 en la columna DERECHOSDEAUTOR.

Conforme se evalúan las filas, cada predicado se evalúa sobre una base individual para determinar si la fila cumple con la condición definida por ese predicado. Regresando al último ejemplo, el primer predicado establece la condición de que los valores deben ser mayores que 1989. Si el valor DERECHOSDEAUTOR para una fila en particular es mayor que 1989, la condición se cumple y el predicado evalúa como verdadero. Si el valor no es mayor que 1989, el predicado evalúa como falso. Si SQL no puede determinar si el valor cumple o no la condición (como sería el caso si el valor es nulo), el predicado evalúa como desconocido.

Cada predicado evalúa como verdadero, falso o desconocido. Si se incluye más de un predicado en la cláusula WHERE, éstos se unen por la palabra clave OR o por la palabra clave AND. Si se utiliza OR, entonces al menos uno de los predicados de cualquier lado de OR debe evaluar como verdadero para que la fila pase el filtro, y por lo tanto aparece en los resultados de la consulta. Si se utiliza AND, entonces los predicados de cualquier lado deben evaluar como verdaderos para que la fila pase el filtro. Por ejemplo, la cláusula WHERE en el último ejemplo incluye dos predicados que se conectan por la palabra clave AND. Esto significa que el primer predicado debe evaluar como verdadero y el segundo predicado debe evaluar como verdadero. Si se utiliza OR en lugar de AND, entonces sólo uno de los predicados debe evaluarse como verdadero, que carece un poco de sentido en este caso, ya que todos los valores, excepto los valores nulos, están por encima de 1989 o por debajo de 2000.

Por último, la cláusula WHERE en su conjunto debe evaluar como verdadero a fin de que una fila se incluya en los resultados de una consulta. Si la cláusula WHERE incluye más de un predicado, SQL sigue las directrices específicas para la forma en que una instrucción en conjunto se evalúa. Empecemos viendo la palabra clave OR. La tabla 7-1 enumera la evaluación de una condición de búsqueda si la palabra clave OR se utiliza para separar dos predicados. Para usar la tabla, em-

	Verdadero	Falso	Desconocido
Verdadero	Verdadero	Verdadero	Verdadero
Falso	Verdadero	Falso	Desconocido
Desconocido	Verdadero	Desconocido	Desconocido

Tabla 7-1 Evaluación de los predicados conectados por OR.

pareje una condición de la columna de la izquierda a una condición en la fila superior. El resultado (donde una fila y una columna se intersectan para formar una celda) muestra cómo la condición de búsqueda se evalúa sobre la base de cómo se evalúa cada predicado.

Como muestra la tabla, si ambos predicados evalúan como verdadero, entonces la condición de búsqueda evalúa como verdadero. Si ambos son falsos, entonces la condición de búsqueda evalúa como falso. Se proporciona una condición para cada posible encuentro. Por ejemplo, suponga que la instrucción `SELECT` incluye la siguiente cláusula `WHERE`:

```
WHERE TIPO_INTERPRETE = 'Folk' OR TIPO_INTERPRETE = 'Jazz'
```

Ahora suponga que el primer predicado en este ejemplo (`TIPO_INTERPRETE = 'Folk'`) evalúa como verdadero y el segundo predicado (`TIPO_INTERPRETE = 'Jazz'`) evalúa como falso. Esto significa que la fila evaluada contiene el valor `Folk` en la columna `TIPO_INTERPRETE`, pero no contiene el valor `Jazz` en esa columna. Ahora remítase a la tabla 7-1. Si selecciona Verdadero de la primera columna, selecciona Falso de la fila superior, y luego empareja estos dos valores (buscando dónde se intersectan), se puede observar que la condición de búsqueda evalúa como verdadero; por lo tanto, la fila se incluye en los resultados de la consulta.

Se puede hacer lo mismo con la palabra clave `AND` como se hizo con la palabra clave `OR`.

De nuevo, sólo tiene que coincidir cómo se evalúa cada predicado para determinar si la condición de búsqueda se evaluará como verdadera, falsa o desconocida. Recuerde que la condición de búsqueda debe evaluar como verdadero para que la fila se incluya en los resultados de la consulta. Como se puede observar, la palabra clave `AND` es mucho menos indulgente que la palabra clave `OR`. La única manera para que la condición de búsqueda evalúe como verdadero es que ambos predicados evalúen como verdaderos.

	Verdadero	Falso	Desconocido
Verdadero	Verdadero	Falso	Desconocido
Falso	Falso	Falso	Falso
Desconocido	Desconocido	Falso	Desconocido

Tabla 7-2 Evaluación de los predicados conectados por AND.

NOTA

La comparación de los operadores y predicados en general se discute con mayor detalle en el capítulo 9.

Si una condición de búsqueda incluye más de dos predicados, los predicados se evalúan de acuerdo con un orden escogido por el RDBMS, a menos que se utilicen paréntesis para separar las combinaciones de predicados. Mientras que el estándar SQL no especifica el orden en que múltiples predicados se evalúan, la mayoría de los productos RDBMS evalúan AND antes que OR. Por ejemplo, se tiene la instrucción SELECT que incluye la siguiente cláusula WHERE:

```
WHERE EN_EXISTENCIA = 6 OR EN_EXISTENCIA = 27 AND ID_DISQUERA = 833 OR  
ID_DISQUERA = 829
```

Observe que hay cuatro predicados en esta cláusula y no hay paréntesis. Asumiendo que se evalúa AND antes que OR, la anterior cláusula WHERE se evaluaría como si se escribiera de esta manera:

```
WHERE EN_EXISTENCIA = 6 OR (EN_EXISTENCIA = 27 AND ID_DISQUERA = 833) OR  
ID_DISQUERA = 829
```

Con el fin de evaluar como verdadero, una fila debe contener uno de los siguientes valores o conjunto de valores:

- EN_EXISTENCIA valor de 6
- EN_EXISTENCIA valor de 27 y ID_DISQUERA valor de 833
- ID_DISQUERA valor de 829

Cuando se incluyen ambas palabras clave AND y OR en la misma cláusula WHERE, *siempre* es una buena idea incluir paréntesis para asegurar que se reciba el filtrado intentado, teniendo en cuenta que los predicados entre paréntesis se evalúan siempre primero. Si los RDBMS hacen otras suposiciones, o si los paréntesis se utilizan alrededor de otros conjuntos de predicados, los resultados serán diferentes de lo que se ha visto. Por ejemplo, suponga que utiliza los paréntesis de la siguiente manera:

```
WHERE (EN_EXISTENCIA = 6 O EN_EXISTENCIA = 27) AND (ID_DISQUERA = 833 O  
ID_DISQUERA = 829)
```

Los predicados primero se evalúan dentro del contexto de los paréntesis y luego se comparan con otros predicados en consecuencia. En este caso, una fila debe contener uno de los dos valores EN_EXISTENCIA y la fila debe contener uno de los dos valores ID_DISQUERA. Como resultado, una fila debe contener uno de los siguientes conjuntos de valores para evaluar como verdadero:

- Valor de EN_EXISTENCIA de 6 y valor de ID_DISQUERA de 833
- Valor de EN_EXISTENCIA de 6 y valor de ID_DISQUERA de 829
- Valor de EN_EXISTENCIA de 27 y valor de ID_DISQUERA de 833
- Valor de EN_EXISTENCIA de 27 y valor de ID_DISQUERA de 829

NOTA

SQL incluye tres operadores que se pueden utilizar si una condición de búsqueda se vuelve demasiado complicada. Estos operadores son IS TRUE, IS FALSE e IS UNKNOWN. Por ejemplo, se puede especificar la siguiente condición de búsqueda: (PRIMER_NOMBRE = 'Joni' AND APELLIDO = 'Mitchell') IS TRUE. Esto significa que el valor PRIMER_NOMBRE de la fila devuelta debe ser Joni y el valor APELLIDO debe ser Mitchell. En otras palabras, deben evaluar como verdadero. Si se especifica IS FALSE en esta situación, el par de predicado habría resultado como falso, lo que significa que al menos uno de los dos predicados es falso (podría no ser Joni o podría no ser Mitchell).

Otra palabra clave que puede resultar útil es la palabra clave NOT, que puede utilizarse sola o junto con las palabras clave AND y OR para especificar el inverso de un predicado. Por ejemplo, la instrucción SELECT puede incluir la siguiente cláusula WHERE:

```
WHERE NOMBRE_INTERPRETE = 'Joni Mitchell' OR NOT NOMBRE_INTERPRETE =
'Kitaro'
```

En este caso, el valor NOMBRE_INTERPRETE puede ser Joni Mitchell o puede ser cualquier valor distinto de Kitaro. Por supuesto, Joni Mitchell no es igual que Kitaro, de modo que el predicado es redundante y se obtiene el mismo resultado si se elimina. Además, se obtendría el mismo resultado si se utiliza el operador de comparación mayor o menor que (<>), por lo que la cláusula completa WHERE se puede reescribir de manera más simple como:

```
WHERE NOMBRE_INTERPRETE <> 'Kitaro'
```

Defina la cláusula WHERE

Ahora que tiene una visión general de cómo definir la cláusula WHERE, pongamos juntas las cláusulas SELECT y FROM y veamos algunos ejemplos. Los ejemplos que veremos se basan en la tabla INVENTARIO, mostrada en la figura 7-3. La tabla INVENTARIO contiene cinco columnas, algunas de las cuales utilizaremos para definir nuestras condiciones de búsqueda.

El primer ejemplo que veremos incluye la cláusula WHERE, que define qué filas se pueden devolver basadas en los valores EN_EXISTENCIA:

```
SELECT * FROM INVENTARIO
WHERE EN_EXISTENCIA < 20;
```

Si ejecuta esta instrucción, los resultados serán similares a los siguientes:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
99301	Famous Blue Raincoat	1991	16.99	6
99303	Court and Spark	1974	14.99	18
99304	Past Light	1983	15.99	2
99305	Kojiki	1990	15.99	5
99306	That Christmas Feeling	1993	10.99	3

ID_DISCO_COMPACTO: INT	TITULO_CD: VARCHAR(60)	DERECHOSDEAUTOR: INT	PRECIO_MENUDEO: NUMERICO(5,2)	EN_EXISTENCIA: INT
99301	Famous Blue Raincoat	1991	16.99	6
99302	Blue	1971	14.99	26
99303	Court and Spark	1974	14.99	18
99304	Past Light	1983	15.99	2
99305	Kojiki	1990	15.99	5
99306	That Christmas Feeling	1993	10.99	3
99307	Patsy Cline: 12 Greatest Hits	1988	16.99	25

Figura 7-3 Tabla INVENTARIO que contiene datos relacionados con el CD.

Como se puede observar, todas menos dos filas se incluyen en los resultados de la consulta. Las filas no incluidas contienen valores EN_EXISTENCIA mayores que 20. En otras palabras, esas dos filas evalúan como falso.

Ahora tomemos la misma instrucción SELECT y refinemos la cláusula WHERE aún más. En la nueva instrucción, la cláusula WHERE incluye dos predicados que se conectan por la palabra clave AND, como se muestra en el siguiente ejemplo:

```
SELECT * FROM INVENTARIO
WHERE EN_EXISTENCIA < 20 AND PRECIO_MENUDEO < 15.00;
```

Cuando se ejecuta esta instrucción, se reciben los siguientes resultados:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
-----	-----	-----	-----	-----
99303	Court and Spark	1974	14.99	18
99306	That Christmas Feeling	1993	10.99	3

Observe que sólo dos filas cumplen la condición de búsqueda. En otras palabras, sólo estas dos filas tienen un valor EN_EXISTENCIA menor que 20 y un valor PRECIO_MENUDEO menor que 15.00. Debido a que se utiliza la palabra clave AND, ambos predicados deben evaluar como verdadero, que se aplica para estas dos filas.

Ahora hagamos una pequeña modificación a la instrucción SELECT. En la cláusula WHERE, se cambió la palabra clave AND por las palabras clave AND NOT, como se muestra en el siguiente ejemplo:

```
SELECT * FROM INVENTARIO
WHERE EN_EXISTENCIA < 20 AND NOT PRECIO_MENUDEO < 15.00;
```


La palabra clave NOT cambia los resultados de la consulta. Como se puede observar, tres filas se devuelven:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
99301	Famous Blue Raincoat	1991	16.99	6
99304	Past Light	1983	15.99	2
99305	Kojiki	1990	15.99	5

Cada una de las filas devueltas contiene un valor EN_EXISTENCIA menor que 20 y un valor PRECIO_MENUDEO que *no* es menor que 15.00, o 15.00 o mayor.

Como siguiente paso veamos la misma instrucción SELECT, sólo que en esta ocasión los dos predicados se conectan por la palabra clave OR, como se muestra en el siguiente ejemplo:

```
SELECT * FROM INVENTARIO
WHERE EN_EXISTENCIA < 20 OR PRECIO_MENUDEO < 15.00;
```

Los resultados de la consulta para esta instrucción incluyen muchas más filas que cuando se utilizó la palabra clave AND. Por su naturaleza, la palabra clave OR permite que existan mayores oportunidades para que la búsqueda de la cláusula evalúe como verdadera. Como se puede observar, ahora se devuelven seis filas:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
99301	Famous Blue Raincoat	1991	16.99	6
99302	Blue	1971	14.99	26
99303	Court and Spark	1974	14.99	18
99304	Past Light	1983	15.99	2
99305	Kojiki	1990	15.99	5
99306	That Christmas Feeling	1993	10.99	3

Cada fila en los resultados de la búsqueda contiene un valor EN_EXISTENCIA menor que 20 o un valor PRECIO_MENUDEO menor que 15.00. Debido a que la palabra clave OR se utiliza, sólo uno de los predicados necesita evaluar como verdadero, aunque es aceptable si ambos predicados evalúan como verdadero.

En el siguiente ejemplo se añade un predicado más que limita las filas devueltas a aquellas con un valor EN_EXISTENCIA mayor que 5, puesto entre paréntesis para hacerlo claro:

```
SELECT * FROM INVENTARIO
WHERE (EN_EXISTENCIA < 20 AND EN_EXISTENCIA > 5) OR PRECIO_MENUDEO <
15.00;
```

Para devolver una fila, el valor EN_EXISTENCIA debe estar entre el rango de 5 y 20 o el valor PRECIO_MENUDEO debe ser menor que 15.00. Los resultados de la consulta de esta instrucción SELECT serían como sigue:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
99301	Famous Blue Raincoat	1991	16.99	6
99302	Blue	1971	14.99	26
99303	Court and Spark	1974	14.99	18
99306	That Christmas Feeling	1993	10.99	3

Ahora hagamos un cambio más a la cláusula WHERE. Suponga que desea que el valor EN_EXISTENCIA sea menor que 20 y mayor que 5 o el valor EN_EXISTENCIA sea menor que 20 y el valor PRECIO_MENUDEO sea menor que 15. Una forma de hacer esto es colocando paréntesis en los últimos dos predicados:

```
SELECT * FROM INVENTARIO
  WHERE EN_EXISTENCIA < 20 AND (EN_EXISTENCIA > 5 OR PRECIO_MENUDEO <
15.00) ;
```

Los resultados recibidos en esta ocasión son algo diferentes ya que la fila Blue ya no evalúa como verdadero:

ID_DISCO_COMPACTO	TITULO_CD	DERECHOSDEAUTOR	PRECIO_MENUDEO	EN_EXISTENCIA
99301	Famous Blue Raincoat	1991	16.99	6
99303	Court and Spark	1974	14.99	18
99306	That Christmas Feeling	1993	10.99	3

Con la combinación de predicados se pueden crear muchas condiciones de búsqueda que permitan devolver exactamente los datos necesarios. La clave para escribir condiciones de búsqueda eficaces es una comprensión profunda de los predicados y los operadores utilizados para formar esos predicados. El capítulo 9 muestra muchos de los operadores que se pueden utilizar y los tipos de predicados que se pueden crear. Con esa información se pueden crear condiciones eficaces y concisas de búsqueda.

Utilice la cláusula GROUP BY para agrupar los resultados de una consulta

La siguiente cláusula en la instrucción SELECT es la cláusula GROUP BY. La cláusula GROUP BY tiene una función muy diferente de la cláusula WHERE. Como su nombre lo indica, la cláusula GROUP BY se utiliza para agrupar tipos de información con el fin de resumir datos relacionados. La cláusula GROUP BY se puede incluir en la instrucción SELECT aun si la cláusula WHERE se utiliza o no.

Como se vio en la sección “Utilice la instrucción SELECT para la recuperación de datos”, la sintaxis para la cláusula GROUP BY, como aparece en la sintaxis de la instrucción SELECT, se ve como se muestra a continuación:

```
[ GROUP BY <especificaciones de grupo> ]
```

Sin embargo, el marcador de posición <especificaciones de grupo> se puede dividir en elementos más pequeños:

```
<nombre de columna>[ {,<nombre de columna> }... ]
|{ ROLLUP | CUBE }( <nombre de columna> [ { , <nombre de columna> }... ] )
```

En realidad, la sintaxis <especificaciones de grupo>, como algunas de las otras sintaxis en este libro, es incluso más compleja de la que se presenta aquí; sin embargo, para propósitos de este capítulo, esta sintaxis proveerá los detalles necesarios para usar la cláusula GROUP BY de manera efectiva.

Ahora veamos la sintaxis. La primera línea se explica por sí sola. Se especifican uno o más nombres de columna que contengan valores que se agrupan juntos. Esto normalmente aplica a columnas que representan algunos tipos de categorías cuyos valores se repiten dentro de la tabla. Por ejemplo, la base de datos puede incluir una tabla que enumere a los empleados en la organización. Suponga que la tabla incluye el título del puesto para cada empleado. Posiblemente encuentre que desea agrupar la información en la tabla por el título del puesto, con una fila en el conjunto de resultados para cada valor del título del puesto, quizá para determinar ciertas cosas como el salario promedio de cada puesto o el número de empleados que tiene cada título del puesto. Si se necesita especificar más de un nombre de columna, asegúrese de separarlos con una coma seguida de cada nombre (excepto el último).

En relación con la sintaxis, se puede especificar la segunda línea en lugar de la primera. En este caso, se puede usar la palabra clave `ROLLUP` o `CUBE`, junto con el listado de los nombres de columna, entre paréntesis. De nuevo, asegúrese de separar los nombres de columna con comas. Con respecto a las palabras clave `ROLLUP` y `CUBE`, la mejor manera de entender estos operadores es mediante el uso de ejemplos. De hecho, la mejor forma de entender totalmente la cláusula `GROUP BY` es a través de ejemplos. Sin embargo, antes de entrar en esto, echemos un vistazo a la tabla en la cual se basan los ejemplos. La figura 7-4 muestra la tabla `EXISTENCIA_DISCO_COMPACTO`, que contiene una lista de CD, si son interpretados o instrumentales, el precio, y cuántos de cada título hay actualmente en existencia.

Ahora podemos empezar con los ejemplos. En el primero que veremos se usará la cláusula `GROUP BY` para agrupar filas basadas en la columna `CATEGORIA` de la tabla `EXISTENCIA_DISCO_COMPACTO`, como se muestra en la siguiente instrucción `SELECT`:

```
SELECT CATEGORIA, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CATEGORIA;
```

Primero echemos un vistazo a la cláusula `GROUP BY`, que especifica qué filas se deben agrupar basadas en la columna `CATEGORIA`. Si observa la figura 7-4, verá que la columna contiene sólo dos valores: Vocal o Instrumental. Como resultado, la instrucción `SELECT` sólo devuelve dos columnas, una para Instrumental y otra para Vocal:

CATEGORIA	TOTAL_A_LA_MANO
Instrumental	78
Vocal	217

Ahora veamos la cláusula `SELECT` en el ejemplo anterior de la instrucción `SELECT`. Observe que la lista seleccionada incluye la función `SUM`, que añade información a la columna `A_LA_MANO`. La columna resultante se llama `TOTAL_A_LA_MANO`. La otra columna incluida en la lista seleccionada es la columna `CATEGORIA`. La lista seleccionada puede incluir sólo esas columnas que se especifican en la cláusula `GROUP BY` o que de alguna manera se pueden resumir.

Lo que hace esta instrucción, entonces, es añadir los valores totales `A_LA_MANO` para cada valor en la columna `CATEGORIA`. En este caso, hay un total de 217 CD en existencia que entran en la categoría de interpretados, y 78 en existencia que entran en la categoría de instrumentales. Si hubiera otra categoría, entonces aparecería otra fila para esa también.

DISCO_COMPACTO: VARCHAR(60)	CATEGORIA: VARCHAR(15)	PRECIO: NUMERICO(5,2)	A_LA_MANO: INT
Famous Blue Raincoat	Vocal	16.99	13
Blue	Vocal	14.99	42
Court and Spark	Vocal	14.99	22
Past Light	Instrumental	15.99	17
Kojiki	Instrumental	15.99	6
That Christmas Feeling	Vocal	14.99	8
Patsy Cline: 12 Greatest Hits	Vocal	16.99	32
Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
After the Rain: The Soft Sounds of Erik Satie	Instrumental	16.99	21
Out of Africa	Instrumental	16.99	29
Leonard Cohen The Best of	Vocal	15.99	12
Fundamental	Vocal	15.99	34
Blues on the Bayou	Vocal	14.99	27
Orlando	Instrumental	14.99	5

Figura 7-4 Información del CD en la tabla EXISTENCIA_DISCO_COMPACTO.

Pregunta al experto

P: ¿Existen consideraciones de ejecución con respecto al uso de la cláusula GROUP BY?

R: Sí, el uso de GROUP BY puede causar problemas de ejecución, ya que los RDBMS por lo general deben realizar un ordenamiento adecuado con el fin de agrupar los grupos de filas, y ordenar un gran número de filas (decenas de miles o más) puede consumir considerables recursos. Pero también esté consciente que la cláusula ORDER BY (presentada más adelante en este capítulo) y la palabra clave DISTINCT también suelen requerir ordenaciones, y por lo tanto tienen consideraciones de ejecución similares. Esto no significa que deba tener miedo de utilizarlos, sino que debe esforzarse por aprender el impacto de ejecución de las instrucciones de SQL para ganar experiencia, y de esa manera estar más capacitado al escribir instrucciones que tengan una mejor ejecución en la implementación del proveedor. Por ejemplo, en Oracle, la cláusula GROUP BY que enumera todas las columnas en la cláusula SELECT es más eficaz que utilizar la palabra clave DISTINCT, y sin embargo los resultados de la consulta de los dos métodos son idénticos.

Como se dijo anteriormente, se puede utilizar la cláusula **WHERE** en la instrucción **SELECT** que incluya la cláusula **GROUP BY**. Por ejemplo, suponga que desea ver los totales sólo de los CD que se vendieron por menos de \$16.00. Para hacer esto, simplemente modifique la instrucción **SELECT** como se muestra a continuación:

```
SELECT CATEGORIA, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
WHERE PRICE < 16.00
GROUP BY CATEGORIA;
```

Los resultados de la consulta de esta instrucción serían ligeramente diferentes si no se incluyera la cláusula **WHERE**:

CATEGORIA	TOTAL_A_LA_MANO
-----	-----
Instrumental	28
Vocal	172

Observe que como los CD que se venden por \$16.00 o más se excluyen, los resultados ahora muestran sólo 28 CD instrumental y 172 CD vocal.

En los dos ejemplos anteriores, la cláusula **GROUP BY** especifica sólo una columna. Sin embargo, se pueden especificar columnas adicionales como sea necesario. Esto permite crear subgrupos que agrupen datos en el ámbito de los grupos principales. Por ejemplo, suponga que desea agrupar datos no sólo por los valores en la columna **CATEGORIA**, sino también de acuerdo con los valores en la columna **PRECIO**. Para hacer esto, se debe incluir la columna **PRECIO** en la lista de selección, así como en la cláusula **GROUP BY**, como se muestra en la siguiente instrucción **SELECT**:

```
SELECT CATEGORIA, PRECIO, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CATEGORIA, PRECIO;
```

Ahora los resultados de la consulta incluyen seis filas, en lugar de dos:

CATEGORIA	PRECIO	TOTAL_A_LA_MANO
-----	-----	-----
Instrumental	14.99	5
Vocal	14.99	99
Instrumental	15.99	23
Vocal	15.99	73
Instrumental	16.99	50
Vocal	16.99	45

Observe que para cada valor en **CATEGORIA**, hay tres filas, una para cada uno de los valores **PRECIO**. Por ejemplo, en el grupo Vocal, hay 99 CD a 14.99, 73 CD a 15.99 y 45 CD a 16.99. El número de filas depende de cuántos valores diferentes existan en las columnas especificadas en la cláusula **GROUP BY**. En este ejemplo, hay dos valores diferentes en la columna **CATEGORIA** y tres valores diferentes en la columna **PRECIO**, lo que significa que seis (dos veces tres) filas se devuelven.

NOTA

El orden en el que los resultados de la consulta se devuelven puede variar de una aplicación a otra. Por ejemplo, algunos productos pueden devolver todas las filas de instrumental juntas, seguidas por todas las filas de los interpretados. Sin embargo, independientemente de cómo aparece la información en la interfaz de usuario, los resultados finales deben ser los mismos. También la mayoría de las aplicaciones permiten añadir la cláusula ORDER BY (que se analiza después en el capítulo) para ordenar las filas resumidas.

Ahora echemos un vistazo a los operadores ROLLUP y CUBE. Ambos operadores son similares en funcionamiento, ya que devuelven datos adicionales en los resultados de la consulta cuando se añade la cláusula GROUP BY. La principal diferencia entre los dos es que el operador CUBE devuelve más información que el operador ROLLUP. Empecemos con el ejemplo del operador ROLLUP para que se pueda demostrar la diferencia.

En la siguiente instrucción SELECT, en la cláusula GROUP BY aplica el operador ROLLUP a las columnas CATEGORIA y PRECIO:

```
SELECT CATEGORIA, PRECIO, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY ROLLUP (CATEGORIA, PRECIO);
```

NOTA

Las implementaciones pueden variar con respecto a cómo soportan a los operadores ROLLUP y CUBE. Por ejemplo, en SQL Server se debe añadir WITH ROLLUP o WITH CUBE al final de la cláusula GROUP BY, en lugar de definir la cláusula en la forma que la especifica el estándar SQL:2006. Asegúrese de verificar la documentación del producto para determinar cómo se respaldan estos operadores.

Ahora cuando se ejecute la instrucción SELECT, los resultados de la consulta incluyen una fila adicional para cada valor en la columna CATEGORIA, además de una fila con el gran total al final:

CATEGORIA	PRECIO	TOTAL_A_LA_MANO
-----	-----	-----
Instrumental	14.99	5
Instrumental	15.99	23
Instrumental	16.99	50
Instrumental	NULL	78
Vocal	14.99	99
Vocal	15.99	73
Vocal	16.99	45
Vocal	NULL	217
NULL	NULL	295

Las dos filas adicionales de CATEGORIA proporcionan los totales para cada valor en la columna CATEGORIA. En el ejemplo anterior, el grupo Instrumental incluye un total de 78 CD, y el grupo Vocal incluye un total de 217 CD. Observe que la columna PRECIO incluye un valor nulo para estas filas en particular. Un valor no se puede calcular para esta columna, ya que los tres subgrupos

(de la columna PRECIO) se representan aquí. La última fila (la fila con NULL para ambas columnas CATEGORIA y PRECIO) contiene el gran total de todos los CD incluidos por la consulta (ambas categorías de grupos y los tres subgrupos de precios).

El operador CUBE devuelve los mismos datos que el operador ROLLUP y algunos otros. Observe que en la siguiente instrucción SELECT, simplemente se reemplazó la palabra clave CUBE por ROLLUP:

```
SELECT CATEGORIA, PRECIO, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CUBE (CATEGORIA, PRECIO);
```

Esta instrucción devuelve los siguientes resultados de la consulta:

CATEGORIA	PRECIO	TOTAL_A_LA_MANO
-----	-----	-----
Instrumental	14.99	5
Instrumental	15.99	23
Instrumental	16.99	50
Instrumental	NULL	78
Vocal	14.99	99
Vocal	15.99	73
Vocal	16.99	45
Vocal	NULL	217
NULL	NULL	295
NULL	14.99	104
NULL	15.99	96
NULL	16.99	95

Se puede observar que tres filas adicionales se añaden a los resultados de la consulta, una fila para cada valor diferente en la columna PRECIO. A diferencia del operador ROLLUP, el operador CUBE resume los valores para cada subgrupo. También observe que un valor nulo se muestra para la columna CATEGORIA para esas filas. Esto es debido a que ambos valores (vocal e instrumental) se incluyen en el resumen de cada subgrupo.

Como se puede observar, la cláusula GROUP BY puede ser una herramienta muy útil cuando se trata de resumir datos, particularmente cuando se hace uso de las múltiples funciones disponibles en SQL, tales como SUM y AVG. En el capítulo 10 se analizan éstas y muchas otras funciones que se pueden utilizar para hacer la instrucción SELECT más robusta y aplicable a sus necesidades.

Utilice la cláusula HAVING para especificar un grupo de condiciones de búsqueda

La cláusula HAVING es similar a la cláusula WHERE ya que define una condición de búsqueda. Sin embargo, a diferencia de la cláusula WHERE, la cláusula HAVING se refiere a grupos, no a filas individuales:

- Si se especifica la cláusula GROUP BY, la cláusula HAVING se aplica a los grupos creados por la cláusula GROUP BY.

- Si se especifica la cláusula WHERE y no se especifica la cláusula GROUP BY, la cláusula HAVING se aplica a la salida de la cláusula WHERE y se trata esa salida como un grupo.
- Si no se especifica la cláusula WHERE ni la cláusula GROUP BY, la cláusula HAVING se aplica a la salida de la cláusula FROM y se trata esa salida como un grupo.

La mejor forma de entender la cláusula HAVING es recordando que las cláusulas en la instrucción SELECT se procesan con un orden definido. La cláusula WHERE sólo puede recibir la entrada de la cláusula FROM, pero la cláusula HAVING puede recibir la entrada de las cláusulas GROUP BY, WHERE o FROM. Ésta es una sutil pero importante distinción, y la mejor manera de ilustrarlo es viendo un par de ejemplos.

En el primer ejemplo, que se basa en la tabla EXISTENCIA_DISCO_COMPACTO en la figura 7-4, se utiliza la cláusula WHERE para especificar que los resultados de la consulta deben incluir sólo filas cuyo valor A_LA_MANO es menor que 20, como se muestra en la siguiente instrucción SELECT:

```
SELECT CATEGORIA, AVG(PRECIO) AS PROM_PRECIO
FROM EXISTENCIA_DISCO_COMPACTO
WHERE A_LA_MANO < 20
GROUP BY CATEGORIA;
```

La instrucción devuelve dos columnas: CATEGORIA y PROM_PRECIO, que es el promedio de todos los precios para cada categoría. Los promedios incluyen sólo las columnas donde los valores A_LA_MANO son menores que 20. Si se ejecuta esta instrucción, los resultados serían similares a los siguientes:

CATEGORIA	PROM_PRECIO
Instrumental	15.656666
Vocal	15.990000

Como era de esperarse, el resultado de la consulta devuelve dos filas (una para el grupo instrumental y una para el grupo vocal).

Si se fuera a utilizar la cláusula HAVING, en lugar de la cláusula WHERE, para limitar los valores a menos de 20, podría utilizar la siguiente instrucción SELECT:

```
SELECT CATEGORIA, AVG(PRECIO) AS PROM_PRECIO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CATEGORIA
HAVING A_LA_MANO < 20;
```

Sin embargo, si se trata de ejecutar esta instrucción, se recibiría un error, ya que no se aplican los valores individuales A_LA_MANO a los grupos. Para que una columna se incluya en la cláusula HAVING, debe ser un grupo de columnas o deben estar resumidas de alguna manera.

Ahora veamos otro ejemplo que utilice la cláusula HAVING. En este caso, la cláusula incluye una columna resumida:

```
SELECT PRECIO, CATEGORIA, SUM(A_LA_MANO) AS TOTAL_A_LA_MANO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY PRECIO, CATEGORIA
HAVING SUM(A_LA_MANO) > 10;
```


La cláusula HAVING en esta instrucción funciona porque los valores A_LA_MANO se suman, lo que significa que pueden trabajar dentro de la estructura del grupo. Los resultados de la consulta serían como se muestra a continuación:

PRECIO	CATEGORIA	TOTAL_A_LA_MANO
-----	-----	-----
15.99	Instrumental	23
16.99	Instrumental	50
14.99	Vocal	99
15.99	Vocal	73
16.99	Vocal	45

La cláusula HAVING se aplica a los resultados después de haberse agrupado (en la cláusula GROUP BY). Para cada grupo, los valores A_LA_MANO se añaden juntos, pero sólo se incluyen los grupos con los valores TOTAL_A_LA_MANO de más de 10. Si no se incluye la cláusula HAVING, los resultados de la consulta se incluirían en una fila adicional para el grupo de 14.99/Instrumental.

En su mayor parte, probablemente encuentre que utilizará la cláusula HAVING en conjunto con la cláusula GROUP BY. Mediante el uso de estos dos se pueden agrupar datos relevantes y filtrar los datos para refinar su búsqueda aún más. La cláusula HAVING también tiene la ventaja de permitir el uso de funciones establecidas tales como AVG o SUM, que no se pueden utilizar en la cláusula WHERE a menos que se coloquen dentro de una subconsulta. Los puntos importantes que deben tenerse en cuenta con la cláusula HAVING es que es la última cláusula en la expresión de tabla que debe aplicarse, y que se trata de datos agrupados en lugar de filas individuales.

Utilice la cláusula ORDER BY para ordenar los resultados de una consulta

La cláusula ORDER BY, cuando se usa en la instrucción SELECT, es la última cláusula procesada. La cláusula ORDER BY toma la salida de la cláusula SELECT y ordena los resultados de la consulta de acuerdo con las especificaciones dentro de la cláusula ORDER BY. La cláusula no agrupa las filas, como se agrupan por la cláusula GROUP BY, ni filtra las filas, como se filtran por la cláusula WHERE o la cláusula HAVING. Sin embargo, se puede especificar si las filas se organizan en un orden ascendente (utilizando la palabra clave ASC) o en orden descendente (usando la palabra clave DESC).

Para utilizar la cláusula ORDER BY, simplemente se especifica una o más columnas y las palabras clave opcionales ASC o DESC (una por columna). Si no se especifica la palabra clave, se toma ASC. Las filas se organizan de acuerdo con la columna que especifique. Si se define más de una columna en la cláusula ORDER BY, las filas se organizan en el orden en que las columnas se especifican.

Echemos un vistazo a algunos ejemplos para aclarar cómo funciona la cláusula ORDER BY. (Los ejemplos se basan en la tabla EXISTENCIA_DISCO_COMPACTO en la figura 7-4.) En el primer ejemplo se ordenan las filas basadas en la columna PRECIO:

```
SELECT * FROM EXISTENCIA_DISCO_COMPACTO
WHERE PRECIO < 16.00
ORDER BY PRECIO;
```

Observe que la columna PRECIO se especifica en la cláusula ORDER BY. También observe que si no se especifica la palabra clave ASC ni la palabra clave DESC, entonces se asume la palabra clave ASC. Si se ejecuta esta consulta, se recibirán resultados similares a los siguientes:

DISCO_COMPACTO	CATEGORIA	PRECIO	A_LA_MANO
-----	-----	-----	-----
Blue	Vocal	14.99	42
Court and Spark	Vocal	14.99	22
That Christmas Feeling	Vocal	14.99	8
Blues on the Bayou	Vocal	14.99	27
Orlando	Instrumental	14.99	5
Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
Leonard Cohen The Best Of	Vocal	15.99	12
Fundamental	Vocal	15.99	34
Past Light	Instrumental	15.99	17
Kojiki	Instrumental	15.99	6

Las filas se enumeran de acuerdo con la columna PRECIO. Los valores en la columna PRECIO aparecen en orden ascendente (del precio más bajo al precio más alto). Ya que la cláusula WHERE se especifica, ninguna fila con precios por encima de 15.99 se incluye en los resultados de la consulta. Ya que se incluye sólo la columna PRECIO en ORDER BY, también el orden de las filas que tienen el mismo precio es impredecible. Por ejemplo, las cinco filas con un PRECIO de 14.99, aparecen todas antes que aquellas con un PRECIO de 15.99, pero esas cinco filas pueden aparecer en cualquier orden.

En el siguiente ejemplo, la instrucción SELECT es casi la misma que la última instrucción, excepto que se especifica una columna adicional en la cláusula ORDER BY:

```
SELECT * FROM EXISTENCIA_DISCO_COMPACTO
WHERE PRECIO < 16.00
ORDER BY PRECIO, A_LA_MANO DESC;
```

En este caso, a la columna A_LA_MANO le sigue la palabra clave DESC, lo que significa que las filas se enumeran en orden descendente. Sin embargo, ya que hay dos columnas, las filas primero se ordenan por la columna PRECIO y luego por la columna A_LA_MANO. Si se ejecuta esta instrucción SELECT, se recibirán los siguientes resultados:

DISCO_COMPACTO	CATEGORIA	PRECIO	A_LA_MANO
-----	-----	-----	-----
Blue	Vocal	14.99	42
Blues on the Bayou	Vocal	14.99	27
Court and Spark	Vocal	14.99	22
That Christmas Feeling	Vocal	14.99	8
Orlando	Instrumental	14.99	5
Fundamental	Vocal	15.99	34
Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
Past Light	Instrumental	15.99	17
Leonard Cohen The Best Of	Vocal	15.99	12
Kojiki	Instrumental	15.99	6

Pregunta al experto

P: ¿Cómo afecta la cláusula **ORDER BY** los resultados de la consulta en SQL incrustado y módulos de SQL?

R: Se puede utilizar la cláusula **ORDER BY** sólo en invocación directa y cuando se definen cursores. (Se analizarán los cursores en el capítulo 15.) No se puede utilizar la cláusula **ORDER BY** en otras situaciones. Esto es debido a las limitaciones en los lenguajes de aplicación (no pueden manejar un número indeterminado de filas de un resultado de la consulta). Los lenguajes de aplicación no saben qué hacer con este tipo de incertidumbre. Y ya que la cláusula **ORDER BY** aplica sólo a los resultados de la consulta de múltiples filas, la cláusula no es aplicable a los entornos que requieren que las filas se devuelvan una a la vez. Sin embargo, los cursores ofrecen una forma para que los lenguajes de aplicación hagan frente a esa incertidumbre, permitiendo que la cláusula **ORDER BY** se utilice en definiciones de cursores. Los cursores se analizan con más detalle en el capítulo 15.

Como se puede observar, las filas se enumeran según el orden de los valores de **PRECIO**, que es de forma ascendente. Además, los valores **A_LA_MANO** se enumeran con un orden descendente para cada precio. Por lo tanto, para el conjunto de valores **PRECIO** de 14.99, las filas comienzan con el valor de 42 en la columna **A_LA_MANO** y terminan con el valor de 5. Luego brinca al siguiente grupo de valores **PRECIO**: 15.99. Una vez más, el valor **A_LA_MANO** más grande para el rango de **PRECIO** de 15.99 aparece en primer lugar y la última fila contiene el valor **A_LA_MANO** más pequeño para el rango de **PRECIO** de 15.99.

Siempre que se utilice la cláusula **ORDER BY**, debe estar consciente del orden en el que enumera los nombres de columna dentro de la cláusula. En el ejemplo anterior, la columna **PRECIO** aparece en primer lugar; por lo tanto, las filas se ordenan primero por la columna **PRECIO** y luego por la columna **A_LA_MANO**. Sin embargo, puede invertir los nombres de columna, como se muestra en la siguiente instrucción **SELECT**:

```
SELECT * FROM EXISTENCIA_DISCO_COMPACTO
WHERE PRECIO < 16.00
ORDER BY A_LA_MANO, PRECIO DESC;
```

Esta vez, la columna **A_LA_MANO** aparece en primer lugar y la columna **PRECIO** aparece en segundo lugar, y a la columna **PRECIO** se le asigna la palabra clave **DESC**. Como resultado, las filas se ordenan primero por la columna **A_LA_MANO**, como se muestra en los siguientes resultados de la consulta:

DISCO_COMPACTO	CATEGORIA	PRECIO	A_LA_MANO
-----	-----	-----	-----
Orlando	Instrumental	14.99	5
Kojiki	Instrumental	15.99	6
That Christmas Feeling	Vocal	14.99	8
Leonard Cohen The Best Of	Vocal	15.99	12
Past Light	Instrumental	15.99	17
Court and Spark	Vocal	14.99	22

Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
Blues on the Bayou	Vocal	14.99	27
Fundamental	Vocal	15.99	34
Blue	Vocal	14.99	42

Observe que los valores A_LA_MANO están en orden ascendente. Las filas se ordenan según el valor PRECIO. Sin embargo, ya que sólo hay dos filas que comparten los mismos valores A_LA_MANO (27), éstas son las únicas filas que afecta la columna ORDER BY con respecto a la columna PRECIO.

La cláusula ORDER BY es una herramienta conveniente para organizar los resultados de la consulta, pero recuerde que no afecta qué datos se muestran. Sólo las otras cláusulas pueden nombrar, filtrar y agrupar datos. La cláusula ORDER BY no es más que un organizador para lo que ya existe. Y de hecho, si bien no es una práctica muy buena, se pueden incluir columnas en la cláusula ORDER BY que no están en la cláusula SELECT, y de este modo clasificar las columnas que no son visibles en los resultados de la consulta.

Pruebe esto 7-1 **Consulte la base de datos inventario**

Para los ejercicios de los capítulos anteriores se creó un número continuo de tablas base capaces de almacenar datos. En este capítulo aprenderá cómo crear instrucciones SELECT que permitan consultar datos en las tablas base. Como resultado, este ejercicio se centra en la creación de instrucciones SELECT que consulte datos en las tablas que se crearon. Sin embargo, antes de que realmente pueda consultar estas tablas, los datos se deben almacenar dentro de éstas. Ya que no se cubre la inserción de datos hasta el capítulo 8, se proporcionan las instrucciones necesarias para insertar los datos en el archivo Try_This_07.txt (en inglés), que puede descargar del sitio web. El archivo contiene una serie de instrucciones INSERT que permiten llenar las tablas, junto con las instrucciones SELECT utilizadas en este ejercicio. También se pueden ver estas instrucciones en el apéndice C.

Si observa el archivo Try_This_07.txt, encontrará una serie de instrucciones INSERT que se agrupan de acuerdo con las tablas que se crearon en la base de datos INVENTARIO. Por ejemplo, el primer conjunto de instrucciones INSERT son para la tabla DISQUERAS_CD, como se muestra en las siguientes instrucciones:

```
--Inserta datos en la tabla DISQUERAS_CD
INSERT INTO DISQUERAS_CD VALUES ( 827, 'Private Music' );
INSERT INTO DISQUERAS_CD VALUES ( 828, 'Reprise Records' );
INSERT INTO DISQUERAS_CD VALUES ( 829, 'Asylum Records' );
INSERT INTO DISQUERAS_CD VALUES ( 830, 'Windham Hill Records' );
INSERT INTO DISQUERAS_CD VALUES ( 831, 'Geffen' );
INSERT INTO DISQUERAS_CD VALUES ( 832, 'MCA Records' );
INSERT INTO DISQUERAS_CD VALUES ( 833, 'Decca Record Company' );
INSERT INTO DISQUERAS_CD VALUES ( 834, 'CBS Records' );
INSERT INTO DISQUERAS_CD VALUES ( 835, 'Capitol Records' );
INSERT INTO DISQUERAS_CD VALUES ( 836, 'Sarabande Records' );
--Fin de inserción en la tabla DISQUERAS_CD
```

Tendrá que copiar estas instrucciones en la aplicación de cliente y ejecutarlas. Cada instrucción INSERT añade una fila de datos a la tabla aplicable. Por ejemplo, la primera instrucción INSERT que aparece en el código anterior añade una fila de datos a la tabla DISQUERAS_CD. Los valores que se añaden son 827 (para la columna ID_DISQUERA) y Private Music (para la columna NOMBRE_COMPANIA). De nuevo se presenta la instrucción INSERT con más detalle en el capítulo 8. Si se siente incómodo insertando estos datos antes de leer acerca de la instrucción INSERT, le sugiero que revise la información del capítulo 8 antes de trabajar en este ejercicio y luego vuelva aquí para realizar cada paso. Sin embargo, si decide hacer este ejercicio ahora, entonces simplemente necesita ejecutar cada instrucción, como se describe en los siguientes pasos.

NOTA

Como probablemente ha notado, cada bloque de instrucciones INSERT comienza y termina con una línea que empieza con guiones dobles (--). Los guiones dobles indican que la línea de texto que sigue es un comentario. La aplicación de SQL simplemente omite estas líneas. Los comentarios están sólo para proporcionar información a los programadores de SQL para que puedan entender mejor el código.

Paso a paso

1. Abra la aplicación de clientes de su RDBMS y conecte la base de datos INVENTARIO.
2. Abra el archivo Try_This_07.txt y copie las instrucciones INSERT en la aplicación de cliente. La mayoría de las aplicaciones permiten ejecutar bloques de instrucciones en lugar de tener que introducir los datos de una fila a la vez. Si la aplicación soporta la ejecución de múltiples instrucciones, ejecute las instrucciones una tabla a la vez copiando y pegando los bloques de instrucciones en la aplicación de clientes. Debe introducir los datos para cada tabla en el orden que los datos aparecen en el archivo Try_This_07.txt. Por ejemplo, debe insertar los valores en la tabla DISQUERAS_CD antes que en la tabla DISCO_COMPACTO.

Para cada instrucción INSERT que ejecute, debe recibir un mensaje que reconozca que la fila se insertó en la tabla. Después de llenar cada tabla con los datos, estará listo para continuar con el siguiente paso.

3. Ahora consulte todos los datos de la tabla ARTISTAS. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT *  
FROM ARTISTAS;
```

El resultado de su consulta debe incluir las columnas ID_ARTISTA, NOMBRE_ARTISTA y LUGAR_DE_NACIMIENTO. Debe componerse de 18 filas de datos en todas.

4. Ahora debe crear una consulta que especifique qué columnas se incluyen en los resultados de la consulta. Para la siguiente instrucción SELECT, consulte la tabla DISCO_COMPACTO, pero devuelva sólo las columnas TITULO_CD y EN_EXISTENCIA. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA  
FROM DISCOS_COMPACTOS;
```

Los resultados de la consulta deben incluir sólo las dos columnas que se especifican en la instrucción SELECT. Además, la consulta debe devolver 15 filas de datos.

5. En el capítulo 5, Pruebe esto 5-1, se creó la vista CD_EN_EXISTENCIA. La vista devuelve los mismos datos que se especifican en la instrucción SELECT en el paso 4, excepto que limita los resultados de las filas con los valores EN_EXISTENCIA mayor que 10. Ahora puede consultar esa vista. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT *
FROM CD_EN_EXISTENCIA;
```

Observe que la instrucción SELECT es la misma que si fuera para una tabla base persistente. Puede incluso especificar los nombres de columna de la vista si se desea. (De hecho, debe hacerlo si consulta la vista de cualquier otra manera que a través de la invocación directa.) En la última instrucción SELECT, la consulta devolvió 15 filas, pero esta consulta devuelve sólo 12 filas, ya que los valores EN_EXISTENCIA deben estar por arriba de 10. La parte agradable sobre la vista es que ya está configurada para devolver exactamente la información deseada, sin tener que definir la cláusula WHERE.

6. Ahora consulte la tabla DISCOS_COMPACTOS, pero refine la instrucción SELECT utilizando la cláusula WHERE. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA > 10 AND EN_EXISTENCIA < 30;
```

Debido a que se añade la cláusula WHERE, los resultados de la consulta ahora incluyen sólo nueve filas, y cada fila debe contener un valor EN_EXISTENCIA entre 10 y 30.

7. La siguiente instrucción SELECT que se crea agrupa la información en la tabla DISCOS_COMPACTOS. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL_EN_EXISTENCIA
FROM DISCOS_COMPACTOS
GROUP BY ID_DISQUERA;
```

Se devuelve una fila para cada valor diferente ID_DISQUERA, y para cada uno de estos valores se devuelve el total para los valores EN_EXISTENCIA. Hay 10 filas. Observe que en los resultados de la consulta, el nombre de la columna con los totales EN_EXISTENCIA es TOTAL_EN_EXISTENCIA. Cuando aprenda más sobre la unión de tablas, será capaz de agrupar los datos basados en consultas más complejas. La unión de tablas se analiza en el capítulo 11.

8. Ahora se añade la cláusula HAVING a la instrucción SELECT que se acaba de ejecutar. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL_EN_EXISTENCIA
FROM DISCOS_COMPACTOS
GROUP BY ID_DISQUERA
HAVING SUM(EN_EXISTENCIA) > 10;
```

(continúa)

La cláusula HAVING limita las filas que se devuelven a aquellas cuyos valores TOTAL_EN_EXISTENCIA sean mayores que 10. Ahora sólo se devuelven ocho filas.

9. También se puede ejecutar la instrucción SELECT que ordene los datos devueltos por la consulta. Introduzca y ejecute la siguiente instrucción SQL:

```
SELECT *  
  FROM DISCOS_COMPACTOS  
 WHERE EN_EXISTENCIA > 10  
 ORDER BY TITULO_CD DESC;
```

Los resultados de la consulta deben estar organizados de acuerdo con la columna TITULO_CD, con las columnas enumeradas en orden descendente. Ya que se utiliza la cláusula WHERE, sólo 12 filas se deben devolver.

10. Cierre la aplicación clientes.

Resumen de Pruebe esto

En este ejercicio se insertaron datos en las tablas de la base de datos INVENTARIO. A continuación se crearon instrucciones SELECT que permiten consultar los datos en esas tablas. Debe sentirse libre de experimentar con las instrucciones SELECT y probar diferentes tipos de consultas. A medida que se sienta más cómodo utilizando la instrucción SELECT y aprenda técnicas más avanzadas para consultar datos, será capaz de escribir instrucciones SELECT para acceder a múltiples tablas, calcular datos y resumir información. Sin embargo, incluso las técnicas más avanzadas se basan en el fundamento básico que ha demostrado en este ejercicio. Todo lo demás se basa en esto.

Autoexamen Capítulo 7

1. ¿Cuáles cláusulas en la instrucción SELECT son parte de la expresión de la tabla?
A SELECT
B FROM
C WHERE
D ORDER BY
2. ¿En qué orden se aplican las cláusulas de la instrucción SELECT?
3. Se está escribiendo la instrucción SELECT que recupera la columna TITULO_CD y todas las filas de la tabla INVENTARIO. ¿Cuál instrucción SELECT debe utilizarse?
4. Se está escribiendo una instrucción SELECT que recupera la columna TITULO_CD y todas las filas de la tabla INVENTARIO. Desea que la columna en los resultados de la consulta sea llamada DISCO_COMPACTO. ¿Cuál instrucción SELECT deberá utilizarse?

5. ¿Cuáles cláusulas se requieren en una instrucción SELECT?
 - A SELECT
 - B FROM
 - C WHERE
 - D GROUP BY
6. ¿Cuál palabra clave se debe añadir a la cláusula SELECT para asegurarse que cada fila de los resultados de la consulta sea única?
 - A ALL
 - B ROLLUP
 - C DISTINCT
 - D CUBE
7. Se está creando una instrucción SELECT para la tabla INVENTARIO y desea asegurarse que sólo las filas con un valor PRECIO_MENUDEO menor a \$16.00 sean incluidas en los resultados de la consulta. ¿Qué cláusula WHERE deberá utilizarse?
8. Se está creando una instrucción SELECT que incluye una cláusula WHERE. La cláusula WHERE contiene dos predicados. Se desea que la condición de uno de los predicados se cumpla, pero no es necesario que ambas condiciones se cumplan. ¿Qué palabra clave deberá utilizarse para conectar los dos predicados?
9. ¿Cuál de las siguientes declaraciones puede evaluar una cláusula WHERE?
 - A Verdadero
 - B No
 - C Falso
 - D Desconocido
10. ¿Cuál cláusula permite agrupar valores en una columna específica?
 - A ROLLUP
 - B HAVING
 - C ORDER BY
 - D GROUP BY
11. ¿Cuáles dos operadores pueden utilizarse en una cláusula GROUP BY para arrojar datos de resumen adicionales en los resultados de una consulta?
 - A ROLLUP
 - B HAVING
 - C CUBE
 - D DISTINCT

- 12.** Se está escribiendo la instrucción SELECT que recupera las columnas CATEGORIA y PRECIO de la tabla EXISTENCIA_DISCO_COMPACTO. Quiere agrupar los datos primero por la columna CATEGORIA y luego por la columna PRECIO. ¿Cuál instrucción SELECT deberá utilizarse?
- 13.** Se está escribiendo la instrucción SELECT que recupera las columnas CATEGORIA y PRECIO de la tabla EXISTENCIA_DISCO_COMPACTO. Quiere agrupar los datos primero por la columna CATEGORIA y luego por la columna PRECIO. A continuación, desea filtrar cualquier grupo que tenga un valor PRECIO superior a \$15.99. ¿Cuál instrucción SELECT deberá utilizarse?
- 14.** Se crea una instrucción SELECT que incluye una cláusula SELECT, una cláusula FROM, una cláusula WHERE, una cláusula GROUP BY y una cláusula HAVING. ¿Desde cuál cláusula recibirá resultados la cláusula HAVING?
- A** SELECT
 - B** FROM
 - C** WHERE
 - D** GROUP BY
- 15.** ¿En qué aspecto la cláusula HAVING es diferente de la cláusula WHERE?
- 16.** ¿De cuál cláusula recibe resultados la cláusula ORDER BY?
- 17.** ¿Cuál palabra clave deberá agregarse a una cláusula ORDER BY para clasificar los datos en orden descendente?

Capítulo 8

Modificar datos SQL

Habilidades y conceptos clave

- Insertar datos SQL
 - Actualizar datos SQL
 - Eliminar datos SQL
-

Una de las funciones principales de cualquier base de datos es la capacidad de manejar los datos almacenados dentro de sus tablas. Los usuarios designados deben ser capaces de insertar, actualizar y eliminar los datos según sea necesario para mantener el flujo de la base de datos y asegurarse que sólo los datos adecuados están siendo almacenados. SQL proporciona tres instrucciones para el manejo básico de datos: INSERT, UPDATE y DELETE. En este capítulo se examinarán cada una de estas instrucciones y se demostrará cómo pueden ser utilizadas en un ambiente SQL para modificar los datos de la base de datos.

Insertar datos SQL

En el capítulo 7, el ejercicio 7-1 contenía una breve introducción acerca de la instrucción INSERT. Como se puede ver en ese ejercicio, la instrucción INSERT permite agregar datos a las diferentes tablas en una base de datos. Se presenta la sintaxis básica en esta sección y una sintaxis alternativa en la siguiente sección (“Insertar valores desde una instrucción SELECT”). La sintaxis de una instrucción INSERT básica es relativamente sencilla:

```
INSERT INTO <nombre de la tabla>
[ ( <nombre de la columna> ) [ { , <nombre de la columna> }... ] ]
VALUES ( <valor> [ { , <valor> }... ] )
```

En la sintaxis, sólo son obligatorias la primera y la última líneas. La segunda línea es opcional. Tanto la primera como la segunda línea son parte de la cláusula INSERT INTO. En esta cláusula se debe identificar el nombre de la tabla (o vista) en la cual se agregarán los datos. El nombre de la tabla sigue a las palabras clave INSERT INTO. Entonces el usuario tiene la opción de identificar los nombres de la columna en la tabla que recibirán los datos. Éste es el propósito de la segunda línea en la sintaxis. Es posible especificar una o más columnas, y todas ellas deberán estar dentro de paréntesis. Si se especifican más columnas, éstas deberán estar separadas por comas.

NOTA

La mayoría de las implementaciones de SQL soportan las vistas INSERT INTO. Sin embargo, existen restricciones. Por ejemplo, no es posible insertar INSERT INTO en una vista si existen columnas de la tabla que no están incluidas en la vista y que no permiten valores nulos ni tienen un valor por defecto definido. Adicionalmente, si la vista tiene más de una tabla base, podría no ser posible insertar INSERT INTO en lo absoluto, y si se pudiera, se le requeriría al usuario nombrar columnas desde sólo una de las tablas base debido a que un INSERT sólo puede afectar a una tabla base. Siempre revise la documentación del fabricante para mayor información.

En la tercera línea de la sintaxis, que se refiere a la cláusula **VALUES**, se deben especificar uno o más valores que serán ingresados en la tabla. La lista de valores deberá estar encerrada en paréntesis y, si se especifica más de uno, deberán estar separados utilizando comas. Asimismo, los valores deben cumplir los siguientes requisitos:

- Si los nombres de columna no se especifican en la cláusula **INSERT INTO**, entonces deberá haber un valor por cada columna en la tabla y los valores deberán estar en el mismo orden en el que están definidos en la tabla.
- Si los nombres de columna se especifican en la cláusula **INSERT INTO**, entonces deberá haber exactamente un valor por cada columna especificada y esos valores deberán estar en el mismo orden en el que están definidos en la cláusula **INSERT INTO**. Sin embargo, los nombres y valores de columna no tienen que estar en el mismo orden que las columnas en la definición de la tabla.
- Se debe proporcionar un valor por cada columna en la tabla *excepto* para las columnas que permiten valores nulos o que tienen un valor definido por defecto.
- Cada valor con un carácter del tipo de datos de cadena debe estar encerrado en comillas sencillas.
- Se puede utilizar la palabra clave **NULL** (o **null**) como el valor de los datos en la cláusula **VALUES** para asignar un valor nulo a cualquier columna que permita valores nulos.

NOTA

Muchos programadores de SQL prefieren especificar los nombres de columna dentro de la cláusula **INSERT INTO**, incluso si no es necesario, debido a que proporciona un método para documentar cuáles columnas suponen recibir los datos. Esta práctica también hace a la instrucción **INSERT** menos propensa a errores y a otros problemas provocados al añadir nuevas columnas o al cambiar el orden de las columnas en algún momento futuro. Por estas razones, muchas organizaciones requieren el uso de los nombres de columna en todas las instrucciones **INSERT**.

Ahora veamos algunos ejemplos de la instrucción **INSERT**. Para estos ejemplos se utilizará la tabla **INVENTARIO_CD**. La tabla está basada en la siguiente definición de tabla:

```
CREATE TABLE INVENTARIO_CD
( NOMBRE_CD          VARCHAR(60)                NOT NULL,
  TIPO_MUSICA        VARCHAR(15),
  EDITOR             VARCHAR(50) DEFAULT 'Independiente' NOT NULL,
  EN_EXISTENCIA      INT                        NOT NULL );
```

En el primer ejemplo se mostrarán los valores insertados en cada columna en la tabla **INVENTARIO_CD**:

```
INSERT INTO INVENTARIO_CD
VALUES ( 'Patsy Cline: 12 Greatest Hits', 'Country', 'MCA Records', 32 );
```

Observe que la cláusula **INSERT INTO** incluye solamente el nombre de la tabla **INVENTARIO_CD**, pero no especifica ninguna columna. En la cláusula **VALUES** se han especificado cuatro valores. Los valores están separados por comas, y los valores con caracteres del tipo de datos de

cadena están encerrados en comillas sencillas. Si hacemos referencia a la definición de tabla, se verá que los valores especificados en la cláusula VALUES se encuentran en el mismo orden que las definiciones de columna.

Cuando se ejecuta la instrucción INSERT mostrada en el ejemplo, los datos son añadidos a la tabla INVENTARIO_CD, como se muestra en la figura 8-1.

Si se intentara ejecutar una instrucción INSERT como en el último ejemplo, pero incluyendo solamente tres valores en lugar de cuatro, se recibiría como resultado un error. Por ejemplo, no sería posible ejecutar la siguiente instrucción:

```
INSERT INTO INVENTARIO_CD
VALUES ( 'Patsy Cline: 12 Greatest Hits', 'MCA Records', 32 );
```

En este ejemplo se han especificado solamente tres valores. En este caso, el valor faltante es para la columna TIPO_MUSICA. A pesar de que esta columna acepta valores nulos, la implementación de SQL no tiene forma de saber cuál valor está siendo omitido, y por lo tanto se arroja un error.

En lugar de dejar el valor fuera de la cláusula VALUES, se puede especificar un valor nulo, como se muestra en el siguiente ejemplo:

```
INSERT INTO INVENTARIO_CD
VALUES ( 'Out of Africa', null, 'MCA Records', 29 );
```

Si se ejecuta la instrucción INSERT, la tabla no incluirá una fila adicional. La figura 8-2 muestra cómo se vería la tabla, asumiendo que las dos instrucciones INSERT han sido ejecutadas.

El valor nulo se ingresó en la columna TIPO_MUSICA, y los otros valores fueron ingresados en sus columnas apropiadas. Si no se permitiera un valor nulo en la columna TIPO_MUSICA, se tendría que haber especificado un valor.

NOTA

La figura 8-2 muestra la nueva fila que está siendo insertada después de la fila existente en la tabla. Sin embargo, la fila podría ser insertada en cualquier lugar en una tabla, dependiendo de cómo inserte filas la implementación de SQL. El estándar de SQL no especifica dónde se insertará una fila en una tabla. De hecho, jamás se deberá confiar en que las filas en una tabla estén en un orden en particular (se deberá utilizar la cláusula ORDER BY cuando los resultados de SELECT necesiten estar en una secuencia en particular).

En lugar de proporcionar un valor para cada columna cuando se inserta una fila, se puede especificar cuáles columnas recibirán valores. Por ejemplo, se puede especificar los valores para las

NOMBRE_CD: VARCHAR(60)	TIPO_MUSICA: VARCHAR(15)	EDITOR: VARCHAR(50)	EN_EXISTENCIA: INT
Patsy Cline: 12 Greatest Hits	Country	MCA Records	32

Figura 8-1 La tabla INVENTARIO_CD con la nueva fila de datos.

NOMBRE_CD: VARCHAR(60)	TIPO_MUSICA: VARCHAR(15)	EDITOR: VARCHAR(50)	EN_EXISTENCIA INT
Patsy Cline: 12 Greatest Hits	Country	MCA Records	32
Out of Africa	NULL	MCA Records	29

Figura 8-2 La tabla INVENTARIO_CD con dos filas de datos.

columnas NOMBRE_CD, EDITOR y EN_EXISTENCIA de la tabla INVENTARIO_CD, como se muestra en el siguiente ejemplo:

```
INSERT INTO INVENTARIO_CD ( NOMBRE_CD, EDITOR, EN_EXISTENCIA )
VALUES ( 'Fundamental', 'Capitol Records', 34 );
```

En este caso, un solo valor ha sido especificado para cada una de las columnas identificadas dentro de la cláusula INSERT INTO, y los valores se especifican en el mismo orden que las columnas en la cláusula INSERT INTO. Observe que la instrucción INSERT no incluye a la columna TIPO_MUSICA dentro de la cláusula INSERT INTO ni de la cláusula VALUES. Se puede omitir esta columna debido a que los valores nulos se permiten en esa columna. Si se ejecutara esta instrucción, la tabla INVENTARIO_CD ahora tendría una tercera fila (mostrada la figura 8-3).

Una vez más, el valor nulo (NULL) es agregado automáticamente a la columna TIPO_MUSICA. Si un valor por defecto hubiera sido definido para la columna, ese valor habría sido agregado. Por ejemplo, la siguiente instrucción INSERT omite la columna EDITOR en lugar de la columna TIPO_MUSICA:

```
INSERT INTO INVENTARIO_CD ( NOMBRE_CD, TIPO_MUSICA, EN_EXISTENCIA )
VALUES ( 'Orlando', 'Soundtrack', 5 );
```

Cuando la fila es agregada a la tabla INVENTARIO_CD, el valor por defecto (Independiente) es agregado a la columna EDITOR, como se muestra la figura 8-4.

NOMBRE_CD: VARCHAR(60)	TIPO_MUSICA: VARCHAR(15)	EDITOR: VARCHAR(50)	EN_EXISTENCIA: INT
Patsy Cline: 12 Greatest Hits	Country	MCA Records	32
Out of Africa	NULL	MCA Records	29
Fundamental	NULL	Capitol Records	34

Figura 8-3 La tabla INVENTARIO_CD con tres filas de datos.

NOMBRE_CD: VARCHAR(60)	TIPO_MUSICA: VARCHAR(15)	EDITOR: VARCHAR(50)	EN_EXISTENCIA: INT
Patsy Cline: 12 Greatest Hits	Country	MCA Records	32
Out of Africa	NULL	MCA Records	29
Fundamental	NULL	Capitol Records	34
Orlando	Soundtrack	Independent	5

Figura 8-4 La tabla INVENTARIO_CD con cuatro filas de datos.

Si se intenta ejecutar una instrucción INSERT omitiendo una columna que no permita valores nulos y que no tenga un valor por defecto definido, se arrojará como resultado un error. Cuando se inserta una nueva fila, el RDBMS debe obtener un valor para cada columna desde algún lado, por lo que si no se permiten los valores nulos, entonces el valor debe venir ya sea de un valor por defecto (si está definido) o de la cláusula VALUES de la instrucción INSERT.

NOTA

Los valores que se especifiquen en la cláusula VALUES deberán hacerse de acuerdo con todas las restricciones localizadas en una tabla. Esto significa que los valores deben cumplir con los tipos de datos o dominios asociados con una columna. Además, los valores están confinados por cualquier restricción definida en la tabla. Por ejemplo, una restricción de clave foránea pudiera evitar que se agregue cualquier valor que viole la limitación, o una limitación de revisión pudiera limitar el rango de valores que pueden ser ingresados a la tabla. Asegúrese de conocer ampliamente las restricciones localizadas en una tabla antes de intentar agregar datos en ella. Se puede aprender más acerca de los tipos de datos en el capítulo 3. Se puede aprender más acerca de dominios y restricciones en el capítulo 4.

Desde luego, también es posible especificar todas las columnas en la cláusula INSERT INTO. Si se realiza esto, debe especificarse correctamente el mismo número de valores, en el mismo orden en el cual están especificadas las columnas. La siguiente instrucción INSERT ingresa valores a todas las columnas de la tabla INVENTARIO_CD:

```
INSERT INTO INVENTARIO_CD ( NOMBRE_CD, MUSIC_TYPE, EDITOR, EN_EXISTENCIA )
VALUES ( 'Court and Spark', 'Pop', 'Asylum Records', 22 );
```

Cuando se ejecuta esta instrucción, una fila es agregada a la tabla INVENTARIO_CD, con un valor para cada columna. La figura 8-5 muestra la nueva fila, junto con las cuatro filas anteriores que se habían insertado. Si se omitiera uno de los valores desde la cláusula VALUES (incluso cuando en esa columna relacional se permitan los valores nulos) se recibiría un mensaje de error cuando se ejecute la instrucción.

Insertar valores desde una instrucción SELECT

Anteriormente en este capítulo, al inicio de la sección “Insertar datos SQL”, se mencionó que la cláusula VALUES es obligatoria y que es necesario especificar por lo menos un valor. Sin embar-

NOMBRE_CD: VARCHAR(60)	TIPO_MUSICA: VARCHAR(15)	EDITOR: VARCHAR(50)	EN_EXISTENCIA: INT
Patsy Cline: 12 Greatest Hits	Country	MCA Records	32
Out of Africa	NULL	MCA Records	29
Fundamental	NULL	Capitol Records	34
Orlando	Soundtrack	Independent	5
Court and Spark	Pop	Asylum Records	22

Figura 8-5 La tabla INVENTARIO_CD con cinco filas de datos.

go, existe una alternativa a la cláusula VALUES. Se puede utilizar una instrucción SELECT para especificar los valores que se quieran ingresar en una tabla. La clave para utilizar una instrucción SELECT, al igual que al utilizar una cláusula VALUES, es asegurarse que el número de valores aplicados por la instrucción SELECT coincida con el número requerido de valores, y que éstos cumplan con cualquier restricción de la tabla correspondiente. Observemos un ejemplo.

Supongamos que, además de la tabla INVENTARIO_CD que se ha estado utilizando en los ejemplos anteriores, la base de datos también tuviera una segunda tabla llamada INVENTARIO_CD_2, que incluyera dos columnas, como se muestra en la siguiente definición de tabla:

```
CREATE TABLE INVENTARIO_CD_2
( NOMBRE_CD_2      VARCHAR(60)  NOT NULL,
  EN_EXISTENCIA_2  INT          NOT NULL );
```

La columna NOMBRE_CD_2 en la tabla INVENTARIO_CD_2 tiene los mismos tipos de datos que la columna NOMBRE_CD en la tabla INVENTARIO_CD, y la columna EN_EXISTENCIA_2 en la tabla INVENTARIO_CD_2 tiene los mismos tipos de datos que la columna EN_EXISTENCIA en la tabla INVENTARIO_CD. Como resultado, los valores tomados de las dos columnas en una tabla pueden ser insertados en las dos columnas de la segunda tabla.

NOTA

Una columna en una tabla no tiene que contener los mismos tipos de datos que una columna en otra tabla para que los valores sean copiados de una a la otra; basta con que los valores insertados en la tabla destino cumplan con las restricciones de datos de esa tabla.

Al utilizar la instrucción INSERT, se pueden copiar valores de la tabla INVENTARIO_CD a la tabla INVENTARIO_CD_2. La siguiente instrucción INSERT incluye una instrucción SELECT que consulta a la tabla INVENTARIO_CD:

```
INSERT INTO INVENTARIO_CD_2
SELECT NOMBRE_CD, EN_EXISTENCIA
FROM INVENTARIO_CD;
```


NOMBRE_CD: VARCHAR(60)	EN_EXISTENCIA: INT
Patsy Cline: 12 Greatest Hits	32
Out of Africa	29
Fundamental	34
Orlando	5
Court and Spark	22

Figura 8-6 La tabla INVENTARIO_CD_2 con cinco filas de datos.

Como se puede ver, ninguna columna está especificada en la cláusula INSERT INTO; como resultado, los valores serán insertados en ambas columnas en la tabla INVENTARIO_CD_2. En la segunda línea de la instrucción se utiliza una instrucción SELECT para tomar los valores desde las columnas NOMBRE_CD y EN_EXISTENCIA de la tabla INVENTARIO_CD. Los valores serán entonces insertados en sus columnas respectivas en la tabla INVENTARIO_CD_2, como se muestra la figura 8-6.

Observe que la tabla INVENTARIO_CD_2 contiene las mismas cinco filas de datos que se muestran en la figura 8-5, sólo que la tabla INVENTARIO_CD_2 contiene únicamente dos columnas: NOMBRE_CD_2 e EN_EXISTENCIA_2.

Al igual que cualquier otra instrucción SELECT, la instrucción SELECT que se utiliza en una instrucción INSERT puede contener una cláusula WHERE. En la siguiente instrucción INSERT, la instrucción SELECT contiene una cláusula WHERE que limita los valores de EN_EXISTENCIA a una cantidad mayor a 10:

```
INSERT INTO INVENTARIO_CD_2
  SELECT NOMBRE_CD, EN_EXISTENCIA
  FROM INVENTARIO_CD
  WHERE EN_EXISTENCIA > 10;
```

Si se ejecutara esta instrucción, solamente cuatro filas serían añadidas a la tabla INVENTARIO_CD_2, en lugar de las cinco filas que se vieron en el ejemplo anterior. En este caso, la cláusula WHERE funciona exactamente igual que una cláusula WHERE en cualquier instrucción SELECT. Como resultado, cualquier fila con un valor EN_EXISTENCIA que no sea mayor a 10 es eliminado de los resultados de la consulta. Esos nuevos resultados filtrados se insertan entonces en la tabla INVENTARIO_CD_2.

Actualizar datos SQL

Como su nombre lo indica, la instrucción UPDATE permite actualizar los datos en una base de datos SQL. Con la instrucción UPDATE se pueden modificar datos en una o más filas para una o más columnas. La sintaxis para la instrucción UPDATE se puede mostrar de la manera siguiente:

```
UPDATE <nombre de la tabla>
SET <determinar expresión de la cláusula> [ {, <determinar expresión de la cláusula> }... ]
[ WHERE <condición de búsqueda> ]
```

Como se puede ver, la cláusula UPDATE y la cláusula SET son obligatorias, mientras que la cláusula WHERE es opcional. En la cláusula UPDATE se debe especificar el nombre de la tabla (o vista) que se está actualizando. En la cláusula SET se debe especificar una o más expresiones de cláusula, lo cual se discutirá con más detalle posteriormente en este capítulo. En la cláusula WHERE, al igual que con la cláusula WHERE en una instrucción SELECT (véase el capítulo 7), se debe especificar una condición de búsqueda. La cláusula WHERE funciona aquí de una forma muy parecida a como lo hace en la instrucción SELECT. Se especifica una condición o conjunto de condiciones que actúa como un filtro para las filas que se están actualizando. Solamente las filas que cumplen con estas condiciones son actualizadas. En otras palabras, solamente las filas cuyo resultado es verdadero son actualizadas.

NOTA

SQL permite utilizar nombres de vistas en las instrucciones UPDATE. Sin embargo, si la vista está basada en múltiples tablas, todas las columnas que se están actualizando deben venir desde una sola tabla base, y no debe haber otras restricciones como se describe en su documentación de DBMS.

Ahora regresemos a la cláusula SET. Como se puede ver, la cláusula incluye el marcador de posición < determinar expresión de la cláusula >. Se deben especificar una o más determinar expresión de la cláusula. Si se especifica más de una, se deben separar utilizando comas. La sintaxis del marcador de posición < determinar expresión de la cláusula > puede descomponerse como se muestra a continuación:

< nombre de la columna > = < expresión de valor >

Básicamente, se debe especificar un nombre de columna (desde la tabla que se está actualizando) y proporcionar un valor que el valor en la columna deberá igualar. Por ejemplo, supongamos que se requiere que un valor en la columna EN_EXISTENCIA sea cambiado a 37. (Sin importar cuál sea el valor actual en esa columna.) Determinar expresión de la cláusula quedaría como sigue: EN_EXISTENCIA = 37. En este caso, la expresión de valor es 37; sin embargo, la expresión de valor puede ser más complicada que eso. Por ejemplo, se puede basar el nuevo valor en un valor antiguo: EN_EXISTENCIA = (EN_EXISTENCIA + 1). En este caso, la expresión de valor es EN_EXISTENCIA + 1, la cual agrega 1 al valor actual en la columna EN_EXISTENCIA para darle un nuevo valor. En este ejemplo, si el valor original era 37, el nuevo valor será 38.

Ahora que se ha dado un vistazo a las diferentes partes de la instrucción UPDATE, pongámonos todas juntas utilizando algunos ejemplos. Los ejemplos que se utilizarán estarán basados en la tabla INVENTARIO_CD, que se muestra en la figura 8-5.

En el primer ejemplo se utiliza la instrucción UPDATE para cambiar los valores de la columna EN_EXISTENCIA a 27, como se muestra en la siguiente instrucción SQL:

```
UPDATE INVENTARIO_CD
SET EN_EXISTENCIA = 27;
```

Esta instrucción realiza exactamente lo que se pudiera esperar: cambia cada fila en la tabla INVENTARIO_CD para que la columna EN_EXISTENCIA contenga un valor de 27 para cada fila. El resultado es óptimo si eso es lo que se quería, pero es poco probable que se quiera cambiar cada fila en una tabla para que todos los valores de esa columna sean iguales en cada fila. Es mucho más probable que se quiera precisar la actualización utilizando una cláusula WHERE.

En el siguiente ejemplo se modifica la instrucción UPDATE anterior para incluir una cláusula WHERE:

```
UPDATE INVENTARIO_CD
  SET EN_EXISTENCIA = 27
  WHERE NOMBRE_CD = 'Out of Africa';
```

La instrucción UPDATE aún cambia la columna EN_EXISTENCIA a un valor de 27, pero sólo lo hace para las filas que concuerdan con la condición de búsqueda en la cláusula WHERE. En este caso, solamente una fila cumple con esa condición: Out of Africa.

Algunas veces también se requiere cambiar un valor basado en un valor que ya exista, por ejemplo, la cantidad de inventario en existencia. Por ejemplo, se pueda añadir 2 al valor en la columna EN_EXISTENCIA:

```
UPDATE INVENTARIO_CD
  SET EN_EXISTENCIA = (EN_EXISTENCIA + 2)
  WHERE NOMBRE_CD = 'Out of Africa';
```

Si la fila Out of Africa contiene el valor 27 en la columna EN_EXISTENCIA, y se ejecuta esta instrucción UPDATE, el nuevo valor será 29. Si se ejecuta esta instrucción sin la cláusula WHERE, el valor 2 será agregado al valor EN_EXISTENCIA para cada fila en la tabla.

La cláusula WHERE también permite especificar más de un predicado, al igual que se puede hacer con una cláusula WHERE en la instrucción SELECT. En el siguiente ejemplo, se sustrae 2 del valor EN_EXISTENCIA para cualquier fila que contenga un valor TIPO_MUSICA de Country y un valor EN_EXISTENCIA mayor a 30:

```
UPDATE INVENTARIO_CD
  SET EN_EXISTENCIA = (EN_EXISTENCIA - 2)
  WHERE TIPO_MUSICA = 'Country' AND EN_EXISTENCIA > 30;
```

Solamente una fila (Patsy Cline: 12 Greatest Hits) cumple con las condiciones de búsqueda especificadas en la cláusula WHERE. El valor EN_EXISTENCIA para esa fila ha sido cambiado de 32 a 30.

También es posible especificar múltiples expresiones en la cláusula SET. En otras palabras, se pueden cambiar los valores de más de una columna al mismo tiempo. Por ejemplo, supongamos que se quiere cambiar el valor EDITOR y el valor EN_EXISTENCIA de la fila Orlando. La instrucción UPDATE necesaria podría lucir de la siguiente forma:

```
UPDATE INVENTARIO_CD
  SET PUBLISHER = 'Sarabande Records',
      EN_EXISTENCIA = (EN_EXISTENCIA * 2)
  WHERE NOMBRE_CD = 'Orlando';
```

Observe que las dos expresiones en la cláusula SET están separadas por una coma. Cuando se ejecuta esta instrucción, el valor EDITOR se cambia de Independiente a Sarabande Records, y el valor EN_EXISTENCIA se cambia de 5 a 10. (El valor 5 se multiplica por 2.)

Algo que no se puede realizar, sin embargo, es cambiar el valor para la misma columna para dos diferentes filas si se está tratando de dar diferentes valores para esas filas. Veamos un ejemplo para hacer esto más claro. Supongamos que se quiere actualizar el valor para la fila Out of Africa y para la fila Fundamental, pero se quiere actualizar estas filas con diferentes valores. La fila Out of Africa debe contener el valor Soundtrack para TIPO_MUSICA, y la fila Fundamental debe con-

tener el valor Blues para TIPO_MUSICA. Como resultado, se puede intentar ejecutar una instrucción similar a la siguiente:

```
UPDATE INVENTARIO_CD
    SET TIPO_MUSICA = 'Soundtrack',
        TIPO_MUSICA = 'Blues'
WHERE NOMBRE_CD = 'Out of Africa' OR NOMBRE_CD = 'Fundamental';
```

Si se intenta ejecutar esta instrucción, la implementación de SQL no sabrá cuál valor de TIPO_MUSICA debe poner en qué fila, y arrojará un error. Para manejar una situación como ésta, se necesitarán crear dos instrucciones UPDATE separadas:

```
UPDATE INVENTARIO_CD
    SET TIPO_MUSICA = 'Soundtrack'
WHERE NOMBRE_CD = 'Orlando';
UPDATE INVENTARIO_CD
    SET TIPO_MUSICA = 'Blues'
WHERE NOMBRE_CD = 'Fundamental';
```

Actualizar valores desde una instrucción SELECT

En la sección “Insertar valores desde una instrucción SELECT” ya presentada en este capítulo, se dijo que se puede utilizar una instrucción SELECT en lugar de una cláusula VALUES. También se puede utilizar una instrucción SELECT en la cláusula SET de la instrucción UPDATE. La instrucción SELECT arroja el valor que está definido en la porción <expresión de valor> de la expresión determinar cláusula. En otras palabras, la instrucción SELECT se agrega a la derecha del signo de igual.

Pongamos algunos ejemplos para ver cómo funciona esto. Los siguientes ejemplos están basados en los datos originales de la tabla INVENTARIO_CD (mostrada en la figura 8-5) y la tabla INVENTARIO_CD_2 (mostrada en la figura 8-6). Supongamos que se quieren actualizar los datos en la tabla INVENTARIO_CD_2 utilizando valores de la tabla INVENTARIO_CD. Se puede crear una instrucción UPDATE que sea similar a la siguiente:

```
UPDATE INVENTARIO_CD_2
    SET EN_EXISTENCIA_2 =
        ( SELECT AVG(EN_EXISTENCIA)
          FROM INVENTARIO_CD );
```

La instrucción SELECT calcula el promedio de los valores EN_EXISTENCIA en la tabla INVENTARIO_CD, el cual es 24, por lo que determinar la expresión de la cláusula puede ser interpretado como sigue: EN_EXISTENCIA_2 = 24. Como resultado, todos los valores EN_EXISTENCIA_2 en la tabla INVENTARIO_CD_2 son determinados a 24. Desde luego, probablemente no se requiere que todos los valores de EN_EXISTENCIA_2 sean iguales, por lo que puede limitarse cuáles filas serán actualizadas al agregar una cláusula WHERE a la instrucción UPDATE:

```
UPDATE INVENTARIO_CD_2
    SET EN_EXISTENCIA_2 =
        ( SELECT AVG(EN_EXISTENCIA)
          FROM INVENTARIO_CD )
WHERE NOMBRE_CD_2 = 'Orlando';
```

Ahora, solamente la fila Orlando será actualizada y el valor de EN_EXISTENCIA_2 será cambiado a 24.

Incluso se puede agregar una cláusula WHERE a la instrucción SELECT, como se muestra en el siguiente ejemplo:

```
UPDATE INVENTARIO_CD_2
SET EN_EXISTENCIA_2 =
    ( SELECT EN_EXISTENCIA
      FROM INVENTARIO_CD
      WHERE NOMBRE_CD = 'Orlando' )
WHERE NOMBRE_CD_2 = 'Orlando';
```

En este caso, el valor EN_EXISTENCIA de 5 se toma directamente desde la fila Orlando de la tabla INVENTARIO_CD y se utiliza como la porción <expresión de valor> de la expresión determinar cláusula. Como resultado, determinar la expresión de la cláusula puede ser interpretado como lo siguiente: EN_EXISTENCIA_2 = 5. (Por supuesto, el valor en la tabla INVENTARIO_CD_2 no cambiará debido a que ya es 5, pero si tuviera cualquier otro valor diferente, éste se habría actualizado a 5.)

Se puede agregar una capa más de complejidad a la instrucción UPDATE modificando la cláusula SET todavía más. Por ejemplo, supongamos que se quiere incrementar el valor en 2 antes de insertarlo en la columna EN_EXISTENCIA_2. Para hacer eso, se puede cambiar la expresión de valor a la siguiente:

```
UPDATE INVENTARIO_CD_2
SET EN_EXISTENCIA_2 =
    ( SELECT EN_EXISTENCIA
      FROM INVENTARIO_CD
      WHERE NOMBRE_CD = 'Orlando' ) + 2
WHERE NOMBRE_CD_2 = 'Orlando';
```

Una vez más, la cláusula SELECT toma el valor de 5 desde la columna EN_EXISTENCIA de la tabla INVENTARIO_CD, pero esta vez el valor 2 es agregado al valor arrojado por la instrucción SELECT, resultando un total de 7. Como resultado, el nuevo determinar la expresión de la cláusula puede ser representado así: EN_EXISTENCIA_2 = (5) + 2. Si se ejecutara esta instrucción, el valor de EN_EXISTENCIA_2 cambiaría a 7 en la fila Orlando de la tabla INVENTARIO_CD_2.

Al combinar la cláusula SET con la cláusula WHERE, se pueden crear instrucciones UPDATE que puedan calcular valores muy específicos que se utilicen para modificar cualquier número de filas y columnas que se necesite actualizar. Sin embargo, al igual que con la instrucción INSERT, cualquier valor que se modifique deberá cumplir con las restricciones de la tabla. En otras palabras, los nuevos valores deben regirse por los tipos de datos, dominios y limitaciones aplicables.

Eliminar datos SQL

De todas las instrucciones para modificación de datos soportadas por SQL, probablemente la instrucción DELETE sea la más sencilla. Contiene solamente dos cláusulas, y una sola de ellas es obligatoria. La siguiente sintaxis muestra qué tan básica es la instrucción DELETE:

```
DELETE FROM <nombre de la tabla>
[ WHERE <condición de búsqueda> ]
```

Como se puede ver, la cláusula DELETE FROM requiere que se especifique el nombre de la tabla (o vista) de la cual se quieren eliminar filas. La cláusula WHERE, la cual es similar a la cláusula WHERE en una instrucción SELECT y en una instrucción UPDATE, requiere que se especifique una condición de búsqueda. Si no se incluye una cláusula WHERE en la instrucción DELETE, todas las filas serán eliminadas de la tabla especificada. Es importante comprender que la instrucción DELETE no elimina la tabla en sí, sino solamente filas en la tabla (la instrucción DROP TABLE, como se describió en el capítulo 3, se utiliza para eliminar definiciones de tabla de la base de datos).

NOTA

SQL soporta vistas de referencia en la instrucción DELETE, pero la eliminación real se realiza en filas en la tabla base. Prácticamente en ninguna de las implementaciones se pueden eliminar filas utilizando vistas que hagan referencia a más de una tabla base (véase la documentación del fabricante para detalles más específicos).

Observe que en la instrucción DELETE no hay nombres de columna especificados. Esto se debe a que no es posible eliminar valores de columna individuales desde una tabla. Solamente se pueden eliminar filas. Si se necesita eliminar un valor de columna específico, se deberá utilizar una instrucción UPDATE para determinar el valor a nulo. Pero sólo es posible realizar esto si en esa columna se soportan valores nulos.

Ahora veamos un par de ejemplos de la instrucción DELETE. El primer ejemplo elimina todos los datos (todas las filas) de la tabla INVENTARIO_CD, mostrado en la figura 8-5:

```
DELETE FROM INVENTARIO_CD;
```

Eso es todo lo que se necesita. Desde luego, deberá utilizarse esta instrucción solamente si se quieren eliminar *todos* los datos de INVENTARIO_CD. Aunque en ocasiones se puede ejecutar esta instrucción donde sea necesario eliminar cada fila de una tabla, es un poco más probable que se prefiera utilizar la cláusula WHERE, para especificar cuáles filas serán eliminadas. Se puede modificar la instrucción que acabamos de ver para eliminar solamente filas donde el valor TIPO_MUSICA sea Country:

```
DELETE FROM INVENTARIO_CD
WHERE TIPO_MUSICA = 'Country';
```

Cuando se ejecuta esta instrucción, todas las filas cuyo valor TIPO_MUSICA es Country serán eliminadas de la tabla INVENTARIO_CD, que en este caso es la fila Patsy Cline: 12 Greatest Hits.

Ahora modificamos la instrucción DELETE un poco más incluyendo dos predicados en la cláusula WHERE:

```
DELETE FROM INVENTARIO_CD
WHERE TIPO_MUSICA = 'Pop'
OR EDITOR = 'Independiente';
```

Esta instrucción eliminará cualquier fila en la tabla CD_INVENTORY que incluya a un valor TIPO_MUSICA de Pop o un valor EDITOR de Independiente, lo que significa que la fila Court and Spark y la fila Orlando serán eliminadas.

Como se puede ver, el número de filas que se eliminan de una tabla depende de las condiciones de búsqueda definidas dentro de la cláusula WHERE. Cuando no se especifica una cláusula

WHERE, todas las filas que arrojan un valor verdadero son eliminadas de la tabla. La cláusula WHERE permite especificar exactamente cuáles filas deberán ser eliminadas de la tabla.

Pruebe esto 8-1 Modificar datos SQL

En este ejercicio se utilizarán todas las instrucciones de modificación de datos analizadas en este capítulo para cambiar los datos en la base de datos INVENTARIO. Se utilizará la instrucción INSERT para agregar datos, la instrucción UPDATE para modificar los datos, y la instrucción DELETE para eliminar los datos de la base de datos. Debido a que se estará trabajando solamente con datos, no se afectará la estructura subyacente de las tablas. Se puede descargar el archivo Try_This_08.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. Primero se añadirá una nueva compañía a la tabla DISQUERAS_CD. La compañía es DRG Records y tendrá un valor ID_DISQUERA de 837. Ingrese y ejecute la siguiente instrucción SQL:

```
INSERT INTO DISQUERAS_CD
VALUES ( 837, 'DRG Records' );
```

Una fila se añadirá a la tabla DISQUERAS_CD.

3. Ahora agreguemos un nuevo CD a la tabla DISCOS_COMPACTOS. El CD tiene el nombre *Ann Hampton Callaway*, el cual tiene un valor ID_DISCO_COMPACTO de 116. Se encuentran 14 de estos CD en existencia y el valor ID_DISQUERA deberá ser 836. (Éste no es el valor ID_DISQUERA correcto, pero se utilizará aquí para propósitos de este ejercicio.) Ingrese y ejecute la siguiente instrucción SQL:

```
INSERT INTO DISCOS_COMPACTOS
VALUES ( 116, 'Ann Hampton Callaway', 836, 14 );
```

Una fila se añadirá a la tabla DISCOS_COMPACTOS. El valor ID_DISQUERA de 836 representa Sarabande Records.

4. Ahora agregaremos otra fila en la tabla DISCOS_COMPACTOS; sólo que esta vez, la instrucción INSERT especificará los nombres de las columnas de la tabla destino. Se insertará un CD llamado *Rhythm Country and Blues*. La nueva fila contendrá un valor ID_DISCO_COMPACTO de 117, un valor ID_DISQUERA de 832 (MCA Records) y un valor EN_EXISTENCIA de 21. Ingrese y ejecute la siguiente instrucción SQL:

```
INSERT INTO DISCOS_COMPACTOS
( ID_DISCO_COMPACTO, TITULO_CD, ID_DISQUERA, EN_EXISTENCIA )
VALUES ( 117, 'Rhythm Country and Blues', 832, 21 );
```

Una fila se añadirá a la tabla DISCOS_COMPACTOS.

5. Después de ingresar la fila Rhythm Country and Blues, se puede observar que el valor EN_EXISTENCIA es incorrecto y que es necesario actualizar ese valor a 25. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = 25
    WHERE ID_DISCO_COMPACTO = 117;
```

El valor EN_EXISTENCIA de la fila Rhythm Country and Blues será cambiado a 25.

6. Ahora se puede observar que se ingresó el valor incorrecto ID_DISQUERA para la fila Ann Hampton Callaway. Sin embargo, se quiere modificar el valor existente especificando el nombre de la compañía en lugar del valor ID_DISQUERA. El nombre de la compañía es DRG Records, que se añadió a la tabla DISQUERAS_CD en el paso 2. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE DISCOS_COMPACTOS
    SET ID_DISQUERA =
        ( SELECT ID_DISQUERA
          FROM DISQUERAS_CD
          WHERE NOMBRE_COMPAÑIA = 'DRG Records' )
    WHERE ID_DISCO_COMPACTO = 116;
```

En esta instrucción se utilizó la instrucción SELECT para tomar el valor ID_DISQUERA de la tabla DISQUERAS_CD. La instrucción arrojó un valor de 837. El valor 837 fue entonces utilizado como el valor ID_DISQUERA para la tabla DISCOS_COMPACTOS. Observe que no hubiera sido posible ingresar el valor de 837 en la columna ID_DISQUERA de la tabla DISCOS_COMPACTOS si no hubiera existido antes en la tabla DISQUERAS_CD. No sólo sucede esto debido a que una instrucción SELECT fue utilizada para tomar el valor, sino también debido a que la columna ID_DISQUERA en la tabla DISCOS_COMPACTOS es una clave externa que hace referencia a la tabla DISQUERAS_CD. Como resultado, el valor debe existir en la tabla referenciada antes de que pueda ser agregado a la tabla que hace referencia a ella. Véase el capítulo 4 para mayor información acerca de estas claves externas.

7. Ahora echemos un vistazo a los datos que se han ingresado y actualizado. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT *
    FROM DISCOS_COMPACTOS
    WHERE ID_DISCO_COMPACTO = 116
       OR ID_DISCO_COMPACTO = 117;
```

La instrucción SELECT pide los datos de todas las columnas en la tabla DISCOS_COMPACTOS, pero sólo para aquellas filas que tengan un valor ID_DISCO_COMPACTO de 116 o 117. Los resultados de la consulta deberán incluir dos filas. Verifique que la información en esas filas esté correcta. La fila Ann Hampton Callaway deberá tener un valor ID_DISQUERA de 837 y un valor EN_EXISTENCIA de 14, y la fila Rhythm Country and Blues deberá tener un valor ID_DISQUERA de 832 y un valor EN_EXISTENCIA de 25.

(continúa)

8. Ahora elimine las dos filas que se agregaron a la tabla DISCOS_COMPACTOS. Ingrese y ejecute la siguiente instrucción SQL:

```
DELETE FROM DISCOS_COMPACTOS
WHERE ID_DISCO_COMPACTO = 116
      OR ID_DISCO_COMPACTO = 117;
```

La fila Ann Hampton Callaway y Rhythm Country and Blues deberán haber sido eliminadas de la tabla DISCOS_COMPACTOS.

9. Por último elimine la fila que se añadió a la tabla DISQUERAS_CD. Ingrese y ejecute la siguiente instrucción SQL:

```
DELETE FROM DISQUERAS_CD
WHERE ID_DISQUERA = 837;
```

La fila DRG Records deberá haber sido eliminada de la tabla DISQUERAS_CD.

NOTA

Si se hubiera intentado eliminar esta fila anteriormente eliminando la fila Ann Hampton Callaway en la tabla DISCOS_COMPACTOS, se habría recibido un error debido a que el valor ID_DISQUERA en DISCOS_COMPACTOS hace referencia a la fila DRG Records en DISQUERAS_CD. La fila Ann Hampton Callaway tenía que ser eliminada primero, o el valor ID_DISQUERA tenía que ser cambiado a otro valor que cumpliera con la limitante de la clave externa.

10. Cierre la aplicación cliente.

Resumen de Pruebe esto

En este ejercicio se agregó una fila a la tabla ID_DISQUERA y dos filas a la tabla DISCOS_COMPACTOS. Luego se actualizaron las dos filas en la tabla DISCOS_COMPACTOS. Después de eso se eliminaron todas las filas que se habían creado. Al momento de terminar el ejercicio, la base de datos INVENTARIO deberá contener lo mismo que cuando se empezó el ejercicio. Como se puede ver, modificar los datos dentro de las tablas es un proceso bastante simple; sin embargo, las instrucciones de modificación de datos individuales pueden volverse mucho más complejas. Cuando aprenda técnicas más avanzadas para consultas de datos, usted será capaz de refinar sus instrucciones a un grado superior para que le proporcionen mayor flexibilidad al insertar, actualizar y eliminar datos.

Autoexamen Capítulo 8

1. ¿Cuál instrucción SQL deberá utilizarse para agregar datos a una tabla?
- A** SELECT
 - B** INSERT
 - C** UPDATE
 - D** DELETE

2. ¿Cuáles dos cláusulas son obligatorias en una instrucción INSERT?
3. ¿En cuál cláusula dentro de la instrucción INSERT se identifica la tabla que recibirá los nuevos datos?

4. Se crea la siguiente instrucción INSERT para agregar datos a la tabla ARTISTAS_INTERPRETES:

```
INSERT INTO ARTISTAS_INTERPRETES VALUES ( 12, 'Frank Sinatra' );
```

La tabla ARTISTAS_INTERPRETES incluye tres columnas. ¿Qué sucederá cuando se intente ejecutar esta instrucción?

5. ¿Qué información se debe especificar en la cláusula VALUES de una instrucción INSERT?
6. ¿Qué requerimientos deberán ser cumplidos por los valores en una cláusula VALUES?
7. Se está creando una instrucción INSERT para introducir datos en la tabla TIPOS_ARTISTA. La tabla incluye solamente dos columnas: ID_ART y NOMBRE_TIPO. Se quiere insertar una fila que incluye el valor ID_ART de 27 y el valor NOMBRE_TIPO de Gospel. ¿Cuál instrucción SQL deberá ser utilizada?
8. Se está creando una instrucción INSERT que inserta valores tomados desde otra tabla. ¿Qué tipo de instrucción o cláusula se puede utilizar en lugar de la cláusula VALUES para tomar datos desde la otra tabla?
 - A SELECT
 - B SET
 - C SELECT
 - D WHERE
9. ¿Cuál instrucción deberá utilizarse para modificar los datos existentes en una o más filas en una tabla?
 - A SELECT
 - B INSERT
 - C UPDATE
 - D DELETE
10. ¿Cuáles cláusulas son obligatorias en una instrucción UPDATE?
11. ¿Cuál es el propósito de la cláusula WHERE en una instrucción UPDATE?
12. Se está creando una instrucción UPDATE para actualizar los datos en la tabla ARTISTAS_INTERPRETES. Se quiere actualizar el valor ID_ART en la fila que contenga el valor ID_ART_INTER de 139. El nuevo valor ID_ART es 27. ¿Cuál instrucción SQL deberá ser utilizada?
13. Se está creando una instrucción UPDATE para actualizar los datos en la tabla ARTISTAS_INTERPRETES. Se quiere actualizar el valor ID_ART de cada fila a 27. ¿Cuál instrucción SQL deberá ser utilizada?

- 14.** Se están actualizando dos columnas en la tabla INVENTARIO_CD. Se quiere cambiar el valor EDITOR a MCA Records y se quiere duplicar el valor EN_EXISTENCIA. ¿Cuál cláusula SET deberá ser utilizada?
- 15.** Se está creando una instrucción UPDATE que incluye una cláusula SET con una expresión de valor. Se requiere que la expresión de valor tome un valor desde otra tabla en la base de datos. ¿Cuál instrucción o cláusula se puede utilizar como una expresión de valor para seleccionar datos desde otra tabla?
- A** SELECT
 - B** WHERE
 - C** UPDATE
 - D** INSERT
- 16.** ¿Cuál cláusula es requerida en una instrucción DELETE?
- 17.** ¿Cuál instrucción o cláusula se utiliza en una instrucción DELETE para especificar las filas que serán eliminadas en una tabla?

Capítulo 9

Utilizar predicados



Habilidades y conceptos clave

- Comparar datos SQL
 - Arrojar valores nulos
 - Arrojar valores similares
 - Hacer referencia a fuentes adicionales de datos
 - Determinar la cantidad de predicados de comparación
-

Hasta este punto en el libro se ha presentado gran cantidad de información acerca de diversos aspectos de los objetos de las bases de datos y los datos que éstos almacenan. En relación a esto, se analizó la consulta de datos (capítulo 7) y la modificación de datos (capítulo 8). Ahora es necesario dar un paso atrás y concentrarse en uno de los aspectos de estas discusiones: la cláusula `WHERE`. Ésta, como puede recordarse, permite especificar una condición de búsqueda que filtra aquellas filas que no se quiere que sean arrojadas por una instrucción `SELECT` o que sean modificadas por una instrucción `UPDATE` o `DELETE`. La condición de búsqueda incluye uno o más predicados que declaran cada uno un hecho acerca de la fila que será arrojada o modificada. SQL soporta varios tipos de predicados, y todos ellos permiten probar si una condición es verdadera, falsa o desconocida. Este capítulo se concentrará en aquellos predicados que son más comúnmente utilizados por programadores SQL, y se proporcionarán ejemplos sobre cómo son utilizados para ver y modificar los datos en una base de datos SQL.

Comparar datos SQL

El primer tipo de predicado que se planea analizar es el tipo de aquellos que comparan datos. Estos predicados, como cualquier predicado, están incluidos en la cláusula `WHERE`. Se puede incluir una cláusula `WHERE` en una instrucción `SELECT`, `UPDATE` o `DELETE`, y en cada caso la cláusula puede contener uno o más predicados de comparación.

Cada predicado en la cláusula `WHERE` (ya sea un predicado de comparación o de cualquier otro tipo) se evalúa en un principio individual para determinar si éste cumple la condición definida por ese predicado. Después de que se evalúan los predicados, la cláusula `WHERE` se evalúa como un todo. La cláusula debe evaluarse como verdadera si se requiere que una fila sea incluida en los resultados de una búsqueda, sea actualizada o eliminada. Si la cláusula se evaluara como falsa o desconocida, la fila no sería incluida o modificada. Para una discusión completa sobre cómo se evalúan los predicados y la cláusula `WHERE` véase el capítulo 7.

Un predicado de comparación es un tipo de predicado que compara los valores en una columna especificada con un valor especificado. Un operador de comparación se utiliza para comparar esos valores. Hemos visto ya un número de operadores de comparación (y subsecuentemente, predicados de comparación) a lo largo de este libro. La tabla 9-1 lista los seis operadores de comparación soportados por SQL y proporciona un ejemplo para cada uno.

Operador	Símbolo	Ejemplo
Igual a	=	EN_EXISTENCIA = 47
Desigual a	<>	EN_EXISTENCIA <> 47
Menor que	<	EN_EXISTENCIA < 47
Mayor que	>	EN_EXISTENCIA > 47
Menor que o igual a	<=	EN_EXISTENCIA <= 47
Mayor que o igual a	>=	EN_EXISTENCIA >= 47

Tabla 9-1 Operadores de comparación de SQL.

Sin duda usted reconocerá muchos de estos operadores, e incluso aquellos que no reconozca serán muy fáciles de comprender. Pero demos un vistazo a los ejemplos en la tabla 9-1 para comprender más cómo funciona un predicado de comparación. En la primera fila en la tabla (la fila Igual a), el predicado de ejemplo es EN_EXISTENCIA = 47. Si ésta fuera a aparecer en una cláusula WHERE, se vería de la manera siguiente:

```
WHERE EN_EXISTENCIA = 47
```

EN_EXISTENCIA es el nombre de la columna en la tabla identificada en la instrucción que contiene la cláusula WHERE. El signo de igual (=) es el operador de comparación que se utiliza para comparar los valores en la columna EN_EXISTENCIA con el valor a la derecha del signo de igual, el cual en este caso es 47. Por lo tanto, para que una fila sea evaluada como verdadera, el valor EN_EXISTENCIA para esa fila debe ser de 47. Todos los seis operadores de comparación funcionan de la misma manera. En cada caso, la cláusula WHERE debe evaluarse como verdadera para que la fila sea arrojada en los resultados de la consulta o para que sea modificada.

Aun cuando es tradicional colocar el nombre de la columna a la izquierda del operador de comparación y el valor de la constante a la derecha, pueden ser invertidos y formar una instrucción equivalente, asumiendo que también se ajuste el operador de comparación si fuera necesario. Por ejemplo, las siguientes dos cláusulas WHERE son lógicamente idénticas, y cada una selecciona filas con valores EN_EXISTENCIA mayores a 5:

```
WHERE EN_EXISTENCIA > 5
WHERE 5 < EN_EXISTENCIA
```

NOTA

Como se aprendió en el capítulo 7, es posible combinar predicados utilizando la palabra clave AND o la palabra clave OR para colocar juntos dos o más predicados en una cláusula WHERE. También se puede utilizar la palabra clave NOT para crear una condición inversa para un predicado particular. Recuerde, sin importar cuántos predicados sean incluidos en la cláusula WHERE, la cláusula aún debe evaluarse como verdadera para que una fila dada sea seleccionada.

Ahora que se ha dado un vistazo a los seis tipos de predicados de comparación, pongamos algunos ejemplos. Estos ejemplos se basan en la figura 9-1, que muestra los datos almacenados en la tabla CDS_A_LA_MANO.

TITULO_CD: VARCHAR(60)	DERECHOSDEAUTOR: INT	PRECIO_MENUDEO: NUMERIC(5,2)	INVENTARIO: INT
Famous Blue Raincoat	1991	16.99	6
Blue	1971	14.99	26
Court and Spark	1974	14.99	18
Past Light	1983	15.99	2
Kojiki	1990	15.99	5
That Christmas Feeling	1993	10.99	3
Patsy Cline: 12 Greatest Hits	1988	16.99	25

Figura 9-1 Comparación de datos en la tabla CDS_A_LA_MANO.

En el primer ejemplo que veremos, la cláusula WHERE utiliza un operador Igual a para comparar los valores en la columna TITULO_CD con uno de los títulos de CD:

```
SELECT TITULO_CD, DERECHOSDEAUTOR
FROM CDS_A_LA_MANO
WHERE TITULO_CD = 'Past Light';
```

Esta instrucción arrojará una fila con sólo dos valores (uno por cada columna especificada en la cláusula SELECT), como se muestra en los siguientes resultados de consulta:

```
TITULO_CD    DERECHOSDEAUTOR
-----
Past Light   1983
```

Ahora cambiemos un poco esta instrucción SELECT. En lugar de utilizar el operador Igual a, utilizaremos el operador Desigual a:

```
SELECT TITULO_CD, DERECHOSDEAUTOR
FROM CDS_A_LA_MANO
WHERE TITULO_CD <> 'Past Light';
```

Cuando se ejecuta esta instrucción, se arrojan seis filas:

```
TITULO_CD    DERECHOSDEAUTOR
-----
Famous Blue Raincoat   1991
Blue                   1971
Court and Spark        1974
Kojiki                 1990
That Christmas Feeling  1993
Patsy Cline: 12 Greatest Hits  1988
```

Observe que en los resultados de la consulta se incluyen todas las filas excepto la fila Past Light. En este caso, la cláusula WHERE se evalúa como verdadera solamente cuando el valor TITULO_CD es diferente de Past Light. Sin embargo, obsérvese que si una fila tuviera un valor nulo para TITULO_CD, esa fila no aparecería en ninguno de los conjuntos de resultados anteriores. Tenga en mente que una comparación con un valor nulo *siempre* se evaluará como desconocida (NULL) (el DBMS no puede saber si el valor es Igual a o Desigual a debido a que es un valor desconocido). Como se verá posteriormente en la sección “Arrojar valores nulos”, existen operadores especiales utilizados para tratar con los valores nulos.

Demos ahora un vistazo al operador Menor que y al operador Mayor que. En el siguiente ejemplo se combinan dos predicados de comparación utilizando la palabra clave AND:

```
SELECT TITULO_CD, INVENTARIO
FROM CDS_A_LA_MANO
WHERE INVENTARIO > 2
AND INVENTARIO < 25;
```

Como se puede ver, las filas arrojadas por esta instrucción SELECT deben contener un valor INVENTARIO entre 2 y 25. Si se ejecuta esta instrucción, se obtendrán cuatro filas:

TITULO_CD	INVENTARIO
Famous Blue Raincoat	6
Court and Spark	18
Kojiki	5
That Christmas Feeling	3

Al definir los predicados en una cláusula WHERE, no se está limitado a utilizar solamente una columna. Por ejemplo, si se desean seleccionar filas basadas tanto en los valores de la columna INVENTARIO y de la columna PRECIO_MENUDEO, se puede modificar la última instrucción SELECT como se muestra el siguiente ejemplo:

```
SELECT TITULO_CD, INVENTARIO
FROM CDS_A_LA_MANO
WHERE INVENTARIO > 2
AND INVENTARIO < 25
AND PRECIO_MENUDEO <> 16.99;
```

Debido a que las condiciones se conectan utilizando AND, cualquier fila que se arroje en los resultados de la consulta debe cumplir con las tres condiciones definidas en la cláusula WHERE. Como resultado, solamente se obtienen tres filas cuando se ejecuta esta instrucción:

TITULO_CD	INVENTARIO
Court and Spark	18
Kojiki	5
That Christmas Feeling	3

Observe que los resultados de la consulta no incluyen a la columna PRECIO_MENUDEO. Esto se debe a que esta columna no está especificada en la cláusula SELECT. Aun así, se puede utilizar esa columna en un predicado en la cláusula WHERE para definir una condición de búsqueda.

Ahora demos un vistazo al operador Menor que o igual a y al operador Mayor que o igual a. En el siguiente ejemplo, ambos operadores se utilizan para limitar las filas resultantes a aquellas con un valor DERECHOSDEAUTOR con un rango entre 1971 y 1989.

```
SELECT TITULO_CD, DERECHOSDEAUTOR
FROM CDS_A_LA_MANO
WHERE DERECHOSDEAUTOR >= 1971
      AND DERECHOSDEAUTOR <= 1989;
```

Esta instrucción arrojará resultados ligeramente diferentes a aquellos que se habrían obtenido si simplemente se hubieran utilizado los operadores Mayor que y Menor que. Utilizando el operador Mayor que o igual a y el operador Menor que o igual a, los valores iguales al valor especificado también son arrojados, como se muestra en los siguientes resultados de consulta:

TITULO_CD	DERECHOSDEAUTOR
-----	-----
Blue	1971
Court and Spark	1974
Past Light	1983
Patsy Cline: 12 Greatest Hits	1988

Observe que la fila Blue incluye un valor DERECHOSDEAUTOR de 1971. Éste no habría sido incluido si se hubiera utilizado solamente el operador Mayor que.

Hasta este punto, todos los ejemplos mostrados han estado basados en las instrucciones SELECT. Sin embargo, se puede agregar una cláusula WHERE a una instrucción UPDATE o a una instrucción DELETE. Supongamos que se quiere incrementar el valor INVENTARIO para la fila That Christmas Feeling. Se puede utilizar la siguiente instrucción UPDATE:

```
UPDATE CDS_A_LA_MANO
SET INVENTARIO = 10
WHERE TITULO_CD = 'That Christmas Feeling';
```

Cuando se ejecuta esta instrucción, el valor INVENTARIO es incrementado a 10 para la fila That Christmas Feeling, pero únicamente para esa fila debido a que la cláusula WHERE se evalúa como verdadera sólo para esa fila. De manera muy sencilla se pudo haber agregado esta cláusula WHERE a una instrucción DELETE, en cuyo caso la fila That Christmas Feeling hubiera sido eliminada.

Al igual que con la cláusula WHERE en una instrucción SELECT, es posible combinar dos o más predicados para formar una condición de búsqueda:

```
UPDATE CDS_A_LA_MANO
SET INVENTARIO = 3
WHERE TITULO_CD = 'That Christmas Feeling'
      AND DERECHOSDEAUTOR = 1993;
```

Cuando se especifica la palabra clave AND, ambos predicados deben evaluarse como verdaderos para que la cláusula WHERE pueda evaluarse como verdadera. Si se especifica la palabra clave OR en lugar de AND, entonces sólo será necesario que un predicado se evalúe como verdadero para que la fila sea seleccionada.

Utilizar el predicado BETWEEN

Hablando estrictamente, el predicado BETWEEN no es un predicado de comparación, al menos no lo es tal como se presenta en SQL:2006 estándar. Sin embargo, es lo suficientemente similar en funciones al operador Mayor que o Igual a y a los operadores Menor que o Igual a, que vale la pena analizar aquí.

El predicado BETWEEN se utiliza en conjunción con la palabra clave AND para identificar un rango de valores que pueden ser incluidos como una condición de búsqueda en la cláusula WHERE. Los valores en la columna identificada deben entrar en ese rango para poder evaluarse como verdaderos. Cuando se utiliza la cláusula BETWEEN, se debe especificar la columna aplicable, el valor más bajo del rango y el valor más alto del rango. El siguiente ejemplo (que está basado en la tabla CDS_A_LA_MANO en la figura 9-1) especifica un rango entre 14 y 16.

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO BETWEEN 14 AND 16;
```

El valor PRECIO_MENUDEO para cada fila seleccionada debe entrar en ese rango, incluyendo los puntos límite. Si se ejecuta esta instrucción, solamente cuatro filas serán incluidas en los resultados de la consulta:

TITULO_CD	PRECIO_MENUDEO
-----	-----
Blue	14.99
Court and Spark	14.99
Past Light	15.99
Kojiki	15.99

Ahora demos un vistazo a una consulta similar a la presentada en el último ejemplo, sólo que esta vez utilizando los predicados de comparación en lugar del predicado BETWEEN:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO >= 14
AND PRECIO_MENUDEO <= 16;
```

Observe que se utilizan dos predicados: uno con el operador Mayor que o igual a, y el otro con el operador Menor que o igual a. Esta instrucción SELECT producirá los mismos resultados de búsqueda que la instrucción SELECT anterior.

Ahora regresemos al predicado BETWEEN. Al igual que con cualquier predicado, se puede combinar el predicado BETWEEN con otros predicados. En la siguiente instrucción, la cláusula WHERE incluye un predicado BETWEEN y un predicado de comparación:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO BETWEEN 14 AND 16
AND INVENTARIO > 10;
```

Como resultado de ambos predicados, los resultados de la consulta pueden incluir solamente aquellas filas con un valor `PRECIO_MENUDEO` que quepa en el rango entre 14 y 16 y con un valor `INVENTARIO` mayor a 10. Cuando se ejecuta esta consulta, solamente se obtienen dos filas:

TITULO_CD	PRECIO_MENUDEO
-----	-----
Blue	14.99
Court and Spark	14.99

Una vez más, puede notarse que los resultados de la consulta no incluyen a la columna `INVENTARIO` incluso cuando esa columna se especifica en un predicado dentro de la cláusula `WHERE`. Observe también que se hace referencia a más de una columna en la cláusula `WHERE`.

Además de lo que se ha visto hasta ahora para el predicado `BETWEEN`, también puede utilizarse la cláusula para especificar el inverso de una condición. Esto se hace utilizando la palabra clave `NOT` dentro del predicado. Por ejemplo, supongamos que se cambia el último ejemplo a lo siguiente:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO NOT BETWEEN 14 AND 16;
```

Las filas arrojadas en los resultados de la consulta incluirán todas las filas que no tengan un valor `PRECIO_MENUDEO` dentro del rango entre 14 y 16. Cuando se ejecuta la instrucción, se obtienen tres filas:

TITULO_CD	PRECIO_MENUDEO
-----	-----
Famous Blue Raincoat	16.99
That Christmas Feeling	10.99
Patsy Cline: 12 Greatest Hits	16.99

Observe que todos los valores dentro del rango especificado han sido excluidos de los resultados de la consulta. Si usted encuentra confusa la función de la palabra clave `NOT` (como le pasa a mucha gente), puede escribir una cláusula equivalente `WHERE` o utilizar la palabra clave `OR` y los operadores Mayor que y Menor que de la manera siguiente:

```
WHERE PRECIO_MENUDEO < 14
      OR PRECIO_MENUDEO > 16;
```

Arrojar valores nulos

Como se puede recordar del capítulo 4, un valor nulo es utilizado en lugar de un valor cuando dicho valor no está definido o no es conocido. Un valor nulo indica que el valor está ausente. Esto no es lo mismo que un valor cero, un espacio en blanco o un valor por defecto. Por opción predeterminada, SQL permite utilizar valores nulos en lugar de valores regulares (aunque se puede anular el valor por opción predeterminada incluyendo una restricción `NOT NULL` en la definición de la columna). En aquellos casos donde se permiten los valores nulos, puede ser necesario especificar que esos valores nulos sean arrojados cuando se consulta una tabla. Por esta razón, SQL proporciona un predicado `NULL`, que permite definir las condiciones de búsqueda que arrojaran los valores nulos.

El predicado NULL resulta muy simple de implementar. Utilizado en conjunción con la palabra clave IS, el predicado se agrega a una cláusula WHERE de la misma forma que cualquier otro predicado, y se aplica sólo a los valores nulos que pudieran existir en la columna que se está consultando. La mejor forma de ilustrar esto es mediante el uso de ejemplos. En estos ejemplos se utiliza la tabla BIO_ARTISTAS, mostrada en la figura 9-2.

El primer ejemplo es una instrucción SELECT que arroja filas con un valor LUGAR_DE_NACIMIENTO nulo:

```
SELECT *
  FROM BIO_ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NULL;
```

La instrucción arroja todas las columnas de la tabla BIO_ARTISTAS; sin embargo, arroja solamente dos filas, como se puede ver en los siguientes resultados de la consulta (los resultados pueden lucir diferentes dependiendo de cómo la aplicación cliente de SQL despliega los valores nulos):

NOMBRE_INTERPRETE	LUGAR_DE_NACIMIENTO	AÑO_NACIMIENTO
William Ackerman	NULL	NULL
Bing Crosby	NULL	1904

El hecho de que la columna AÑO_NACIMIENTO contenga un valor nulo para la fila William Ackerman no tiene relación directa con el hecho de que se utilice un predicado NULL. En este caso el predicado NULL solamente identifica a la columna LUGAR_DE_NACIMIENTO, no a la columna AÑO_NACIMIENTO. Sin embargo, es posible reemplazar la columna LUGAR_DE_NACIMIENTO en el predicado con la columna AÑO_NACIMIENTO, en cuyo caso las filas arrojadas serán aquellas con un valor AÑO_NACIMIENTO nulo.

NOMBRE_INTERPRETE: VARCHAR(60)	LUGAR_DE_NACIMIENTO CARCHAR(60)	AÑO_NACIMIENTO: INT
Jennifer Warnes	Seattle, Washington, USA	1947
Joni Mitchell	Fort MacLeod, Alberta, Canada	1943
William Ackerman	NULL	NULL
Kitaro	Toyohashi, Japan	NULL
Bing Crosby	NULL	1904
Patsy Cline	Winchester, Virginia, United States	1932
Jose Carreras	Barcelona, Spain	NULL
Luciano Pavarotti	Modena, Italy	1935
Placido Domingo	Madrid, Spain	1941

Figura 9-2 Valores nulos arrojados de la tabla BIO_ARTISTAS.

De acuerdo al estándar SQL:2006, también es posible especificar ambas columnas en el predicado NULL, como se muestra en el siguiente ejemplo:

```
SELECT *
  FROM BIO_ARTISTAS
 WHERE (LUGAR_DE_NACIMIENTO, AÑO_NACIMIENTO) IS NULL;
```

Cuando se incluyen ambas columnas, tanto la columna LUGAR_DE_NACIMIENTO como la columna AÑO_NACIMIENTO deben arrojar valores nulos para que una fila aparezca en la búsqueda, que en el caso de la tabla BIO_ARTISTAS sería solamente una fila.

NOTA

A pesar de que el estándar SQL permite especificar múltiples columnas en el predicador NULL, existen muchas implementaciones que no pueden soportar esto. En su lugar, se deben especificar dos predicados NULL conectados con la palabra clave AND.

Como una alternativa para incluir ambas columnas en un predicado, se puede escribir la instrucción SELECT de la forma siguiente:

```
SELECT *
  FROM BIO_ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NULL
        AND AÑO_NACIMIENTO IS NULL;
```

Si se ejecuta esta instrucción, se obtendrán los siguientes resultados de consulta:

PERFORMER_NAME	LUGAR_DE_NACIMIENTO	AÑO_NACIMIENTO
-----	-----	-----
William Ackerman	NULL	NULL

SQL soporta otra característica en el predicador NULL. Se puede utilizar la palabra clave NOT para encontrar los resultados inversos de predicado. Por ejemplo, supongamos que se quieren obtener todas las filas que incluyan un valor actual en la columna LUGAR_DE_NACIMIENTO en lugar de un valor nulo. La instrucción podría lucir de esta manera:

```
SELECT *
  FROM BIO_ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NOT NULL;
```

Los resultados de la consulta ahora incluirán siete filas, todas ellas conteniendo valores en la columna LUGAR_DE_NACIMIENTO:

NOMBRE_INTERPRETE	LUGAR_DE_NACIMIENTO	AÑO_NACIMIENTO
-----	-----	-----
Jennifer Warnes	Seattle, Washington, Estados Unidos	1947
Joni Mitchell	Fort MacLeod, Alberta, Canadá	1943
Kitaro	Toyohashi, Japón	NULL
Patsy Cline	Winchester, Virginia, Estados Unidos	1932
Jose Carreras	Barcelona, España	NULL

Luciano Pavarotti	Modena, Italia	1935
Plácido Domingo	Madrid, España	1941

Observe que los valores nulos pueden aún existir en otras columnas. Debido a que solamente la columna LUGAR_DE_NACIMIENTO está especificada en el predicado NULL, solamente esa columna debe contener un valor para que una fila sea arrojada.

Al igual que con los predicados que se vieron anteriormente en este capítulo, se puede combinar el predicado NULL con otros tipos de predicados. Por ejemplo, se puede modificar el último ejemplo para limitar los valores de AÑO_NACIMIENTO a sólo ciertos años, como se muestra en el siguiente ejemplo:

```
SELECT *
FROM BIO_ARTISTAS
WHERE LUGAR_DE_NACIMIENTO IS NOT NULL
AND AÑO_NACIMIENTO > 1940;
```

Ahora cualquier fila arrojada deberá incluir un valor en la columna LUGAR_DE_NACIMIENTO y el valor de AÑO_NACIMIENTO deberá ser mayor a 1940. Si se ejecuta esta consulta, se arrojarán los siguientes resultados:

NOMBRE_INTERPRETE	LUGAR_DE_NACIMIENTO	AÑO_NACIMIENTO
Jennifer Warnes	Seattle, Washington, Estados Unidos	1947
Joni Mitchell	Fort MacLeod, Alberta, Canadá	1943
Plácido Domingo	Madrid, España	1941

Como se puede ver, solamente se obtienen tres filas. Ninguna fila con valor LUGAR_DE_NACIMIENTO nulo es arrojada debido a que el valor nulo se evalúa como desconocido, y solamente las cláusulas WHERE que se evalúan como verdaderas pueden ser incluidas en los resultados de la consulta.

Arrojar valores similares

Si algún predicado puede ser divertido, éste es el predicado LIKE. El predicado LIKE proporciona un ambiente flexible en el cual es posible especificar valores que son solamente *similares* a los valores almacenados en la base de datos. Esto es particularmente beneficioso si sólo se conoce parte de un valor pero aún se necesita recuperar información basada en ese valor. Por ejemplo, supongamos que no se conoce el título completo de un CD, sino solamente una parte de ese título. O quizá sólo se conoce una parte del nombre del artista. Al utilizar el predicado LIKE, se pueden solicitar valores que sean similares a la parte que se conoce y desde esos resultados determinar si la información que se necesita está ahí.

Antes de dar un vistazo al predicado LIKE en sí mismo, veamos dos símbolos utilizados dentro del predicado. El predicado LIKE utiliza dos caracteres especiales, el signo de porcentaje (%) y el guión bajo (_), para ayudar a definir las condiciones de búsqueda especificadas en el predicado. El signo de porcentaje representa cero o más caracteres desconocidos, y el guión bajo representa exactamente un carácter desconocido. Se pueden utilizar estos caracteres al inicio de un valor, en medio de él, o al final, y se pueden combinar entre sí según sea necesario. La forma en que se utilizan estos dos caracteres determina el tipo de datos que serán recuperados de la base de datos.

Valor de ejemplo	Posibles resultados de la consulta
'J%'	Jennifer Warnes, Joni Mitchell, Jose Carreras
'%Spark'	Court and Spark
'%Blue%'	Famous Blue Raincoat, Blue, Blues on the Bayou
'%Cline%Hits'	Patsy cline: 12 Greatest Hits
'194_'	1940, 1941, 1947
'19__'	1900, 1907, 1938, 1963, 1999
'__ue'	Blue
'9__01'	90201, 91401, 95501, 99301, 99901
'9_3%'	9032343, 903, 95312, 99306, 983393300333

Tabla 9-2 Utilizar caracteres especiales en un predicado LIKE.

La tabla 9-2 proporciona varios ejemplos de cómo pueden ser utilizados estos caracteres especiales en un predicado LIKE.

Como se puede ver, los caracteres de signo de porcentaje y el guión bajo proporcionan una gran flexibilidad y permiten consultar un amplio rango de datos.

NOTA

Algunas implementaciones de los fabricantes pueden ser configuradas para ser insensibles al uso de mayúsculas y minúsculas para las comparaciones de datos, lo que significa que las letras en minúsculas o en mayúsculas en una comparación se consideran iguales. De hecho, éste es el comportamiento por defecto para SQL Server y Sybase; sin embargo, Oracle siempre es sensible a mayúsculas y minúsculas. Observando la tabla 9-2, las letras en mayúscula en la columna Valor de ejemplo pueden ser cambiadas a minúsculas con los mismos resultados siempre que DBMS esté configurado como insensible al uso de mayúsculas y minúsculas. Por ejemplo, el valor %spark del predicado LIKE aún seleccionaría a la fila Court and Spark en una implementación insensible al uso de mayúsculas y minúsculas.

Ahora que están mejor comprendidos los caracteres especiales, demos un vistazo al predicado LIKE como un todo. El predicado LIKE incluye el nombre de la columna, la palabra clave LIKE, y un valor encerrado en un conjunto de comillas simples, a su vez encerrado en un conjunto de paréntesis (los paréntesis son opcionales para la mayoría de las implementaciones de los fabricantes). Por ejemplo, la siguiente cláusula WHERE incluye un predicado LIKE:

```
WHERE ID_CD LIKE ('%01')
```

El predicado incluye la columna ID_CD, la palabra clave LIKE y un valor de %01. Sólo las filas que contengan el valor correcto en la columna ID_CD son arrojadas en los resultados de la consulta. La columna ID_CD es parte de la tabla CDS, que se muestra en la figura 9-3. Se estará utilizando esta tabla para los ejemplos en esta sección. Observe que basándose en el predicado LIKE definido en la cláusula WHERE precedente, solamente una fila puede ser arrojada por esta cláusula, la fila con el valor ID_CD de 99301.

ID_CD: INT	TITULO_CD: VARCHAR(60)
99301	Famous Blue Raincoat
99302	Blue
99303	Court and Spark
99304	Past Light
99305	Kojiki
99306	That Christmas Feeling
99307	Patsy Cline: 12 Greatest Hits

Figura 9-3 Arrojar valores similares desde la tabla CDS.

Ahora demos un vistazo a algunos ejemplos de instrucciones SELECT que incluyan un predicado LIKE. Supongamos que se quiere encontrar cualquier CD que contenga la palabra Christmas en el título. Se puede crear la siguiente instrucción SELECT para consultar la tabla CDS:

```
SELECT *
  FROM CDS
 WHERE TITULO_CD LIKE ('%Christmas%');
```

Los resultados de esta consulta incluirán solamente una fila:

```
ID_CD      TITULO_CD
-----
99306      That Christmas Feeling
```

Si se hubiera incluido solamente un signo de porcentaje, no se habría obtenido ninguna fila. Por ejemplo, si se eliminara el primer signo de porcentaje, la implementación SQL habría interpretado esto como el significado de que el valor debe comenzar con la palabra Christmas, lo cual no es así. Lo mismo sucede para el otro signo de porcentaje. Si éste se hubiera eliminado, la implementación habría asumido que Christmas debe ser la última palabra de la cadena de caracteres. Además de esto, si no se hubiera utilizado ningún signo de porcentaje, ninguna fila hubiera sido arrojada porque ningún valor habría coincidido exactamente con la palabra Christmas.

También se puede agregar la palabra clave NOT a un predicado LIKE si quieren obtenerse todas las filas, excepto aquellas especificadas por el predicado. Tomemos el último ejemplo. Si se agregara la palabra clave NOT, luciría de la siguiente manera:

```
SELECT *
  FROM CDS
 WHERE TITULO_CD NOT LIKE ('%Christmas%');
```


Esta vez los resultados de la consulta incluyen todas las filas que no contengan la palabra Christmas:

ID_CD	TITULO_CD
-----	-----
99301	Famous Blue Raincoat
99302	Blue
99303	Court and Spark
99304	Past Light
99305	Kojiki
99307	Patsy Cline: 12 Greatest Hits

Observe que ahora falta la fila That Christmas Feeling.

También se puede combinar un predicado LIKE con otro predicado LIKE. Supongamos, por ejemplo, que aún se quiere excluir el valor Christmas, pero se quiere incluir el valor Blue, como se muestra en el siguiente ejemplo:

```
SELECT *
FROM CDS
WHERE TITULO_CD NOT LIKE ('%Christmas%')
AND TITULO_CD LIKE ('%Blue%');
```

La cláusula WHERE en la instrucción SELECT elimina cualquier fila que tenga la palabra Christmas en cualquier parte del valor TITULO_CD. Además, el valor TITULO_CD debe incluir la palabra Blue. Como resultado, sólo se obtienen dos filas.

ID_CD	TITULO_CD
-----	-----
99301	Famous Blue Raincoat
99302	Blue

Pero ¿qué sucede si el título del CD incluye ambas palabras? Por ejemplo, *Blue Christmas* de Elvis Presley que está ahora disponible en CD. La palabra clave AND utilizada para conectar los predicados significa que *ambos* predicados deben ser verdaderos para que una fila pueda ser arrojada. Incluso si existía una fila Blue Christmas, no sería incluida en los resultados de la consulta debido a que el primer predicado (el que tiene NOT LIKE) sería evaluado como falso.

Pruebe esto 9-1

Utilizar predicados en instrucciones SQL

Antes de moverse hacia otros predicados, es una buena idea revisar los predicados que ya han sido discutidos. Éstos incluyen los seis tipos de predicados de comparación, el predicado BETWEEN, el predicado NULL y el predicado LIKE. En este ejercicio se probarán varios de estos predicados mediante el uso de instrucciones SELECT que incluirán las cláusulas WHERE apropiadas. Se consultarán las tablas que se crearon en la base de datos INVENTARIO. Debido a que se utilizarán solamente instrucciones SELECT, de cualquier manera no se modificarán las tablas o las estructuras de las bases de datos. Solamente se solicitarán datos basándose en los predicados que se definen. Puede descargar el archivo Try_This_09.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para sus RDBMS y conéctese con la base de datos INVENTARIO.
2. En la primera instrucción creada, se consultará la tabla TIPOS_MUSICA para arrojar los nombres de aquellas filas cuyo valor ID_TIPO sea igual a 11 o 12. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_TIPO, NOMBRE_TIPO
FROM TIPOS_MUSICA
WHERE ID_TIPO = 11
      OR ID_TIPO = 12;
```

La instrucción deberá arrojar dos filas, una para Blues y otra para Jazz. Observe que la palabra clave OR se utiliza para indicar que cualesquiera de los dos valores es aceptable.

3. Ahora se consultará la tabla ARTISTAS para buscar artistas diferentes a Patsy Cline y Bing Crosby. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT NOMBRE_ARTISTA, LUGAR_DE_NACIMIENTO
FROM ARTISTAS
WHERE NOMBRE_ARTISTA <> 'Patsy Cline'
      AND NOMBRE_ARTISTA <> 'Bing Crosby';
```

La consulta deberá arrojar 16 filas y no deberá incluir las filas Patsy Cline o Bing Crosby.

4. Ahora combinemos un par de predicados de comparación para crear un tipo diferente de condición de búsqueda. En esta instrucción se consultará una vez más la tabla ARTISTAS, pero ahora sólo se petitionarán aquellas filas cuyos valores ID_ARTISTA se encuentren entre 2004 y 2014 (excluyendo los puntos límite). Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_ARTISTA, NOMBRE_ARTISTA
FROM ARTISTAS
WHERE ID_ARTISTA > 2004
      AND ID_ARTISTA < 2014;
```

La consulta deberá arrojar nueve filas.

5. Ahora modifiquemos la instrucción SELECT que se acaba de ejecutar. Deberá utilizarse un predicado BETWEEN en lugar de los dos predicados de comparación. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_ARTISTA, NOMBRE_ARTISTA
FROM ARTISTAS
WHERE ID_ARTISTA BETWEEN 2004 AND 2014;
```

Ahora deberán aparecer 11 filas, en lugar de las nueve que se arrojaron en el paso anterior, debido a que BETWEEN siempre incluye los puntos límite. Si en el paso anterior se hubiera

(continúa)

utilizado el operador Mayor que o igual a y el operador Menor que o igual a, los resultados de la consulta habrían sido los mismos que en este paso.

- 6.** Ahora consultemos una vez más la tabla ARTISTAS, sólo que esta vez se utilizará el predicado NULL. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT *
  FROM ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NULL;
```

La búsqueda no arrojará resultados debido a que la columna LUGAR_DE_NACIMIENTO no contiene valores nulos.

- 7.** Intentemos la misma consulta que en el paso anterior, sólo que esta vez se agregará la palabra clave NOT al predicado NULL. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT *
  FROM ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NOT NULL;
```

La consulta ahora deberá arrojar todas las filas de la tabla (18 en total).

- 8.** En la siguiente instrucción se utilizará el predicado LIKE para encontrar títulos de CD que incluyan la palabra Best o la palabra Greatest. El predicado hará referencia a la columna TITULO_CD de la tabla DISCOS_COMPACTOS. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE TITULO_CD LIKE ('%Greatest%')
        OR TITULO_CD LIKE ('%Best%');
```

La consulta deberá arrojar tres filas. En todas estas filas, el valor TITULO_CD deberá contener las palabras Greatest o Best.

- 9.** Esta vez se modificará la instrucción en el paso anterior para incluir la palabra clave NOT en ambos predicados. También deberá cambiarse la palabra clave OR por la palabra AND. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE TITULO_CD NOT LIKE ('%Greatest%')
        AND TITULO_CD NOT LIKE ('%Best%');
```

Los resultados de la consulta deberán ahora incluir 12 filas. Si no se hubiera cambiado la palabra clave OR por AND, los resultados habrían incluido todas las 15 filas. Esto se debe a que la instrucción *siempre* se habría evaluado como verdadera (el primer predicado se evaluaría como verdadero para una fila que contenga Greatest; el segundo predicado se evaluaría como verdadero para una fila que contenga Best, y ambos predicados se evaluarían como verdaderos para cualquier otra fila).

- 10.** Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio se crearon varias instrucciones `SELECT` que incluían varios predicados. Los predicados estaban contenidos en cláusulas `WHERE` que eran parte de las instrucciones `SELECT`; sin embargo, estas cláusulas también pudieron haber sido parte de las instrucciones `UPDATE` y `DELETE`. Al avanzar en el resto de este capítulo, se aprenderá acerca de otros predicados y cómo pueden ser utilizados en diferentes tipos de instrucciones. Estos predicados pueden ser utilizados en conjunción con los que ya han sido analizados utilizándolos por sí mismos para crear condiciones de búsqueda más complejas y para arrojar resultados más precisos.

Hacer referencia a fuentes adicionales de datos

SQL soporta muchos tipos de predicados que permiten hacer referencia a fuentes diferentes a la tabla principal que se está consultando o modificando. Como resultado, se pueden crear condiciones de búsqueda que comparen datos entre tablas para determinar cuáles filas deberán ser incluidas en los resultados de la consulta, cuáles filas deberán ser actualizadas o cuáles eliminadas. En esta sección se verán dos predicados importantes que se pueden utilizar para hacer referencia a otras tablas: el predicado `IN` y el predicado `EXISTS`.

Ambos predicados pueden utilizar subconsultas para hacer referencia a datos en la tabla que se está consultando o modificando, o más comúnmente, a otras tablas. Se introdujo por primera vez el tema de las subconsultas en el capítulo 4. Como se puede recordar de ese capítulo, una subconsulta es una expresión que se utiliza como un componente dentro de otra expresión. En su uso más común, una subconsulta es simplemente una instrucción `SELECT` incrustada dentro de otra instrucción. Cuando se utiliza en un predicado, una subconsulta se convierte en parte de ese predicado y consecuentemente está incrustada en la cláusula `WHERE` de una instrucción `SELECT`, `UPDATE` o `DELETE`. A pesar de que las subconsultas se analizan con detalle en el capítulo 12, se mencionan aquí debido a que son una parte integral de los predicados que se estarán discutiendo en la siguiente parte de este capítulo. En cada uno de estos predicados, las subconsultas se utilizan para hacer referencia a datos en otras tablas. Para los propósitos de este capítulo, se mantendrán los ejemplos y las subconsultas simples, pero tenga en mente que éstas pueden ser mucho más elaboradas que lo que se verá aquí, y una vez que haya completado el capítulo 12, será capaz de aplicar ese conocimiento a los predicados que se aprendieron en este capítulo.

Utilizar el predicado `IN`

El predicado `IN` permite determinar si los valores en la columna especificada de una tabla están contenidos en una lista definida o contenidos dentro de otra tabla. En el primer caso, se debe especificar el nombre de la columna, la palabra clave `IN` y una lista de valores que son comparados a los valores en la columna especificada. En el segundo caso, se debe especificar el nombre de la columna, la palabra clave `IN` y una subconsulta, que hace referencia a la segunda tabla. En cada caso, si el valor de la columna coincide con uno de los valores en la lista o en los resultados de la subconsulta, el predicado se evalúa como verdadero y la fila es arrojada en los resultados de la consulta.

INVENTARIO_DISCO_COMPACTO		ARTISTAS_DISCO_COMPACTO	
NOMBRE_CD: VARCHAR(60)	EN_EXISTENCIA: INT	TITULO: VARCHAR(60)	ARTISTA: VARCHAR(60)
Famous Blue Raincoat	13	Famous Blue Raincoat	Jennifer Warnes
Blue	42	Blue	Joni Mitchell
Court and Spark	22	Court and Spark	Joni Mitchell
Past Light	17	Past Light	William Ackerman
Kojiki	6	Kojiki	Kitaro
That Christmas Feeling	8	That Christmas Feeling	Bing Crosby
Out of Africa	29	Patsy Cline: 12 Greatest Hits	Patsy Cline
Blues on the Bayou	27	After the Rain: The Soft Sounds of Erik Satie	Pascal Roge
Orlando	5	Out of Africa	John Barry
		Leonard Cohen The Best of	Leonard Cohen
		Fundamental	Bonnie Raitt
		Blues on the Bayou	B.B. King
		Orlando	David Motion

Figura 9-4 Consultar datos de la tabla INVENTARIO_DISCO_COMPACTO y de la tabla ARTISTAS_DISCO_COMPACTO.

La mejor forma de ilustrar ambos de estos métodos es por medio de ejemplos. Sin embargo, antes de verlos, refiérase a las tablas mostradas en la figura 9-4. Éstas son las tablas que se estarán utilizando para los ejemplos.

Como ya se mencionó, el primer método para utilizar el predicado IN es definir una lista. La lista deberá incluir todos los valores que van a ser comparados a los valores en la columna especificada. Por ejemplo, supongamos que se quieren limitar los resultados de la consulta a las filas en la tabla INVENTARIO_DISCO_COMPACTO que tengan un valor EN_EXISTENCIA de 12, 22, 32 o 42. Se puede crear una instrucción SELECT como la siguiente:

```
SELECT NOMBRE_CD, EN_EXISTENCIA FROM INVENTARIO_DISCO_COMPACTO
WHERE EN_EXISTENCIA IN ( 12, 22, 32, 42 );
```

Esta instrucción arroja solamente dos filas debido a que son las únicas filas que tienen los valores EN_EXISTENCIA correctos:

NOMBRE_CD	EN_EXISTENCIA
-----	-----
Blue	42
Court and Spark	22

Como se puede ver, es un proceso bastante sencillo utilizar el predicado IN para definir una lista, y es bastante útil cuando se sabe exactamente con cuáles valores se quieren comparar las columnas. También es un método más simple que definir predicados separados para cada valor, como en el siguiente ejemplo:

```
SELECT NOMBRE_CD, EN_EXISTENCIA
FROM INVENTARIO_DISCO_COMPACTO
WHERE EN_EXISTENCIA = 12
      OR EN_EXISTENCIA = 22
      OR EN_EXISTENCIA = 32
      OR EN_EXISTENCIA = 42;
```

Esta instrucción arrojará los mismos resultados que la instrucción SELECT del ejemplo anterior; sin embargo, como se puede ver, es mucho más incómoda.

Ahora veamos una instrucción SELECT que utiliza una subconsulta en el predicado IN. Supongamos que se quiere crear una consulta que arroje nombres de CD y sus artistas. Se requiere que los resultados de la consulta incluyan solamente CD que tengan más de 10 copias en existencia. Si nos referimos a la figura 9-4, se verá que la tabla ARTISTAS_DISCO_COMPACTO incluye los nombres de CD y sus artistas. Sin embargo, como también se puede ver, los valores EN_EXISTENCIA están almacenados en la tabla INVENTARIO_DISCO_COMPACTO, lo que significa que se necesitará hacer referencia a esa tabla para poder arrojar las filas correctas. Para hacer eso, se puede crear la siguiente instrucción SELECT:

```
SELECT TITULO, ARTISTA
FROM ARTISTAS_DISCO_COMPACTO
WHERE TITULO IN
      ( SELECT NOMBRE_CD
        FROM INVENTARIO_DISCO_COMPACTO
        WHERE EN_EXISTENCIA > 10 );
```

Si se ejecuta esta instrucción, se recibirán los siguientes resultados:

TITULO	ARTISTA
-----	-----
Famous Blue Raincoat	Jennifer Warnes
Blue	Joni Mitchell
Court and Spark	Joni Mitchell
Past Light	William Ackerman
Out of Africa	John Barry
Blues on the Bayou	B.B. King

Observe que sólo seis filas han sido arrojadas. Sólo hay seis CD enlistados en la tabla INVENTARIO_DISCO_COMPACTO que tienen un valor EN_EXISTENCIA mayor a 10.

Ahora veamos más de cerca la instrucción SELECT para obtener una mejor comprensión de cómo funciona el predicado IN. La cláusula WHERE contiene solamente un predicado. Éste comienza con el nombre de la columna (TITULO) cuyos valores se quieren verificar. La columna TITULO es seguida por la palabra clave IN. Luego la palabra clave es seguida por una sub-

consulta, que está encerrada en paréntesis. La subconsulta consiste de la siguiente instrucción SELECT:

```
SELECT NOMBRE_CD
      FROM INVENTARIO_DISCO_COMPACTO
      WHERE EN_EXISTENCIA > 10
```

Si se ejecutara sólo esta instrucción, se arrojarían los siguientes resultados:

```
NOMBRE_CD
-----
Famous Blue Raincoat
Blue
Court and Spark
Past Light
Out of Africa
Blues on the Bayou
```

Cada fila en los resultados de la consulta, que se derivan de la tabla INVENTARIO_DISCO_COMPACTO, contiene un valor EN_EXISTENCIA mayor a 10. Los valores en la columna TITULO de la tabla ARTISTAS_DISCO_COMPACTO son entonces comparados con estos resultados. Cualquier fila que contenga un valor TITULO que coincida con uno de los seis valores NOMBRE_CD (en los resultados de la subconsulta) está incluida en los resultados de la consulta de la instrucción SELECT principal.

NOTA

Cuando se incluye una subconsulta en un predicado IN, la cláusula SELECT de la subconsulta debe arrojar solamente una columna de datos. Si se especifica más de una columna en el conjunto de resultados o se especifica un asterisco, se recibirá un error.

Al igual que muchos otros predicados, el predicado IN permite especificar el inverso de una condición al utilizar la palabra clave NOT. Supongamos que se reescribe la instrucción SELECT del último ejemplo para incluir la palabra clave NOT en el predicado IN:

```
SELECT TITULO, ARTISTA
      FROM ARTISTAS_DISCO_COMPACTO
      WHERE TITULO NOT IN
        ( SELECT NOMBRE_CD
          FROM INVENTARIO_DISCO_COMPACTO
          WHERE EN_EXISTENCIA > 10 );
```

Los resultados de la consulta incluirán todas aquellas filas que no fueron arrojadas por la instrucción SELECT anterior y excluirá aquellas que sí fueron arrojadas, como se muestra en los siguientes resultados:

TITULO	ARTISTA
-----	-----
Kojiki	Kitaro
That Christmas Feeling	Bing Crosby
Patsy Cline: 12 Greatest Hits	Patsy Cline

After the Rain: The Soft Sounds of Erik Satie
 Leonard Cohen The Best of
 Fundamental
 Orlando

Pascal Roge
 Leonard Cohen
 Bonnie Raitt
 David Motion

Como se puede ver, el predicado IN es una herramienta muy flexible para comparar valores en una columna especificada con datos en otras tablas. Usted encontrará esto extremadamente útil cuando aprenda más acerca de las subconsultas y pueda crear predicados más complejos.

Utilizar el predicado EXISTS

A pesar de ser similar al predicado IN, el predicado EXISTS tiene un enfoque ligeramente diferente. Está dedicado únicamente a determinar si la subconsulta arroja alguna fila o no. Si ésta arroja una o más filas, el predicado se evalúa como verdadero; de otra manera, el predicado se evalúa como falso. El predicado consiste de la palabra clave EXISTS y una subconsulta. Para que la subconsulta sea un valor real (y subsecuentemente el predicado EXISTS por sí mismo), debe incluir un predicado que coincida con dos columnas en diferentes tablas. Por ejemplo, en la figura 9-4, la tabla INVENTARIO_DISCO_COMPACTO incluye la columna TITULO. Las dos columnas pueden hacerse coincidir juntas para asegurar que sólo las filas relevantes sean arrojadas por la subconsulta. Veamos un ejemplo para ayudar a aclarar este tema.

Supongamos que se quieren recuperar filas desde la tabla INVENTARIO_DISCO_COMPACTO para poder determinar cuántos CD de Joni Mitchell se tienen en existencia. Solamente se quieren desplegar los nombres de los CD y el número de ellos en existencia. No se quiere desplegar el nombre del artista, ni los CD de otros artistas. Para cumplir con esto, se puede utilizar la siguiente instrucción SELECT:

```
SELECT *
  FROM INVENTARIO_DISCO_COMPACTO
 WHERE EXISTS
    ( SELECT TITULO
      FROM ARTISTAS_DISCO_COMPACTO
     WHERE ARTISTA = 'Joni Mitchell'
       AND INVENTARIO_DISCO_COMPACTO.NOMBRE_CD =
         ARTISTAS_DISCO_COMPACTO.TITULO );
```

Si se ejecuta esta instrucción, se obtendrán los siguientes resultados de consulta:

NOMBRE_CD	EN_EXISTENCIA
Blue	42
Court and Spark	22

La mejor forma de comprender cómo funciona esta instrucción es observar cómo se evalúan las filas individuales. Como se aprenderá en el capítulo 12, las subconsultas como ésta son llamadas *subconsultas correlacionadas* debido a que la subconsulta se ejecuta para cada fila arrojada en la instrucción SELECT principal. Debido a que la cláusula WHERE de la subconsulta hace coincidir el valor NOMBRE_CD con el valor TITULO, el valor TITULO en la fila que se está eva-

luando (en la subconsulta) debe coincidir con el valor NOMBRE_CD para que esa fila pueda ser arrojada. Por ejemplo, la primera fila en la tabla INVENTARIO_DISCO_COMPACTO contiene un valor NOMBRE_CD de Famous Blue Raincoat. Cuando esta fila es comparada con el predicado EXISTS, el valor Famous Blue Raincoat se hace coincidir con el valor Famous Blue Raincoat de la columna TITULO en la tabla ARTISTAS_DISCO_COMPACTO. Además, el valor Joni Mitchell se hace coincidir con el valor ARTISTA para la fila Famous Blue Raincoat. Debido a que el valor ARTISTA es Jennifer Warnes, y no Joni Mitchell, la condición de búsqueda especificada en la cláusula WHERE de la subconsulta se evalúa como falsa; por lo tanto, no se arroja ninguna fila de subconsulta para la fila Famous Blue Raincoat. Como resultado, la cláusula WHERE en la instrucción SELECT principal se evalúa como falsa para la fila Famous Blue Raincoat de la tabla INVENTARIO_DISCO_COMPACTO, y la fila no se incluye en los resultados de la consulta.

Este proceso se repite para cada fila en la tabla INVENTARIO_DISCO_COMPACTO. Si la cláusula WHERE en la subconsulta se evalúa como verdadera, entonces el predicado EXISTS se evalúa como verdadero, lo que significa que la cláusula WHERE en la instrucción SELECT principal se evalúa como verdadera. En el caso de nuestro último ejemplo de la instrucción SELECT, solamente dos filas coinciden con este criterio.

NOTA

No tiene importancia cuáles columnas o cuántas columnas se especifican en la cláusula SELECT de la subconsulta en un predicado EXISTS. Este tipo de predicado se concentra solamente en si las filas serán arrojadas, en lugar de en el contenido de esas filas. Se puede especificar cualquier nombre de columna o simplemente un asterisco.

El predicado EXISTS, como se puede esperar, permite utilizar el inverso de la condición del predicado utilizando la palabra clave NOT:

```
SELECT *
FROM INVENTARIO_DISCO_COMPACTO
WHERE NOT EXISTS
    ( SELECT TITULO
      FROM ARTISTAS_DISCO_COMPACTO
      WHERE ARTISTA = 'Joni Mitchell'
        AND INVENTARIO_DISCO_COMPACTO.NOMBRE_CD =
          ARTISTAS_DISCO_COMPACTO.TITULO );
```

En este caso, todos los CD, *excepto* los CD de Joni Mitchell, se incluyen en los resultados de la consulta. Eso significa que si la cláusula WHERE de la subconsulta se evalúa como verdadera (lo que significa que la subconsulta arroja una fila), el predicado en sí mismo se evalúa como falso y no se arroja ninguna fila. Por otro lado, si la subconsulta no arroja una fila, el predicado se evalúa como verdadero y la fila se arroja en los resultados de la consulta de la instrucción SELECT principal.

Pregunta al experto

P: Se han proporcionado ejemplos que muestran que existe a menudo más de una forma para lograr el mismo resultado. ¿Cómo puede uno saber cuál opción seleccionar cuando se está escribiendo una instrucción SQL?

R: Encontrará que, mientras más aprende acerca de programación SQL y logra un mejor entendimiento de las facetas de cada instrucción, encontrará a menudo más de una forma de lograr los mismos resultados. En estos casos, su elección de los métodos dependerá a menudo de cuál instrucción sea la más sencilla de escribir o cuál realiza mejor el trabajo en una implementación SQL en particular. Al mismo tiempo que crece su comprensión de SQL, también lo hará su habilidad para elegir el método que sea mejor para cada situación. En muchos casos, la diferencia entre un método y otro no será muy grande, y su elección dependerá simplemente de sus preferencias personales. Sin embargo, también puede toparse con situaciones en las cuales la implementación de SQL en la que se está trabajando no soporte todos los métodos proporcionados en el estándar SQL. Por lo tanto, se debe seleccionar el método que pueda ser implementado en ese ambiente en particular. Cualquiera que sea el método que se utilice en cualquier ambiente dado, lo mejor por ahora es tener una base tan completa como sea posible acerca de los conceptos básicos de SQL. De esa forma se estará mejor preparado para las diferentes situaciones y mejor equipado para moverse de una implementación a otra. Adicionalmente, deberá aprender acerca de los temas de rendimiento relacionados a la implementación con la que se está trabajando. Deberá considerar los temas de rendimiento cuando se tome una decisión acerca de cuáles instrucciones SQL se deben utilizar.

P: Cuando se proporcionaron ejemplos del predicado EXISTS, las subconsultas siempre hacían coincidir columnas dentro de la cláusula WHERE de la subconsulta. ¿Esto es necesario?

R: Es posible, si se desea, crear un predicado EXISTS que no haga coincidir columnas en la subconsulta, como la siguiente instrucción:

```
SELECT TITULO, ARTISTA
FROM ARTISTAS_DISCO_COMPACTO
WHERE EXISTS
    ( SELECT NOMBRE_CD
      FROM INVENTARIO_DISCO_COMPACTO
      WHERE EN_EXISTENCIA > 10 );
```

En este caso, la subconsulta simplemente revisa si es que existe en la tabla INVENTARIO_DISCO_COMPACTO algún valor EN_EXISTENCIA mayor a 10. Si esa fila o filas existen, el predicado se evaluó como verdadero, lo que significa que la cláusula WHERE en la instrucción SELECT principal se evalúa como verdadera. Como resultado, se arrojan todas las filas en la tabla ARTISTAS_DISCO_COMPACTO. Utilizar este tipo de subconsulta generalmente no es muy útil debido a que ofrece pocas ventajas sobre una instrucción SELECT simple. Cuando se utiliza EXISTS, hacer coincidir columnas de diferentes tablas dentro de la subconsulta es esencial para proporcionar un filtrado significativo para la instrucción SELECT principal.

Determinar la cantidad de predicados de comparación

SQL incluye otro tipo de predicado denominado *predicado de comparación cuantificado*, que es un tipo de predicado utilizado en conjunción con un operador de comparación para determinar si *alguno* o *todos* los valores arrojados coinciden con los requerimientos de la búsqueda. SQL soporta tres predicados de comparación cuantificados: SOME, ANY y ALL. Se hace referencia a los predicados SOME y ANY como *cuantificadores existenciales*, y se encargan de comprobar si cualquier valor arrojado corresponde a los requisitos de la búsqueda. Estos dos predicados son idénticos en significado y pueden ser utilizados intercambiabilmente. El predicado ALL es llamado *cuantificador universal* y se ocupa de comprobar si *todos* los valores arrojados corresponden a los requisitos de la búsqueda. Ahora demos un vistazo más cercano a cada uno.

Utilizar los predicados SOME y ANY

Como se mencionó, los predicados SOME y ANY arrojan resultados idénticos. Para cada fila, los predicados comparan el valor en una columna especificada con los resultados de una subconsulta. Si la comparación se evalúa como verdadera para *cualquiera* de los resultados, la condición se toma como satisfactoria y se arroja esa fila. Para crear uno de estos predicados, se debe especificar el nombre de la columna que contiene los valores que se quieren comparar, el operador de comparación (véase la sección “Comparar datos SQL”), la palabra clave SOME o ANY, y la subconsulta. A pesar de que se pueden utilizar cualquiera de las palabras, particularmente yo prefiero utilizar ANY porque me parece más intuitiva, pero siéntase libre de utilizar la que guste.

Ahora veamos un ejemplo para ilustrar mejor cómo funcionan estos predicados. El ejemplo está basado en la tabla MENUDEO_CD y en la tabla REBAJA_CD, que se muestran en la figura 9-5.

En este ejemplo se busca consultar datos de la tabla REBAJA_CD. Se quiere arrojar sólo aquellas filas que tengan un valor MENUDEO menor que algunos de los valores MENUDEO en la tabla MENUDEO_CD.

MENUDEO_CD			REBAJA_CD	
NOMBRE_CD: VARCHAR(60)	MENUDEO: NUMERIC(5,2)	EN_EXISTENCIA: INT	TITULO: VARCHAR(60)	VENTA: NUMERIC(5,2)
Famous Blue Raincoat	16.99	5	Famous Blue Raincoat	14.99
Blue	14.99	10	Blue	12.99
Court and Spark	14.99	12	Court and Spark	14.99
Past Light	15.99	11	Past Light	14.99
Kojiki	15.99	4	Kojiki	13.99
That Christmas Feeling	10.99	8	That Christmas Feeling	10.99
Patsy Cline: 12 Greatest Hits	16.99	14	Patsy Cline: 12 Greatest Hits	16.99

Figura 9-5 Utilizar predicados de comparación cuantificados en las tablas MENUDEO_CD y REBAJA_CD.

Los valores RETAIL deberán ser de filas que tengan un valor EN_EXISTENCIA mayor a 9. En otras palabras, la consulta deberá arrojar solamente aquellos CD cuyo precio rebajado (REBAJA) sea menor que *cualquier* precio de lista (MENUDEO) en aquellos CD que haya una existencia mayor a nueve. Para cumplir este requisito se utilizará la siguiente instrucción SELECT:

```
SELECT TITULO, REBAJA
FROM REBAJA_CD
WHERE REBAJA < ANY
  ( SELECT MENUDEO
    FROM MENUDEO_CD
    WHERE EN_EXISTENCIA > 9 );
```

Si se prefiere, se puede utilizar la palabra clave SOME en lugar de la palabra clave ANY. Los resultados de la consulta serían los mismos, como se muestra en los siguientes:

TITULO	REBAJA
-----	-----
Famous Blue Raincoat	14.99
Blue	12.99
Court and Spark	14.99
Past Light	14.99
Kojiki	13.99
That Christmas Feeling	10.99

Ahora analicemos la instrucción SELECT más de cerca. El predicado ANY contiene la siguiente subconsulta:

```
SELECT RETAIL
FROM CD_RETAIL
WHERE EN_EXISTENCIA > 9
```

Si se ejecutara tal subconsulta por sí sola, se recibirían los siguientes resultados:

```
MENUDEO
-----
14.99
14.99
15.99
16.99
```

El valor REBAJA en cada fila en la tabla REBAJA_CD es por lo tanto comparado a los resultados de la subconsulta. Por ejemplo, la fila Past Light tiene un valor REBAJA de 14.99. Este valor es comparado a los resultados de la subconsulta para verificar si 14.99 es menor que *cualquier* valor. Debido a que éste es menor que 15.99 y 16.99, el predicado se evalúa como verdadero y la fila es arrojada. La única fila que no se evalúa como verdadera es la fila Patsy Cline: 12 Greatest Hits debido a que el valor REBAJA es 16.99, y éste no es menor a ninguno de los valores arrojados por los resultados de la consulta.

Puede utilizar cualquiera de los seis operadores de comparación en un predicado ANY o SOME. Por ejemplo, si se hubiera utilizado el operador Mayor que, solamente la fila Patsy Cline:

12 Greatest Hits habría sido arrojado debido a que hubiera sido la única fila con un valor REBAJA mayor que cualquier fila en los resultados de la subconsulta.

NOTA

Los predicados de comparación cuantificados no soportan una condición inversa como lo hacen otros predicados. En otras palabras, no se puede agregar la palabra NOT antes de ANY o SOME. Sin embargo, se pueden lograr los mismos resultados utilizando el operador desigual a (<>).

Utilizar el predicado ALL

El predicado ALL funciona muy parecido a los predicados SOME y ANY porque también compara valores de columna con los resultados de la subconsulta. Sin embargo, en lugar de que los valores de columna tengan que evaluarse como verdaderos para *cualquiera* de los valores resultantes, los valores de columna deben evaluarse como verdaderos para *todos* los valores resultantes; si no es así, la fila no será arrojada.

Regresemos al ejemplo anterior, solamente que esta vez se sustituirá la palabra clave ANY por la palabra clave ALL. La nueva instrucción SELECT lucirá de la siguiente manera:

```
SELECT TITULO, REBAJA
FROM REBAJA_CD
WHERE REBAJA < ALL
      ( SELECT MENUDEO
        FROM MENUDEO_CD
        WHERE EN_EXISTENCIA > 9 );
```

Si se ejecuta esta instrucción, se encontrará que los resultados de la consulta son bastante diferentes a lo que eran en el ejemplo anterior:

TITULO	REBAJA
Blue	12.99
Kojiki	13.99
That Christmas Feeling	10.99

Esta vez solamente se arrojaron tres filas debido a que son las únicas que cumplen la condición del predicado WHERE.

Si se da un vistazo más cercano a la instrucción, se encontrará que la subconsulta arroja los mismos valores que en los ejemplos anteriores. Sin embargo, el valor SALE para cada fila en la tabla REBAJA_CD debe ahora ser menor que todos los valores en los resultados de la subconsulta. Por ejemplo, la fila Kojiki contiene un valor REBAJA de 13.99. Los resultados de la subconsulta incluyen los valores 14.99, 15.99 y 16.99. El valor 13.99 es menor que todos los tres valores resultantes de la subconsulta, lo cual significa que el predicado se evalúa como verdadero y que esa fila se incluye en los resultados de la consulta. Por otro lado, la fila Past Light contiene un valor REBAJA de 14.99, el cual no es menor que el valor de la subconsulta 14.99, por lo que esa fila no se incluye en los resultados de la consulta.

Pregunta al experto

P: En los análisis acerca de los predicados de comparación cuantificados, se incluyeron ejemplos sobre cómo utilizar estos predicados; sin embargo, los ejemplos incluían solamente un predicado en la cláusula WHERE. ¿Es posible utilizar múltiples predicados cuando se utiliza un predicado de comparación cuantificado?

R: Sí. Es posible utilizar múltiples predicados. Al igual que con cualquier otro tipo de predicado, simplemente deben conectarse los predicados utilizando la palabra clave AND o la palabra clave OR. Pero debe asegurarse de que la lógica que se está utilizando no solamente tenga sentido en términos de los datos que serán arrojados, sino también que tenga sentido para que pueda comprenderse la instrucción por sí misma. Como resultado, la mejor forma de tratar con este tipo de situaciones es destacando cada predicado entre paréntesis y luego conectando estas expresiones parentéticas con AND u OR. Por ejemplo, supongamos que se quiere utilizar el ejemplo en la sección “Utilizar los predicados SOME y ANY” y agregarle un predicado LIKE. (El ejemplo está basado en la figura 9-5.) Se puede crear una instrucción SELECT similar a la siguiente:

```
SELECT TITULO, SALE
FROM REBAJA_CD
WHERE ( REBAJA < ANY ( SELECT MENUDEO
                        FROM REBAJA_CD
                        WHERE EN_EXISTENCIA > 9 ) )
AND ( TITULO LIKE ('%Blue%') );
```

Observe que cada predicado ha sido encerrado en un conjunto de paréntesis y que éstos están unidos por AND. Si se ejecuta esta instrucción, los resultados de la consulta acatarán la condición del predicado ANY y del predicado LIKE, la cual especifica que el valor TITLE incluya la palabra Blue. Si se prefiere, se pueden escribir estas instrucciones sin encerrar los predicados entre paréntesis, pero entonces las instrucciones pueden empezar a confundirse, y en estructuras más complejas pueden también empezar a producir resultados inesperados.

Al igual que con los predicados ANY y SOME, es posible utilizar cualquiera de los seis operadores de comparación con el predicado ALL. Adicionalmente, se puede crear cualquier tipo de subconsulta, siempre y cuando encaje lógicamente con la instrucción SELECT principal. El punto que debe recordarse es que el valor de la columna debe ser verdadero para todos los resultados de la subconsulta, y no solamente para algunos de ellos.

Pruebe esto 9-2 Utilizar subconsultas en predicados

Este ejercicio básicamente comienza donde se quedó el anterior. Una vez más, trabajará con predicados, sólo que esta vez será con aquellos que utilizan subconsultas. Son estos los predicados que se analizaron desde el último ejercicio. Incluyen a los predicados IN, EXISTS, ANY y ALL. Al igual que con el ejercicio anterior, se aplicarán estos predicados a las tablas que se crearon en la base de datos INVENTARIO. Puede descargar el archivo Try_This_09.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para sus RDBMS y conéctese con la base de datos INVENTARIO.
2. En su primera instrucción utilizará un predicado IN para consultar datos de la tabla DISCOS_COMPACTOS. Se quiere ver la información de CD y de inventario para los CD publicados por la compañía Decca Record Company. Para averiguar cuáles son estos CD, se debe crear una subconsulta que consulte los datos de la tabla DISQUERAS_CD. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE ID_DISQUERA IN
      ( SELECT ID_DISQUERA
        FROM DISQUERAS_CD
        WHERE NOMBRE_COMPAÑIA = 'Decca Record Company' );
```

Los resultados de la consulta deberán incluir solamente dos filas. Ambas filas tendrán un valor ID_DISQUERA de 833, que es el valor arrojado por la subconsulta.

3. Ahora se intentará una instrucción SELECT similar a la del paso 2, sólo que esta vez se utilizará un predicado EXISTS para arrojar los datos. Adicionalmente, tendrá que agregar un predicado a la cláusula WHERE de la subconsulta que hará coincidir el valor ID_DISQUERA en la tabla DISCOS_COMPACTOS con el valor ID_DISQUERA en la tabla DISQUERAS_CD. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_COMPACTOS
WHERE EXISTS
      ( SELECT ID_DISQUERA FROM DISQUERAS_CD
        WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA
        AND ID_DISQUERA > 830 );
```

Observe que uno de los predicados en la cláusula WHERE de la subconsulta utiliza un operador de comparación para buscar los valores ID_DISQUERA mayores a 830. Si se busca en la tabla DISQUERAS_CD, se encontrará que hay seis filas que contienen valores ID_DISQUE-

RA mayores a 830. Entonces, si se fuera a hacer coincidir estos seis valores con los valores ID_DISQUERA en la tabla DISCOS_COMPACTOS, se encontraría que 11 filas se evaluarían como verdaderas. Éstas serán las 11 filas arrojadas por la instrucción SELECT.

4. En esta instrucción se utilizará un predicado ANY para comparar los valores ID_DISQUERA en la tabla DISQUERAS_CD con los valores ID_DISQUERA en la tabla DISCOS_COMPACTOS que están incluidos en filas con un valor EN_EXISTENCIA mayor a 20. Los valores ID_DISQUERA en la tabla DISQUERAS_CD pueden coincidir con cualquier valor en los resultados de la subconsulta. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, NOMBRE_COMPANIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ANY
    ( SELECT ID_DISQUERA
      FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 20 );
```

La consulta deberá arrojar solamente cinco filas.

5. Ahora trate de crear la misma instrucción SELECT del paso 4, sólo que esta vez utilizando un predicado ALL en lugar de un predicado ANY. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, NOMBRE_COMPANIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ALL
    ( SELECT ID_DISQUERA
      FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 20 );
```

Encontrará que con esta consulta no se arroja ninguna fila. Esto se debe a que la subconsulta arroja ocho filas con cinco diferentes valores. El valor ID_DISQUERA para cada fila en la tabla DISQUERAS_CD no puede hacer coincidir todos los valores, solamente uno o algunos de ellos. La única forma en que se arrojaría alguna fila en este caso sería si la subconsulta arroja solamente una fila o múltiples filas, todas con el mismo valor.

6. Ahora intente modificar la instrucción SELECT cambiando el predicado de comparación en la cláusula WHERE de la subconsulta a mayor que 40. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, NOMBRE_COMPANIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ALL
    ( SELECT ID_DISQUERA
      FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 40 );
```

Los resultados de la consulta ahora arrojarán una fila. Esto se debe a que la subconsulta arroja sólo una fila que cumple la condición del predicado ALL.

7. Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio se utilizaron los predicados IN, EXISTS, ANY y ALL para consultar datos de la base de datos INVENTARIO. También pudo haberse utilizado el predicado SOME en lugar del predicado ANY. Combinados con los pasos del ejercicio anterior, las instrucciones aquí deberán haberle permitido probar con una gran variedad de predicados. Según vaya aprendiendo más acerca de las subconsultas, será capaz de crear predicados incluso más elaborados, unos que pueda utilizar no solamente en las instrucciones SELECT, sino también en las instrucciones UPDATE y DELETE. Entretanto, se le sugiere experimentar con diferentes tipos de instrucciones SELECT e intentar diferentes predicados dentro de esas instrucciones para ver exactamente qué tipos de resultados de consulta pueden arrojar.

Autoexamen Capítulo 9

1. ¿En cuál cláusula de la instrucción SELECT se incluyen predicados?
2. ¿Cuál símbolo de operador de comparación deberá utilizarse para expresar una condición desigual?
 - A <=
 - B >=
 - C <>
 - D =<
3. ¿Cuáles palabras clave pueden utilizarse para combinar predicados en una cláusula WHERE?
4. Se quiere consultar una tabla que incluye la columna PRECIO. Es necesario asegurarse que todas las filas arrojadas tengan un valor PRECIO de 13.99. ¿Cuál predicado deberá utilizarse?

5. Se crea la siguiente instrucción SQL:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO >= 14
AND PRECIO_MENUDEO <= 16;
```

¿Qué predicado puede utilizarse en lugar de los dos predicados mostrados en esta instrucción?

6. ¿Qué palabra clave puede agregarse a un predicado BETWEEN para encontrar el inverso de la condición especificada por el predicado?
7. ¿Cuándo se utiliza un valor nulo en una columna?
8. Se quiere consultar una tabla para determinar cuáles valores son nulos. ¿Qué tipo de predicado deberá utilizarse?

9. Se está creando una instrucción SELECT que consulta la tabla BIO_ARTISTAS. Se quieren arrojar todas las columnas en la tabla, pero se quieren arrojar sólo aquellas columnas que no contengan valores nulos en la columna LUGAR_DE_NACIMIENTO. ¿Cuál instrucción SELECT deberá utilizarse?
10. Se está consultando la tabla INVENTARIO_CD. Se quiere ver todas las columnas, pero se requiere ver sólo aquellas columnas que contengan la palabra Christmas en el nombre del CD. Los nombres están almacenados en la columna TITULO_CD. ¿Cuál instrucción SELECT deberá utilizarse?
11. ¿Cuál es la diferencia entre un signo de porcentaje y un guión bajo cuando se usan en un predicado LIKE?
12. ¿Cuáles dos tipos de fuentes de datos pueden utilizarse en un predicado IN?
13. ¿Cuál tipo de predicado se ocupa solamente de determinar si una subconsulta arroja cualquier fila o no?
14. ¿Cuáles nombres de columna deben ser especificados en un predicado EXISTS?
15. Se está creando una instrucción SELECT que incluye un predicado en la cláusula WHERE. Se quiere utilizar un operador de comparación para comparar los valores en una de las columnas con los resultados de una subconsulta. Se quiere que el predicado se evalúe como verdadero para cualquiera de los resultados de la subconsulta. ¿Qué tipo de predicado deberá utilizarse?
 - A EXISTS
 - B ANY
 - C ALL
 - D IN
16. ¿Cuál es la diferencia entre un predicado SOME y un predicado ANY?
17. ¿Cómo difiere el predicado ALL del predicado SOME?

Capítulo 10

Trabajar con funciones
y expresiones de valor

Habilidades y conceptos clave

- Utilizar funciones Set
 - Utilizar funciones de valor
 - Utilizar expresiones de valor
 - Utilizar valores especiales
-

En partes anteriores de este libro se han presentado brevemente varias funciones y expresiones relacionadas con valores. Estos valores y expresiones se utilizan en ejemplos y ejercicios en varios capítulos para demostrar diferentes componentes de SQL. En este capítulo se dará un vistazo más cercano a muchos de estos valores y expresiones, centrándose en aquellos que más probablemente pueda utilizar un programador principiante de SQL. Deberá conservar en mente, sin embargo, que este capítulo cubre solamente una porción de los muchos tipos de funciones y expresiones soportados por SQL. Adicionalmente, las implementaciones de SQL pueden variar respecto a cuáles funciones y expresiones SQL pueden soportar, cómo son implementados esos valores y expresiones, y cuáles funciones y expresiones estándar y no estándar incluyen en sus productos. Asegúrese de revisar la documentación del producto para determinar cuáles funcionalidades son soportadas. En general, se incluyen en este capítulo aquellas funciones y expresiones más comúnmente soportadas por las implementaciones SQL.

Utilizar funciones Set

En el capítulo 3 se introdujo el concepto de función. Como se puede recordar, una función es una operación nombrada que realiza tareas definidas que normalmente no se pueden realizar utilizando solamente instrucciones SQL. Es un tipo de rutina que toma parámetros de entrada, los cuales están encerrados en paréntesis, y arroja valores basados en esos paréntesis. Una propiedad importante de las funciones es que cada ejecución de una función arroja exactamente un valor de datos, y ésta es la razón por la que las funciones pueden ser utilizadas en lugar de los nombres de columna de una tabla en la lista SELECT de una consulta (la función arroja un valor único para cada fila procesada por la consulta). Ya hemos visto ejemplos de funciones como SUM y AVG. Ambas funciones son conocidas como funciones set. Una *función set*, a veces nombrada función *agregada*, procesa o calcula datos y arroja los valores apropiados. Las funciones set requieren que los datos estén agrupados de cierta manera, como sería el caso si la cláusula GROUP BY fuera utilizada en una instrucción SELECT. Si las filas no están agrupadas explícitamente de esa manera, la tabla completa es tratada como un grupo.

En esta sección se analizan cinco funciones set: COUNT, MAX, MIN, SUM y AVG. Todas estas funciones son comúnmente soportadas en las implementaciones SQL. Para todas las funciones set, se proporcionan ejemplos acerca de cómo se utilizarían en la cláusula SELECT de una instrucción SELECT. Los ejemplos están basados en la tabla mostrada en la figura 10-1.

NOMBRE_ARTISTA: VARCHAR(60)	NOMBRE_CD: VARCHAR(60)	VENDIDOS: INT
Jennifer Warnes	Famous Blue Raincoat	23
Joni Mitchell	Blue	45
Joni Mitchell	Court and Spark	34
William Ackerman	Past Light	12
Bing Crosby	That Christmas Feeling	34
Patsy Cline	Patsy Cline: 12 Greatest Hits	54
John Barry	Out of Africa	23
Leonard Cohen	Leonard Cohen The Best of	20
Bonnie Raitt	Fundamental	29
B.B. King	Blues on the Bayou	18

Figura 10-1 Utilizar las funciones set en la tabla CDS_ARTISTA.

Utilizar la función COUNT

La primera función set que veremos será la función COUNT. Como su nombre sugiere, la función COUNT cuenta el número de filas en una tabla o el número de valores en una columna, según se especifique en la instrucción SELECT. Cuando se utiliza la función COUNT, se debe especificar un nombre de columna para contar el número de valores que no sean nulos en una columna, o un asterisco para contar todas las filas en una tabla independientemente de los valores nulos. Por ejemplo, si se quiere saber el número total de filas en la tabla CDS_ARTISTA, se puede utilizar la siguiente instrucción SELECT:

```
SELECT COUNT(*) AS FILAS_TOTALES
FROM CDS_ARTISTA;
```

En esta instrucción, la función COUNT es utilizada con un asterisco (en paréntesis) para contar cada fila en la tabla CDS_ARTISTA y arrojar la cuenta total. El valor arrojado se enlista en la columna FILAS_TOTALES, que es el nombre dado a la columna arrojada en los resultados de la consulta, como se muestra en los siguientes resultados:

```
FILAS_TOTALES
-----
10
```

Como se puede ver, los resultados de la consulta incluyen solamente un valor (una fila con una columna), como podría esperarse de una función set utilizada sin ningún agrupamiento de fila. El valor 10 indica que la tabla CDS_ARTISTA contiene 10 filas.

Al igual que con cualquier otra clase de instrucción `SELECT`, se pueden cualificar los resultados de la consulta agregando a la instrucción las cláusulas necesarias. Por ejemplo, supongamos que se quiere averiguar cuántas filas incluye un valor `VENDIDOS` mayor a 20. Se puede modificar la instrucción `SELECT` para incluir una cláusula `WHERE`:

```
SELECT COUNT(*) AS FILAS_TOTALES
FROM CDS_ARTISTA
WHERE VENDIDOS > 20;
```

El valor arrojado ahora será 7 en lugar de 10, debido a que solamente siete filas cumplen con la condición de búsqueda especificada en la cláusula `WHERE`.

Se puede encontrar con que, en lugar de consultar el número de filas en una tabla, se quiera conocer el número de valores en una columna dada (sin incluir los valores nulos). En este caso, se especificaría el nombre de la columna en lugar del asterisco (*). Por ejemplo, supongamos que se modifica la instrucción `SELECT` mostrada en el último ejemplo para contar valores en la columna `NOMBRE_ARTISTA`:

```
SELECT COUNT(NOMBRE_ARTISTA) AS TOTAL_DE_ARTISTAS
FROM CDS_ARTISTA
WHERE VENDIDOS > 20;
```

Cuando se ejecuta esta consulta, el valor arrojado una vez más es 7. Esto significa que siete valores `NOMBRE_ARTISTA` tienen un valor `VENDIDOS` mayor a 20. Sin embargo, esta instrucción no contabiliza aquellos valores `NOMBRE_ARTISTA` que pudieran estar duplicados. Si se quiere llegar a un conteo que tome en consideración los valores duplicados, se puede agregar la palabra clave `DISTINCT` a la función `COUNT`:

```
SELECT COUNT(DISTINCT NOMBRE_ARTISTA) AS TOTAL_DE_ARTISTAS
FROM CDS_ARTISTA
WHERE VENDIDOS > 20;
```

Esta vez el valor arrojado es de 6 en lugar de 7. Esto se debe a que la columna `NOMBRE_ARTISTA` incluye dos instancias del valor Joni Mitchell. La columna contiene solamente seis valores únicos que cumplen la condición establecida en el criterio de búsqueda.

NOTA

Tenga en cuenta que la instrucción `SELECT` es procesada en un orden específico: primero la cláusula `FROM`, luego la cláusula `WHERE`, y después la cláusula `SELECT`. Como resultado, la función `COUNT` aplica solamente a las filas que cumplen la condición de búsqueda definida en la cláusula `WHERE`. Las filas que no estén incluidas en los resultados de la cláusula `WHERE` no tendrán relevancia en la función `COUNT`. Para mayor información acerca de la instrucción `SELECT`, véase el capítulo 7.

Como ya se mencionó, si la columna especificada en la función aún contiene valores nulos, esos valores no estarán incluidos en la cuenta. Por ejemplo, si se agregara una fila a la tabla `CDS_ARTISTA` con un valor `NOMBRE_ARTISTA` nulo y un valor `VENDIDOS` mayor a 20, la instrucción `SELECT` mostrada en el ejemplo anterior aún arrojaría un valor de 6 debido a que el valor nulo no sería contado. Sin embargo, si se utilizara un asterisco en lugar de un nombre de columna en la función `COUNT`, todas las filas serían contadas, incluso si algunas tuvieran valores nulos.

Utilizar las funciones MAX y MIN

Las funciones MAX y MIN son tan similares que resulta ideal revisarlas juntas. La función MAX arroja el valor más alto para la columna especificada, y la función MIN arroja el valor más bajo. Ambas funciones requieren que se especifique un nombre de columna. Por ejemplo, supongamos que se quiere arrojar el valor más alto de la columna VENDIDOS en la tabla CDS_ARTISTA. La instrucción SELECT quedaría de la manera siguiente:

```
SELECT MAX(VENDIDOS) AS MAX_VENDIDOS
FROM CDS_ARTISTA;
```

Cuando se ejecuta esta instrucción, los resultados de la consulta incluirán solamente un valor (una fila y una columna), como se muestra en los siguientes resultados:

```
MAX_VENDIDOS
-----
54
```

Este resultado, por sí mismo, no resulta de mucha ayuda. Sería mejor si los resultados de la consulta también incluyeran el nombre del artista y el CD. Sin embargo, SQL no soporta una instrucción SELECT como la siguiente:

```
SELECT NOMBRE_ARTISTA, NOMBRE_CD, MAX(VENDIDOS)
FROM CDS_ARTISTA;
```

Debido a que las funciones set tratan los datos como grupos, no es posible especificar el nombre del artista y el nombre del CD sin agrupar los datos de alguna forma. De la manera en que está ahora, la función MAX trata la tabla entera como un grupo; sin embargo, ni los valores de NOMBRE_ARTISTA ni los de NOMBRE_CD están agrupados de ninguna forma, por lo que la cláusula SELECT se vuelve ilógica.

Cada vez que se incluya una función set en una instrucción SQL, cada argumento en la lista SELECT deberá ser una función set o estar incluido en un grupo (utilizando la cláusula GROUP BY descrita posteriormente en este tema). Una forma relativa a esto es utilizar una subconsulta en la cláusula WHERE para arrojar el valor máximo y luego arrojar la información necesaria pasada en ese valor, como se muestra en el siguiente ejemplo:

```
SELECT NOMBRE_ARTISTA, NOMBRE_CD, VENDIDOS
FROM CDS_ARTISTA
WHERE VENDIDOS = ( SELECT MAX(VENDIDOS)
                   FROM CDS_ARTISTA );
```

La subconsulta encuentra el valor máximo (54) y utiliza ese valor como una condición en la cláusula WHERE. El valor VENDIDOS debe ser igual a 54 mientras ése sea el valor más alto de VENDIDOS en la tabla. Una vez que se define la condición de búsqueda necesaria en la cláusula WHERE, se pueden utilizar esos resultados para arrojar la información que se necesita. Si se ejecuta esta instrucción, se arrojará solamente una fila:

NOMBRE_ARTISTA	NOMBRE_CD	VENDIDOS
-----	-----	-----
Patsy Cline:	Patsy Cline 12 Greatest Hits	54

Como se puede ver, ahora se tiene toda la información necesaria para determinar cuál artista y cuáles CD han tenido las ventas más altas.

Como se dijo anteriormente, las funciones MAX y MIN son muy similares. Si se reemplaza MAX por MIN en el ejemplo anterior, los resultados de la consulta serán los siguientes:

NOMBRE_ARTISTA	NOMBRE_CD	VENDIDOS
William Ackerman	Past Light	12

Se arroja la fila Past Light debido a que es la fila con el valor VENDIDOS más bajo.

Las funciones MAX y MIN no están limitadas a datos numéricos. También pueden ser utilizadas para comparar cadenas de caracteres. Por ejemplo, supongamos que se quiere saber cuál artista aparecería primero alfabéticamente. La siguiente instrucción arrojará B.B. King:

```
SELECT MIN(NOMBRE_ARTISTA) AS LOW_NOMBRE
FROM CDS_ARTISTA;
```

Si se utiliza la función MAX, la instrucción arrojará William Ackerman.

NOTA

Es muy probable que las tablas en su base de datos agrupen separadamente los nombres de los apellidos debido a que ése es un diseño más flexible. Aquí se han incluido ambos nombres a una sola columna para proporcionarle ejemplos simples sobre cómo funcionan las diferentes instrucciones. Si los nombres estuvieran separados en dos columnas, la función MIN o MAX necesitaría ser utilizada en la columna apropiada.

Ahora volvamos un poco a la idea de la agrupación de datos. Como se mencionó, una función set trata a una tabla como un grupo si ningún agrupamiento ha sido implementado. Sin embargo, se puede utilizar fácilmente una cláusula GROUP BY para agrupar los datos. Supongamos que se quiere conocer la cantidad máxima vendida por cada artista. Se pueden agrupar los datos basados en los valores NOMBRE_ARTISTA:

```
SELECT NOMBRE_ARTISTA, MAX(VENDIDOS) AS MAX_VENDIDOS
FROM CDS_ARTISTA
WHERE VENDIDOS > 30
GROUP BY NOMBRE_ARTISTA;
```

La cláusula WHERE arroja solamente aquellas filas con un valor VENDIDOS mayor a 30. Luego las filas son agrupadas de acuerdo con los valores NOMBRE_ARTISTA. Una vez que han sido agrupadas, la cantidad máxima se arroja para cada artista, como se muestra en los siguientes resultados de la consulta:

NOMBRE_ARTISTA	MAX_VENDIDOS
Bing Crosby	34
Joni Mitchell	45
Patsy Cline	54

La cláusula GROUP BY crea tres grupos, uno para cada artista que cumple con la condición de la búsqueda definida en la cláusula WHERE. De estos tres grupos, solamente uno consta de valores duplicados: Joni Mitchell. Debido a que hay dos filas Joni Mitchell en la tabla CDS_ARTISTA, también hay dos valores VENDIDOS: 45 y 34. Como se puede ver, el valor más alto es 45, que es el que está incluido en los resultados de la consulta para el grupo Joni Mitchell. Si la función MIN hubiera sido utilizada en la instrucción SELECT, sería el valor 34 el que hubiera sido arrojado. Para los otros dos grupos de artistas, debido a que sólo existe un valor para cada uno, es ese mismo valor el que es arrojado sin importar si se utiliza la función MAX o la función MIN.

Utilizar la función SUM

A diferencia de las funciones MIN y MAX, que seleccionan los valores más alto y más bajo de una columna, la función SUM agrupa valores de columna. Esto es particularmente útil cuando se necesita encontrar los totales para datos agrupados (a pesar de que la función SUM, al igual que cualquier otra función set, trata a la tabla entera como un grupo único si ningún dato ha sido explícitamente agrupado).

Para entender mejor la función SUM, tomemos el último ejemplo que vimos y hagámosle algunas modificaciones:

```
SELECT NOMBRE_ARTISTA, SUM(VENDIDOS) AS TOTAL_VENDIDOS
FROM CDS_ARTISTA
WHERE VENDIDOS > 30
GROUP BY NOMBRE_ARTISTA;
```

Como se vio anteriormente, la cláusula WHERE arroja solamente aquellas filas con un valor VENDIDOS mayor a 30. Luego estas filas son agrupadas de acuerdo con los valores de NOMBRE_ARTISTA. Una vez agrupadas, las cantidades totales de cada grupo ARTIST son arrojadas en los resultados de la consulta:

NOMBRE_ARTISTA	TOTAL_VENDIDOS
Bing Crosby	34
Joni Mitchell	79
Patsy Cline	54

Observe que los resultados de la consulta incluyen los mismos tres grupos que fueron arrojados en el ejemplo anterior. La única diferencia es que el valor TOTAL_VENDIDOS en la fila Joni Mitchell es de 79, a diferencia de 45 o 34. La función SUM junta estos dos valores y arroja un valor de 79. Debido a que los otros dos grupos solamente constan de una entrada, sus valores TOTAL_VENDIDOS son los mismos que sus valores VENDIDOS en la tabla CDS_ARTISTA.

No es obligatorio utilizar una cláusula GROUP BY en una instrucción SELECT que utilice una función SUM. Se puede crear una instrucción SELECT tan simple como la siguiente:

```
SELECT SUM(VENDIDOS) AS TOTAL_VENDIDOS
FROM CDS_ARTISTA;
```

Esta instrucción simplemente coloca juntos todos los valores en la columna VENDIDOS y arroja un valor de 292. Por sí misma, esta información no siempre es de mucha ayuda, y es por eso que utilizar la función junto con una cláusula GROUP BY resulta mucho más efectivo.

Utilizar la función AVG

Como se puede imaginar, la función AVG simplemente promedia los valores en una columna especificada. Al igual que la función SUM, es más efectiva cuando se utiliza junto con una cláusula GROUP BY, a pesar de que puede ser utilizada sin la cláusula, como se muestra en el siguiente ejemplo:

```
SELECT AVG(VENDIDOS) AS PROM_VENDIDOS
FROM CDS_ARTISTA;
```

Esta instrucción arroja un valor de 29, que está basado en los valores VENDIDOS en la tabla CDS_ARTISTA. Esto significa que, para todos los CD enlistados en la tabla, se ha vendido un promedio de 29 de cada uno. A pesar de que esta información puede ser bastante útil, sería de mucha más ayuda si se estableciera una instrucción que agrupe los datos:

```
SELECT NOMBRE_ARTISTA, AVG(VENDIDOS) AS PROM_VENDIDOS
FROM CDS_ARTISTA
WHERE VENDIDOS > 30
GROUP BY NOMBRE_ARTISTA;
```

Si se ejecuta esta instrucción, se obtendrán los siguientes resultados de la consulta:

NOMBRE_ARTISTA	PROM_VENDIDO
-----	-----
Bing Crosby	34
Joni Mitchell	39
Patsy Cline	54

Como en los ejemplos anteriores, se crearon tres grupos, y para cada grupo se calcula un promedio basado en los valores de la columna VENDIDOS. Para la fila Joni Mitchell, este promedio está basado en los valores VENDIDOS de 45 y 34. Para las otras dos filas, el promedio es el mismo que el valor VENDIDOS debido a que solamente hay una fila por cada artista.

NOTA

La precisión de los valores arrojados por la función AVG depende del tipo de datos de la columna, si se utilizan decimales, y de cómo promedia los números la implementación SQL. Por ejemplo, el promedio exacto para la fila Joni Mitchell es 39.5, pero debido a que la columna VENDIDOS está configurada con un tipo de datos INT, solamente se utilizan números enteros. Para algunas implementaciones, el .5 es ignorado y no se redondea, como se muestra en los últimos resultados de consulta de ejemplo.

Utilizar funciones de valor

Las funciones de valor son un tipo de función que permite arrojar un valor que de alguna manera calcula o deriva información de los datos almacenados dentro de las tablas o de la misma imple-

mentación SQL. Las funciones de valor son similares a las funciones set en el sentido de que realizan algún tipo de acción por debajo del agua para llegar a ese valor. Sin embargo, las funciones de valor son diferentes de las funciones set en que no requieren que los datos sean agrupados.

SQL soporta diferentes funciones de valor. Cuáles funciones son soportadas por cuáles implementaciones SQL puede variar ampliamente. Además, el significado de un nombre de función puede a veces variar de una implementación a otra. Aun con eso, existen algunas consistencias entre las diferentes implementaciones, y esas son las funciones de valor en las cuales nos enfocaremos.

Las funciones de valor que se discutirán se dividen en dos categorías: funciones de valor de cadena y funciones de valor fecha y hora. Para ilustrar cómo trabajan estas funciones se utilizará la tabla FECHAS_VENTAS, mostrada la figura 10-2.

Trabajar con funciones de valor de cadena

Una función de valor de cadena permite manipular datos de cadenas de caracteres para producir un valor preciso que esté basado en la cadena de caracteres original. Cuando se utiliza una función de valor de cadena, se debe proporcionar la cadena de caracteres como un parámetro de la función. Ese parámetro es entonces convertido a un nuevo valor de acuerdo con el propósito de esa función y de otros parámetros que pudieran ser especificados. En esta sección se presentarán tres funciones de valor de cadena: SUBSTRING, UPPER y LOWER.

Utilizar la función de valor de cadena SUBSTRING

La función del valor de cadena SUBSTRING extrae un número definido de caracteres de una cadena de caracteres identificada para crear una nueva cadena. Esa cadena de caracteres original puede ser derivada de una columna o puede ser declarada explícitamente. En ambos casos, la cade-

DISCO_COMPACTO: VARCHAR(60)	FECHA_VENTA: TIMESTAMP
Famous Blue Raincoat	2002-12-22 10:58:05.120
Blue	2002-12-22 12:02:05.033
Court and Spark	2002-12-22 16:15:22.930
Past Light	2002-12-23 11:29:14.223
That Christmas Feeling	2002-12-23 13:32:45.547
Patsy Cline: 12 Greatest Hits	2002-12-23 15:51:15.730
Out of Africa	2002-12-23 17:01:32.270
Leonard Cohen The Best of	2002-12-24 10:46:35.123
Fundamental	2002-12-24 12:19:13.843
Blues on the Bayou	2002-12-24 14:15:09.673

Figura 10-2 Uso de funciones de valor en la tabla FECHAS_VENTAS.

na de caracteres pasa como un parámetro de la función SUBSTRING, junto con un punto de inicio y, opcionalmente, una especificación de longitud. Por ejemplo, supongamos que se quieren arrojar solamente los primeros 10 caracteres de los valores en la columna DISCO_COMPACTO en la tabla FECHAS_VENTAS. Se puede crear una instrucción SELECT similar a la siguiente:

```
SELECT SUBSTRING(DISCO_COMPACTO FROM 1 FOR 10) AS NOMBRE_ABREVIADO
FROM FECHAS_VENTAS;
```

La función SUBSTRING incluye tres parámetros. El primero es el nombre de la columna, DISCO_COMPACTO, que identifica la fuente utilizada para la cadena de caracteres. El siguiente parámetro, FROM 1, indica que la función empezará a contar a partir del primer carácter. El tercer parámetro, 10, sigue a la palabra clave FOR. El parámetro FOR 10, que es opcional, indica que serán incluidos hasta 10 caracteres en la nueva cadena de caracteres.

NOTA

La mayoría de las implementaciones, incluyendo SQL Server, MySQL y Oracle, no utilizan las palabras clave FROM y FOR (simplemente se separan los parámetros utilizando comas). Además, en Oracle la función es llamada SUBSTR. Aquí está la misma instrucción modificada para Oracle:

```
SELECT SUBSTR(DISCO_COMPACTO, 1, 10) AS NOMBRE_ABREVIADO
FROM FECHAS_VENTAS;
```

Si se ejecuta esta instrucción SELECT, se obtendrán los siguientes resultados de la consulta:

```
NAME_ABREVIADO
```

```
-----
```

```
Famous Blu
Blue
Court and
Past Light
That Chris
Patsy Clin
Out of Afr
Leonard Co
Fundamenta
Blues on t
```

Observe que solamente los primeros 10 caracteres de cada valor DISCO_COMPACTO están incluidos en los resultados. Para aquellos valores que tiene menos de 10 caracteres, aparece el nombre completo.

El parámetro FROM puede aceptar un número negativo o un cero como parámetro, asumiendo que la implementación SQL lo permite. Cuando utilice un número negativo o un cero, tenga en cuenta que el 1 representa lo que se puede considerar como una posición de inicio normal. El siguiente carácter a la izquierda del 1 es 0. El carácter a la izquierda de 0 es -1, y así sucesivamente. El parámetro FOR cuenta caracteres empezando desde el punto de inicio. Si se utiliza un cero o un número negativo, la función SUBSTRING actúa como si de todas maneras existieran caracteres

en esos lugares. Por ejemplo, supongamos que se modifica la instrucción SELECT anterior de la siguiente manera:

```
SELECT SUBSTRING(DISCO_COMPACTO FROM -2 FOR 10) AS NOMBRE_ABREVIADO
FROM FECHAS_VENTAS;
```

Si se ejecuta esta instrucción, solamente los primeros siete caracteres de cada nombre serían arrojados. Si en lugar de eso se utiliza un cero, solamente los primeros nueve caracteres serían arrojados. Solamente cuando se utiliza un parámetro FROM de 1 se arroja exactamente el número de caracteres (desde la cadena de caracteres) que están especificados por el parámetro FOR.

La función SUBSTRING no está limitada a la cláusula SELECT. De hecho, utilizarla en una cláusula WHERE puede ser muy útil cuando se define una condición de búsqueda. Por ejemplo, la siguiente instrucción SELECT utiliza la función SUBSTRING para arrojar filas que inician con Blue:

```
SELECT DISCO_COMPACTO, FECHA_VENTA
FROM FECHAS_VENTAS
WHERE SUBSTRING(DISCO_COMPACTO FROM 1 FOR 4) = 'Blue';
```

En esta instrucción, la función SUBSTRING arroja los primeros cuatro caracteres de los valores DISCO_COMPACTO y los compara con el valor Blue. Solamente dos filas son incluidas en los resultados de la consulta:

DISCO_COMPACTO	FECHA_VENTA
Blue	2002-12-22 12:02:05.033
Blues on the Bayou	2002-12-24 14:15:09.673

Ambas filas en los resultados de la consulta tienen un valor DISCO_COMPACTO que inicia con Blue. Ninguna otra fila cumple con la condición de búsqueda especificada en la cláusula WHERE.

NOTA

El manejo de datos de fecha y hora varía considerablemente a través de las implementaciones de SQL, por lo que los resultados de su DBMS pueden variar enormemente con respecto a la columna FECHA_VENTA.

Utilizar las funciones de valor de cadena UPPER y LOWER

Las funciones de valor de cadena UPPER y LOWER son muy similares en que ambas son utilizadas para convertir caracteres entre mayúsculas y minúsculas. La función UPPER permite convertir una cadena de caracteres completa a mayúsculas. La función LOWER permite convertir una cadena de caracteres completa a minúsculas. Por ejemplo, supongamos que se quiere modificar la instrucción SELECT mostrada en el último ejemplo para arrojar todos los valores DISCO_COMPACTO en mayúsculas. Su instrucción SELECT ahora incluiría una función UPPER:

```
SELECT UPPER(DISCO_COMPACTO) AS TITULO, FECHA_VENTA
FROM FECHAS_VENTAS
WHERE SUBSTRING(DISCO_COMPACTO FROM 1 FOR 4) = 'Blue';
```

Los resultados de la consulta son los mismos que en el último ejemplo, sólo que esta vez en los títulos de CD están todos en mayúsculas, como se muestra en los siguientes resultados:

Título	FECHA_VENTA
-----	-----
BLUE	2002-12-22 12:02:05.033
BLUES ON THE BAYOU	2002-12-24 14:15:09.673

Si se hubiera utilizado la función LOWER en lugar de UPPER, los títulos de CD estarían todos en minúsculas, incluso sin mayúscula al inicio de las palabras. Estas funciones también son muy útiles para comparar datos en implementaciones que son sensibles a mayúsculas y minúsculas cuando no se sabe cuál de las dos fue utilizada para almacenar los datos, o cuando se quiere estar seguro de que los datos aparezcan en mayúsculas o minúsculas al momento que se están insertando, actualizando o convirtiendo datos de una base de datos a otra.

Trabajar con funciones de valor de fecha y hora

Las funciones de valor de fecha y hora proporcionan información acerca de la fecha y la hora actuales. Cada función arroja un valor basado en la fecha u hora (o ambos) tal como están configurados en el sistema operativo. SQL:2006 soporta cinco funciones de valor de fecha y hora, que se describen en la tabla 10-1.

NOTA

Las implementaciones SQL varían ampliamente en cuanto a cómo implementan la funcionalidad fecha y hora; consecuentemente, la implementación de funciones fecha y hora también varía. Por ejemplo, SQL Server soporta solamente la función de valor fecha y hora CURRENT_TIMESTAMP. Por otro lado, Oracle soporta las funciones de valor fecha y hora CURRENT_DATE, CURRENT_TIMESTAMP y LOCALTIMESTAMP, pero no las funciones CURRENT_TIME y LOCALTIME. No obstante, MySQL soporta todas las cinco. Adicionalmente, los valores exactos generados por estas funciones también pueden variar de implementación a implementación. Por ejemplo, los resultados de consulta no siempre incluirán información acerca del huso horario actual, y algunos representan la hora utilizando un formato de 24 horas en lugar de A.M. y P.M.

Función de valor	Descripción
CURRENT_DATE	Arroja un valor que representa la fecha actual.
CURRENT_TIME	Arroja un valor que representa la hora actual. El valor incluye información acerca del huso horario actual, concerniente a Coordinated Universal Time (UCT), o lo que antes era llamado Greenwich Mean Time (GMT).
CURRENT_TIMESTAMP	Arroja un valor que representa la fecha y la hora actuales. El valor incluye información acerca del huso horario actual, concerniente al UCT.
LOCALTIME	Arroja un valor que representa la hora actual.
LOCALTIMESTAMP	Arroja un valor que representa la fecha y la hora actuales.

Tabla 10-1 Funciones de valor fecha y hora soportadas por SQL:2006.

Debido a que la función de valor fecha y hora `CURRENT_TIMESTAMP` es soportada tanto por SQL Server como por Oracle, démosle un vistazo más detallado. No obstante, tenga en mente que implementar cualquiera de las funciones fecha y hora de SQL es un proceso similar, dependiendo de cuáles funciones se soporten por la implementación específica SQL en la que se esté trabajando. Al comprender cómo trabaja la función `CURRENT_TIMESTAMP`, se obtendrá una mucho mejor comprensión de cómo trabajan todas las funciones. Sin embargo, asegúrese de revisar la documentación de su implementación para mayor información sobre cualquiera de las funciones que sean soportadas por ese producto.

Dependiendo de la implementación SQL, se puede utilizar la función `CURRENT_TIMESTAMP` en una instrucción `SELECT` para simplemente recuperar la información actual de la marca de fecha y hora. Como se puede esperar de cualquier cuestión relacionada con la funcionalidad fecha y hora, la forma en que se convoca una función puede variar. Sin embargo, en algunos casos es posible utilizar una instrucción tan sencilla como la siguiente:

```
SELECT CURRENT_TIMESTAMP
```

Esta instrucción recuperará la fecha y hora actuales en algunas implementaciones. En otras implementaciones, puede ser necesario agregar una cláusula `FROM` a la instrucción para recuperar esta información. Por ejemplo, Oracle proporciona una tabla simulada llamada específicamente `DUAL` porque requiere una cláusula `FROM` en todas las instrucciones `SELECT`. Sin importar cómo se necesite escribir la instrucción `SELECT`, con toda probabilidad no será muy aprovechable utilizar una función `CURRENT_TIMESTAMP` de esta manera. Probablemente se hará un mejor uso de las funciones fecha y hora utilizándolas para comparar datos o para insertar datos automáticamente.

NOTA

La mayoría de las implementaciones SQL tienen funciones especiales para manejar los datos de fecha y hora. Por ejemplo, SQL Server proporciona la función `getdate` para arrojar la fecha actual, mientras que Oracle proporciona el valor especial `SYSDATE` para el mismo propósito. Las primeras implementaciones SQL no incluían ningún soporte para los tipos de datos fecha y hora, pero tan pronto se volvieron populares las bases de datos relacionales en aplicaciones de negocios, los usuarios empezaron a pedirlos. Esto empezó a confundir a los fabricantes al agregar nuevas características, y debido a que no había un estándar SQL a seguir, el resultado fue una amplia variación entre las implementaciones. Productos como MySQL que fueron desarrollados después del estándar tienen muy pocas de esas variaciones. Como siempre, revise la información del fabricante para mayores detalles.

Por ejemplo, supongamos que se quiere utilizar la tabla `FECHAS_VENTAS` (mostrada en la figura 10-2) para insertar la hora y fecha actuales automáticamente en la tabla cada vez que se agregue una nueva fila. La definición de dicha tabla podría lucir de la siguiente manera:

```
CREATE TABLE FECHAS_VENTAS
( DISCO_COMPACTO VARCHAR(60),
  FECHA_VENTA          DATETIME          DEFAULT CURRENT_TIMESTAMP );
```

En esta definición de tabla se ha asignado un valor por defecto a la columna `FECHA_VENTA` que está basado en la función `CURRENT_TIMESTAMP`. Cada vez que una fila sea agregada a la tabla, el valor de fecha y hora será insertado en la columna `DATE_SOLD` para esa fila. Como

resultado, se pueden crear instrucciones INSERT que especifiquen solamente el valor DISCO_COMPACTO. La fecha y la hora actuales, por lo tanto, son agregadas automáticamente a la columna DATE_SOLD al momento que se agrega la fila.

Utilizar expresiones de valor

Una expresión de valor es un tipo de expresión que arroja un valor de datos. La expresión puede incluir nombres de columna, valores, operadores matemáticos, palabras clave u otros elementos que juntos creen algún tipo de fórmula o expresión que arroje un valor único. Por ejemplo, se pueden combinar los valores de dos columnas para crear un valor, o se pueden realizar operaciones en el valor de una columna para crear un nuevo valor.

En esta sección daremos un vistazo a las expresiones de valor numéricas y también a las expresiones de valor CASE y CAST. Para demostrar cómo funcionan varias de estas expresiones, utilizaremos la tabla RASTREO_CD, mostrada en la figura 10-3.

Trabajar con expresiones de valor numéricas

Las expresiones de valor numéricas son expresiones que utilizan operadores matemáticos para realizar cálculos sobre valores de datos numéricos almacenados en tablas. Se pueden utilizar estos operadores para sumar, sustraer, multiplicar y dividir estos valores. La tabla 10-2 muestra los cuatro operadores que se pueden utilizar para crear expresiones de valor numéricas.

Es posible construir expresiones de valor numéricas de una forma muy parecida a como se construyen las fórmulas matemáticas. Los principios básicos son los mismos. Por ejemplo, la multiplicación y la división toman precedencia sobre la suma y la resta, y los elementos que deben ser

NOMBRE_CD: VARCHAR(60)	CATEGORÍA_CD CHAR(4)	EN_EXISTENCIA: INT	EN_PEDIDO: INT	VENDIDOS: INT
Famous Blue Raincoat	FROK	19	16	34
Blue	CPOP	28	22	56
Court and Spark	CPOP	12	11	48
Past Light	NEWA	6	7	22
That Christmas Feeling	XMAS	14	14	34
Patsy Cline: 12 Greatest Hits	CTRY	15	18	54
Out of Africa	STRK	8	5	26
Leonard Cohen The Best of	FROK	6	8	18
Fundamental	BLUS	10	6	21
Blues on the Bayou	BLUS	11	10	17

Figura 10-3 Utilizar expresiones de valor en la tabla RASTREO_CD.

Expresión	Operador	Ejemplo
Suma	+	EN_EXISTENCIA + EN_PEDIDO
Sustracción	-	VENDIDOS - (EN_EXISTENCIA + EN_ORDEN)
Multiplicación	*	EN_EXISTENCIA * 2
División	/	VENDIDOS / 2

Tabla 10-2 Utilizar expresiones de valor numéricas para calcular datos.

calculados primero se encierran en paréntesis; de otra manera, cada operación se calcula de acuerdo con la precedencia y orden en la cual fue descrita. Por ejemplo, la fórmula $2 + 2 * 5 / 4$ es igual a 4.5; sin embargo, la fórmula $(2 + 2) * 5 / 4$ es igual a 5. En la primer fórmula, el 2 fue multiplicado por 5, luego dividido entre 4 y luego se le sumaron 2. En la segunda fórmula, al 2 se le sumaron 2, luego fue multiplicado por 5 y luego dividido entre 4.

Ahora demos un vistazo a un ejemplo de una expresión de valor numérica. Supongamos que se quiere agregar la columna EN_EXISTENCIA a la columna EN_PEDIDO en la tabla RASTREO_CD. Se puede crear una instrucción SELECT similar a la siguiente:

```
SELECT NOMBRE_CD, EN_EXISTENCIA, EN_PEDIDO, (EN_EXISTENCIA + EN_PEDIDO)
AS TOTAL
FROM RASTREO_CD
```

Como se puede ver, la cláusula SELECT especifica primero tres nombres de columna: NOMBRE_CD, EN_EXISTENCIA y EN_PEDIDO. Por lo regular éstos son seguidos por una expresión de valor numérica: (EN_EXISTENCIA + EN_PEDIDO). Los valores de las columnas EN_EXISTENCIA y EN_PEDIDO se agregan juntos y se incluyen en los resultados de la consulta bajo una columna llamada TOTAL, como se muestra en los siguientes resultados:

NOMBRE_CD	EN_EXISTENCIA	EN_PEDIDO	TOTAL
-----	-----	-----	-----
Famous Blue Raincoat	19	16	35
Blue	28	22	50
Court and Spark	12	11	23
Past Light	6	7	13
That Christmas Feeling	14	14	28
Patsy Cline: 12 Greatest Hits	15	18	33
Out of Africa	8	5	13
Leonard Cohen The Best Of	6	8	14
Fundamental	10	6	16
Blues on the Bayou	11	10	21

Para cada fila se ha agregado un valor a la columna TOTAL que junta los valores en la columna EN_EXISTENCIA y en la columna EN_PEDIDO.

Las expresiones de valor numéricas no están limitadas a la cláusula SELECT. Por ejemplo, se puede utilizar una en una cláusula WHERE para especificar una condición de búsqueda. Supongamos que se quieren arrojar los mismos resultados que en la instrucción SELECT anterior pero sólo

para aquellos CD con un valor TOTAL mayor a 25. Se puede modificar la instrucción de la manera siguiente:

```
SELECT NOMBRE_CD, EN_EXISTENCIA, EN_PEDIDO, (EN_EXISTENCIA + EN_PEDIDO)
AS TOTAL
FROM RASTREO_CD
WHERE (EN_EXISTENCIA + EN_PEDIDO) > 25
```

Ahora los resultados de la búsqueda incluirán solamente 4 filas, como se muestra a continuación:

NOMBRE_CD	EN_EXISTENCIA	EN_PEDIDO	TOTAL
Famous Blue Raincoat	19	16	35
Blue	28	22	50
That Christmas Feeling	14	14	28
Patsy Cline: 12 Greatest Hits	15	18	33

Los operadores de valor numéricos pueden también ser combinados entre sí para crear expresiones más complejas. En el siguiente ejemplo se incluye una expresión adicional que calcula tres conjuntos de valores y los combina en una columna en los resultados de la consulta:

```
SELECT NOMBRE_CD, EN_EXISTENCIA, EN_PEDIDO, (EN_EXISTENCIA + EN_PEDIDO)
AS TOTAL,
SOLD, (SOLD - (EN_EXISTENCIA + EN_PEDIDO)) AS ESCASEZ
FROM RASTREO_CD
WHERE (EN_EXISTENCIA + EN_PEDIDO) > 25
```

Esta instrucción permite calcular cuántos CD se tienen disponibles (EN_EXISTENCIA + EN_PEDIDO) y se comparan con cuántos se han vendido. Luego la diferencia se agrega en la columna ESCASEZ en los resultados de la consulta. Si se han vendido más CD de los que están disponibles, un número positivo se coloca en la columna ESCASEZ. Si, por otro lado, hay suficientes CD disponibles, se crea un número negativo. Los siguientes resultados de la consulta muestran las cantidades calculadas cuando se ejecuta la instrucción SELECT:

NOMBRE_CD	EN_EXISTENCIA	EN_PEDIDO	TOTAL	VENDIDOS	ESCASEZ
Famous Blue Raincoat	19	16	35	34	-1
Blue	28	22	50	56	6
That Christmas Feeling	14	14	28	34	6
Patsy Cline: 12 Greatest Hits	15	18	33	54	21

Los resultados de la consulta ahora incluyen dos columnas calculadas: TOTAL y ESCASEZ. Todos los demás valores (EN_EXISTENCIA, EN_PEDIDO y VENDIDOS) se toman directamente de la tabla.

Como se puede ver, las expresiones de valor numéricas son muy flexibles y pueden ser utilizadas de muchas diferentes formas. Como un agregado a los métodos que se han visto hasta ahora, se pueden combinar valores de columna con valores específicos. Por ejemplo, supongamos que se

quiere revisar cuántos CD se tendrían disponibles si se duplicara la cantidad que se tenía bajo pedido para aquellos CD en los que hay menos de 15 disponibles:

```
SELECT NOMBRE_CD, EN_EXISTENCIA, EN_PEDIDO, (EN_EXISTENCIA + EN_PEDIDO)
AS TOTAL,
      (EN_EXISTENCIA + EN_PEDIDO * 2) AS DOBLE_ORDEN
FROM RASTREO_CD
WHERE (EN_EXISTENCIA + EN_PEDIDO) < 15
```

La segunda expresión de valor numérica en esta instrucción multiplica el valor EN_PEDIDO por 2, lo suma al valor EN_EXISTENCIA, e inserta el total en la columna DOBLE_ORDEN de los resultados de la consulta, como se muestra en los siguientes resultados:

NOMBRE_CD	EN_EXISTENCIA	EN_PEDIDO	TOTAL	DOBLE_ORDEN
-----	-----	-----	-----	-----
Past Light	6	7	13	20
Out of Africa	8	5	13	18
Leonard Cohen The Best Of	6	8	14	22

Los resultados de la consulta incluyen solamente tres filas que cumplen la condición de la cláusula WHERE. Para estas filas, las columnas EN_EXISTENCIA y EN_PEDIDO se calculan para proporcionar los datos que puedan ser útiles para el usuario, dependiendo de sus necesidades. La mejor parte es que estos valores no tienen que ser almacenados en la base de datos. En su lugar, son calculados cuando se ejecuta la instrucción SELECT, en lugar de tener que mantener tablas con datos adicionales.

Utilizar la expresión de valor CASE

Una expresión de valor CASE permite determinar una serie de condiciones que modifican valores específicos arrojados por la instrucción SQL. Se puede cambiar la forma en que un valor está representado o se calcula un nuevo valor. Cada valor es modificado de acuerdo con la condición especificada dentro de la expresión CASE. Una expresión de valor incluye la palabra clave CASE y una lista de condiciones. La última condición proporciona una condición por defecto si ninguna de las condiciones anteriores ha sido cumplida. Entonces la expresión de valor se cierra utilizando la palabra clave END.

Demos un vistazo a un ejemplo para proporcionar una mejor idea de cómo funciona esto. Supongamos que se quiere incrementar el número de CD que se tienen bajo pedido, pero se quiere incrementar la cantidad solamente para ciertos CD. Además, se quiere fundamentar cuántos CD se agregan al pedido sobre la cantidad actual. Antes de actualizar realmente la tabla, se puede ver lo que serían los nuevos valores al crear una instrucción SELECT que consulte la tabla RASTREO_CD, como se muestra en el siguiente ejemplo:

```
SELECT NOMBRE_CD, EN_PEDIDO, NUEVAS_ORDENES =
CASE
  WHEN EN_PEDIDO < 6 THEN EN_PEDIDO + 4
  WHEN EN_PEDIDO BETWEEN 6 AND 8 THEN EN_PEDIDO + 2
  ELSE EN_PEDIDO
END
FROM RASTREO_CD
WHERE EN_PEDIDO < 11;
```

En esta instrucción se especifican tres columnas: NOMBRE_CD, EN_PEDIDO y NUEVAS_ORDENES. La columna NUEVAS_ORDENES es la columna creada por los resultados de la búsqueda. Contendrá los valores actualizados para la expresión de valor CASE. La expresión en sí misma consta del nombre de columna (NUEVAS_ORDENES), el signo igual, la palabra clave CASE, dos cláusulas WHEN/THEN, una cláusula ELSE y la palabra clave END. Cada cláusula WHEN/THEN representa una de las condiciones. Por ejemplo, la primera cláusula especifica que si el valor EN_PEDIDO es menor a 6, entonces 4 deberá ser agregado a ese valor. La segunda cláusula WHEN/THEN especifica que si el valor EN_PEDIDO cae dentro del rango de 6 a 8, entonces 2 deberá ser agregado al valor.

Después de las cláusulas WHEN/THEN, la cláusula ELSE especifica la condición final. Si el valor no cumple las condiciones definidas en las cláusulas WHEN/THEN, la cláusula ELSE especifica una condición por defecto. En el caso de la instrucción SELECT anterior, la cláusula ELSE simplemente se refería a la columna EN_PEDIDO, sin especificar ninguna modificación. (Esto resulta lo mismo que definir $EN_PEDIDO + 0$.) En otras palabras, si ninguna de las condiciones WHEN/THEN se cumplen, el valor EN_PEDIDO permanece igual. Si se ejecutara la instrucción SELECT, se recibirían los siguientes resultados:

NOMBRE_CD	EN_PEDIDO	NUEVAS_ORDENES
-----	-----	-----
Past Light	7	9
Out of Africa	5	9
Leonard Cohen The Best Of	8	10
Fundamental	6	8
Blues on the Bayou	10	10

Como se puede ver, la fila Out of Africa se incrementa en 4, la fila Blues on the Bayou no cambia en lo absoluto, y las otras tres filas se incrementan en 2.

Además de modificar los valores, se puede utilizar una expresión de valor CASE para renombrar valores. Esto es particularmente útil si los resultados de la consulta incluyen valores que no son reconocibles fácilmente. Por ejemplo, supongamos que se quiere crear una consulta que arroje datos desde la columna CD_CATEGORY de la tabla RASTREO_CD. Se pueden renombrar los valores en la columna para que la información arrojada sea más entendible para los usuarios, como se muestra en la siguiente instrucción SELECT:

```
SELECT NOMBRE_CD, CATEGORIA_CD =
CASE
  WHEN CATEGORIA_CD = 'FROK' THEN 'Folk Rock'
  WHEN CATEGORIA_CD = 'CPOP' THEN 'Classic Pop'
  WHEN CATEGORIA_CD = 'NEWA' THEN 'New Age'
  WHEN CATEGORIA_CD = 'XMAS' THEN 'Christmas'
  WHEN CATEGORIA_CD = 'CTRY' THEN 'Country'
  WHEN CATEGORIA_CD = 'STRK' THEN 'Soundtrack'
  WHEN CATEGORIA_CD = 'BLUS' THEN 'Blues'
  ELSE NULL
END
FROM RASTREO_CD;
```

NOTA

No es necesario colocar los diferentes componentes de la expresión de valor CASE en líneas separadas, como se hizo aquí. Se hizo de esa forma para mostrar más claramente cada componente. También devuelve el código más legible para cualquiera que lo revise.

En esta instrucción SELECT, los diferentes valores en la columna CATEGORIA_CD fueron renombrados a nombres más comunes. Observe que no se necesitan repetir los nombres de columna a la derecha de la palabra clave THEN. La construcción del predicado se asume por el contexto de la cláusula. Cuando se ejecuta esta instrucción, se reciben los siguientes resultados de la consulta:

NOMBRE_CD	CATEGORIA_CD
-----	-----
Famous Blue Raincoat	Folk Rock
Blue	Classic Pop
Court and Spark	Classic Pop
Past Light	New Age
That Christmas Feeling	Christmas
Patsy Cline: 12 Greatest Hits	Country
Out of Africa	Soundtrack
Leonard Cohen The Best Of	Folk Rock
Fundamental	Blues
Blues on the Bayou	Blues

Como se puede ver, nombres más fáciles de utilizar aparecen en la columna CATEGORIA_CD. Si ninguno de los valores originales cumple con la condición definida en las cláusulas WHEN/THEN, un valor nulo será insertado en los resultados de la consulta.

Pregunta al experto

P: ¿Se puede utilizar una expresión de valor CASE en una instrucción diferente a SELECT?

R: También se puede hacer uso de la expresión de valor CASE en la cláusula SET o en una instrucción UPDATE. Por ejemplo, supongamos que se quieren actualizar los valores en la columna EN_PEDIDO en la tabla RASTREO_CD (mostrada en la figura 10-3). Es posible actualizar esos valores definiendo condiciones específicas en una expresión CASE:

```
UPDATE RASTREO_CD
  SET EN_PEDIDO =
CASE
  WHEN EN_PEDIDO < 6 THEN EN_PEDIDO + 4
  WHEN EN_PEDIDO BETWEEN 6 AND 8 THEN EN_PEDIDO + 2
  ELSE EN_PEDIDO
END
```

(continúa)

Esta instrucción agregará 4 a los valores EN_PEDIDO que sean menores a 6, y agregará 2 a los valores EN_PEDIDO que caigan dentro del rango de 6 a 8. De otra manera, no se cambiarán filas adicionales.

P: ¿Se puede hacer referencia a más de una columna en la expresión de valor CASE?

R: Sí, se puede hacer referencia a más de una columna. Supongamos que se quieren actualizar los valores EN_PEDIDO, pero basando esas actualizaciones en los valores CATEGORIA_CD. Se puede crear una instrucción similar a la siguiente:

```
UPDATE RASTREO_CD
  SET EN_PEDIDO =
  CASE
    WHEN CATEGORIA_CD = 'CPOP' THEN EN_PEDIDO * 3
    WHEN CATEGORIA_CD = 'BLUS' THEN EN_PEDIDO * 2
    ELSE EN_PEDIDO
  END
```

En esta instrucción, los valores EN_PEDIDO son multiplicados por 3 cuando los valores CATEGORIA_CD igualan a CPOP, y los valores EN_PEDIDO son multiplicados por 2 cuando los valores CATEGORIA_CD igualan BLUS. De otra manera, no se cambia ningún valor.

P: ¿Existe alguna implicación de rendimiento cuando se utiliza la expresión de valor CASE?

R: Aunque no existen detalles inherentes al rendimiento relacionados al uso de las expresiones de valor CASE, sí es posible pasar los límites. En general, mientras más compleja es la instrucción, especialmente en términos de anidado y de una lógica condicional elaborada, mayores recursos se necesitarán para analizar gramáticamente y ejecutar la instrucción.

Utilizar la expresión de valor CAST

La expresión de valor CAST sirve a un propósito muy diferente al de la expresión CASE. La expresión CAST permite cambiar el tipo de datos de un valor por un valor literal o cuando se recupera ese valor desde la base de datos. Sin embargo, no cambia el tipo de datos de la columna fuente. Esto es particularmente útil cuando se trabaja con lenguajes de programación en los cuales los tipos de datos no coinciden y se necesita utilizar un común denominador para trabajar con el valor.

Para utilizar la expresión de valor CAST, se debe especificar la palabra clave CAST, y, en paréntesis, proporcionar el nombre de columna, la palabra clave AS, y el nuevo tipo de datos, en ese orden. Para ilustrar esto, regresemos a la tabla FECHAS_VENTAS mostrada en la figura 10-2. La tabla incluye la columna DISCO_COMPACTO y la columna FECHA_VENTA. La columna FECHA_VENTA se configura con el tipo de datos TIMESTAMP. Supongamos que se desea cam-

biar los valores de fecha y hora a cadenas de caracteres. Se puede utilizar la expresión CAST en la cláusula SELECT, como se muestra en la siguiente instrucción:

```
SELECT DISCO_COMPACTO, CAST(FECHA_VENTA AS CHAR(25)) AS CHAR_FECHA
FROM FECHAS_VENTAS
WHERE DISCO_COMPACTO LIKE ('%Blue%')
```

Esta instrucción convierte los valores FECHA_VENTA de los valores TIMESTAMP a los valores CHAR. Como se puede ver, todo lo que se necesita hacer es especificar la palabra clave CAST, seguida por los parámetros en paréntesis que identifican la columna fuente y el nuevo tipo de datos, junto con la palabra clave AS. Cuando se ejecuta esta instrucción, se reciben resultados de consulta similares a lo que se vería si no se hubiera utilizado CAST:

DISCO_COMPACTO	CHAR_FECHA
-----	-----
Famous Blue Raincoat	Dec 22 2002 10:58AM
Blue	Dec 22 2002 12:02PM
Blues on the Bayou	Dec 24 2002 2:15PM

Observe que se puede asignar un nombre a la columna que contiene los nuevos resultados de fecha y hora. En este caso, el nombre de la nueva columna es CHAR_FECHA.

NOTA

Se puede encontrar que, dependiendo de la implementación SQL, cuando un valor de fecha y hora es convertido, el formato cambia ligeramente. Por ejemplo, en SQL Server, un valor de fecha es expresado numéricamente y un valor de hora es expresado como un reloj de 24 horas (horario militar), pero cuando valor es convertido a un tipo de datos CHAR, el valor de tiempo es expresado en caracteres alfanuméricos, y la hora se expresa como un reloj de 12 horas (A.M. y P.M).

Utilizar valores especiales

En el capítulo 6 se analizaron valores especiales soportados por SQL que permiten determinar a los usuarios actuales. Un valor especial existe para cada tipo de usuario. Estos valores actúan como marcadores de posición para los valores actuales relacionados con los usuarios. Pueden utilizarse en expresiones para arrojar el valor del usuario específico. SQL soporta cinco valores especiales, que se describen en la tabla 10-3. (Véase el capítulo 6 para mayor información acerca de los diferentes tipos de usuarios SQL.)

Los valores especiales pueden ser utilizados en diferentes formas en una base de datos SQL, por ejemplo para establecer conexiones o ejecutar un procedimiento almacenado. El valor especial, en lugar del nombre del usuario actual, es incrustado en el código para permitir al código permanecer flexible de una situación a la otra. Otra forma en la que un valor especial puede ser utilizado es para almacenar datos de usuario en una tabla. Para ilustrar esto, demos un vistazo a la tabla PEDIDOS_CD en la figura 10-4.

Esta vez se agrega una fila a la tabla, y se inserta un valor para CURRENT_USER dentro de la columna PEDIDO_POR. Esto facilita rastrear cuál usuario ha colocado la orden. Si se observara

Valor	Descripción
CURRENT_USER	Identifica al identificador de usuario actual. Si el identificador de usuario de la sesión SQL es el identificador de usuario actual, entonces CURRENT_USER, USER y SESSION_USER tienen el mismo valor, lo que puede ocurrir si el par de identificador inicial es el único par identificador de usuario activo/nombre de rol (el par en la parte superior de la pila de autenticación).
USER	Identifica al identificador de usuario actual. USER tiene el mismo significado que CURRENT_USER.
SESSION_USER	Identifica al identificador de usuario de la sesión SQL.
CURRENT_ROLE	Identifica el nombre de rol actual.
SYSTEM_USER	Identifica al usuario del sistema operativo actual que invocó un módulo SQL.

Tabla 10-3 Utilizar los valores especiales de SQL:2006.

la definición de la tabla, se vería que un valor por defecto ha sido definido para la columna PEDIDO_POR, como se muestra en la siguiente instrucción CREATE TABLE:

```
CREATE TABLE PEDIDOS_CD
( TITULOCD VARCHAR(60),
  PEDIDO INT,
  PEDIDO_POR CHAR(30) DEFAULT CURRENT_USER );
```

Si fueran a insertarse datos a esta tabla, se tendría que especificar solamente un valor TITULO_CD y un valor PEDIDO. El valor PEDIDO_POR sería insertado automáticamente, y ese valor

NOMBRE_CD: VARCHAR(60)	PEDIDO: INT	PEDIDO_POR: CHAR(30)
Famous Blue Raincoat	16	Mngr
Blue	22	AsstMngr
Court and Spark	11	Mngr
Past Light	7	AsstMngr
That Christmas Feeling	14	Mngr
Patsy Cline: 12 Greatest Hits	18	AsstMngr
Out of Africa	5	AsstMngr
Leonard Cohen The Best of	8	Mngr
Fundamental	6	Mngr
Blues on the Bayou	10	Mngr

Figura 10-4 Utilizar el valor especial CURRENT_USER en la tabla PEDIDOS_CD.

sería el identificador de usuario actual. Si no se especifica un valor por defecto para la columna PEDIDO_POR, se puede utilizar el valor especial para insertar al usuario. Por ejemplo, la siguiente instrucción INSERT inserta una fila en la tabla PEDIDOS_CD:

```
INSERT INTO PEDIDOS_CD
VALUES ( 'Rhythm Country and Blues', 14, CURRENT_USER );
```

Cuando se ejecuta la instrucción, un valor representando al identificador de usuario actual (por ejemplo Mngrr) se inserta en la columna PEDIDO_POR.

Para determinar la extensión para la que se pueden utilizar los valores especiales, deberá revisar la documentación del producto para su implementación SQL. Encontrará que las formas en las que se pueden utilizar estos valores tendrán variaciones de una implementación a otra; sin embargo, una vez que se sienta cómodo utilizando valores especiales en su implementación, encontrará que son una herramienta muy útil, al mismo tiempo que usted se hace más eficiente con la programación SQL.

Pruebe esto 10-1

Utilizar funciones y expresiones de valor

En este capítulo aprendió acerca de muchas de las funciones y expresiones de valor soportadas por SQL. Ahora se ejercitarán esas funciones y expresiones consultando datos desde la base de datos INVENTARIO. Específicamente, se crearán instrucciones SELECT que contengan las funciones COUNT, MIN, SUM, SUBSTRING y UPPER, y aquellas que contengan expresiones de valor numéricas, CASE y CAST. Puede descargar el archivo Try_This_10.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. En la primera instrucción se determinará el número de valores NOMBRE_ARTISTA únicos en la tabla ARTISTAS. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT COUNT(DISTINCT NOMBRE_ARTISTA) AS ARTISTAS
FROM ARTISTAS;
```

La consulta deberá arrojar una cuenta de 18.

3. En la siguiente instrucción se determinará el número mínimo de CD en existencia, como se enlistan en la tabla DISCOS_COMPACTOS. Se nombra la columna en los resultados de la consulta MIN_EXISTENCIA. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT MIN(EN_EXISTENCIA) AS MIN_EXISTENCIA
FROM DISCOS_COMPACTOS;
```

Los resultados de la consulta deberán incluir solamente una columna y una fila, y mostrar un valor de 5. Esto significa que 5 es el número menor de CD que se tienen en existencia para cualquier CD.

(continúa)

4. Ahora se determinará el número total de CD en existencia. Sin embargo, esta vez se agruparán estos totales de acuerdo con los valores ID_DISQUERA. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL
FROM DISCOS_COMPACTOS
GROUP BY ID_DISQUERA;
```

La consulta arrojará 10 filas una por cada valor ID_DISQUERA. El valor TOTAL para cada fila representa el número total de CD para ese grupo particular ID_DISQUERA.

5. En los pasos anteriores se utilizaron funciones set al consultar datos desde la base de datos INVENTARIO. Ahora se practicarán un par de funciones de valor. La primera de éstas es SUBSTRING. En la instrucción SELECT se extraerán los datos desde la columna LUGAR_DE_NACIMIENTO en la tabla ARTISTAS. Se necesita extraer ocho caracteres, empezando por el primer carácter en la cadena. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT NOMBRE_ARTISTA,
       SUBSTRING(LUGAR_DE_NACIMIENTO FROM 1 FOR 8) AS LUGAR_NACIMIENTO
FROM ARTISTAS;
```

Los resultados de la consulta deberán arrojar 18 filas e incluir dos columnas: NOMBRE_ARTISTA y LUGAR_NACIMIENTO. La columna LUGAR_NACIMIENTO contiene los valores extraídos, los cuales están basados en la columna LUGAR_DE_NACIMIENTO de la tabla.

6. La siguiente función de valor que se ejercitará es la función UPPER. En esta instrucción SELECT, los nombres de todos los CD se convertirán a mayúsculas. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT UPPER(TITULO_CD) AS NOMBRE_CD
FROM DISCOS_COMPACTOS;
```

Esta instrucción deberá arrojar 15 filas con una sola columna que enlista el nombre de los CD en la tabla DISCOS_COMPACTOS. Todos los títulos de los CD deberán estar en mayúsculas.

7. Ahora nos moveremos a las expresiones de valor numéricas. La siguiente instrucción que se utilizará crea dos columnas en los resultados de la consulta que duplican y triplican los valores en la columna EN_EXISTENCIA de la tabla DISCOS_COMPACTOS. Sin embargo, la instrucción arroja valores solamente para aquellas filas con un valor EN_EXISTENCIA menor a 25. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA,
       (EN_EXISTENCIA * 2) AS DOBLE, (EN_EXISTENCIA * 3) AS TRIPLE
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA < 25;
```

La instrucción SELECT deberá arrojar nueve filas, y cada una deberá incluir valores EN_EXISTENCIA que han sido multiplicados por 2 y por 3.

8. La siguiente expresión de valor será la expresión CASE. Esta instrucción proporcionará valores EN_EXISTENCIA actualizados a la columna EN_PEDIDO de los resultados de la consulta. Para los valores EN_EXISTENCIA menores a 10, los valores serán duplicados. Para los valores EN_EXISTENCIA que caigan dentro del rango entre 10 y 15, será sumado un 3. Todos los demás valores EN_EXISTENCIA permanecerán iguales. La instrucción opera solamente en aquellas filas cuyo valor EN_EXISTENCIA original sea menor a 20. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA, EN_PEDIDO =  
CASE  
    WHEN EN_EXISTENCIA < 10 THEN EN_EXISTENCIA * 2  
    WHEN EN_EXISTENCIA BETWEEN 10 AND 15 THEN EN_EXISTENCIA + 3  
    ELSE EN_EXISTENCIA  
END  
FROM DISCOS_COMPACTOS  
WHERE EN_EXISTENCIA < 20;
```

Los resultados de la consulta deberán incluir solamente siete filas, y la columna EN_PEDIDO de los resultados de la consulta deberá contener los valores actualizados.

9. Ahora se ejercitará la expresión de valor CAST. Se consultará la tabla TIPOS_MUSICA, pero se convertirá el tipo de datos de la columna NOMBRE_TIPO en los resultados de la consulta. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT ID_TIPO, CAST(NOMBRE_TIPO AS CHAR(20)) AS CHAR_TIPO  
FROM TIPOS_MUSICA;
```

Los resultados de la consulta deberán arrojar 11 filas. Los resultados deberán incluir una columna CHAR_TIPO que contenga los valores convertidos.

10. Cierre la aplicación de cliente.

Resumen de Pruebe esto

Ahora se deberá sentir mucho más cómodo con las diferentes funciones y expresiones de valor que se revisaron en este capítulo. Recuerde que cada implementación SQL soporta diferentes funciones y expresiones de valor, usualmente muchas más de las que hemos visto aquí. De hecho, en muchos casos las funciones y expresiones de valor que se vieron en este capítulo representan solamente la punta del *iceberg*. Asegúrese de revisar la documentación de su producto para averiguar qué funciones y valores de expresión se soportan y cómo están implementadas. Encontrará que son herramientas muy útiles en una variedad de situaciones y que vale la pena el esfuerzo que puso en este capítulo.

Autoexamen Capítulo 10

1. ¿Qué es una función set?
2. Se está creando una instrucción SELECT que consulta la tabla CDS_ARTISTA. La tabla incluye las columnas NOMBRE_ARTISTA y NOMBRE_CD. Se requiere que la instrucción arroje el número total de filas en la tabla. ¿Cuál función COUNT deberá incluirse en la cláusula SELECT?
 - A COUNT(*)
 - B COUNT(NOMBRE_ARTISTA)
 - C COUNT(NOMBRE_CD)
 - D COUNT(NOMBRE_ARTISTA, NOMBRE_CD)
3. ¿Cuál función set deberá utilizarse para sumar los valores en una columna?
 - A MAX
 - B COUNT
 - C SUM
 - D AVG
4. Las funciones set requieren que los datos estén_____ de alguna manera.
5. ¿Qué son las funciones de valor?
6. Se está utilizando la función SUBSTRING para extraer caracteres de la columna DISCO_COMPACTO de la tabla FECHAS_VENTAS. Se quiere iniciar con el tercer carácter y extraer ocho caracteres. ¿Qué parámetros deberán utilizarse en la función SUBSTRING?
7. Se está utilizando la función LOWER en el valor Past Light de la columna NOMBRE_CD. ¿Qué valor será arrojado?
8. ¿Qué función arroja un valor que represente la fecha y la hora actuales al igual que la información relacionada con UCT?
 - A LOCALTIMESTAMP
 - B CURRENT_DATE
 - C LOCALTIME
 - D CURRENT_TIMESTAMP
9. ¿Cuáles son los cuatro tipos de operadores que se utilizan en una expresión de valor numérica?
10. Se están consultando datos de la tabla RASTREO_CD. Se quiere agregar valores en la columna EN_EXISTENCIA a los valores en la columna EN_PEDIDO. Luego se quiere duplicar los totales de la columna. ¿Cómo se establece la expresión de valor numérica?

- 11.** ¿Cuál expresión de valor se utiliza para establecer una serie de condiciones que modifiquen valores?
- 12.** Se está creando una instrucción SELECT que incluye una expresión de valor CASE. Se requiere que una de las condiciones especifique que cualquier valor EN_PEDIDO mayor a 10 se incremente en 5. ¿Cómo deberá establecerse la cláusula WHEN/THEN?
- 13.** ¿Cuál es la última palabra en una expresión de valor CASE?
- 14.** ¿Qué es la expresión de valor CAST?
- 15.** Se está consultando la columna FECHA_VENTA en la tabla FECHAS_VENTAS. Se requiere convertir los valores a un tipo de datos CHAR(25), y que los datos sean desplegados en la columna CHAR_FECHA en los resultados de la consulta. ¿Cómo se define la expresión de valor CAST?
- 16.** ¿Qué valor especial puede utilizarse para identificar al identificador de usuario de sesión SQL actual?

Capítulo 11

Acceder a múltiples
tablas



Habilidades y conceptos clave

- Realizar operaciones básicas join
 - Unir tablas con nombres de columna compartidos
 - Utilizar el método join de condición
 - Realizar operaciones de unión
-

Un componente importante de cualquier base de datos relacional es la correlación que puede existir entre dos tablas cualesquiera. Esta relación le permite al usuario publicar datos en una tabla con datos en otra tabla. Este tipo de relaciones es particularmente útil cuando se necesita consultar datos relacionados de más de una tabla y se requiere recuperar esos datos de una forma significativa para que las relaciones entre las tablas sean, para todos los propósitos prácticos, invisibles. Un método que soporta SQL:2006 para consultar datos de esta manera es unir las tablas en una instrucción. Una operación *join* es una operación que hace coincidir las filas en una tabla con las filas de manera tal que las columnas de ambas tablas puedan ser colocadas lado a lado en los resultados de la consulta como si éstos vinieran de una sola tabla. SQL define muchos tipos de operaciones join. El tipo de operación que puede utilizarse en cualquier situación dada depende de la implementación SQL (depende de las instrucciones soportadas y cómo pueda ser impactado el rendimiento), cuáles datos se quieren arrojar y cómo han sido definidas las tablas. En este capítulo se analizarán varias operaciones que combinen datos desde múltiples tablas, incluyendo joins y unions, y proporcionando detalles acerca de cómo son implementadas, además de los resultados que se puedan esperar al utilizarlas.

Realizar operaciones básicas join

Uno de los tipos más simples de operaciones *join* para implementar son las operaciones separadas por comas. En este tipo de operación, sólo se necesita proporcionar una lista de tablas (separadas por comas) en la cláusula FROM de la instrucción SELECT. Se puede, por supuesto, cualificar la operación join en la cláusula WHERE (lo cual es necesario para obtener datos significativos desde las tablas), pero no es obligatorio hacerlo. Sin embargo, antes de analizar la cláusula WHERE, demos primero un vistazo a las operaciones join separadas por comas desde sus puntos más básicos.

Supongamos que se quieren desplegar los datos de la tabla INVENTARIO_CD y de la tabla INTERPRETES, mostradas en la figura 11-1. (La figura también incluye la tabla TIPO_INTER, la cual se estará utilizando en la sección “Crear operaciones join con más de dos tablas”). Se pueden visualizar los datos de las tablas INVENTARIO_CD e INTERPRETES consultando cada tabla por separado, o se pueden unir las tablas en una instrucción.

Para unir las dos tablas, se puede crear una instrucción SELECT tan simple como la siguiente:

```
SELECT *  
FROM INVENTARIO_CD, INTERPRETES;
```

INVENTARIO_CD			INTERPRETES			TIPO_INTER	
NOMBRE_CD: VARCHAR(60)	ID_INTER: INT	EN_EXISTENCIA: INT	ID_INTER: INT	NOMBRE_INTER: VARCHAR(60)	ID_TIPO: INT	ID_TIPO: INT	NOMBRE_TIPO: CHAR(20)
Famous Blue Raincoat	102	12	101	Joni Mitchell	10	10	Popular
Blue	101	24	102	Jennifer Warnes	12	11	Blues
Court and Spark	101	17	103	B.B. King	11	12	Folk
Past Light	105	9	104	Bonnie Raitt	10	13	Rock
Fundamental	104	22	105	William Ackerman	15	14	Classical
Blues on the Bayou	103	19	106	Bing Crosby	16	15	New Age
Longing in Their Hearts	104	18	107	Patsy Cline	17	16	Classic Pop
Luck of the Draw	104	25	108	John Barry	18	17	Country
Deuces Wild	103	17	109	Leonard Cohen	12	18	Soundtrack
Nick of Time	104	11					
Both Sides Now	101	13					

Figura 11-1 Uniendo las tablas INVENTARIO_CD, INTERPRETES y TIPO_INTER.

La consulta produce lo que se conoce como una *tabla de producto cartesiano* (nombrada así debido al matemático y filósofo francés René Descartes), que es una lista de cada fila en una tabla, unida con cada una de las filas en la otra tabla, como se muestra (parcialmente) en los siguientes resultados de la consulta:

NOMBRE_CD	ID_INTER	EN_EXISTENCIA	ID_INTER	NOMBRE_INTER	ID_TIPO
-----	-----	-----	-----	-----	-----
Famous Blue Raincoat	102	12	102	Jennifer Warnes	12
Blue	101	24	102	Jennifer Warnes	12
Court and Spark	101	17	102	Jennifer Warnes	12
Past Light	105	9	102	Jennifer Warnes	12
Fundamental	104	22	102	Jennifer Warnes	12
Blues on the Bayou	103	19	102	Jennifer Warnes	12
Longing in Their Hearts	104	18	102	Jennifer Warnes	12
Luck of the Draw	104	25	102	Jennifer Warnes	12
Deuces Wild	103	17	102	Jennifer Warnes	12
Nick of Time	104	11	102	Jennifer Warnes	12
Both Sides Now	101	13	102	Jennifer Warnes	12
Famous Blue Raincoat	102	12	101	Joni Mitchell	10
Blue	101	24	101	Joni Mitchell	10
Court and Spark	101	17	101	Joni Mitchell	10
Past Light	105	9	101	Joni Mitchell	10

En realidad, la instrucción SELECT anterior arrojará muchas más filas de las que se muestran aquí. Estos resultados representan solamente una lista parcial. Debido a que la tabla INVENTARIO_CD contiene 11 filas y la tabla INTERPRETES contiene 9 filas, los resultados completos de la consulta contendrían 99 filas. Demos un vistazo más profundo a esto. La fila Famous Blue

Raincoat en la tabla INVENTARIO_CD ha sido unida con cada fila en la tabla INTERPRETES, lo cual nos da un total de 9 filas. Cada una de las 10 filas restantes en la tabla INVENTARIO_CD se hace coincidir con cada una de las filas en la tabla INTERPRETES de la misma manera. Como resultado, habrá 99 filas ($11 \times 9 = 99$).

Como se puede ver, los resultados de esta consulta no son de gran ayuda. Sin embargo, es posible generar resultados más significativos si se utiliza una cláusula WHERE para crear una operación *equi-join* (también escrita como *equijoin*), que es un tipo de operación join que equipara los valores de una o más columnas en la primera tabla con los valores de una o más columnas correspondientes en la segunda tabla. Como puede uno imaginar por el nombre, el operador de comparación en una operación equi-join es el operador igual a (=). Por ejemplo, se puede cualificar la instrucción SELECT previa, de la siguiente forma:

```
SELECT *
FROM INVENTARIO_CD, INTERPRETES
WHERE INVENTARIO_CD.ID_INTER = INTERPRETES.ID_INTER;
```

Ahora los resultados de la consulta incluirán solamente a aquellas filas en las cuales el valor en la columna ID_INTER de la tabla INVENTARIO_CD coincida con el valor en la columna ID_INTER de la tabla INTERPRETES. Observe que se deben cualificar los nombres de columna agregando los nombres de tabla. Esto debe hacerse cada vez que las columnas de diferentes tablas tengan el mismo nombre. Si se ejecuta esta instrucción, se arrojarán los siguientes resultados de consulta:

NOMBRE_CD	ID_INTER	EN_EXISTENCIA	ID_INTER	NOMBRE_INTER	ID_TIPO
-----	-----	-----	-----	-----	-----
Famous Blue Raincoat	102	12	102	Jennifer Warnes	12
Blue	101	24	101	Joni Mitchell	10
Court and Spark	101	17	101	Joni Mitchell	10
Past Light	105	9	105	William Ackerman	15
Fundamental	104	22	104	Bonnie Raitt	10
Blues on the Bayou	103	19	103	B.B. King	11
Longing in Their Hearts	104	18	104	Bonnie Raitt	10
Luck of the Draw	104	25	104	Bonnie Raitt	10
Deuces Wild	103	17	103	B.B. King	11
Nick of Time	104	11	104	Bonnie Raitt	10
Both Sides Now	101	13	101	Joni Mitchell	10

Los datos arrojados por esta consulta son ahora mucho más significativos. Cada CD coincide con el artista apropiado, y solamente se despliegan 11 filas, en lugar de 99. Sin embargo, incluso estos resultados de la consulta incluyen datos repetitivos (la columna ID_INTER). Además, puede suceder que no sólo se requieran las columnas duplicadas, sino que se requiera desplegar solamente ciertas columnas y quizá cualificar incluso más la condición de la búsqueda.

Modifiquemos la instrucción SELECT sobre la que se ha estado trabajando al especificar los nombres de las columnas en la cláusula SELECT para limitar las columnas arrojadas y agregar otro predicado a la cláusula WHERE y así limitar las filas arrojadas, como se muestra en el siguiente ejemplo:

```
SELECT INVENTARIO_CD.NOMBRE_CD, INTERPRETES.NOMBRE_INTER, INVENTARIO_
CD.EN_EXISTENCIA
FROM INVENTARIO_CD, INTERPRETES
WHERE INVENTARIO_CD.ID_INTER = INTERPRETES.ID_INTER
AND INVENTARIO_CD.EN_EXISTENCIA < 15;
```

En esta instrucción se ha especificado que tres columnas deberán estar incluidas en los resultados de la consulta. Observe que se han cualificado los nombres de las columnas al incluir los nombres de las tablas. Observe también que la cláusula `WHERE` incluye un predicado adicional, conectado con el primer predicado por medio de la palabra clave `AND`. Ahora cualquier fila que sea arrojada deberá también tener valores `EN_EXISTENCIA` menores a 15. Si se ejecuta esta instrucción, se recibirán los siguientes resultados de la consulta:

NOMBRE_CD	NOMBRE_INTER	EN_EXISTENCIA
-----	-----	-----
Famous Blue Raincoat	Jennifer Warnes	12
Both Sides Now	Joni Mitchell	13
Past Light	William Ackerman	9
Nick of Time	Bonnie Raitt	11

Como se puede ver, se han refinado los resultados de la consulta a solamente la información más esencial. Desde luego, se pueden crear todo tipo de consultas, dependiendo de las necesidades, siempre y cuando se sigan los lineamientos básicos para crear una operación `join` separada por comas:

- La cláusula `FROM` debe incluir todos los nombres de las tablas.
- La cláusula `WHERE` deberá definir una condición `join`, evitando un producto cartesiano.
- Las referencias de columna deberán ser cualificadas cuando los nombres de columna estén compartidos entre las tablas.

Aparte de estos lineamientos, existe la libertad de crear cualquier tipo de instrucción `SELECT` que sea necesaria para extraer la información que se requiera de las tablas participantes. Y mientras que utilizar la cláusula `WHERE` para especificar la condición `join` era la forma original de realizar operaciones `join` en SQL, posteriormente en este capítulo se verá que existen otras variaciones de la sintaxis utilizando la palabra clave `JOIN`, que la mayoría de los programadores de SQL prefieren en lugar de la sintaxis original. Pero sin importar qué sintaxis se utilice, siempre tenga en mente que necesita existir algún tipo de conexión lógica entre las tablas. Esta conexión a menudo se aprecia en la forma de una clave externa, pero no resulta obligatorio de esta manera. (Para mayor información acerca de las claves externas, véase el capítulo 4.) Las tablas pueden ser unidas aun si existe o no una clave externa. Y a pesar de que `equi-join` es el más común, puede encontrarse ocasionalmente que resulta muy útil usar un operador de comparación `join` diferente, por ejemplo menor que (`<`) o `BETWEEN`, en cuyo caso el operador `join` es llamado *theta-join*.

Utilizar nombres de correlación

Como se estipuló anteriormente, se deben cualificar las referencias de columna agregando nombres de tabla a aquellas columnas que compartan un nombre. Sin embargo, como una política general, siempre es una buena idea cualificar las referencias de columna cuando se unan tablas, sea necesario o no. Esto vuelve mucho más fácil referenciar el código en algún momento posterior si la instrucción es totalmente autodocumentada. Sin embargo, cuando sus consultas se vuelvan mucho más complejas, se puede volver muy tedioso reingresar los nombres de las tablas cada vez que se hace referencia a una columna. Debido a esto, SQL soporta nombres de correlación, o alias, que pueden ser utilizados durante la duración de una instrucción. Un *nombre de correlación* es simple-

mente una versión más corta del nombre de tabla actual que se utiliza para simplificar el código y hacerlo más legible.

Tomemos, por ejemplo, la última instrucción SELECT que se vio. Se puede moldear esta instrucción utilizando nombres de correlación para ambas tablas:

```
SELECT c.NOMBRE_CD, p.NOMBRE_INTER, c.EN_EXISTENCIA
FROM INVENTARIO_CD AS c, INTERPRETES AS p
WHERE c.ID_INTER = p.ID_INTER
AND c.EN_EXISTENCIA < 15;
```

La instrucción SELECT produce exactamente los mismos resultados que la instrucción anterior, excepto en la cláusula FROM. De hecho, se puede utilizar la cláusula FROM para definir los alias que son utilizados en el resto de la instrucción. En este caso, la tabla INVENTARIO_CD se renombra como c, y la tabla INTERPRETES se renombra como p. Como resultado, c y p deben utilizarse en cualquier otra posición dentro de la instrucción SELECT cuando se haga referencia a esas tablas. Una vez que se haya definido el nombre de la correlación, ya no podrá utilizarse el nombre actual de la tabla. Y esto puede resultar muy confuso debido a que se utiliza el alias en la cláusula SELECT, pero no está aún definido sino hasta la cláusula FROM posterior a la cláusula SELECT. Sin embargo, esto toma sentido si recordamos que la cláusula FROM siempre se procesa primero.

Para comprender mejor cómo funciona el proceso de renombrado, recordemos el tema de cómo son procesadas en las instrucciones SELECT. En el capítulo 7 se estableció que la cláusula FROM se procesa en primer lugar y la cláusula SELECT se procesa en el último. Ésta es la razón por la que los nombres de correlación se definen en la cláusula FROM. Una vez que son definidos, todas las demás cláusulas pueden (y deben) utilizar esos alias cuando se definan las referencias de columna. Los nombres de correlación se utilizan durante todo el resto de la instrucción, pero solamente aplican a la instrucción en la cual están definidos. Si se crea una nueva instrucción SELECT, deben redefinirse esos nombres para poder utilizarlos en la nueva instrucción.

Como se pudo ver en la anterior instrucción SELECT, un nombre de correlación es definido inmediatamente después del nombre actual de la tabla. El nuevo nombre sigue a la palabra clave AS. Sin embargo, la palabra clave AS no es obligatoria. En la mayoría de las implementaciones también puede utilizarse la siguiente convención para renombrar las tablas dentro de una consulta:

```
SELECT c.NOMBRE_CD, p.NOMBRE_INTER, c.EN_EXISTENCIA
FROM INVENTARIO_CD c, INTERPRETES p
WHERE c.ID_INTER = p.ID_INTER
AND c.EN_EXISTENCIA < 15;
```

Observe que sólo se especifica el nuevo nombre, sin la palabra clave AS. Esto hace a la instrucción SQL mucho más sencilla. De hecho, algunas implementaciones como Oracle no permiten utilizar la palabra clave AS en lo absoluto, aun cuando ésta es parte del estándar SQL. Una vez más, esta última instrucción SELECT proporcionará los mismos resultados de la consulta que se vieron en los dos ejemplos anteriores. Sólo que la instrucción en sí misma ha sido cambiada.

Crear operaciones join con más de dos tablas

Hasta este punto, los ejemplos que se han visto han unido solamente dos tablas. Sin embargo, se puede utilizar una operación join separada por comas para desplegar datos de más de dos tablas.

Si nos referimos una vez más a la figura 11-1, se verá que la tabla TIPO_INTER está incluida en la ilustración. Es posible, si se desea, unir las tres tablas en una sola instrucción SELECT, como se muestra en el siguiente ejemplo:

```
SELECT c.NOMBRE_CD, p.NOMBRE_INTER, t.NOMBRE_TIPO
FROM INVENTARIO_CD c, INTERPRETES p, TIPO_INTER t
WHERE c.ID_INTER = p.ID_INTER
      AND p.ID_TIPO = t.ID_TIPO
      AND NOMBRE_TIPO = 'Popular';
```

En esta instrucción, la cláusula FROM incluye todas las tres tablas. Adicionalmente, la cláusula WHERE proporciona dos condiciones equi-join: una que traza las columnas ID_INTER y otra que traza las columnas ID_TIPO. Si se ejecuta esta instrucción, se recibirán los siguientes resultados de la consulta:

NOMBRE_CD	NOMBRE_INTER	NOMBRE_TIPO
-----	-----	-----
Blue	Joni Mitchell	Popular
Court and Spark	Joni Mitchell	Popular
Fundamental	Bonnie Raitt	Popular
Longing in Their Hearts	Bonnie Raitt	Popular
Luck of the Draw	Bonnie Raitt	Popular
Nick of Time	Bonnie Raitt	Popular
Both Sides Now	Joni Mitchell	Popular

Observe que la información de las tres tablas está incluida en los resultados: el nombre del CD, el nombre del artista y la categoría del artista. A pesar de que pudiera existir una relación entre la tabla INVENTARIO_CD y la tabla INTERPRETES, al igual que entre la tabla INTERPRETES y la tabla TIPO_INTER, los resultados de la consulta proporcionan un desplegado uniforme que oculta esas relaciones y muestra solamente la información necesaria.

Crear la operación cross join

Además de la operación join separada por comas, SQL soporta otro tipo de operación llamada cross join. Esta operación es prácticamente idéntica a la operación join separada por comas. La única diferencia es que en lugar de separar los nombres de columna con una coma, se utilizan las palabras clave CROSS JOIN. Por ejemplo, tomemos una instrucción que se utilizó anteriormente y modifiquémosla reemplazándola con las palabras clave CROSS JOIN:

```
SELECT c.NOMBRE_CD, p.NOMBRE_INTER, c.EN_EXISTENCIA
FROM INVENTARIO_CD c CROSS JOIN INTERPRETES p
WHERE c.ID_INTER = p.ID_INTER
      AND c.EN_EXISTENCIA < 15;
```

Esta instrucción arroja tres columnas desde dos tablas, y la cláusula WHERE contiene una condición equi-join. Decidir entre utilizar una o la otra puede simplemente ser una cuestión de determinar cuál instrucción es soportada por la implementación SQL, y, si ambas son soportadas, cuál de ellas proporciona un mejor rendimiento. Con toda probabilidad, se convertirá en un asunto de preferencia personal, con pocas ventajas entre una y otra.

Pregunta al experto

P: Si se están uniendo tablas, parece probable que en algunos casos se arrojarán filas duplicadas en los resultados de la consulta, dependiendo de cómo se construya la instrucción SELECT. ¿Cómo pueden evitarse las filas duplicadas?

R: Al igual que con la mayoría de las consultas, es posible generar filas duplicadas. Por ejemplo, la siguiente instrucción arrojará tipos y nombres de artista duplicados:

```
SELECT P.NOMBRE_INTER, T.NOMBRE_TIPO
FROM INVENTARIO_CD C, INTERPRETES P, TIPO_INTER T
WHERE C.ID_INTER = P.ID_INTER
AND P.ID_TIPO = T.ID_TIPO;
```

Para aquellos artistas que tienen más de un CD, los resultados de la consulta contendrán alguna fila para cada uno de esos CD. Sin embargo, al igual que con cualquier otra instrucción SELECT, puede añadirse la palabra clave DISTINCT a la cláusula SELECT, como se muestran el siguiente ejemplo:

```
SELECT DISTINCT P.NOMBRE_INTER, T.NOMBRE_TIPO
FROM INVENTARIO_CD C, INTERPRETES P, TIPO_INTER T
WHERE C.ID_INTER = P.ID_INTER
AND P.ID_TIPO = T.ID_TIPO;
```

Esta instrucción arrojará menos filas que la instrucción anterior (5 comparadas con 11), y ninguna fila estará duplicada. Observe también que se pueden lograr los mismos resultados sin la palabra clave DISTINCT utilizando una cláusula GROUP BY que enliste ambas columnas.

Crear la operación self-join

Otro tipo de operación join que puede crearse es self-join, que puede ser tanto una operación separada por comas como una operación cross join. En una operación self-join se crea una condición join que se refiere a la misma tabla dos veces, esencialmente uniendo la tabla consigo misma. Esto se realiza casi siempre para resolver una relación recursiva, al encontrar otras filas en la misma tabla que están relacionadas con las filas seleccionadas. Por ejemplo, supongamos que se agrega la tabla EMPLEADOS a la base de datos, como se muestra en la figura 11-2. La tabla EMPLEADOS incluye una lista de números de identificación de empleados, nombres de empleados y los números de identificación de los jefes de los empleados, quienes también están enlistados en la tabla. Por ejemplo, el jefe de Mr. Jones (ID_EMP 102) es Ms. Smith (ID_EMP 101).

Para crear una operación self-join en esta tabla se debe crear una operación join que trate a la tabla como si fueran dos tablas separadas con el mismo nombre, mismas columnas y mismos datos:

```
SELECT a.ID_EMP, a.NOMBRE_EMP, b.NOMBRE_EMP AS ADMINISTRADOR
FROM EMPLEADOS a, EMPLEADOS b
WHERE a.ADMIN = b.ID_EMP
ORDER BY a.ID_EMP;
```

ID_EMP: INT	NOMBRE_EMP: VARCHAR(60)	ADMIN: INT
101	Ms. Smith	NULL
102	Mr. Jones	101
103	Mr. Roberts	101
104	Ms. Hanson	103
105	Mr. Fields	102
106	Ms. Lee	102
107	Mr. Carver	103

Figura 11-2 La tabla EMPLEADOS unida por una operación self-join.

En esta instrucción, a cada instancia de la tabla se le da un nombre de correlación. Como resultado, ahora se tiene (en este ejemplo) una tabla a y una tabla b. Se toman los valores ID_EMP y NOMBRE_EMP de la tabla a, pero se toma el valor ADMINISTRADOR de la tabla b. La condición equi-join se define en la cláusula WHERE al igualar el valor ADMIN en la tabla a con el valor ID_EMP en la tabla b. Esto proporciona el vínculo que trata una tabla física como dos tablas lógicas. Cuando se ejecuta esta instrucción, se reciben los siguientes resultados de la consulta:

ID_EMP	NOMBRE_EMP	ADMINISTRADOR
102	Mr. Jones	Ms. Smith
103	Mr. Roberts	Ms. Smith
104	Ms. Hanson	Mr. Roberts
105	Mr. Fields	Mr. Jones
106	Ms. Lee	Mr. Jones
107	Mr. Carver	Mr. Roberts

Los resultados incluyen el número de identificación de empleado y el número de cada empleado, junto con el nombre del jefe del empleado. Como se puede ver, la operación self-join puede ser una herramienta muy útil en casos como éste en donde una tabla haga referencia a sí misma.

Unir tablas con nombres de columna compartidos

SQL proporciona dos métodos para configurar operaciones join que pueden utilizarse cuando se está trabajando con columnas que tienen los mismos nombres. Estos dos métodos (join natural y join de columna nombrada) permiten fácilmente especificar una condición join entre dos tablas cuando una o más columnas son iguales dentro de esas tablas. Para poder utilizar cualquiera de estos dos métodos, las tablas deben cumplir con las siguientes condiciones:

- Las columnas unidas deberán compartir el mismo nombre y tener tipos de datos compatibles.
- Los nombres de las columnas unidas no pueden ser cualificados con nombres de tabla.

Cuando se está utilizando ya sea el método join natural o un join de columna nombrada, cada tabla debe compartir por lo menos una columna en común. Por ejemplo, las tablas TITULOS_EN_EXISTENCIA y COSTOS_TITULO, mostradas en la figura 11-3, tienen dos columnas que son iguales: TITULO_CD y TIPO_CD. Observe que cada conjunto de columnas a unir está configurado con el mismo tipo de datos.

Se puede utilizar tanto un método join natural o un join de columna nombrada para unir estas dos tablas. Se describen cada uno de estos tipos de operaciones join en varias de las siguientes secciones, y se utilizan las tablas en la figura 11-3 para ilustrar cómo funciona cada uno de estos métodos.

NOTA

No todas las implementaciones de SQL soportan métodos join naturales o métodos join de columna nombrada. Por ejemplo, SQL Server no soporta ninguno de estos métodos, MySQL soporta joins naturales pero no los joins de columna nombrada, y Oracle soporta ambos.

Crear el método join natural

El método join natural hace coincidir automáticamente las filas de aquellas columnas con el mismo nombre. No es necesario especificar ningún tipo de condición equi-join para los joins naturales. La implementación SQL determina cuáles columnas tienen los mismos nombres y luego intenta hacerlas coincidir. El inconveniente de hacer esto es que no se puede especificar cuáles columnas son comparadas, aunque sí se puede especificar cuáles columnas son incluidas en los resultados de la consulta.

En el siguiente ejemplo se utiliza un join natural para unir la tabla TITULOS_EN_EXISTENCIA con la tabla COSTOS_TITULO:

```
SELECT TITULO_CD, TIPO_CD, c.MENUDEO
FROM TITULOS_EN_EXISTENCIA s NATURAL JOIN COSTOS_TITULO c
WHERE s.INVENTARIO > 15;
```

TITULOS_EN_EXISTENCIA

TITULO_CD: VARCHAR(60)	TIPO_CD: CHAR(20)	INVENTARIO: INT
Famous Blue Raincoat	Folk	12
Blue	Popular	24
Past Light	New Age	9
Blues on the Bayou	Blues	19
Luck of the Draw	Popular	25
Deuces Wild	Blues	17
Nick of Time	Popular	11
Both Sides Now	Popular	13

COSTOS_TITULO

TITULO_CD: VARCHAR(60)	TIPO_CD: CHAR(20)	MAYOREO: NUMERIC(5,2)	MENUDEO: NUMERIC(5,2)
Famous Blue Raincoat	Folk	8.00	16.99
Blue	Popular	7.50	15.99
Court and Spark	Popular	7.95	15.99
Past Light	New Age	6.00	14.99
Fundamental	Popular	8.25	16.99
Blues on the Bayou	Blues	7.25	15.99
Longing in their Hearts	Popular	7.50	15.99
Deuces Wild	Blues	7.45	14.99
Nick of Time	Popular	6.95	14.99

Figura 11-3 Unir las tablas TITULOS_EN_EXISTENCIA y COSTOS_TITULO.

En esta instrucción, las tablas están unidas mediante las columnas TITULO_CD y TIPO_CD. Observe que ningún nombre de columna está cualificado (los nombres cualificados no se permiten en operaciones join naturales). Si cualquiera de estos nombres de columna hubiera sido incluido en la cláusula WHERE, tampoco estaría cualificado de esta manera. Cuando se ejecute esta instrucción, se recibirán los siguientes resultados de consulta:

TITULO_CD	TIPO_CD	MENUDEO
-----	-----	-----
Blues on the Bayou	Blues	15.99
Deuces Wild	Blues	14.99
Blue	Popular	15.99

Como se puede ver, solamente son arrojadas tres filas. Existen filas en las cuales los valores de TITULO_CD en ambas tablas son iguales y los valores de TIPO_CD son iguales. Adicionalmente, los valores INVENTARIO son mayores a 15.

Crear el método join de columna nombrada

A pesar de que las operaciones join naturales pueden ser de mucha utilidad para operaciones join simples, encontrará que no siempre se quiere incluir cada columna coincidente como parte de la condición join. La forma de evitar eso es utilizando una operación *join de columna nombrada*, que permite especificar cuáles columnas coincidentes serán agregadas. Por ejemplo, supongamos que se quiere incluir solamente TITULO_CD en la condición join. Se puede modificar el ejemplo anterior de esta manera:

```
SELECT TITULO_CD, s.TIPO_CD, c.MENUDEO
  FROM TITULOS_EN_EXISTENCIA s JOIN COSTOS_TITULO c
    USING (TITULO_CD)
 WHERE s.Inventario > 15;
```

En esta instrucción se ha removido la palabra clave NATURAL y se ha agregado una cláusula USING, que identifica las columnas coincidentes. Observe que el nombre de columna TIPO_CD ahora ha sido cualificado, pero no así la columna TITULO_CD. Solamente las columnas identificadas en la cláusula USING están sin cualificar. Esta instrucción arroja los mismos resultados que el ejemplo anterior, aunque no necesariamente será siempre el caso, y dependerá de los datos en las tablas. Sin embargo, si se incluyen ambas columnas coincidentes en la cláusula USING, definitivamente se obtendrían los mismos resultados que se vieron en el join natural. Al identificar todas las columnas coincidentes en la cláusula USING, se está realizando la misma función que con un join natural.

Utilizar el método join de condición

Hasta ahora en este capítulo se han visto las operaciones join separadas por coma, las cross joins, joins naturales y los joins de columna nombrada. En las operaciones join separadas por coma y cross joint, la condición equi-join es definida en la cláusula WHERE. En las operaciones join naturales, la condición equi-join es asumida automáticamente en todas las columnas coincidentes. Y en las join de columna nombrada, la condición equi-join está localizada en cualquier columna coincidente definida en la cláusula USING. El *join de condición* realiza un método diferente a cualquiera

de éstos. En un join de condición, la condición equi-join está definida en la cláusula ON, que funciona de manera muy similar a la cláusula WHERE. Sin embargo, a pesar de utilizar la cláusula ON, una condición básica join es similar de muchas maneras a las operaciones join previas que se han visto, excepto que, a diferencia de las join naturales y de las join de columna nombrada, la condición join permite hacer coincidir cualquier columna compatible de una tabla con cualquier otra de otra tabla. Los nombres de columna no necesitan ser iguales. La join de condición es la sintaxis preferida por la mayoría de los programadores SQL debido a su claridad, flexibilidad y amplio soporte entre todas las implementaciones SQL.

Una join de condición puede ser separada en dos tipos de uniones: *inner joins* y *outer joins*. La diferencia entre estas dos uniones es la cantidad de datos arrojados por la consulta. Una inner join arroja solamente aquellas filas que coinciden con la condición equi-join definida en la instrucción SELECT. En otras palabras, la inner join arroja solamente filas coincidentes. Ésta era la join original disponible en SQL, y por lo tanto algunos programadores la llaman “join estándar”, a pesar de que esto es un error debido a que todas las joins presentadas en este capítulo están descritas en el estándar SQL. Una outer join, por otro lado, arroja las filas coincidentes y alguna o todas las filas no coincidentes, dependiendo del tipo de outer join.

NOTA

De acuerdo con el estándar SQL:2006, tanto las operaciones join naturales como las join de columna nombrada soportan joins inner y outer. Sin embargo, esto puede variar entre las diferentes implementaciones SQL, por lo que debe asegurarse de revisar la documentación del producto. De manera predeterminada, una join se procesa como una inner join, a menos que esté específicamente definida como una outer join.

Crear la inner join

Ahora que se tiene una idea general de la condición join, demos un vistazo más de cerca a la inner join. Ésta es la más común de las condiciones join y está especificada al utilizar las palabras clave INNER JOIN. Sin embargo, la palabra clave INNER no es necesaria. Si se utiliza JOIN por sí sola, se asume que es una inner join. Además de la palabra clave JOIN (especificada en la cláusula FROM), también se puede definir una cláusula ON, que se coloca inmediatamente después de la cláusula FROM. Echemos un vistazo a un ejemplo para ver cómo funciona esto.

Supongamos que se quiere unir las tablas TITULO_CDS y ARTISTAS_TITULOS, mostradas en la figura 11-4. En el siguiente ejemplo se ha creado una inner join que está basada en las columnas ID_TITULO en las dos tablas:

```
SELECT t.TITULO, ta.ID_ARTISTA
FROM TITULO_CDS t INNER JOIN ARTISTAS_TITULOS ta
    ON t.ID_TITULO = ta.ID_TITULO
WHERE t.TITULO LIKE ('%Blue%');
```

La instrucción utiliza las palabras clave INNER JOIN para unir las tablas TITULO_CDS y ARTISTAS_TITULOS. La condición equi-join se define en la cláusula ON, utilizando la columna TITLE_ID en cada tabla. Observe que los nombres de correlación han sido definidos en ambas tablas. La instrucción SELECT es cualificada adelante por la cláusula WHERE, que arroja solamente aquellas filas que contienen el valor Blue en la columna TITLE de la tabla TITULO_CDS. Cuando se ejecute esta consulta, se recibirán los siguientes resultados de consulta:

TITULO_CDS		ARTISTAS_TITULOS		ARTISTAS_CD	
ID_TITULO: INT	TITULO: VARCHAR(60)	TITLE_ID: INT	ID_ARTISTA: INT	ID_ARTISTA: INT	ARTISTA: VARCHAR(60)
101	Famous Blue Raincoat	101	2001	2001	Jennifer Warnes
102	Blue	102	2002	2002	Joni Mitchell
103	Court and Spark	103	2002	2003	William Ackerman
104	Past Light	104	2003	2004	Kitaro
105	Kojiki	105	2004	2005	Bing Crosby
106	That Christmas Feeling	106	2005	2006	Patsy Cline
107	Patsy Cline: 12 Greatest Hits	107	2006	2007	Jose Carreras
108	Carreras Domingo Pavarotti in Concert	108	2007	2008	Luciano Pavarotti
109	Out of Africa	108	2008	2009	Placido Domingo
110	Leonard Cohen The Best of	108	2009	2010	John Barry
111	Fundamental	109	2010	2011	Leonard Cohen
112	Blues on the Bayou	110	2011	2012	Bonnie Raitt
113	Orlando	111	2012	2013	B.B. King
		112	2013	2014	David Motion
		113	2014	2015	Sally Potter
		113	2015		

Figura 11-4 Unir las tablas TITULO_CDS, TITLES_ARTISTS y CD_ARTISTS.

TITULO	ID_ARTISTA
-----	-----
Famous Blue Raincoat	2001
Blue	2002
Blues on the Bayou	2013

Como se puede ver, los resultados incluyen información de ambas tablas: la columna TITULO de la tabla TITULO_CDS en la columna ID_ARTISTA de la tabla ARTISTAS_TITULOS. A pesar de que esta información puede ser muy útil, podría resultar mejor para algunos usuarios si ellos pudieran ver los nombres reales de los artistas en lugar de números. La forma de lograr esto es incluir una tercera tabla en la unión.

Volvamos al ejemplo anterior y agreguemos una segunda condición join a la tabla ARTISTAS_CD (mostrada en la figura 11-4). En el siguiente ejemplo se agrega la segunda condición inmediatamente después de la cláusula original ON:

```
SELECT t.TITULO, a.ARTISTA
FROM TITULO_CDS t INNER JOIN ARTISTAS_TITULOS ta
    ON t.ID_TITULO = ta.ID_TITULO
    INNER JOIN ARTISTAS_CD a
    ON ta.ID_ARTISTA = a.ID_ARTISTA
WHERE t.TITULO LIKE ('%Blue%');
```

Observe que se repiten las palabras clave INNER JOIN, seguidas del nombre de la tercera tabla, la cual entonces sigue a la otra cláusula ON. En esta cláusula, la condición equi-join se define en las columnas ID_ARTISTA dentro de las tablas ARTISTAS_TITULOS y ARTISTAS_CD. Tenga en mente que no es necesario incluir la palabra clave INNER, ni tampoco las columnas especificadas en la cláusula ON necesitan tener el mismo nombre.

Si se ejecuta esta instrucción, se obtendrán los siguientes resultados de la consulta:

TITULO	ARTISTA
-----	-----
Famous Blue Raincoat	Jennifer Warnes
Blue	Joni Mitchell
Blues on the Bayou	B.B. King

Observe que los nombres de artista ahora están listados en los resultados. Observe también que el hecho de que se hayan utilizado tres tablas para recuperar esta información es invisible a cualquier persona que vea los resultados de la consulta.

Crear la outer join

Como se mencionó anteriormente en este capítulo, una operación outer join arroja todas las filas coincidentes y alguna o todas las filas no coincidentes, dependiendo del tipo de outer join que se cree. SQL soporta tres tipos de outer joins:

- **Left** Arroja todas las filas coincidentes y todas las filas no coincidentes de la tabla de la izquierda (la tabla a la izquierda de la palabra clave JOIN).
- **Right** Arroja todas las filas coincidentes y todas las filas no coincidentes de la tabla de la derecha (la tabla a la derecha de la palabra clave JOIN).
- **Full** Arroja todas las filas coincidentes y todas las filas no coincidentes de ambas tablas.

NOTA

Debido a que es una característica relativamente nueva, pocas implementaciones SQL soportan actualmente las operaciones full outer join (Oracle y SQL Server pueden hacerlo, pero no MySQL).

Una operación outer join sigue la misma sintaxis que una operación inner join, sólo que en lugar de utilizar las palabras clave INNER JOIN (o solamente la palabra clave JOIN), se aplica LEFT OUTER JOIN, RIGHT OUTER JOIN o FULL OUTER JOIN. Observe que la palabra clave OUTER es opcional. Por ejemplo, se puede especificar LEFT JOIN en lugar de LEFT OUTER JOIN.

La mejor forma de ilustrar las diferencias entre los tipos de outer joins es mostrar diferentes ejemplos de resultados de consulta para cada tipo. Para ilustrar las diferencias, se utilizan las tablas INFO_CD y CD_TIPO, mostradas en la figura 11-5.

INFO_CD

TITULO: VARCHAR(60)	ID_TIPO: CHAR(4)	EXISTENCIA: INT
Famous Blue Raincoat	FROK	19
Blue	CPOP	28
Past Light	NEWA	6
Out of Africa	STRK	8
Fundamental	NPOP	10
Blues on the Bayou	BLUS	11

TIPO_CD

ID_TIPO: CHAR(4)	NOMBRE_TIPO: CHAR(20)
FROK	Folk Rock
CPOP	Classic Pop
NEWA	New Age
CTRY	Country
STRK	Soundtrack
BLUS	Blues
JAZZ	Jazz

Figura 11-5 Unir las tablas INFO_CD y TIPO_CD.

En el primer ejemplo, se define una operación inner join en las dos tablas, solamente para mostrar cómo lucirían normalmente los resultados de la consulta:

```
SELECT i.TITULO, t.NOMBRE_TIPO, i.EXISTENCIA
FROM INFO_CD i JOIN TIPO_CD t
ON i.ID_TIPO = t.ID_TIPO;
```

Esta instrucción arroja los siguientes resultados de la consulta:

TITULO	NOMBRE_TIPO	EXISTENCIA
-----	-----	-----
Famous Blue Raincoat	Folk Rock	19
Blue	Classic Pop	28
Past Light	New Age	6
Out of Africa	Soundtrack	8
Blues on the Bayou	Blues	11

En la mayoría de los casos la operación inner join proporcionará toda la información necesaria. Pero supongamos que se quieren incluir las filas no coincidentes de la tabla INFO_CD. En este caso, se deberá crear una operación left outer join, como se muestra en el siguiente ejemplo:

```
SELECT i.TITULO, t.NOMBRE_TIPO, i.EXISTENCIA
FROM INFO_CD i LEFT OUTER JOIN TIPO_CD t
ON i.ID_TIPO = t.ID_TIPO;
```

Observe que se ha reemplazado JOIN (por INNER JOIN) con LEFT OUTER JOIN. Como se mencionó anteriormente, se puede omitir la palabra clave OUTER en la mayoría de las implementaciones. Si se ejecuta esta instrucción, se recibirán los siguientes resultados de la consulta:

TITULO	NOMBRE_TIPO	EXISTENCIA
-----	-----	-----
Famous Blue Raincoat	Folk Rock	19
Blue	Classic Pop	28
Past Light	New Age	6
Out of Africa	Soundtrack	8
Fundamental	NULL	10
Blues on the Bayou	Blues	11

Como podrá haber notado, la fila Fundamental está ahora incluida en los resultados de la consulta. A pesar de que esta fila no incluye columnas coincidentes, está aún incluida en los resultados de la consulta debido a que es parte de la tabla de la izquierda (*left*). Para esta fila, a la columna NOMBRE_TIPO se le asigna un valor nulo debido a que ningún valor lógico puede ser arrojado para esta columna. El valor nulo sirve como un marcador de posición.

También es posible arrojar las filas no coincidentes de la tabla TIPO_CD, que es la tabla a la derecha de la palabra clave JOIN:

```
SELECT i.TITULO, t.NOMBRE_TIPO, i.EXISTENCIA
FROM INFO_CD i RIGHT OUTER JOIN TIPO_CD t
ON i.ID_TIPO = t.ID_TIPO;
```

Esta instrucción es prácticamente la misma que la instrucción anterior, excepto que ahora se ha especificado RIGHT. La instrucción arroja los siguientes resultados de la consulta:

TITULO	NOMBRE_TIPO	EXISTENCIA
-----	-----	-----
Famous Blue Raincoat	Folk Rock	19
Blue	Classic Pop	28
Past Light	New Age	6
NULL	Country	NULL
Out of Africa	Soundtrack	8
Blues on the Bayou	Blues	11
NULL	Jazz	NULL

Esta vez las columnas no coincidentes de la tabla de la *derecha* se incluyen en los resultados, y los valores nulos se muestran para las columnas TITLE y STOCK.

Si se quiere arrojar todas las filas no coincidentes, se necesitaría modificar la instrucción para definir una operación full outer join:

```
SELECT i.TITULO, t.NOMBRE_TIPO, i.EXISTENCIA
FROM INFO_CD i FULL OUTER JOIN TIPO_CD t
ON i.ID_TIPO = t.ID_TIPO;
```

Esta instrucción arrojará los siguientes resultados de la consulta:

TITULO	NOMBRE_TIPO	EXISTENCIA
-----	-----	-----
Famous Blue Raincoat	Folk Rock	19
Blue	Classic Pop	28
Past Light	New Age	6
Out of Africa	Soundtrack	8
Fundamental	NULL	10
Blues on the Bayou	Blues	11
NULL	Jazz	NULL
NULL	Country	NULL

Como se puede ver, todas las filas coincidentes y no coincidentes se incluyen en los resultados de la consulta. Observe que están incluidas todas las seis filas de la tabla CD_INFO y todas las siete filas de la tabla TIPO_CD.

Realizar operaciones de unión

SQL proporciona un método más para combinar datos desde diferentes tablas de una forma que resulta un poco diferente de las operaciones join mostradas anteriormente en este capítulo. El operador UNION es un método que puede utilizarse para combinar los resultados de múltiples instrucciones SELECT en un solo conjunto de resultados, esencialmente uniendo filas de una consulta con filas de otra. A diferencia de las operaciones join, que agregan columnas desde múltiples tablas y las colocan lado a lado, las operaciones de unión agregan filas al final del conjunto de resultados. Para poder utilizar un operador UNION, cada instrucción SELECT debe producir columnas *compatibles con unión*, lo que significa que cada una debe producir el mismo número de columnas, y las columnas correspondientes deben tener tipos de datos compatibles. Por ejemplo, si la primera columna de la instrucción SELECT produce una columna de caracteres, entonces otras instrucciones SELECT combinadas con ella que utilizan el operador UNION deberán tener un tipo de datos de caracteres en la primera columna en lugar de un tipo de datos numérico o de fecha y hora.

Demos un vistazo a un ejemplo para mostrar lo que esto significa. La figura 11-6 muestra dos tablas: la tabla CDS_CONTINUADOS y la tabla CDS_DESCONTINUADOS. Las tablas son prácticamente idénticas en estructura pero sirven para dos propósitos muy diferentes, que deberán ser obvios si observamos los nombres de las tablas.

Supongamos que se quiere combinar los datos en estas dos tablas para que pueda visualizarse la información de las dos tablas. Es posible, desde luego, ejecutar dos instrucciones SELECT separadas, o se pueden combinar esas instrucciones en una sola que combine la información, como se muestra en el siguiente ejemplo:

```
SELECT *
  FROM CDS_CONTINUADOS
UNION
SELECT *
  FROM CDS_DESCONTINUADOS;
```


CDS_CONTINUADOS

NOMBRE_CD: VARCHAR(60)	TIPO_CD: CHAR(4)	EN_EXISTENCIA: INT
Famous Blue Raincoat	FROK	19
Blue	CPOP	28
Past Light	NEWA	6
Out of Africa	STRK	8
Fundamental	NPOP	10
Blues on the Bayou	BLUS	11

CDS_DESCONTINUADOS

NOMBRE_CD: VARCHAR(60)	TIPO_CD: CHAR(4)	EN_EXISTENCIA: INT
Court and Spark	FROK	3
Kojiki	NEWA	2
That Christmas Feeling	XMAS	2
Patsy Cline: 12 Greatest Hits	CTRY	4
Leonard Cohen The Best of	FROK	3
Orlando	STRK	1

Figura 11-6 Unir las tablas CDS_CONTINUADOS y CDS_DESCONTINUADOS.

Como se puede ver, las dos instrucciones SELECT están combinadas utilizando el operador UNION. Si se ejecuta esta instrucción, se recibirán los siguientes resultados:

NOMBRE_CD	TIPO_CD	EN_EXISTENCIA
-----	-----	-----
Blue	CPOP	28
Blues on the Bayou	BLUS	11
Court and Spark	FROK	3
Famous Blue Raincoat	FROK	19
Fundamental	NPOP	10
Kojiki	NEWA	2
Leonard Cohen The Best Of	FROK	3
Orlando	STRK	1
Out of Africa	STRK	8
Past Light	NEWA	6
Patsy Cline: 12 Greatest Hits	CTRY	4
That Christmas Feeling	XMAS	2

Los resultados incluyen 12 filas de datos, 6 filas para cada tabla. Se pueden limitar los resultados incluso más al especificar condiciones de búsqueda en las cláusulas WHERE. También es posible especificar que la búsqueda solamente arroje columnas específicas, como se muestra en la siguiente instrucción:

```
SELECT TIPO_CD
FROM CDS_CONTINUADOS
UNION
SELECT TIPO_CD
FROM CDS_DESCONTINUADOS;
```

Ahora cuando se genere la consulta, solamente serán desplegados los valores de la columna TIPO_CD:

```
TIPO_CD
-----
BLUS
CPOP
CTRY
FROK
NEWA
NPOP
STRK
XMAS
```

Observe que sólo son arrojadas 8 filas en lugar de 12. Esto se debe a que de manera predeterminada se filtran las filas para evitar duplicados. Si se quieren incluir todas las filas en los resultados de la consulta, sin importar la existencia de valores duplicados, se puede agregar la palabra clave ALL después del operador UNION, como se muestra en el siguiente ejemplo:

```
SELECT TIPO_CD
      FROM CDS_CONTINUADOS
UNION ALL
SELECT TIPO_CD
      FROM CDS_DESCONTINUADOS;
```

Esta instrucción arrojará 12 filas en lugar de 8, con varios valores duplicados.

Como se puede ver, el operador UNION solamente resulta útil en casos muy específicos. Si se requiere mayor control sobre los resultados de la consulta, se deberá utilizar uno de los diferentes tipos de operadores join soportados por SQL.

Pregunta al experto

P: ¿Hay algún tipo de operador join que sea comparable a utilizar el operador UNION?

R: SQL:2006 soporta actualmente un operador union join que realiza muchas de las mismas funciones que el operador UNION. El operador union join es similar al operador full outer join, en términos de cómo son consolidados los resultados de la consulta. Sin embargo, el operador full outer join permite especificar (en la cláusula ON) cuáles columnas se harán coincidir, a diferencia del operador union join. Además, el operador union join no ha sido implementado de manera general en los RDBMS, y ha sido desaprobado en el estándar SQL:2006, lo que significa que es un candidato a eliminarse de futuras versiones de SQL. Así que para cualquier propósito práctico, el operador union join no es algo que deba preocuparle.

Adicionalmente, el estándar SQL:2006 soporta los operadores INTERSECT y EXCEPT, que tienen una sintaxis similar a UNION. INTERSECT funciona de forma parecida a UNION, excepto que arroja solamente las filas que aparecen en los resultados de ambas instrucciones SELECT. EXCEPT, por otro lado, arroja solamente filas que aparezcan en los resultados de la primera instrucción SELECT pero no en los resultados de la segunda. SQL Server soporta tanto INTERSECT como EXCEPT. Oracle y MySQL soportan INTERSECT pero utilizan el operador MINUS en lugar de EXCEPT.

Pruebe esto 11-1 Consultar múltiples tablas

En este capítulo se han introducido gran variedad de operaciones join así como el operador UNION, el cual técnicamente no está considerado como una operación join. Ahora tendrá la oportunidad de practicar varias de estas técnicas join para consultar datos desde la base de datos INVENTARIO. Específicamente, consultará algunas de las tablas que están configuradas con relaciones de clave externa, que son el tipo de relaciones que ligan datos de una tabla con datos en otra tabla. Debido a que no se cambiará ninguno de los datos, siéntase libre de intentar diferentes tipos de operaciones join, incluso más allá de las que se vieron en este ejercicio. Puede descargar el archivo Try_This_11.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. El primer tipo de operación a realizar será una operación join separada por comas en las tablas ARTISTAS y CDS_ARTISTA. La operación join utilizará la columna ID_ARTISTA para establecer la condición equi-join. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT * FROM ARTISTAS a, CDS_ARTISTA c
WHERE a.ID_ARTISTA = c.ID_ARTISTA;
```

Los resultados de la consulta deberán incluir 19 filas, y también las columnas ID_ARTISTA de ambas tablas al igual que las columnas NOMBRE_ARTISTA, LUGAR_DE_NACIMIENTO e ID_DISCO_COMPACTO.

3. Ahora se modificará la instrucción anterior para que también pueda unir la tabla DISCOS_COMPACTOS. De esa manera, se puede desplegar el nombre real de los CD. Adicionalmente, se especifican los nombres de las columnas que deberán ser arrojadas. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT d.TITULO_CD, a.NOMBRE_ARTISTA, a.LUGAR_DE_NACIMIENTO
FROM ARTISTAS a, CDS_ARTISTA c, DISCOS_COMPACTOS d
WHERE a.ID_ARTISTA = c.ID_ARTISTA
AND d.ID_DISCO_COMPACTO = c.ID_DISCO_COMPACTO;
```

Los resultados de la consulta deberán incluir una vez más 19 filas. Sin embargo, esta vez los resultados desplegarán solamente las columnas TITULO_CD, NOMBRE_ARTISTA y LUGAR_DE_NACIMIENTO.

- 4.** Ahora convirtamos la anterior instrucción SELECT en una operación cross join. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT d.TITULO_CD, a.NOMBRE_ARTISTA, a.LUGAR_DE_NACIMIENTO
FROM ARTISTAS a CROSS JOIN CDS_ARTISTA c CROSS JOIN DISCOS_COMPACTOS d
WHERE a.ID_ARTISTA          = c.ID_ARTISTA
      AND d.ID_DISCO_COMPACTO = c.ID_DISCO_COMPACTO;
```

Se recibirán los mismos resultados de la consulta que se arrojaron en la anterior instrucción SELECT.

- 5.** El siguiente tipo de instrucción que deberá ejercitar es una condición join. La primer condición será del tipo inner join. En esta instrucción se unirán tres tablas: DISCOS_COMPACTOS, TIPOS_DISCO_COMPACTO y TIPOS_MUSICA. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT d.TITULO_CD, t.NOMBRE_TIPO
FROM DISCOS_COMPACTOS d JOIN TIPOS_DISCO_COMPACTO dt
      ON d.ID_DISCO_COMPACTO = dt.ID_DISCO_COMPACTO
      JOIN TIPOS_MUSICA t
      ON dt.ID_TIPO_MUSICA = t.ID_TIPO;
```

Los resultados de la consulta deberán incluir 24 filas. Solamente las columnas TITULO_CD y NOMBRE_TIPO deberán ser desplegadas.

- 6.** Ahora modifiquemos la anterior instrucción SELECT para crear una operación full outer join en ambas condiciones join. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT d.TITULO_CD, t.NOMBRE_TIPO
FROM DISCOS_COMPACTOS d FULL JOIN TIPOS_DISCO_COMPACTO dt
      ON d.ID_DISCO_COMPACTO = dt.ID_DISCO_COMPACTO
      FULL JOIN TIPOS_MUSICA t
      ON dt.ID_TIPO_MUSICA = t.ID_TIPO;
```

Los resultados de la consulta deberán ahora incluir 26 filas en lugar de 24. Esto se debe a que la tabla TIPOS_MUSICA incluye dos filas que son no coincidentes: la fila Jazz y la fila Internacional. En otras palabras, ningún CD coincide con ninguno de estos dos tipos de música.

- 7.** Cierre la aplicación cliente.

Resumen de Pruebe esto

En este ejercicio se crearon operaciones join separadas por comas, operaciones cross y operaciones de condición. Las operaciones join de condición incluían inner y outer joins. Como se puede ver, las operaciones join proporcionan gran flexibilidad cuando se consultan datos desde las tablas en la base de datos. Sin embargo, no son la única solución cuando se necesitan datos de más de una tabla. Una subconsulta a menudo proporcionará la misma funcionalidad que una operación join. En el capítulo 12 se analizarán las subconsultas con gran detalle. Como se verá, éstas proporcionan una forma más para acceder a datos desde múltiples tablas.



Autoexamen Capítulo 11

1. Se está utilizando una operación join separada por comas para unir dos tablas. La primera tabla contiene cinco filas y la segunda tres filas. ¿Cuántas filas contendrá la tabla de producto cartesiano?
2. ¿Qué constituye una condición equi-join en una cláusula WHERE?
3. ¿Cuál cláusula contiene la condición equi-join en una operación join separada por comas?
4. ¿Qué lineamientos básicos deberán seguirse cuando se crea una operación join separada por comas?
5. Se está creando una operación join sobre dos tablas. Se asignan nombres de correlación para cada una de estas tablas. ¿Cuáles nombres deberán utilizarse en la cláusula SELECT: los nombres de correlación o los nombres reales de las tablas?
6. ¿Qué tipo de operación join es prácticamente idéntica a la operación join separada por comas?
 - A Join de condición
 - B Join natural
 - C Cross join
 - D Join de columna nombrada
7. ¿Cuántas tablas están contenidas en una operación self-join?
8. ¿Qué lineamientos deberán seguirse cuando se crean operaciones join naturales o de columna nombrada?
9. ¿Cuál es la diferencia entre una operación join natural y una de columna nombrada?
10. ¿Qué tipo de operación join contiene una cláusula USING para especificar la condición equi-join?
11. ¿Cuáles son los dos tipos de operaciones join de condición?
12. ¿Cuáles son los tres tipos de operación outer join?
13. ¿Cuál tipo de operación join de condición deberá utilizarse si se quieren arrojar solamente filas coincidentes?
 - A Inner join
 - B Left outer join
 - C Right outer join
 - D Full outer join
14. ¿Qué tipo de operación join de condición arroja todas las filas coincidentes y no coincidentes?
 - A Inner join
 - B Left outer join
 - C Right outer join
 - D Full outer join

- 15.** ¿Qué tipo de operación join contiene una cláusula ON?
- A** Cross join
 - B** Join separada por comas
 - C** Join natural
 - D** Join de condición
- 16.** Un operador _____ permite combinar instrucciones SELECT separadas en una sola instrucción para unir los datos en un solo resultado de consulta.
- 17.** ¿Qué palabra clave puede utilizarse con un operador UNION para arrojar todas las filas en los resultados de la consulta, sin importar si existen valores duplicados?

Capítulo 12

Utilizar subconsultas
para acceder
y modificar datos

Habilidades y conceptos clave

- Crear subconsultas que arrojen múltiples filas
 - Crear subconsultas que arrojen un solo valor
 - Trabajar con subconsultas correlacionadas
 - Utilizar subconsultas anidadas
 - Utilizar subconsultas para modificar datos
-

Las subconsultas, al igual que las operaciones join, proporcionan una forma de acceder a datos en múltiples tablas con una sola consulta. Una subconsulta puede agregarse a una instrucción SELECT, INSERT, UPDATE o DELETE para permitir a esa instrucción utilizar los resultados de la consulta arrojados por la subconsulta. La subconsulta es esencialmente una instrucción SELECT incrustada que actúa como una puerta de entrada a los datos en una segunda tabla. Los datos arrojados por la subconsulta se utilizan como la instrucción primaria para cumplir cualquier condición que haya sido definida en la instrucción. En este capítulo se discutirá cómo se utilizan las subconsultas en diferentes instrucciones, particularmente en instrucciones SELECT, y se proporcionarán ejemplos que demuestren cómo crear subconsultas y qué tipo de resultados de consulta esperar.

Crear subconsultas que arrojen múltiples filas

En el capítulo 9 se incluyeron muchos ejemplos de subconsultas que se utilizan para demostrar ciertos tipos de predicados, por ejemplo IN y EXISTS. De varias maneras, este capítulo es una extensión de esa discusión debido a la forma en la cual las subconsultas son implementadas más comúnmente (en la cláusula WHERE de una instrucción SELECT). La comprensión sobre este tipo de subconsultas va de la mano con la comprensión sobre cómo se formulan los predicados para crear condiciones específicas de búsqueda; condiciones de búsqueda que dependen de esas subconsultas para arrojar datos desde una tabla referenciada.

Se pueden dividir las subconsultas de una cláusula WHERE en dos categorías generales: aquellas que pueden arrojar múltiples filas y aquellas que pueden arrojar solamente un valor. En esta sección se analizará la primera de estas categorías. En la siguiente sección, “Crear subconsultas que arrojan un solo valor”, se analizará la segunda categoría. Mientras se expande cada tema, sin duda usted reconocerá los formatos de instrucción desde la discusión de los predicados. A pesar de que esta información pudiera parecer un poco repetitiva (es por eso que se tocará de manera breve), se presenta aquí no solamente para proporcionar un repaso cohesivo de las subconsultas, sino también para proporcionar una perspectiva diferente. En otras palabras, en lugar de mirar las subconsultas a través de la perspectiva del predicado, miraremos directamente a la subconsulta en sí.

A pesar del hecho de que la discusión se enfoca en las subconsultas que son implementadas a través de la cláusula `WHERE`, el uso de las subconsultas no se limita a esa cláusula. De hecho, se pueden incluir subconsultas en una cláusula `SELECT` o en una cláusula `HAVING`. Sin embargo, utilizar subconsultas en una cláusula `SELECT` no es muy común. Además, solamente se utilizarían subconsultas en una cláusula `HAVING` cuando se definan condiciones de búsqueda en datos agrupados. Con todo esto, los principios para utilizar subconsultas en una cláusula `HAVING` son similares a utilizarlas en una cláusula `WHERE`. Por estas razones, la discusión aquí se enfocará en utilizar subconsultas en la cláusula `WHERE`. Conforme usted se convierta en un programador SQL más avanzado, muy probablemente querrá utilizar subconsultas en otros lugares dentro de la instrucción `SELECT`.

Utilizar el predicado IN

El primer tipo de subconsulta que se verá es el tipo utilizado dentro del predicado `IN`. Como podrá recordar del capítulo 9, el predicado `IN` compara valores de una columna en la tabla primaria con valores arrojados por la subconsulta. Si el valor de la columna se encuentra en los resultados de la subconsulta, esa fila (de la tabla primaria) se arroja en los resultados de la consulta de la instrucción `SELECT`. Por ejemplo, supongamos que se quieren consultar los datos de la tabla `EXISTENCIA_CD`, mostrada en la figura 12-1.

EXISTENCIA_CD

TITULO_CD: VARCHAR(60)	EXISTENCIA: INT
Famous Blue Raincoat	13
Blue	42
Court and Spark	22
Past Light	17
Kojiki	6
That Christmas Feeling	8
Out of Africa	29
Blues on the Bayou	27
Orlando	5

ARTISTAS_CD

TITULO: VARCHAR(60)	ARTIST_NAME: VARCHAR(60)
Famous Blue Raincoat	Jennifer Warnes
Blue	Joni Mitchell
Court and Spark	Joni Mitchell
Past Light	William Ackerman
Kojiki	Kitaro
That Christmas Feeling	Bing Crosby
Patsy Cline: 12 Greatest Hits	Patsy Cline
After the Rain: The Soft Sounds of Erik Sat	Pascal Roge
Out of Africa	John Barry
Leonard Cohen The Best of	Leonard Cohen
Fundamental	Bonnie Raitt
Blues on the Bayou	B.B. King
Orlando	David Motion

Figura 12-1 Consultar las tablas `EXISTENCIA_CD` y `ARTISTAS_CD`.

Los resultados de la consulta deberán incluir solamente aquellas filas cuyo valor TITULO_CD coincida con uno de los valores arrojados por la subconsulta. Los resultados de la subconsulta deberán incluir solamente aquellas filas que contengan un valor NOMBRE_ARTISTA de Joni Mitchell (de la tabla ARTISTAS_CD). La siguiente instrucción SELECT arrojará estos datos:

```
SELECT *
  FROM EXISTENCIA_CD
 WHERE TITULO_CD IN
    ( SELECT TITULO
      FROM ARTISTAS_CD
      WHERE NOMBRE_ARTISTA = 'Joni Mitchell' );
```

Demos un vistazo más cercano a la subconsulta en esta instrucción. Como se puede ver, está incluida en el predicado IN, después de la palabra clave IN. La subconsulta es básicamente una instrucción SELECT que incluye una condición de búsqueda definida en la cláusula WHERE:

```
SELECT TITULO
  FROM ARTISTAS_CD
 WHERE NOMBRE_ARTISTA = 'Joni Mitchell'
```

Si se quisiera ejecutar solamente la subconsulta, se arrojarían los siguientes resultados de la consulta:

```
TITLE
-----
Blue
Court and Spark
```

Estos resultados son entonces utilizados por el predicado IN para compararlos con los valores TITULO_CD en la tabla EXISTENCIA_CD. Cuando se ejecuta la instrucción SELECT completa, se reciben los siguientes resultados:

TITULO_CD	EXISTENCIA
Blue	42
Court and Spark	22

Observe que solamente dos filas son arrojadas de la tabla EXISTENCIA_CD. Esas filas representan los dos CD grabados por Joni Mitchell. A pesar de que la tabla EXISTENCIA_CD no incluye información sobre el artista, aún es posible ligar los datos desde las dos tablas debido a que éstas incluyen columnas similares, permitiendo utilizar los datos arrojados por una subconsulta.

NOTA

En el caso de la tabla de ejemplo mostrada en la figura 12-1, es concebible que una clave externa pueda ser configurada en la columna TITULO_CD de la tabla EXISTENCIA_CD para hacer referencia a la columna TITULO de la tabla ARTISTAS_CD. Sin embargo, no es necesaria una relación de clave externa. El requerimiento principal que una subconsulta debe cumplir es que debe arrojar resultados que sean lógicamente comparables a los valores de columna referenciados. De otra manera, el propósito de la subconsulta sería nulo, y ninguna fila sería arrojada por la instrucción SELECT principal debido a que no se puede cumplir la condición del predicado IN.

Utilizar el predicado EXISTS

En algunas circunstancias se puede requerir que la subconsulta arroje solamente un valor de verdadero o falso. El contenido de los datos en sí mismo no es importante, en términos de cumplir con la condición de un predicado. En este caso, se puede utilizar un predicado EXISTS para definir la subconsulta. El predicado EXISTS se evalúa como verdadero si una o más filas son arrojadas por la subconsulta; de otra manera, se evalúa como falso.

Para que un predicado EXISTS sea de utilidad, la subconsulta asociada deberá incluir una condición de búsqueda que haga coincidir los valores en las dos tablas que están siendo vinculadas a través de la subconsulta. (Se explica este tipo de subconsulta con mayor detalle en la sección “Trabajar con subconsultas correlacionadas”, posteriormente en este capítulo.) Esta condición de búsqueda es similar a la condición equi-join utilizada en ciertas operaciones join. (Véase el capítulo 11 para mayor información acerca de condiciones join y equi-join.) Por ejemplo, regresando a la tabla EXISTENCIA_CD y a la tabla ARTISTAS_CD (mostradas en la figura 12-1), podemos crear una instrucción SELECT que utilice un predicado EXISTS para consultar la tabla ARTISTAS_CD:

```
SELECT *
  FROM EXISTENCIA_CD s
 WHERE EXISTS
    ( SELECT TITULO
      FROM ARTISTAS_CD a
      WHERE a.NOMBRE_ARTISTA = 'Joni Mitchell'
        AND s.TITULO_CD      = a.TITULO );
```

En esta instrucción, cada fila arrojada por la instrucción SELECT principal se evalúa con la subconsulta. Si la condición especificada en el predicado EXISTS es verdadera, la fila se incluye en los resultados de la consulta; de otra manera, la fila es omitida. Cuando la condición especificada es verdadera, eso significa que al menos una fila ha sido arrojada por la subconsulta. En este caso, la fila arrojada incluirá un valor NOMBRE_ARTISTA de Joni Mitchell. Además, el valor TITULO_CD en la tabla EXISTENCIA_CD permanecerá sin cambios igual que el valor TITULO en la tabla de ARTISTAS_CD. Como resultado, solamente se arrojarán dos filas de esa instrucción SELECT completa:

TITULO_CD	EXISTENCIA
Blue	42
Court and Spark	22

Como en el caso del predicado IN, el predicado EXISTS permite utilizar una subconsulta para acceder a la información en otra tabla. A pesar de que la tabla EXISTENCIA_CD no incluye información acerca de los artistas, la subconsulta permite arrojar los datos que estén basados en la información del artista.

NOTA

La forma en que se procesa un predicado EXISTS a veces puede ser un tanto confusa. Asegúrese de referirse al capítulo 9 para una discusión completa acerca de este predicado.

Utilizar predicados de comparación cuantificados

Los predicados IN y EXISTS no son los únicos predicados que se basan en el tipo de subconsultas que pueden arrojar una o más filas para que la condición de búsqueda se evalúe como verdadera. Los predicados de comparación cuantificados (SOME, ANY y ALL) también utilizan subconsultas que pueden arrojar múltiples filas. Estos predicados se utilizan en conjunción con operadores de comparación para determinar si alguno o todos los valores arrojados (de la subconsulta) cumplen la condición de búsqueda determinada por el predicado. Los predicados SOME y ANY, que realizan la misma función, comprueban si *algún* valor arrojado cumple con la condición de búsqueda. El predicado ALL comprueba si *todos* los valores arrojados cumplen con la condición de búsqueda.

Cuando se utiliza un predicado de comparación cuantificado, los valores en una columna de la tabla primaria se comparan con los valores arrojados por la subconsulta. Demos un vistazo a un ejemplo para dejar más claro cómo funciona esto. Supongamos que la base de datos incluye la tabla PRECIOS_MENUDEO y PRECIOS_VENTA, mostradas en la figura 12-2.

Ahora supongamos que se decide consultar la tabla PRECIOS_MENUDEO, pero solamente se quieren arrojar aquellas filas con un valor P_MENUDEO mayor que *todos* los valores en la columna P_VENTA de la tabla PRECIOS_VENTA para aquellos valores P_VENTA menores a 15.99. Para realizar esta consulta, se puede crear una instrucción similar a la siguiente:

```
SELECT NOMBRE_CD, P_MENUDEO
FROM PRECIOS_MENUDEO
WHERE P_MENUDEO > ALL
( SELECT P_VENTA
  FROM PRECIOS_VENTA
  WHERE P_VENTA < 15.99 );
```

Observe que la subconsulta arroja solamente una columna de datos (los valores P_VENTA que son menores a 15.99). Los valores en la columna P_MENUDEO son entonces comparados a los resultados de la subconsulta.

PRECIOS_MENUDEO			PRECIOS_VENTA	
NOMBRE_CD: VARCHAR(60)	P_MENUDEO: NUMERIC(5,2)	CANTIDAD: INT	TITULO_CD: VARCHAR(60)	P_VENTA: NUMERIC(5,2)
Famous Blue Raincoat	16.99	5	Famous Blue Raincoat	14.99
Blue	14.99	10	Blue	12.99
Court and Spark	14.99	12	Court and Spark	14.99
Past Light	15.99	11	Past Light	14.99
Kojiki	15.99	4	Kojiki	13.99
That Christmas Feeling	10.99	8	That Christmas Feeling	10.99
Patsy Cline: 12 Greatest Hits	16.99	14	Patsy Cline: 12 Greatest Hits	16.99

Figura 12-2 Consultar las tablas PRECIOS_MENUDEO y PRECIOS_VENTA.

Si un valor P_MENUDEO específico es mayor que *todos* los resultados de la subconsulta, la fila es arrojada. Cuando se ejecuta la instrucción SELECT completa, se reciben los siguientes resultados:

NOMBRE_CD	P_MENUDEO
-----	-----
Famous Blue Raincoat	16.99
Past Light	15.99
Kojiki	15.99
Patsy Cline: 12 Greatest Hits	16.99

Como se puede ver, sólo cuatro filas son arrojadas. Para cada fila, el valor P_MENUDEO es mayor al precio más alto arrojado por la subconsulta, el cual en este caso es de 14.99.

Pregunta al experto

P: Se dijo que la cláusula SELECT puede incluir una subconsulta. ¿Cómo se incluiría la subconsulta en esa cláusula?

R: Se puede incluir la subconsulta en una cláusula SELECT justo de la misma manera que se haría con un nombre de columna. Los valores arrojados desde las subconsultas son insertados en los resultados de la consulta de la misma forma en que serían insertados los valores de columna. Por ejemplo, se puede insertar una subconsulta en una cláusula SELECT de una instrucción que es utilizada para consultar la tabla EXISTENCIA_CD (mostrada en la figura 12-1). La subconsulta toma los datos desde la tabla ARTISTAS_CD, como se muestran en el siguiente ejemplo:

```
SELECT TITULO_CD,
       ( SELECT NOMBRE_ARTISTA
         FROM ARTISTAS_CD a
        WHERE s.TITULO_CD = a.TITULO ) AS ARTIST,
EXISTENCIA FROM EXISTENCIA_CD s;
```

En la parte principal de esta instrucción, los valores son tomados desde las columnas TITULO_CD y EXISTENCIA. Además de estos valores, una lista de artistas es arrojada por la subconsulta. Los nombres de artista se hacen coincidir con sus CD utilizando un predicado de comparación para comparar los valores en las columnas TITULO_CD y TITULO.

Cuando se utiliza una subconsulta en una cláusula SELECT, se debe ser cuidadoso de no crear una subconsulta que arroje solamente un valor cuando se necesitan múltiples valores. Cuando se arroja un solo valor, éste puede ser insertado en todas las filas arrojadas por la instrucción principal SELECT, dependiendo de cómo se haya construido la consulta.

Crear subconsultas que arrojen un solo valor

Hasta ahora hemos observado subconsultas que pueden arrojar una o más filas de datos. Esto es conveniente en muchas circunstancias; sin embargo, puede haber ocasiones en que se requiera que la subconsulta arroje solamente un valor para que puedan compararse los valores en una columna con un valor único de subconsulta. En estos casos se pueden utilizar los operadores de comparación.

Como se aprendió en el capítulo 9, los operadores de comparación incluyen igual a (=), desigual a (<>), menor que (<), mayor que (>), menor que o igual a (<=) y mayor que o igual a (>=). Por ejemplo, demos otro vistazo a las tablas PRECIOS_MENUDEO y PRECIOS_VENTA (mostradas en la figura 12-2). Supongamos que se quieren recuperar datos desde la tabla PRECIOS_MENUDEO. Se requiere que los valores P_MENUDEO igualen el precio máximo listado en la columna P_VENTA de la tabla PRECIOS_VENTA. La siguiente consulta permite arrojar los datos necesarios:

```
SELECT NOMBRE_CD, P_MENUDEO
FROM PRECIOS_MENUDEO
WHERE P_MENUDEO =
      ( SELECT MAX(P_VENTA)
        FROM PRECIOS_VENTA );
```

Observe que la subconsulta arroja solamente un valor, el cual en este caso es de 16.99. Como resultado, la instrucción SELECT arroja solamente filas con un valor P_MENUDEO de 16.99, como se muestra en los siguientes resultados de la consulta:

NOMBRE_CD	P_MENUDEO
-----	-----
Famous Blue Raincoat	16.99
Patsy Cline: 12 Greatest Hits	16.99

No se tiene que utilizar una función agregada (por ejemplo MAX) para arrojar un valor único en una subconsulta. Por ejemplo, la cláusula WHERE de la subconsulta pudiera incluir una condición que arrojaría un único valor. El punto importante a recordar es que debe asegurarse que la subconsulta arroje un único valor; de otra manera, se recibirá un error cuando se utilice un operador de comparación. Sin embargo, si se ha establecido la subconsulta apropiadamente, se puede utilizar cualquiera de los operadores de comparación para comparar los valores de columna. Además, no se está limitado en números. Las cadenas de caracteres también pueden ser comparadas en los predicados de comparación.

NOTA

En muchos casos se pueden utilizar predicados como IN con subconsultas que arrojan solamente un valor. Sin embargo, estos predicados pueden soportar solamente las condiciones igual a (=), o desigual a (<>), y no soportan las condiciones menor que (<), mayor que (>), menor que o igual a (<=) y mayor que o igual a (>=), y todas ellas se pueden utilizar con los operadores de comparación.

Trabajar con subconsultas correlacionadas

En la sección “Utilizar el predicado EXISTS” anterior en este capítulo, se mencionó que, para que el predicado EXISTS sea de utilidad, deberá incluir en la subconsulta una condición de búsqueda que coincida con los valores en las dos tablas que están siendo vinculadas a través de la subconsulta. Para ilustrar este punto, se incluye en esta sección una instrucción SELECT de ejemplo que contiene tal subconsulta. Se repetirá esa instrucción aquí para mayor facilidad:

```
SELECT *
  FROM EXISTENCIA_CD s
 WHERE EXISTS
    ( SELECT TITULO
      FROM ARTISTAS_CD a
     WHERE a.NOMBRE_ARTISTA = 'Joni Mitchell'
           AND s.TITULO_CD   = a.TITLE );
```

Esta instrucción hace referencia a las tablas EXISTENCIA_CD y ARTISTAS_CD en la figura 12-1. Observe que la subconsulta incluye un predicado que hace coincidir los valores TITULO_CD en la tabla EXISTENCIA_CD con los valores TITLE en la tabla ARTISTAS_CD. Esta función de hacer coincidir los valores es similar a las condiciones equi-join que se definen cuando se unen tablas.

La razón por la que se ha regresado a esta instrucción es que ésta incluye un tipo de subconsulta que no se había analizado antes (la subconsulta correlacionada). Una *subconsulta correlacionada* es aquella que está pendiente de alguna manera de la instrucción outer. En este caso, la instrucción outer es la instrucción SELECT principal que incluye una cláusula SELECT, una cláusula FROM y una cláusula WHERE, la cual contiene en sí misma una subconsulta. Debido a que esa subconsulta hace referencia a la tabla EXISTENCIA_CD, la cual es un componente de la instrucción outer, la subconsulta es dependiente de esa instrucción para poder arrojar los datos.

En la mayoría de los ejemplos de subconsulta que hemos visto en este capítulo, las subconsultas han permanecido independientes de la instrucción outer. Por ejemplo, en la siguiente instrucción SELECT (la cual fue utilizada como ejemplo en la sección “Utilizar el predicado IN” anteriormente en este capítulo), la subconsulta no es dependiente de la instrucción outer:

```
SELECT *
  FROM EXISTENCIA_CD
 WHERE TITULO_CD IN
    ( SELECT TITULO
      FROM ARTISTAS_CD
     WHERE NOMBRE_ARTISTA = 'Joni Mitchell' );
```

En este caso, la subconsulta simplemente arroja resultados, que son luego utilizados en una instrucción outer. La subconsulta es evaluada (ejecutada) sólo una vez, y los resultados son utilizados por la instrucción principal según sea necesario. Sin embargo, con una subconsulta correlacionada, la subconsulta debe a menudo ser revaluada para cada fila arrojada por la instrucción outer. La subconsulta correlacionada no puede ser evaluada solamente una vez debido a que al menos uno de los valores cambia para cada fila. Por ejemplo, si miramos otra vez a la instrucción SELECT que contiene la subconsulta correlacionada (como parte del predicado EXISTS), se puede ver que el valor TITULO_CD cambia para cada fila arrojada por la instrucción SELECT outer. Esto puede tener un severo impacto en el rendimiento, particularmente cuando se está arrojando un gran número de valores. En estos casos, se puede encontrar que crear una operación join proporciona mejor rendimiento que una subconsulta correlacionada, pero desde luego depende en gran parte de cómo la implementación SQL maneje las operaciones join y las subconsultas correlacionadas.

Pregunta al experto

P: Se dijo que crear una operación join puede ser una mejor alternativa que crear una subconsulta correlacionada. ¿Cómo se restablecería la instrucción SELECT anterior como una operación join?

R: En la instrucción SELECT anterior ya se ha identificado la condición equi-join en la subconsulta, y ya se conocen los nombres de las dos tablas que están siendo unidas. Una forma en que se puede modificar esta instrucción es utilizando una operación join separada por comas, como se muestra en el siguiente ejemplo:

```
SELECT TITULO_CD, EXISTENCIA
FROM EXISTENCIA_CD s, ARTISTAS_CD a
WHERE a.NOMBRE_ARTISTA = 'Joni Mitchell'
AND s.TITULO_CD = a.TITULO;
```

Observe que las columnas TITULO_CD y TITULO están aún equiparadas una con otra. Esta instrucción produce los mismos resultados que la instrucción que incluía la subconsulta correlacionada, sólo que la implementación SQL no está siendo forzada a reprocesar una subconsulta para cada fila arrojada por la instrucción outer. En su lugar, la cláusula WHERE simplemente toma los resultados arrojados por la cláusula FROM y aplica las condiciones de búsqueda definidas en los dos predicados. Para mayor información acerca de las operaciones join, véase el capítulo 11.

Utilizar subconsultas anidadas

Hasta este punto hemos visto las instrucciones SELECT que incluyen solamente una subconsulta. Sin embargo, una instrucción SELECT puede contener múltiples subconsultas. El estándar SQL:2006 no limita el número de subconsultas que pueden ser incluidas en la instrucción, aunque la aplicación práctica, el rendimiento y las limitaciones de la implementación SQL juegan un papel importante al determinar cuál podría ser un número razonable. Asegúrese de referirse a la documentación de su implementación SQL para determinar cuáles restricciones pueden aplicar al uso de las subconsultas.

Una forma de poder incluir múltiples subconsultas en una instrucción SELECT es incluirlas como diferentes componentes de la instrucción. Por ejemplo, la cláusula WHERE podría incluir dos predicados, y cada uno de ellos contener una subconsulta. Otra forma en la que múltiples subconsultas pueden ser incluidas en una instrucción SELECT es anidar una subconsulta dentro de otra. Éstos son los tipos de subconsultas que se verán en esta sección.

Una *subconsulta anidada* es aquella que es un componente de otra subconsulta. La subconsulta “outer” actúa como una instrucción SELECT primaria que incluye una subconsulta dentro de una de sus cláusulas. En la mayoría de los casos, la subconsulta anidada será parte de un predicado en la cláusula WHERE de la subconsulta outer. Veamos un ejemplo para ayudar a explicar mejor

INVENTARIO_DISCO			ARTISTAS_DISCO			TIPOS_DISCO	
NOMBRE_DISCO: VARCHAR(60)	ID_ARTISTA: INT	CANTIDAD_EXISTENCIA: INT	ID_ARTISTA: INT	NOMBRE_ARTISTA: VARCHAR(60)	ID_TIPO_DISCO: INT	ID_TIPO_DISCO: INT	NOMBRE_TIPO_DISCO: CHAR(20)
Famous Blue Raincoat	102	12	101	Joni Mitchell	10	10	Popular
Blue	101	24	102	Jennifer Warnes	12	11	Blues
Court and Spark	101	17	103	B.B. King	11	12	Folk
Past Light	105	9	104	Bonnie Raitt	10	13	Rock
Fundamental	104	22	105	William Ackerman	15	14	Classical
Blues on the Bayou	103	19	106	Bing Crosby	16	15	New Age
Longing in Their Hearts	104	18	107	Patsy Cline	17	16	Classic Pop
Luck of the Draw	104	25	108	John Barry	18	17	Country
Deuces Wild	103	17	109	Leonard Cohen	12	18	Soundtrack
Nick of Time	104	11					
Both Sides Now	101	13					

Figura 12-3 Consultar las tablas INVENTARIO_DISCO, ARTISTAS_DISCO y TIPOS_DISCO.

este concepto. El ejemplo utiliza las tablas INVENTARIO_DISCO, ARTISTAS_DISCO y TIPOS_DISCO, mostradas en la figura 12-3.

Supongamos que se quieren desplegar los nombres de los CD y la cantidad en existencia de los CD que son interpretados por artistas de blues. La tabla INVENTARIO_DISCO contiene los nombres de los CD y la cantidad en existencia para cada uno, la tabla ARTISTAS_DISCO contiene los nombres de los artistas, y la tabla TIPOS_DISCO contiene los nombres de los tipos de artista. Las tablas INVENTARIO_DISCO y ARTISTAS_DISCO están relacionadas a través de la columna ID_ARTISTA en cada tabla. Las tablas ARTISTAS_DISCO y TIPOS_DISCO están relacionadas a través de la columna ID_TIPO_DISCO en cada tabla. Para arrojar la información necesaria se deberán consultar las tres tablas, como se muestra en la siguiente instrucción SELECT:

```
SELECT NOMBRE_DISCO, CANTIDAD_EXISTENCIA
FROM INVENTARIO_DISCO
WHERE ID_ARTISTA IN
( SELECT ID_ARTISTA
  FROM ARTISTAS_DISCO
  WHERE ID_TIPO_DISCO IN
    ( SELECT ID_TIPO_DISCO
      FROM TIPOS_DISCO
      WHERE NOMBRE_TIPO_DISCO = 'Blues' ) );
```

En esta instrucción, la instrucción SELECT primaria consulta la tabla INVENTARIO_DISCO. La instrucción incluye una subconsulta en un predicado IN en la cláusula WHERE. La subconsulta es una instrucción SELECT que consulta la tabla ARTISTAS_DISCO. La subconsulta, al igual que la instrucción SELECT primaria, incluye un predicado IN en la cláusula WHERE, y este predicado también incluye una subconsulta. Como es el caso con la subconsulta outer, la subconsulta inner incluye una instrucción SELECT. Sin embargo, en este caso la instrucción está consultando la tabla TIPOS_DISCO.

Para comprender mejor cómo funciona la instrucción `SELECT` completa, primero veamos la subconsulta `inner`. Si se quisiera ejecutar esta instrucción por sí sola, arrojaría un valor de 11, el cual es el valor `ID_TIPO_DISCO` para el valor `NOMBRE_TIPO_DISCO` de Blues. La subconsulta `outer` utiliza este valor en el predicado `IN` para arrojar aquellas filas con un valor `ID_TIPO_DISCO` de 11. En este caso, la única fila arrojada es la fila B.B. King, que tiene un valor `ID_ARTISTA` de 103. El valor `ID_ARTISTA` es entonces utilizado en el predicado `IN` de la instrucción `SELECT` primaria para arrojar solamente aquellas filas que contengan un valor `ID_ARTISTA` de 103. Si se ejecuta la instrucción `SELECT` completa, se arrojarán los siguientes resultados de la consulta:

NOMBRE_DISCO	CANTIDAD_EXISTENCIA
-----	-----
Blues on the Bayou	19
Deuces Wild	17

Como se puede ver, solamente dos filas son arrojadas. Observe que los resultados no incluyen ninguna información de la tabla `ARTISTAS_DISCO` o de la tabla `TIPOS_DISCO`, a pesar de que estas dos tablas son de gran importancia para llegar a estos resultados. Si se hubiera deseado, se podrían haber anidado subconsultas adicionales en la instrucción. Cada una se habría procesado de la misma manera que las subconsultas mostradas en el ejemplo anterior.

Utilizar subconsultas para modificar datos

Al inicio de este capítulo se dijo que es posible utilizar subconsultas para modificar datos al igual que para consultar datos. Ahora veremos las tres instrucciones de modificación de datos primarias (`INSERT`, `UPDATE` y `DELETE`) y cómo éstas utilizan subconsultas para modificar los datos en la base de datos. Para cada instrucción se proporciona un ejemplo que modifica los datos en la tabla `TIPOS_TITULO`, mostrada en la figura 12-4. Cada ejemplo incluye una subconsulta que arroja datos desde la tabla `INVENTARIO_TITULOS`. Esta información es utilizada como una base para la modificación de datos en la tabla `TIPOS_TITULO`.

NOTA

Esta sección se centra en las subconsultas utilizadas en las instrucciones `INSERT`, `UPDATE` y `DELETE`. Para mayor información acerca de estas instrucciones por sí mismas, véase el capítulo 8.

Utilizar subconsultas para insertar datos

Una instrucción `INSERT`, como sin duda recordará, permite agregar datos a una tabla existente. Se pueden agregar esos datos directamente a la tabla o a través de una vista que permita insertar datos a la tabla subyacente. Si se utiliza una subconsulta en una instrucción `INSERT`, debe ser incluido como uno de los valores definidos en la cláusula `VALUES`. Por ejemplo, supongamos que se quieren insertar datos a la tabla `TIPOS_TITULO`. La cláusula `VALUES` deberá incluir un valor para la columna `TITULO_CD` y la columna `TIPO_CD`. Ahora supongamos que se conoce el valor

INVENTARIO_TITULOS

ID_TITULO: INT	TITULO: VARCHAR(60)	EXISTENCIA: INT
101	Famous Blue Raincoat	12
102	Blue	24
103	Past Light	9
104	Blues on the Bayou	19
105	Luck of the Draw	25
106	Deuces Wild	17
107	Nick of Time	11
108	Both Sides Now	13

TIPOS_TITULO

TITULO_CD: VARCHAR(60)	TIPO_CD: CHAR(20)
Famous Blue Raincoat	Folk
Blue	Popular
Court and Spark	Popular
Past Light	New Age
Fundamental	Popular
Blues on the Bayou	Blues
Longing in their Hearts	Popular
Deuces Wild	Blues
Nick of Time	Popular

Figura 12-4 Modificar la tabla TIPOS_TITULO.

ID_TITULO (de la tabla INVENTARIO_TITULOS), pero no se conoce el exacto nombre del CD. Se puede crear una instrucción INSERT que tome el nombre del CD desde la tabla INVENTARIO_TITULOS e inserte ese valor en la tabla TIPOS_TITULO, como se muestra en el siguiente ejemplo:

```
INSERT INTO TIPOS_TITULO VALUES
( ( SELECT TITULO FROM INVENTARIO_TITULOS WHERE ID_TITULO = 108 ),
'Popular' );
```

Observe que la subconsulta aparece como uno de los valores en la cláusula VALUES. La subconsulta arroja el valor Both Sides Now. Este valor y el valor Popular son insertados en la tabla TIPOS_TITULO.

En su mayoría, utilizar una subconsulta en una instrucción INSERT es relativamente un proceso simple. Sin embargo, debe asegurarse de que la subconsulta arroje solamente un valor; de otra manera, se recibirá un error. Además, el valor debe ser compatible con el tipo de datos y con cualquier otro requerimiento definido en la columna objetivo.

NOTA

No todas las implementaciones SQL soportan el uso de una subconsulta como un valor en la instrucción INSERT. Por ejemplo, SQL Server no permite insertar valores de esta manera, mientras que Oracle y MySQL sí lo hacen.

Utilizar subconsultas para actualizar datos

Una instrucción UPDATE permite modificar los datos existentes en una tabla. Al igual que con una instrucción INSERT, se pueden modificar los datos directamente o a través de una vista, si esa vista es actualizable. Para utilizar una subconsulta en una instrucción UPDATE, se puede incluir en un predicado en la cláusula WHERE, como se hizo con las instrucciones SELECT que se vieron anteriormente en este capítulo. Por ejemplo, si se quiere actualizar el valor Both Sides Now que fue insertado en el ejemplo anterior de instrucción INSERT, se puede crear una instrucción UPDATE similar a la siguiente:

```
UPDATE TIPOS_TITULO
  SET CD_TYPE = 'Folk'
 WHERE TITULO_CD IN
    ( SELECT TITULO
      FROM INVENTARIO_TITULOS
      WHERE TITLE_ID = 108 );
```

En esta instrucción, el predicado IN compara los valores en la columna TITULO_CD de la tabla TIPOS_TITULO con el valor arrojado por la subconsulta. La subconsulta es una instrucción SELECT simple que arroja datos desde la tabla INVENTARIO_TITULOS. La subconsulta funciona aquí de la misma manera que se vio en los ejemplos anteriores de instrucciones SELECT. En este caso, la subconsulta arroja un valor Both Sides Now. Este valor es por lo tanto utilizado para determinar cuál fila actualizar en la tabla TIPOS_TITULO. Una vez que esta fila es determinada, el valor TIPO_CD es cambiado a Folk.

Las subconsultas no están limitadas a la cláusula WHERE de una instrucción UPDATE. También se puede utilizar una subconsulta en la cláusula SET para proporcionar un valor para la columna identificada. Por ejemplo, supongamos que se quiere una vez más actualizar el valor Both Sides Now que fue insertado en el ejemplo anterior de instrucción INSERT. Se puede tomar un valor desde la tabla INVENTARIO_TITULOS para utilizarlo como el nuevo valor para la tabla TIPOS_TITULO, como se muestra en la siguiente instrucción UPDATE:

```
UPDATE TIPOS_TITULO
  SET TITULO_CD =
    ( SELECT TITULO
      FROM INVENTARIO_TITULOS
      WHERE ID_TITULO = 108 )
 WHERE TITULO_CD = 'Both Sides Now';
```

Observe que, en lugar de especificar un valor en la cláusula SET (a la derecha del signo de igual), se puede especificar una subconsulta. La subconsulta arroja un valor de Both Sides Now e inserta ese valor en la tabla TIPOS_TITULO.

NOTA

En el ejemplo anterior, todo lo que se ha hecho es escribir el mismo valor sobre uno existente. El propósito de esta instrucción solamente es demostrar cómo una subconsulta puede ser utilizada en una cláusula SET. Incluso si un nuevo valor hubiera sido escrito dentro de la fila, los principios hubieran sido los mismos. Por ejemplo, si el título hubiera cambiado en la tabla INVENTARIO_TITULOS, la instrucción anterior habría actualizado el título en la tabla TIPOS_TITULO.

Utilizar subconsultas para eliminar datos

Una instrucción DELETE es similar a una instrucción UPDATE, en términos de cómo puede ser utilizada una subconsulta en la cláusula WHERE. Simplemente se incluye un predicado que contenga una subconsulta. En el siguiente ejemplo se borra el valor Both Sides Now que se modificó en el ejemplo anterior de instrucción UPDATE. Para determinar cuál fila debe eliminarse, se utiliza una subconsulta para arrojar el valor apropiado TITULO de la tabla INVENTARIO_TITULOS:

```
DELETE TIPOS_TITULO
WHERE TITULO_CD IN
  ( SELECT TITLE
    FROM INVENTARIO_TITULOS
    WHERE TITLE_ID = 108 );
```

Como puede esperarse, la subconsulta arroja el valor de Both Sides Now. El predicado IN compara este valor a los valores en la columna TITULO_CD de la tabla TIPOS_TITULO. Cada fila con valores coincidentes es eliminada. En este caso, solamente una fila tiene un valor TITULO_CD de Both Sides Now; por lo tanto, ésta es la fila que es eliminada.

Pruebe esto 12-1 Trabajar con subconsultas

En este capítulo se analizó cómo se pueden utilizar subconsultas para consultar y modificar datos. Las subconsultas que se han visto, dependen en su mayor parte del uso de predicados para definir la condición de la subconsulta. En este ejercicio creará varias instrucciones SELECT que incluyan cláusulas WHERE. Cada una de esas cláusulas contendrá un predicado que defina una subconsulta, permitiendo el acceso a los datos para más de una tabla. También modificará los datos utilizando una instrucción UPDATE que contenga subconsultas en la cláusula SET y en la cláusula WHERE. Para este ejercicio, al igual que con los ejercicios anteriores, se utilizará la base de datos INVENTARIO. Puede descargar el archivo Try_This_12.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. La primera instrucción SELECT que se va a crear permite arrojar el nombre y el número de CD que son producidos por MCA Records. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE ID_DISQUERA IN
  ( SELECT ID_DISQUERA
    FROM DISQUERAS_CD
    WHERE NOMBRE_COMPANIA = 'MCA Records' );
```

(continúa)

Esta instrucción utiliza una subconsulta para arrojar el valor ID_DISQUERA para MCA Records, que está almacenado en la tabla DISQUERAS_CD. El valor es entonces utilizado en el predicado IN para compararlo con los valores ID_DISQUERA en la tabla DISCOS_COMPACTOS. La consulta deberá arrojar cuatro filas.

3. En la siguiente instrucción se utilizará un predicado EXISTS para definir una subconsulta. El predicado determina si es que la tabla DISCOS_COMPACTOS contiene alguna fila con el valor TITULO_CD de Out of Africa. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT NOMBRE_COMPANIA
FROM DISQUERAS_CD l
WHERE EXISTS
( SELECT *
  FROM DISCOS_COMPACTOS d
  WHERE l.ID_DISQUERA = d.ID_DISQUERA
    AND TITULO_CD = 'Out of Africa' );
```

La instrucción arrojará el nombre de la compañía que produce el CD *Out of Africa*, el cual en este caso es MCA Records. La fila MCA Records en la tabla DISQUERAS_CD es la única fila que se evalúa como verdadera para la subconsulta en el predicado EXISTS.

4. En la siguiente instrucción a crear, se determinarán los nombres de los distribuidores para aquellos CD en los cuales el valor ID_DISQUERA en la tabla DISQUERAS_CD es igual a cualesquiera que sean los valores ID_DISQUERA arrojados por la subconsulta. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT NOMBRE_COMPANIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ANY
( SELECT ID_DISQUERA
  FROM DISCOS_COMPACTOS
  WHERE EN_EXISTENCIA > 30 );
```

La subconsulta arroja solamente aquellos valores ID_DISQUERA para las filas que contienen un valor EN_EXISTENCIA mayor a 30. Cuando se ejecuta esta instrucción, los nombres de sólo tres compañías deberán ser arrojados.

5. Ahora se creará una instrucción SELECT que utilice un predicado de comparación para definir una subconsulta. La subconsulta arroja el valor ID_DISQUERA (de la tabla DISQUERAS_CD) para Capitol Records. El valor es entonces comparado con los valores ID_DISQUERA en la tabla DISCOS_COMPACTOS. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE ID_DISQUERA =
( SELECT ID_DISQUERA
  FROM DISQUERAS_CD
  WHERE NOMBRE_COMPANIA = 'Capitol Records' );
```

Esta instrucción deberá arrojar solamente dos filas.

6. Ahora hagamos de nuevo la instrucción del paso 5, pero ahora la convertiremos en una operación join separada por comas. Recuerde que deberá asignar nombres de correlación a las tablas para simplificar el código. También recuerde que la cláusula WHERE deberá incluir una condición equi-join que hará coincidir los valores ID_DISQUERA. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
WHERE d.ID_DISQUERA = l.ID_DISQUERA
AND NOMBRE_COMPANIA = 'Capitol Records';
```

Como se puede ver, esta instrucción es mucho más simple que la subconsulta utilizada en la instrucción anterior, y arroja los mismos resultados.

7. En la siguiente instrucción a crear, se utilizará una subconsulta anidada para arrojar los valores a la subconsulta outer. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT NOMBRE_ARTISTA
FROM ARTISTAS
WHERE ID_ARTISTA IN
( SELECT ID_ARTISTA
  FROM CDS_ARTISTA
  WHERE ID_DISCO_COMPACTO IN
    ( SELECT ID_DISCO_COMPACTO
      FROM DISCOS_COMPACTOS
      WHERE TITULO_CD = 'Past Light' ) );
```

La subconsulta inner arroja el valor ID_DISCO_COMPACTO para el CD Past Light. La subconsulta outer entonces utiliza ese valor para determinar el valor ID_ARTISTA para ese CD. Este valor es luego utilizado en la instrucción SELECT principal, la cual arroja un solo valor: William Ackerman. Él es el artista del CD Past Light.

8. Ahora vamos empezar a utilizar subconsultas en una instrucción UPDATE. Sin embargo, primero demos un vistazo a la tabla que se va a actualizar, que es la tabla TIPOS_DISCO_COMPACTO. Para saber qué se va a actualizar, vamos a utilizar valores de la tabla DISCOS_COMPACTOS y de la tabla TIPOS_MUSICA para ayudar a identificar los números de ID utilizados en la tabla TIPOS_DISCO_COMPACTO. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, NOMBRE_TIPO
FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
AND t.ID_TIPO_MUSICA = m.ID_TIPO
AND TITULO_CD = 'Kojiki';
```

En esta instrucción se unieron tres tablas para arrojar el valor TITULO_CD y el valor TYPE_NAME para el CD Kojiki. El CD está clasificado como New Age.

9. En este paso se actualizará la fila en la tabla TIPOS_DISCO_COMPACTO que coincida con ID_DISCO_COMPACTO para el CD Kojiki con el valor ID_TIPO_MUSICA para el tipo de

(continúa)

música New Age. Se cambiará el tipo de música de New Age a Classical. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE TIPOS_DISCO_COMPACTO
  SET ID_TIPO_MUSICA =
    ( SELECT ID_TIPO
      FROM TIPOS_MUSICA
      WHERE NOMBRE_TIPO = 'Classical' )
WHERE ID_DISCO_COMPACTO =
  ( SELECT ID_DISCO_COMPACTO
    FROM DISCOS_COMPACTOS
    WHERE TITULO_CD = 'Kojiki' )
AND ID_TIPO_MUSICA =
  ( SELECT ID_TIPO
    FROM TIPOS_MUSICA
    WHERE NOMBRE_TIPO = 'New Age' );
```

La instrucción utiliza una subconsulta en la cláusula SET para tomar el valor TYPE_ID de la tabla MUSIC_TYPES. La instrucción también utiliza dos subconsultas en la cláusula WHERE de la instrucción UPDATE para determinar cuál fila será actualizada en la tabla TIPOS_DISCO_COMPACTO. La primera subconsulta en la cláusula WHERE arroja el valor ID_DISCO_COMPACTO para el CD Kojiki. La segunda subconsulta arroja el valor ID_TIPO para el tipo de música Classical.

- 10.** Ahora consultemos la tabla TIPOS_DISCO_COMPACTO para ver los cambios. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, NOMBRE_TIPO
  FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
 WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
    AND t.ID_TIPO_MUSICA = m.ID_TIPO
    AND TITULO_CD = 'Kojiki';
```

El valor NOMBRE_TIPO ahora deberá ser Classical.

- 11.** Finalmente, se requiere arrojar la tabla TIPOS_DISCO_COMPACTO en su estado original. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE TIPOS_DISCO_COMPACTO
  SET ID_TIPO_MUSICA =
    ( SELECT ID_TIPO
      FROM TIPOS_MUSICA
      WHERE NOMBRE_TIPO = 'New Age' )
WHERE ID_DISCO_COMPACTO =
  ( SELECT ID_DISCO_COMPACTO
    FROM DISCOS_COMPACTOS
    WHERE TITULO_CD = 'Kojiki' )
  AND ID_TIPO_MUSICA =
    ( SELECT ID_TIPO
      FROM TIPOS_MUSICA
      WHERE NOMBRE_TIPO = 'Classical' );
```

Esta instrucción es similar a la instrucción UPDATE anterior, sólo que ahora el tipo de música New Age será utilizado (éste era el tipo de música original).

- 12.** Ahora revisemos la tabla una vez más. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, NOMBRE_TIPO
FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
AND t.ID_TIPO_MUSICA = m.ID_TIPO
AND TITULO_CD = 'Kojiki';
```

La tabla COMPACT_TIPOS_DISCO deberá ahora contener los mismos valores que tenía cuando se inició este ejercicio.

- 13.** Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio se crearon varias instrucciones SELECT que contenían subconsultas. Estas subconsultas fueron incluidas en predicados que permitían a las subconsultas arrojar una o más filas. Específicamente, las cláusulas WHERE incluyeron los predicados IN, EXISTS y ANY. Además, se creó una instrucción SELECT que incluía un predicado de comparación, el cual permite a la subconsulta arrojar solamente una fila. También se creó una instrucción SELECT que incluía subconsultas anidadas. Estas consultas utilizaron el predicado IN. Además de consultar datos de la base de datos INVENTARIO, se actualizó la tabla TIPOS_DISCO_COMPACTO utilizando subconsultas que accedieron a otras tablas. Como se puede ver, las subconsultas proporcionan una herramienta versátil para acceder a los datos en la base de datos. Sin embargo, cuando se crean instrucciones que incluyen subconsultas, siempre se deberá tratar de determinar si una operación join pudiera tener un mejor rendimiento en cualquier situación dada.

Autoexamen Capítulo 12

1. ¿En cuál tipo de instrucción se pueden incluir subconsultas?
 - A** SELECT
 - B** INSERT
 - C** UPDATE
 - D** DELETE
2. ¿Qué es una subconsulta?

3. ¿En cuáles cláusulas de una instrucción SELECT se puede incluir una subconsulta?
 - A SELECT
 - B WHERE
 - C GROUP BY
 - D HAVING
4. ¿En cuáles dos categorías generales se pueden dividir las subconsultas de una cláusula WHERE?
5. ¿Cuáles tipos de predicados se deben evitar utilizar con subconsultas que arrojen múltiples filas?
 - A Predicados IN y EXISTS
 - B Predicados SOME, ANY y ALL
 - C Predicados de comparación
 - D Predicados de comparación cuantificados
6. ¿Cuándo se evalúa una condición EXISTS como verdadera?
7. ¿Qué deberá incluirse en la condición de búsqueda de una subconsulta cuando se utiliza un predicado EXISTS?
8. Además de números, los datos _____ pueden ser comparados en los predicados de comparación.
9. ¿Cuáles son los tres predicados de comparación cuantificados?
10. ¿Qué tipos de predicados permiten utilizar subconsultas que arrojen múltiples filas?
 - A Predicados IN y EXISTS
 - B Predicados SOME, ANY y ALL
 - C Predicados de comparación
 - D Predicados de comparación cuantificados
11. ¿Qué es una subconsulta correlacionada?
12. ¿Qué tan seguido es evaluada una subconsulta correlacionada cuando se procesa una instrucción SELECT?
13. Un(a) _____ es una subconsulta que es un componente de otra subconsulta.
14. ¿Cuántas subconsultas pueden ser incluidas en una instrucción SELECT, según especifica el estándar SQL?
15. ¿Cuál cláusula en una instrucción INSERT puede contener una subconsulta?
16. ¿Cuántos valores puede arrojar una subconsulta si es utilizada dentro de una instrucción INSERT?
17. ¿Cuáles cláusulas en una instrucción UPDATE pueden contener una subconsulta?

Parte III

Acceso avanzado a los datos



Capítulo 13

Crear rutinas invocadas
por SQL

Habilidades y conceptos clave

- Entender las rutinas invocadas por SQL
- Crear procedimientos invocados por SQL
- Agregar parámetros de entrada a sus procedimientos
- Agregar variables locales a sus procedimientos
- Trabajar con instrucciones de control
- Agregar parámetros de salida a sus procedimientos
- Crear funciones invocadas por SQL

Antes de la liberación de SQL:1999, el Instituto Americano de Estándares (ANSI, o *American National Standards Institute*) y la Organización Internacional de Normalización (ISO, o *International Organization for Standardization*) publicaron un estándar provisional en 1996 que agregaba procedimientos y funciones, junto con el lenguaje relacionado, al estándar SQL existente como Part 4. Esta nueva publicación, también conocida como SQL/PSM, o PSM-96 (PSM se refiere a *persistent stored module*), representó el primer paso para incluir capacidades de procedimiento dentro del mismo SQL. Part 4 (SQL/PSM) fue revisada e incorporada al estándar SQL:1999, y revisada una vez más para el estándar SQL:2003. Estas capacidades de procedimiento definen, entre otros componentes, la creación de rutinas invocadas por SQL (específicamente, procedimientos invocados por SQL y funciones invocadas por SQL). En este capítulo veremos de cerca ambos procedimientos y funciones, incluyendo cómo crearlos y cómo invocarlos una vez que han sido creados. También se verán varios ejemplos para demostrar los diferentes tipos de procedimientos y funciones y los componentes que forman a cada uno.

Entender las rutinas invocadas por SQL

Se introdujo por primera vez el concepto de las rutinas invocadas por SQL en el capítulo 2, donde se describieron los objetos de esquema que pueden existir dentro de un ambiente SQL. Como se podrá recordar, una *rutina invocada por SQL* es una función o procedimiento que puede ser invocado desde SQL. Ambas funciones y procedimientos son conjuntos almacenados de instrucciones SQL predefinidas que realizan algún tipo de acción sobre los datos en la base de datos. Por ejemplo, se puede definir una instrucción SELECT y almacenarla como un procedimiento invocado por SQL. Una vez que se ha creado ese procedimiento, puede invocarse simplemente citando su nombre y, si resulta apropiado, proporcionando los parámetros necesarios.

A diferencia de las vistas, todas las rutinas invocadas por SQL soportan el uso de *parámetros*, que son valores que pasan desde y hacia una rutina cuando se invoca esa rutina. Una función puede recibir parámetros de entrada y arrojar un valor basado en la expresión incluida en la definición de la función. Un procedimiento puede pasar parámetros de entrada y de salida. Sin importar si es un procedimiento o una función, una rutina invocada por SQL puede ser un objeto de esquema o

puede estar incrustada en un módulo de SQL Server, que también es un objeto de esquema. (Un *módulo* es un objeto que contiene instrucciones o rutinas SQL.)

NOTA

SQL:2006 también soporta un tercer tipo de rutina invocada por SQL (el método invocado por SQL). Un método, que es utilizado en los tipos definidos de usuario, es un tipo de función que realiza tareas predefinidas. SQL soporta dos tipos de tipos definidos por el usuario: tipos estructurados y tipos distintos. Los métodos se utilizan en los tipos estructurados. El tema de los tipos estructurados definidos por el usuario sale del campo de acción de este libro, por lo que no se cubrirán métodos en este capítulo.

La mayoría de las implementaciones SQL soportan alguna forma de la rutina invocada por SQL en sus productos. En varias implementaciones SQL, los procedimientos invocados por SQL a menudo son nombrados *procedimientos almacenados*, y las funciones invocadas por SQL a menudo son nombradas *funciones definidas por el usuario*. Sin importar los nombres utilizados, los conceptos fundamentales son los mismos, y la funcionalidad básica soportada es similar de un producto a otro. Sin embargo, mientras los conceptos y la funcionalidad son similares, la implementación de las rutinas invocadas por SQL puede variar ampliamente, y los detalles acerca de cómo son creadas y nombradas las rutinas invocadas por SQL difieren no solamente entre el estándar SQL y los productos individuales, sino también entre los mismos productos. La principal razón de esto es que muchos productos habían ya implementado tecnología PSM antes de la publicación inicial del estándar SQL/PSM en 1996. Como resultado, la funcionalidad propietaria ha persistido entre las diferentes implementaciones, con pocos productos SQL ajustándose al estándar real SQL/PSM, o, consecuentemente, la porción relacionada con PSM del estándar SQL:2006.

A pesar de las diferencias de los productos, resulta conveniente dar un vistazo a los conceptos básicos detrás de las rutinas invocadas por SQL, ya que están definidos en el estándar SQL. El estándar proporciona una mirada hacia la estructura subyacente utilizada por varias implementaciones SQL y puede darle una idea general cohesiva de los conceptos básicos compartidos por todos los productos que implementan los procedimientos y las funciones invocadas por SQL. Sin embargo, al igual que con otras tecnologías relacionadas a SQL, deberá referirse a la documentación del producto para su implementación específica SQL. En muy pocos casos se podrá utilizar el estándar SQL puro para crear una rutina invocada por SQL de implementación específica.

Procedimientos y funciones invocadas por SQL

Como se mencionó anteriormente, una rutina invocada por SQL puede ser ya sea un procedimiento invocado por SQL o una función invocada por SQL (o, en el caso de los tipos de datos de usuario, un método invocado por SQL). Los procedimientos y funciones invocados por SQL son similares de muchas maneras, aunque existen algunas diferencias básicas. La tabla 13-1 proporciona una vista general de las principales diferencias y similitudes.

La forma más sencilla de distinguir entre los procedimientos y funciones invocados por SQL es considerar un procedimiento como un conjunto de una o más instrucciones SQL almacenadas, similar a cómo una vista almacena una instrucción SELECT (como se describió en el capítulo 5), y considerar una función como un tipo de operación que arroja un valor, similar a las funciones SET como SUM o AVG (descritas en el capítulo 10).

Procedimientos	Funciones
Se invoca desde las instrucciones SQL, no desde un lenguaje de programación.	Se invoca desde las instrucciones SQL, no desde un lenguaje de programación.
Puede ser escrito en SQL o en otro lenguaje de programación.	Puede ser escrito en SQL o en otro lenguaje de programación.
Se invoca utilizando la instrucción CALL.	Se invoca como un valor en una expresión.
Soporta parámetros de entrada y salida, aunque ninguno de ellos es obligatorio.	Soporta parámetros de entrada, aunque no son obligatorios. No se pueden definir parámetros de entrada o salida para una función. La función arroja un único valor de salida.

Tabla 3-1 Diferencias y similitudes entre los procedimientos y funciones SQL.

Trabajar con la sintaxis básica

Existen muchas similitudes entre la sintaxis que se usa para crear procedimientos y la que se usa para crear funciones. De hecho, están definidas como un solo elemento sintáctico en SQL:2006. Además, la sintaxis es, en su nivel más básico, similar a como se crean los procedimientos en la mayoría de las implementaciones SQL. Demos un vistazo a la sintaxis de cada uno para comprender mejor sus elementos básicos.

Utilizar la instrucción CREATE PROCEDURE

La primera sintaxis que veremos es aquella que sirve para crear un procedimiento. En su nivel más básico, la instrucción CREATE PROCEDURE luce de la siguiente manera:

```
CREATE PROCEDURE <nombre del procedimiento>
( [ <declaración de parámetro> [ { , <declaración de parámetro> } ... ] ] )
[ <característica de la rutina>... ]
<cuerpo de la rutina>
```

Como se puede ver, debe proporcionarse un nombre para el procedimiento (en la cláusula CREATE PROCEDURE) seguido por cero o más declaraciones de parámetro, que van encerradas en paréntesis. Se deben proporcionar los paréntesis incluso si no se define ninguna declaración. Después de las declaraciones de parámetro se tiene la opción de definir una o más características de la rutina. Por ejemplo, se puede especificar si una rutina es SQL o una escrita en otro lenguaje, por ejemplo C o Java.

NOTA

El tipo de características de rutina que se puede definir varía ampliamente entre las implementaciones SQL, no solamente en términos de cuáles opciones son soportadas, sino también acerca de cómo están definidas. Consecuentemente, el análisis sobre estas opciones se mantendrá corto, por lo que deberá asegurarse de revisar la documentación del producto para mayor información. Por ejemplo, las extensiones de procedimiento en Oracle están definidas utilizando un lenguaje que Oracle llama PL/SQL, mientras que en SQL Server y Sybase, las extensiones de procedimiento son parte de un lenguaje llamado Transact-SQL; ambos lenguajes son significativamente diferentes del estándar SQL. Por otro lado, MySQL y DB2 generalmente acatan el estándar SQL al definir funciones y procedimientos almacenados.

Después de haber definido las características del procedimiento, todo está listo para agregar las instrucciones SQL, que están representadas por el marcador de posición <cuerpo de la rutina>. Muchas de las instrucciones que se utilizarán en esta sección serán similares a aquellas que ya se han visto en este libro. Sin embargo, el estándar SQL/PSM introdujo nuevos elementos del lenguaje que hicieron más dinámicos los procedimientos. Mientras continuamos a través de este capítulo, se verán muchos de estos elementos y cómo son utilizados para extender la funcionalidad de los procedimientos invocados por SQL.

Utilizar la instrucción CREATE FUNCTION

Ahora demos un vistazo a la instrucción utilizada para crear una función invocada por SQL. Como se puede ver en la siguiente sintaxis, una función contiene unos cuantos elementos más que un procedimiento:

```
CREATE FUNCTION <nombre de la función>
( [ <declaración de parámetro> [ { , <declaración de parámetro> } ... ] ] )
RETURNS <tipo de datos>
[ <característica de la rutina> . . . ]
[ STATIC DISPATCH ]
<cuerpo de la rutina>
```

Al igual que con los procedimientos, primero se debe proporcionar un nombre a la función, seguido por la lista de declaraciones de parámetro. Las funciones soportan solamente parámetros entrada, y si no se proporciona ninguno, deberán utilizarse los paréntesis de todas formas. Si se proporciona más de un parámetro de entrada, deberán separarse utilizando comas. Siguiendo a las declaraciones de parámetro está la cláusula RETURNS. Se debe proporcionar el tipo de datos para el valor que es arrojado por la función. Después de eso, se pueden incluir cualquiera de las características opcionales de rutina, dependiendo de qué opciones soporte la implementación SQL. Luego está la cláusula STATIC DISPATCH. Se debe especificar esta cláusula si se utiliza un tipo definido por el usuario, un tipo de datos de referencia, o un tipo de datos de arreglo. Debido a que todos estos tipos salen del campo de acción de este libro, por el momento no es necesario preocuparse acerca de la cláusula STATIC DISPATCH.

Lo último que se debe incluir en la definición del procedimiento es, desde luego, el cuerpo de la rutina. Al igual que con los procedimientos, éstas son las instrucciones SQL que componen el núcleo del procedimiento. Sin embargo, hay un elemento adicional que se puede encontrar en el cuerpo de la rutina y que no está incluido en el cuerpo de la rutina del procedimiento (una instrucción RETURN, que no debe ser confundida con la cláusula RETURNS). La instrucción RETURN especifica el valor que será arrojado por la función. Posteriormente en este capítulo, en la sección “Crear funciones invocadas por SQL”, se discutirán más a detalle la instrucción RETURN y otros elementos de la instrucción CREATE FUNCTION.

Crear procedimientos invocados por SQL

Ahora que se tiene una idea general de las rutinas invocadas por SQL y de la sintaxis utilizada para crearlas, demos un vistazo más cercano a cómo crear los procedimientos invocados por SQL. Un procedimiento puede realizar la mayoría de funciones que se pueden realizar utilizando directamente las instrucciones SQL. Además, los procedimientos pueden utilizarse para pasar parámetros

INVENTARIO_CD			TIPOS_CD	
TITULO_CD: VARCHAR(60)	ID_TIPO_CD: CHAR(4)	EXISTENCIA_CD: INT	ID_TIPO_CD: CHAR(4)	NOMBRE_TIPO_CD: CHAR(20)
Famous Blue Raincoat	FROK	19	FROK	Folk Rock
Blue	CPOP	28	CPOP	Classic Pop
Past Light	NEWA	6	NEWA	New Age
Out of Africa	STRK	8	CTRY	Country
Fundamental	NPOP	10	STRK	Soundtrack
Blues on the Bayou	BLUS	11	BLUS	Blues
Kojiki	NEWA	10	JAZZ	Jazz

Figura 13-1 Utilizar procedimientos para acceder a las tablas INVENTARIO_CD y TIPOS_CD.

y definir variables, algo en que nos adentraremos posteriormente en este capítulo. Por ahora, veamos un procedimiento desde su nivel más básico, uno que no incluya parámetros o tipos especiales de instrucciones SQL.

Supongamos que se necesitan consultar los datos en las tablas INVENTARIO_CD y TIPOS_CD mostrados en la figura 13-1. Se requiere que los resultados de la consulta arrojen los nombres de los CD y el número en existencia para todos los CD de New Age.

Para ver esta información, se puede crear una instrucción SELECT que una las dos tablas, como se muestra en el siguiente ejemplo:

```
SELECT TITULO_CD, EXISTENCIA_CD
  FROM INVENTARIO_CD i, TIPOS_CD t
 WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
       AND NOMBRE_TIPO_CD = 'New Age';
```

Desde luego, cada vez que se quiera ver esta información, se tendrá que volver a crear la instrucción SELECT. Sin embargo, otra opción es almacenar la instrucción SELECT dentro del esquema. De esta manera, todo lo que se necesita hacer es convocar la instrucción cada vez que se quiera ver la información de los CD de New Age. Una forma de almacenar la instrucción SELECT es dentro de una definición de vista:

```
CREATE VIEW NEW_AGE AS
  SELECT TITULO_CD, EXISTENCIA_CD
    FROM INVENTARIO_CD i, TIPOS_CD t
   WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
        AND NOMBRE_TIPO_CD = 'New Age';
```

Una vez que se ha creado la vista, se puede utilizar una instrucción SELECT para convocar la vista, como se muestra en la siguiente instrucción:

```
SELECT * FROM NEW_AGE;
```

Sin embargo, las vistas son muy limitadas respecto a los tipos de instrucción y a la funcionalidad que soportan. Por ejemplo, no se puede incluir una instrucción UPDATE en una vista, ni se pueden pasar parámetros desde y/o hacia las vistas. Como resultado, una mejor forma de almacenar esta instrucción SELECT es como un procedimiento invocado por SQL. Para hacer esto, se debe crear un objeto de esquema utilizando la instrucción CREATE PROCEDURE, como se muestra en el siguiente ejemplo:

```
CREATE PROCEDURE NEW_AGE_CDS ( )
SELECT TITULO_CD, EXISTENCIA_CD
FROM INVENTARIO_CD i, TIPOS_CD t
WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
AND NOMBRE_TIPO_CD = 'New Age';
```

Esta instrucción representa la cantidad mínima de información que se debe proporcionar para crear un procedimiento. Incluye una cláusula CREATE PROCEDURE que le da nombre al procedimiento (CDS_NEW_AGE), un conjunto de paréntesis y un cuerpo de la rutina, que es la instrucción SELECT. Si se fueran a definir parámetros, sus instrucciones estarían encerradas en los paréntesis.

Como bien puede imaginarse, una instrucción CREATE PROCEDURE puede ser mucho más compleja de lo que se ve aquí. Sin embargo, la instrucción en el ejemplo representa la estructura básica sobre la cual pueden construirse instrucciones más extensas. Antes de analizar procedimientos más complicados, primero toquemos el tema de cómo se crea esta instrucción en diferentes implementaciones SQL.

Anteriormente en el capítulo se dijo que las implementaciones SQL pueden variar ampliamente respecto a los puntos específicos de cómo se crean y se convocan las rutinas invocadas por SQL. Como resultado, pocas implementaciones soportan el SQL puro cuando intentan definir los procedimientos. Por ejemplo, tanto SQL Server como Oracle requieren que se utilice la palabra clave AS antes del cuerpo de la rutina. Adicionalmente, SQL Server no utiliza paréntesis después del nombre del procedimiento, sin importar si los parámetros están siendo definidos o no. Oracle, por otro lado, sí utiliza los paréntesis, y también requiere algunas instrucciones adicionales que encierran instrucciones ejecutables en bloques BEGIN...END. Como se mencionó anteriormente, MySQL y DB2 acatan más cercanamente el estándar SQL. Con esto debe estar claro que usted simplemente *debe* consultar la documentación de su producto cada vez que se esté creando un procedimiento para determinar cómo difiere el lenguaje específico de ese producto del estándar SQL.

Invocar procedimientos invocados por SQL

Una vez que se ha creado el procedimiento, puede invocarse (o convocarse) utilizando una instrucción CALL. La sintaxis básica para la instrucción CALL es la siguiente:

```
CALL <nombre del procedimiento>
([ <valor> [ { , <valor> } ... ] ])
```

Como se puede ver, debe identificarse el nombre del procedimiento en la cláusula CALL y seguirla con los valores (en paréntesis) que pasan al procedimiento como parámetros. Los paréntesis deben utilizarse incluso si ningún parámetro es definido para el procedimiento. Si más de un pará-

metro es definido para el procedimiento, deben separarse por comas. Además, deben seguirse estos lineamientos cuando se ingresen los valores:

- La instrucción CALL deberá incluir el mismo número de valores que el número de parámetros definidos en el procedimiento.
- Los valores deberán ser ingresados en el mismo orden en que fueron definidos en el procedimiento.
- Los valores deben ajustarse a los tipos de datos que están asignados a los parámetros.

Se analizarán los parámetros con más detalle en la siguiente sección, “Agregar parámetros de entrada a sus procedimientos”.

Ahora veamos un ejemplo de la instrucción CALL. Si se quiere convocar el procedimiento que fue creado en el ejemplo anterior, se puede utilizar la siguiente instrucción:

```
CALL CDS_NEW_AGE ( );
```

En esta instrucción, el nombre del procedimiento sigue a la palabra clave CALL. Nótese el uso de paréntesis a pesar de que ningún parámetro fue definido para el procedimiento. Si algún parámetro hubiera sido definido, estaría encerrado en los paréntesis. Cuando se ejecuta esta instrucción, se arrojarán los mismos resultados que se hubieran obtenido si se hubiera ejecutado la instrucción SELECT separadamente, como se muestra en los siguientes resultados de la consulta:

TITULO_CD	EXISTENCIA_CD
-----	-----
Past Light	6
Kojiki	10

La instrucción CALL, al igual que la instrucción CREATE PROCEDURE, pueden variar de implementación a implementación SQL en cómo se utilizan y si son soportadas o no. De hecho, probablemente encontrará que, para la mayoría de las implementaciones, se debe utilizar una instrucción EXECUTE, en lugar de CALL, para invocar un procedimiento.

Agregar parámetros de entrada a sus procedimientos

El procedimiento CDS_NEW_AGE que se vio en los ejemplos anteriores puede ser de mucha utilidad porque evita que se tenga que crear una instrucción SQL cada vez que se quiera ver la información acerca de los CD de New Age. Sin embargo, para poder arrojar información acerca de otros tipos de CD, por ejemplo Blues o Country, se debe crear una nueva consulta o estructurar un procedimiento para el tipo específico de música. Pero existe otra alternativa. Se puede crear un procedimiento que no defina específicamente el tipo de música sino que en su lugar permita al usuario ingresar el tipo cada vez que quiera convocar ese procedimiento. De esa manera, solamente se necesita un procedimiento para revisar cualquier tipo deseado de música.

Para soportar este tipo de procedimiento se debe declarar un parámetro dentro de la definición del procedimiento que permita al procedimiento aceptar valores de entrada cuando sea convocado. Regresemos a las tablas INVENTARIO_CD y TIPOS_CD mostradas en la figura 13-1. Si modificamos el lenguaje del procedimiento que se creó anteriormente, podemos crear un nuevo procedi-

miento que incluya el parámetro de entrada necesario, como se muestra en la siguiente instrucción CREATE PROCEDURE:

```
CREATE PROCEDURE CDS_POR_TIPO ( IN p_TIPO_CD CHAR(20) )
    SELECT TITULO_CD, EXISTENCIA_CD
    FROM INVENTARIO_CD i, TIPOS_CD t
    WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
    AND NOMBRE_TIPO_CD = p_TIPO_CD;
```

En la primera línea de código se define un parámetro después de la cláusula CREATE PROCEDURE. La instrucción del parámetro incluye la palabra clave IN, el nombre del parámetro (p_TIPO_CD) y el tipo de datos para ese parámetro (CHAR(20)), y todos ellos están encerrados en paréntesis.

NOTA

La convención “p_” utilizada para nombrar los parámetros no es obligatoria. Sin embargo, resulta más fácil utilizar algún tipo de convención de nombrado para separar los parámetros, haciéndolos más fáciles de escoger en el código.

SQL soporta tres tipos de parámetros: de entrada, de salida y de entrada/salida. Los tres tipos se representan por las palabras clave de modo de parámetro IN, OUT e INOUT, respectivamente. Los parámetros de entrada permiten proporcionar valores cuando se invoca un procedimiento. Esos valores son después utilizados dentro del cuerpo de la rutina cuando se ejecutan las instrucciones SQL. Los parámetros de salida permiten al procedimiento proporcionar valores como un resultado de invocar el procedimiento. Los parámetros de entrada/salida son aquellos que proporcionan la funcionalidad de parámetros tanto de entrada como de salida. No se necesita especificar alguna de las palabras clave de modo de parámetro cuando se definan los parámetros. Sin embargo, si no se especifica una de las palabras clave, SQL asume que se está definiendo un parámetro de entrada.

NOTA

Al igual que con muchos otros aspectos de la instrucción CREATE PROCEDURE, las declaraciones de parámetro pueden variar de producto a producto. En SQL Server, por ejemplo, los nombres de parámetro deben ser precedidos del símbolo arroba (@), como en @p_Tipo_CD; las declaraciones de parámetro no están encerradas en paréntesis, y no se utiliza la palabra clave IN. Oracle, por otro lado, no requiere el símbolo arroba y no utiliza los paréntesis. Oracle también utiliza la palabra clave IN, pero se posiciona después del nombre del parámetro, como en p_CD_Type IN CHAR(20).

Ahora regresemos al procedimiento CDS_POR_TIPO que está definido en la instrucción anterior CREATE PROCEDURE. Una vez que se define el parámetro de entrada, se querrá utilizarlo de alguna forma significativa dentro del cuerpo de la rutina. En este caso, el parámetro p_TIPO_CD se utiliza en el segundo predicado en la cláusula WHERE (NOMBRE_TIPO_CD = p_TIPO_CD). Esto significa que el valor que se ingresa cuando se invoca el procedimiento es comparado con los valores NOMBRE_TIPO_CD de la tabla TIPOS_CD cuando se ejecuta la instrucción SELECT. En consecuencia, los resultados de la consulta incluirán la información de CD acerca del tipo de música especificado.

Una vez que se ha creado el procedimiento, puede invocarlo utilizando una instrucción `CALL` que especifique un valor para el parámetro. Por ejemplo, si se quiere arrojar información de los CD de Folk Rock, se puede utilizar la siguiente instrucción `CALL`:

```
CALL CDS_POR_TIPO('Folk Rock');
```

Nótese que se incluye el valor para el parámetro en paréntesis después del nombre del procedimiento. El valor debe obedecer al tipo de datos asignado al parámetro, el cual en este caso es `CHAR(20)`. Al igual que con cualquier otra instancia en la que se esté trabajando con valores de cadenas de caracteres, se debe cerrar el valor en comillas sencillas. Cuando se invoca este procedimiento, el valor Folk Rock se inserta al predicado en la cláusula `WHERE` y el procedimiento arroja los siguientes resultados de la consulta:

TITULO_CD	EXISTENCIA_CD
-----	-----
Famous Blue Raincoat	19

Como puede verse, ahora se tiene un procedimiento que puede usarse para arrojar la información de CD de cualquier tipo de música. Simplemente se proporciona el nombre del tipo de música cuando se convoca el procedimiento. Sin embargo, los procedimientos no están limitados a solamente un parámetro. Se pueden incluir múltiples parámetros en cualquier definición de procedimiento. Por ejemplo, supongamos que se quiere modificar la definición de procedimiento anterior para permitir ingresar una cantidad. Se quiere utilizar esa cantidad para arrojar la información de CD solamente para aquellos CD con un valor `EXISTENCIA_CD` que exceda la cantidad especificada. Al mismo tiempo, todavía se quiere arrojar la información de CD solamente para los tipos de música especificados. Como resultado, se necesita definir dos parámetros, como se muestra en la siguiente instrucción `CREATE PROCEDURE`:

```
CREATE PROCEDURE CDS_POR_TIPO ( IN p_Tipo_CD CHAR(20), IN p_Cantidad INT )
SELECT TITULO_CD, EXISTENCIA_CD
FROM INVENTARIO_CD i, TIPOS_CD t
WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
AND NOMBRE_TIPO_CD = p_Tipo_CD
AND EXISTENCIA_CD > p_Cantidad;
```

Nótese que la cláusula de instrucción de parámetro ahora incluye dos parámetros de entrada: `p_Tipo_CD` y `p_Cantidad`. El parámetro `p_Cantidad` se configura con el tipo de datos `INT`. El parámetro `p_Cantidad`, al igual que el parámetro `p_Tipo_CD`, se utiliza en un predicado en la cláusula `WHERE` (`EXISTENCIA_CD > p_Cantidad`). Como resultado, las filas arrojadas por el procedimiento deben incluir valores `EXISTENCIA_CD` mayores a la cantidad especificada que cuando se convocó el procedimiento.

Una vez que se ha creado el procedimiento, puede ser convocado utilizando una instrucción `CALL` que incluya valores para ambos parámetros, como se muestra en el siguiente ejemplo:

```
CALL CDS_POR_TIPO('New Age', 5);
```

Ahora la instrucción `CALL` incluye dos valores (separados por una coma) dentro de los paréntesis. Los valores deben ser enlistados en el mismo orden en el que los parámetros estén definidos en la instrucción `CREATE PROCEDURE`. Cuando se invoca esta instrucción, se asigna el valor New Age al parámetro `p_Tipo_CD`, y el valor 5 al parámetro `p_Cantidad`, haciendo que la instrucción

SELECT incrustada en la definición del procedimiento se comporte como si se ingresaran los valores directamente, como se muestra en el siguiente ejemplo:

```
SELECT TITULO_CD, EXISTENCIA_CD
FROM INVENTARIO_CD i, TIPOS_CD t
WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
      AND NOMBRE_TIPO_CD = 'New Age'
      AND EXISTENCIA_CD > 5;
```

Si se ejecutara esta instrucción, se arrojarían los mismos resultados que si se hubiera ejecutado la instrucción CALL utilizando el valor New Age y el valor 5, como se muestra en el siguiente resultado:

TITULO_CD	EXISTENCIA_CD
-----	-----
Past Light	6
Kojiki	10

Ahora modificamos la instrucción CALL para ver cómo el especificar un valor diferente pudiera afectar los resultados. Supongamos que se utiliza un valor numérico de 8 en lugar de 5, como se muestra en la siguiente instrucción:

```
CALL CDS_POR_TIPO ('New Age', 8);
```

Si se ejecutara esta instrucción, solamente una fila sería arrojada:

TITULO_CD	EXISTENCIA_CD
-----	-----
Kojiki	10

Si regresamos a la tabla INVENTARIO_CD en la figura 13-1, se verá que únicamente la fila Kojiki es un CD de New Age con un valor EXISTENCIA_CD que rebasa la cantidad de 8, el valor que se especificó en la instrucción CALL. Como se puede ver, utilizar múltiples parámetros puede proporcionar una variedad de opciones para convertir a los procedimientos en una herramienta útil y flexible que puede eliminar la necesidad de escribir múltiples instrucciones que se usaban para lograr resultados similares. Si se definen los parámetros necesarios, simplemente se tendrán que añadir los valores necesarios para lograr los resultados que se deseen.

Utilizar procedimientos para modificar datos

Hasta este punto, los procedimientos invocados por SQL que se han visto han contenido instrucciones SELECT que consultan datos. Sin embargo, los procedimientos no están limitados a solamente las instrucciones SELECT. Se pueden incluir instrucciones de modificación de datos tales como INSERT, UPDATE y DELETE. Regresemos a las tablas INVENTARIO_CD y TIPOS_CD, mostradas en la figura 13-1. Puede notarse que la tabla INVENTARIO_CD incluye una fila para el CD Fundamental. El tipo de música para ese CD es New Pop, que se representa por NPOP (el valor en la columna ID_TIPO_CD). También puede haberse notado que no hay una entrada correspondiente en la tabla TIPOS_CD para el tipo New Pop. Se puede crear un procedimiento que permita insertar va-

lores a esa tabla. Simplemente se necesita definir ese procedimiento con los parámetros apropiados de entrada y la instrucción INSERT, como se muestran en el siguiente ejemplo:

```
CREATE PROCEDURE INSERTAR_TIPO( IN p_Tipo CHAR(4), IN p_Nombre CHAR(20) )  
    INSERT INTO TIPOS_CD VALUES ( p_Tipo, p_Nombre );
```

Nótese que la definición del procedimiento incluye dos parámetros de entrada: p_Tipo y p_Nombre, y que ambos están definidos con el tipo de datos CHAR. Estos parámetros son por lo tanto utilizados en la instrucción INSERT, de la misma forma en la cual se hubieran especificado normalmente los valores a ser insertados en una tabla. Cualquier parámetro que se declare para este propósito debe ser definido con un tipo de datos que sea compatible con el tipo de datos definido en la columna que contenga los datos a ser modificados. Una vez que se crea el procedimiento, se puede utilizar una instrucción CALL similar a la del siguiente ejemplo para invocar el procedimiento:

```
CALL INSERTAR_TIPO('NPOP', 'New Pop');
```

Nótese que la instrucción CALL incluye los valores NPOP y New Pop. Estos valores se pasan a los dos parámetros definidos en el procedimiento e INSERTAR_TIPO. Como resultado, son insertados en la tabla TIPOS_CD como si se hubiera ejecutado la instrucción INSERT directamente.

De la misma forma en que se creó el procedimiento INSERTAR_TIPO, se pueden crear procedimientos para actualizar y eliminar datos incluyendo la instrucción apropiada UPDATE y DELETE, en lugar de la instrucción INSERT. Simplemente tendrá que crear los parámetros de entrada necesarios y asignar los valores apropiados a esos parámetros cuando se convoque el procedimiento. Sin embargo, tenga en mente que el valor que se pase utilizando los parámetros debe acatar no solamente los tipos de datos definidos en las instrucciones de parámetro, sino también los tipos de datos y restricciones en las columnas que contienen los datos que se están tratando de modificar.

Pregunta al experto

P: Hasta este punto se nos ha mostrado cómo crear procedimientos invocados por SQL, pero no cómo modificarlos. ¿Existe alguna forma de alterar o eliminar los procedimientos?

R: El estándar SQL soporta tanto a la instrucción ALTER PROCEDURE como a la instrucción DROP PROCEDURE. La instrucción ALTER PROCEDURE permite alterar algunas de las características de rutina del procedimiento, pero no permite alterar el cuerpo del procedimiento. Sin embargo, la funcionalidad soportada por la instrucción ALTER PROCEDURE puede variar tan ampliamente entre una implementación SQL y otra que se necesitará revisar la documentación del producto para ver si es que la instrucción es soportada y qué se puede hacer con esa instrucción. En SQL Server, por ejemplo, la instrucción ALTER PROCEDURE permite modificar la mayoría de los aspectos del procedimiento, mientras que la misma instrucción en Oracle es utilizada principalmente para recopilar el procedimiento y evitar así la compilación en tiempo real (que también evita la sobrecarga en tiempo real). Sin embargo, Oracle proporciona la sintaxis CREATE OR REPLACE PROCEDURE que reemplaza a un

procedimiento existente (incluyendo el cuerpo del procedimiento) o crea un nuevo procedimiento si éste no existía aún.

En el caso de la instrucción `DROP PROCEDURE`, es soportada por la mayoría de las implementaciones y por lo general es muy sencilla de utilizar. Simplemente se proporciona el nombre del procedimiento en la instrucción y, dependiendo de la implementación SQL, las palabras clave `RESTRICT` o `CASCADE`, como se ha visto su uso en otras instrucciones `DROP`. Nótese que resulta lo mismo para las instrucciones `ALTER FUNCTION` y `DROP FUNCTION`. A pesar de ser soportada por muchas implementaciones, la instrucción `ALTER FUNCTION` puede variar entre un producto y otro, y la instrucción `DROP FUNCTION` es prácticamente similar entre las diferentes implementaciones.

Agregar variables locales a sus procedimientos

Además de permitir pasar los parámetros a un procedimiento, SQL también proporciona una forma para crear variables locales en la definición del procedimiento que puedan ser utilizadas dentro del cuerpo del procedimiento. Una *variable local* puede ser considerada como un tipo de marcador de posición que mantiene un valor en la memoria durante la ejecución de las instrucciones en el cuerpo de la rutina. Una vez que las instrucciones han sido ejecutadas, la variable deja de existir.

Cuando se define una variable local, se debe declarar primero la variable y luego determinar un valor inicial para ella. Entonces ya se puede utilizar esa variable en el resto del bloque de instrucciones. La sintaxis básica para definir una variable es la siguiente:

```
DECLARE <nombre de la variable> <tipo de datos>;
```

Como se puede ver, la sintaxis es muy sencilla. Se debe proporcionar un nombre para la variable y asignar un tipo de datos. Una vez que se ha declarado la variable, debe asignársele un valor antes de que pueda ser referenciada. (Sin embargo, algunas implementaciones asignan automáticamente un valor nulo a las variables en el momento de ser definidas.) Se puede utilizar la instrucción `SET` para asignar un valor a una variable utilizando la siguiente sintaxis:

```
SET <nombre de la variable> = <expresión de valor>;
```

En esta instrucción se debe proporcionar primero el nombre de la variable y luego el valor, que puede ser cualquier tipo de expresión de valor, como un número, una cadena de caracteres o una subconsulta.

Después de que se ha declarado la variable y se le ha asignado un valor, todo está listo para utilizar la variable en el cuerpo de la rutina. La mejor forma de ilustrar esto es mostrando un ejemplo de un procedimiento que utilice la variable. Por ejemplo, utilizaremos una vez más la tabla `INVENTARIO_CD`, mostrada en la figura 13-1. La siguiente instrucción crea un procedimiento que recupera la información de CD para un tipo de música en específico:

```
CREATE PROCEDURE CANTIDAD_CD ( IN p_ID_Tipo CHAR(4) )
BEGIN
    DECLARE v_Cantidad INT;
    SET v_Cantidad = ( SELECT AVG(EXISTENCIA_CD)
                      FROM INVENTARIO_CD );
```

```
SELECT TITULO_CD, EXISTENCIA_CD
FROM INVENTARIO_CD
WHERE ID_TIPO_CD = p_ID_Tipo
AND EXISTENCIA_CD < v_Cantidad;
END;
```

Vayamos a través de esta instrucción línea por línea. En la primera línea se crea un procedimiento llamado CD_AMOUNT y un parámetro de entrada llamado p_ID_Tipo. La segunda línea contiene la palabra clave BEGIN. La palabra clave BEGIN es emparejada con la palabra clave END en la última línea. Juntas encierran un bloque de instrucciones que son procesadas como una unidad. Daremos un vistazo más cercano al bloque BEGIN...END posteriormente en la sección “Trabajar con instrucciones de control”.

La tercera línea de la definición del procedimiento incluye una instrucción DECLARE que declara la variable v_Cantidad, que se define con el tipo de datos INT. Las dos nuevas líneas utilizan una instrucción SET para asignar un valor inicial al parámetro. Este valor se deriva de una subconsulta que encuentra el promedio para todos los valores EXISTENCIA_CD. En este caso el promedio es cercano a 13. En las siguientes cuatro líneas de la definición del procedimiento, una instrucción SELECT recupera datos de la tabla INVENTARIO_CD basados en los valores proporcionados por el parámetro y la variable.

Una vez que se ha creado el procedimiento, puede ejecutarse utilizando una instrucción CALL y proporcionando un valor para el parámetro, como se muestran en el siguiente ejemplo:

```
CALL CANTIDAD_CD ( 'NEWA' );
```

Cuando se procesa el procedimiento, utiliza el valor NEWA del parámetro y el promedio EXISTENCIA_CD de la variable en la instrucción SELECT definida en la definición del procedimiento. Lo anterior sería similar a ejecutar la siguiente instrucción:

```
SELECT TITULO_CD, EXISTENCIA_CD
FROM INVENTARIO_CD
WHERE ID_TIPO_CD = 'NEWA'
AND EXISTENCIA_CD < 13;
```

Esta instrucción SELECT, al igual que el procedimiento en sí mismo, arrojará los siguientes resultados de la consulta:

TITULO_CD	EXISTENCIA_CD
Past Light	6
Kojiki	10

Nótese que ambas filas contienen valores EXISTENCIA_CD menores que la cantidad promedio (13) y ambos son CD de New Age.

En una definición de procedimiento no se está limitado solamente a una variable. Se puede crear una instrucción DECLARE para cada variable que se quiera incluir. También es posible incluir múltiples variables en una instrucción, si esas variables están asignadas al mismo tipo de datos. Por ejemplo, supongamos que se desea declarar muchas variables con un tipo de datos INT, como se muestra en la siguiente instrucción DECLARE:

```
DECLARE Var1, Var2, Var3 INT;
```

Esta instrucción declara las variables Var 1, Var 2 y Var 3, y cada una es asignada al tipo de datos INT. Una vez que se asignan valores iniciales a las variables, se les puede utilizar en el cuerpo de la rutina de la misma forma que cualquier otra variable local.

Trabajar con instrucciones de control

Cuando se liberó el estándar SQL/PSM en 1996, no sólo incluía lenguaje que soportaba rutinas invocadas por SQL, sino también lenguaje que pudiera ser utilizado dentro de esas rutinas para hacerlas más robustas. Tales características como agrupar instrucciones en bloques y repetir instrucciones para que pudieran ser ejecutadas múltiples veces (comportamiento tradicionalmente asociado con lenguajes del tipo de procedimientos) hacía a los procedimientos y funciones mucho más valiosos para los usuarios que necesitaban acceder y manipular datos en sus bases de datos. El estándar SQL:2006 se refería a estos elementos del lenguaje como *instrucciones de control* debido a que éstas afectaban la manera como se pueden controlar los datos en las rutinas invocadas por SQL. En esta sección se verán muchas de estas instrucciones de control, incluyendo aquellas que permiten agrupar instrucciones en un bloque, crear instrucciones condicionales y establecer instrucciones dentro de un bucle de repetición.

Crear instrucciones compuestas

La más básica de las instrucciones de control es la instrucción compuesta, que permite agrupar instrucciones en un bloque. La instrucción compuesta inicia con la palabra clave BEGIN y termina con la palabra clave END. Todo lo que esté entre estas dos palabras es parte del bloque. La instrucción compuesta está conformada por una o más instrucciones SQL individuales, que pueden incluir instrucciones como DECLARE, SET, SELECT, UPDATE, INSERT, DELETE u otras instrucciones de control.

Ya se ha presentado un ejemplo de una instrucción compuesta en la instrucción CREATE PROCEDURE anterior que define el procedimiento CANTIDAD_CD. (Éste es el ejemplo mostrado en la sección “Agregar variables locales a sus procedimientos”). Si miramos otra vez este ejemplo, se observará que la definición del procedimiento incluye una instrucción compuesta. Como se pudiera esperar, inicia con la palabra clave BEGIN y termina con la palabra clave END. El bloque creado por estas palabras clave incluye una instrucción DECLARE, una instrucción SET y una instrucción SELECT. Nótese que cada instrucción es terminada con punto y coma. A pesar de que la instrucción BEGIN...END es considerada una sola instrucción, las instrucciones encerradas en estas palabras clave son instrucciones individuales por sí mismas.

NOTA

En algunas implementaciones SQL, la instrucción compuesta pudiera no ser necesaria bajo ciertas circunstancias. En estos casos, el terminador punto y coma pudiera ser suficiente como señal para la implementación de que una instrucción ha terminado y otra ha empezado. Incluso en aquellas implementaciones que no requieren el punto y la coma, como en SQL Server, a veces se procesarán múltiples instrucciones como un bloque incluso sin haber utilizado la construcción BEGIN...END. Cuando la implementación alcanza el final de una instrucción, simplemente continúa con la siguiente. Sin embargo, como una regla general, deberá utilizarse la construcción compuesta para mantener juntas aquellas instrucciones que deberán ser procesadas como una unidad. Cuando no se utilice, dependiendo de la implementación, a veces se puede experimentar un comportamiento imprevisible.

Se puede utilizar la instrucción compuesta en cualquier parte donde se necesite mantener las instrucciones SQL juntas. Esto significa que pueden ser incrustadas dentro de otras instrucciones compuestas o dentro de otros tipos de instrucciones de control. Las palabras clave BEGIN y END no afectan la manera en que los datos pueden pasarse de una instrucción a la siguiente, como en el caso de los parámetros.

El aspecto positivo acerca de las instrucciones compuestas y la construcción BEGIN...END es que ambas son soportadas por la mayoría de las implementaciones SQL, a pesar de que puede haber ligeras variaciones de un producto a otro, en términos de los aspectos específicos de cómo sean implementadas. Asegúrese de revisar la documentación del producto cuando se utilicen estas instrucciones.

Crear instrucciones condicionales

El siguiente tipo de instrucción de control que veremos es la instrucción condicional. Esta instrucción determina si una instrucción (o una serie de instrucciones) se ejecuta basada en si una condición específica se evalúa como verdadera. La instrucción utiliza las palabras clave IF, THEN y ELSE para establecer las condiciones y definir las acciones a tomar: *si (if)* se cumple la condición, *entonces (then)* se ejecuta la instrucción SQL, o se toma *alguna otra (else)* acción.

NOTA

La instrucción condicional a veces toma a otros nombres, como instrucción IF, instrucción IF...ELSE, instrucción IF...END IF o instrucción IF...THEN...ELSE.

Demos un vistazo a un ejemplo que utilice una instrucción condicional para definir los cursos diferentes de acción, dependiendo de la condición. En la siguiente definición de procedimiento se modificó el cuerpo de la rutina del procedimiento CANTIDAD_CD (que utilizamos en el ejemplo anterior) para incluir una instrucción condicional:

```
CREATE PROCEDURE CANTIDAD_CD ( IN p_ID_Tipo CHAR (4) )
BEGIN
    DECLARE v_Cantidad INT;
    SET v_Cantidad = ( SELECT SUM(EXISTENCIA_CD)
                      FROM INVENTARIO_CD
                      WHERE ID_TIPO_CD = p_ID_Tipo );
    IF v_Cantidad < 20 THEN
        SELECT TITULO_CD, EXISTENCIA_CD
        FROM INVENTARIO_CD
        WHERE ID_TIPO_CD = p_ID_Tipo;
    ELSE
        SELECT TITULO_CD, EXISTENCIA_CD
        FROM INVENTARIO_CD;
    END IF;
END;
```

Nótese que el bloque BEGIN...END ahora incluye una instrucción IF...END IF. La cláusula IF introduce la instrucción y establece la condición. Para que la condición se evalúe como verdadera, el valor de la variable v_Cantidad debe ser menor a 20. Si la condición se evalúa como verdadera,

se ejecuta la primera instrucción SELECT. Ésta es la instrucción SELECT que sigue a la palabra clave THEN. Si la condición se evalúa como falsa, entonces se ejecuta la segunda instrucción SELECT. Ésta es la instrucción que sigue a la palabra clave ELSE. Para resumir, si v_Cantidad es menor a 20, los valores TITULO_CD y EXISTENCIA_CD de la tabla INVENTARIO_CD son arrojados para aquellas filas que contengan el ID Tipo (la columna ID_TIPO_CD) especificada por el parámetro p_ID_Tipo. Si v_Cantidad no es menor a 20, los valores TITULO_CD y EXISTENCIA_CD serán arrojados para todas las filas en la tabla INVENTARIO_CD.

Una vez que se ha creado el procedimiento, puede invocarse utilizando una instrucción CALL, como se ha hecho en procedimientos previos. Por ejemplo, si se quieren arrojar los CD de New Age (NEWA), puede utilizarse la siguiente instrucción CALL:

```
CALL CANTIDAD_CD ('NEWA');
```

Esta instrucción arrojará ambas filas New Age: Past Light y Kojiki. Esto se debe a que el número total de CD de New Age (16) es menor a 20, por lo que se ejecuta la primera instrucción SELECT. Si se hubiera especificado la categoría Classic Pop (CPOP) cuando se invocó el procedimiento CANTIDAD_CD, todas las filas habrían sido arrojadas. Esto se debe a que el número total de CD de Classic Pop (28) es mayor a 20. Como resultado, si la condición IF no puede ser cumplida, será ejecutada la instrucción ELSE.

Si se quiere crear una instrucción condicional que incluya más de una instrucción SQL ya sea en la cláusula IF o en la cláusula ELSE, se pueden encerrar esas instrucciones en una instrucción de control. Por ejemplo, si agregamos la instrucción UPDATE a la condición en el ejemplo anterior y utilizamos una instrucción de control para encerrar las instrucciones UPDATE y SELECT, la definición del procedimiento luciría parecida a lo siguiente:

```
CREATE PROCEDURE CANTIDAD_CD ( IN p_ID_Tipo CHAR (4) )
BEGIN
    DECLARE v_Cantidad INT;
    SET v_Cantidad = ( SELECT SUM(EXISTENCIA_CD)
                      FROM INVENTARIO_CD
                      WHERE ID_TIPO_CD = p_ID_Tipo );
    IF v_Cantidad < 20 THEN
        BEGIN
            UPDATE INVENTARIO_CD
                SET EXISTENCIA_CD = EXISTENCIA_CD + 1
                WHERE ID_TIPO_CD = p_ID_Tipo;
            SELECT TITULO_CD, EXISTENCIA_CD
                FROM INVENTARIO_CD
                WHERE ID_TIPO_CD = p_ID_Tipo;
        END;
    ELSE
        SELECT * FROM INVENTARIO_CD;
    END IF;
END;
```

La instrucción compuesta agrupa las dos instrucciones dentro de un bloque de códigos. De esta manera, las tablas serán actualizadas y los resultados de la actualización serán desplegados en los resultados de la consulta.

Pregunta al experto

P: La instrucción de condición en el ejemplo anterior muestra solamente dos condiciones y cursos de acción: la condición/acción definida en la cláusula IF y la condición/acción definida en la cláusula ELSE. ¿Qué pasa si se quieren incluir más condiciones?

R: El estándar SQL:2006 soporta más de dos construcciones de condición/acción en una instrucción condicional. Si se necesitan más de dos, se trata a la cláusula IF y a la cláusula ELSE tal como se mostró en el ejemplo. Las condiciones adicionales se insertan entre las dos cláusulas añadiendo la cláusula ELSE IF o una cláusula ELSEIF. La sintaxis para esto sería como la siguiente:

```
IF <condición> THEN <acción>
ELSE IF <condición> THEN <acción>
ELSE <acción>
```

La forma exacta de implementar la tercera condición/acción dependerá de la implementación. Además, no todas las implementaciones soportan ELSEIF, y algunas utilizan la palabra clave ELSIF. Como siempre, asegúrese de referirse a la documentación de su producto.

Crear instrucciones de repetición

Ahora demos un vistazo a otro tipo de instrucción de control (la instrucción de repetición). En realidad, SQL soporta muchos tipos de instrucciones de repetición. Centraremos nuestra atención en dos de ellas: la instrucción LOOP y la instrucción WHILE, las cuales realizan funciones similares.

La instrucción LOOP utiliza las palabras clave LOOP y END LOOP para encerrar un bloque de instrucciones que se ejecutan repetitivamente hasta que el bucle de repetición sea explícitamente terminado, utilizando por lo general la palabra clave LEAVE. Oracle utiliza la palabra clave EXIT en lugar de la palabra clave LEAVE, y SQL Server no soporta la instrucción LOOP. Demos un vistazo a un ejemplo para ilustrar cómo funciona esto. Una vez más se utilizarán las tablas de la figura 13-1, así como una instrucción LOOP para actualizar la tabla INVENTARIO_CD.

NOTA

Si se creó y probó por completo el procedimiento CANTIDAD_CD del ejemplo anterior, se asume que la tabla INVENTARIO_CD ha regresado a su condición original mostrada en la figura 13-1 y que ningún dato ha sido modificado.

En la siguiente definición de procedimiento se incluye una instrucción LOOP que continúa para actualizar la columna EXISTENCIA_CD hasta que alcance una cantidad mayor a 14:

```
CREATE PROCEDURE ACTUALIZAR_EXISTENCIAS ( IN p_Titulo CHAR(20) )
BEGIN
    DECLARE v_Cantidad INT;
```

```

SET v_Cantidad = ( SELECT EXISTENCIA_CD
                    FROM INVENTARIO_CD
                    WHERE TITULO_CD = p_Titulo );

Loop1:
LOOP
    SET v_Cantidad = v_Cantidad + 1;
    UPDATE INVENTARIO_CD
        SET EXISTENCIA_CD = v_Cantidad
        WHERE TITULO_CD = p_Titulo;
    IF v_Cantidad > 14
        THEN LEAVE Loop1;
    END IF;
END LOOP;
END;
```

En esta instrucción se le asigna primero un nombre a la repetición (Loop1:), lo que a veces se denomina etiqueta de instrucción. Se deben incluir los dos puntos con el nombre la primera vez que se asigna. Después se crea el bloque de repetición, que empieza con la palabra clave LOOP y termina con las palabras clave END LOOP. Dentro del bloque están las instrucciones SET y UPDATE. Estas dos instrucciones se ejecutan hasta que termine el bucle de repetición. Nótese que el valor EXISTENCIA_CD aumenta en un incremento de 1 cada vez que se ejecutan las instrucciones en la repetición. Estas dos instrucciones están seguidas por una instrucción IF, que especifica la condición en la cual se termina el bucle de repetición. Si el valor para la variable v_Cantidad pasa de 14, entonces se termina la repetición (LEAVE Loop1). La instrucción IF es entonces finalizada con las palabras clave END IF.

NOTA

Si no se incluye la instrucción IF dentro del bucle de repetición (con el operador de terminación LEAVE), la repetición continuará para incrementar el valor EXISTENCIA_CD hasta que llene todo el espacio disponible o algún otro evento o finalice la operación. Éste es un error de programación común conocido como bucle de repetición infinito.

Se puede entonces invocar el procedimiento proporcionando el nombre del procedimiento y un valor para el parámetro. Por ejemplo, supongamos que se quiere actualizar la fila Fundamental en la tabla INVENTARIO_CD. Se puede invocar el procedimiento con la siguiente instrucción CALL:

```
CALL ACTUALIZAR_EXISTENCIAS('Fundamental');
```

Cuando se ejecuta el procedimiento, un valor de 1 es repetidamente agregado a la columna EXISTENCIA_CD hasta que el valor alcance 15, y luego el bucle de repetición es finalizado.

Se pueden recibir los mismos resultados más elegantemente utilizando la instrucción WHILE. En el siguiente ejemplo se modificó la definición de procedimiento ACTUALIZAR_EXISTENCIAS reemplazando la instrucción LOOP con una instrucción WHILE:

```

CREATE PROCEDURE ACTUALIZAR_EXISTENCIAS( IN p_Titulo CHAR(20) )
BEGIN
    DECLARE v_Cantidad INT;
```



```

SET v_Cantidad = ( SELECT EXISTENCIA_CD
                    FROM INVENTARIO_CD
                    WHERE TITULO_CD = p_Titulo );
WHILE v_Cantidad < 15 DO
    SET v_Cantidad = v_Cantidad + 1;
    UPDATE INVENTARIO_CD
        SET EXISTENCIA_CD = v_Cantidad
        WHERE TITULO_CD = p_Titulo;
END WHILE;
END;

```

NOTA

Una vez más, si se ejecutó por completo el procedimiento creado en el ejemplo anterior, se asume que la tabla ha regresado a su condición original mostrada en la figura 13-1 y que ningún dato ha sido modificado.

La instrucción WHILE establece el mismo tipo de condición de repetición que la instrucción LOOP. Sin embargo, en lugar de utilizar una instrucción IF para terminar el bucle, se especificó una condición en la cláusula WHILE que finaliza la repetición automáticamente cuando la condición se evalúa como falsa. En este caso, el valor de parámetro para v_Cantidad debe ser menor a 15 para que la condición WHILE se evalúe como verdadera. Mientras la condición se evalúa como verdadera, la instrucción SET y la instrucción UPDATE serán ejecutadas. Si la condición se evalúa como falsa, se finalizará la repetición WHILE. Nótese que muchas implementaciones, incluyendo Oracle y SQL Server, utilizan un bloque BEGIN en lugar de la palabra clave DO para encerrar las instrucciones que se van a repetir por el bucle de repetición WHILE. Un punto más en el que hay que poner atención es dónde se evalúa la condición en la lógica de repetición. Algunas implementaciones evalúan la condición al principio de la repetición. Otras la evalúan al final de la repetición, lo que significa que las instrucciones en la repetición siempre se ejecutarán por lo menos una vez, incluso si la condición se evalúa como falsa la primera vez que se inicia la repetición.

Pruebe esto 13-1

Crear procedimientos invocados por SQL

En este ejercicio aplicará todo lo que ha aprendido acerca de crear procedimientos invocados por SQL para la base de datos INVENTARIO. Deberá crear, invocar y eliminar procedimientos. Uno de los procedimientos deberá incluir el parámetro, y uno más deberá incluir una variable. Para este ejercicio, incluso más que para la mayoría de los demás ejercicios, necesitará referirse a la documentación del producto de su implementación SQL para asegurarse de tomar en cuenta las diferentes variaciones acerca de cómo un procedimiento es creado, convocado y eliminado. Como se mencionó anteriormente en este capítulo, la implementación del procedimiento puede variar ampliamente entre el estándar SQL y el producto individual. Puede descargar el archivo Try_This_13_1.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. El primer procedimiento que deberá crearse es uno muy básico que consulta la información de las tablas DISCOS_COMPACTOS, CDS_ARTISTA y ARTISTAS. Se unirán las tres tablas para desplegar los nombres de CD y los nombres de artista. El procedimiento no deberá incluir parámetros ni variables. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE PROCEDURE OBTENER_CD_ARTISTAS ( )
  SELECT cd.TITULO_CD, a.NOMBRE_ARTISTA
  FROM DISCOS_COMPACTOS cd, CDS_ARTISTA ac, ARTISTAS a
  WHERE cd.ID_DISCO_COMPACTO = ac.ID_DISCO_COMPACTO
  AND ac.ID_ARTISTA = a.ID_ARTISTA;
```

Deberá recibir un mensaje indicando que el procedimiento OBTENER_CD_ARTISTAS ha sido creado.

3. Después se convocará el procedimiento OBTENER_CD_ARTISTAS. Ingrese y ejecute la siguiente instrucción SQL:

```
CALL OBTENER_CD_ARTISTAS ( );
```

Cuando se invoque el procedimiento, se recibirán resultados de la consulta que incluyan una lista de todos los CD y sus artistas.

4. Ahora se eliminará el procedimiento de la base de datos. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP PROCEDURE OBTENER_CD_ARTISTAS CASCADE;
```

Deberá recibir un mensaje indicando que el procedimiento OBTENER_CD_ARTISTAS ha sido eliminado de la base de datos. Nótese que la palabra clave CASCADE puede no estar soportada por su implementación SQL.

5. El siguiente paso es crear un procedimiento similar al anterior, sólo que esta vez se definirá un parámetro que permita ingresar el nombre del CD. La instrucción SELECT incluirá un predicado que compare el valor TITULO_CD con el valor en el parámetro p_CD. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE PROCEDURE OBTENER_CD_ARTISTAS ( IN p_CD VARCHAR(60) )
  SELECT cd.TITULO_CD, a.NOMBRE_ARTISTA
  FROM DISCOS_COMPACTOS cd, CDS_ARTISTA ac, ARTISTAS a
  WHERE cd.ID_DISCO_COMPACTO = ac.ID_DISCO_COMPACTO
  AND ac.ID_ARTISTA = a.ID_ARTISTA
  AND cd.TITULO_CD = p_CD;
```

Deberá recibir un mensaje indicando que el procedimiento OBTENER_CD_ARTISTAS ha sido creado.

(continúa)

6. Ahora se convocará el procedimiento `OBTENER_CD_ARTISTAS`. La instrucción `CALL` incluirá el valor Fundamental para insertarlo en el parámetro. Ingrese y ejecute la siguiente instrucción SQL:

```
CALL OBTENER_CD_ARTISTAS('Fundamental');
```

Los resultados de la consulta deberán ahora incluir solamente la fila Fundamental.

7. El siguiente procedimiento que se creará es uno que utilice una variable que sostenga un número basado en el promedio de los valores `EN_EXISTENCIA`. La definición del procedimiento incluirá una instrucción compuesta que agrupe a las otras instrucciones en el cuerpo de la rutina. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE PROCEDURE OBTENER_CANTIDAD_CD ( )
BEGIN
    DECLARE v_En_existencia INT;
    SET v_En_Existencia = ( SELECT AVG(EN_EXISTENCIA)
                           FROM DISCOS_COMPACTOS );
    SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
    WHERE EN_EXISTENCIA < v_En_Existencia;
END;
```

Deberá recibir un mensaje indicando que el procedimiento ha sido creado.

8. Ahora se convocará el procedimiento. Ingrese y ejecute la siguiente instrucción SQL:

```
CALL OBTENER_CANTIDAD_CD( );
```

Los resultados de la consulta deberán incluir una lista de los CD que tienen un valor `EN_EXISTENCIA` menor al promedio para todos los valores `EN_EXISTENCIA`.

9. Cierre la aplicación cliente.

Resumen de Pruebe esto

En este ejercicio se crearon tres procedimientos. El primer procedimiento, `OBTENER_CD_ARTISTAS`, no incluía parámetros ni variables. Después se abandonó ese procedimiento y se modificó el procedimiento `OBTENER_CD_ARTISTAS` original para incluir un parámetro. Luego se creó un nuevo procedimiento (`OBTENER_CANTIDAD_CD`) que no incluía procedimientos, pero sí incluía una variable. La base de datos `INVENTARIO` deberá ahora contener estos dos procedimientos. Debido a que ambos procedimientos sólo recuperan datos SQL, pueden ser invocados en cualquier momento.

Agregar parámetros de salida a sus procedimientos

Hasta este punto hemos visto solamente procedimientos que toman valores de parámetro de entrada. Sin embargo, los procedimientos invocados por SQL también soportan parámetros de salida.

Los parámetros de salida proporcionan una forma de crear un procedimiento que arroja un valor (o múltiples valores).

El proceso para definir un parámetro salida es similar a aquél para definir un parámetro de entrada, solamente que se utiliza la palabra clave OUT en lugar de IN. Sin embargo, aún se debe proporcionar un nombre de parámetro y asignar un tipo de datos. Además, se debe asignar un valor a ese parámetro antes de que el procedimiento termine utilizando una instrucción SET, a pesar de que muchas implementaciones arrojan automáticamente valores nulos para los parámetros de salida a los que no se les asignó un valor.

Una definición de procedimiento puede incluir tanto parámetros de entrada como de salida (y parámetros de entrada/salida si la implementación los soporta). También es posible incluir variables o cualquier otro elemento que se ha visto hasta ahora en este capítulo.

Ahora demos un vistazo a un ejemplo de un parámetro de salida. La siguiente instrucción CREATE PROCEDURE crea un procedimiento que incluye un parámetro de salida (pero ningún parámetro de entrada o variable):

```
CREATE PROCEDURE NEW_AGE_TOTAL ( OUT p_Total INT )
BEGIN
    SET p_Total = ( SELECT SUM(EXISTENCIA_CD)
                    FROM INVENTARIO_CD i, TIPOS_CD t
                    WHERE i.ID_TIPO_CD = t.ID_TIPO_CD
                    AND NOMBRE_TIPO_CD = 'New Age' );
END;
```

Al parámetro de salida (p_Total) se le asigna el tipo de datos INT. La instrucción SET define un valor para el parámetro. En este caso, el valor es igual al número total de CD de New Age. Éste es el valor que es arrojado por el procedimiento cuando se invoca.

El proceso para invocar este procedimiento es diferente de lo que hemos visto hasta ahora. Cuando se invoca un procedimiento con un parámetro de salida, primero se debe declarar una variable que luego se utilice en la instrucción CALL, como se muestra en el siguiente ejemplo:

```
BEGIN
    DECLARE p_Total INT;
    CALL NEW_AGE_TOTAL( p_Total );
END;
```

En este caso se utilizó el mismo nombre para la variable que el nombre del parámetro que fue definido en la definición del procedimiento. Sin embargo, la variable y el parámetro no están obligados a tener el mismo nombre, aunque sí deben ser definidos con el mismo tipo de datos.

Crear funciones invocadas por SQL

Anteriormente en este capítulo, en la sección “Entender las rutinas invocadas por SQL”, se introdujeron los dos tipos de rutinas invocadas por SQL (procedimientos y funciones), y se describieron las diferencias y similitudes entre ellas. Las principales diferencias son que los procedimientos soportan la definición de parámetros de entrada y salida y son invocados utilizando la instrucción CALL. Las funciones, por otro lado, solamente soportan la definición de parámetros de entrada y son invocadas como un valor en una expresión. El resultado de una función es el valor arrojado por la ejecución de la función, y no a través de la definición explícita de un parámetro de salida.

Para crear una función se debe utilizar una instrucción `CREATE FUNCTION`. La instrucción es similar a la instrucción `CREATE PROCEDURE`, excepto por algunas diferencias críticas:

- Las definiciones de parámetros de entrada no pueden incluir la palabra clave `IN`.
- Una cláusula `RETURNS` debe seguir a las definiciones de los parámetros. La cláusula asigna un tipo de datos al valor arrojado por la función.
- El cuerpo de la rutina debe incluir una instrucción `RETURN` que defina el valor arrojado por el parámetro.

NOTA

SQL Server también utiliza una cláusula `RETURNS` para asignar un tipo de datos al valor arrojado, mientras que Oracle utiliza una cláusula `RETURN` para el mismo propósito. En ambos casos esta cláusula es seguida por la palabra clave `AS`. Tanto SQL Server como Oracle utilizan una instrucción `RETURN` en el cuerpo de la rutina para definir el valor arrojado por el parámetro.

Una definición de función puede incluir muchos de los elementos que han sido descritos a lo largo de este capítulo. Por ejemplo, se pueden definir variables locales, crear instrucciones compuestas y utilizar instrucciones condicionales. Además, se pueden definir y utilizar parámetros de entrada de la misma forma que se definen y utilizan parámetros de entrada en los procedimientos (excepto que no se utiliza la palabra clave `IN`).

Ahora que se tiene una idea general de cómo crear una función, observemos un ejemplo, el cual está basado en las tablas `CDS_EN_EXISTENCIA` e `INTERPRETES`, mostradas en la figura 13-2.

La siguiente instrucción `CREATE FUNCTION` define una función que arroja el nombre del artista para un CD en específico, como aparece en la tabla `CDS_EN_EXISTENCIA`:

```
CREATE FUNCTION CD_ARTIST ( p_Title VARCHAR(60) )
  RETURNS VARCHAR(60)
  BEGIN
    RETURN
    ( SELECT NOMBRE_ARTISTA
      FROM IN_STOCK_CDS s, PERFORMERS p
      WHERE s.Title = p.Title
        AND s.Title = p.Title );
END;
```

En la primera línea de la instrucción, la función `CD_ARTISTA` y el parámetro `p_Title` han sido definidos. En la siguiente línea, la cláusula `RETURNS` asigna el tipo de datos `VARCHAR(60)` al valor arrojado por la función. En el cuerpo de rutina se puede ver que una instrucción `RETURN` ha sido definida. La instrucción incluye una subconsulta que utiliza el valor del parámetro de entrada para arrojar el nombre del artista.

Como se puede ver, definir una función no es muy diferente a definir un procedimiento; sin embargo, convocar la función es un asunto muy distinto. En lugar de utilizar la instrucción `CALL` para invocar la función, se utiliza la función como sería con cualquiera de las funciones predefini-

CDS_EN_EXISTENCIA

TITULO: VARCHAR(60)	EXISTENCIA: INT
Famous Blue Raincoat	13
Blue	42
Court and Spark	22
Past Light	17
Kojiki	6
That Christmas Feeling	8
Out of Africa	29
Blues on the Bayou	27
Orlando	5

INTERPRETES

TITULO: VARCHAR(60)	NOMBRE_ARTISTA: VARCHAR(60)
Famous Blue Raincoat	Jennifer Warnes
Blue	Joni Mitchell
Court and Spark	Joni Mitchell
Past Light	William Ackerman
Kojiki	Kitaro
That Christmas Feeling	Bing Crosby
Patsy Cline: 12 Greatest Hits	Patsy Cline
After the Rain: The Soft Sounds of Erik Satie	Pascal Roge
Out of Africa	John Barry
Leonard Cohen The Best of	Leonard Cohen
Fundamental	Bonnie Raitt
Blues on the Bayou	B.B. King
Orlando	David Motion

Figura 13-2 Uso de funciones para recuperar valores de las tablas CDS_EN_EXISTENCIA e INTERPRETES.

das SQL. (Se vieron algunas de estas funciones en el capítulo 10.) Por ejemplo, supongamos que se quiere encontrar el nombre de un artista basado en el nombre del CD y que se quiere conocer qué otros CD ha hecho ese artista. Se puede crear una instrucción SELECT similar a la que se muestra en el siguiente ejemplo para recuperar los datos:

```
SELECT TITULO, NOMBRE_ARTISTA
FROM INTERPRETES
WHERE NOMBRE_ARTISTA = ARTISTA_CD ('Blue');
```

La función ARTISTA_CD arroja el valor Joni Mitchell (el artista del CD Blue), que es luego comparado con los valores NOMBRE_ARTISTA. Como resultado, dos filas son arrojadas por la instrucción, como se muestra en los siguientes resultados de la consulta:

TITULO	NOMBRE_ARTISTA
-----	-----
Blue	Joni Mitchell
Court and Spark	Joni Mitchell

Como se puede ver, las funciones ayudan a simplificar las consultas almacenando parte del código como un objeto de esquema (en la forma de una rutina invocada por SQL) y luego invocando ese código según sea necesario al convocar la función como un valor en la instrucción SQL. Las funciones proporcionan un amplio rango de posibilidades para arrojar valores que hagan a las consultas menos complejas y más manejables.

Pruebe esto 13-2 Crear funciones invocadas por SQL

En este ejercicio se creará una función llamada DISQUERA_CD en la base de datos INVENTARIO. La función proporcionará el nombre de la compañía que publica un CD en específico. Una vez que se crea la función, se invocará utilizándola como un valor en una instrucción SELECT. Cuando se haya terminado, se quitará esa función de la base de datos. Puede descargar el archivo Try_This_13_2.txt, que contiene las instrucciones SQL utilizadas en este ejercicio (en inglés).

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. Se creará una función que arroje el nombre de la compañía que publica un CD en específico. La función incluirá un parámetro de entrada que permita pasar el nombre del CD a la función. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE FUNCTION DISQUERA_CD ( p_CD VARCHAR(60) )
  RETURNS VARCHAR(60)
  BEGIN
    RETURN ( SELECT NOMBRE_COMPAÑIA
              FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
              WHERE d.ID_DISQUERA = l.ID_DISQUERA
                    AND TITULO_CD   = p_CD );
  END;
```

Se deberá recibir un mensaje indicando que la función DISQUERA_CD ha sido creada.

3. Ahora que la función ha sido creada, se puede utilizar en las instrucciones SQL como un valor en una expresión. La siguiente instrucción que se creará es una instrucción SELECT que arroja el nombre del CD y la compañía que publica CD para aquellos CD publicados por la misma compañía que el CD especificado. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, NOMBRE_COMPAÑIA
  FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
 WHERE d.ID_DISQUERA   = l.ID_DISQUERA
        AND NOMBRE_COMPAÑIA = DISQUERA_CD ('Blues on the Bayou');
```

Los resultados de la consulta deberán incluir una lista de cuatro CD, todos ellos publicados por la compañía MCA Records, la compañía que publica el CD Blues on the Bayou.

4. Intente ejecutar la misma instrucción utilizando diferentes nombres de CD para ver cuáles son los resultados arrojados.

5. Ahora se puede quitar la función DISQUERA_CD de la base de datos. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP FUNCTION DISQUERA_CD CASCADE;
```

Deberá recibirse un mensaje indicando que la función DISQUERA_CD ha sido abandonada de la base de datos.

6. Cierre la aplicación cliente.

Resumen de Pruebe esto

El ejercicio nos ayudó a crear una función (DISQUERA_CD) que incluye un parámetro (p_CD). El parámetro pasa el valor del nombre de un CD a la instrucción SELECT definida en la instrucción RETURN del parámetro. La instrucción utiliza esta información para determinar el nombre de la compañía que publica el CD. Entonces se utiliza la función DISQUERA_CD en una instrucción SELECT para recuperar los nombres de todos los CD que son publicados por la misma compañía que publicó el CD especificado. Después de eso, se abandona la función de la base de datos. Ahora que ha completado este ejercicio, intente crear otras funciones en la base de datos, y luego utilice las funciones en las instrucciones SELECT para ver qué tipo de datos pueden arrojar.

Autoexamen Capítulo 13

1. ¿Cuáles son los tipos de rutinas invocadas por SQL soportados por el estándar SQL?
 - A CHECK Constraint
 - B Function
 - C Trigger
 - D Procedimiento invocado por SQL
2. ¿Cuáles tipos de parámetros pueden utilizarse en una función invocada por SQL?
 - A De entrada
 - B De salida
 - C De entrada/salida
 - D Variables
3. ¿Cuál instrucción se utiliza para invocar un procedimiento invocado por SQL?
 - A RETURN
 - B CALL

- C** SET
 - D** DECLARE
- 4.** Un(a) _____ es un valor pasado a una instrucción en un procedimiento cuando se invoca ese procedimiento.
- 5.** ¿Cuáles tipos de parámetros pueden utilizarse en una función invocada por SQL?
- A** De entrada
 - B** De salida
 - C** De entrada/salida
 - D** Variables
- 6.** ¿Cuál es otro nombre para un procedimiento invocado por SQL?
- 7.** ¿Cuáles son las dos diferencias principales entre procedimientos y funciones?
- 8.** ¿Qué información debe incluirse en una instrucción CALL cuando se invoca un procedimiento?
- 9.** ¿Qué tipo de instrucciones pueden incluirse en un procedimiento?
- A** SELECT
 - B** INSERT
 - C** UPDATE
 - D** DELETE
- 10.** ¿Cuál instrucción se utiliza para asignar un valor inicial a una variable?
- A** DECLARE
 - B** RETURN
 - C** SET
 - D** CALL
- 11.** Una instrucción _____ permite agrupar las instrucciones SQL en bloques.
- 12.** ¿Qué palabra clave se utiliza para comenzar una instrucción condicional?
- A** IF
 - B** BEGIN
 - C** THEN
 - D** ELSE
- 13.** ¿Qué palabra clave se utiliza en una instrucción LOOP para terminar la repetición?

- 14.** ¿Cuál es la diferencia entre una instrucción condicional y una instrucción compuesta?
- 15.** ¿Cuáles son los dos tipos de instrucciones de repetición?
 - A** BEGIN...END
 - B** IF...END IF
 - C** LOOP...END LOOP
 - D** WHILE...END WHILE
- 16.** ¿Qué tipo de parámetro puede arrojar un valor cuando se invoca un procedimiento?
- 17.** ¿Qué paso debe tomarse cuando se convoque un procedimiento que incluya un parámetro de salida?
- 18.** ¿Qué tanto difiere una instrucción CREATE FUNCTION de una instrucción CREATE PROCEDURE?
- 19.** Se está convocando un procedimiento llamado OBTENER_TOTALES. El procedimiento no incluye ningún parámetro, pero sí incluye una instrucción SELECT que consulta la tabla INVENTARIO_CD. ¿Cuál instrucción SQL deberá utilizarse para invocar este parámetro?
- 20.** Se crea un procedimiento llamado OBTENER_INFO_CD que selecciona datos acerca de un artista de la tabla INFO_CD. El procedimiento incluye un parámetro de entrada. Se quiere convocar ese procedimiento con el valor Bonnie Raitt. ¿Cuál instrucción SQL deberá utilizarse para invocar el procedimiento?
- 21.** ¿Cuáles son los dos tipos de objetos de esquema que pueden utilizarse para almacenar una instrucción SELECT?

Capítulo 14

Crear activadores SQL

Habilidades y conceptos clave

- Entender los activadores SQL
 - Crear activadores SQL
 - Crear activadores de inserción
 - Crear activadores de actualización
 - Crear activadores de eliminación
-

Hasta este punto del libro hemos aprendido a crear varios objetos de esquema que pueden accederse o invocarse utilizando instrucciones SQL. Por ejemplo, se aprendió a crear tablas, vistas y rutinas invocadas por SQL. En cada caso, una vez que se creaban estos objetos, era necesario tomar cierta acción para interactuar directamente con ellos, por ejemplo ejecutando una instrucción SELECT para recuperar datos de una tabla o utilizando una instrucción CALL para invocar un procedimiento. Sin embargo, SQL soporta objetos que realizan acciones automáticamente. Estos objetos de esquema, que son conocidos como *activadores*, responden a modificaciones hechas a los datos dentro de una tabla. Si se ha realizado una modificación específica, el activador es invocado automáticamente, o *disparado*, causando que ocurra una acción adicional. Como resultado, nunca se invoca directamente el disparador (tomar una acción definida en el disparador implícitamente causa la invocación). En este capítulo exploraremos los activadores y cómo son utilizados cuando los datos de la tabla son modificados. También veremos ejemplos de cómo crear los tres tipos básicos de disparadores (de inserción, de actualización y de eliminación) y cómo pueden ser definidos para extender la funcionalidad de la base de datos y ayudar a asegurar la integridad de los datos.

Entender los activadores SQL

Si usted ha trabajado anteriormente con productos SQL, sin duda ha visto a los activadores implementados en alguna de las bases de datos de su organización, o al menos ha escuchado el término en algún momento. La mayoría de los RDBMS implementaron activadores en sus productos desde hace mucho tiempo, a pesar de que no fue sino hasta SQL:1999 que los activadores fueron añadidos al estándar. El resultado de los productos anteriores al estándar es que las implementaciones de los activadores resultan muy propietarias entre los productos SQL, y por lo tanto soportan diferentes tipos de funcionalidad y son implementados de diferentes formas. Por ejemplo, MySQL 5.0 no soporta activadores, pero esa característica adicional está prevista para la versión 5.1. Por otro lado, SQL Server y Oracle soportan actualmente activadores, pero los activadores de SQL Server de alguna manera están limitados en alcance, comparados al estándar SQL, mientras que los activadores de Oracle son más robustos (aun con todo esto, ningún producto implementa los activadores de acuerdo con las especificaciones del estándar SQL). A pesar de estas diferencias, existe un número de similitudes entre los productos (por ejemplo, el uso de la instrucción CREATE TRIGGER para crear un activador), y las implementaciones de activadores en varios productos comparten algunas características básicas, particularmente aquellas que hacen posible dispararlo automáticamente para realizar una acción secundaria relacionada con la acción primaria que invocó el activador.

NOTA

La funcionalidad soportada por los activadores a veces es llamada base de datos activa. De hecho, este término se utiliza para describir uno de los paquetes opcionales que están incluidos en el estándar SQL. El paquete (PKG008) define cómo los disparadores son implementados en SQL. (Un paquete es un conjunto de características a las cuales un producto puede aseverar correspondencia además de Core SQL.) Para mayor información acerca de la correspondencia en SQL:2006, véase el capítulo 1.

Antes de sumergirnos en los aspectos específicos de cómo implementar activadores, demos un vistazo al activador en sí mismo, el cual, como se dijo, es un objeto de esquema (en el mismo sentido que una tabla, vista o rutina invocadas por SQL). Una definición del activador precisa las características del activador y qué acciones van a ser tomadas cuando el activador sea invocado. Estas acciones, que están especificadas en una o más instrucciones SQL (referidas como *instrucciones de activador SQL*), pueden incluir tales eventos como actualizar tablas, eliminar datos, invocar procedimientos o realizar la mayoría de las tareas que pueden efectuarse con las instrucciones SQL. Ninguna limitación colocada en esas instrucciones es igual a aquellas colocadas por la implementación SQL.

Los activadores son invocados cuando se insertan datos en la tabla, se actualizan estos datos o se eliminan. Al definir uno o más activadores en una tabla, se puede especificar cuáles acciones de modificación de datos causarán que el activador se dispare. El activador nunca es invocado a menos que la acción específica sea llevada a cabo. Como se puede concluir fácilmente hasta ahora, SQL soporta tres tipos de activadores: de inserción, de actualización y de eliminación. Cada tipo corresponde a la instrucción de modificación de datos aplicable. Por ejemplo, un activador de inserción se dispara cuando una instrucción INSERT se ejecuta hacia la tabla especificada.

A pesar de que un activador es un objeto de esquema, separado de los objetos de la tabla, puede ser asociado solamente con una tabla, la cual especifica cuándo se crea la definición del activador. Cuando la instrucción de modificación de datos aplicable se invoca dentro de la tabla, el activador se dispara; sin embargo, no se disparará si una instrucción similar es invocada dentro de una tabla diferente, o si una instrucción diferente al tipo especificado se invoca dentro de la misma tabla. En ese sentido, un activador puede ser visto como un objeto de tabla, a pesar del hecho de ser creado en un nivel de esquema.

Si un disparador falla, generando una condición de error, la instrucción SQL que causó que se disparara el activador también falla y se repliega. Ésta es la forma en que los activadores pueden ser utilizados para reforzar restricciones complejas (el activador se escribe para realizar cualquier número de pruebas necesarias para verificar que las condiciones de la restricción sean cumplidas, y si no, termina por generar una condición de error).

Contexto de ejecución del activador

Antes de empezar el análisis de cómo se crea un activador, sería mejor tocar el tema de cómo se ejecuta un activador, con respecto al *contexto de ejecución del activador*, un tipo de contexto de ejecución SQL. Se puede considerar al *contexto de ejecución* como un espacio creado en la memoria que alberga un proceso de instrucción durante la ejecución de esa instrucción. SQL soporta muchos tipos de contextos de ejecución, y uno de ellos está relacionado con los activadores.

Un contexto de ejecución del activador se crea cada vez que un activador es invocado. Si múltiples activadores son invocados, se crea un contexto de ejecución para cada uno. Sin embargo,

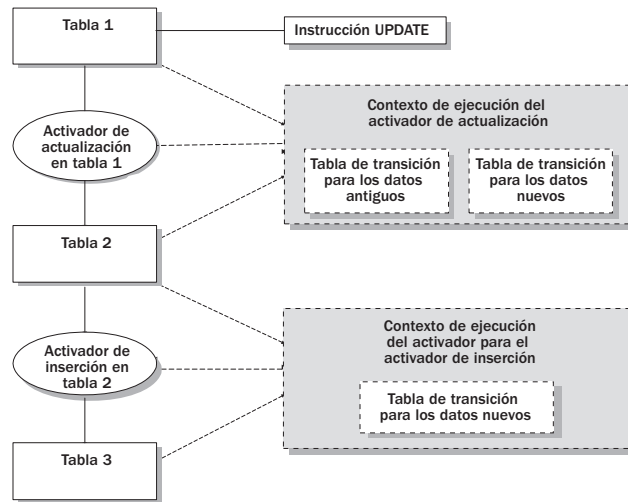


Figura 14-1 Contextos de ejecución del activador para dos activadores.

solamente un contexto de ejecución puede estar activo en una sesión en un momento dado. Esto resulta importante cuando un activador en una tabla causa que un activador en una segunda tabla sea disparado. Demos un vistazo a la figura 14-1 para ayudar a ilustrar este punto.

Observe que la figura contiene tres tablas. Un activador de actualización se define en la tabla 1, y un activador de inserción se define en la tabla 2. Cuando se ejecuta una instrucción UPDATE hacia la tabla 1, el activador de actualización se dispara, creando que un contexto de ejecución del activador se vuelva activo. Sin embargo, el activador de actualización, que está definido para insertar datos en la tabla 2, invoca al activador de inserción en la tabla 2 cuando el primer activador intenta insertar datos en esa tabla. Como resultado, un segundo contexto de ejecución es creado, el cual se convierte en el activo. Cuando la segunda ejecución del activador se ha completado, el segundo contexto de ejecución es destruido, y el primer contexto de ejecución se vuelve activo una vez más. Cuando la primera ejecución del activador se ha completado, el primer contexto de ejecución del activador es destruido.

Un contexto de ejecución del activador contiene la información necesaria para que el activador sea ejecutado correctamente. Esta información incluye detalles acerca del activador en sí y de la tabla en la cual el activador fue definido, a la que se denomina tabla en cuestión. Además, el contexto de ejecución incluye una o más tablas de transición, como se muestra en la figura 14-1. Las tablas de transición son tablas virtuales que albergan datos que son actualizados en, insertados hacia o eliminados de la tabla en cuestión. Si los datos son actualizados, entonces se crean dos tablas de transición, una para los datos antiguos y otra para los datos nuevos. Si los datos son insertados, se crea solamente una tabla de transición para los nuevos datos. Si los datos son eliminados se crea solamente una tabla de transición para los datos antiguos. Las tablas de transición y algunos otros aspectos de información en el contexto de ejecución del activador son utilizadas por las instrucciones SQL que realizan la acción del activador. Aprenderemos más acerca de cómo es utilizada esta información en la siguiente sección, cuando veamos la sintaxis CREATE TRIGGER.

Crear activadores SQL

Ahora que se tiene una idea general acerca de los activadores, demos un vistazo a la sintaxis que se utiliza para crearlos. La mayor parte de la sintaxis está destinada a definir las características del activador, por ejemplo el nombre y el tipo del activador. Es hasta el final de la instrucción donde se definen las instrucciones SQL activadas que especifican las acciones tomadas por el activador cuando sea invocado.

La sintaxis básica para crear una definición de activador es la siguiente:

```
CREATE TRIGGER <nombre del activador>
{ BEFORE | AFTER }
{ INSERT | DELETE | UPDATE [ OF <lista de la columna> ] }
ON <nombre de la tabla> [ REFERENCING <opciones para alias> ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( <condición de búsqueda> ) ]
<instrucciones SQL activadas>
```

Observemos cada línea de la sintaxis. La primera línea resulta muy sencilla. Simplemente se proporciona un nombre para el activador que sigue a las palabras clave CREATE TRIGGER. En la segunda línea se debe designar si el activador será invocado antes o después de que se aplique la instrucción de modificación de datos a la tabla en cuestión. Por ejemplo, si se está definiendo un activador de inserción, se puede especificar si las instrucciones SQL activadas se ejecutarán antes de que los datos sean insertados en la tabla en cuestión (utilizando la palabra clave BEFORE) o después de que los datos sean insertados en la tabla en cuestión (utilizando la palabra clave AFTER). Esta característica resulta particularmente útil cuando una de las tablas está configurada con una restricción de integridad referencial y no puede contener datos antes de que esos datos existan en la otra tabla. (Para mayor información acerca de la integridad referencial, véase el capítulo 4.) Dependiendo de la naturaleza de la acción activada que sea definida, puede no ser relevante designar BEFORE o AFTER debido que la acción activada puede no tener relación directa con los datos modificados en la tabla en cuestión.

En la tercera línea de la sintaxis se especifica si el activador es de inserción, de eliminación o de actualización. Si es un activador de actualización, se tiene la opción de aplicar el activador a una o más columnas específicas. Si se especifica más de una columna, los nombres de las columnas deben estar separados por comas. En la siguiente línea de la sintaxis se debe especificar una cláusula ON que incluya el nombre de la tabla en cuestión. Ésta es la tabla en la cual es aplicado el activador. El activador sólo puede ser aplicado a una tabla.

Hasta este punto, toda la sintaxis que se ha visto es obligatoria, excepto para especificar los nombres de columna en las definiciones del activador de actualización, lo cual sí es opcional. Sin embargo, las varias cláusulas siguientes no son obligatorias, pero agregan importantes capacidades al activador. La primera de estas cláusulas es la cláusula REFERENCING. Esta cláusula permite especificar cómo se referencian los datos que son albergados en el contexto de ejecución del activador dentro de la cláusula WHEN o las instrucciones SQL activadas. Se verá con más detalle la cláusula REFERENCING en la siguiente sección, “Referenciar valores antiguos y nuevos”.

La siguiente línea de la sintaxis contiene la cláusula FOR EACH, que incluye dos opciones: ROW o STATEMENT. Si se especifica ROW, el activador es invocado cada vez que una fila es insertada, actualizada o eliminada. Si se especifica STATEMENT, el activador se invoca solamente una

vez por cada instrucción de modificación de datos aplicable que sea ejecutada, sin importar cuántas filas serán afectadas. Si no se incluye esta cláusula en la definición del activador, la opción STATEMENT se asume automáticamente, y el activador se dispara sólo una vez por cada instrucción.

Posteriormente en la sintaxis está la cláusula opcional WHEN. Esta cláusula permite definir una condición de búsqueda que limita el alcance de cuándo es invocado el activador. La cláusula WHEN es similar a la cláusula WHERE de una instrucción SELECT. Se especifican uno o más predicados que definan una condición de búsqueda. Si la cláusula WHEN se evalúa como verdadera, el activador se dispara; de otra manera, ninguna acción del activador es llevada a cabo. Sin embargo, esto no afecta la instrucción de modificación de datos inicial que fue ejecutada hacia la tabla en cuestión; solamente las instrucciones SQL activadas precisadas en la definición del activador son afectadas.

Finalmente, el último componente que debe incluir la instrucción CREATE TRIGGER es una o más instrucciones SQL (a menudo llamadas el cuerpo del activador) que sean ejecutadas cuando el activador sea invocado y, si se incluye una cláusula WHEN, que esa cláusula se evalúe como verdadera. Si la definición del activador incluye más de una instrucción SQL activada, o si se está utilizando Oracle, esas instrucciones deberán estar encerradas en un bloque BEGIN...END, como aquellas que se vieron en el capítulo 13. Sin embargo, existe una diferencia de lo que se vio anteriormente. Cuando se utiliza en una definición de activador, la palabra clave BEGIN debe estar seguida por la palabra clave ATOMIC para notificar a la implementación SQL que las instrucciones dentro del bloque deben ser manejadas como una unidad. En otras palabras, todas las instrucciones deben ser ejecutadas exitosamente, o ninguno de los resultados de ninguna de las instrucciones podrá persistir. Sin la palabra clave ATOMIC, probablemente algunas instrucciones podrían ser ejecutadas mientras que otras podrían fallar.

NOTA

Muchas implementaciones no soportan el uso de la palabra clave ATOMIC en el bloque BEGIN...END de las instrucciones SQL activadas. Esto incluye tanto a SQL Server como a Oracle. Adicionalmente, con Oracle, todos los activadores y cuerpos de procedimiento deben estar encerrados en los bloques BEGIN...END.

Aparte del hecho de la palabra clave ATOMIC, las instrucciones SQL activadas, incluyendo el bloque BEGIN...END, puede consistir de casi cualquier instrucción SQL, dependiendo de las limitaciones de la implementación SQL. Asegúrese de revisar la documentación del producto para determinar cuáles limitaciones pueden ser colocadas en las instrucciones SQL activadas y cómo los activadores son generalmente creados e implementados.

Referenciar valores antiguos y nuevos

Ahora regresemos a la cláusula REFERENCING de la instrucción CREATE TRIGGER. El propósito de esta cláusula es permitir la definición de nombres de correlación para las filas almacenadas en las tablas de transición o para las tablas de transición como un todo. Como se recordará de la sección “Entender los activadores SQL” anterior en este capítulo, las tablas de transición albergan los datos que han sido actualizados, insertados o eliminados de la tabla en cuestión. Los nombres de correlación, o alias, pueden por lo tanto ser utilizados en las instrucciones SQL activadas para hacer referencia a los datos que están siendo albergados en las tablas de transición. Esto puede ser particularmente útil cuando se intenta modificar los datos en una segunda tabla basados en los

datos modificados en la tabla en cuestión. (Esto se volverá mucho más claro cuando veamos ejemplos posteriormente en este capítulo.)

Si se recuerda la sintaxis de la sección anterior, notaremos que la cláusula opcional REFERENCING incluye el marcador de posición <opciones para alias>. SQL soporta cuatro opciones para esta cláusula:

- REFERENCING OLD [ROW] [AS] <alias>
- REFERENCING NEW [ROW] [AS] <alias>
- REFERENCING OLD TABLE [AS] <alias>
- REFERENCING NEW TABLE [AS] <alias>

Observe que, en las primeras dos opciones, la palabra clave ROW no resulta obligatoria. Si no se especifica la palabra clave ROW, se asume automáticamente. Observe también que la palabra clave AS es opcional en todos los casos. Sin embargo, para los propósitos de mantener un código claro y de autorreferencia, se recomienda utilizar la opción completa cada que se incluya en una definición de activador.

Dependiendo del tipo de activador (de actualización, de inserción o de eliminación) y de la opción FOR EACH (ROW o STATEMENT), se pueden incluir hasta cuatro opciones REFERENCING en la definición del activador, una para cada tipo. Sin embargo, no se puede incluir más de una de un solo tipo. Por ejemplo, no se pueden incluir dos opciones OLD ROW en la definición del activador. Al agregar opciones REFERENCING a la definición del activador, se deben seguir estos lineamientos:

- No se pueden utilizar las opciones NEW ROW y NEW TABLE para eliminar activadores debido a que ningún dato nuevo se está creando.
- No se pueden utilizar las opciones OLD ROW y OLD TABLE para insertar activadores debido a que no existen datos antiguos.
- Se pueden utilizar todas las cuatro opciones en un activador de actualización debido a que existen datos antiguos y datos nuevos cuando se actualiza una tabla.
- Se pueden utilizar las opciones OLD ROW y NEW ROW solamente cuando se especifica la cláusula FOR EACH ROW en la definición del activador.

Una vez que se definen las cláusulas REFERENCING y se asignan los alias apropiados, todo está listo para utilizar esos alias en las instrucciones SQL activadas, de la misma manera que se utilizaban los nombres de correlación en las instrucciones SELECT.

Quitar activadores SQL

A pesar de que el estándar SQL no soporta ninguna clase de instrucción que permite alterar un activador, sí soporta una forma de eliminar un activador, que se logra utilizando la instrucción DROP TRIGGER. Como se puede ver en la siguiente sintaxis, esta instrucción es muy básica:

DROP TRIGGER <nombre>

Todo lo que se necesita hacer es proporcionar el nombre del activador, junto con las palabras clave DROP TRIGGER. Debido a que ningún otro objeto es dependiente del activador, no es

necesario especificar ninguna palabra clave adicional, por ejemplo CASCADE o RESTRICT. Cuando se ejecuta la instrucción DROP TRIGGER, la definición del activador es eliminada del esquema.

Crear activadores de inserción

Hasta ahora en este capítulo se ha proporcionado información importante acerca de los activadores y la sintaxis utilizada para crearlos. Ahora veremos algunos ejemplos de cómo son creados los activadores y qué sucede cuando son invocados. Comenzaremos con el activador de inserción, el cual, como ya se sabe, es invocado cuando una instrucción INSERT se ejecuta hacia la tabla en cuestión (la tabla en la cual ha sido definido el activador). En el primer ejemplo crearemos un activador en la tabla INVENTARIO_MENUDEO (la tabla en cuestión), mostrada en la figura 14-2. El activador, cuando se invoca, insertará los datos a la tabla REGISTRO_INVENTARIO.

La siguiente instrucción CREATE TRIGGER define un activador INSERT que se dispara después de que los datos son insertados en la tabla en cuestión:

```
CREATE TRIGGER INSERTAR_REGISTRO
AFTER INSERT ON INVENTARIO_MENUDEO
FOR EACH ROW
BEGIN ATOMIC
    INSERT INTO REGISTRO_INVENTARIO (TIPO_ACCION)
        VALUES ('INSERT');
END;
```

INVENTARIO_MENUDEO			REGISTRO_INVENTARIO	
NOMBRE_CD: VARCHAR (60)	P_MENUDEO: NUMERIC (5,2)	CANTIDAD: INT	TIPO_ACCION: CHAR (6)	FECHA_MODIFICACION: TIMESTAMP
Famous Blue Raincoat	16.99	5	INSERT	2002-12-22 10:58:05.120
Blue	14.99	10	UPDATE	2002-12-22 12:02:05.033
Court and Spark	14.99	12	UPDATE	2002-12-22 16:15:22.930
Past Light	15.99	11	DELETE	2002-12-23 11:29:14.223
Kojiki	15.99	4	INSERT	2002-12-23 13:32:45.547
That Christmas Feeling	10.99	8	INSERT	2002-12-23 15:51:15.730
Patsy Cline: 12 Greatest Hits	16.99	14	UPDATE	2002-12-23 17:01:32.270
			UPDATE	2002-12-24 10:46:35.123
			DELETE	2002-12-24 12:19:13.843
			UPDATE	2002-12-24 14:15:09.673

Figura 14-2 Crear un activador de inserción en la tabla INVENTARIO_MENUDEO.

NOTA

Como se mencionó al inicio del capítulo, las implementaciones SQL pueden variar ampliamente con respecto a la semántica de la instrucción CREATE TRIGGER. Por ejemplo, SQL Server no permite especificar una cláusula FOR EACH, ni soporta el uso de la palabra clave ATOMIC en la instrucción BEGIN...END. Por otro lado, la definición básica del activador en Oracle es mucho más cercana al estándar SQL, a pesar de que Oracle tampoco soporta el uso de la palabra clave ATOMIC en una definición de activador.

Veamos esta instrucción un elemento a la vez. En la primera línea, la cláusula CREATE TRIGGER define un activador llamado INSERTAR_REGISTRO. En la siguiente, la palabra clave AFTER se utiliza para especificar que las instrucciones SQL activadas serán ejecutadas *después* de que los datos hayan sido insertados en la tabla en cuestión. La palabra clave AFTER es seguida de la palabra clave INSERT, que define al activador como un activador de inserción. Después está la cláusula ON, que especifica el nombre de la tabla en cuestión. En este caso, la tabla en cuestión es INVENTARIO_MENUDEO.

Al movernos a través de la instrucción, llegamos a la cláusula FOR EACH, que especifica la palabra clave ROW. Esta cláusula, cuando se utiliza con ROW, indica que el activador será invocado para cada fila que sea insertada en la tabla, en lugar de para cada instrucción INSERT que sea ejecutada hacia la tabla. Siguiendo a la instrucción FOR EACH, se encuentran las instrucciones SQL activadas.

Las instrucciones SQL activadas incluyen una instrucción BEGIN...END y una instrucción INSERT. No es necesario incluir la instrucción BEGIN...END en la definición del activador debido a que, sin ella, solamente existe una instrucción SQL activada. Sin embargo, por ahora es preferible demostrar cómo sería utilizado el bloque si hubiera más de una instrucción. Observe que el bloque incluye una palabra clave ATOMIC que sigue a la palabra clave BEGIN. De acuerdo con el estándar SQL, ATOMIC es obligatoria, a pesar de que dependerá de la instrucción SQL si es que esta palabra clave es soportada.

El bloque BEGIN...END encierra una instrucción INSERT que agrega datos a la tabla REGISTRO_INVENTARIO cuando el activador es invocado. Cada vez que se inserta una fila en la tabla INVENTARIO_MENUDEO, una fila se inserta en la tabla REGISTRO_INVENTARIO. La fila REGISTRO_INVENTARIO contendrá el valor INSERT para la columna TIPO_ACCION. Un valor de marca de tiempo es agregado automáticamente a la columna FECHA_MODIFICACION, que está definida de manera predeterminada con CURRENT_TIMESTAMP.

Es posible, si se desea, crear otros activadores en la tabla INVENTARIO_MENUDEO. Por ejemplo, se podría requerir la creación de activadores de actualización y de eliminación que inserten filas en la tabla REGISTRO_INVENTARIO cuando sean hechas las modificaciones de datos aplicables. En ese caso, simplemente se creará una definición de activador para cada activador adicional que se necesite.

NOTA

El estándar SQL no coloca un límite en el número de activadores que pueden ser definidos en cualquier tabla; sin embargo, las implementaciones SQL pueden tener muchas restricciones, por lo que es mejor revisar la documentación del producto. Además de esas limitaciones, varias implementaciones pueden soportar diferentes formas en la cual pueden ser implementados múltiples activadores. Por ejemplo, SQL Server permite definir un activador de inserción, uno de actualización y uno de eliminación en una sola instrucción.

Ahora demos un vistazo a lo que sucede cuando se inserta una fila en la tabla INVENTARIO_MENUDEO. Supongamos que se quiere insertar información acerca del CD Fundamental. Se deberá crear una instrucción INSERT de la forma que se haría normalmente, como se muestra en el siguiente ejemplo:

```
INSERT INTO INVENTARIO_MENUDEO
VALUES ( 'Fundamental', 15.99, 18 );
```

Si se ejecutara la instrucción, la fila sería insertada en la tabla INVENTARIO_MENUDEO. Para verificar esto, se puede ejecutar la siguiente instrucción SELECT:

```
SELECT * FROM INVENTARIO_MENUDEO;
```

La instrucción SELECT arrojará los mismos resultados mostrados en la tabla INVENTARIO_MENUDEO de la figura 14-2, además de una fila adicional para el CD Fundamental, exactamente de la forma en que se esperaría. El activador no tiene efecto sobre las modificaciones de datos que se hagan a la tabla INVENTARIO_MENUDEO. Sin embargo, como se recordará de la definición de activador que fue definida en la tabla INVENTARIO_MENUDEO, las instrucciones SQL activadas deberán insertar datos en la tabla REGISTRO_INVENTARIO cuando el activador sea invocado, lo cual deberá haber ocurrido cuando se inserte una fila en la tabla INVENTARIO_MENUDEO. Para verificar esto, se puede ejecutar la siguiente instrucción SELECT:

```
SELECT * FROM REGISTRO_INVENTARIO;
```

Los resultados de la consulta deberán incluir no solamente las filas mostradas en la tabla REGISTRO_INVENTARIO en la figura 14-2, sino también una fila adicional que incluya el valor TIPO_ACCION de INSERT y un valor FECHA_MODIFICACION para la fecha y hora actuales. Cada vez que se inserte una fila en la tabla INVENTARIO_MENUDEO, una fila será insertada en la tabla REGISTRO_INVENTARIO. Se pudieron haber definido las instrucciones SQL activadas para tomar cualquier tipo de acción, no solamente eventos de registro en una tabla de registro. Dependiendo de las necesidades y de la base de datos en la cual se trabaje, se tiene un sinnúmero de posibilidades para el tipo de acciones que los activadores pueden soportar.

Crear activadores de actualización

Ahora que hemos visto un ejemplo de un activador de inserción, demos un vistazo a un par de activadores de actualización. El activador de actualización es invocado cuando una instrucción UPDATE se ejecuta hacia la tabla en cuestión. Al igual que con cualquier otro tipo de activador, cuando el activador es invocado, las instrucciones SQL activadas se ejecutan y una acción se lleva a cabo. Para ilustrar cómo funciona el activador de actualización, utilizaremos las tablas TITULOS_EN_EXISTENCIA y COSTOS_TITULO mostradas en la figura 14-3.

El primer ejemplo que veremos está creado sobre la tabla TITULOS_EN_EXISTENCIA e incluye instrucciones SQL activadas que actualizan la tabla COSTOS_TITULO, como se muestra en la siguiente instrucción CREATE TRIGGER:

```
CREATE TRIGGER ACTUALIZAR_COSTOS_TITULO
AFTER UPDATE ON TITULOS_EN_EXISTENCIA
REFERENCING NEW ROW AS Nueva
FOR EACH ROW
```

TITULOS EN EXISTENCIA			COSTOS_TITULO		
TITULO_CD: VARCHAR (60)	TIPO_CD: CHAR (20)	INVENTARIO: INT	TITULO_CD: VARCHAR (60)	MAYOREO: NUMERIC (5,2)	MENUDEO: NUMERIC (5,2)
Famous Blue Raincoat	Folk	12	Famous Blue Raincoat	8.00	16.99
Blue	Popular	24	Blue	7.50	15.99
Past Light	New Age	9	Past Light	6.00	14.99
Blues on the Bayou	Blues	19	Blues on the Bayou	7.25	15.99
Luck of the Draw	Popular	25	Luck of the Drive	7.50	15.99
Deuces Wild	Blues	17	Deuces Wild	7.45	14.99
Nick of Time	Popular	11	Nick of Time	6.95	14.99

Figura 14-3 Crear un activador de actualización en la tabla TITULOS_EN_EXISTENCIA.

```
BEGIN ATOMIC
  UPDATE COSTOS_TITULO c
    SET MENUDEO = MENUDEO * 0.9
  WHERE c.TITULO_CD = Nuevo.TITULO_CD;
END;
```

Como se puede ver, la definición de este activador es similar en muchas formas al activador de inserción que vimos en el ejemplo anterior. La definición del activador de actualización incluye el nombre del activador (UPDATE_COSTOS_TITULO) y especifica las condiciones AFTER y UPDATE. Luego la cláusula ON sigue a la palabra clave UPDATE y proporciona el nombre de la tabla destino. Siguiendo a todo esto está una línea de código que no se vio en el ejemplo anterior (una cláusula REFERENCING).

La cláusula REFERENCING utiliza la opción NEW ROW para definir un nombre de correlación para la fila que ha sido actualizada en la tabla TITULOS_EN_EXISTENCIA. Sin embargo, la cláusula REFERENCING, y subsecuentemente la condición de búsqueda o las instrucciones SQL activadas que puedan referirse al alias definido en esta cláusula, no hacen referencia directa a la tabla TITULOS_EN_EXISTENCIA. En su lugar, hacen referencia a la tabla de transición para los nuevos datos en el contexto de ejecución del activador. En otras palabras, el nombre de correlación definido en la cláusula REFERENCING hace referencia a la fila actualizada que es copiada a la tabla de transición. En este caso, el nombre de correlación es Nuevo. Como resultado, el nombre de correlación Nuevo puede ser utilizado en la condición de búsqueda en la cláusula WHEN o en las instrucciones SQL activadas para referirse a los datos en la tabla de transición.

Una vez que se ha definido el nombre de correlación en la cláusula REFERENCING, debe utilizársele para cualificar los nombres de columna de la fila modificada cuando se haga referencia a ella en la cláusula WHEN o en las instrucciones SQL activadas. En la instrucción CREATE TRIGGER del ejemplo anterior se puede ver que el alias es utilizado en la cláusula WHERE de la instrucción UPDATE. Observe que la palabra Nuevo precede al nombre de columna y que los dos están separados por un punto.

Esto resulta característico acerca de cómo se cualificaría un nombre. Es similar a la forma en la cual se utiliza el nombre cualificado `c.TITULO_CD` para la columna `TITULO_CD` en la tabla `COSTOS_TITULO`. Si se hubiera especificado un nombre de correlación `NEW ROW` diferente o utilizado el nombre en la cláusula `WHEN` o en alguna otra parte de la instrucción SQL activada, de todas maneras se habría cualificado el nombre de la columna con el alias que hace referencia a las filas en la tabla de transición o la tabla en sí.

NOTA

SQL Server no soporta la cláusula `REFERENCING`. Sin embargo, soporta una funcionalidad similar al asignar automáticamente los nombres `Inserted` y `Deleted` a las tablas de transición (`Inserted` para los nuevos datos y `Deleted` para los datos antiguos). Además, existen algunos casos en los cuales se debe declarar una variable que utilice valores de las tablas `Inserted` y `Deleted`, en lugar de cualificar los nombres de columna, como se hace en el estándar SQL. Oracle, por otro lado, sí soporta la cláusula `REFERENCING`, pero también asigna automáticamente los nombres `New` y `Old` a las tablas de transición, las cuales pueden utilizarse en la cláusula `WHEN` y en las instrucciones SQL activadas sin especificar una cláusula `REFERENCING`. Cuando se utilizan los alias en las instrucciones SQL activadas de una definición de activador de Oracle, se debe preceder el nombre alias con dos puntos, como en `:Nuevo`. Éste no es el caso para la cláusula `WHEN`, en la cual el nombre alias se utiliza sin los dos puntos. Asimismo, no se puede utilizar la palabra clave `ROW` en la cláusula `REFERENCING` de una definición de activador de Oracle.

Además de la cláusula `REFERENCING`, la instrucción `CREATE TRIGGER` también incluye una cláusula `FOR EACH`, que especifica la opción `ROW`. Observe también que las instrucciones SQL activadas incluyen una instrucción `BEGIN...END`, que encierra una instrucción `UPDATE`. Como se puede ver, la instrucción `UPDATE` modifica el valor `MENUDEO` en la tabla `COSTOS_TITULO` para el `CD` que fue actualizado en la tabla `TITULOS_EN_EXISTENCIA`.

Ahora demos un vistazo a lo que sucede cuando se actualiza la columna `TITULOS_EN_EXISTENCIA`. La siguiente instrucción `UPDATE` cambia el valor `INVENTARIO` para la fila `Famous Blue Raincoat`:

```
UPDATE TITULOS_EN_EXISTENCIA
SET INVENTARIO = 30
WHERE TITULO_CD = 'Famous Blue Raincoat';
```

Cuando se ejecuta la instrucción `UPDATE`, el activador `UPDATE_COSTOS_TITULO` es invocado, causando que la tabla `COSTOS_TITULO` sea actualizada. Como resultado, no solamente el valor `INVENTARIO` en la tabla `TITULOS_EN_EXISTENCIA` es cambiado a 30, sino que el valor `MENUDEO` en la tabla `COSTOS_TITULO` se reduce a 15.29 (`RETAIL * 0.9`). Cada vez que se actualice la tabla `TITULOS_EN_EXISTENCIA`, la fila o filas correspondientes en la tabla `COSTOS_TITULO` serán reducidas en 10 por ciento.

A veces se puede necesitar limitar cuándo las instrucciones SQL activadas serán ejecutadas. Por ejemplo, se puede requerir reducir el precio de los `CD` solamente cuando el inventario exceda una cierta cantidad. Como resultado, puede decidirse cambiar la definición del activador para incluir una cláusula `WHEN` que defina la condición de búsqueda necesaria. Sin embargo, como se dijo anteriormente, SQL no soporta una instrucción `ALTER TRIGGER` (a pesar de que Oracle soporta la sintaxis `CREATE OR REPLACE TRIGGER` que puede ser utilizada para reemplazar com-

pletamente un activador existente), por lo que podría ser necesario eliminar primero el activador de la base de datos. La forma de hacer eso es utilizar la siguiente instrucción DROP TRIGGER:

```
DROP TRIGGER ACTUALIZAR_COSTOS_TITULO;
```

Cuando se ejecuta esta instrucción, la definición del activador es eliminada del esquema y ahora es posible volver a crear el activador con las modificaciones necesarias. El siguiente ejemplo creará una vez más el activador ACTUALIZAR_COSTOS_TITULO, pero esta vez se ha añadido una cláusula WHEN a la instrucción:

```
CREATE TRIGGER ACTUALIZAR_COSTOS_TITULO
  AFTER UPDATE ON TITULOS_EN_EXISTENCIA
  REFERENCING NEW ROW AS Nuevo
  FOR EACH ROW
  WHEN ( Nuevo.INVENTARIO > 20 )
  BEGIN ATOMIC
    UPDATE COSTOS_TITULO c
      SET MENUDEO = MENUDEO * 0.9
      WHERE c.TITULO_CD = Nuevo.TITULO_CD;
  END;
```

Como se puede ver, la cláusula WHEN especifica que el valor INVENTARIO debe ser mayor a 20; de otra manera, las instrucciones SQL activadas no serán invocadas. Observe que el nombre de columna CD_TITLE es cualificado en la cláusula WHERE de la instrucción UPDATE. Como resultado, la cláusula WHEN hará referencia a la tabla de transición para los nuevos datos en el contexto de ejecución del activador cuando se comparen los valores.

Ahora demos un vistazo a lo que sucede cuando se actualiza la tabla TITULOS_EN_EXISTENCIA. La siguiente instrucción UPDATE cambia el valor INVENTARIO para la fila Past Light:

```
UPDATE TITULOS_EN_EXISTENCIA
  SET INVENTARIO = 25
  WHERE TITULO_CD = 'Past Light';
```

Como se pudiera esperar, el valor INVENTARIO en la columna TITULOS_EN_EXISTENCIA es cambiado a 25. Además, debido a que se cumple la condición especificada en la cláusula WHEN (Nuevo.INVENTARIO > 20), las instrucciones SQL activadas son ejecutadas y la tabla COSTOS_TITULO es actualizada. Si se consultara la tabla COSTOS_TITULO, se vería que el valor MENUDEO para la fila Past Light ha sido cambiado a 13.49.

Ahora demos un vistazo a una instrucción UPDATE que establece el valor INVENTARIO a una cantidad menor a 20:

```
UPDATE TITULOS_EN_EXISTENCIA
  SET INVENTARIO = 10
  WHERE TITULO_CD = 'Past Light';
```

Esta instrucción actualizará el valor INVENTARIO en la tabla TITULOS_EN_EXISTENCIA, pero no provocará que se ejecuten las instrucciones SQL activadas debido que no se cumple la condición de búsqueda en la cláusula WHEN. Como resultado, ningún cambio se realiza en la tabla COSTOS_TITULO, a pesar de que la tabla TITULOS_EN_EXISTENCIA es de todas maneras actualizada.

Pregunta al experto

P: Cuando se describían los contextos de ejecución del activador, se analizó cómo un activador puede causar que otro activador sea invocado. ¿Existe un punto en el cual múltiples activadores puedan convertirse en un problema si demasiados de ellos son invocados?

R: Pueden surgir problemas cuando múltiples activadores son invocados y éstos causan un efecto de cascada de una tabla a la siguiente. Por ejemplo, un intento de actualizar una tabla puede invocar un activador que actualiza otra tabla. Esa actualización, en turno, podría invocar a otro activador que modifique datos en una tabla más. Este proceso puede continuar mientras un activador siga invocando a uno más, creando resultados indeseables y modificaciones de datos no planeadas. La condición puede todavía empeorar si se crea una repetición en la cual un activador cause la modificación de datos en una tabla para la cual se haya disparado otro activador. Por ejemplo, la modificación de datos en una tabla puede invocar un activador que cause una segunda modificación. Esa modificación pudiera invocar otro activador, que en su curso invoque a otro activador, y que éste invoque a uno más. El último activador pudiera modificar datos en la tabla original, causando que el primer activador se dispare otra vez, y así repetir sin fin todo el proceso desde el principio hasta que el sistema falle o un proceso específico de la implementación finalice la repetición. La mejor manera de prevenir modificaciones no deseadas o repeticiones de activadores es a través de la planeación cuidadosa en el diseño de la base de datos. Los activadores no deben ser implementados a menos que se esté seguro de su impacto. Además de la planeación cuidadosa, deberá revisarse la implementación SQL para determinar cuáles redes de seguridad puedan colocarse para prevenir la repetición de activadores o un indeseado efecto de cascada. Por ejemplo, algunas implementaciones permiten controlar si es que se aprobarán los activadores de cascada, y algunas limitan el número de activadores de cascada que se pueden disparar. Asegúrese de leer la documentación del producto antes de crear múltiples activadores en su base de datos.

P: Anteriormente, se mencionó que SQL permite definir múltiples activadores en una tabla. ¿Cómo se procesan los activadores si se invocan múltiples activadores?

R: En SQL, procesar múltiples activadores constituye una preocupación solamente si los activadores están definidos para dispararse al mismo tiempo (BEFORE o AFTER) y si son el mismo tipo de activador (INSERT, UPDATE o DELETE). Por ejemplo, un escenario de múltiples activadores existiría si dos o más activadores están definidos (en la misma tabla) con las palabras clave AFTER UPDATE. Si esta condición existe, entonces los activadores son invocados en el mismo orden en el cual fueron definidos. Veamos un ejemplo para mostrar lo que esto quiere decir. Si se crea Trigger1, después Trigger2 y luego Trigger3, Trigger1 se invoca primero, después Trigger2 y luego Trigger3. El problema con esto es que SQL no define ninguna forma en la cual se puede cambiar ese orden. Por ejemplo, si se decide que se desea invocar Trigger3 antes de Trigger1, la única opción (basándose en el estándar SQL) es eliminar Trigger1 y Trigger2 del esquema y luego volver a crear los activadores en el orden en que se quieren invocar. Debido a que no se eliminó Trigger3, éste se moverá al punto superior y será el primero en invocarse debido a que será visto como el primero en haber sido creado.

Crear activadores de eliminación

El último tipo de activador que veremos es el activador de eliminación. Como se puede esperar, el activador de eliminación se invoca cuando se ejecuta una instrucción DELETE hacia la tabla en cuestión, y al igual que con los otros activadores, las instrucciones SQL activadas se ejecutan y se lleva a cabo una acción. Ahora demos un vistazo a un ejemplo que utilice la tabla EXISTENCIA_CD y la tabla CD_AGOTADO, como se muestra en la figura 14-4.

Supongamos que se quiere crear un activador en la tabla EXISTENCIA_CD. Se requiere que el activador inserte los valores eliminados en la tabla CD_AGOTADO. La siguiente instrucción CREATE TRIGGER utiliza una cláusula REFERENCING para permitir a la instrucción SQL activada conocer cuáles datos serán insertados en la tabla CD_AGOTADO:

```
CREATE TRIGGER INSERTAR_CD_AGOTADO
AFTER DELETE ON EXISTENCIA_CD
REFERENCING OLD ROW AS Antigua
FOR EACH ROW
INSERT INTO CD_AGOTADO
VALUES ( Antigua.NOMBRE_CD, Antigua.TIPO_CD );
```

En esta instrucción se está creando un activador llamado INSERTAR_CD_LANZADO. La instrucción es definida con las palabras clave a AFTER DELETE, lo que significa que los valores antiguos serán insertados a la tabla CD_AGOTADO después que hayan sido eliminados de la tabla EXISTENCIA_CD. La cláusula ON identifica la tabla EXISTENCIA_CD, la tabla en cuestión.

Siguiendo a la cláusula ON se encuentra la cláusula REFERENCING. La cláusula REFERENCING utiliza la opción OLD ROW para asignar el nombre de correlación Antigua. Recuerde que puede utilizar solamente las opciones OLD ROW y OLD TABLE en la cláusula REFERENCING de una definición de activador de eliminación. Esto se debe a que no existen datos nuevos, solamente los datos antiguos que están siendo eliminados.

EXISTENCIA_CD			CD_AGOTADO	
NOMBRE_CD:	TIPO_CD:	EN_EXISTENCIA:	NOMBRE_CD:	TIPO_CD:
VARCHAR (60)	CHAR (4)	INT	VARCHAR (60)	CHAR (4)
Famous Blue Raincoat	FROK	19	Court and Spark	FROK
Blue	CPOP	28	Kojiki	NEWA
Past Light	NEWA	6	That Christmas Feeling	XMAS
Out of Africa	STRK	8	Patsy Cline: 12 Greatest Hits	CTRY
Fundamental	NPOP	10	Leonard Cohen The Best of	FROK
Blues on the Bayou	BLUS	11	Orlando	STRK

Figura 14-4 Crear un activador de eliminación en la tabla EXISTENCIA_CD.

La cláusula FOR EACH sigue a la cláusula REFERENCING. La cláusula FOR EACH utiliza la opción ROW. Como resultado, una fila es insertada en la tabla CD_AGOTADO por cada fila eliminada de la tabla EXISTENCIA_CD.

Después está la instrucción SQL activada. Observe que en este ejemplo no se utiliza una instrucción BEGIN...END. Debido a que sólo hay una instrucción activada, no se necesita utilizar el bloque BEGIN...END (excepto en Oracle, que siempre requiere de un bloque). La instrucción activada en este caso es una instrucción INSERT que especifica dos valores, cada uno de ellos basado en los valores eliminados de la tabla EXISTENCIA_CD. Se utiliza el alias Antigua para cualificar cada nombre de columna. Como resultado, los valores eliminados pueden ser insertados directamente en la tabla CD_AGOTADO.

Ahora demos un vistazo a un ejemplo de lo que sucede cuando se elimina una fila de la tabla EXISTENCIA_CD. La siguiente instrucción DELETE elimina la fila Past Light de la tabla:

```
DELETE EXISTENCIA_CD  
WHERE NOMBRE_CD = 'Past Light';
```

Una vez que se ejecuta esta instrucción, la fila es eliminada y el activador es invocado. La fila es entonces insertada en la tabla CD_AGOTADO. Se puede verificar la eliminación utilizando la siguiente instrucción SELECT para ver los contenidos de la tabla EXISTENCIA_CD:

```
SELECT * FROM EXISTENCIA_CD;
```

Los resultados de la consulta de esta instrucción ya no deberán incluir la fila Past Light. Sin embargo, si se ejecuta la siguiente instrucción SELECT, se verá que una fila ha sido añadida a la tabla CD_AGOTADO:

```
SELECT * FROM CD_AGOTADO;
```

Cada vez que una fila sea eliminada de la tabla EXISTENCIA_CD, dos valores de esa fila serán insertados a la tabla CD_AGOTADO. Al igual que con otras definiciones de activadores, se pudo haber incluido una cláusula WHEN en la instrucción CREATE TRIGGER para que las instrucciones SQL activadas sean ejecutadas solamente cuando la condición de búsqueda especificada en la cláusula WHEN se evalúe como verdadera. De otra manera, las instrucciones no serán ejecutadas. La fila aún será eliminada de la tabla EXISTENCIA_CD, pero nada será insertado a la tabla CD_AGOTADO.

Pruebe esto 14-1 Crear activadores SQL

A lo largo de este capítulo se ha visto cómo crear los tres tipos básicos de activadores (de inserción, de actualización y de eliminación). Ahora usted creará sus propios activadores (uno para cada uno de los tres tipos) en la base de datos INVENTARIO. Los activadores serán definidos para registrar la actividad de modificación de datos que ocurra en la tabla ARTISTAS. Cada vez que los datos en la tabla ARTISTAS sean modificados, una fila será insertada en una tabla de registro, la cual creará usted mismo. La tabla de registro guardará el tipo de acción tomada (inserción, actualización, eliminación), el valor ID_ARTISTA para la fila modificada, y una marca de tiempo de cuándo la fila fue insertada en la tabla. Como resultado, cada vez que se ejecute una instrucción

INSERT, UPDATE o DELETE hacia la tabla ARTISTS, una fila será insertada en la nueva tabla para cada fila que sea modificada. Al igual que con otros ejercicios en este libro (particularmente en el capítulo 13, cuando se crearon los procedimientos almacenados), deberá referirse a la documentación de su implementación SQL al crear activadores para asegurarse que está siguiendo los estándares del producto. Existen muchas variaciones entre las implementaciones SQL. Puede descargar el archivo Try_This_14.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para sus RDBM y conéctese con la base de datos INVENTARIO.
2. Antes de crear los activadores reales en la tabla ARTISTAS, deberá crear una tabla que registre las modificaciones de datos que se hagan en la tabla ARTISTAS. La tabla de registro, llamada REGISTRO_ARTISTA, incluirá tres columnas para registrar los eventos de modificación de datos. Una de las columnas será configurada con un valor por defecto que registre la fecha y hora actuales. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE TABLE REGISTRO_ARTISTA
( TIPO_ACCION CHAR(6),
  ID_ARTISTA INT,
  FECHA_MOD   TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
```

Deberá recibir un mensaje indicando que la tabla fue creada exitosamente.

3. Ahora se creará un activador de inserción en la tabla ARTISTAS. La definición del activador incluirá una cláusula REFERENCING que especifique un nombre de correlación (Nueva) para cada nueva fila que sea insertada en la tabla ARTISTAS. Ese nombre de correlación será luego utilizado en la instrucción SQL activada como un valor insertado en la tabla REGISTRO_ARTISTA. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE TRIGGER INSERTAR_REGISTRO
AFTER INSERT ON ARTISTAS
REFERENCING NEW ROW AS Nueva
FOR EACH ROW
BEGIN ATOMIC
  INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
    VALUES ( 'INSERT', Nueva.ID_ARTISTA );
END;
```

Deberá recibir un mensaje indicando que el activador fue creado exitosamente.

4. Ahora se creará un activador de actualización. Esta definición del activador es similar a la del paso 3, excepto que aquí se está especificando que es un activador de actualización. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE TRIGGER ACTUALIZAR_REGISTRO
AFTER UPDATE ON ARTISTAS
REFERENCING NEW ROW AS Nueva
FOR EACH ROW
BEGIN ATOMIC
```

(continúa)

```
INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
VALUES ( 'UPDATE', Nueva.ID_ARTISTA );
END;
```

Deberá recibir un mensaje indicando que el activador fue creado exitosamente.

5. Ahora se creará un activador de eliminación. Esta definición del activador es un poco diferente a las de los últimos activadores debido a que la cláusula REFERENCING especifica un nombre de correlación para los valores antiguos, en lugar de para los nuevos. Esto se debe a que los nuevos valores no son creados cuando se eliminan los datos de una tabla. El nombre de correlación (Antigua) es por lo tanto utilizado en la cláusula VALUES de la instrucción INSERT. Ingrese y ejecute la siguiente instrucción SQL:

```
CREATE TRIGGER ELIMINAR_REGISTRO
AFTER DELETE ON ARTISTAS
REFERENCING OLD ROW AS Antigua
FOR EACH ROW
BEGIN ATOMIC
INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
VALUES ( 'DELETE', Antigua.ID_ARTISTA );
END;
```

Deberá recibir un mensaje indicando que el activador fue creado exitosamente.

6. Ahora es posible comenzar a probar los activadores que se han creado. El primer paso es insertar datos en la tabla ARTISTAS. En esta instrucción, los valores son especificados por la columna ID_ARTISTA en la columna NOMBRE_ARTISTA, pero no por la columna LUGAR_DE_NACIMIENTO. Como resultado, el valor predeterminado de Desconocido será insertado en esa columna. Ingrese y ejecute la siguiente instrucción SQL:

```
INSERT INTO ARTISTAS ( ID_ARTISTA, NOMBRE_ARTISTA )
VALUES ( 2019, 'John Lee Hooker' );
```

Deberá recibir un mensaje indicando que la fila fue insertada exitosamente en la tabla ARTISTAS.

7. Ahora se actualizará la fila que acaba de insertarse proporcionando un valor para la columna LUGAR_DE_NACIMIENTO. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE ARTISTAS
SET LUGAR_DE_NACIMIENTO = 'Clarksdale, Mississippi, Estados Unidos'
WHERE ID_ARTISTA = 2019;
```

Deberá recibir un mensaje indicando que la fila fue actualizada exitosamente en la tabla ARTISTAS.

8. El siguiente paso es eliminar la fila que acaba de crearse. Ingrese y ejecute la siguiente instrucción SQL:

```
DELETE ARTISTAS
WHERE ID_ARTISTA = 2019;
```

Deberá recibir un mensaje indicando que la fila fue eliminada exitosamente de la tabla ARTISTAS.

- 9.** Ahora que se han modificado los datos en la tabla ARTISTAS, se revisará la tabla REGISTRO_ARTISTA para verificar que las filas hayan sido ingresadas en la tabla registrando las modificaciones de datos de la tabla ARTISTAS. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT * FROM REGISTRO_ARTISTA;
```

Los resultados de la consulta deberán incluir tres filas, una para cada tipo de acción (INSERT, UPDATE y DELETE). Todas las filas deberán tener el mismo valor ID_ARTISTA (2019) e incluir las fechas y horas actuales.

- 10.** El siguiente paso será quitar los activadores de la base de datos. El primer activador que debe quitarse es el activador de inserción. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP TRIGGER INSERTAR_REGISTRO;
```

Deberá recibir un mensaje indicando que el activador fue eliminado exitosamente de la base de datos.

- 11.** Después se eliminará el activador de actualización. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP TRIGGER ACTUALIZAR_REGISTRO;
```

Deberá recibir un mensaje indicando que el activador fue eliminado exitosamente de la base de datos.

- 12.** Ahora abandone el activador de eliminación. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP TRIGGER ELIMINAR_REGISTRO;
```

Deberá recibir un mensaje indicando que el activador fue eliminado exitosamente de la base de datos.

- 13.** Finalmente, se eliminará la tabla REGISTRO_ARTISTA que se creó en el paso 2. Ingrese y ejecute la siguiente instrucción SQL:

```
DROP TABLE REGISTRO_ARTISTA;
```

Deberá recibir un mensaje indicando que la tabla fue eliminada exitosamente de la base de datos.

- 14.** Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio se creó la tabla REGISTRO_ARTISTA, que fue determinada para almacenar información acerca de las modificaciones de datos a la tabla ARTISTAS. Después se crearon tres activadores sobre la tabla ARTISTAS (un activador de inserción, un activador de actualización y un activador de eliminación). Los tres activadores utilizaron cláusulas REFERENCING para permitir pasar el valor ID_ARTISTA de la fila modificada a la tabla REGISTRO_ARTISTA. Después de

(continúa)

haber creado los activadores, se insertaron, actualizaron y eliminaron datos en la tabla ARTISTAS para probar los activadores. Luego se revisó el contenido de la tabla REGISTRO_ARTISTA para verificar que todas las modificaciones de datos hubieran sido registradas adecuadamente. Después de eso, se abandonaron los tres activadores y la tabla REGISTRO_ARTISTA. Al momento de completar el ejercicio, la base de datos INVENTARIO deberá haber sido regresada a su estado original tal como estaba al principio.

Autoexamen Capítulo 14

1. ¿Qué es un activador?
2. ¿Cuáles son los tres tipos de activadores?
3. ¿Qué tipo de acciones pueden ser realizadas por las instrucciones SQL activadas?
4. ¿Cuáles acciones puede invocar un activador?
 - A Actualización de datos
 - B Consulta de datos
 - C Eliminación de datos
 - D Inserción de datos
5. ¿Cuándo es invocado un activador de inserción?
6. ¿En cuántas tablas puede ser definido un activador?
 - A Solamente una
 - B Una o más
 - C De una a tres
 - D Cualquier número de tablas
7. Un(a) _____ es un espacio creado en la memoria que alberga un proceso de activador durante la ejecución de ese activador.
8. Se insertan datos en la tabla 1, que invoca un activador de inserción definido en esa tabla. El activador actualiza la información en la tabla 2, que invoca un activador de actualización definido en esa tabla. El activador de actualización elimina información en la tabla 3, que invoca un activador de eliminación definido en esa tabla. ¿Cuál contexto de ejecución de activador está activo en este punto?
 - A El contexto de ejecución de activador para el activador de inserción
 - B El contexto de ejecución de activador para el activador de actualización
 - C El contexto de ejecución de activador para el activador de eliminación

9. Si tres activadores son invocados durante una sesión, ¿cuántos contextos de ejecución de activador son creados en esa sesión?
10. ¿Qué información es incluida en un contexto de ejecución de activador?
11. ¿En cuál cláusula de la instrucción CREATE TRIGGER se asignan nombres de correlación a los datos antiguos y nuevos?
 - A FOR EACH
 - B ON
 - C REFERENCING
 - D WHEN
12. ¿En cuál cláusula de la instrucción CREATE TRIGGER se especifica si las instrucciones SQL activadas serán ejecutadas una vez para cada fila o una vez para cada instrucción?
 - A FOR EACH
 - B ON
 - C REFERENCING
 - D WHEN
13. Se está creando una definición de activador para un activador de inserción. ¿Cuáles cláusulas REFERENCING se pueden incluir en la instrucción CREATE TRIGGER?
 - A REFERENCING OLD ROW AS Old
 - B REFERENCING NEW ROW AS New
 - C REFERENCING OLD TABLE AS Old
 - D REFERENCING NEW TABLE AS New
14. Un activador _____ permite especificar los nombres de columna de una tabla en cuestión.
15. ¿Cuáles palabras clave pueden utilizarse para designar si las instrucciones SQL activadas serán ejecutadas antes o después de que la instrucción de modificación de datos sea aplicada a la tabla en cuestión?
16. Se está creando un activador de actualización en la tabla INVENTARIO_CD. La tabla incluye una columna llamada EN_EXISTENCIA. Se requiere que las instrucciones SQL activadas sean ejecutadas solamente cuando el valor EN_EXISTENCIA de la fila actualizada exceda 20. ¿Cuál cláusula deberá incluirse en la instrucción CREATE TRIGGER para restringir cuándo son ejecutadas las instrucciones?
 - A WHERE
 - B HAVING
 - C FOR EACH
 - D WHEN

- 17.** ¿Qué instrucción debe incluirse en la instrucción CREATE TRIGGER si la definición del activador incluye más de una instrucción SQL activada?
- 18.** ¿Cuál instrucción puede utilizarse para eliminar un activador del esquema?
- 19.** ¿Cuál instrucción SQL se utiliza para alterar una definición de activador?

Capítulo 15

Utilizar cursores SQL

Habilidades y conceptos clave

- Entender los cursores SQL
 - Declarar un cursor
 - Abrir y cerrar un cursor
 - Recuperar datos desde un cursor
 - Utilizar instrucciones UPDATE y DELETE posicionadas
-

Las veces que se han analizado los diferentes aspectos de SQL a través de este libro, se ha utilizado una invocación directa para crear y acceder a diferentes objetos de datos. La *invocación directa*, o *SQL interactivo*, es un método de acceso de datos que soporta la ejecución *ad hoc* de instrucciones SQL, usualmente a través de algún tipo de aplicación de cliente. Por ejemplo, se puede utilizar SQL Server Management Studio o iSQL*Plus de Oracle para interactuar directamente con una base de datos SQL. Sin embargo, la invocación directa generalmente representa solamente un pequeño porcentaje de todo el uso de una base de datos. Un método mucho más común que se utiliza para acceder a las bases de datos SQL es el *SQL incrustado*, un modelo de acceso de datos en el cual las instrucciones SQL están incrustadas en una aplicación de lenguaje de programación, por ejemplo C, Java y COBOL. Para soportar SQL incrustado, el estándar SQL permite declarar cursores que actúan como señalizadores para especificar filas de datos en los resultados de la consulta. Este capítulo explica por qué se utilizan los cursores y cómo pueden ser declarados, abiertos y cerrados, dentro de una sesión SQL. También se aprenderá cómo recuperar datos utilizando el cursor para que el lenguaje de programación pueda trabajar con los datos SQL en un formato que la aplicación puede procesar.

Entender los cursores SQL

Una de las características que definen a SQL es el hecho de que los datos en una base de datos SQL pueden ser manejados en conjuntos. De hecho, a menudo se hace referencia a los resultados de una consulta arrojados por las instrucciones SELECT como *conjuntos de resultados*. Cada uno de estos conjuntos de resultados está conformado por una o más filas extraídas desde una o más tablas.

Cuando se trabaja con los datos SQL de forma interactiva, tener los datos arrojados en conjuntos raramente representa un problema debido a que normalmente es posible desplazarse a través de los resultados de la consulta para encontrar la información que se necesita. Si el tamaño de los resultados es demasiado extenso para desplazarse fácilmente, es posible hacer más preciso el enfoque de la expresión de búsqueda para arrojar un conjunto de resultados más manejable. Sin embargo, la mayoría de los accesos de datos se hace a través de medios diferentes a la invocación directa (a pesar del hecho de que accedemos a los datos de forma interactiva a través de todo el libro). Uno de los métodos más comunes, el SQL incrustado, accede a los datos a través de instrucciones SQL incrustadas en un programa de aplicación. Los elementos de datos arrojados por las instrucciones SQL son utilizados por un lenguaje de programación externo (el *lenguaje host*) para soportar procesos de aplicación específicos.

El problema característico que se encuentra con esta disposición es que los lenguajes de programación de aplicación generalmente no están equipados para tratar con los datos arrojados en conjuntos. Como resultado, existe una incongruencia en la impedancia entre SQL y los lenguajes de programación. La *incongruencia de la impedancia* se refiere a las diferencias entre SQL y otros lenguajes de programación. Como se puede recordar del capítulo 3, un ejemplo de incongruencia en la impedancia es la forma en la cual los tipos de datos SQL difieren de los tipos de datos en otros lenguajes de programación. Estas diferencias pueden llevar a la pérdida de información cuando una aplicación extrae datos desde una base de datos SQL. Otro ejemplo de esta incongruencia es el hecho de que SQL arroja datos en conjuntos, pero otros lenguajes de programación no pueden manejar los conjuntos. Generalmente, éstos solamente pueden procesar partes pequeñas de datos (un único registro) a la vez. La forma en que SQL puede corregir este tipo de incongruencia en la impedancia es mediante el uso de cursores.

Un cursor funciona como un señalador que permite al lenguaje de programación de aplicación tratar a los resultados de la consulta una fila a la vez, de manera muy parecida a la que estos lenguajes de programación manejan los registros desde archivos de datos tradicionales (planos). A pesar de que el cursor puede recorrer todas las filas de los resultados de la consulta, se enfoca solamente en una fila a la vez. Un cursor aun así arroja un conjunto de resultados completo, pero permite al lenguaje de programación convocar solamente una fila de ese conjunto. Por ejemplo, supongamos que los resultados de la consulta se derivan de la siguiente instrucción SELECT:

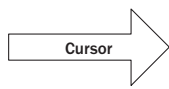
```
SELECT NOMBRE_INTERPRETE, LUGAR_DE_NACIMIENTO  
FROM INTERPRETES;
```

Los resultados de la consulta de esta instrucción arrojarán filas de la tabla INTERPRETES, que incluyen la columna NOMBRE_INTERPRETE y la columna LUGAR_DE_NACIMIENTO. Sin embargo, el lenguaje de programación de aplicación puede tratar solamente con una fila a la vez, por lo que el cursor se declara como una instrucción SQL incrustada dentro del lenguaje de programación de aplicación. Entonces se abre el cursor, de una forma muy parecida a la que estos lenguajes de aplicación abren archivos, y se recupera una fila de los resultados de la consulta. La figura 15-1 ilustra cómo un cursor actúa como un señalador para recuperar solamente una fila de datos.

En este caso, la fila que se recupera a través del cursor es la fila Bing Crosby. Sin embargo, se puede recuperar cualquier fila de los resultados de la consulta, o continuar recuperando filas, siempre y cuando las filas sean recuperadas una a la vez y el cursor permanezca abierto. Una vez que el cursor es cerrado, ya no es posible recuperar ninguna fila de los resultados de la consulta.

Declarar y abrir cursores SQL

La mayoría de los lenguajes de programación de aplicación soportan el uso de cursores para recuperar datos de una base de datos SQL. El lenguaje del cursor está incrustado en el código de programación de una forma muy parecida a la que se incrustaría cualquier instrucción SQL. Cuando se utiliza un cursor en un lenguaje de programación, primero se debe declarar el cursor (similar a como se declararía una variable) y luego utilizar el nombre de la instrucción (el nombre que se le asignó al cursor) en otras instrucciones SQL incrustadas para abrir el cursor, recuperar filas individuales a través del cursor y cerrar el cursor.



NOMBRE_INTERPRETES: VARCHAR (60)	LUGAR_DE_NACIMIENTO VARCHAR (60)
Jennifer Warnes	Seattle, Washington, USA
Joni Mitchell	Fort MacLeod, Alberta, Canada
William Ackerman	Germany
Kitaro	Toyohashi, Japan
Bing Crosby	Tacoma, Washington, United States
Patsy Cline	Winchester, Virginia, United States
Jose Carreras	Barcelona, Spain
Luciano Pavarotti	Modena, Italy
Placido Domingo	Madrid, Spain

Figura 15-1 Utilizar un cursor para acceder a la tabla INTERPRETES.

NOTA

También es posible utilizar cursores en módulos cliente SQL, que son conjuntos de instrucciones SQL que pueden ser convocados desde dentro de un lenguaje de programación de aplicación. Los módulos cliente, en conjunto con SQL incrustado y SQL interactivo, proporcionan un método más para invocar instrucciones SQL. Debido a que los módulos cliente no están implementados tan ampliamente como SQL incrustado, nos enfocaremos a utilizar cursores en SQL incrustado. Para mayor información acerca de los módulos cliente SQL, ver el capítulo 17.

A pesar de que declarar un cursor es fundamental al utilizar ese cursor en la aplicación, la instrucción por sí sola no es suficiente para extraer los datos de una base de datos SQL. De hecho, la funcionalidad completa del cursor es soportada mediante el uso de cuatro instrucciones SQL, cada una de las cuales es incrustada en el lenguaje de programación de aplicación, o lenguaje host. Las siguientes descripciones proporcionan una idea general de estas cuatro instrucciones:

- **DECLARE CURSOR** Declara el cursor SQL al definir el nombre del cursor, sus características y una expresión de consulta que es invocada cuando se abre el cursor.
- **OPEN** Abre el cursor e invoca la expresión de consulta, haciendo que los resultados de consulta estén disponibles para las instrucciones FETCH.
- **FETCH** Recupera datos en las variables que pasan los datos al lenguaje de programación host o a otras instrucciones SQL incrustadas.
- **CLOSE** Cierra el cursor. Una vez que el cursor es cerrado, no pueden recuperarse datos de los resultados de la consulta del cursor.

Las cuatro instrucciones son convocadas desde dentro del lenguaje host. La figura 15-2 ilustra cómo se utilizan las instrucciones relacionadas con el cursor. Las instrucciones SQL incrustadas se muestran en los cuadros sombreados en color gris.

Como se puede ver, primero se debe declarar el cursor y luego abrirlo. Una vez que se ha abierto el cursor, se puede utilizar la instrucción FETCH para recuperar filas de datos. Se puede

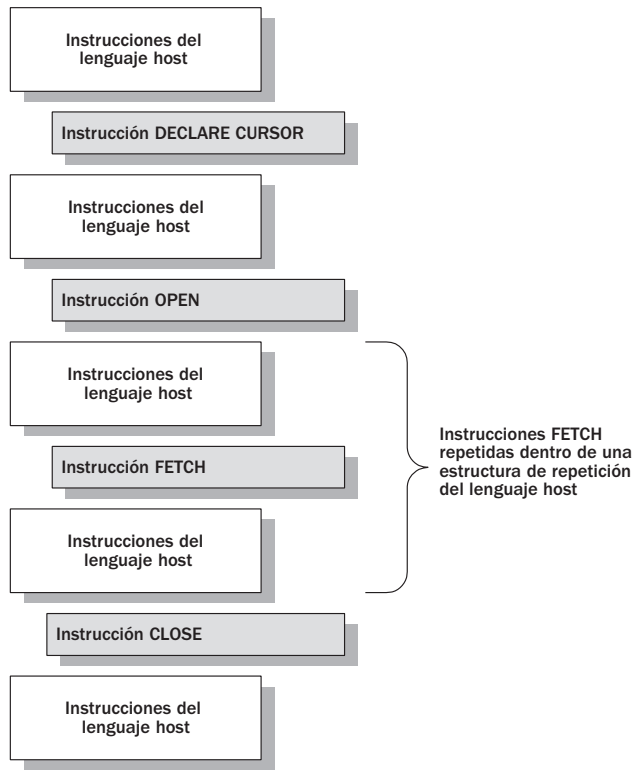


Figura 15-2 Incrustando instrucciones SQL relacionadas con el cursor.

utilizar esta instrucción tantas veces como sea necesario, usualmente dentro de algún tipo de estructura de repetición definida por el lenguaje host. Una vez que se han recuperado todos los datos necesarios, se deberá cerrar el cursor.

NOTA

Para la mayoría de los lenguajes de programación de aplicación, una instrucción SQL incrustada es precedida por EXEC SQL. Esto señala a un preprocesador que la siguiente instrucción es SQL y que debe ser procesada separadamente del lenguaje host. El preprocesador, proporcionado por el proveedor de RDBMS, analiza el código SQL y lo convierte a una forma que pueda ser utilizada por la implementación SQL. El lenguaje host se compila de la forma normal. Para mayor información acerca de SQL incrustado, ver el capítulo 17.

Declarar un cursor

La primera instrucción que veremos será la instrucción DECLARE CURSOR. El cursor debe ser declarado antes de que pueda utilizarse para recuperar datos. Se puede declarar un cursor en cualquier punto del código de aplicación, siempre y cuando sea declarado antes de que se haga referencia al cursor por medio de cualquier otra instrucción.

NOTA

Muchos programadores prefieren declarar todos los cursores y variables al inicio del programa para que se conserven juntas todas las instrucciones. Por lo tanto, se puede hacer referencia a los cursores y variables en cualquier punto del programa.

La sintaxis para una instrucción de cursor incluye muchos elementos, como se muestra en la siguiente sintaxis:

```
DECLARE <nombre del cursor>
[ SENSITIVE | INSENSITIVE | ASENSITIVE ]
[ SCROLL | NO SCROLL ] CURSOR
[ WITH HOLD | WITHOUT HOLD ]
[ WITH RETURN | WITHOUT RETURN ]
FOR <expresión de consulta>
[ ORDER BY <especificación del tipo> ]
[ FOR { READ ONLY | UPDATE [ OF <lista de la columna> ] } ]
```

NOTA

Oracle utiliza la palabra clave IS en lugar de FOR precediendo la expresión de consulta en la instrucción del cursor.

Como se puede ver, la mayoría de los elementos que conforman la instrucción son opcionales. Y como siempre, es necesario revisar la documentación de su implementación SQL para verificar cuáles de éstos son soportados. Se verán estos elementos con mucho mayor detalle en la siguiente sección. Por ahora nos enfocaremos en aquellos elementos que son obligatorios. Para hacerlo, podemos sintetizar la sintaxis hasta los siguientes elementos básicos:

```
DECLARE <nombre del cursor> CURSOR FOR <expresión de consulta>
```

Esta sintaxis solamente permite aquellas partes de la instrucción del cursor que son obligatorias. Como se puede ver, ésta es una parte de código mucho más manejable. Todo lo que se necesita proporcionar es un nombre para el cursor y la expresión de consulta que será invocada cuando el cursor sea abierto. El nombre debe ser diferente de cualquier otro nombre de cursor declarado dentro del mismo programa. La expresión de consulta es básicamente una instrucción SELECT, como las que se han visto a lo largo de este libro.

Eso es todo respecto a la sintaxis básica. En la siguiente sección daremos un vistazo a cada uno de los elementos opcionales que conforman la instrucción del cursor. Después de eso, analizaremos algunos ejemplos.

Trabajar con elementos opcionales de la sintaxis

Si recordamos la sintaxis completa de la instrucción de un cursor (mostrada en la sección anterior), se verá que la mayoría de los elementos son opcionales. En esta sección analizaremos cada uno de esos elementos. Posteriormente en este capítulo, después de que se haya completado esta discusión, encontrará que es necesario regresar a esta sección para detalles acerca de las opciones específicas.

Sensibilidad del cursor

El primer elemento opcional de la instrucción `DECLARE CURSOR` que veremos es la sensibilidad del cursor, que se representa en la siguiente sintaxis:

[`SENSITIVE` | `INSENSITIVE` | `ASENSITIVE`]

La sensibilidad del cursor está relacionada con las instrucciones fuera del cursor que afectan a las mismas filas que aquellas arrojadas por el cursor. Por ejemplo, supongamos que el cursor arroja filas de la tabla `CDS_EN_EXISTENCIA`. Mientras el cursor está abierto, otra instrucción dentro de la misma transacción elimina algunas de las mismas filas en la tabla `CDS_EN_EXISTENCIA` que fueron arrojadas por el cursor. De la sensibilidad del cursor dependerá si es que el cursor puede o no captar estas eliminaciones.

Como se puede ver en la sintaxis, SQL soporta tres opciones de sensibilidad del cursor:

- **SENSITIVE** Cambios significativos hechos por las instrucciones fuera del cursor afectan inmediatamente a los resultados de la consulta dentro del cursor.
- **INSENSITIVE** Cambios significativos hechos por las instrucciones fuera del cursor no afectan a los resultados de la consulta dentro del cursor.
- **ASENSITIVE** La sensibilidad del cursor es definida por la implementación. Los cambios significativos pueden o no ser captados dentro del cursor.

Si no se especifica ninguna opción de sensibilidad del cursor, la opción `ASENSITIVE` se toma de manera preestablecida, en cuyo caso la implementación SQL puede llevar a cabo cualquier acción para la que haya sido diseñada.

NOTA

Algunas implementaciones SQL proporcionan una inicialización o parámetro del sistema que altera el comportamiento de los cursores. Por ejemplo, Oracle proporciona el parámetro de inicialización `CURSOR_SHARING` que especifica qué tipo de instrucciones SQL pueden compartir los mismos cursores. Como siempre, deberá consultar la documentación de su implementación en particular.

Capacidad de desplazamiento del cursor

El siguiente elemento opcional en la instrucción `DECLARE CURSOR` que veremos es la capacidad de desplazamiento del cursor, como muestra la siguiente sintaxis:

[`SCROLL` | `NO SCROLL`]

El desplazamiento está directamente relacionado con la instrucción `FETCH` y las opciones que dicha instrucción puede utilizar para recuperar los datos. Si se especifica la opción `SCROLL`, la instrucción `FETCH` puede ser definida con una de muchas opciones que extienda su habilidad para moverse a través de los resultados de la consulta y arrojar filas específicas. La opción `SCROLL` permite a la instrucción `FETCH` brincar a través de los resultados de la consulta según sea necesario para recuperar la fila específica. Si se especifica `NO SCROLL` en la instrucción del cursor, la instrucción `FETCH` no puede utilizar las opciones de desplazamiento adicionales y solamente puede recuperar la siguiente fila disponible de los resultados de la consulta. Si no se especifica ninguna opción, se toma de modo predeterminado `NO SCROLL`. Para mayor información acerca de las opciones `FETCH`, véase la sección “Recuperar datos desde un cursor”, más adelante en este capítulo.

Capacidad para mantener abierto el cursor

El siguiente elemento que veremos en la sintaxis `DECLARE CURSOR` está relacionado con la capacidad para mantener abierto el cursor, como se muestra en la siguiente sintaxis:

[`WITH HOLD` | `WITHOUT HOLD`]

La capacidad para mantener abierto el cursor hace referencia a una característica en los cursores que está relacionada con la condición de cerrar o no un cursor automáticamente cuando la transacción en la que el cursor fue abierto es completada. Una *transacción* es una unidad atómica de trabajo. Esto significa que todas las instrucciones dentro de la transacción deben ser exitosas o ninguna de ellas podrá tener algún efecto. Si ciertas instrucciones dentro de la transacción son ejecutadas y luego una de ellas falla, ninguna de las instrucciones ejecutadas tendrá efecto y la base de datos permanecerá sin cambios. (Las transacciones se analizan con mayor detalle en el capítulo 16.)

SQL proporciona dos opciones a elegir al activar la definición de la capacidad para mantener abierto el cursor: `WITH HOLD` y `WITHOUT HOLD`. Si se especifica `WITH HOLD`, el cursor permanecerá abierto después de completar la transacción, hasta que el usuario explícitamente lo cierre. Si se especifica `WITHOUT HOLD`, el cursor será cerrado automáticamente cuando la transacción sea completada. Si no se especifica ninguna de estas opciones, se toma de manera preestablecida `WITHOUT HOLD` y el cursor es cerrado automáticamente.

NOTA

Incluso si el cursor se define como un cursor `WITHOUT HOLD` (ya sea explícitamente o de modo predeterminado), por lo general se considera una buena práctica cerrar el cursor explícitamente cuando ya no sea necesario. Esto puede liberar recursos del sistema, y ayuda a asegurarse de que el código se esté autodocumentando claramente. En SQL Server, un cursor deberá también perder sus asignaciones después de haberse cerrado para liberar todos sus recursos.

La ventaja de ocupar un cursor que permanece abierto (uno que esté definido con la opción `WITH HOLD`) es que pueden existir ocasiones, después de que una transacción es completada, en que se necesite que el cursor persista con tal de mantener su posición dentro de los resultados de la consulta arrojados por ese cursor. Cerrar un cursor y volver a abrirlo puede a veces hacer más difícil restaurar las condiciones, incluyendo la posición del cursor en los resultados de la consulta, a como estaban exactamente antes de que se cerrara el cursor.

Capacidad del cursor para arrojar resultados

La capacidad del cursor para arrojar resultados es la siguiente opción que veremos en la definición de la instrucción del cursor, y utiliza la siguiente sintaxis:

[`WITH RETURN` | `WITHOUT RETURN`]

Las opciones para arrojar resultados aplican solamente a los cursores que son abiertos en un procedimiento invocado por SQL. Como se podrá recordar del capítulo 13, un *procedimiento invocado por SQL* es un tipo de rutina que es invocado utilizando la instrucción `CALL`. La instrucción `CALL` es una instrucción SQL que invoca procedimientos y permite pasar valores de parámetros a esos procedimientos. Si el cursor no es abierto dentro de un procedimiento, la opción para arrojar resultados no tiene ningún efecto.

Como muestra la sintaxis, SQL soporta dos opciones del cursor para arrojar resultados `WITH RETURN` y `WITHOUT RETURN`. Si se especifica `WITH RETURN`, el cursor es considerado un cursor del conjunto de resultados. Por lo tanto, si se abre el cursor dentro de un procedimiento

invocado por SQL, el conjunto de resultados del cursor es arrojado hacia lo que haya invocado el procedimiento, que puede ser otra rutina invocada por SQL o un programa de lenguaje host. Si se especifica `WITHOUT RETURN`, el conjunto de resultados del cursor es arrojado de la forma normal, sea abierto o no con un procedimiento invocado por SQL. Si no se especifica ninguna opción, se asume de manera preestablecida `WITHOUT RETURN`.

Capacidad de ordenamiento del cursor

La instrucción `DECLARE CURSOR` incluye una cláusula opcional `ORDER BY`, como muestra la siguiente sintaxis:

```
[ ORDER BY <especificación de clasificación> ]
```

Sin duda reconocerá esta cláusula del capítulo 7, cuando se vieron todas las cláusulas básicas de la instrucción `SELECT`. También se recordará que la cláusula `ORDER BY` puede ser utilizada cuando se invoca directamente SQL, pero no en una instrucción SQL incrustada, a menos que esa instrucción esté contenida dentro de una instrucción de cursor.

La cláusula `ORDER BY` permite clasificar los resultados de la consulta arrojados por la especificación de la consulta. En la cláusula se puede especificar cuáles columnas forman la base para clasificar las filas. Se debe tener cuidado respecto al rendimiento debido que un cursor con una cláusula `ORDER BY` puede forzar al motor SQL a recuperar y clasificar el conjunto completo de resultados antes de que la primera fila pueda ser arrojada, y esto puede ser desastroso para el rendimiento de conjuntos con resultados muy grandes. Si se utiliza una cláusula `ORDER BY`, la instrucción `SELECT` del cursor no puede contener una cláusula `GROUP BY` o una cláusula `HAVING`. Además, la porción de la cláusula `SELECT` de la instrucción `NULL` no puede especificar la palabra clave `DISTINCT` o utilizar una función set.

Si la cláusula `ORDER BY` incluye columnas calculadas en los resultados de la consulta (por ejemplo `COLUMNA_A + COLUMNA_B`), se deberá definir un alias para la columna resultante, como en `(COLUMNA_A + COLUMNA_B) AS TOTALES_COLUMNAS`. Adicionalmente, pueden también utilizarse las palabras clave `ASC` y `DESC` para cualquier columna incluida en la especificación de la clasificación para especificar si la columna será clasificada en orden ascendente o descendente, respectivamente, en donde `ASC` es el valor preestablecido. (Para mayor información acerca de la cláusula `ORDER BY`, ver el capítulo 7.)

Capacidad de actualización del cursor

El último elemento opcional de la instrucción `DECLARE CURSOR` que veremos es la capacidad de actualización del cursor, como se muestra en la siguiente sintaxis:

```
[ FOR { READ ONLY | UPDATE [ OF <lista de la columna> ] } ]
```

La capacidad de actualización del cursor se refiere a la habilidad de utilizar una instrucción `UPDATE` o `DELETE` para modificar los datos arrojados por la instrucción `SELECT` del cursor. Como puede verse en la sintaxis, se debe utilizar la palabra clave `FOR` junto con la opción `READ ONLY` o `UPDATE`. Veamos primero la opción `READ ONLY`. Si se especifica `READ ONLY`, no es posible ejecutar una instrucción `UPDATE` o `DELETE` para los resultados de la consulta arrojados por la instrucción `SELECT` del cursor. Por otro lado, si se especifica `UPDATE`, sí es posible ejecutar las instrucciones. Si no se especifica ninguna opción, se toma `UPDATE` de forma predefinida, a menos que otra opción la sobrescriba.

NOTA

En algunos casos, incluso si no se especifica una opción de actualización, el cursor se define como un cursor de sólo lectura debido a que otras opciones pueden evitar que el cursor sea actualizado. Por ejemplo, si se especifica la opción `INSENSITIVE`, el cursor quedará como de sólo lectura. Se obtiene el mismo resultado si se especifica una cláusula `ORDER BY` o una palabra clave `SCROLL`.

Notará que la opción `UPDATE` también permite especificar cuáles columnas pueden ser actualizadas en la tabla subyacente. Para hacer esto debe incluirse la palabra clave `OF`, seguida de uno o más nombres de columna. Si se especifica más de una columna, los nombres de columna deberán estar separados por comas. Si no se especifica ningún nombre de columna (con la palabra clave `OF`), la opción `UPDATE` se aplica a todas las columnas en la tabla subyacente.

Crear una instrucción de cursor

Ahora que hemos visto cada componente de la instrucción `DECLARE CURSOR`, demos un vistazo a algunos ejemplos que pueden ayudar a ilustrar cómo declarar un cursor. Para estos ejemplos utilizaremos la tabla `INVENTARIO_CD`, mostrada la figura 15-3.

DISCO_COMPACTO: VARCHAR (60)	CATEGORIA: VARCHAR (15)	PRECIO: NUMERIC (5,2)	A_LA_MANO: INT
Famous Blue Raincoat	Vocal	16.99	13
Blue	Vocal	14.99	42
Court and Spark	Vocal	14.99	22
Past Light	Instrumental	15.99	17
Kojiki	Instrumental	15.99	6
That Christmas Feeling	Vocal	14.99	8
Patsy Cline: 12 Greatest Hits	Vocal	16.99	32
Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
After the Rain: The Soft Sounds of Erik Satie	Instrumental	16.99	21
Out of Africa	Instrumental	16.99	29
Leonard Cohen The Best of	Vocal	15.99	12
Fundamental	Vocal	15.99	34
Blues on the Bayou	Vocal	14.99	27
Orlando	Instrumental	14.99	5

Figura 15-3 Declarar cursores en la tabla `INVENTARIO_CD`.

El primer ejemplo que veremos es una instrucción de cursor básica que incluye solamente los elementos obligatorios más una cláusula ORDER BY, como se muestra en la siguiente instrucción DECLARE CURSOR:

```
DECLARE CD_1 CURSOR
FOR
    SELECT *
    FROM INVENTARIO_CD
    ORDER BY DISCO_COMPACTO;
```

En esta instrucción se ha declarado un cursor llamado CD_1 y se ha definido una instrucción SELECT. El nombre del cursor sigue a la palabra clave DECLARE. Después del nombre del cursor, se ha incluido la palabra clave CURSOR y la palabra clave FOR. El único elemento adicional es la instrucción SELECT, que incluye una cláusula ORDER BY. La instrucción arroja todas las filas y columnas para la tabla INVENTARIO_CD. Entonces las filas son ordenadas de acuerdo con los valores en la columna DISCO_COMPACTO. Debido a que no se especificaron las palabras clave ASC o DESC, las filas son arrojadas en orden ascendente.

NOTA

En el capítulo 7, cuando se analizó la instrucción SELECT, se explicó que a pesar de que un asterisco puede ser utilizado para arrojar todas las columnas de una tabla, es una mejor práctica identificar cada una de las columnas que se quieran arrojar. Esto es especialmente importante en SQL incrustado debido a que el lenguaje host depende de que ciertos valores (un número específico en un orden específico) sean arrojados de la base de datos. Si se tuviera que hacer cambios en la base de datos, la aplicación podría no operar apropiadamente, y el código de aplicación tendría que ser modificado. Para los ejemplos de este capítulo, a menudo se utilizó un asterisco para simplificar el código y conservar espacio, pero tenga en cuenta que, en casos reales, usualmente se especificaría cada columna.

La cláusula ORDER BY es un elemento importante debido a que el orden en el cual las filas son arrojadas afecta cuáles filas son recuperadas cuando se utiliza una instrucción FETCH. (Se analizará la instrucción FETCH posteriormente en este capítulo, en la sección “Recuperar datos desde un cursor”.) Esto resulta de esta manera especialmente si se define un cursor con capacidad de desplazamiento, como el que se muestra en el siguiente ejemplo:

```
DECLARE CD_2 SCROLL CURSOR
FOR
    SELECT *
    FROM INVENTARIO_CD
    ORDER BY DISCO_COMPACTO
    FOR READ ONLY;
```

Observe que se han añadido dos elementos a esta instrucción: la palabra clave SCROLL y la cláusula FOR READ ONLY. La palabra clave SCROLL señala a la instrucción FETCH que el cursor tiene capacidad de desplazamiento. Como resultado, opciones adicionales pueden ser utilizadas dentro de la instrucción FETCH que extienden cómo la aplicación puede moverse a través de los resultados del cursor. La cláusula FOR READ ONLY indica que no pueden ser utilizadas ni la instrucción UPDATE ni la instrucción DELETE para modificar los datos arrojados por el cursor. Sin embargo, esta cláusula no es necesaria. Debido que la instrucción del cursor incluye la palabra

clave **SCROLL**, y la instrucción **SELECT** incluye una cláusula **ORDER BY**, el cursor es automáticamente limitado a operaciones de sólo lectura. El uso de cualquiera de estas dos opciones (o el uso de la opción **INSENSITIVE**) automáticamente sobrescribe la capacidad de actualización del cursor de manera predeterminada.

El siguiente tipo de instrucción de sólo lectura que veremos también incluye la palabra clave **INSENSITIVE**, como muestra el siguiente ejemplo:

```
DECLARE CD_3 SCROLL INSENSITIVE CURSOR
FOR
  SELECT *
    FROM INVENTARIO_CD
   ORDER BY DISCO_COMPACTO
  FOR READ ONLY;
```

La instrucción de cursor **CD_3** es exactamente igual que la instrucción de cursor **CD_2**, excepto que **CD_3** también ha sido definida como un cursor no sensitivo. Esto significa que, mientras el cursor esté abierto, ninguna modificación hecha a los datos en la tabla subyacente será reflejada en los resultados de la consulta arrojados por el cursor. Desde luego, si se cierra el cursor y vuelve a abrirse, ninguna modificación hecha mientras el cursor estaba abierto originalmente será reflejada en los datos arrojados por el cursor que volvió a abrirse.

Las tres instrucciones de cursor anteriores que hemos visto han sido de sólo lectura.

Ahora demos un vistazo a un cursor actualizable. En la siguiente instrucción de cursor, la instrucción **SELECT** arroja una vez más todas las filas y columnas para la tabla **INVENTARIO_CD**:

```
DECLARE CD_4 CURSOR
FOR
  SELECT *
    FROM INVENTARIO_CD
  FOR UPDATE;
```

Observe que esta instrucción **DECLARE CURSOR** no incluye la palabra clave **SCROLL**, la palabra clave **INSENSITIVE** o una cláusula **ORDER BY**, ya que cualquiera de éstas hubiera evitado crear un cursor con capacidad de actualización. Se pudieron haber especificado las opciones **NO SCROLL** y **SENSITIVE**, pero no son necesarias. Observe también, sin embargo, que la instrucción de cursor sí incluye la cláusula **FOR UPDATE**. La cláusula en sí tampoco es necesaria en esta instrucción en particular debido a que el cursor es actualizable de manera preestablecida, ya que no contiene ninguna opción para limitar su capacidad de actualización.

Sin embargo, si se desea que el cursor sea actualizable solamente para cierta columna, se deberá incluir la cláusula **FOR UPDATE**, junto con el nombre de la columna, como se muestran en el siguiente ejemplo:

```
DECLARE CD_5 CURSOR
FOR
  SELECT *
    FROM INVENTARIO_CD
  FOR UPDATE OF DISCO_COMPACTO;
```

Ahora la cláusula **FOR UPDATE** incluye la palabra clave **OF** y el nombre de columna, **DISCO_COMPACTO**. Si se intentara modificar los datos en los resultados del cursor en las columnas diferentes a la columna **DISCO_COMPACTO**, se recibiría un error.

Una vez que se ha declarado el cursor, puede abrirse y recuperar con él los datos de los resultados de la consulta. Sin embargo, como se ha visto en las instrucciones de cursor anteriores, las acciones que se pueden llevar a cabo están limitadas por las restricciones definidas usando la instrucción `DECLARE CURSOR`.

Abrir y cerrar un cursor

El proceso para abrir un cursor es bastante sencillo. Solamente es necesario proporcionar la palabra clave `OPEN` y el nombre del cursor, como se muestra en la siguiente sintaxis:

```
OPEN <nombre del cursor>
```

Por ejemplo, para abrir el cursor `CD_1` se invoca la siguiente instrucción SQL:

```
OPEN CD_1;
```

No es posible abrir un cursor hasta que se haya declarado. Una vez declarado, puede abrirse en cualquier lugar dentro del programa. La instrucción `SELECT` dentro del cursor no es invocada hasta que realmente se abra el cursor. Eso significa que ninguna modificación a los datos hecha entre el momento en que el cursor es declarado y el momento en el que el cursor es abierto se refleja en los resultados de la consulta arrojados por el cursor. Si se cierra el cursor y vuelve a abrirse, las modificaciones de datos que se llevarán a cabo entre el momento en que se cerró y el momento en que volvió a abrirse se reflejan en los nuevos resultados de la consulta.

Una vez que se ha terminado de utilizar el cursor, éste deberá cerrarse para que puedan liberarse los recursos del sistema. Para cerrar un cursor se puede utilizar la instrucción `CLOSE`, como se muestra en la siguiente sintaxis:

```
CLOSE <nombre del cursor>
```

La instrucción `CLOSE` no realiza otra función que cerrar el cursor, lo que significa que los resultados de la consulta de la instrucción `SELECT` del cursor son liberados. Por ejemplo, para cerrar el cursor `CD_1` se utiliza la siguiente instrucción SQL:

```
CLOSE CD_1;
```

Una vez que se cierra el cursor, no es posible recuperar ninguna otra fila desde los resultados de la consulta del cursor. En otras palabras, no se puede utilizar una instrucción `FETCH` para recuperar datos desde un cursor cerrado. Si vuelve a abrirse el cursor, es posible recuperar datos una vez más, pero se obtendrá un nuevo conjunto de resultados y (asumiendo que no hay ninguna opción de desplazamiento) se comenzará con la primera fila de los resultados de la consulta, lo que puede significar recuperar filas que ya hayan sido procesadas por una invocación anterior del cursor.

Recuperar datos desde un cursor

Hasta ahora hemos aprendido a declarar un cursor, a abrirlo y a cerrarlo. Sin embargo, estas acciones por sí solas no permiten recuperar ninguno de los datos proporcionados por el cursor. Para poder hacer eso se debe utilizar una instrucción `FETCH`.

Antes de dar un vistazo a la sintaxis para la instrucción `FETCH`, repasemos brevemente el propósito de un cursor y sus instrucciones relacionadas. Como se dijo anteriormente, uno de los problemas al incrustar instrucciones `SQL` en un lenguaje `host` de programación es la incongruencia en la impedancia. Una forma de esta incongruencia es que `SQL` arroja datos en conjuntos y los lenguajes de programación de aplicación tradicionales no pueden manejar conjuntos de datos. En general, éstos pueden tratar solamente con valores individuales. Para poder resolver esta forma de incongruencia en la impedancia, se pueden utilizar cursores para recuperar datos una fila a la vez (sin importar cuántas filas sean arrojadas) desde los cuales se pueden extraer valores individuales que pueden ser utilizados por el lenguaje `host`.

Como ya hemos visto, una instrucción de cursor incluye una instrucción `SELECT` que arroja un conjunto de datos. La instrucción `OPEN` ejecuta la instrucción `SELECT`, y la instrucción `CLOSE` libera los resultados de la consulta de la instrucción `SELECT`. Sin embargo, es la instrucción `FETCH` la que identifica las filas individuales dentro de ese conjunto de datos y extrae los valores individuales de esas filas, que pasan entonces a las variables `host`. Una *variable host* es un tipo de parámetro que pasa un valor al lenguaje `host`.

Una o más instrucciones `FETCH` pueden ser ejecutadas mientras un cursor está abierto. Cada instrucción apunta a una fila específica en los resultados de la consulta, y los valores son entonces extraídos de esas filas. La siguiente sintaxis muestra los elementos básicos que conforman la instrucción `FETCH`:

```
FETCH [ [ <orientación para búsqueda> ] FROM ]  
<nombre del cursor> INTO <variables host>
```

Como puede verse en la sintaxis, se debe especificar la palabra clave `FETCH`, el nombre del cursor y una cláusula `INTO` que identifique las variables `host` que recibirán los valores arrojados por la instrucción `FETCH`. Esos valores se derivan de los resultados de consulta que son generados por la instrucción `SELECT` del cursor cuando éste está abierto. Si la instrucción `FETCH` incluye más de una variable `host`, las variables deberán estar separadas por comas.

Además de los componentes obligatorios de la instrucción `FETCH`, la sintaxis también incluye el marcador de posición opcional `<orientación para búsqueda>` y la palabra clave `FROM`. Si se especifica una opción de orientación para búsqueda en la instrucción `FETCH`, se debe incluir la palabra clave `FROM`, o se puede especificar `FROM` sin la orientación para búsqueda.

`SQL` soporta seis opciones de orientación para búsqueda que identifican cuál fila es seleccionada de los resultados de la consulta del cursor. La mayoría de estas opciones están disponibles solamente si se declara el cursor como desplazable. Un cursor con capacidad de desplazamiento, como podrá recordar, es uno que extiende la habilidad de la instrucción `FETCH` para moverse a través de los resultados de la consulta del cursor. Un cursor tiene la capacidad de desplazamiento si la instrucción del cursor incluye la palabra clave `SCROLL`. Si se incluye una orientación para búsqueda en la instrucción `FETCH`, es posible escoger una de las siguientes opciones:

- **NEXT** Recupera la siguiente fila de los resultados de la consulta. Si se utiliza `NEXT` en la primera instrucción `FETCH` después de que se abre el cursor, será arrojada la primera fila en los resultados de la consulta. Una segunda instrucción `FETCH NEXT` arrojará la segunda fila.
- **PRIOR** Recupera directamente la fila anterior a la última que se había recuperado. Si se utiliza `PRIOR` en la primera instrucción `FETCH` después de abrir el cursor, ninguna fila será arrojada debido a que ninguna fila precede a la primera fila.

- **FIRST** Recupera la primera fila de los resultados de la consulta del cursor, sin importar cuántas instrucciones FETCH hayan sido ejecutadas desde que se abrió el cursor.
- **LAST** Recupera la última fila de los resultados de la consulta del cursor, sin importar cuántas instrucciones FETCH hayan sido ejecutadas desde que se abrió el cursor.
- **ABSOLUTE <valor>** Recupera la fila especificada por el marcador de posición <valor>. El valor debe ser un numérico exacto, a pesar de que puede ser derivado de una variable host. El numérico identifica cuál fila es arrojada por la instrucción FETCH. Por ejemplo, ABSOLUTE 1 arroja la primera fila, ABSOLUTE 2 arroja la segunda fila y ABSOLUTE -1 arroja la última fila.
- **RELATIVE <valor>** Recupera la fila especificada por el marcador de posición <valor>, relativo a la posición actual del cursor. Si se utiliza RELATIVE en la primera instrucción FETCH después de abrir el cursor, RELATIVE 1 arroja la primera fila de los resultados de la consulta del cursor, y RELATIVE -1 arroja la última fila. Sin embargo, si el cursor no está al comienzo de los resultados de la consulta, como lo está cuando se abre el cursor por primera vez, RELATIVE 1 y RELATIVE -1 arrojan filas relativas a la posición del cursor donde se quedó después de la última instrucción FETCH ejecutada.

En cualquier momento que se abre un cursor, el cursor apunta al comienzo de los resultados de la consulta. La instrucción FETCH mueve el cursor a la fila designada por la opción de orientación para búsqueda. Si no se especifica ninguna opción, se asume NEXT de manera preestablecida, y el cursor siempre apunta a la siguiente fila en los resultados de la consulta.

Para ayudar a ilustrar cómo funcionan las opciones de orientación para búsqueda, veamos otra vez el cursor que declaramos anteriormente en este capítulo:

```
DECLARE CD_2 SCROLL CURSOR
FOR
  SELECT *
    FROM INVENTARIO_CD
   ORDER BY DISCO_COMPACTO
  FOR READ ONLY;
```

Observe que la palabra clave SCROLL está especificada y que la instrucción SELECT recupera todas las filas y columnas de la tabla INVENTARIO_CD. Observe también que la instrucción SELECT incluye una cláusula ORDER BY que ordena los resultados de la consulta en un orden ascendente de acuerdo con los valores en la columna DISCO_COMPACTO. Esto es importante debido a que las instrucciones FETCH se mueven a través de las filas en los resultados de la consulta en el orden especificado por la cláusula ORDER BY, sin importar cuántas filas sean ordenadas en la tabla subyacente.

Ahora demos otro vistazo a los resultados de la consulta arrojados por la instrucción SELECT en el cursor CD_2. Los resultados de la consulta se muestran en la figura 15-4 en forma de una tabla virtual. Observe que la ilustración incluye señaladores que representan los diferentes tipos de instrucciones FETCH (basándose en su orientación para búsqueda). En cada caso, el señalador está basado en una instrucción FETCH que es la primera en ejecutarse después de que el cursor ha sido abierto.

	DISCO_COMPACTO	CATEGORIA	PRECIO	A_LA_MANO
FETCH PRIOR				
FETCH FIRST	After the Rain: The Soft Sounds of Erik Satie	Instrumental	16.99	21
FETCH NEXT	Blue	Vocal	14.99	42
	Blues on the Bayou	Vocal	14.99	27
	Carreras Domingo Pavarotti in Concert	Vocal	15.99	27
FETCH ABSOLUTE 5	Court and Spark	Vocal	14.99	22
	Famous Blue Raincoat	Vocal	16.99	13
	Fundamental	Vocal	15.99	34
	Kojiki	Instrumental	15.99	6
	Leonard Cohen The Best of	Vocal	15.99	12
FETCH RELATIVE 10	Orlando	Instrumental	14.99	5
	Out of Africa	Instrumental	16.99	29
	Past Light	Instrumental	15.99	17
	Patsy Cline: 12 Greatest Hits	Vocal	16.99	32
FETCH LAST	That Christmas Feeling	Vocal	14.99	8

Figura 15-4 Los resultados de la consulta (tabla virtual) arrojados por el cursor CD_2.

Observe que cada uno de los señaladores `FETCH FIRST` y `FETCH NEXT` apuntan hacia la fila *After the Rain*. Ésta es la primera fila en los resultados de la consulta del cursor. `FETCH FIRST` siempre señalará a esta fila, asumiendo que los datos en las tablas subyacentes no cambien. `FETCH NEXT` siempre señalará a la primera fila cada vez que sea la primera instrucción `FETCH` ejecutada después de que el cursor es abierto. Adicionalmente, el señalador `FETCH LAST` siempre señalará a la fila *That Christmas Feeling*. Sin embargo, el señalador `FETCH PRIOR` no señala hacia ninguna fila. En su lugar, señala hacia un espacio anterior a la primera fila de los resultados de la consulta. Esto se debe a que `PRIOR` no puede recuperar una fila si está siendo utilizada en la primera instrucción `FETCH` después de que el cursor es abierto.

Ahora demos un vistazo al señalador `FETCH ABSOLUTE 5`. Como se puede ver, éste apunta a la fila *Court and Spark*, que es la quinta fila en los resultados de la consulta del cursor. `FETCH ABSOLUTE 5` siempre arrojará esta fila. Por otro lado, `FETCH RELATIVE 10` señala a la fila *Orlando*, que es la décima fila en los resultados de la consulta del cursor. Sin embargo, si `RELATIVE`

fuera utilizada en una instrucción `FETCH` diferente a la primera, `FETCH RELATIVE 10` probablemente estaría señalando a una fila diferente.

Como se puede ver, las opciones de la sexta orientación para búsqueda proporcionan una gran flexibilidad al moverse a través de los resultados de la consulta del cursor. Tenga en mente, sin embargo, que la mayoría de estas opciones pueden ser utilizadas solamente en cursores de sólo lectura, como el cursor `CD_2` que hemos estado viendo. La única opción que puede ser utilizada para cursores actualizables es `NEXT`, que es la orientación para búsqueda predeterminada. Ahora veamos algunos ejemplos de las instrucciones `FETCH` para demostrar cómo pueden ser utilizadas para recuperar datos de los resultados de la consulta de un cursor.

La primera instrucción `FETCH` que veremos utiliza la opción de orientación para búsqueda `NEXT` para recuperar una fila del cursor `CD_2`:

```
FETCH NEXT
FROM CD_2
INTO :CD, :Categoria, :Precio, :A_la_mano;
```

La instrucción identifica la orientación para búsqueda y el nombre del cursor. Como puede recordarse, la palabra clave `NEXT` es opcional debido a que `NEXT` es la orientación para búsqueda predeterminada. La instrucción también incluye una cláusula `INTO`, que identifica las variables host que recibirán los valores arrojados por la instrucción `FETCH`. Existen cuatro variables host que coinciden con el número de variables arrojadas por la instrucción `FETCH`. El número de variables debe ser el mismo que el número de columnas arrojadas por la instrucción `SELECT` del cursor, y las variables deben ser listadas en el mismo orden que las columnas arrojadas. Observe que las variables host están separadas por comas y sus nombres empiezan con dos puntos. De acuerdo con el estándar SQL, las variables host deben comenzar con dos puntos, a pesar de lo cual puede variar entre una implementación SQL y otra.

Ahora que se ha visto cómo trabaja la instrucción `FETCH NEXT`, se puede crear cualquier instrucción `FETCH` para cualquier orientación `FETCH` que se quiera especificar. Simplemente es necesario reemplazar una opción con la otra. Por ejemplo, la siguiente instrucción `FETCH` utiliza la orientación para búsqueda `ABSOLUTE`:

```
FETCH ABSOLUTE 5
FROM CD_2
INTO :CD, :Categoria, :Precio, :A_la_mano;
```

Observe que con la opción `ABSOLUTE`, al igual que con la opción `RELATIVE`, se debe especificar un valor numérico. En este caso, el cursor recuperará la quinta fila de los resultados de la consulta del cursor. Las opciones `ABSOLUTE`, `FIRST` y `LAST` son las únicas opciones de orientación para búsqueda que arrojarán siempre la misma fila para los resultados de la consulta del cursor, asumiendo que los datos en la tabla subyacente no han cambiado. Por otro lado, las opciones `NEXT`, `PRIOR` y `RELATIVE` arrojan filas basadas en la última posición del cursor. Como resultado, es necesario estar seguro acerca del diseño de los cursores y de las instrucciones `FETCH` teniendo la posición en mente.

Pregunta al experto

P: Se ha mencionado que una instrucción `SELECT` de un cursor no se ejecuta hasta que el cursor esté abierto. ¿Cómo afecta esto a valores especiales como `CURRENT_USER` o `CURRENT_TIME`?

R: Debido a que una instrucción `SELECT` de un cursor no se ejecuta hasta que el cursor esté abierto, no se asignan valores a los valores especiales sino hasta que el cursor es abierto, y no cuando el cursor es declarado. Por ejemplo, si se incluye el valor especial `CURRENT_TIME` en la instrucción `SELECT` del cursor y se declara ese cursor al inicio del código del programa, la hora asignada al valor `CURRENT_TIME` será el momento en que el cursor sea abierto, y no la hora de cuando el cursor sea declarado. Adicionalmente, si se cierra el cursor y vuelve a abrirse, el valor `CURRENT_TIME` tendrá la hora de cuando el cursor volvió a abrirse, y no de cuando fue abierto por primera vez.

P: Se dijo que las variables host son un tipo de parámetro que se utiliza en SQL incrustado. ¿Cómo difieren las variables host de otros tipos de parámetros?

R: Para todo propósito práctico, una variable host es igual que cualquier otro parámetro. La principal diferencia es que una variable host es utilizada en SQL incrustado para pasar valores entre el lenguaje host y SQL. La otra distinción real es que el símbolo dos puntos debe ser añadido al nombre de la variable. La razón por la que los dos puntos deben ser incluidos cuando se utiliza una instrucción SQL incrustada es para indicar que el nombre es de una variable host y no de una columna. Como resultado, se pueden utilizar nombres de variable que sean significativos a la aplicación sin preocuparse por nombrar accidentalmente a una variable con el mismo nombre que una columna. Los dos puntos no tienen nada que ver con la variable en sí misma, pero sí para distinguirla como una variable. Los dos puntos deben ser utilizados en los módulos cliente SQL. Sin embargo, los valores son pasados hacia los módulos a través de parámetros, en lugar de a través de variables host. Los parámetros de módulo son esencialmente lo mismo que las variables host; solamente los nombres son diferentes. Si quisiera referirse a todos ellos como parámetros, no se estaría muy equivocado.

Utilizar instrucciones `UPDATE` y `DELETE` posicionadas

Una vez que se busca una fila desde los resultados de la consulta de un cursor actualizable, es posible que sea necesario que la aplicación actualice o elimine esa fila. Para hacerlo se debe utilizar una instrucción `UPDATE` o `DELETE` posicionada. Las instrucciones `UPDATE` y `DELETE` posicionadas contienen una cláusula especial `WHERE` que hace referencia al cursor abierto. Demos un vistazo a cada una de estas dos instrucciones para mostrar cómo pueden utilizarse para modificar los datos arrojados por el cursor.

Utilizar la instrucción UPDATE posicionada

La instrucción UPDATE posicionada es, en su mayor parte, igual que una instrucción UPDATE regular, excepto que ésta requiere una cláusula especial WHERE, como se muestra en la siguiente sintaxis:

```
UPDATE <nombre de la tabla>  
    SET <lista de conjunto>  
    WHERE CURRENT OF <nombre del cursor>
```

Una instrucción UPDATE regular, como sin duda recordará, contiene la cláusula UPDATE y la cláusula SET, al igual que como se ve en la sintaxis para una instrucción UPDATE posicionada. Sin embargo, en la instrucción UPDATE regular la cláusula WHERE es opcional, mientras que en una instrucción UPDATE posicionada es obligatoria. Además, la cláusula WHERE debe estar definida con la opción CURRENT OF, que identifica al cursor abierto. Al utilizar la opción CURRENT OF, se está avisando a la aplicación que utilice los valores arrojados por la instrucción FETCH más reciente para el cursor al que hace referencia. Por ejemplo, si el cursor está señalando a la fila Past Light de la tabla INVENTARIO_CD (la fila arrojada más recientemente por la instrucción FETCH), es esa fila a la que se está haciendo referencia con la cláusula WHERE de la instrucción UPDATE posicionada.

Demos un vistazo a un ejemplo para demostrar cómo funciona esto. El siguiente conjunto de instrucciones SQL declara el cursor CD_4, abre ese cursor, busca una fila desde los resultados de la consulta del cursor, actualiza esa fila y luego cierra el cursor:

```
DECLARE CD_4 CURSOR  
FOR  
    SELECT *  
    FROM INVENTARIO_CD  
    FOR UPDATE;  
  
OPEN CD_4;  
  
FETCH CD_4  
    INTO :CD, :Categoria, :Precio, :A_la_mano;  
  
UPDATE INVENTARIO_CD  
    SET A_LA_MANO = :A_la_mano * 2  
    WHERE CURRENT OF CD_4;  
  
CLOSE CD_4;
```

Se añadieron algunas líneas vacías para hacer la lectura más fácil, pero desde luego que no son necesarias, y si se incluyeran, el motor SQL simplemente las ignoraría. La primera instrucción declara el cursor CD_4 y define la instrucción SELECT que arroja todas las filas y columnas de la tabla INVENTARIO_CD. Luego se abre el cursor y se realiza la búsqueda en la siguiente fila, que en este caso es la primera fila, Famous Blue Raincoat. Después de realizar la búsqueda y la fila, se utiliza una instrucción UPDATE posicionada para duplicar la cantidad del valor A_LA_MANO para esa fila. Observe que la instrucción UPDATE incluye una cláusula WHERE que contiene la opción CURRENT OF, que identifica al cursor CD_4. Después de actualizar la fila, se cierra el cursor.

NOTA

Tenga en mente que las instrucciones mostradas en el ejemplo anterior podrían estar incrustadas en un lenguaje host, por lo que no es muy probable que estén agrupadas tan cercanamente y a su vez es muy probable que tengan otros elementos del lenguaje host, como instrucciones de variables, estructuras de repetición e instrucciones condicionales.

En el ejemplo anterior fue posible actualizar la columna A_LA_MANO debido a que estaba incluida implícitamente en la cláusula FOR UPDATE de la instrucción SELECT del cursor. Cuando no se especifican nombres de columna, todas las columnas son actualizables. Sin embargo, veamos otro ejemplo que define explícitamente una columna. En el siguiente conjunto de instrucciones SQL, se declara el cursor CD_5 y se utiliza para intentar actualizar una fila en la tabla INVENTARIO_CD:

```
DECLARE CD_5 CURSOR
FOR
    SELECT *
      FROM INVENTARIO_CD
     FOR UPDATE OF DISCO_COMPACTO;

OPEN CD_5;

FETCH CD_5
  INTO :CD, :Categoria, :Precio, :A_la_mano;

UPDATE INVENTARIO_CD
  SET A_LA_MANO = :A_la_mano * 2
 WHERE CURRENT OF CD_5;

CLOSE CD_5;
```

Como se puede ver, la instrucción del cursor especifica la columna DISCO_COMPACTO en la cláusula FOR UPDATE. Si se intenta ejecutar la instrucción UPDATE, se recibirá un error indicando que la columna A_LA_MANO no es una de las columnas especificadas en la instrucción del cursor.

Utilizar la instrucción DELETE posicionada

La instrucción DELETE posicionada, al igual que la instrucción UPDATE posicionada, requiere una cláusula WHERE que debe incluir la opción CURRENT OF. (Una instrucción DELETE regular, como puede recordarse, no requiere de una cláusula WHERE.) Una instrucción DELETE posicionada utiliza la siguiente sintaxis:

```
DELETE <nombre de la tabla>
WHERE CURRENT OF <nombre del cursor>
```

Como se puede ver, es necesario definir una cláusula DELETE que identifique a la tabla y una cláusula WHERE que identifique al cursor. La cláusula WHERE, en una instrucción DELETE posicionada, funciona de la misma manera que la cláusula WHERE en una instrucción UPDATE posicionada: la fila arrojada por la última instrucción FETCH es la fila que se modifica. En este caso, la fila es eliminada.

Ahora veamos un ejemplo de una instrucción DELETE posicionada. Las siguientes instrucciones SQL declaran el cursor CD_4, abren el cursor, arrojan una fila desde el cursor, eliminan esa fila y cierran el cursor:

```

DECLARE CD_4 CURSOR
FOR
    SELECT *
      FROM INVENTARIO_CD
     FOR UPDATE;

OPEN CD_4;

FETCH CD_4
  INTO :CD, :Categoria, :Precio, :A_la_mano;

DELETE INVENTARIO_CD
  WHERE CURRENT OF CD_4;

CLOSE CD_4;

```

Todas estas instrucciones han sido utilizadas antes. El único elemento nuevo es la instrucción DELETE posicionada. Esta instrucción elimina la fila arrojada por la instrucción FETCH, que es la fila Famous Blue Raincoat. Una vez que la fila es eliminada, el cursor es cerrado utilizando una instrucción CLOSE. Como ya se dijo anteriormente, siempre es una buena idea cerrar explícitamente los cursores cuando ya no sean necesarios.

Pruebe esto 15-1 Trabajar con cursores SQL

En este capítulo se vio cómo declarar cursores, abrir esos cursores, recuperar datos desde ellos y luego cómo cerrarlos. Además, se repasaron las instrucciones UPDATE y DELETE posicionadas. Sin embargo, como se mencionó anteriormente, los cursores son utilizados principalmente en SQL incrustado, lo cual hace difícil probar por completo la funcionalidad del cursor si se está limitado a invocar de manera directa las instrucciones SQL (como se está en este ejercicio). Idealmente, sería mejor incrustar las instrucciones SQL relacionadas con el cursor en un lenguaje host, pero esto excede el campo de acción de este libro. Lo que complica incluso más este tema es el hecho de que diferentes implementaciones SQL soportan de diferente forma el uso de cursores en un ambiente interactivo, lo que hace muy difícil invocar directamente las instrucciones relacionadas con el cursor. A pesar de eso, deberá ser posible ejecutar la mayoría de las instrucciones relacionadas con el cursor de manera interactiva, pero habrá que tener en mente que los cursores están diseñados para utilizarse en SQL incrustado y en los módulos de cliente SQL, por lo que puede ser necesario modificar las instrucciones en un grado alto para poder ejecutarlas. Puede descargar el archivo Try_This_15.txt (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

NOTA

Lo ideal sería llevar al lector a través de cada paso para declarar y abrir un cursor, recuperar los datos y cerrar un cursor, pero debido a la naturaleza de la invocación directa, utilizaremos menos pasos y bloques más largos de instrucciones.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. El primer cursor que se declarará y accederá es un cursor básico de sólo lectura que recupera datos desde la tabla DISCO_COMPACTO. Lo primero que notará de este conjunto de ins-

(continúa)

trucciones a crear es que se declarará una variable llamada `v_NOMBRE_CD`. Se creará esta variable para poder probar completamente la instrucción `FETCH`. Se debe tener en mente que, dependiendo de la situación, el lenguaje host, y el producto, puede o no utilizarse este método para definir la variable. Observe también que el nombre de la variable en la instrucción `FETCH` no está precedido por dos puntos. Esto se debe a que se estará utilizando invocación directa para ejecutar estas instrucciones y, para la mayoría de las implementaciones, el nombre de la variable en la instrucción `FETCH` tendrá que ser el mismo que el nombre que se declaró al principio de este conjunto de instrucciones.

Al igual que con cualquier instrucción SQL, se encontrará que el lenguaje exacto a utilizar para crear las instrucciones varía de un producto al otro. Además, el hecho de estar invocando las instrucciones directamente en lugar de incrustándolas puede llevar a otras variaciones entre SQL y la implementación (por ejemplo, a no utilizar un símbolo de punto y coma en el nombre de la variable). Como ejemplo, si se ejecutan estas instrucciones en SQL Server, se tendrá que preceder los nombres de la variable con el carácter arroba (`@`). Oracle se desvía aún más del estándar. En Oracle se declara el cursor y la variable en un bloque de instrucciones. Además, la palabra clave `CURSOR` precede al nombre del cursor, y se debe utilizar la palabra clave `IS` en lugar de la palabra clave `FOR`. También se deben encerrar las instrucciones `OPEN`, `FETCH` y `CLOSE` en un bloque `BEGIN...END`. De la misma manera, se encontrará que no todas las opciones SQL son soportadas en todas las implementaciones SQL, y muchos productos incluyen características adicionales no definidas en el estándar SQL. Asegúrese de revisar la documentación de su producto antes de intentar declarar y acceder a cualquier cursor.

Ahora vamos a crear las instrucciones relacionadas con el cursor. Ingrese y ejecute las siguientes instrucciones SQL:

```
DECLARE v_NOMBRE_CD VARCHAR (60);

DECLARE CD_cursor_1 CURSOR
FOR
    SELECT TITULO_CD
        FROM DISCOS_COMPACTOS
        ORDER BY TITULO_CD ASC;

OPEN CD_cursor_1;

FETCH CD_cursor_1 INTO v_NOMBRE_CD;

CLOSE CD_cursor_1;
```

En estas instrucciones, primero se declaró una variable llamada `v_NOMBRE_CD`. Después, se declaró un cursor llamado `CD_cursor_1`. La definición del cursor contenía una instrucción `SELECT` que fue cualificada con la cláusula `ORDER BY`. Debido a que se incluyó la cláusula `ORDER BY`, el cursor fue de sólo lectura. Después de que se declaró el cursor, fue abierto, se buscó una fila de los resultados de la consulta del cursor y luego se cerró el cursor. La instrucción `FETCH` arrojó el valor *After the Rain: The Soft Sounds of Erik Satie*, que pudo entonces haber sido utilizada en alguna otra operación, que hubiera incrustado estas instrucciones. Después de haber ejecutado las instrucciones, se deberá recibir un mensaje indicando que las instrucciones fueron ejecutadas exitosamente.

3. Ahora se declarará y accederá a un segundo cursor. Esta vez se especificará que el cursor sea no sensitivo y con capacidad de desplazamiento. Además, se especificará que el cursor sea de sólo lectura, aunque esta cláusula es opcional debido a que se está haciendo que el cursor sea no sensitivo y con capacidad de desplazamiento. También se buscará la última fila de los resultados de la consulta del cursor en lugar de la primera. Ingrese y ejecute la siguiente instrucción SQL:

```
DECLARE v_NOMBRE_CD VARCHAR(60);

DECLARE CD_cursor_2 SCROLL INSENSITIVE CURSOR
FOR
    SELECT TITULO_CD
    FROM DISCOS_COMPACTOS
    ORDER BY TITULO_CD ASC
    FOR READ ONLY;

OPEN CD_cursor_2;

FETCH LAST FROM CD_cursor_2 INTO v_NOMBRE_CD;

CLOSE CD_cursor_2;
```

Esta vez la instrucción `FETCH` recuperó el valor `That Christmas Feeling` debido a que se especificó `LAST`. Este valor fue insertado en la variable `v_NOMBRE_CD`. Después de haber ejecutado las instrucciones, se deberá recibir un mensaje indicando que las instrucciones fueron ejecutadas exitosamente.

4. El siguiente cursor tendrá capacidades de actualización, lo que significa que no puede incluir una cláusula `ORDER BY` y no puede ser definido como no sensitivo o con capacidad de desplazamiento. Debido a que el cursor es actualizable, también se creará una instrucción `UPDATE` que duplique el valor de la columna `EN_EXISTENCIA` para la fila arrojada por la instrucción `FETCH`. Ingrese y ejecute la siguiente instrucción SQL:

```
DECLARE v_NOMBRE_CD VARCHAR(60);

DECLARE CD_cursor_3 CURSOR
FOR
    SELECT TITULO_CD
    FROM DISCOS_COMPACTOS
    FOR UPDATE;

OPEN CD_cursor_3;

FETCH CD_cursor_3 INTO v_NOMBRE_CD;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = EN_EXISTENCIA * 2
    WHERE CURRENT OF CD_cursor_3;

CLOSE CD_cursor_3;
```

(continúa)

Observe que esa instrucción UPDATE incluye una cláusula WHERE que contiene la opción CURRENT OF, que especifica el cursor CD_cursor_3. Esta cláusula es obligatoria. Debido a que no se utilizó una cláusula ORDER BY, la primera fila en los resultados de la consulta del cursor fue Famous Blue Raincoat. Ésta es la fila que fue actualizada. Después de haber ejecutado las instrucciones, se deberá recibir un mensaje indicando que una fila ha sido actualizada.

- 5.** Ahora demos un vistazo a la tabla DISCOS_COMPACTOS para verificar que el cambio que se ha hecho es correcto. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT * FROM DISCOS_COMPACTOS;
```

El valor IN_STOCK de la fila Famous Blue Raincoat deberá ser ahora de 26, el doble de su cantidad original.

- 6.** Ahora regresemos la base de datos a su estado original. Ingrese y ejecute la siguiente instrucción SQL:

```
UPDATE DISCOS_COMPACTOS
  SET EN_EXISTENCIA = 13
  WHERE ID_DISCO_COMPACTO = 101;
```

Se deberá recibir un mensaje indicando que la fila ha sido actualizada.

- 7.** Cierre la aplicación cliente.

Resumen de Pruebe esto

En este ejercicio se declararon y se accedió a tres cursores, dos que eran de sólo lectura y uno que era actualizable. Para cada uno de los tres cursores se declaró una variable. La variable fue luego utilizada en la instrucción FETCH para recibir el valor arrojado por esa instrucción. Para el cursor con capacidades de actualización, se creó una instrucción UPDATE que modificó el valor EN_EXISTENCIA para la fila arrojada por la instrucción FETCH. Después se actualizó la tabla DISCOS_COMPACTOS, y se actualizó una vez más para arrojar la base de datos a su estado original. Debido a que ningún otro cambio fue realizado a la base de datos, éstos deberán estar de la misma manera que cuando se comenzó este ejercicio.

Autoexamen Capítulo 15

- 1.** ¿Qué es un cursor?
- 2.** ¿Cuáles métodos de invocación soportan el uso de cursores?
- 3.** ¿Cuál forma de incongruencia en la impedancia es cubierta a través del uso de cursores?
- 4.** Un(a) _____ funciona como un señalador que permite al lenguaje de programación de aplicación tratar con los resultados de la consulta una fila a la vez.

5. Cuando se utilizan cursores en SQL incrustado, ¿cuál es el primer paso que se debe tomar antes de que se puedan recuperar datos a través de ese cursor?
 - A Buscar el cursor.
 - B Declarar el cursor.
 - C Cerrar el cursor.
 - D Abrir el cursor.
6. ¿Cuáles son las cuatro instrucciones relacionadas con el cursor que pueden ser incrustadas en un lenguaje host?
7. ¿Cuáles opciones pueden ser utilizadas en las instrucciones de cursor de sólo lectura?
 - A SCROLL
 - B WITH HOLD
 - C ORDER BY
 - D INSENSITIVE
8. ¿Cuáles son los elementos obligatorios de una instrucción DECLARE CURSOR?
9. ¿Qué tipo de cursor no ve los cambios hechos por instrucciones fuera del cursor?
10. ¿Cuál opción deberá utilizarse en una instrucción de cursor para extender las capacidades de recuperación de una instrucción FETCH?
 - A WITHOUT HOLD
 - B ASENSITIVE
 - C SCROLL
 - D FOR UPDATE
11. La capacidad para _____ el cursor hace referencia a una característica en los cursores que está relacionada con la condición de cerrar o no un cursor automáticamente cuando la transacción en la que el cursor fue abierto es completada.
12. Se está creando una instrucción de cursor. La instrucción SELECT incluye una cláusula ORDER BY. ¿Cuáles cláusulas no pueden ser incluidas en la instrucción SELECT?
 - A SELECT
 - B HAVING
 - C GROUP BY
 - D WHERE
13. ¿Cuál opción deberá incluirse en una instrucción de cursor para definir que ese cursor tiene capacidades para mantenerse abierto?
14. La instrucción de cursor incluye una cláusula FOR UPDATE que no especifica ninguna columna. ¿Cuáles columnas en la tabla subyacente pueden ser actualizadas?
15. ¿Cuál instrucción SQL deberá utilizarse si se quiere abrir el cursor ARTISTAS_CD?

- 16.** ¿Cuál instrucción SQL ejecuta la instrucción SELECT en un cursor?
- 17.** Una instrucción _____ recupera filas desde los resultados de la consulta del cursor una vez que se abre ese cursor.
- 18.** ¿Qué tipo de cursor permite utilizar todas las opciones de orientación para búsqueda en la instrucción FETCH?
- 19.** ¿Cuál opción de orientación para búsqueda deberá utilizarse en una instrucción FETCH si se requiere asegurarse de recuperar la primera fila en los resultados de la consulta del cursor?
 - A** PRIOR
 - B** NEXT
 - C** ABSOLUTE -1
 - D** FIRST
- 20.** ¿Cuál cláusula es requerida en una instrucción UPDATE posicionada para poder actualizar una fila arrojada por la instrucción FETCH más reciente?

Capítulo 16

Manejar transacciones
SQL

Habilidades y conceptos clave

- Entender las transacciones SQL
- Configurar las propiedades de la transacción
- Iniciar una transacción
- Determinar el aplazamiento de una restricción
- Crear puntos de recuperación en una transacción
- Finalizar una transacción

En el capítulo 4 se utilizó una cantidad considerable de tiempo para analizar la integridad de los datos y los métodos soportados por SQL para poder asegurar tal integridad. Estos métodos incluyen la creación de restricciones, dominios y afirmaciones, todos ellos utilizados por la base de datos de una manera u otra para asegurar que los datos SQL permanezcan bien fundamentados. Sin embargo, por sí solos, estos métodos no son siempre suficientes para mantener la integridad de esos datos. **Tomemos, por ejemplo, la situación que puede surgir cuando más de un usuario intenta acceder y modificar datos en la misma tabla al mismo tiempo, o cuando sus acciones se superponen e impactan los mismos datos.** Algunas acciones pueden ser tomadas por un usuario basándose en los datos que ya no son válidos como resultado de las acciones tomadas por el otro usuario. Los datos pueden convertirse en inconsistentes o inexactos, sin que ninguno de los dos usuarios se dé cuenta de que el problema existe. Para encargarse de las situaciones de este tipo, **SQL soporta el uso de transacciones para asegurarse que las acciones actuales no impacten la integridad de los datos que están siendo revisados por cualquier usuario.** En este capítulo se describirá cómo se implementan las transacciones en un ambiente SQL y cómo puede controlarse su comportamiento. El usuario aprenderá cómo configurar las propiedades de las transacciones, iniciar transacciones, finalizarlas y utilizar otras opciones que extiendan su funcionalidad.

Entender las transacciones SQL

Relativamente, existen muy pocas bases de datos en las cuales solamente un usuario esté intentando acceder a los datos contenidos en ellas en algún momento dado. La mayoría de las veces, las bases de datos son utilizadas por diferentes tipos de usuarios para muy distintos propósitos, y a menudo estos usuarios están intentando acceder a los datos al mismo tiempo. Mientras mayor sea el número de usuarios, mayor será la probabilidad de que existan problemas cuando los usuarios intenten ver o modificar los mismos datos en el mismo momento. Sin embargo, dependiendo de la naturaleza de las operaciones, los problemas pueden surgir incluso si solamente dos usuarios están accediendo a los datos al mismo tiempo. Por ejemplo, un usuario pudiera estar viendo los datos en una tabla, tomar cierto tipo de acción basada en esos datos, y luego arrojarlos a la tabla para verificar los datos una vez más. Sin embargo, si otro usuario actualiza la tabla entre los dos momentos que el primer usuario la visualiza, el primer usuario encontrará datos diferentes la segunda vez que

lo hace, y pudiera incluso notar que la acción tomada por el segundo usuario invalidó los cambios que ambos hicieron después de haber visto la tabla por primera vez. Por ejemplo, el primer usuario pudiera haber notado que el número telefónico de un cliente es incorrecto y quisiera aplicar la corrección. Sin embargo, un segundo usuario pudiera estar buscando los datos del mismo cliente, y mientras éste actualiza el estatus del crédito del cliente, pudiera sin darse cuenta regresar el anterior número telefónico al registro de la base de datos (debido a que los datos que se estaban buscando contenían el número anterior), sobrescribiendo los cambios que realizó el primer usuario.

Para encargarse de este tipo de inconsistencias en los datos, SQL utiliza transacciones para controlar las acciones de los usuarios individuales. **Una transacción es una unidad de trabajo que se compone de una o más instrucciones SQL que realizan un conjunto de acciones relacionadas.** Por ejemplo, la aplicación podría utilizar una transacción para cambiar el número de CD de las existencias. El proceso de actualizar la tabla o tablas aplicables y reportar la información actualizada de regreso al usuario es tratado como una sola transacción. La transacción puede incluir varias instrucciones SQL, realizando cada una de ellas una tarea específica.

Para que un conjunto de acciones califique como una transacción, debe pasar la prueba ACID. ACID es el acrónimo comúnmente utilizado para referirse a los nombres en inglés de las cuatro características de una transacción (*Atomic, Consistent, Isolated y Durable*, respectivamente) que veremos a continuación:

- **Atómica** Esta característica se refiere a la naturaleza todo-o-nada de una transacción. Se realizan ya sea todas las operaciones en una transacción, o ninguna de ellas. Aunque algunas instrucciones sean ejecutadas, los resultados de éstas regresan a su punto inicial si la transacción falla en cualquier punto antes de ser completada. Solamente cuando todas las instrucciones se ejecutan apropiadamente y todas las acciones se realizan, se considera completa una transacción y sus resultados se aplican a la base de datos.
- **Consistente** La base de datos debe ser consistente al inicio y al final de la transacción. De hecho, se puede considerar una transacción como un conjunto de acciones que lleva a la base de datos de un estado consistente a otro. Todas las reglas que definen y limitan los datos deben ser aplicadas a esos datos como resultado de cualquier cambio que ocurra durante la transacción. Además, todas las estructuras dentro de la base de datos deben estar correctas al final de la transacción.
- **Aislada (Isolated)** Los datos que pudieran encontrarse en un estado inconsistente temporalmente durante una transacción no deberán estar disponibles a otras transacciones hasta que los datos sean consistentes una vez más. En otras palabras, ningún usuario deberá ser capaz de acceder a los datos inconsistentes durante una transacción implementada por otro usuario cuando los datos impactados por esa transacción están en un estado inconsistente. Además, cuando una transacción se encuentra aislada, ninguna otra transacción puede afectarla.
- **Durable** Una vez que los cambios hechos en una transacción sean completados, esos cambios deberán ser preservados, y los datos deberán estar en un estado confiable y consistente, incluso si ocurren errores de aplicación o de hardware.

Si surge cualquier problema en cualquier momento durante una transacción, la transacción completa regresa a su punto inicial y la base de datos regresa al estado en que se encontraba antes de que la transacción iniciara. Cualquier acción que se tome es cancelada y los datos se restauran a su estado original. Si la transacción se completa exitosamente, entonces todos los cambios son im-

plementados. A través de todo el proceso, la transacción siempre asegura la integridad de la base de datos, sin importar si la transacción fue completada exitosamente o debió regresar a su punto inicial.

SQL soporta diferentes instrucciones relacionadas con el proceso de transacción. Estas transacciones pueden ser utilizadas para iniciar y finalizar transacciones, configurar sus propiedades, aplazar la ejecución de las restricciones durante la transacción e identificar los puntos dentro de una transacción que actúan como puntos de recuperación cuando las transacciones vuelven a su punto inicial. En el resto de este capítulo examinaremos cómo se utiliza cada una de estas instrucciones dentro de una transacción. Sin embargo, antes de ir a una discusión más detallada de las instrucciones, sería mejor proporcionar un breve repaso de cada una de ellas para tener un mejor entendimiento de cómo funcionan las transacciones.

El estándar SQL:2006 define siete instrucciones relacionadas al proceso de transacción:

- **SET TRANSACTION** Configura las propiedades de la siguiente transacción que deberá ser ejecutada.
- **START TRANSACTION** Configura las propiedades de una transacción e inicia esa transacción.
- **SET CONSTRAINTS** Determina el modo de restricción dentro de una transacción actual. El modo de restricción se refiere a si una restricción es aplicada inmediatamente a los datos cuando éstos son modificados o si la aplicación de la restricción es aplazada hasta un punto posterior en la transacción.
- **SAVEPOINT** Crea un punto de recuperación dentro de una transacción. Un *punto de recuperación* marca una zona dentro de la transacción que actúa como un punto para detenerse cuando una transacción tiene que regresar a su punto inicial.
- **RELEASE SAVEPOINT** Libera un punto de recuperación.
- **ROLLBACK** Finaliza una transacción y reinvierte todos los cambios al comienzo de la transacción o a un punto de recuperación.
- **COMMIT** Finaliza una transacción y permite completar todos los cambios a la base de datos.

A pesar de que todas estas siete instrucciones serán vistas con más detalle, algunas de ellas son esenciales para entender la naturaleza de una transacción. Veamos la figura 16-1 para ayudar a ilustrar este punto.

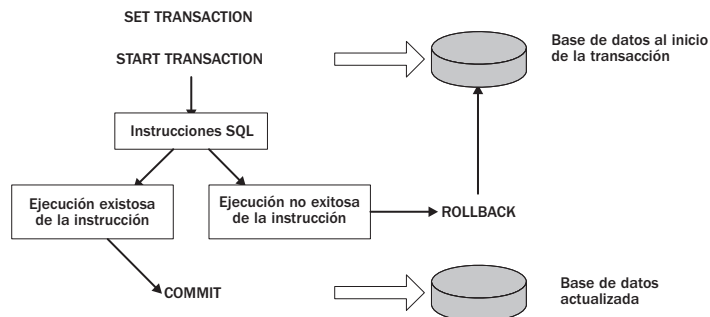


Figura 16-1 Una transacción SQL básica.

Observe que la figura incluye cuatro de las instrucciones SQL relacionadas con las transacciones: SET TRANSACTION, START TRANSACTION, COMMIT y ROLLBACK. Si se utiliza una instrucción SET TRANSACTION, ésta se ejecuta antes de que la transacción inicie. Después de eso, una instrucción START TRANSACTION inicia la transacción.

NOTA

Como se verá posteriormente en este capítulo, sería raro en un ambiente SQL puro que se quisieran utilizar ambas instrucciones: SET TRANSACTION y START TRANSACTION, debido a que ambas instrucciones establecen las mismas propiedades. Sin embargo, se encontrará que las implementaciones SQL varían respecto a cuáles instrucciones relacionadas con las transacciones soportan y cómo implementan esas instrucciones. Se señalarán diferencias específicas a lo largo de este capítulo.

Cuando se inicia la transacción, la base de datos se encuentra en su estado original (los datos son consistentes y correctos). Después se procesan las instrucciones SQL dentro de la transacción. Si este proceso es exitoso, se ejecuta una instrucción COMMIT. La instrucción COMMIT provoca que la implementación SQL actualice la base de datos y finalice la transacción. Si el proceso de ejecución de la instrucción no es exitoso, se ejecuta una instrucción ROLLBACK y la implementación regresa la base de datos a su estado original. Una ejecución no exitosa no significa necesariamente que las instrucciones hayan fallado. Una instrucción ROLLBACK puede ser ejecutada de acuerdo con las condiciones de una cláusula WHERE, un error predefinido, o por cualquier otra condición que sea definida dentro de la transacción. El punto es que, bajo ciertas circunstancias, se ejecuta la instrucción ROLLBACK, y bajo otras circunstancias se ejecuta la instrucción COMMIT.

Configurar las propiedades de la transacción

La primera instrucción que veremos a detalle es la instrucción SET TRANSACTION. La instrucción SET TRANSACTION permite configurar la mayoría de las propiedades asociadas con el proceso de transacción. Se puede ejecutar esta instrucción solamente cuando ninguna transacción está activa (o en el caso de Oracle, sólo como la primera instrucción después de COMMIT o ROLLBACK). Cuando sí se utiliza una instrucción SET TRANSACTION, las propiedades configuradas dentro de la instrucción son aplicadas solamente a la siguiente transacción que es iniciada. Las propiedades no se trasladan de una transacción a la otra.

La instrucción SET TRANSACTION no es obligatoria para iniciar una transacción. Si la instrucción no es ejecutada, la transacción utiliza ya sea las propiedades de manera predeterminada, o las propiedades suministradas en la instrucción subsecuente START TRANSACTION. Si se ejecuta la instrucción SET TRANSACTION, la transacción utiliza las propiedades especificadas en esa instrucción. Si la instrucción se ejecuta pero no se definen todas las propiedades, la transacción utiliza los valores de forma preestablecida para las propiedades no definidas. Sin importar cuáles propiedades sean configuradas, ninguna de ellas es aplicable a ninguna transacción, excepto a la primera iniciada después de que SET TRANSACTION es ejecutada.

Ahora demos un vistazo a la sintaxis utilizada para una instrucción SET TRANSACTION. En su forma más básica, la sintaxis luce de la siguiente manera:

```
SET [ LOCAL ] TRANSACTION <modo> [ { , <modo> } ... ]
```


El primer aspecto que deberá notar acerca de esta sintaxis es la palabra clave opcional `LOCAL`. La palabra clave `LOCAL` aplica solamente a las transacciones que comprenden múltiples implementaciones SQL. Si se está trabajando con este tipo de transacciones, se puede utilizar la opción `LOCAL` para aplicar las propiedades a la porción local de la transacción. Para poder utilizar la opción `LOCAL`, la transacción debe haber sido iniciada en un servidor SQL diferente a aquél donde las propiedades de transacción locales sean configuradas.

NOTA

El tema de abarcar transacciones y propiedades locales excede el campo de acción de este libro. Se mencionan aquí solamente para proporcionar una imagen completa de la instrucción `SET TRANSACTION`. Como un programador SQL principiante, es muy probable que no tenga que preocuparse por abarcar las transacciones. Además, el soporte para abarcar las transacciones varía a través de las diferentes implementaciones SQL. Por ejemplo, Oracle no las soporta, y mientras que MySQL sí, sólo utiliza los términos global y sesión.

Regresando a la sintaxis de `SET TRANSACTION`, se puede ver que el único tipo diferente de opción que se necesita especificar es aquel representado por el marcador de posición `<modo>`. Existen tres tipos de modos de transacción que se pueden especificar:

- Nivel de acceso
- Nivel de aislamiento
- Tamaño de diagnóstico

Se deben especificar uno o más modos de transacción. Si se especifica más de uno, deben quedar separados por comas. Además, no se puede incluir más de uno del mismo tipo de los modos de transacción. Por ejemplo, es posible especificar un nivel de acceso y un nivel de aislamiento, pero no es posible especificar dos niveles de aislamiento.

La instrucción `SET TRANSACTION` soporta dos opciones del nivel de acceso: `READ ONLY` y `READ WRITE`. Si se selecciona la opción `READ ONLY`, no se puede incluir ninguna instrucción dentro de la transacción que modifique la base de datos. Esto incluye a las instrucciones que modifican datos (por ejemplo, la instrucción `UPDATE`) y a las instrucciones que modifican la estructura de la base de datos (por ejemplo, la instrucción `CREATE TABLE`). Si se selecciona la opción `READ WRITE`, se pueden ejecutar ambos tipos de instrucciones en la transacción. Como se verá en la siguiente sección, “Especificar un nivel de aislamiento”, el nivel de acceso depende predeterminadamente del nivel de aislamiento. Sin embargo, si no se especifica ningún nivel de aislamiento y ningún nivel de acceso, el nivel de acceso por defecto es `READ WRITE`.

Especificar un nivel de aislamiento

Cuando se crea una instrucción `SET TRANSACTION`, se pueden especificar los niveles de aislamiento cero o uno. Un nivel de aislamiento define cómo sería aislar una transacción desde las acciones de otras transacciones. Una instrucción `SET TRANSACTION` soporta cuatro opciones del nivel de aislamiento:

- `READ UNCOMMITTED`
- `READ COMMITTED`

- REPEATABLE READ
- SERIALIZABLE

Los niveles de aislamiento están enlistados del menos restrictivo al más restrictivo, siendo la opción READ UNCOMMITTED la menos efectiva en términos de aislar los datos, y la opción SERIALIZABLE la más efectiva. Si no se especifica ningún nivel de aislamiento, se asume SERIALIZABLE de manera preestablecida.

NOTA

El soporte para los niveles de aislamiento varía entre las diferentes implementaciones de SQL. Por ejemplo, Oracle soporta solamente las opciones READ COMMITTED y SERIALIZABLE, mientras que SQL Server soporta las cuatro opciones definidas en el estándar SQL, además de añadir una nueva llamada SNAPSHOT que proporciona consistencia de lectura para todas las instrucciones en una transacción al igual que al inicio de la transacción.

Irregularidades de los datos

La mejor manera de comprender los niveles de aislamiento es dar un vistazo a los tipos básicos de irregularidades que pueden ocurrir a los datos, dependiendo de qué tan aislada está una transacción de la otra. En general, pueden ocurrir tres tipos de irregularidades:

- Lecturas sucias
- Lecturas no repetibles
- Lecturas fantasma

El tipo de irregularidad de datos que puede experimentarse durante una transacción depende de cuál nivel de aislamiento se configure para la transacción. Sin embargo, antes de internarnos en mayores especificaciones acerca de los niveles de aislamiento, demos un vistazo a estos tres tipos de irregularidades.

Lecturas sucias La primera irregularidad que veremos es la lectura sucia. Una *lectura sucia* puede ocurrir cuando una transacción modifica los datos, una segunda transacción ve esas modificaciones antes de que sean realmente completadas en la base de datos, y después la primera transacción retira las modificaciones, regresando a la base de datos a su estado original. Sin embargo, la segunda transacción, habiendo leído los datos modificados, pudo haber tomado alguna acción basada en los datos incorrectos. Para ayudar a ilustrar el concepto de una lectura sucia, veamos la figura 16-2, que muestra dos transacciones operando al mismo tiempo.

Cuando comienza Transacción 1, ésta lee la tabla en su estado original. Luego, la transacción actualiza la tabla, cambiando el valor EXISTENCIA de cada fila. Después de que se realizan los cambios, Transacción 2 es iniciada y lee los datos actualizados. Por ejemplo, Transacción 2 verá que el valor EXISTENCIA para la fila Past Light es 11. Basada en esa información, Transacción 2 toma algún tipo de acción, por ejemplo, ordenar los CD adicionales de Past Light. Después de que Transacción 2 ha leído los datos de la tabla, Transacción 1, por una u otra razón, reinvierte la actualización, y la base de datos regresa a su estado original. Como resultado, la fila Past Light ahora tiene en realidad un valor EXISTENCIA de 22, a pesar de que Transacción 2 piensa que ésta tiene un valor de 11. Por lo tanto, se dice que Transacción 2 ha experimentado una lectura sucia.

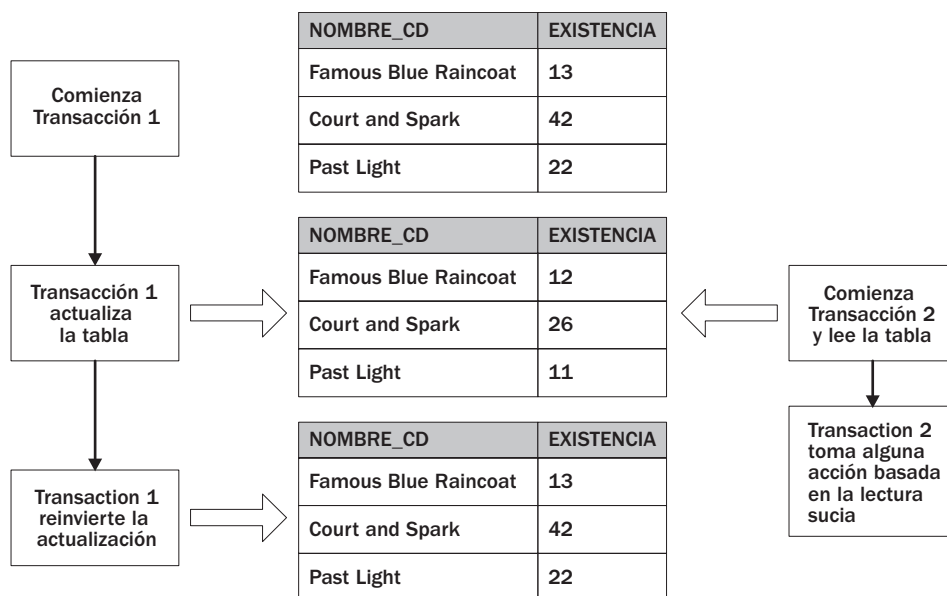


Figura 16-2 Transacciones simultáneas arrojando el resultado de una lectura sucia.

Lecturas no repetibles La siguiente irregularidad que puede ocurrir cuando se inician transacciones simultáneas es la lectura no repetible. La *lectura no repetible* puede ocurrir cuando una transacción lee datos desde una tabla, luego otra transacción actualiza la tabla, y la primera transacción vuelve a leer los datos, sólo para descubrir que los datos han sido cambiados. Como resultado, la primera lectura es no repetible. Veamos la figura 16-3 para comprender mejor este concepto.

Cuando se inicia Transacción 1, ésta lee los datos en la tabla. En ese punto, la transacción podría estar involucrada en otros procesos o estar esperando la respuesta de un usuario. Por ejemplo, el usuario pudiera recibir una llamada de un administrador que está tratando de averiguar cuántas copias hay en existencia de un CD en particular. El usuario verifica esa información. Entonces el administrador pone al usuario en espera durante un corto tiempo, por lo que el usuario debe esperar para completar la transacción. Durante ese tiempo, se inicia Transacción 2 y actualiza la tabla. Después de la actualización, Transacción 1 vuelve a leer los datos (el administrador regresa al teléfono) y encuentra información diferente a la de la primera lectura, dando como resultado una lectura no repetible.

Lecturas fantasma La última irregularidad que veremos es la lectura fantasma. A pesar de ser parecida a la lectura no repetible, la lectura fantasma tiene algunas diferencias sutiles, que se convierten en factores cuando se intenta determinar un nivel de aislamiento. Una *lectura fantasma* puede ocurrir cuando una transacción lee una tabla basada en algún tipo de condición de búsqueda, después una segunda transacción actualiza los datos en la tabla, y la primera transacción intenta volver a leer los datos, sólo que esta vez se arrojan diferentes filas debido a como está definida la condición de búsqueda. Para dejar esto más claro, demos un vistazo a la figura 16-4.

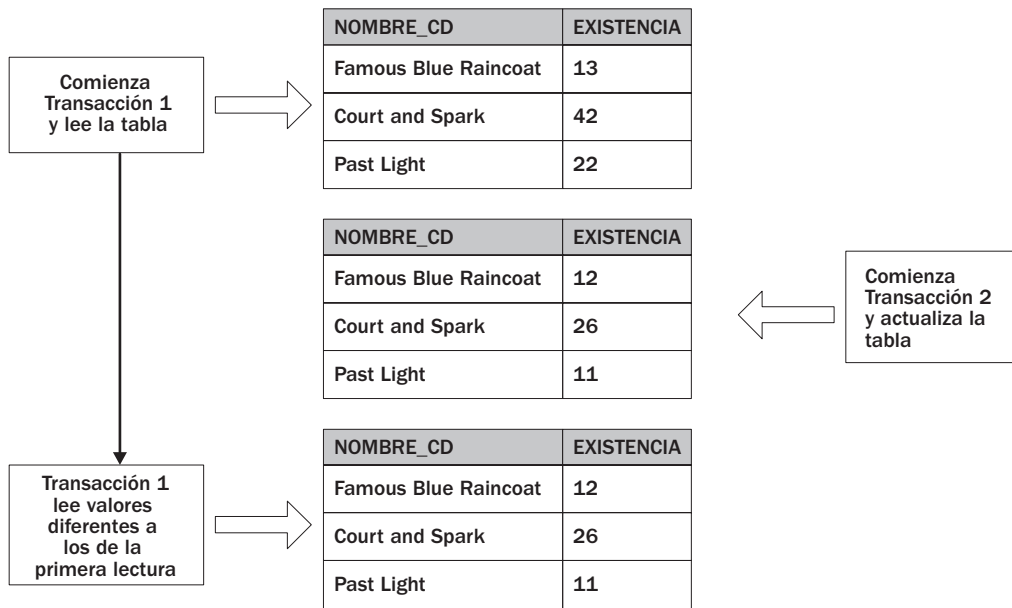


Figura 16-3 Transacciones simultáneas resultando en una lectura no repetible.

Cuando se inicia Transacción 1, ésta lee los datos en la tabla ejecutando una instrucción `SELECT` que consulta los datos. La instrucción incluye una cláusula `WHERE` que arroja solamente aquellas filas con un valor `EXISTENCIA` mayor a 20. Eso significa que la fila `Court and Spark` y la fila `Past Light` son arrojadas. Después de que Transacción 1 recupera (lee) los datos, se inicia Transacción 2 y actualiza la tabla. Ahora, cuando Transacción 1 vuelve a leer los datos (utilizando los mismos criterios de búsqueda), solamente la fila `Famous Blue Raincoat` es arrojada debido a que ahora es la única fila con un valor `EXISTENCIA` mayor a 20. Como resultado, la transacción ha experimentado una lectura fantasma, y las filas que está leyendo no son las mismas filas que ésta había visto anteriormente.

Escoger un nivel de aislamiento

Ahora que se tiene una vista más general acerca de los tipos de irregularidades de datos con los que es posible toparse cuando se tienen transacciones simultáneas, se deberá estar mejor preparado para escoger un nivel de aislamiento para la transacción. El punto importante a recordar es que mientras más restrictivo sea el nivel de aislamiento, más tipos de irregularidades pueden ser eliminadas.

Demos un vistazo al nivel de aislamiento `READ UNCOMMITTED`, el menos restrictivo de los cuatro niveles. Una transacción configurada con esta opción puede experimentar cualquiera de las irregularidades de datos que se vieron anteriormente (lectura sucia, lectura no repetible y lectura fantasma). Como es fácil imaginarse, normalmente éste no es un estado deseable. De hecho, si se define una transacción con la opción `READ UNCOMMITTED`, la transacción no puede incluir instrucciones que modifiquen datos. Estas transacciones, de manera predeterminada, solamente

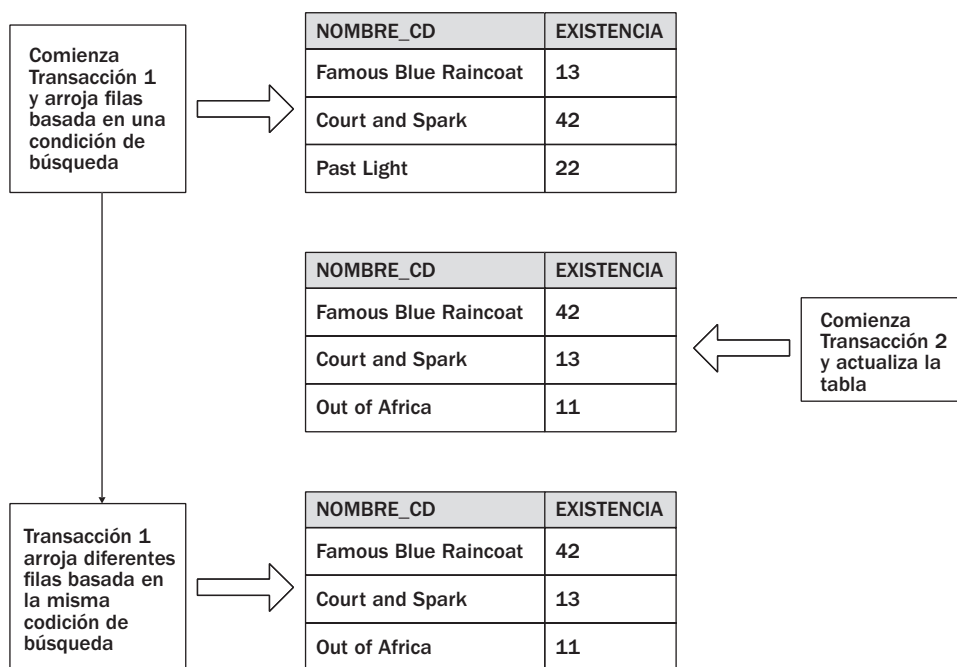


Figura 16-4 Transacciones simultáneas resultando en una lectura fantasma.

tienen un nivel de acceso de sólo lectura (READ ONLY), y no es posible especificar un nivel READ WRITE. (Todos los demás niveles de aislamiento tienen un nivel de acceso preestablecido de READ WRITE.) El nivel de aislamiento READ UNCOMMITTED deberá utilizarse solamente para transacciones que generen información aproximada, por ejemplo, algunos tipos de datos estadísticos en los que los resultados no son muy críticos en términos de precisión.

El nivel de aislamiento READ COMMITTED es sólo ligeramente más restrictivo que READ UNCOMMITTED. La opción READ COMMITTED evita las lecturas sucias, pero las lecturas no repetibles y las lecturas fantasma aún pueden ocurrir. La siguiente opción, REPEATABLE READ, es más restrictiva que READ COMMITTED. Evita las lecturas sucias y las lecturas no repetibles, pero no evita las lecturas fantasma. La única opción que evita los tres tipos de irregularidades en los datos es SERIALIZABLE.

Una transacción que está definida con el nivel de aislamiento SERIALIZABLE puede aislar completamente a esa transacción de todas las otras transacciones. Como resultado, se dice que la transacción se puede serializar, lo que significa que interactúa con transacciones simultáneas en una forma en que ordena las transacciones secuencialmente para que una transacción no pueda impactar a la otra. Esto no significa que una transacción deba cerrarse antes de que otra pueda abrirse, pero sí significa que los resultados de esas transacciones tienen que ser iguales a los resultados de las operaciones que se realizan una a la vez. Mientras ninguna transacción que se puede serializar puede influir sobre otra transacción que se puede serializar, las transacciones estarán en conformidad con el nivel de aislamiento SERIALIZABLE.

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

Tabla 16-1 Posibles irregularidades de datos para los niveles de aislamiento.

La tabla 16-1 proporciona una idea general de la irregularidad que puede ocurrir para cada nivel de aislamiento. Por ejemplo, observe que, para la opción `READ UNCOMMITTED`, es posible que las tres irregularidades de datos puedan ocurrir.

Como se puede observar en la tabla, el nivel de aislamiento `SERIALIZABLE` proporciona la más alta protección de datos, y el nivel de aislamiento `READ UNCOMMITTED` proporciona la más baja. Es por eso que `SERIALIZABLE` es el nivel de aislamiento predeterminado si no se define ningún otro nivel.

NOTA

Seguramente debe preguntarse por qué no utilizar simplemente el nivel de aislamiento `SERIALIZABLE` para todas las transacciones. Existen pros y contras al respecto: mientras más restrictivo es el nivel de aislamiento, es mayor el impacto al rendimiento, por lo que si se quiere estar seguro de utilizar un nivel de aislamiento lo suficientemente restrictivo para cubrir las necesidades, no se requiere definir un nivel que sea más restrictivo de lo necesario.

Especificar un tamaño de diagnóstico

Como podrá recordar de la sintaxis `SET TRANSACTION`, uno de los tipos de modo de transacción que puede definirse es el tamaño de diagnóstico. El tamaño de diagnóstico se refiere a un área de diagnóstico que es utilizada para condiciones que surgen cuando se ejecuta una instrucción SQL. Una *condición* es una advertencia, una excepción u otro tipo de mensaje generado por la ejecución de una instrucción. El tamaño de diagnóstico realmente se refiere al número de condiciones que serán almacenadas para la ejecución de una instrucción SQL. Por ejemplo, si el tamaño de diagnóstico es de 10, hasta 10 condiciones serán almacenadas para la instrucción ejecutada. Si se generan más de 10 condiciones para esa instrucción, solamente 10 condiciones serán guardadas en el área de diagnóstico.

NOTA

No se puede asumir que será guardado algún tipo de condición especificado en el área de diagnóstico si se generan más condiciones que el número definido por el tamaño de diagnóstico. Por ejemplo, si el tamaño de diagnóstico es de 15 y la instrucción genera 20 condiciones, no puede asumirse que serán guardadas las primeras 15 o las últimas 15 condiciones. Para mayor información acerca de cómo se manejan las condiciones en una implementación SQL en particular, véase la documentación del producto. Dicho sea de paso, el uso de `SET TRANSACTION` para determinar el tamaño de diagnóstico no es soportado por SQL Server, Oracle, MySQL, ni por la mayoría de las implementaciones SQL actuales.

Si no se especifica un tamaño de diagnóstico en la instrucción SET TRANSACTION, la implementación SQL determina el tamaño del área de diagnóstico.

Crear una instrucción SET TRANSACTION

Ahora que hemos visto los diferentes componentes de la instrucción SET TRANSACTION, demos un vistazo a un par de ejemplos. El primer ejemplo define una transacción con un nivel de acceso READ ONLY, un nivel de aislamiento READ UNCOMMITTED y un tamaño de diagnóstico de 5:

```
SET TRANSACTION
  READ ONLY,
  ISOLATION LEVEL READ UNCOMMITTED,
  DIAGNOSTICS SIZE 5;
```

Observe que los modos de transacción están separados por comas. Observe también que la opción del nivel de aislamiento incluye las palabras clave ISOLATION LEVEL, y que la opción del tamaño de diagnóstico incluye las palabras clave DIAGNOSTICS SIZE. La transacción está configurada con el nivel de aislamiento restrictivo más bajo, y ésta es la razón de que el nivel de acceso debe ser READ ONLY. No es posible definir un nivel de acceso READ WRITE para esta instrucción. Debido a que el nivel de aislamiento es READ COMMITTED, la instrucción no tiene que especificar el nivel de acceso READ ONLY (debido a que ya es asumido). Sin embargo, el incluirlo no causa ningún problema y documenta el código de mejor manera.

En el siguiente ejemplo, la instrucción SET TRANSACTION define una transacción con un nivel de acceso READ WRITE, un nivel de aislamiento SERIALIZABLE y un tamaño de diagnóstico de 8:

```
SET TRANSACTION
  READ WRITE,
  ISOLATION LEVEL SERIALIZABLE,
  DIAGNOSTICS SIZE 8;
```

Debido a que SERIALIZABLE es el nivel de aislamiento preestablecido, no es necesario especificarlo en la instrucción SET TRANSACTION. Además, debido a que el nivel de acceso READ WRITE es el nivel predeterminado para las transacciones que se pueden serializar, tampoco es necesario especificarlo. La instrucción, por lo tanto, puede tener un aspecto como el siguiente:

```
SET TRANSACTION
  DIAGNOSTICS SIZE 8;
```

Esta instrucción SET TRANSACTION producirá los mismos resultados que la anterior.

Como se puede ver, la instrucción SET TRANSACTION es una instrucción relativamente simple de ejecutar. Sin embargo, es necesario asegurarse de revisar la documentación de la implementación SQL para determinar la sintaxis exacta utilizada para determinar las propiedades de la transacción. Por ejemplo, SQL Server soporta una instrucción SET TRANSACTION ISOLATION LEVEL que permite establecer solamente el nivel de aislamiento. No es posible establecer el nivel de acceso o el tamaño de diagnóstico. Oracle, por otro lado, soporta una instrucción SET TRANSACTION que permite establecer el nivel de acceso de la transacción y el nivel de aislamiento (pero no el tamaño de diagnóstico) y asigna la transacción a un segmento para poder reinvertirla, mientras que MySQL solamente permite establecer el nivel de aislamiento.

Iniciar una transacción

En SQL:2006, una transacción puede ser iniciada ya sea implícita o explícitamente. Una transacción inicia implícitamente cuando ciertos tipos de instrucciones SQL son ejecutadas, por ejemplo, las instrucciones SELECT, DELETE, UPDATE y CREATE TABLE. Estos tipos de instrucciones deben ejecutarse dentro del contexto de una transacción. Si ninguna transacción está activa, una de ellas es iniciada.

Las transacciones también pueden ser iniciadas explícitamente utilizando la instrucción START TRANSACTION. Esta instrucción sirve para dos propósitos: establecer las propiedades de la transacción e iniciar la transacción. En términos de establecer las propiedades, la instrucción START TRANSACTION funciona de la misma manera que la instrucción SET TRANSACTION. Se puede establecer el nivel de acceso, el nivel de aislamiento y el tamaño de diagnóstico. Al igual que para iniciar una transacción, simplemente se ejecuta la instrucción START TRANSACTION.

La sintaxis para la instrucción START TRANSACTION es similar a la de la instrucción SET TRANSACTION, como se puede ver en la siguiente sintaxis:

```
START TRANSACTION <modo> [ { , <modo> } ... ]
```

Después de especificar las palabras clave START TRANSACTION, se deben especificar uno o más modos de transacción. Al igual que con la instrucción SET TRANSACTION, solamente es posible incluir un modo para cada tipo.

Ahora demos un vistazo a un ejemplo que define un nivel de acceso READ ONLY, un nivel de aislamiento READ UNCOMMITTED y un tamaño de diagnóstico de 5:

```
START TRANSACTION
    READ ONLY,
    ISOLATION LEVEL READ UNCOMMITTED,
    DIAGNOSTICS SIZE 5;
```

Como se puede ver, esto luce prácticamente idéntico a la instrucción SET TRANSACTION. Los modos de transacción se aplican de la misma manera, y si se especifica más de un modo de transacción, éstos se separan por comas. La diferencia básica entre una instrucción START TRANSACTION y una instrucción SET TRANSACTION es que la instrucción START TRANSACTION iniciará la transacción al igual que determinará sus propiedades.

NOTA

La instrucción START TRANSACTION fue agregada a SQL con la liberación de SQL:1999. Como resultado, no todas las implementaciones SQL soportan una instrucción START TRANSACTION o cualquier instrucción que inicie explícitamente una transacción. Las transacciones en Oracle, por ejemplo, pueden ser iniciadas solamente implícitamente. Por otro lado, SQL Server soporta una instrucción BEGIN TRANSACTION, pero no permite definir ningún modo de transacción.

Pregunta al experto

- P:** ¿Es importante si las transacciones incluyen instrucciones del lenguaje de definición de datos (DDL, por sus siglas en inglés) o instrucciones del lenguaje de manipulación de datos (DML)?
- R:** SQL permite incluir ambos tipos de instrucciones en la transacción, pero éste no es el caso para todas las implementaciones SQL. Algunas implementaciones no permiten mezclar los dos tipos de instrucciones en una sola transacción. Otros productos permiten mezclar los dos tipos de instrucciones, pero limitan cuáles instrucciones pueden ser combinadas en una transacción. Incluso otras implementaciones no permiten que las instrucciones del lenguaje de definición de datos sean ejecutadas dentro del contexto de una transacción. Por ejemplo, Oracle finaliza implícitamente cualquier transacción actual cuando se encuentra con una instrucción DDL, y nunca maneja DDL como parte de una transacción. Las restricciones que las diferentes implementaciones colocan al mezclar los tipos de instrucciones pueden variar ampliamente. La razón para esto es que las interacciones entre los dos tipos de instrucciones pueden ser complicadas, por lo que cada implementación determina cuáles mezclas de instrucciones soportará en su propio ambiente de bases de datos. Asegúrese de revisar la documentación del producto para determinar cuáles tipos de instrucciones pueden ser incluidos en una transacción y cómo pueden ser mezclados.

Determinar el aplazamiento de una restricción

Pueden existir ocasiones en una transacción en que se quieran modificar los datos en una tabla que temporalmente viole una restricción colocada en esa tabla. Por ejemplo, se puede tener una tabla que incluya una columna configurada con la restricción NOT NULL. Es posible que durante el curso de la transacción se necesite insertar una fila en la tabla, pero no se tenga aún un valor para la columna NOT NULL. Por esta razón, el estándar SQL permite definir una restricción como aplazable. Esto significa que la restricción no tiene que ser aplicada a los datos inmediatamente (cuando se ejecuta la instrucción SQL que hace modificaciones); ésta puede ser aplazada hasta un punto posterior en una transacción, por ejemplo, hasta que se pueda insertar un valor en la columna NOT NULL.

Si una restricción se define como aplazable, se puede utilizar la instrucción SET CONSTRAINTS dentro de la transacción para aplazar la aplicación de la restricción o para aplicar la restricción inmediatamente. (Definir una restricción como aplazable no pospone automáticamente la aplicación de esa restricción. Se debe aplazarla explícitamente dentro de la transacción.) Si se aplaza explícitamente una restricción, entonces se puede violar de manera temporal la restricción aplazada hasta que la restricción sea aplicada explícitamente o la transacción finalice. Para una mejor comprensión de cómo funciona esto, demos un vistazo a la figura 16-5.

En esta ilustración se notará que, después de que la transacción ha sido iniciada, se pueden definir las restricciones como aplazadas. No se tienen que aplazar las restricciones inmediatamente después de que inicia la transacción, pero deben aplazarse antes de ejecutar cualquier otra instrucción SQL que pudiera violar las restricciones. Una vez que las instrucciones SQL aplicables han

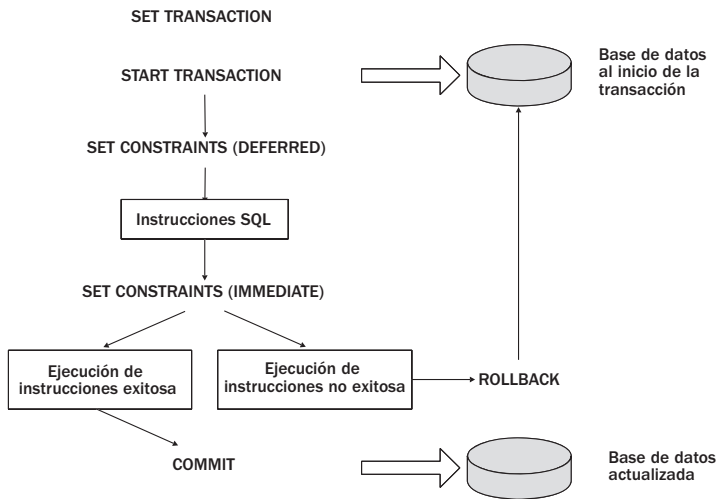


Figura 16-5 Aplazar restricciones en una transacción.

sidó ejecutadas y se está seguro de que ningún dato SQL viola ninguna de las restricciones aplazadas, es posible aplicar las restricciones a los datos aplicables. Si las restricciones se violan en este punto, la transacción se considera no exitosa y todas las actualizaciones son reinvertidas. De otra manera, las actualizaciones son completadas para la base de datos.

Para aplazar o aplicar las restricciones dentro de una transacción, debe utilizarse la instrucción `SET CONSTRAINTS`, como se muestra en la siguiente sintaxis:

```
SET CONSTRAINTS { ALL | <nombrés de las restricciones> }
{ DEFERRED | IMMEDIATE }
```

Como puede verse en la sintaxis, se debe escoger entre dos conjuntos de opciones. El primer conjunto de opciones permite especificar las restricciones aplazables que serán afectadas por la instrucción. Si la instrucción debiera aplicarse a todas las restricciones aplazables, puede usarse la palabra clave `ALL`; de otra manera, se deberán listar los nombres de las restricciones, separados por una columna. En la instrucción `SET CONSTRAINTS` solamente se pueden especificar restricciones aplazables.

El siguiente conjunto de opciones que se debe especificar es si se debe aplazar la aplicación de las restricciones identificadas (`DEFERRED`) o aplicarlas inmediatamente (`IMMEDIATE`). Se deberán aplazar las restricciones antes de insertar o modificar datos, y se deberán aplicar las restricciones después de que sean modificados los datos.

Normalmente, se utilizará la instrucción `SET CONSTRAINTS` en conjuntos por pares: una instrucción para aplazar las restricciones y otra para aplicarlas. Sin embargo, realmente no se necesita utilizar la instrucción `SET CONSTRAINTS` para aplicarlas debido a que todas las restricciones se aplican antes de que se complete la transacción, hayan sido las restricciones aplicadas explícitamente o no. Sin embargo, generalmente es una buena práctica documentar todas las acciones para estar seguro de aplicar explícitamente las restricciones.

Demos ahora un vistazo a un ejemplo de la instrucción SET CONSTRAINTS que aplaza RESTRICCION_1 y RESTRICCION_2:

```
SET CONSTRAINTS RESTRICCION_1, RESTRICCION_2 DEFERRED;
```

Como se puede ver, todo lo que se necesita hacer es listar los nombres de las restricciones y la palabra clave DEFERRED. Si se requiere que la instrucción aplique a todas las restricciones aplazables, puede utilizarse la palabra clave ALL en lugar de los nombres de las restricciones.

Una vez que se ejecuten todas las instrucciones que se necesita ejecutar, con respecto a las restricciones aplazables, se pueden aplicar las restricciones a los datos nuevos y modificados. Para aplicar las restricciones, se utiliza la siguiente instrucción SET CONSTRAINTS:

```
SET CONSTRAINTS RESTRICCION_1, RESTRICCION_2 IMMEDIATE;
```

La única diferencia entre esta instrucción y la que se presentó anteriormente es que se utiliza la palabra clave IMMEDIATE en lugar de la palabra clave DEFERRED.

NOTA

Para poder utilizar la instrucción SET CONSTRAINTS, la implementación de SQL debe soportar tanto esta instrucción (o una instrucción similar) y las restricciones aplazables. Si no es posible definir restricciones aplazables en una base de datos SQL determinada, la instrucción no resulta muy útil. Actualmente, Oracle soporta la instrucción SET CONSTRAINTS, pero no así MySQL ni SQL Server.

Crear puntos de recuperación en una transacción

Una vez que se han instalado todas las transacciones, se encontrará que el conjunto de acciones que se necesita realizar resultan muy directas y sencillas cuando se tratan como una unidad. Sin embargo, pueden existir ocasiones en que algunas de las transacciones no sean tan simples como otras y que diferentes grados de complejidad entre las acciones hagan que tratarlas como una unidad sea un poco más difícil, a pesar de que aún se requiera mantenerlas a todas dentro de la misma transacción. Una forma de tratar con este tipo de situación es a través del uso de *puntos de recuperación*, que son marcadores designados dentro de la transacción que actúan como puntos para reinvertir porciones de la transacción.

NOTA

El soporte para los puntos de recuperación varía entre las diferentes implementaciones SQL. Oracle y MySQL soportan la instrucción SAVEPOINT, pero SQL Server utiliza en su lugar una instrucción SAVE TRANSACTION.

Digamos, por ejemplo, que la primera parte de la transacción contiene código relativamente simple que, a pesar de no complicarse en forma particular, represente una carga pesada para el rendimiento del sistema. Ahora supongamos que posteriormente en la transacción se deben realizar acciones más complejas, acciones que tengan una mayor probabilidad de causar una reinversión que el primer conjunto de acciones. A pesar de eso, no se quiere realizar la reinversión para no perder el trabajo realizado por el primer conjunto de acciones debido al impacto al rendimiento. Si se inserta un punto de recuperación entre los dos conjuntos de acciones y luego el segundo conjunto

necesita ser reinvertido, solamente será reinvertido hasta ese punto de recuperación en lugar de hasta el principio de la transacción, evitando la necesidad de realizar el primer conjunto de acciones otra vez. Para ayudar a ilustrar cómo funcionan los puntos de recuperación, demos un vistazo a la figura 16-6.

Como se puede ver en el diagrama, un punto de recuperación puede ser insertado en cualquier lugar que se desee preservar un conjunto de acciones. Cualquier cambio realizado anteriormente al punto de recuperación será guardado. En este caso, se han definido dos puntos de recuperación, cada uno después de un conjunto de instrucciones SQL que han sido exitosamente ejecutadas. Si una reinversión es necesaria en cualquier parte posterior al punto de recuperación que ha sido definido, la base de datos puede reinvertirse a ese punto de recuperación, sin tener que ir hasta el

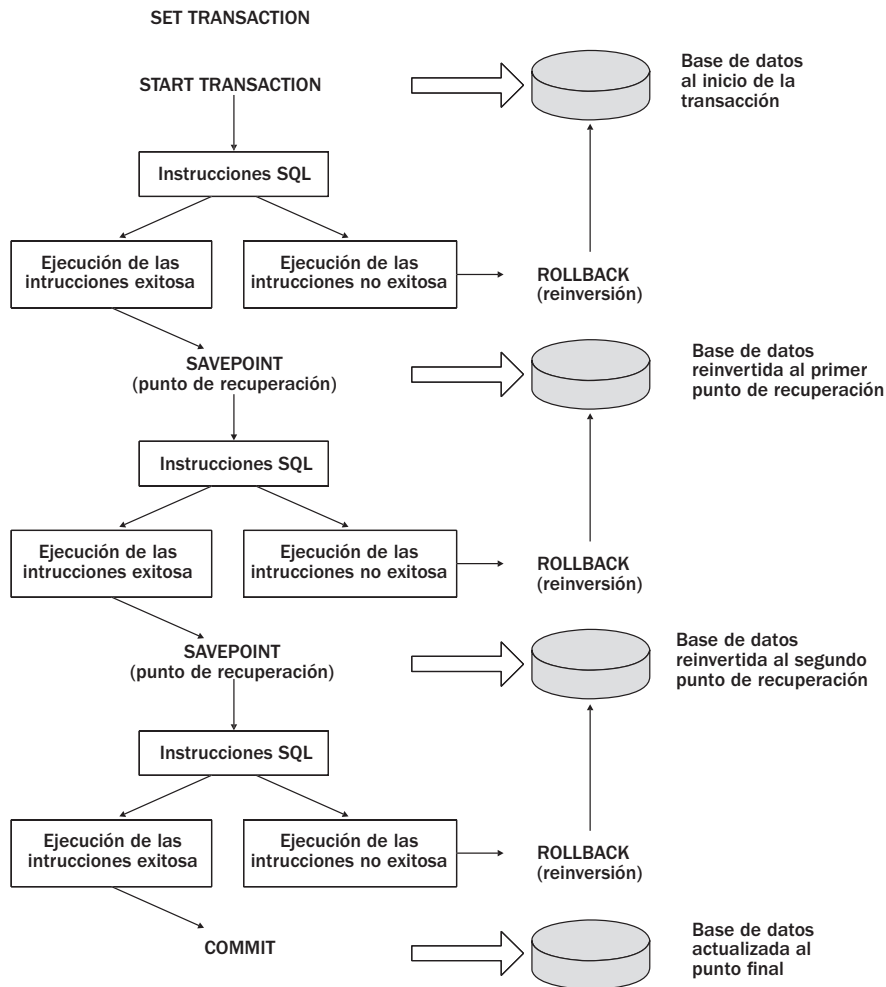


Figura 16-6 Aplazar restricciones en una transacción.

inicio de la transacción, y así las acciones anteriores a ese punto de recuperación no tendrán que ser repetidas. Además, SQL permite nombrar los puntos de recuperación para que, de ser necesario, se pueda reinvertir la transacción a ese punto de recuperación específico, en lugar de aquél directamente anterior a la reinversión. Como resultado, se puede ser más específico acerca de cuáles operaciones conservar y cuáles reinvertir en caso de que surjan problemas en la transacción.

NOTA

Como se verá en la sección “Finalizar una transacción”, más adelante en este capítulo, una transacción es reinvertida a un punto de recuperación solamente si ese punto de recuperación se identifica con la instrucción ROLLBACK. De otra manera, se reinvierte la transacción completa, la transacción es finalizada, y la base de datos es regresada a su estado original anterior a cuando la transacción fue iniciada.

Crear un punto de recuperación en la transacción es muy simple, como se muestra en la siguiente sintaxis:

SAVEPOINT <nombre del punto de recuperación>

Todo lo que se necesita utilizar es la palabra clave **SAVEPOINT**, seguida del nombre del punto de recuperación. Por ejemplo, para crear un punto de recuperación llamado **SECCION_1**, se deberá utilizar la siguiente instrucción:

```
SAVEPOINT SECCION_1;
```

Una vez que se crea el punto de recuperación, se puede utilizar el nombre **SECCION_1** para identificar el punto de recuperación posteriormente en la transacción.

Liberar un punto de recuperación

Después de algunas operaciones dentro de una transacción, se puede encontrar que se necesita liberar un punto de recuperación. Si un punto de recuperación es liberado, ya no se puede reinvertir la transacción a ese punto de recuperación. Liberar un punto de recuperación lo elimina de la transacción. Además, también son liberados todos los puntos de recuperación definidos posteriormente al punto de recuperación liberado. Esto significa que si la transacción incluye tres puntos de recuperación y se libera el primero de ellos, los tres puntos de recuperación son eliminados de la transacción. La sintaxis utilizada para liberar un punto de recuperación es la siguiente:

RELEASE SAVEPOINT <nombre del punto de recuperación>

Como se puede ver, esta instrucción es similar a la instrucción **SAVEPOINT**. Por ejemplo, para liberar el punto de recuperación creado en el ejemplo anterior, se utilizaría la siguiente instrucción:

```
RELEASE SAVEPOINT SECCION_1;
```

Cuando se ejecuta esta instrucción, el punto de recuperación **SECCION_1** es eliminado de la transacción, junto con cualquier otro punto de recuperación definido subsecuente al punto de recuperación **SECCION_1**. Liberar puntos de recuperación es una adición relativamente nueva al estándar SQL, por lo que la mayoría de las implementaciones SQL no lo soportan aún.

Finalizar una transacción

Anteriormente en este capítulo se aprendió que una transacción puede ser iniciada ya sea explícita o implícitamente. Lo mismo ocurre para finalizar una transacción. Es posible completar o reinvertir explícitamente una transacción, lo que finaliza la transacción, o la transacción es finalizada implícitamente cuando las circunstancias fuerzan esa finalización.

En SQL existen cuatro circunstancias primarias que finalizarán una transacción:

- Se define explícitamente una instrucción `ROLLBACK` en la transacción. Cuando la instrucción es ejecutada, las acciones se reinvierten, la base de datos regresa al estado en que estaba cuando la transacción fue iniciada, y la transacción es finalizada. Si la instrucción `ROLLBACK` hace referencia a un punto de recuperación, solamente se reinvierten las acciones realizadas posteriormente a ese punto de recuperación y la transacción no es finalizada.
- Se define explícitamente una instrucción `COMMIT` en la transacción. Cuando la instrucción es ejecutada, todos los cambios relacionados a la transacción se guardan en la base de datos y la transacción es finalizada.
- Se interrumpe el programa que inició la transacción, causando que el programa aborte. En el caso de una interrupción anormal, que puede ser resultado de problemas de hardware o software, todos los cambios se reinvierten, la base de datos regresa a su estado original y la transacción es finalizada. Una transacción finalizada de esta manera es similar a finalizar una transacción utilizando la instrucción `ROLLBACK`.
- El programa completa exitosamente su ejecución. Todos los cambios relacionados con la transacción se guardan en la base de datos y la transacción es finalizada. Una vez que estos cambios son completados, no pueden ser reinvertidos. Una transacción finalizada de esta manera es similar a finalizar una transacción con la instrucción `COMMIT`.

Como se puede ver, las instrucciones `ROLLBACK` y `COMMIT` permiten finalizar explícitamente una transacción, mientras que una transacción es finalizada implícitamente cuando el programa finaliza o es interrumpido. Estos métodos de finalización aseguran que se mantenga la integridad de los datos y que la base de datos esté protegida. No se realiza ningún cambio a la base de datos a menos que la transacción sea completada.

Ahora demos un vistazo más cercano a las dos instrucciones que se pueden utilizar para finalizar explícitamente una transacción.

Completar una transacción

Una vez que todas las instrucciones han sido ejecutadas en una transacción, ésta debe ser finalizada. El tipo de finalización preferido es uno que complete todos los cambios en la base de datos. Después de todo, ¿por qué intentar realizar cambios si no se quiere completarlos? Para completar explícitamente los cambios y finalizar la transacción se debe utilizar la instrucción `COMMIT`, como muestra la siguiente sintaxis:

```
COMMIT [ WORK ] [ AND [ NO ] CHAIN ]
```

En su punto más básico, la instrucción COMMIT requiere solamente la palabra clave COMMIT. Todos los demás elementos de la instrucción son opcionales. Si se desea, se puede incluir la palabra clave WORK, que es simplemente un traslado de versiones anteriores de SQL. En otras palabras, COMMIT y COMMIT WORK realizan la misma función. La única razón para utilizar la palabra clave WORK es que la implementación SQL así lo requiera.

El siguiente elemento opcional en la instrucción COMMIT es la cláusula AND CHAIN, que no es soportada ampliamente en las implementaciones SQL actuales. La cláusula le dice al sistema que inicie una nueva transacción tan pronto finalice la transacción actual. La nueva transacción utiliza los mismos modos de transacción que la transacción actual. Si se utiliza la opción AND CHAIN, ya no se necesita utilizar las instrucciones SET TRANSACTION o START TRANSACTION para la siguiente transacción a menos que se quieran especificar modos diferentes.

Además de especificar AND CHAIN en la instrucción COMMIT, también se puede especificar AND NO CHAIN, que le dice al sistema que no inicie una nueva transacción basada en las configuraciones de la transacción actual. Si se especifica AND NO CHAIN, no será iniciada una nueva transacción automáticamente cuando la transacción actual finalice. Se debe iniciar una nueva transacción utilizando un método implícito o un método explícito. Si no se especifica la cláusula AND CHAIN ni tampoco la cláusula AND NO CHAIN, se toma AND NO CHAIN de manera preestablecida.

Con toda probabilidad, la instrucción COMMIT se verá como la del siguiente ejemplo:

```
COMMIT;
```

Como se puede ver, la palabra clave COMMIT es el único elemento requerido. Sin embargo, si se requiere que una nueva transacción inicie después de la actual, se deberá utilizar la siguiente instrucción COMMIT:

```
COMMIT AND CHAIN;
```

Si no se requiere que una nueva transacción inicie, no se deberá utilizar la cláusula AND CHAIN.

Reinvertir una transacción

A pesar de que el objetivo de cualquier transacción es completar los cambios hechos por las instrucciones en esa transacción, no hay duda que existirán ocasiones cuando se quiera reinvertir esos cambios. Para poder controlar estas reinversiones, se debe utilizar una instrucción ROLLBACK para deshacer los cambios y eliminar la transacción, o para regresar los cambios a un punto de recuperación específico. La siguiente sintaxis muestra los diferentes elementos que pueden ser incluidos en una instrucción ROLLBACK:

```
ROLLBACK [ WORK ] [ AND [ NO ] CHAIN ]  
[ TO SAVEPOINT <nombre del punto de recuperación> ]
```

La primera línea de la sintaxis es muy similar a la instrucción COMMIT. Se debe especificar la palabra clave ROLLBACK. Además, se pueden especificar WORK, AND CHAIN o AND NO CHAIN, las cuales funcionan de la misma manera en que lo hicieron en la instrucción COMMIT, y una vez más, AND NO CHAIN es el valor de manera predeterminada.

Sin embargo, la instrucción `ROLLBACK`, a diferencia de la instrucción `COMMIT`, incluye la cláusula opcional `TO SAVEPOINT`. La cláusula `TO SAVEPOINT` especifica un punto de recuperación que se utiliza si los cambios tienen que ser reinvertidos. Esto aplica a cualquier cambio realizado después del punto de recuperación especificado. Si se incluye la cláusula `TO SAVEPOINT` en la instrucción `ROLLBACK`, la transacción se reinvertirá a ese punto de recuperación, pero no será finalizada. Si no se incluye la cláusula `TO SAVEPOINT`, todos los cambios son reinvertidos y la transacción es finalizada.

El tipo más básico de instrucción `ROLLBACK` es aquel que no incluye elementos opcionales, como en el siguiente ejemplo:

```
ROLLBACK;
```

Se podría haber incluido la palabra clave `WORK` y la cláusula `AND NO CHAIN`, y de todas maneras la instrucción habría realizado la misma función. Si se requiere que sea iniciada una nueva transacción cuando la transacción actual sea finalizada, se deberá especificar la cláusula `AND CHAIN`. Tenga en mente, sin embargo, que no es posible especificar la cláusula `AND CHAIN` y también la cláusula `TO SAVEPOINT` debido a que `AND CHAIN` depende de que la transacción sea finalizada para poder iniciar una nueva transacción.

Si no se especifica la cláusula `TO SAVEPOINT`, se debe incluir el nombre del punto de recuperación. Por ejemplo, la siguiente instrucción `ROLLBACK` especifica el punto de recuperación `SECCION_1`:

```
ROLLBACK TO SAVEPOINT SECCION_1;
```

Si se ejecuta esta instrucción, todos los cambios que ocurrieron después de que fue creado el punto de recuperación `SECCION_1` son reinvertidos al punto en que se encontraba la base de datos cuando se ejecutó la instrucción `SAVEPOINT`. Incluso si se crearon otros puntos de recuperación después del punto de recuperación `SECCION_1`, los cambios se reinvierten hasta `SECCION_1`.

Pregunta al experto

P: ¿Existe alguna consecuencia al rendimiento al utilizar transacciones?

R: Por supuesto que sí. Las DBMS deben registrar cuidadosamente los efectos de cada instrucción SQL dentro de una transacción para que los cambios sean reinvertidos cuando se necesite. Y es posible llenar todas las áreas de la memoria reservadas para seguir la pista a las transacciones. Para contrarrestar esto, los programadores SQL a veces rompen los procesos que aplican muchos cambios a la base de datos en múltiples transacciones utilizando culminaciones. Revise la documentación de su DBMS para verificar si existen restricciones al utilizar culminaciones durante las repeticiones del programa, particularmente cuando los cursores son abiertos.

(continúa)

P: Anteriormente en este capítulo se dijo que las instrucciones como **SELECT**, **DELETE**, **UPDATE** y **CREATE TABLE** deben ser ejecutadas dentro del contexto de una transacción. Sin embargo, no se han utilizado estas transacciones en los ejemplos ni en los ejercicios del libro. ¿Cuándo son utilizadas estas transacciones?

R: A lo largo del libro se ha utilizado SQL interactivo (invocación directa) para comunicarse con la base de datos. La mayoría de las instrucciones SQL que se han estado ejecutando dentro de este ambiente se han hecho dentro del contexto de una transacción, a pesar de que el usuario no esté consciente de que eso suceda. Para la mayoría de las implementaciones SQL, cada instrucción SQL es considerada su propia transacción, un modo de procesamiento que a menudo es llamado autoculminación. Cuando se ejecuta la instrucción, se inicia una transacción. Si la instrucción es exitosa, cualquier cambio es completado en la base de datos y la transacción es finalizada, de la misma manera que si se hubiera ejecutado una instrucción **COMMIT**. Si la instrucción no es exitosa, los cambios son reinvertidos, la base de datos es regresada al estado en el que estaba cuando la instrucción se ejecutó por primera vez y la transacción es finalizada como si se hubiera ejecutado una instrucción **ROLLBACK**. A pesar de que SQL interactivo tiende a tratar cada instrucción como su propia transacción, usualmente se pueden ejecutar instrucciones relacionadas con la transacción en este ambiente. Sin embargo, cuáles instrucciones se pueden ejecutar y qué opciones soportan varía entre los diferentes productos; por lo tanto, asegúrese de revisar la documentación. En general, no es necesario definir específicamente una transacción en SQL interactivo.

Pruebe esto 16-1 Trabajar con transacciones

En este ejercicio se crearán varias transacciones que ejecutan instrucciones para la base de datos **INVENTARIO**. Para cada transacción se iniciará explícitamente la transacción y se ejecutarán una o más instrucciones SQL. Para este ejercicio se trabajará con la instrucción **COMMIT** y con la instrucción **ROLLBACK** en transacciones separadas debido a que se está trabajando con invocación directa de SQL (en la aplicación cliente). Sin embargo, si se estuvieran iniciando las transacciones desde dentro de un lenguaje de programación de aplicación, sin duda se estarían utilizando **COMMIT** y **ROLLBACK** juntas en algún tipo de estructura condicional. De esa manera, ciertos resultados causarían que la transacción se reinvierta, y otros resultados causarían que la transacción se complete, dependiendo de cómo se determinen las condiciones en el lenguaje de programación. Aunque, para este ejercicio, se mantendrán separadas para que pueda pasarse fácilmente a través de estos pasos. Se puede descargar el archivo **Try_This_16.txt** (en inglés), que contiene las instrucciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos **INVENTARIO**.
2. La primera transacción que se creará utiliza una instrucción **START TRANSACTION** para determinar el nivel de aislamiento a **READ UNCOMMITTED**, recupera información de la tabla **ARTISTAS**, y luego completa la transacción. Ingrese y ejecute la siguiente instrucción SQL:

```

START TRANSACTION
    ISOLATION LEVEL READ UNCOMMITTED;

SELECT *
    FROM ARTISTAS;

COMMIT;

```

La transacción deberá arrojar todas las filas y columnas de la tabla ARTISTAS.

3. La siguiente transacción que se creará también utiliza una instrucción `START TRANSACTION` para determinar el nivel de aislamiento. Pero esta vez se determinará el nivel como `SERIALIZABLE`. Debido a que `SERIALIZABLE` es el nivel preestablecido, no se requiere definirlo; sin embargo, para los propósitos de este ejercicio, vamos a incluirlo. Después de que se inicia la transacción, se intentará actualizar la tabla `DISCO_COMPACTO` incrementando el valor `EN_EXISTENCIA` por 2 para todas las filas con un valor `ID_DISQUERA` igual a 832. Después de la instrucción `UPDATE`, se reinvertirá la transacción para que ningún dato sea modificado en la base de datos. Ingrese y ejecute la siguiente instrucción SQL:

```

START TRANSACTION
    ISOLATION LEVEL SERIALIZABLE;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = EN_EXISTENCIA + 2
    WHERE ID_DISQUERA = 832;

ROLLBACK;

```

Se deberá recibir algún tipo de mensaje avisando la finalización de la transacción.

4. Ahora se confirmará que la actualización que se intentó en el paso anterior haya sido en realidad reinvertida. Ingrese y ejecute la siguiente instrucción SQL:

```

SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
    WHERE ID_DISQUERA = 832;

```

La instrucción `SELECT` deberá arrojar los siguientes resultados de la consulta:

TITULO_CD	EN_EXISTENCIA
-----	-----
That Christmas Feeling	8
Patsy Cline: 12 Greatest Hits	32
Out of Africa	29
Blues on the Bayou	27

Los valores `EN_EXISTENCIA` mostrados en estos resultados son los que estaban contenidos en la tabla `DISCO_COMPACTO` antes de que se ejecutara la transacción. Si la transacción no hubiera sido reinvertida, cada uno de estos valores se habría incrementado por 2.

(continúa)

5. Ahora vamos a agregar un punto de recuperación a la transacción que se creó en el paso anterior. Se requiere asegurarse de referirse al punto de recuperación en la instrucción **ROLLBACK**. También se agregará una instrucción **SELECT** antes del punto de recuperación. Ingrese y ejecute la siguiente instrucción SQL:

```
START TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE ID_DISQUERA = 832;

SAVEPOINT SECCION_1;

UPDATE DISCOS_COMPACTOS
  SET EN_EXISTENCIA = EN_EXISTENCIA + 2
 WHERE ID_DISQUERA = 832;

ROLLBACK TO SAVEPOINT SECCION_1;
```

Ahora la transacción solamente se reinvertirá al punto anterior a la instrucción **UPDATE**. Además, debido a que la transacción incluía una instrucción **SELECT**, se deberán recibir los resultados de la consulta que se recibieron en el paso anterior.

6. En la transacción anterior, la instrucción **SELECT** estaba antes del punto de recuperación, lo que significa que la instrucción **SELECT** fue ejecutada antes que la instrucción **UPDATE**. Si la transacción no reinvertiera la actualización, los resultados de la consulta no reflejarían la información correcta. Como resultado, se deberá verificar que la instrucción **UPDATE** haya sido reinvertida. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE ID_DISQUERA = 832;
```

Los resultados de la consulta deberán mostrar los mismos valores **EN_EXISTENCIA** que los resultados de la consulta arrojados en los dos pasos anteriores.

7. Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio Pruebe esto se crearon e iniciaron tres transacciones. En la primera, simplemente se consultaron los datos y se completó la transacción. En las siguientes dos se actualizaron los datos y luego se reinvertieron esas actualizaciones. Sin embargo, como se vio en la tercera transacción, es posible reinvertirla a un punto de recuperación específico. Esto permite proteger ciertas porciones de la transacción sin tener que volver a procesar las instrucciones que habían sido ejecutadas exitosamente. Debido a que se reinvertieron las actualizaciones realizadas, la base de datos INVENTARIO deberá permanecer en el mismo estado en que estaba cuando se comenzó este ejercicio.

✓ Autoexamen Capítulo 16

1. ¿Cuál característica de una transacción se refiere a la naturaleza todo-o-nada de una transacción?
 - A Atómica
 - B Consistente
 - C Aislada
 - D Durable
2. Un(a) _____ es una unidad de trabajo que consta de una o más instrucciones SQL que realizan un conjunto de acciones relacionado.
3. ¿Qué instrucción puede utilizarse para iniciar explícitamente una transacción?
4. ¿Cuáles instrucciones SQL finalizarán una transacción?
 - A SAVEPOINT
 - B SET TRANSACTION
 - C ROLLBACK
 - D COMMIT
5. ¿Cuáles son los tres tipos de modo de transacción que pueden especificarse en una instrucción SET TRANSACTION?
6. ¿Cuáles opciones del nivel de acceso pueden incluirse en una instrucción START TRANSACTION?
 - A READ ONLY
 - B UPDATE
 - C LOCAL
 - D READ WRITE
7. Dos transacciones simultáneas están activas en el sistema. La primera transacción modifica los datos en una tabla. La segunda transacción ve esas modificaciones antes de que sean completadas realmente en la base de datos. Luego, la primera transacción reinvierte las modificaciones. ¿Qué tipo de irregularidad de datos ha ocurrido?
 - A Lectura fantasma
 - B Lectura repetible
 - C Lectura sucia
 - D Lectura no repetible

8. Una lectura _____ puede ocurrir cuando una transacción lee una tabla basada en algún tipo de condición de búsqueda, después una segunda transacción actualiza los datos en la tabla, y la primera transacción intenta volver a leer los datos, sólo que esta vez se arrojan diferentes filas debido a como está definida la condición de búsqueda.
9. ¿Qué tipo de restricción puede especificarse en una instrucción SET CONSTRAINTS?
10. ¿Cuál de los niveles de aislamiento aísla por completo una transacción de otra?
11. Se está utilizando una instrucción SET TRANSACTION para configurar los modos de transacción. Se requiere asegurarse de que no puedan ocurrir lecturas no repetibles ni lecturas sucias dentro de la transacción. Sin embargo, no es necesario preocuparse acerca de las lecturas fantasma. ¿Cuál nivel de aislamiento deberá utilizarse?
- A** READ UNCOMMITTED
 - B** READ COMMITTED
 - C** REPEATABLE READ
 - D** SERIALIZABLE
12. Se está configurando una transacción que aplaze la aplicación de la restricción REST_CD_EXISTENCIA hasta que se ejecuten varias instrucciones SQL. Después de ejecutar las instrucciones, se requiere aplicar explícitamente la restricción a los cambios que se hicieron a la base de datos. ¿Qué instrucción SQL deberá utilizarse para aplicar las restricciones?
13. Un(a) _____ es un marcador designado dentro de la transacción que actúa como un punto para reinvertir una porción de la transacción.
14. Se necesita crear un punto de recuperación llamado svpt_Seccion2. ¿Qué instrucción SQL deberá utilizarse?
15. Se crea una transacción que incluye cuatro puntos de recuperación: Seccion1, Seccion2, Seccion3 y Seccion4. Cerca del final de la transacción, posterior a los cuatro puntos de recuperación, se define RELEASE SAVEPOINT, que especifica el punto de recuperación Seccion2. ¿Cuál punto o cuáles puntos de recuperación son eliminados de la transacción cuando se ejecuta la instrucción RELEASE SAVEPOINT?
- A** Seccion1
 - B** Seccion2
 - C** Seccion3
 - D** Seccion4
16. ¿Qué circunstancias finalizarán una transacción?
17. Se está creando una instrucción ROLLBACK en la transacción. Se necesita que la reinversión deshaga los cambios hasta el punto de recuperación svpt_Seccion2. ¿Qué instrucción SQL deberá utilizarse?
18. Se está creando una instrucción COMMIT en la transacción. Después de que la transacción es finalizada, se requiere que inicie una nueva transacción. La nueva transacción deberá estar configurada con los mismos modos de transacción que la primera transacción. ¿Cómo deberá crearse la instrucción COMMIT?

Capítulo 17

Acceder a datos SQL
desde un programa
host

Habilidades y conceptos clave

- Invocar SQL directamente
 - Incrustar instrucciones SQL en el programa
 - Crear módulos cliente de SQL
 - Utilizar una interfaz de nivel de llamada de SQL
-

A lo largo de todo el libro se han realizado ejercicios y ejemplos de prueba utilizando una aplicación de cliente para trabajar interactivamente con la base de datos SQL. Por ejemplo, se podría haber utilizado SQL Server Management Studio para acceder a una base de datos de SQL Server, SQL*Plus o iSQL*Plus para acceder a una base de datos de Oracle, o quizá MySQL Command Line Client para acceder a una base de datos de MySQL. Este método de acceso de datos se conoce como invocación directa o SQL interactivo. El estándar SQL:2006 también hace posible el uso de otros tipos de acceso de datos, incluyendo SQL incrustado, módulos cliente de SQL y la interfaz de nivel de llamada (CLI, por sus siglas en inglés); sin embargo, los tipos de acceso de datos soportados por una implementación SQL a menudo varían de producto a producto. Algunos, por ejemplo, no soportan SQL incrustado, y pocos soportan los módulos cliente de SQL. En este capítulo se introducirán los cuatro tipos de métodos de acceso de datos y se explicará cómo pueden ser utilizados para recuperar y modificar datos en la base de datos SQL. Debido a que SQL y CLI son los dos métodos más comúnmente utilizados por los programas para acceder a los datos SQL, se cubrirán estos dos temas con mayor detalle que la invocación directa y los módulos cliente SQL, a pesar de que se proporciona una base para los cuatro tipos de acceso.

Invocar SQL directamente

Al llegar hasta este punto del libro, ya deberá sentirse muy cómodo con SQL interactivo. Utilizando la aplicación cliente, que viene con la mayoría de los productos de manejo de bases de datos, se han podido crear instrucciones SQL *ad hoc* que arrojan resultados inmediatos a la aplicación. Estos resultados normalmente son desplegados en una ventana separada de donde se ejecutó la instrucción SQL. Por ejemplo, demos un vistazo a la figura 17-1, que nos muestra SQL Server Management Studio. Observe que la ventana superior incluye una instrucción SELECT y que la ventana inferior incluye los resultados de la consulta de haber ejecutado esa instrucción. La mayoría de las aplicaciones cliente de invocación directa se comportan de manera similar a ésta.

Los tipos de instrucciones SQL soportados por el método de invocación directa pueden variar de una implementación SQL a otra. A pesar de que la mayoría de las implementaciones permitirán ejecutar los tipos básicos de instrucciones, por ejemplo SELECT o UPDATE, puede que no permitan la ejecución de instrucciones específicas para otro método de acceso de datos. Por ejemplo, algunas implementaciones pudieran no permitir declarar un cursor dentro de un ambiente interactivo.

A pesar de las diferencias entre las implementaciones SQL, el estándar SQL no define cuáles tipos de instrucciones deberán ser soportadas en un ambiente interactivo. Esto incluye a las

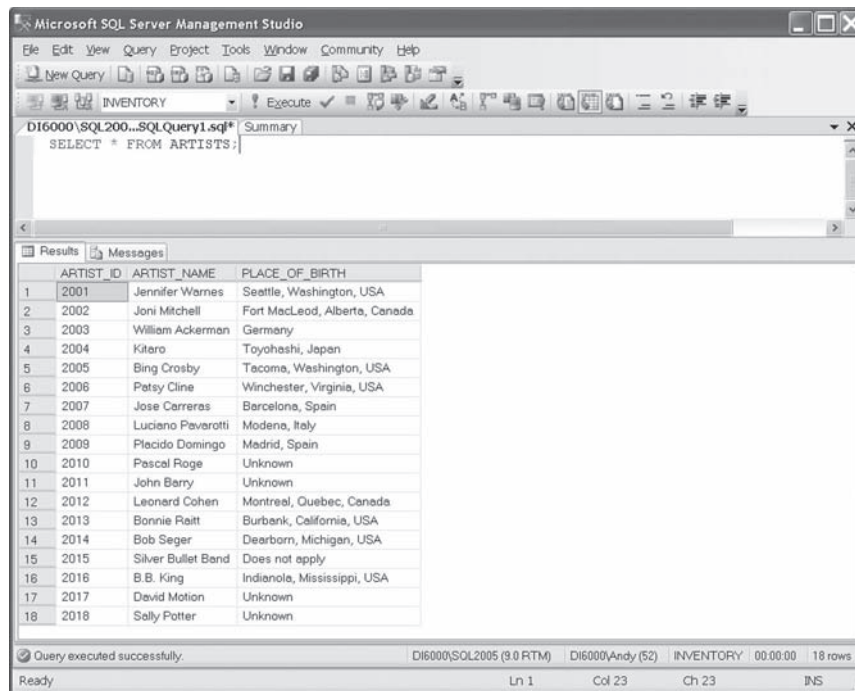


Figura 17-1 SQL Server Management Studio (SQL Server 2005 y 2008).

instrucciones `SELECT`, `INSERT`, `UPDATE` y `DELETE`, y a las instrucciones relacionadas con las definiciones de esquema, transacciones, conexiones y sesiones. También se deberá ser capaz de declarar tablas temporales en un ambiente interactivo. De hecho, prácticamente cualquier acción crítica para el mantenimiento de los datos y de la estructura de la base de datos subyacente deberá ser soportada por invocación directa.

Una de las principales ventajas de SQL interactivo (además de la habilidad de ejecutar instrucciones *ad hoc*) es la eliminación de cualquier incongruencia en la impedancia. Como se podrá recordar de análisis anteriores, una incongruencia en la impedancia puede ocurrir debido a diferencias en los tipos de datos entre SQL y los lenguajes de programación de aplicación, y también debido a la forma en que son manejados los resultados de la consulta (conjuntos de resultados) entre SQL y esos lenguajes. Sin embargo, SQL interactivo es un ambiente SQL puro, lo que significa que solamente pueden ser utilizados los tipos de datos soportados por la implementación, y los conjuntos de resultados no generan ningún problema para la aplicación de cliente debido a que simplemente el usuario puede desplazarse a través de los resultados de la consulta. También es común para los desarrolladores de las aplicaciones utilizar SQL interactivo para probar las instrucciones SQL que intentan incrustar en otros módulos. Aun con eso, la invocación directa representa solamente un pequeño porcentaje de usuarios. Encontrará que la mayoría de los accesos de datos se hacen a través de SQL incrustado y de mecanismos del tipo CLI, y algunos otros a través de los módulos cliente SQL, pero relativamente muy pocos usuarios utilizan sólo SQL interactivo.

Incrustar instrucciones SQL en el programa

En el capítulo 15, cuando se analizaron los cursores SQL, se introdujo SQL incrustado. Como se podrá recordar de ese análisis, SQL incrustado se refiere a las instrucciones SQL que están dispersas en algún tipo de lenguaje de programación de aplicación. Las instrucciones SQL son fusionadas en el lenguaje host para permitir al programa fuente acceder y modificar los datos SQL y la estructura de la base de datos subyacente.

De acuerdo con el estándar SQL:2006 es posible incrustar instrucciones SQL en los siguientes lenguajes de programación:

- Ada
- C
- COBOL
- Fortran
- MUMPS
- Pascal
- PL/I

A pesar de que el estándar soporta instrucciones SQL incrustadas en estos lenguajes, las implementaciones SQL raramente soportan las instrucciones incrustadas en todos estos lenguajes. Una implementación pudiera estar limitada a solamente uno o dos lenguajes de programación, y algunas implementaciones pudieran no soportar del todo SQL incrustado (a pesar de que la mayoría de las implementaciones proporcionan SQL incrustado para al menos un lenguaje). Además, muchas implementaciones soportan SQL incrustado en lenguajes diferentes a aquellos especificados en el estándar SQL. Por ejemplo, Oracle ofrece SQLJ, que soporta SQL incrustado en programas de Java.

Cuando un programa contiene instrucciones SQL incrustadas, debe ser compilado en una forma diferente a los programas regulares. La figura 17-2 ilustra el proceso que se sigue cuando se compilan estos programas.

Como se puede ver en la figura, se inicia con un archivo de programa que contiene el lenguaje de programación host y las instrucciones SQL incrustadas. Antes de que el programa sea compilado, éste es sometido a un precompilador que es específico al lenguaje de programación host y a la implementación SQL. El precompilador retira las instrucciones SQL del código del lenguaje host, a menudo convirtiéndolas en comentarios para documentar las instrucciones originales, y las reemplaza con llamadas a las rutinas de la biblioteca proporcionada por el proveedor que acceden a las instrucciones SQL. Como resultado, se crean dos archivos, uno para el lenguaje host y otro para las instrucciones SQL.

Una vez que se crea un archivo para el lenguaje host, el programa fuente es compilado de manera normal, como se esperaría de un lenguaje específico. El resultado del compilador del lenguaje host es el código objeto, que está vinculado a varias rutinas de biblioteca. Desde esto, se genera un programa ejecutable que se vincula al plan de la aplicación. Ésto es creado por una utilidad de enlace que valida y optimiza las instrucciones SQL. El plan contiene las instrucciones SQL y la información que el programa necesita para acceder a la base de datos.

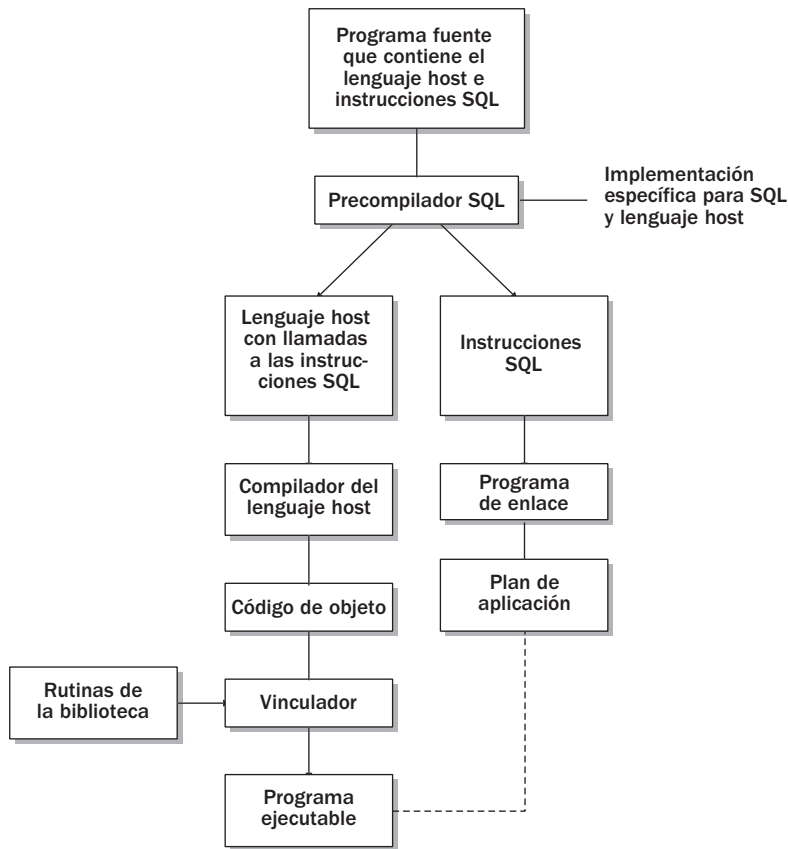


Figura 17-2 Compilar programas que contienen SQL incrustado.

Crear una instrucción SQL incrustada

Cuando se desarrolla un programa que contiene SQL incrustado, se deben seguir convenciones específicas que determinan cómo el código SQL debe ser agregado al programa. Estas convenciones están basadas en una combinación de elementos especiales del lenguaje SQL y los requerimientos del lenguaje de programación host. Para ser utilizada en un lenguaje host, una instrucción SQL incrustada debe cumplir con las siguientes pautas:

- Cada instrucción SQL debe iniciar con un prefijo cualificado.
- Cada instrucción SQL puede o no requerir un finalizador cualificado, dependiendo del lenguaje host.
- Las nuevas líneas dentro de la instrucción SQL deben ser manejadas de acuerdo con el estilo del lenguaje host.
- La colocación de los comentarios debe ser manejada de acuerdo con el estilo del lenguaje host.

Lenguaje	Prefijo	Finalizador
Ada	EXEC SQL	;
C	EXEC SQL	;
COBOL	EXEC SQL	END-EXEC
Fortran	EXEC SQL	(sin finalizador)
MUMPS	&SQL()
Pascal	EXEC SQL	;
PL/I	EXEC SQL	;

Tabla 17-1 Iniciar y finalizar una instrucción SQL.

La mayoría de las instrucciones SQL incrustadas requieren un prefijo cualificado y un finalizador. La tabla 17-1 proporciona el prefijo y el finalizador para cada lenguaje de programación soportado.

Como se puede ver en la tabla, los lenguajes Ada, C, Pascal y PL/I manejan las instrucciones SQL incrustadas de la misma manera. Por ejemplo, supongamos que se quiere incrustar una instrucción SELECT que recupere las columnas NOMBRE_CD y EN_EXISTENCIA de la tabla INVENTARIO_CD. Para hacerlo, se utilizaría la siguiente instrucción:

```
EXEC SQL
  SELECT NOMBRE_CD, EN_EXISTENCIA
  FROM INVENTARIO_CD;
```

Observe que la instrucción está precedida por el prefijo EXEC SQL y terminada con el finalizador punto y coma. Si fuera a crearse una instrucción similar en otro lenguaje, ya sea el prefijo o el finalizador podrían ser diferentes. En el caso de MUMPS, ambos serían diferentes.

NOTA

Normalmente, no tiene importancia si las instrucciones SQL incrustadas aparecen en mayúsculas o minúsculas. Los programadores generalmente siguen las convenciones del lenguaje host. Sin embargo, para los propósitos de este capítulo, se tratarán las instrucciones SQL incrustadas al igual que se ha hecho con otras instrucciones SQL a lo largo del libro: se utilizarán mayúsculas tanto para las palabras clave de SQL como para los identificadores de SQL.

Utilizar variables host en las instrucciones SQL

Para utilizar SQL incrustado de manera efectiva, debe ser posible pasar valores entre el programa host y las instrucciones SQL. Por ejemplo, la instrucción SQL incrustada pudiera incluir una cláusula WHERE que requiera un valor específico para poder evaluar la condición de búsqueda. El valor pudiera ser proporcionado por un usuario o por una operación dentro del programa host. En cualquiera de los casos, el valor deberá ser pasado de alguna manera del programa a la instrucción SQL.

Para pasar valores desde y hacia una instrucción SQL, se pueden utilizar las variables host. Una *variable host* es un tipo de parámetro que es declarado dentro del lenguaje host y que luego es referenciado dentro de la instrucción SQL incrustada. Cuando una variable host es utilizada dentro de una instrucción SQL, el nombre de la variable deberá estar precedido por dos puntos. Éstos le

avisan al precompilador que el elemento nombrado es una variable y no un objeto de la base de datos, como una tabla o una columna. Como resultado, no representa una preocupación si la variable host comparte el mismo nombre que un objeto de la base de datos.

Una variable host puede utilizarse en una instrucción SQL incrustada en cualquier lugar que sea necesaria para definir un valor. Por ejemplo, la siguiente instrucción SELECT puede ser incrustada en un programa C:

```
EXEC SQL
    SELECT NOMBRE_CD, EN_EXISTENCIA
    FROM INVENTARIO_CD
    WHERE ID_CD = :v_ID_CD;
```

Observe que la variable host `v_ID_CD` está precedida por dos puntos. Éstos se utilizan únicamente dentro de la instrucción SQL. Cuando se declara la variable (anterior en el programa host) no se utilizan los dos puntos.

NOTA

El ejemplo anterior inicia con EXEC SQL y termina con punto y coma. Además, no se requiere ningún carácter de continuación específico para indicar una nueva línea en la instrucción SQL. Estas convenciones son consistentes con lo que se esperaría encontrar en un programa C. Para los ejemplos en este capítulo se utilizarán instrucciones SQL incrustadas tal como aparecerían dentro de un programa C. Varias de las implementaciones SQL soportan sin problema SQL incrustado en C.

No se está limitado a referenciar una variable host en la cláusula WHERE de una instrucción SELECT. Por ejemplo, se puede referenciar una variable en la cláusula SET de una instrucción UPDATE, en la cláusula VALUES de una instrucción INSERT, o en la cláusula WHERE de una instrucción UPDATE o DELETE. Sin embargo, no se puede utilizar una variable host en lugar de un identificador SQL. En otras palabras, no se puede pasar un nombre de objeto, por ejemplo, un nombre de tabla o nombre de columna, a través de una variable host.

Declarar variables host

Como se mencionó anteriormente, se deben declarar las variables host dentro del programa host. Es posible declararlas en cualquier punto del programa donde normalmente se declararían variables en un lenguaje en particular. Además, se deben declarar las variables de acuerdo con las convenciones del lenguaje host. La única diferencia es que se deben iniciar las instrucciones con la instrucción BEGIN DECLARE SECTION y finalizar las instrucciones con la instrucción END DECLARE SECTION. Estas dos instrucciones notifican al precompilador que las variables encerradas en las instrucciones pueden utilizarse en otras instrucciones SQL incrustadas.

Demos un vistazo a un ejemplo de lo que se está tratando de describir. Supongamos que se quieren declarar dos variables, una para recibir un valor que identifique el CD y una que reciba el nombre del CD. La instrucción de la variable en C podría lucir como la siguiente:

```
EXEC SQL
    BEGIN DECLARE SECTION;
        long    v_ID_CD;           /* ID del disco compacto */
        varchar v_NOMBRE_CD[60]; /* nombre del disco compacto */
EXEC SQL
    END DECLARE SECTION;
```

Como se puede ver, las instrucciones de la variable están encerradas en dos instrucciones SQL relacionadas con las instrucciones. Observe que estas instrucciones son tratadas exactamente de la misma manera que cualquiera otra instrucción SQL incrustada en C. Cada instrucción inicia con EXEC SQL y finaliza con punto y coma.

Dos variables host están siendo declaradas en esta sección. La primera de ellas, `v_ID_CD`, es declarada con el tipo de datos long, y la segunda variable, `v_NOMBRE_CD`, es declarada con el tipo de datos varchar. Las dos instrucciones de variables siguen las convenciones del lenguaje host. Observe que un comentario sigue a cada instrucción. Los comentarios también se adhieren a las convenciones del lenguaje host.

Cuando se utilizan las variables host en instrucciones SQL, puede ocurrir una incongruencia en la impedancia como resultado de las diferencias entre los tipos de datos del lenguaje host y los tipos de datos SQL. Como se pudo ver en el ejemplo anterior, las variables se declaran con tipos de datos C; sin embargo, las variables se utilizarán en instrucciones SQL para pasar datos a las columnas que están configuradas con tipos de datos SQL. Si los tipos de datos son compatibles, entonces los datos pueden ser pasados a través de las variables; de otra manera, la incongruencia en la impedancia entre los tipos de datos evita que los valores puedan ser pasados. Por ejemplo, la variable `v_ID_CD` está configurada con el tipo de datos long, que es compatible con el tipo de datos INTEGER en SQL, y la variable `v_NOMBRE_CD` está configurada con el tipo de datos varchar, que es compatible con el tipo de datos CHARACTER VARYING (usualmente abreviado como VARCHAR) en SQL. Como resultado, es posible pasar datos a través de estas variables siempre y cuando las columnas que los reciban estén configuradas con tipos de datos compatibles.

Pregunta al experto

P: Se dijo que los datos pueden ser pasados desde la variable hacia la instrucción SQL si los tipos de datos son compatibles. ¿Cómo sería posible pasar datos si no son compatibles?

R: La mayoría de los lenguajes de programación contienen por lo menos algunos tipos de datos que no coinciden con los tipos de datos en SQL. Si surge esta situación, se puede utilizar la expresión de valor CAST dentro de la instrucción SQL para convertir el valor de la variable en un valor que pueda ser utilizado por la instrucción SQL. De manera efectiva, la expresión de valor CAST cambia el tipo de datos del valor. Por ejemplo, es posible modificar la instrucción incrustada anterior para convertir la variable host `v_ID_CD`, como se muestra en el siguiente ejemplo:

```
EXEC SQL
  SELECT NOMBRE_CD, EN_EXISTENCIA
    FROM INVENTARIO_CD
   WHERE ID_CD = CAST(:v_ID_CD AS INT);
```

Como se puede ver, se utiliza CAST para convertir el valor en la variable host a un tipo de datos INTEGER. (Para mayor información acerca de CAST, véase el capítulo 10.) Para determinar cuáles tipos de datos en un lenguaje host son compatibles con los tipos de datos en SQL, deberá referirse al estándar SQL:2006, a la documentación específica del lenguaje o la documentación específica del producto.

Recuperar datos SQL

Como se ha visto lo largo de este libro, el proceso de consultar datos en una base de datos SQL involucra ejecutar una instrucción `SELECT` que en su momento recupera los datos desde la tabla o tablas aplicables y arroja esos datos en un conjunto de resultados. Un conjunto de resultados puede estar conformado por una o más filas y una o más columnas. Cuando se están consultando los datos de forma interactiva, las filas múltiples no representan ningún problema debido a que la aplicación de cliente puede manejar más de una fila. Sin embargo, cuando se consultan los datos desde una instrucción SQL incrustada, las filas múltiples tienen que ser manejadas a través de un cursor para poder permitir que el lenguaje host trabaje con una fila a la vez. Un cursor, como se podrá recordar, actúa como un señalador para especificar las filas en el conjunto de resultados. La instrucción del cursor define la instrucción `SELECT` que recupera los datos desde la base de datos, y las instrucciones relacionadas al cursor son entonces utilizadas para recuperar las filas individuales de ese conjunto de resultados. (Para mayor información acerca de los cursores, véase el capítulo 15.)

Los cursores, por lo tanto, proporcionan una solución a un tipo de incongruencia en la impedancia que pueda ocurrir entre SQL y el lenguaje host. Específicamente, SQL arroja datos en conjuntos, y la mayoría de los lenguajes de programación no pueden manejar conjuntos. Utilizando algún tipo de construcción de repetición dentro del lenguaje de programación y luego usando la instrucción `FETCH` de SQL, es posible alternar entre cada fila del conjunto de resultados para recuperar los datos que se necesiten.

A pesar de la disponibilidad de los cursores en SQL incrustado, existen ocasiones en las que se sabe que la consulta de la base de datos arrojará solamente una fila. Por ejemplo, pudiera requerirse recuperar los datos acerca de un CD específico o de un artista específico, en cuyo caso un cursor es innecesario. Para facilitar las recuperaciones de una sola fila, SQL incrustado soporta la instrucción `SELECT` *de instancia única*. Una instrucción `SELECT` de instancia única es similar a una instrucción `SELECT` regular excepto por dos detalles:

- No se incluye una cláusula `GROUP BY`, `HAVING` u `ORDER BY`.
- Se incluye una cláusula `INTO` que especifica las variables host que recuperarán los datos arrojados por la instrucción `SELECT` hacia el programa host.

Por ejemplo, supongamos que la instrucción `SELECT` arroja el nombre del CD y el número de existencias, como se muestra en la siguiente instrucción incrustada:

```
EXEC SQL
  SELECT NOMBRE_CD, EN_EXISTENCIA
  INTO :v_NOMBRE_CD, :v_EN_EXISTENCIA
  FROM INVENTARIO_CD
  WHERE ID_CD = :v_ID_CD;
```

Como se puede ver en esta instrucción, la variable `v_ID_CD` es utilizada para especificar cuál CD deberá ser arrojado. El valor es ingresado por el usuario, y la variable pasa ese valor del programa host a la instrucción `SELECT`.

Ahora demos un vistazo a la cláusula `INTO`. Observe que la cláusula contiene dos variables, el mismo número de variables que el número de columnas recuperadas de la tabla `INVENTARIO_CD`. Estas variables son declaradas de la misma forma que otras variables host que se han

visto. Debido a que esta instrucción `SELECT` arroja solamente una fila y dos columnas, solamente dos valores son arrojados. Estos valores son transferidos a las variables. Las variables deben ser especificadas en el mismo orden que son especificados los nombres de las columnas.

NOTA

La variable `v_ID_CD` en la cláusula `WHERE` es una variable host de entrada, y las variables `v_NOMBRE_CD` y `v_EN_EXISTENCIA` en la cláusula `INTO` son variables host de salida. La única diferencia radica en cómo son utilizadas por la instrucción `SQL` (ya que son definidas exactamente de la misma manera dentro del lenguaje host).

Manejar valores nulos

En el capítulo 4 se analizaron los valores nulos y cómo son utilizados para representar datos desconocidos o no disponibles. Como se recordará de esa discusión, la mayoría de las columnas `SQL`, de manera predeterminada, permiten valores nulos, a pesar de que puede sobrescribirse ese valor de forma preestablecida o al definir la restricción `NOT NULL` en la columna. Sin embargo, si no se sobrescribe el valor de manera predeterminada y los valores nulos son permitidos, se puede generar un problema con el lenguaje host debido a que la mayoría de los lenguajes de programación de aplicación no soportan los valores nulos.

Para encargarse de este tema, `SQL` permite declarar variables host de indicador. Una *variable host de indicador* es un tipo de variable que acompaña a una variable host regular, que también se conoce como *variable host de datos*. La variable de indicador contiene un valor que especifica si un valor es o no nulo en la variable de datos asociada. Las variables de indicador son declaradas de la misma forma que otras variables host.

Observemos un ejemplo de las variables de indicador para ilustrar cómo funcionan. En la siguiente instrucción `SELECT` incrustada, una variable de indicador ha sido agregada a cada una de las variables de datos en la cláusula `INTO`:

```
EXEC SQL
  SELECT NOMBRE_CD, EN_EXISTENCIA
    INTO :v_NOMBRE_CD :ind_NOMBRE_CD, :v_EN_EXISTENCIA :ind_EN_EXISTENCIA
  FROM INVENTARIO_CD
 WHERE ID_CD = :v_ID_CD;
```

La cláusula `INTO` incluye dos variables de indicador: `ind_NOMBRE_CD` e `ind_EN_EXISTENCIA`. Observe que cada variable de indicador sigue a la variable de datos asociada. La colocación de la variable de indicador es el único indicio que la implementación `SQL` tiene para determinar que una variable host en particular es una variable de indicador. No existe nada en la instrucción de la variable o en su denominación que distinga a las variables de indicador de las variables de datos. Cuando la implementación ve que una variable sigue a la otra y que ninguna coma las separa, la implementación asigna un valor de 0 a la variable de indicador si la variable asociada contiene un valor real (si no es nula). Si la variable asociada contiene un valor nulo, la implementación asigna un valor de -1 a la variable de indicador. Entonces el programa host toma la acción apropiada basado en esta información. Es *esencial* que el programa host verifique las variables de indicador acerca de los valores nulos *antes* de intentar utilizar los valores en la variable host debido a que cuando se encuentran valores nulos, el valor de la variable host correspondiente *no cambia*, y por lo tanto podría contener valores sobrantes de una instrucción anterior o datos desconocidos que los programadores llaman basura.

NOTA

Al declarar una variable de indicador, asegúrese de utilizar un tipo de datos numérico que soporte los valores 0 y -1. El valor 0 es lógicamente una condición falsa debido a que todos los bits serán determinados a un binario 0, mientras que el valor -1 es lógicamente una condición verdadera debido a que todos los bits serán determinados a un binario 1.

Manejo de errores

Cuando se incrusten instrucciones SQL en el lenguaje host, se deberá proporcionar una forma de tomar acciones específicas si se recibe un mensaje de error o de advertencia cuando se intenta acceder a los datos. SQL proporciona un método relativamente sencillo que puede utilizarse para monitorear los errores y advertencias y tomar acciones dependiendo de los resultados de ese monitoreo. Al incrustar instrucciones **WHENEVER** en el lenguaje host, se le otorga al programa un nivel efectivo de manejo de errores que funciona lado a lado con las otras instrucciones incrustadas.

La instrucción **WHENEVER** incluye dos conjuntos de opciones, como se muestra en la siguiente sintaxis:

```
WHENEVER
{ SQLEXCEPTION | SQLWARNING | NOT FOUND }
{ CONTINUE | GOTO <objetivo> }
```

Como se puede ver, primero se debe especificar la palabra clave **WHENEVER** y luego especificar las opciones necesarias. El primer conjunto de opciones indica la condición en la cual aplica la instrucción **WHENEVER**. Si se cumple esa condición, se toma alguna acción específica. Una instrucción **WHENEVER** puede incluir una de tres condiciones:

- **SQLEXCEPTION** Se cumple la condición en cualquier momento que una instrucción SQL genera una excepción. Por ejemplo, una excepción puede ser generada cuando se intenta insertar datos inválidos a una columna.
- **SQLWARNING** Se cumple la condición en cualquier momento que una instrucción SQL genera una advertencia. Por ejemplo, una instrucción puede generar una advertencia si un número ha sido redondeado.
- **NOT FOUND** Se cumple la condición en cualquier momento que una instrucción **SELECT** no puede arrojar los datos en los resultados de su consulta. Esto puede aplicar a una instrucción **SELECT** de instancia única o a una instrucción **FETCH** al final del conjunto de resultados del cursor.

Una vez que se especifica una condición en la instrucción **WHENEVER**, se debe especificar una acción. La instrucción **WHENEVER** soporta dos acciones:

- **CONTINUE** El programa continuará ejecutándose en la siguiente instrucción.
- **GOTO <objetivo>** El programa brincarà a una sección dentro del lenguaje host que sea denominada en el marcador de posición <objetivo>.

Ahora que hemos visto las opciones disponibles en la instrucción **WHENEVER**, demos un vistazo a un ejemplo. Supongamos que se requiere que las instrucciones SQL vayan a una cierta parte del programa si ocurre un error. En la siguiente instrucción **WHENEVER**, una excepción causará que el programa se mueva a la sección **Error1**:


```
EXEC SQL
  WHENEVER SQLEXCEPTION GOTO Error1;
```

Observe que la opción `SQLEXCEPTION` y la opción `GOTO` están especificadas en esta instrucción. La opción `SQLEXCEPTION` le dice al programa que tome una acción específica si una instrucción SQL genera una excepción. La opción `GOTO` define la acción que deberá ser tomada. En este caso, la opción específica que el programa deberá moverse a la sección `Error1` del lenguaje host.

Una instrucción `WHENEVER` aplica a las instrucciones SQL incrustadas que le siguen. Es posible incrustar tantas instrucciones `WHENEVER` en el lenguaje host como sean necesarias. La última instrucción en aparecer es la única que se aplica a las otras instrucciones.

Pruebe esto 17-1 Incrustar instrucciones SQL

En la mayoría de los ejercicios del libro se utilizó una aplicación cliente para acceder a la base de datos SQL de forma interactiva. Sin embargo, debido a la naturaleza del tema en este capítulo, particularmente respecto a SQL incrustado, este ejercicio tomará un enfoque diferente que los anteriores. Para este ejercicio se utilizará algún tipo de programa de edición de texto (por ejemplo, el Bloc de notas de Microsoft) para completar los pasos. Debido a que programar en un lenguaje host está fuera del enfoque de este libro, solamente se crearán las instrucciones SQL que están incrustadas en el lenguaje host. Las instrucciones se ajustarán al lenguaje C, a pesar de que pueden aplicarse a otros lenguajes host. Durante el ejercicio se determinarán instrucciones de variables, se creará una instrucción para manejar errores y se incrustará una instrucción `SELECT` de SQL que consulte los datos de la base de datos `INVENTARIO`. Se puede descargar el archivo `Try_This_17-1.txt` (en inglés), que contiene las instrucciones SQL incrustadas utilizadas en este ejercicio.

Paso a paso

1. Abra un programa de edición de texto como el Bloc de notas de Microsoft.
2. El primer paso es crear una variable host de entrada y dos variables host de salida. El propósito de la variable host de entrada es poder recibir del usuario un identificador de CD. El identificador puede ser utilizado en la cláusula `WHERE` de la instrucción `SELECT` para determinar cuál fila de datos será arrojada de la tabla `DISCOS_COMPACTOS`. Junto con la instrucción de las variables, se incluirán comentarios que identifican el propósito de esas variables. Escriba las siguientes instrucciones SQL incrustadas y las instrucciones de variables en el documento de texto:

```
EXEC SQL
  BEGIN DECLARE SECTION;
    long    v_ID_CD;          /* variable de entrada para el
                              identificador de CD */
    varchar v_TITULO_CD[60]; /* variable de salida para el título del
                              CD */
    long    v_EN_EXISTENCIA; /* variable de salida para el valor EN_
                              EXISTENCIA */
EXEC SQL
  END DECLARE SECTION;
```

Observe que la sección de la instrucción de variable es obligatoria, por lo que se tuvo que incluir tanto la instrucción `BEGIN DECLARE SECTION` como la instrucción `END DECLARE SECTION`. Estas instrucciones son necesarias para notificar al precompilador que las instrucciones de variables serán utilizadas en las instrucciones SQL incrustadas.

- Después de crear la sección de la instrucción, se verá que es necesario incluir las variables de indicador para las variables de datos de salida. Como resultado, se deben agregar dos instrucciones a la sección de instrucción. Escriba las siguientes instrucciones en el documento de texto:

```
short ind_TITULO_CD;      /* variable de indicador para v_TITULO_CD */
short ind_EN_EXISTENCIA; /* variable de indicador para v_EN_EXISTENCIA */
```

Es posible agregar las instrucciones en cualquier lugar de la sección de instrucción. Sin embargo, para que el código sea limpio, se sugiere agregarlas cerca de cada una de las variables de datos asociadas, como se muestra en la siguiente sección de instrucción:

```
EXEC SQL BEGIN DECLARE SECTION;
    long    v_CD_ID;          /* variable de entrada para el
                               identificador de CD */
    varchar v_TITULO_CD[60]; /* variable de salida para el título del
                               CD */
    short   ind_TITULO_CD;    /* variable de indicador para v_TITULO_CD */
    long    v_EN_EXISTENCIA; /* variable de salida para el valor EN_
                               EXISTENCIA */
    short   ind_EN_EXISTENCIA; /* variable de indicador para v_EN_
                               EXISTENCIA */
EXEC SQL END DECLARE SECTION;
```

Observe que las dos nuevas instrucciones han sido insertadas debajo de sus respectivas variables de datos.

- Ahora incluiremos una instrucción de manejo de errores en el documento de texto. La instrucción representará una sección llamada `Error1` en el lenguaje host. Se asumirá que si una instrucción SQL incrustada genera una excepción, el programa brincaré a la sección `Error1` y tomará cualquier acción que esté definida en esa sección. Escriba la siguiente instrucción SQL incrustada en el documento de texto:

```
EXEC SQL
    WHENEVER SQLEXCEPTION GOTO Error1;
```

Observe que el código SQL incrustado contiene una instrucción `WHENEVER` que especifica las opciones `SQLEXCEPTION` y `GOTO`.

- Ahora se está listo para crear la instrucción `SELECT` incrustada. La instrucción contendrá las variables definidas en la sección de instrucción. Además, se utilizará una instrucción `SELECT` de instancia única debido a que la instrucción recuperará solamente una fila a la vez. La cláusula `WHERE` está basada en un valor `ID_DISCO_COMPACTO` específico, y cada valor es único dentro de la tabla `DISCOS_COMPACTOS`. (La columna `ID_DISCO_COMPACTO` es la clave primaria, por lo que los valores deben ser únicos dentro de esa columna.) Escriba la siguiente instrucción SQL incrustada en el documento de texto:

```
EXEC SQL
  SELECT TITULO_CD, EN_EXISTENCIA
  INTO :v_TITULO_CD :ind_TITULO_CD, :v_EN_EXISTENCIA :ind_EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
  WHERE ID_DISCO_COMPACTO = :v_ID_CD;
```

Observe que una cláusula INTO está incluida en esta instrucción. La cláusula INTO contiene las variables de datos de salida y sus variables de indicador asociadas. El documento de texto deberá ahora lucir como el siguiente código:

```
EXEC SQL
  BEGIN DECLARE SECTION;
    long    v_CD_ID;           /* variable de entrada para el
                                identificador de CD */
    varchar v_TITULO_CD[60];   /* variable de salida para el título del
                                CD */
    short   ind_TITULO_CD;     /* variable de indicador para v_TITULO_
                                CD */
    long    v_EN_EXISTENCIA;   /* variable de salida para el valor EN_
                                EXISTENCIA */
    short   ind_EN_EXISTENCIA; /* variable de indicador para v_EN_
                                EXISTENCIA */

EXEC SQL
  END DECLARE SECTION;
EXEC SQL
  WHENEVER SQLEXCEPTION GOTO Error1;
EXEC SQL
  SELECT TITULO_CD, EN_EXISTENCIA
  INTO :v_TITULO_CD :ind_TITULO_CD, :v_EN_EXISTENCIA :ind_EN_
EXISTENCIA
  FROM DISCOS_COMPACTOS
  WHERE ID_DISCO_COMPACTO = :v_CD_ID;
```

Si éste fuera un programa C real, se vería también el código C rodeando a las instrucciones SQL incrustadas. El código C representaría a ese programa real y tomaría acciones apropiadas a ese programa. Por ejemplo, el lenguaje host incluiría código que permitiría al programa recibir el identificador de CD para el usuario. El identificador sería pasado a la variable `v_ID_CD` para ser utilizado en la instrucción `SELECT` incrustada.

6. Guarde el archivo y cierre la aplicación.

Resumen de Pruebe esto

En este ejercicio se creó una sección de instrucción de variable host, se declararon cinco variables host, se agregó una instrucción para manejo de errores y se incrustó una instrucción `SELECT` de instancia única. Si éste fuera un programa C completo, el lenguaje host habría utilizado los datos en las variables de salida para tomar cualquier acción apropiada para el programa. El programa C también incluiría una sección llamada `Error1` que especificaría una acción en específico a ser tomada si se generara una excepción cuando la instrucción SQL es ejecutada. Es posible, por supuesto, incluir muchas más instrucciones SQL incrustadas que las que se usaron en este ejercicio,

y se pueden incluir otros tipos de instrucciones, por ejemplo UPDATE o DELETE. Sin embargo, el propósito de este ejercicio fue proporcionar una base para SQL incrustado. Para mayores detalles acerca de incrustar instrucciones SQL, deberá referirse a la documentación específica del lenguaje host y a la documentación de la implementación SQL aplicable.

Crear módulos cliente de SQL

Ahora que se tiene una comprensión básica sobre SQL incrustado, demos un vistazo a los módulos cliente de SQL. Los módulos cliente de SQL son colecciones autocontenidas de las instrucciones SQL. A diferencia de SQL incrustado, en el que las instrucciones SQL se insertan en el lenguaje de programación host, los módulos cliente de SQL se encuentran separados del lenguaje host. El lenguaje host contiene llamadas que invocan al módulo, que en su momento ejecuta las instrucciones SQL dentro de ese módulo.

Un módulo cliente de SQL consta de las propiedades que definen al módulo, las tablas temporales, los cursores y los procedimientos que contienen las instrucciones SQL. Cada procedimiento puede contener solamente una instrucción SQL. La siguiente sintaxis proporciona los elementos básicos de un módulo cliente de SQL:

```
MODULE <nombre del módulo> [ NAMES ARE <conjunto de caracteres> ]
LANGUAGE { ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI }
[ SCHEMA <nombre del esquema> ] [ AUTHORIZATION <identificador de autorización> ]
[ <instrucciones temporales de tabla> ] [ <instrucciones de cursor> ]
PROCEDURE <nombre del procedimiento> ( <instrucciones de parámetro> )
<instrucción SQL>;
```

Demos un vistazo a cada cláusula dentro de la sintaxis para que se pueda tener una mejor comprensión de todos los elementos que conforman un módulo cliente de SQL. La cláusula MODULE especifica un nombre para el módulo. Ésta es seguida por la cláusula opcional NAMES ARE, que se utiliza para especificar un conjunto de caracteres para los identificadores en el módulo. Si no se especifica la cláusula NAMES ARE, se utiliza el conjunto de caracteres preestablecido para la implementación SQL. El siguiente elemento en la sintaxis es la cláusula LANGUAGE, que especifica el lenguaje host que estará llamando al módulo. Siempre se debe especificar un lenguaje.

Después de haber definido la cláusula LANGUAGE, se debe definir una cláusula SCHEMA, una cláusula AUTHORIZATION, o ambas. La cláusula SCHEMA identifica el esquema predeterminado para que sea utilizado por las instrucciones SQL en el módulo. La cláusula AUTHORIZATION identifica al identificador de autorización para que sea utilizado para ejecutar las instrucciones dentro del módulo. Si no se especifica ninguna cláusula AUTHORIZATION, se asume el identificador de autorización actual.

También es posible declarar tanto tablas como cursores temporales dentro de un módulo. Las tablas temporales deben ser declaradas antes que cualquier cursor o procedimiento. Es posible declarar tantas tablas temporales como sea necesario. A diferencia de las instrucciones temporales de tablas, las instrucciones de cursor pueden ser mezcladas entre los procedimientos; sin embargo, una instrucción de cursor siempre debe preceder al procedimiento que hace referencia a ese cursor.

La porción final de la instrucción del módulo es el procedimiento. Como se mencionó anteriormente, el módulo puede contener uno o más procedimientos. Sin embargo, cada procedimiento

puede contener solamente una instrucción SQL y *debe* contener por lo menos una instrucción de parámetro, que es el parámetro de estatus SQLSTATE.

NOTA

A veces se hace referencia al procedimiento en un módulo cliente de SQL como un procedimiento invocado externamente.

El parámetro de estatus SQLSTATE proporciona una forma de reportar errores de vuelta al lenguaje host. Al igual que cualquier otro parámetro host, los valores pasan entre la base de datos SQL (el DBMS) y el programa host. En el caso de SQLSTATE, los valores están relacionados con el estatus de la ejecución de la instrucción SQL. Al incluir el parámetro SQLSTATE en los módulos, se le está permitiendo al programa host ver el estatus de la ejecución de la instrucción. Como resultado, el programa puede monitorear los errores y tomar las acciones apropiadas si esos errores ocurren.

Además del parámetro de estatus SQLSTATE, se deben declarar todos los demás parámetros host utilizados en la instrucción SQL del procedimiento. Los nombres de los parámetros (excepto SQLSTATE) deben estar precedidos por dos puntos tanto cuando son declarados como cuando son utilizados en la instrucción SQL. Como se puede ver en la sintaxis, las instrucciones de parámetro deben estar encerradas en paréntesis. Además, si se declara más de un parámetro, esas instrucciones deberán estar separadas por comas.

Definir módulos cliente de SQL

Ahora que hemos repasado la sintaxis para un módulo cliente de SQL, veamos un ejemplo de cómo crear uno. En la siguiente instrucción se creó un módulo que contiene un procedimiento:

```
MODULE CONSULTA_INVENTARIO_CD
LANGUAGE C
SCHEMA INVENTARIO AUTHORIZATION Ventas
PROCEDURE CONSULTA_1
( SQLSTATE, :p_ID_CD INT, :p_NOMBRE_CD VARCHAR(60) )
SELECT NOMBRE_CD
  INTO :p_NOMBRE_CD
  FROM INVENTARIO_CD
 WHERE ID_CD = :p_ID_CD;
```

Como se puede ver en este ejemplo, se está creando un modo llamado CONSULTA_INVENTARIO_CD. El módulo será llamado desde un programa C (LANGUAGE C). La instrucción SQL dentro del módulo accederá a una tabla en el esquema INVENTARIO y se ejecutará bajo el contexto del identificador de autorización Ventas. La instrucción MODULE incluye solamente un procedimiento, que es llamado QUERY_1. Si se definiera más de un procedimiento, cada uno de ellos estaría finalizado por punto y coma. Ahora demos un vistazo más cercano al procedimiento CONSULTA_1.

Lo primero que podrá notarse es que tres parámetros host han sido declarados. El parámetro SQLSTATE proporciona información de estatus al programa host. El parámetro p_ID_CD es un parámetro de entrada que recibirá un valor desde el programa host. El parámetro p_NOMBRE_CD es un parámetro de salida que tomará el valor arrojado por la instrucción SELECT y lo pasará al programa host. Observe que tanto el parámetro p_ID_CD como el parámetro p_NOMBRE_CD están precedidos por dos puntos y declarados con un tipo de datos. El parámetro SQLSTATE no requiere de punto y coma ni de ningún tipo de datos.

Una vez que se declaran los parámetros, es posible definir la instrucción `SELECT`. Como se puede ver, el parámetro de entrada es utilizado en la cláusula `WHERE`, y el parámetro de salida es utilizado en la cláusula `INTO`. El uso de los parámetros en esta forma permite al módulo interactuar con el programa host. Un valor para el parámetro de entrada es pasado al módulo cuando el módulo es convocado dentro del lenguaje host, y el parámetro de salida es enviado de vuelta al lenguaje host para ser utilizado por el programa según sea necesario.

NOTA

El proceso de convocar un módulo dentro de un programa host y pasar un parámetro a ese módulo es específico de cada lenguaje. Asegúrese de revisar la documentación para el lenguaje de programación específico y para la implementación SQL aplicable.

Como se puede ver, un módulo cliente de SQL puede ser una herramienta muy útil para desarrollar el componente SQL de una aplicación sin tener que incrustar las instrucciones SQL en el lenguaje host. Desafortunadamente, los módulos cliente de SQL no son soportados ampliamente en las implementaciones SQL, y si lo fueran, a menudo estarían no muy bien documentados. Sin embargo, si están ampliamente implementados o no se está volviendo irrelevante debido a que la industria se mueve cada vez más de SQL incrustado y los módulos cliente de SQL hacia los accesos de datos CLI o del tipo de CLI, que se cubren en la siguiente sección.

Utilizar una interfaz de nivel de llamada de SQL

Como se ha visto hasta ahora en este capítulo, un programa puede acceder a una base de datos SQL utilizando SQL incrustado y los módulos cliente SQL. En SQL incrustado, las instrucciones SQL se insertan directamente en el lenguaje de programación host. Para los módulos cliente de SQL, el programa host convoca los módulos que contienen instrucciones SQL ejecutables. Las instrucciones están separadas del lenguaje host. SQL proporciona aún otro método para acceder a los datos SQL desde dentro del lenguaje de programación (la interfaz de nivel de llamada), o CLI, por sus siglas en inglés (call-level interface).

Una CLI es una interfaz de programación de aplicaciones (API, por sus siglas en inglés) que soporta un conjunto de rutinas predefinidas que permiten que un lenguaje de programación se comunique con una base de datos SQL. El lenguaje de programación convoca las rutinas, que luego se conectan a la base de datos. Las rutinas acceden a los datos y a la información de estatus de la base de datos, según se requiera, y arrojan esa información al programa. La figura 17-3 proporciona la idea general de cómo una CLI permite que un programa se comunique con una base de datos SQL.

El programa invoca rutinas CLI a través del uso de funciones. Cuando se convoca una función, el programa debe especificar valores para los argumentos de la función. Estos valores definen qué acciones y cuáles datos serán accedidos. La función pasa los valores a la rutina designada, que actúa como una interfaz entre el programa y la base de datos SQL. La CLI, en efecto, oculta los detalles de acceder a la base de datos desde el programa, haciendo posible para el programa el acceso a las bases de datos en diferentes sistemas de administración.

Una de las implementaciones mejor conocidas del modelo CLI es la interfaz de programación Open Database Connectivity de Microsoft (ODBC), a pesar de que otros fabricantes han sacado al mercado productos del tipo CLI que soportan tipos similares de acceso a la base de datos. Además, nuevas generaciones de API de acceso de datos están ganando popularidad, por ejemplo, OLE-DB

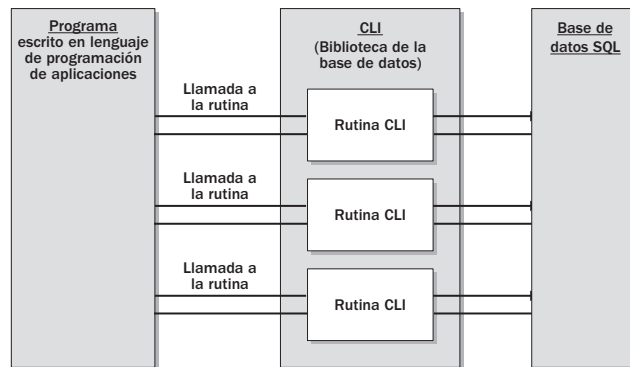


Figura 17-3 Utilizar una CLI para acceder a los datos en una base de datos SQL.

de Microsoft, que es más eficiente que ODBC y soporta acceso tanto a las fuentes de datos SQL como a otros tipos de fuentes de datos. Para los programadores de Java existe una API similar llamada JDBC. También encontrará que tales productos, como ActiveX Data Object (ADO), proporcionan una interfaz orientada a objetos entre los lenguajes de conjunto de instrucciones o lenguajes orientados a objetos y la API OLE-DB. Muchas herramientas de desarrollo también han hecho que acceder a una fuente de datos SQL sea más fácil que nunca. Por ejemplo, Visual Studio .NET permite construir aplicaciones de manejo de datos en lenguajes como Visual Basic, C++ y C#. Al utilizar las herramientas integradas ADO.NET se pueden crear aplicaciones que puedan acceder a una variedad de fuentes de datos, por ejemplo SQL Server y Oracle.

NOTA

A pesar de que Microsoft recibe todo el crédito por ODBC, fue desarrollado por Microsoft en asociación con Simba Technologies, y está basado en especificaciones CLI del SQL Access Group, X/Open (ahora parte de The Open Group) y del ISO/IEC.

La clave para todos estos productos es proporcionar un método uniforme de acceso a la base de datos desde dentro del lenguaje de programación. Las especificaciones CLI en SQL:2006 estandarizan la interfaz de acceso a la base de datos proporcionando un conjunto de funciones CLI predefinido que permite al programa conectarse a la base de datos, modificar y recuperar datos, pasar información desde y hacia la base de datos, y obtener información de estatus acerca de la ejecución de la instrucción. En esta sección se verán muchas funciones CLI y cómo pueden ser utilizadas en un lenguaje de programación para acceder a los datos SQL.

NOTA

A pesar de que el estándar SQL:2006 ha definido extensamente el modelo CLI, las aplicaciones pueden variar ampliamente en los métodos que utilizan para acceder a una fuente de datos. Como resultado, se encontrará que es necesario utilizar un método de acceso de datos que sea soportado en el ambiente seleccionado. Por ejemplo, si se está desarrollando una aplicación C que se conecta a una fuente de datos a través de ODBC, los datos específicos para el acceso de datos descritos en esta sección serán de mucha ayuda. Sin embargo, si se está desarrollando una aplicación C# o una aplicación de páginas de servidor activas (ASP,

por sus siglas en inglés) utilizando VBScript y se está conectando a la fuente de datos a través de ADO, se tendrá que referir a la documentación relacionada con esa tecnología en particular y también revisar la información en esta sección.

Asignar indicadores

El primer paso que se debe tomar cuando se accede a una base de datos a través de una interfaz CLI es establecer los indicadores de asignación necesarios. Un *indicador de asignación* es un objeto arrojado por la base de datos SQL cuando un recurso es asignado. El indicador es utilizado por el programa host para acceder a la base de datos. Se deben establecer tres tipos de indicadores de asignación en el programa host para poder acceder a los datos SQL desde dentro del programa:

- **Indicador de ambiente** Establece el ambiente en el cual todas las funciones CLI son convocadas y proporciona un contexto en el que se establecen uno o más indicadores de conexión.
- **Indicador de conexión** Establece un contexto de conexión a una base de datos SQL específica. El indicador de conexión debe establecerse dentro del contexto del indicador de ambiente. Un indicador de conexión no se conecta realmente a la base de datos. Simplemente proporciona el contexto para hacer posible esa conexión. Una vez que un indicador de conexión ha sido establecido, deberá utilizarse el contexto de ese indicador para hacer la conexión real a la base de datos.
- **Indicador de instrucción** Establece un contexto en el cual las instrucciones SQL pueden ser ejecutadas. Cualquier instrucción invocada a través de CLI debe ser ejecutada dentro del contexto de un indicador de instrucción, y el indicador de instrucción debe estar definido dentro del contexto de un indicador de conexión.

Para comprender mejor cómo operan los indicadores de asignación, demos un vistazo a la figura 17-4. Como se puede ver en la figura, dos indicadores de conexión se alojan dentro de un

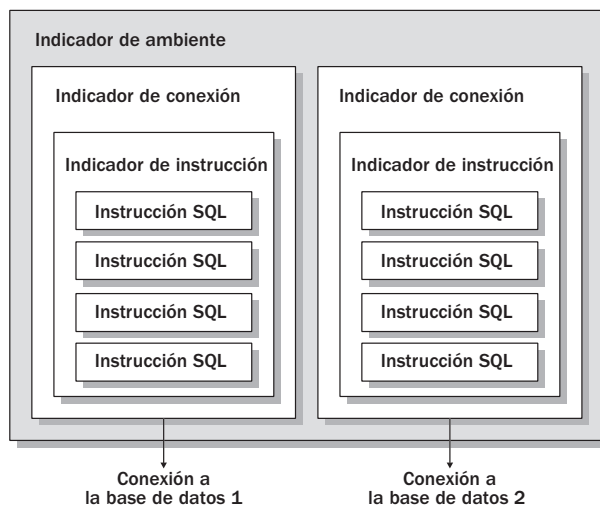


Figura 17-4 Establecer indicadores de asignación.

indicador de ambiente, y un indicador de ambiente se aloja dentro de cada indicador de conexión. Cada instrucción SQL es ejecutada dentro del contexto de un indicador de asignación.

Establecer un indicador de ambiente

Para establecer un indicador de ambiente en el cual soportar el acceso a la base de datos, se puede utilizar la función `AllocHandle()`, que requiere tres argumentos. El primer argumento (`SQL_HANDLE_ENV`) especifica el tipo de indicador (de ambiente) que se está asignando. El segundo argumento (`SQL_NULL_HANDLE`) indica que el indicador de ambiente no depende de ningún indicador existente. El tercer argumento es una variable host que identifica al indicador de ambiente. Cuando una variable host es utilizada en este contexto, está precedida del símbolo de unión `&` (ampersand, en inglés).

NOTA

Las variables host son declaradas de acuerdo con las convenciones del lenguaje host. Además, el programa host pudiera contener otros elementos que soporten la funcionalidad CLI. Por ejemplo, un programa C podría requerir archivos de inclusión especiales necesarios para interactuar con la interfaz de programación CLI. También es necesario observar que las bibliotecas del lenguaje pueden variar de fabricante a fabricante; por ejemplo, existen varias inconsistencias conocidas entre las bibliotecas C proporcionadas por Microsoft y Borland. Además, el programa host puede contener funciones especiales para manejo de errores que pueden monitorear el éxito o fracaso de una llamada de rutina CLI. Para información acerca de los elementos específicos de cada lenguaje que deberán estar incluidos en el programa host, asegúrese de revisar la documentación para el lenguaje específico, la interfaz de programación de aplicaciones CLI y la implementación SQL.

Ahora que se han repasado los elementos individuales necesarios para establecer un indicador de ambiente, demos un vistazo a la función `AllocHandle()` como si ya estuviera incluida en el programa C:

```
SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
```

Lo primero que notará es que la función `AllocHandle` está precedida por el prefijo SQL. En los programas C, el prefijo SQL es agregado a todas las funciones CLI. El prefijo puede variar de acuerdo con el lenguaje host. También se notará en esta instrucción que los tres argumentos están encerrados en paréntesis y separados por comas. Además, la variable host que identifica al indicador está precedida por el símbolo de unión `&`.

Establecer un indicador de conexión

Una vez que se ha establecido el indicador de ambiente, es posible establecer uno o más indicadores de conexión dentro del contexto de ese ambiente. Para hacerlo, se utilizará una vez más la función `AllocHandle()`, junto con los tres argumentos. El primer argumento (`SQL_HANDLE_DBC`) especifica el tipo de indicador (de conexión) que se está asignando. El segundo argumento identifica el ambiente en el cual la conexión está siendo establecida. (Ésta es la variable host identificada cuando se estableció el indicador de ambiente.) El tercer argumento es una variable host que identifica al indicador de conexión. Una vez más, éste está precedido por el símbolo de unión `&`, como se muestra en la siguiente instrucción de función:

```
SQLAllocHandle ( SQL_HANDLE_DBC, henv, &hdbc );
```

Después de establecer el indicador de conexión es necesario explícitamente conectarse a la base de datos dentro del contexto del indicador. Para hacer esto se debe utilizar la función `SQLConnect()`, que utiliza siete argumentos: el indicador de conexión, el servidor SQL objetivo, la longitud del nombre del servidor, el usuario de conexión, la longitud del nombre del usuario, la contraseña de conexión y la longitud de la contraseña. Para las cadenas `C` se puede utilizar `SQL_NTS` en lugar de los argumentos de longitud para indicar que una longitud no tiene que ser especificada por la cadena anterior, como se muestra en el siguiente ejemplo:

```
SQLConnect ( hdbc, servidor01, SQL_NTS, AdminVentas, SQL_NTS, CVentas,
SQL_NTS );
```

Como se puede ver, la función especifica al indicador de conexión `hdbc` y al servidor SQL `servidor01`. La conexión será establecida utilizando la cuenta de usuario `AdminVentas` y la contraseña `CVentas`. Se utiliza `SQL_NTS` en lugar de especificar la longitud de cualquiera de las cadenas.

Una vez que se está conectado con la base de datos, es posible crear indicadores de instrucción y ejecutar instrucciones SQL.

Establecer un indicador de instrucción

Para poder ejecutar una instrucción SQL desde dentro del programa host, se debe crear un indicador de instrucción dentro del contexto del indicador de conexión. Al igual que con otros tipos de indicadores, se puede utilizar la función `AllocHandle()` para establecer el indicador de instrucción.

Como se ha visto, la función `AllocHandle()` requiere tres argumentos. En el caso de un indicador de instrucción, esos argumentos son `SQL_HANDLE_STMT`, la variable `host` que identifica al indicador de conexión y la variable `host` que identifica al indicador de instrucción. La variable `host` que identifica al indicador de instrucción está precedida por el símbolo de unión `&`, como se muestra en el siguiente ejemplo:

```
SQLAllocHandle ( SQL_HANDLE_STMT, hdbc, &hstmt );
```

En esta función el indicador de conexión referenciado es `hdbc`, y la variable que identifica al indicador de instrucción es `hstmt`.

Ejecutar instrucciones SQL

Ahora que se han establecido los indicadores de asignación y la conexión a la base de datos, es posible establecer funciones que permitan ejecutar las instrucciones SQL. El modelo CLI soporta dos métodos que pueden utilizarse para ejecutar instrucciones SQL. El primero es la ejecución directa y el segundo es la preparación de la instrucción para su ejecución posterior.

Utilizar la función `ExecDirect()`

El primer método que puede utilizarse para ejecutar una instrucción SQL es la función `ExecDirect()`. Esta función utiliza tres argumentos. El primer argumento es el nombre del indicador de instrucción en cuyo contexto se estará ejecutando la instrucción. El segundo argumento es la instrucción SQL en sí, encerrada entre comillas. El tercer argumento es la longitud de la instrucción. En un programa `C`, generalmente se utiliza `SQL_NTS` para indicar que no es necesario especificar la longitud de la cadena.

Demos un vistazo a un ejemplo de la función `ExecDirect()` para demostrar cómo puede ser utilizada para ejecutar una instrucción SQL. La siguiente función hace referencia al indicador de instrucción `hstmt` y define una instrucción `DELETE`:

```
SQLExecDirect ( hstmt, "DELETE FROM INVENTARIO_CD WHERE ID_CD = 5731",  
SQL_NTS );
```

Como se puede ver, la instrucción SQL es pasada como un argumento a la rutina CLI. En este caso, cualquier fila con un valor `ID_CD` de 5731 será eliminada de la tabla `INVENTARIO_CD`. Observe que el valor `SQL_NTS` es utilizado para indicar que no es necesario especificar la longitud de la cadena (la instrucción SQL en sí).

Utilizar las funciones `Prepare()` y `Execute()`

Otro método que puede utilizarse para ejecutar una instrucción es preparar primero la instrucción y ejecutarla después. Este método puede utilizarse si es necesario ejecutar la instrucción más de una vez.

La primera función que se utiliza en este proceso de dos pasos es la función `Prepare()`, que requiere los mismos tres argumentos que la función `ExecDirect()`, como se muestra en el siguiente ejemplo:

```
SQLPrepare ( hstmt, "DELETE FROM INVENTARIO_CD WHERE ID_CD = 5731", SQL_NTS );
```

Observe que se hace referencia al mismo indicador de instrucción y se define la misma instrucción SQL que en el ejemplo anterior sobre `ExecDirect()`. La única diferencia es que, en el caso de la función `Prepare()`, la instrucción no es ejecutada en realidad, sino que solamente es preparada para su ejecución. Cuando una instrucción debe ser ejecutada múltiples veces, este proceso ayuda a evitar la sobrecarga de información debido a que la instrucción tiene que ser analizada y optimizada solamente una ocasión.

Una vez que se ha preparado la instrucción, puede entonces utilizarse la función `Execute()` para ejecutar la instrucción. La función `Execute()` solamente ocupa un argumento (el indicador de instrucción que contiene la instrucción preparada), como se muestra en el siguiente ejemplo:

```
SQLExecute ( hstmt );
```

Debido que la instrucción ya estaba preparada, simplemente se necesita hacer referencia al indicador de instrucción para ejecutar la instrucción. Puede ejecutarse la instrucción de manera tan frecuente como sea necesario simplemente al invocar la función `Execute()` y especificar el indicador de instrucción.

Trabajar con variables host

En los ejemplos anteriores, las instrucciones SQL que se han ejecutado eran relativamente sencillas debido a que no se utilizaron variables host en la instrucción. Sin embargo, si se planea pasar valores de variable host desde o hacia una instrucción SQL, se debe tomar un paso extra para enlazar esas variables host a la instrucción SQL.

Por ejemplo, supongamos que se quiere determinar una instrucción `DELETE` que tome una variable de entrada identificando a la fila que va a ser eliminada. La función `Prepare()` sería similar a la siguiente:

```
SQLPrepare ( hstmt, "DELETE FROM INVENTARIO_CD WHERE ID_CD = ?", SQL_NTS );
```

Observe que se utilizan las comillas para indicar la posición de la variable. Se utiliza el signo de interrogación en el lugar de la variable host. Esto a menudo se denomina como *marcador de posición*.

Una vez que se ha preparado la instrucción SQL, ahora debe enlazarse la variable host a la instrucción. Para hacer eso en un programa C, se debe utilizar una función `BindParameter()` que identifique al indicador de instrucción, la posición de la variable host dentro de la instrucción SQL, el nombre de la variable host y algunos otros argumentos, como se muestra en el siguiente ejemplo:

```
SQLBindParameter ( hstmt, 1, SQL_PARAMETER_MODE_IN, SQL_INT,
                  SQL_INT, 4, 0, &v_ID_CD, 4, &ind_ID_CD );
```

Como se puede ver, la función `BindParameter()` utiliza 10 argumentos. La tabla 17-2 lista los argumentos utilizados en el ejemplo anterior y proporciona una descripción para cada uno de esos argumentos.

Si se incluye más de una variable host en la instrucción SQL, deberá ser utilizada una instrucción de función `BindParameter()` para cada variable, y la posición (el segundo argumento) deberá ser incrementada en uno para cada variable adicional. Una vez que se enlazan las variables host a la instrucción SQL, es posible ejecutar la instrucción utilizando la función `Execute()`.

Argumento	Ejemplo	Descripción
1	hstmt	Identifica al indicador de instrucción que da el contexto para la ejecución de la instrucción SQL
2	1	Identifica la posición de la variable host en la instrucción SQL
3	SQL_PARAMETER_MODE_IN	Identifica si una variable host es una variable de entrada, de salida o de entrada/salida
4	SQL_INT	Identifica el tipo de datos del valor suministrado
5	SQL_INT	Identifica el tipo de datos de la variable host
6	4	Especifica el tamaño de columna de la variable host
7	0	Especifica el número de dígitos a la derecha del decimal requerido por la variable host
8	&v_CD_ID	Identifica el nombre de la variable host, como se declaró en el programa host
9	4	Especifica la longitud en octetos (bytes) de la variable host
10	&ind_CD_ID	Identifica el nombre de la variable de indicador, como se declaró en el programa host

Tabla 17-2 Argumentos utilizados en la función `BindParameter()`.

Recuperar datos SQL

Hasta este punto, las instrucciones SQL que se han ejecutado en el ambiente CLI no han arrojado ningún dato. Sin embargo, a menudo se encontrarán situaciones en las que el programa necesitará consultar la base de datos y procesar los valores que son arrojados por esa consulta. Como resultado, se necesitará algún tipo de mecanismo para enlazar el resultado de esa consulta a las variables que se declararon en el lenguaje host.

Por ejemplo, supongamos que se quiere ejecutar la siguiente instrucción SELECT:

```
SQLExecDirect ( hstmt, "SELECT NOMBRE_CD, EN_EXISTENCIA FROM
INVENTARIO_CD", SQL_NTS );
```

Como se puede ver, la instrucción arrojará una lista de valores NOMBRE_CD y valores EN_EXISTENCIA de la tabla INVENTARIO_CD. Para poder tratar con esos valores, deben ser enlazados a las variables host aplicables. Para hacer esto en un programa C, deberá utilizarse la función BindCol(). Esta función es un poco más simple que la función BindParameter() y solamente ocupa seis argumentos, como se muestra en el siguiente ejemplo:

```
SQLBindCol ( hstmt, 1, SQL_CHAR, &v_NOMBRE_CD, 60, &ind_NOMBRE_CD );
SQLBindCol ( hstmt, 2, SQL_INT, &v_EN_EXISTENCIA, 5, &ind_EN_EXISTENCIA );
```

La tabla 17-3 lista los argumentos utilizados en la primera instrucción de este ejemplo y proporciona una descripción de cada uno de esos argumentos.

Observe que se han definido dos instrucciones de función, una para cada columna recuperada por la instrucción SELECT. Se debe definir una instrucción de función por cada columna que esté listada en la cláusula SELECT de la instrucción SELECT. Una vez que se enlazan los valores de columna a las variables host, pueden utilizarse esas variables en el programa host para procesar los datos dentro del programa según sea necesario.

Argumento	Ejemplo	Descripción
1	hstmt	Identifica el indicador de instrucción que da el contexto para la ejecución de la instrucción SQL
2	1	Identifica la columna como está listada en la cláusula SELECT de la instrucción SELECT
3	SQL_CHAR	Identifica el tipo de datos de la variable host
4	&v_NOMBRE_CD	Identifica el nombre de la variable host, como se declaró en el programa host
5	60	Especifica la longitud en octetos de la variable host
6	&ind_NOMBRE_CD	Identifica el nombre de la variable de indicador, como fue declarada por el programa host

Tabla 17-3 Argumentos utilizados en la función BindCol().

Pruebe esto 17-2 Utilizar la interfaz de nivel de llamada de SQL

En el ejercicio anterior, Pruebe esto 17-1, se utilizó un programa de edición de texto para crear instrucciones SQL incrustadas. En este ejercicio se realizarán acciones similares, excepto que esta vez se definirán las funciones necesarias para realizar llamadas de la rutina CLI. Como parte de este proceso, se establecerán los indicadores de asignación necesarios, se creará una conexión a la base de datos, se establecerá la ejecución de la instrucción SQL, se enlazarán las variables host a las instrucciones SQL y se enlazará el resultado de la instrucción a las variables host. Las funciones CLI que se estarán utilizando serán aquellas típicamente utilizadas en un programa C. Tenga en mente, sin embargo, que el modelo CLI soporta muchas más funciones de las que hemos cubierto en este capítulo. Asegúrese de revisar la documentación apropiada para mayores detalles acerca de las funciones que sean diferentes a las descritas aquí. Se puede descargar el archivo `Try_This_17_2.txt` (en inglés), que contiene las instrucciones de función CLI utilizadas en este ejercicio.

Paso a paso

1. Abra un programa de edición de texto como el Bloc de notas de Microsoft.
2. El primer paso que debe tomarse es establecer un indicador de ambiente. Se utilizará la variable host `henv` para determinar el indicador. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
```

Observe que las funciones incluyen tres argumentos, encerrados en paréntesis y separados por comas. Observe también que se utiliza un signo de unión `&` para la variable host.

3. Ahora es posible establecer el ambiente de conexión. El ambiente de conexión se establecerá dentro del contexto del indicador de ambiente que se creó en el paso 2. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLAllocHandle ( SQL_HANDLE_DBC, henv, &hdbc );
```

Como se puede ver, la variable host `henv` se utiliza para mostrar el indicador de ambiente, y la variable host `hdbc` se utiliza para identificar al indicador de conexión.

4. Ahora que se ha establecido el indicador de conexión, es posible crear la conexión real. Para esta conexión se utilizará `ServidorBD` como el servidor SQL, `AdminBD` como la cuenta de usuario, y `CAdmin` como la contraseña para esa cuenta. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLConnect ( hdbc, ServidorBD, SQL_NTS, AdminBD, SQL_NTS, CAdmin, SQL_NTS );
```

Observe que la instrucción incluye el valor `SQL_NTS` para indicar que no es necesario especificar una longitud de cadena.

(continúa)

5. A continuación se establecerá el indicador de instrucción dentro del contexto de la conexión que se creó en el paso 3. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLAllocHandle ( SQL_HANDLE_STMT, hdbc, &hstmt );
```

Como se puede ver, la variable host hdbc es utilizada para identificar al indicador de conexión, y la variable hstmt es utilizada para identificar al indicador de instrucción.

6. Ahora que se han establecido los indicadores de asignación y se ha creado la conexión, todo está listo para ejecutar una instrucción SQL. Se utilizará la función ExecDirect() para especificar una instrucción DELETE. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLExecDirect ( hstmt, "DELETE FROM DISCOS_COMPACTOS  
WHERE ID_DISCO_COMPACTO = 122", SQL_NTS );
```

La instrucción DELETE está incluida como uno de los argumentos de la función. Observe que está encerrada entre comillas. Observe también que la instrucción está siendo preparada dentro del contexto de la variable host hstmt, que está asignada al ambiente de la instrucción.

7. En el paso anterior se ejecutó la instrucción SQL en un paso utilizando la función ExecDirect(). En este paso se preparará una instrucción SQL para su ejecución, pero en realidad se ejecutará hasta un paso posterior. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLPrepare ( hstmt, "SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_  
COMPACTOS  
WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
```

Observe que la cláusula WHERE de la instrucción SELECT incluye un signo de interrogación para indicar que un valor será pasado hacia la instrucción a través de una variable host.

8. Para poder ejecutar la instrucción del paso anterior será necesario enlazar la variable host a la instrucción. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLBindParameter ( hstmt, 1, SQL_PARAMETER_MODE_IN, SQL_INT,  
SQL_INT, 3, 0, &v_ID_CD, 4, &ind_ID_CD );
```

Como se puede ver, la variable host v_ID_CD está siendo enlazada a la instrucción SQL en el contexto del ambiente de la instrucción creado anteriormente. Debido a que sólo se hace referencia a una variable host en la instrucción SQL, solamente se requiere una instrucción de función BindParameter().

9. Ahora es posible ejecutar la instrucción preparada en el paso 7. Escriba la siguiente instrucción de función en el documento de texto:

```
SQLExecute ( hstmt );
```

La instrucción será ejecutada en el contexto del ambiente de instrucción hstmt.

10. A continuación se deben enlazar los resultados de la consulta con las variables host. Debido a que dos columnas son identificadas en la cláusula SELECT de la instrucción SELECT, se deben incluir dos instrucciones de función BindCol(). Escriba las siguientes instrucciones de función en el documento de texto:

```
SQLBindCol ( hstmt, 1, SQL_CHAR, &v_TITULO_CD, 60, &ind_TITULO_CD );  
SQLBindCol ( hstmt, 2, SQL_INT, &v_EN_EXISTENCIA, 5, &ind_EN_EXISTENCIA );
```

El programa C deberá ahora ser capaz de utilizar los valores arrojados por la instrucción `SELECT`. Si se revisa el documento que se ha creado, deberá contener el siguiente código:

```
SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
SQLAllocHandle ( SQL_HANDLE_DBC, henv, &hdbc );
SQLConnect ( hdbc, ServidorBD, SQL_NTS, AdminBD, SQL_NTS, CAdmin, SQL_
NTS );
SQLAllocHandle ( SQL_HANDLE_STMT, hdbc, &hstmt );
SQLExecDirect ( hstmt, "DELETE FROM DISCOS_COMPACTOS
WHERE ID_DISCO_COMPACTO = 122", SQL_NTS );
SQLPrepare ( hstmt, "SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_
COMPACTOS
WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
SQLBindParameter ( hstmt, 1, SQL_PARAMETER_MODE_IN, SQL_INT,
SQL_INT, 3, 0, &v_ID_CD, 4, &ind_ID_CD );
SQLExecute ( hstmt );
SQLBindCol ( hstmt, 1, SQL_CHAR, &v_TITULO_CD, 60, &ind_TITULO_CD );
SQLBindCol ( hstmt, 2, SQL_INT, &v_EN_EXISTENCIA, 5, &ind_EN_EXISTENCIA );
```

11. Guarde el archivo y cierre la aplicación.

Resumen de Pruebe esto

Este ejercicio introdujo las funciones básicas necesarias para utilizar CLI para acceder a una base de datos desde un programa host. Lo que no cubre el ejercicio es el código C real que proporcionaría la base para el programa. Por ejemplo, un programa C usualmente incluiría instrucciones de variable, archivos de inclusión, capacidades para manejo de errores, operaciones relacionadas con el usuario y lenguaje condicional que permita utilizar los valores arrojados por la base de datos SQL. Las funciones CLI que se cubrirían en este ejercicio usualmente serían dispersadas en el lenguaje host y trabajarían en conjunción con él. Aun con eso, este ejercicio deberá haberle ayudado a comprender los conceptos básicos involucrados al utilizar CLI, y debe haberlo preparado mejor para trabajar en el ambiente del lenguaje host cuando se intente acceder a datos SQL.

Autoexamen Capítulo 17

1. ¿Cuál método de acceso de datos deberá utilizarse si se quieren crear y ejecutar instrucciones SQL *ad hoc*?
 - A** CLI
 - B** De módulos cliente de SQL
 - C** Invocación directa
 - D** SQL incrustado

2. ¿Qué es SQL incrustado?
3. ¿Qué hace el precompilador con el archivo del programa?
4. ¿Cuáles archivos son creados por un precompilador SQL?
 - A Un archivo para las funciones CLI
 - B Un archivo para el lenguaje host
 - C Un archivo para las llamadas CLI
 - D Un archivo para las instrucciones SQL incrustadas
5. ¿Qué cláusula se utiliza en una instrucción MODULE para especificar el lenguaje de programación host?
6. ¿Qué prefijo deberán utilizar las instrucciones SQL incrustadas cuando esas instrucciones son incrustadas en el lenguaje de programación MUMPS?
 - A &SQL(
 - B EXEC SQL
 - C START-EXEC
 - D Las instrucciones incrustadas en MUMPS no requieren un prefijo
7. Un(a) _____ es un tipo de parámetro que es declarado dentro del lenguaje host y luego es referenciado dentro de la instrucción SQL incrustada.
8. ¿Qué instrucción deberá utilizarse al principio de la sección de instrucción para las variables host?
9. ¿Qué prefijo deberá proporcionarse para una variable host cuando esté incluida en una instrucción SQL?
 - A Signo de interrogación
 - B Signo de unión &
 - C Punto y coma
 - D Dos puntos
10. Se planea incrustar instrucciones SQL en el programa host. Se quiere declarar diferentes variables host para ser utilizadas en las instrucciones SQL. ¿Qué instrucción SQL deberá utilizarse para finalizar la sección de instrucción del programa?
 - A TERMINATE DECLARE SECTION
 - B END DECLARE SECTION
 - C TERMINATE DECLARATIONS
 - D END DECLARATIONS
11. ¿Qué puede provocar que ocurra una incongruencia en la impedancia cuando se pasa una variable de un programa host a una instrucción SQL?
12. ¿Qué tipo de instrucción SELECT puede utilizarse en SQL incrustado cuando se recupera solamente una fila de datos?

13. Un(a) _____ es un tipo de variable que especifica si una variable de datos asociada contiene un valor nulo.
14. ¿Qué instrucción puede utilizarse en SQL incrustado para proporcionarle al programa host la información de excepción y de advertencia?
 - A WHENEVER
 - B INTO
 - C CAST
 - D PROCEDURE
15. Un(a) _____ es una colección de instrucciones SQL autocontenidas que están separadas de un lenguaje de programación host, pero que pueden ser convocadas desde dentro de ese lenguaje.
16. ¿Qué indicadores de asignación deben establecerse para poder ejecutar una instrucción SQL a través de una interfaz de programación CLI?
17. ¿Cuántas instrucciones SQL pueden incluirse en un procedimiento en un módulo cliente de SQL?
18. ¿Qué función deberá utilizarse para establecer un indicador de conexión CLI?
 - A ExecDirect()
 - B Connect()
 - C Prepare()
 - D AllocHandle()
19. Se está asignando un indicador de ambiente dentro de un programa C y asignando el indicador con la variable host henv. ¿Qué instrucción de función deberá utilizarse?
20. Se está creando la siguiente instrucción de función Prepare() en el programa host:


```
SQLPrepare ( hstmt, "SELECT ID_CD, TITULO_CD, EN_EXISTENCIA FROM
DISCOS_COMPACTOS WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
```

¿Cuántas instrucciones de función BindCol() deberán crearse?

 - A Una
 - B Dos
 - C Tres
 - D Cuatro
21. ¿Qué función CLI deberá utilizarse si se quiere ejecutar una instrucción SQL en un solo paso?

Capítulo 18

Trabajar con datos XML

Habilidades y conceptos clave

- Aprender los conceptos básicos de XML
 - Aprender acerca de SQL/XML
-

El lenguaje de marcado extensible (XML, por sus siglas en inglés) es un lenguaje de marcado de propósito general utilizado para describir documentos en un formato conveniente para desplegarse en páginas Web y para intercambiar datos entre diferentes partes. Las especificaciones para almacenar datos XML en las bases de datos SQL fueron agregadas al estándar SQL en SQL:2003 como Part 14 (parte 14, discutiblemente la mejora más importante a esa versión del estándar). La parte 14, también conocida como SQL/XML, fue expandida en SQL:2006 y algunas correcciones de errores se publicaron en el 2007.

NOTA

SQL/XML es totalmente diferente a SQLXML de Microsoft, que es una tecnología propietaria utilizada en SQL Server. Como es fácil imaginar, el desafortunado parecido en los nombres ha causado gran confusión. Microsoft participó en los estándares que precedieron a SQL/XML, pero después escogió no implementarlo.

Aprender los conceptos básicos de XML

Para poder comprender SQL/XML, se deben comprender primero los conceptos básicos de XML. A pesar de que una explicación completa acerca de XML excede por mucho los alcances de este libro, este tema proporciona una breve explicación. Se puede encontrar mucha más información al buscar en Internet.

Es posible que usted ya esté familiarizado con HTML, el lenguaje de marcado utilizado para definir páginas Web. De ser así, la sintaxis de XML le será muy familiar. Esto se debe a que ambos están basados en el lenguaje de marcado generalizado estándar (SGML, por sus siglas en inglés), el cual está basado en el lenguaje de marcado generalizado (GML, por sus siglas en inglés), que fue desarrollado por IBM en los años sesenta. Un *lenguaje de marcado* es un conjunto de anotaciones, a menudo llamadas *etiquetas*, que se utilizan para describir cómo el texto será estructurado, formateado o acomodado. El texto etiquetado tiene la intención de ser leído por humanos. Una de las diferencias fundamentales entre HTML y XML es que HTML proporciona un conjunto predefinido de etiquetas, mientras que XML permite al autor crear sus propias etiquetas.

Demos un vistazo a un documento XML de ejemplo que contiene los resultados de una consulta SQL. La figura 18-1 muestra dos artistas de la tabla INTERPRETES y cinco de sus CD de la tabla INVENTARIO_CD. Como se aprendió en el capítulo 11, podemos unir fácilmente las dos tablas utilizando una instrucción SELECT de SQL como la siguiente:

```
SELECT a.NOMBRE_INTER, a.ID_INTER, b.NOMBRE_CD, b.EN_EXISTENCIA
FROM INTERPRETES a JOIN INVENTARIO_CD b
ON a.ID_INTER = b.ID_INTER
ORDER BY a.NOMBRE_INTER, b.NOMBRE_CD;
```

INTERPRETES		INVENTARIO_CD		
ID_INTER: INT	NOMBRE_INTER: VARCHAR(60)	NOMBRE_CD: VARCHAR(60)	ID_INTER: INT	EN_EXISTENCIA INT
101	Joni Mitchell	Both Sides Now	101	13
104	Bonnie Raitt	Blue	101	24
		Court and Spark	101	17
		Longing in Their Hearts	104	18
		Fundamental	104	22

Figura 18-1 Las tablas INTERPRETES e INVENTARIO_CD.

Observe que se utilizó la cláusula ORDER BY para especificar el orden de las filas en el conjunto de resultados. Los resultados de la consulta deberán lucir parecidos a éstos:

ID_INTER	NOMBRE_INTER	NOMBRE_CD	EN_EXISTENCIA
-----	-----	-----	-----
104	Bonnie Raitt	Fundamental	22
104	Bonnie Raitt	Longing in Their Hearts	18
101	Joni Mitchell	Blue	24
101	Joni Mitchell	Both Sides Now	11
101	Joni Mitchell	Court and Spark	17

Los resultados de la consulta están en un formato conveniente para desplegarse o imprimirse, pero no están en un formato que pudiera desplegarse fácilmente en una página Web o pasar a otra aplicación de computadora para procesamiento posterior. Una forma de hacer más fácil esto es convertir los resultados de la consulta a XML como se muestra aquí:

```
<artistas>
  <artista id="104">
    <nombre>Bonnie Raitt</nombre>
    <CD>
      <CD existencia="22"><nombre>Fundamental</nombre></CD>
      <CD existencia="18"><nombre>Longing in Their Hearts</
        nombre></CD>
    </CD>
  </artista>
  <artista id="101">
    <nombre>Joni Mitchell</nombre>
    <CD>
      <CD existencia="24"><nombre>Blue</nombre></CD>
      <CD existencia="11"><nombre>Both Sides Now</nombre></CD>
```

```
        <CD existencia="17"><nombre>Court and Spark</nombre></CD>
    </CD>
</artista>
<!-- Artistas adicionales disponibles pronto-->
</artistas>
```

Como se puede ver en el listado de código, las etiquetas están encerradas en corchetes angulares y cada etiqueta de inicio tiene una etiqueta de finalización correspondiente que es idéntica, excepto por una diagonal (/) que el nombre tiene enfrente. HTML utiliza una convención idéntica. Por ejemplo, la etiqueta <artistas> inicia la lista de los artistas, mientras que le etiqueta </artistas> la finaliza. Dentro de la lista de artistas, la información para cada uno de los artistas individuales comienza con la etiqueta <artista>, que incluye un valor de datos para el atributo o de identificación del artista, y finaliza con la etiqueta </artista>. Se acostumbra (y se considera la mejor práctica) nombrar una lista utilizando el plural del nombre de la etiqueta utilizado para cada uno de los elementos en la lista. Los comentarios pueden ser agregados utilizando una etiqueta especial que comienza con <!-- y finaliza con -->, como se muestra en la penúltima línea del ejemplo.

Tanto los elementos de datos como los valores de datos (por ejemplo aquellos que serían almacenados en una columna de tabla relacional) pueden ser codificados como pares de nombre y pares de valor en una de dos formas. La primera forma es utilizando un *atributo* XML al nombrar el atributo dentro de otra etiqueta, seguido por el signo de igual y el valor de datos encerrado en comillas dobles, como se hizo con los atributos de identificación y de existencias. La segunda forma es utilizando un *elemento* XML al crear una etiqueta separada para el elemento de datos con el valor de datos que quedó entre las etiquetas de inicio y de finalización, como se hizo con los atributos del nombre de artista y con los atributos del nombre de CD. La cuestión acerca de cuál forma utilizar ha sido tema de mucho debate entre los desarrolladores XML. Sin embargo, el consenso general es utilizar elementos en cualquier momento que el elemento de datos pudiera ser separado en elementos adicionales posteriormente, como al separar el nombre del artista en nombre y apellido, o al dividir un elemento único de datos que contiene una lista de nombres de artista de respaldo separados por comas en una lista de elementos. Una consideración adicional es cuando se quiere permitir al procesador XML ignorar los caracteres que no aparecen en un documento impreso, como sucedería para los atributos, pero no para los elementos.

Probablemente se notará que, a diferencia del conjunto de resultados de SQL, XML puede mostrar la jerarquía de los datos. En este caso, la lista de los CD grabados por cada artista es anidada dentro de la información acerca del artista. Se ha tratado de que las declaraciones XML hagan el anidado más obvio. Y aunque la sangría de las etiquetas anidadas es la mejor práctica, no resulta significativo debido a que los caracteres que no aparecen en un documento impreso entre las etiquetas son ignorados cuando se procesa el XML.

La codificación XML puede ser muy tediosa. Afortunadamente, existen herramientas disponibles para realizar la conversión entre XML y texto plano, y funciones SQL/XML (cubiertas posteriormente en este capítulo) para convertir los datos de una base de datos relacional (SQL) a XML. Durante un tiempo, las bases de datos especializadas para almacenar y recuperar XML fueron ganando popularidad, pero los fabricantes de bases de datos relacionales más importantes agregaron características para permitir que XML nativo sea almacenado directamente en sus bases de datos. Al mismo tiempo, el estándar SQL fue expandido para incluir provisiones de datos XML, como se discute en el resto de este capítulo.

Pregunta al experto

P: ¿Existe un estándar para el lenguaje XML por sí mismo?

R: Aun cuando ISO no publica actualmente un estándar para XML, ISO 8879 proporciona un estándar para el lenguaje de marcado generalizado estándar (SGML), y XML está basado en SGML. Más relevantemente, el consorcio World Wide Web Consortium (W3C) publica especificaciones XML que comprenden el estándar aceptado generalmente por toda la industria de TI.

P: Se mencionó que XML es una forma conveniente para que diferentes partes intercambien información. ¿Significa eso que dos compañías pueden intercambiar libremente datos sin tener que crear un elaborado software de interfaz debido a que ambos utilizan XML?

R: Bueno, no exactamente. XML sólo proporciona una forma estándar para formatear los datos. Para que una compañía interprete correctamente los datos XML que otra compañía le ha enviado, debe conocer los nombres y definiciones de las etiquetas que la otra compañía formateó para ellos, particularmente los elementos y atributos que contienen los datos. Afortunadamente, existen varios estándares de la industria que pueden ayudar. Por ejemplo, HR/XML proporciona un estándar para intercambiar datos sobre recursos humanos (HR), para que una compañía pueda, por ejemplo, enviar datos de sus empleados a una compañía que proporcione seguros médicos a esos empleados. En algunas industrias, XML está empezando a reemplazar en un estándar más antiguo conocido como EDI (intercambio de datos electrónicos, por sus siglas en inglés).

Aprender acerca de SQL/XML

Como ya se mencionó, XML es comúnmente utilizado para representar datos en las páginas Web, y a menudo esos datos vienen de bases de datos relacionales. Sin embargo, como se ha visto, los dos modelos son bastante diferentes en que los datos relacionales son almacenados en tablas donde ni la jerarquía ni la secuencia tienen ninguna importancia, mientras que XML está basado en árboles jerárquicos en los que el orden se considera importante. El término *bosque* es a menudo utilizado para referirse a una colección de estructuras de árbol de XML. XML es utilizado para las páginas Web debido a que su estructura coincide cercanamente con la estructura que sería utilizada para desplegar los mismos datos en HTML. De hecho, muchas páginas Web son una mezcla de HTML para sus porciones estáticas, y XML para los datos dinámicos. Quizá es esta implementación tan amplia lo que ha llevado a los fabricantes principales, incluyendo a Oracle e IBM, a soportar las extensiones XML.

SQL/XML puede ser dividido en tres partes principales: el tipo de datos XML, las funciones SQL/XML y las reglas de trazado de SQL/XML. Se cubre cada una de ellas como tema principal en el resto de este capítulo.

El tipo de datos XML

El tipo de datos XML es manejado, en general, de la misma manera que todos los otros tipos de datos que se habían discutido desde el capítulo 3. Aun cuando almacenar datos en el formato XML

directamente en la base de datos no es la única manera de utilizar SQL y XML juntos, es una forma muy simple de empezar, debido a que es una extensión lógica de las implementaciones anteriores donde los desarrolladores de SQL simplemente almacenaban el texto o XML en una columna definida con un tipo de datos de carácter general como CHARACTER VARYING (VARCHAR). Sin embargo, es mucho mejor comunicarle al DBMS que la columna contiene XML, y la forma particular en que XML está codificado, para que así DBMS pueda proporcionar características adicionales hechas a la medida para el formato XML.

La especificación para el tipo de datos XML tiene este formato general:

XML (<modificador de tipo> {(<modificador secundario de tipo>)})

El modificador de tipo es obligatorio y debe estar encerrado en un par de paréntesis como se muestra, mientras que el modificador secundario de tipo es opcional, y de hecho ni siquiera es soportado por todos los modificadores de tipo. El estándar no es específico acerca de cómo una implementación SQL particular deberá tratar a los diferentes tipos, pero algunas convenciones y reglas sintácticas sí han sido especificadas. Los modificadores de tipo válidos son:

- **DOCUMENT** El tipo DOCUMENT está diseñado para el almacenaje de documentos de texto que fueron formateados utilizando XML. En general, se espera que los valores de datos estén compuestos en caracteres legibles para los humanos como letras, números y símbolos como aparecerían en un documento de texto sin estructurar.
- **CONTENT** El tipo CONTENT está diseñado para datos más complejos que puedan incluir datos binarios como imágenes y archivos de sonido.
- **SEQUENCE** El tipo SEQUENCE está diseñado para los documentos XQuery, que a menudo son llamados secuencias XQuery. XQuery es un tema avanzado que se encuentra fuera de los alcances de este libro.

El modificador secundario de tipo, usado solamente con los modificadores de tipo primarios DOCUMENT y CONTENT, puede tener uno de estos valores:

- **UNTYPED** Los datos XML no tienen un tipo en particular.
- **ANY** Los datos XML son de cualquiera de los tipos soportados por la implementación SQL.
- **XMLSCHEMA** El tipo XMLSCHEMA se refiere a un esquema XML registrado que ha sido dado conocer al servidor de la base de datos. Los tres más comunes son:

Prefijo común	Identificador uniforme de recurso (URI) objetivo de espacio de nombre
Xs	http://www.w3.org/2001/XMLSchema
Xsi	http://www.w3.org/2001/XMLSchema-instance
Sqlxml	http://standards.iso.org/iso/9075/2003/sqlxml

Para las implementaciones SQL que no soportan el modificador secundario de tipo, se asume ANY de manera preestablecida.

NOTA

Debido a que SQL/XML es un estándar relativamente nuevo, el soporte para la implementación de los fabricantes varía. Oracle soporta un tipo de datos XMLType en lugar del tipo XML. DB2 UDB de IBM soporta un tipo XML, pero sin los modificadores de tipo. Como ya se mencionó, SQL Server de Microsoft soporta XML y un tipo de datos XML, pero de una forma un poco diferente del estándar SQL/XML. Al menos en la versión 5.0, MySQL no proporciona soporte para XML, pero existe una promesa de la directiva para una liberación futura.

Supongamos que se quiere agregar una biografía del artista a nuestra tabla de artistas que pueda ser desplegada en una página Web. Si los datos biográficos vinieran desde diferentes fuentes, y por lo tanto estuvieran formateados de forma diferente dependiendo de la fuente, XML sería una excelente forma de almacenar los datos en nuestra tabla de artistas. En el siguiente ejemplo se agregó la columna para la definición de la tabla ARTISTS que apareció en el capítulo 3:

```
CREATE TABLE ARTISTS
( ID_ARTISTA          INT,
  NOMBRE_ARTISTA      VARCHAR(60),
  FDN_ARTIST          DATE,
  POSTER_EN_EXISTENCIA  BOOLEAN,
  BIOGRAFIA_ARTISTA    XML(DOCUMENT(UNTYPED)) );
```

Funciones SQL/XML

Una función SQL/XML (también llamada función de valor XML) es simplemente una función que arroja un valor como un tipo XML. Por ejemplo, se puede escribir una consulta que seleccione los datos que no sean XML (esto es, los datos almacenados en tipos de datos diferentes a XML) y formatee los resultados de la consulta en un documento XML apropiado para desplegarse en una página Web o para la transmisión a alguna otra parte. La tabla 18-1 muestra las funciones básicas SQL/XML.

Existen más funciones que aquellas que están en el listado, y las funciones SQL/XML pueden ser utilizadas en combinación para formar consultas extremadamente poderosas (si no son complicadas). Además, las funciones disponibles varían entre las implementaciones SQL. Demos un vistazo a un ejemplo simple para clarificar cómo pueden ser utilizadas estas funciones. En este ejemplo, se enlistarán los CD de la artista Bonnie Raitt de las tablas INTERPRETES e INVENTARIO_CD mostradas en la figura 18-1. Aquí está la instrucción SQL, utilizando las funciones XMLELEMENT y XMLFOREST:

```
SELECT XMLELEMENT(NAME "CDArtista",
                  XMLFOREST(a.NOMBRE_INTER as Artista, a.ID_INTER, b.NOMBRE_CD,
b.EN_EXISTENCIA)
FROM INTERPRETES a JOIN INVENTARIO_CD b
ON a.ID_INTER = b.ID_INTER
WHERE a.ID_INTER = '104'
AND a.ID_INTER = b.ID_INTER
ORDER BY b.NOMBRE_CD;
```

Función	Valor arrojado
XMLAGG	Un valor XML único que contiene un bosque XML formado al combinar (agregar) una colección de filas que contienen cada una un valor XML único
XMLATTRIBUTE	Un atributo en la forma nombre=valor dentro de la función XMLELEMENT
XMLCOMMENT	Un comentario XML
XMLCONCAT	Una lista unida de valores XML, que crea un valor único que contiene un bosque XML
XMLDOCUMENT	Un valor XML que contiene un nodo único de documento
XMLELEMENT	Un valor XML que puede ser secundario de un nodo de documento, con el nombre especificado en el parámetro del nombre
XMLFOREST	Un elemento XML que contiene una secuencia de elementos XML formada a partir de las columnas de la tabla, que utiliza el nombre de cada columna como el nombre del elemento correspondiente
XMLPARSE	Un valor XML formado al analizar gramaticalmente la cadena suministrada sin validarla
XMLPI	Un valor XML que contiene una instrucción de procesamiento XML
XMLQUERY	El resultado de una expresión XQuery (XQuery es un sublenguaje utilizado para buscar XML almacenado en la base de datos; se encuentra fuera de los alcances de este libro)
XMLTEXT	Un valor XML que contiene un nodo único de texto, que puede ser secundario de un nodo de documento
XMLVALIDATE	Una secuencia XML que es el resultado de validar un valor XML

Tabla 18-1 Funciones SQL/XML.

Los resultados arrojados deberán lucir de forma parecida a éstos:

```

<CDArtista>
  <Artista>BonnieRait</Artista>
  <ID_INTER>104</ID_INTER>
  <NOMBRE_CD>Fundamental</NOMBRE_CD>
  <EN_EXISTENCIA>22</EN_EXISTENCIA>
</CDArtista>
<CDArtista>
  <Artista>BonnieRait</Artista>
  <ID_INTER>104</ID_INTER>
  <NOMBRE_CD>Longing in Their Hearts</NOMBRE_CD>
  <EN_EXISTENCIA>18</EN_EXISTENCIA>
</CDArtista>

```

Observe que los nombres de elemento XML son tomados de los nombres de las columnas, en mayúsculas y con guión bajo, como se acostumbra en SQL. Sin embargo, usando los alias de columna, como se hizo para la columna NOMBRE_INTER, es posible cambiar los nombres de columna a cualquier otro que se decida.

Reglas de trazado de SQL/XML

Hasta ahora no se ha analizado cómo se traducen y representan los valores SQL como valores XML y viceversa. El estándar SQL describe a detalle cómo los valores SQL pueden ser trazados hacia y desde valores XML. Este tema contiene un vistazo a las reglas de trazado de SQL/XML.

Trazados desde SQL hacia XML

Los trazados en este tema aplican para traducir datos de tipos de datos SQL hacia XML.

Trazar conjuntos de caracteres SQL a Unicode *Unicode* es un estándar de la industria que permite a los sistemas de computadora representar (codificar) consistentemente caracteres de texto expresados en la mayoría de los lenguajes escritos en el mundo. XML es codificado a menudo como caracteres Unicode para permitir su utilización en múltiples idiomas. Los caracteres SQL de datos se almacenan en cualquier conjunto de caracteres que sea especificado cuando es creada la tabla o la base de datos, y aunque muchas implementaciones de SQL soportan Unicode, también pueden ser utilizados muchos otros conjuntos de caracteres. El estándar SQL requiere que cada carácter en un conjunto de caracteres SQL tenga un trazado hacia un carácter Unicode equivalente.

Trazar identificadores SQL a nombres XML Es necesario definir un trazado de identificadores SQL, como los nombres de tabla y de columna, a nombres XML debido a que no todos los identificadores SQL son nombres aceptables en XML. Los caracteres que no son válidos en los nombres XML se convierten a una secuencia de dígitos hexadecimal derivada de la codificación Unicode del carácter, encerrados entre un guión bajo inicial y una x en minúscula, y un guión bajo final. Por ejemplo, el símbolo de dos puntos (:) sería traducido de un identificador SQL a un nombre XML como `_x003A_`.

Trazar tipos de datos SQL a tipos de datos de esquema XML Quizá ésta es la más complicada de las formas de trazado. Para cada tipo o dominio SQL, se le exige a la implementación SQL un trazado del tipo de esquema XML apropiado. El trazado detallado de los tipos SQL estándar a los tipos de datos de esquema XML es proporcionado en el estándar con suficiente detalle. Se encuentran resumidos en la tabla 18-2.

Valores de trazado de tipos de datos SQL a valores de tipos de datos de esquema XML

Para tipo o dominio SQL, con excepción de los tipos estructurados y los tipos de referencia, existe también un trazado de valores para el tipo hacia el espacio de valor del tipo de esquema XML correspondiente. Los valores nulos se representan utilizando ya sea ausencia (no utilizando el elemento) o utilizando la faceta `xsi:nil="true"` para determinar explícitamente el valor nulo.

Trazar una tabla SQL a un documento XML y un documento de esquema XML El estándar SQL define un trazado de una tabla SQL a uno o ambos de dos documentos: un documento de esquema XML que describe la estructura del XML trazado, y ya sea un documento XML o una secuencia de elementos XML. Este trazado aplica solamente a las tablas base y las tablas vistas, y solamente a las columnas visibles para la base de datos que el usuario puede trazar. La implementación puede proporcionar opciones para lo siguiente:

- Ya sea trazar la tabla a una secuencia de elementos XML o como un documento XML con un nombre de raíz único derivado del nombre de la tabla.
- La asignación de nombres objetivo del esquema XML a ser trazada.

Tipo SQL	Tipo de esquema XML	Notas
CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT	xs:string	Se utiliza la faceta XML xs:length para especificar la longitud para las cadenas de longitud fija. (Una <i>faceta</i> es un elemento utilizado para definir una propiedad de otro elemento.)
NUMERIC DECIMAL	xs:decimal	Se especifican la precisión y la escala utilizando facetas XML xs:precision y xs:scale.
INTEGER SMALLINT BIGINT	xs:integer	Este trazado se enlista como definido por la implementación, lo que significa que es opcional.
FLOAT REAL DOUBLE PRECISION	xs:float, xs:double	Se utiliza xs:float para precisiones de hasta 24 dígitos binarios (bits) y un exponente entre -149 y 104 inclusive; de otra manera, se utiliza xs_double.
BOOLEAN	xs:Boolean	
DATE	xs:date	Se utiliza la faceta xs:pattern para descartar la posibilidad de inexactitud en la zona horaria.
TIME WITH TIME ZONE TIME WITHOUT TIME ZONE	xs:time	Se utiliza la faceta xs:pattern para descartar la posibilidad de inexactitud en la zona horaria o para especificar la zona horaria, según sea el caso.
TIMESTAMP WITH TIME ZONE; TIMESTAMP WITHOUT TIME ZONE	xs:dateTime	Se utiliza la faceta xs:pattern para descartar la posibilidad de inexactitud en la zona horaria o para especificar la zona horaria, según sea el caso.
Tipos de intervalo	xdt:yearMonthDuration, xdt:day-TimeDuration	
Tipo de fila	Tipo complejo de esquema XML	El documento XML contiene un elemento para cada campo del tipo de fila SQL.
Dominio	Tipo de datos de esquema XML	El tipo de datos del dominio es trazado a XML con una anotación que identifica el nombre del dominio.
Tipo distinto de SQL	Tipo simple de esquema XML	
Tipo de colección SQL	Tipo complejo de esquema XML	El tipo complejo tiene un elemento único llamado <i>elemento</i> .
Tipo XML	Tipo complejo de esquema XML	

Tabla 18-2 Trazado de los tipos de datos SQL a los tipos de esquema XML.

- Ya sea trazar los valores nulos como elementos ausentes o elementos marcados con la faceta `xsi:nil="true"`.
- Ya sea trazar la tabla hacia datos XML, un documento de esquema de XML o ambos.

Trazar un esquema SQL a un documento XML y un documento de esquema XML El estándar SQL define el trazado entre las tablas de un esquema SQL y ya sea un documento XML que represente los datos en las tablas, un documento de esquema XML, o ambos. Solamente las tablas y columnas visibles para el usuario de la base de datos pueden ser trazadas. La implementación puede proporcionar opciones para lo siguiente:

- Ya sea trazar cada tabla como una secuencia de elementos XML o como un documento XML con un nombre raíz único derivado del nombre de la tabla.
- La asignación de nombres objetivo del esquema XML que será trazada.
- Ya sea trazar los valores nulos como elementos ausentes o elementos marcados con la faceta `xsi:nil="true"`.
- Ya sea trazar el esquema hacia datos XML, un documento de esquema de XML, o ambos.

Trazar un catálogo SQL a un documento XML y un documento de esquema XML El estándar SQL define el trazado entre las tablas de un catálogo SQL y ya sea un documento XML que represente los datos en las tablas del catálogo, un documento de esquema XML, o ambos. Sin embargo, esta parte del estándar no especifica ninguna sintaxis para invocar tal trazado debido a que está diseñada para ser utilizada por las aplicaciones o referenciada por otros estándares. Solamente los esquemas visibles para el usuario SQL pueden ser trazados. La implementación puede proporcionar opciones para lo siguiente:

- Ya sea trazar cada tabla como una secuencia de elementos XML o como un documento XML con un nombre raíz único derivado del nombre de la tabla.
- La asignación de nombres objetivo del esquema XML que habrá de trazarse.
- Ya sea trazar los valores nulos como elementos ausentes o elementos marcados con la faceta `xsi:nil="true"`.
- Ya sea trazar el catálogo hacia datos XML, un documento de esquema de XML, o ambos.

Trazados desde XML hacia SQL

El tema contiene dos trazados desde XML hacia SQL.

Trazar Unicode a conjuntos de caracteres SQL Al igual que con el trazado de conjuntos de caracteres SQL a Unicode, el estándar SQL requiere que exista un trazado definido por la implementación de caracteres Unicode a los caracteres en cada uno de los conjuntos de caracteres SQL soportados por la implementación SQL.

Trazar nombres y XML a identificadores SQL Éste es lo contrario de canalizar identificadores SQL a nombres de XML donde los caracteres que fueron convertidos debido a que no eran válidos en los nombres XML son convertidos de regreso a su forma original. Por lo tanto, si un símbolo de dos puntos en un identificador SQL fue convertido a `_x003A_` cuando se tradujo el identificador SQL hacia XML, sería convertido de regreso a dos puntos cuando el proceso fuera revertido. El estándar SQL recomienda ampliamente que la implementación SQL utilice un algoritmo simple para la traducción en ambas direcciones.

Pruebe esto 18-1 Utilizar funciones SQL/XML

En este ejercicio se utilizarán las funciones XML para seleccionar datos formateados XML de la base de datos INVENTARIO. Obviamente, la implementación SQL tiene que proporcionar soporte para XML para que se pueda completar este ejercicio, y como siempre, es posible que se tenga que modificar el código incluido en este ejercicio para poder ejecutarlo en el DBMS. Se puede descargar el archivo Try_This_18.txt (en inglés), que contiene las declaraciones SQL utilizadas en este ejercicio.

Paso a paso

1. Abra la aplicación de cliente para su RDBMS y conéctese con la base de datos INVENTARIO.
2. Se va a crear una consulta SQL que utilice las tres funciones SQL/XML para formatear XML que contenga un elemento para cada CD en la tabla DISCOS_COMPACTOS, con cada elemento incluyendo la ID del CD, seguido por un elemento separado que contenga el título del CD. Ingrese y ejecute la siguiente instrucción SQL:

```
SELECT XMLELEMENT(NOMBRE CD,
                  XMLATTRIBUTE(ID_DISCO_COMPACTO AS ID),
                  XMLFOREST(TITULO_CD as Titulo))
FROM DISCOS_COMPACTOS
ORDER BY ID_DISCO_COMPACTO;
```

3. El resultado producido deberá lucir parecido al siguiente listado. Observe que por cuestiones de espacio, solamente se muestran los primeros dos CD.

```
<CD ID='101'>
  <Titulo>Famous Blue Raincoat</Titulo>
</CD>
<CD ID='102'>
  <Titulo>Blue</Titulo>
</CD>
```

4. Cierre la aplicación de cliente.

Resumen de Pruebe esto

En este ejercicio, la instrucción SELECT de SQL utiliza tres funciones SQL/XML para formatear datos de la tabla DISCOS_COMPACTOS a XML. La función XMLELEMENT fue utilizada para crear un elemento para cada CD. La función XMLATTRIBUTE fue utilizada para incluir el valor ID_DISCO_COMPACTO con el nombre de ID como un valor dentro del elemento CD. Finalmente, la función XMLFOREST fue utilizada para crear un elemento para la columna TITULO_CD.

Autoexamen Capítulo 18

1. ¿Qué es XML?
2. ¿Cuáles de los siguientes son usos comunes de XML?
 - A Desplegar datos de la base de datos en una página Web
 - B La creación de páginas Web estáticas
 - C La transmisión de datos de una base de datos a otra parte
 - D El reforzamiento de reglas de negocios en documentos
3. ¿Qué tanto varían las bases de datos SQL y los documentos XML en términos de estructura de datos?
4. Si dos organizaciones están usando XML, ¿significa eso que ellos tienen una forma estándar de intercambiar datos sin tener que crear software de interfaz?
5. ¿Cuáles de los siguientes son modificadores de tipo válidos para el tipo de datos XML?
 - A DOCUMENT
 - B SEQUENCE
 - C SQLXML
 - D CONTENT
6. ¿Cuáles son los modificadores secundarios de tipo válidos para el modificador de tipo SEQUENCE?
7. ¿Cuál de las siguientes funciones SQL/XML crea un elemento basado en una columna de una tabla?
 - A XMLQUERY
 - B XMLELEMENT
 - C XMLFOREST
 - D XMLDOCUMENT
 - E XMLPARSE
8. ¿Cuál tipo de esquema XML es trazado desde el tipo de datos SQL NUMERIC?
 - A xs:integer
 - B xs:float
 - C xs:decimal
 - D xs:double

- 9.** ¿Cuál tipo de esquema XML es trazado desde el tipo de datos SQL DATE?
- A** xs:dateTime
 - B** xdt:yearMonthDuration
 - C** xs:time
 - D** xs:date
 - E** xdt:dat-TimeDuration
- 10.** ¿Cuáles son las dos formas en que los valores nulos de la base de datos pueden ser representados por SQL/XML?
- A** Absent document
 - B** Absent element
 - C** xsi:null= "true"
 - D** xsi:nil= "true"
 - E** <elementname=nil>

Parte IV

Apéndices



Apéndice A

Respuestas a los
autoexámenes



Capítulo 1: Introducción a las bases de datos relacionales y a SQL

1. ¿Qué es una base de datos?

Una base de datos es una colección de datos organizados en un formato estructurado definido por los metadatos que describen la estructura.

2. ¿Cuál de los siguientes objetos conforma una relación?

- A** Tipos de datos
- B** Tuplas
- C** Atributos
- D** Formas

Las respuestas correctas son **B** y **C**.

3. Un(a) _____ es un conjunto de datos cuyos valores conforman una instancia de cada uno de los atributos definidos para esa relación.

Tupla

4. ¿Cuáles son las diferencias entre la primera forma normal y la segunda forma normal?

De acuerdo con la primera forma normal, cada atributo de una tupla debe contener solamente un valor, cada tupla en una relación debe contener el mismo número de valores, y cada tupla en una relación debe ser diferente. De acuerdo con la segunda forma normal, una relación debe estar en la primera forma normal y todos los atributos en una relación deben ser dependientes de toda la clave de candidato.

5. Una relación está en la tercera forma normal si está en la segunda forma normal y si cumple con las demás directrices de esa forma. ¿Cuáles son esas directrices?

Todos los atributos sin clave deben ser independientes entre sí y dependientes de la clave.

6. ¿Cuáles son los tres principales tipos de relaciones soportados por una base de datos relacional?

Una a una, una a varias, varias a varias.

7. En el modelo de datos hay dos relaciones asociadas cada una entre sí por una relación varias a varias. ¿Cómo se implementa físicamente esta relación en una base de datos relacional?

La relación será implementada agregando una tercera relación entre las dos relaciones originales para poder crear dos relaciones uno a muchos.

8. ¿Cómo se diferencia SQL de los lenguajes de programación como C, COBOL y Java?

Los lenguajes de programación como C, COBOL y Java son lenguajes de procedimiento que definen cómo deben ser realizadas las operaciones de una aplicación y el orden en que son realizadas. Sin embargo, SQL no es un lenguaje de procedimiento y está más relacionado con los resultados de una operación; el ambiente subyacente de software determina cómo las operaciones serán procesadas. Sin embargo, SQL sí soporta algunas funcionalidades de procedimiento.

9. ¿Qué factores contribuyeron a que el estándar SQL:2006 incorporara capacidades orientadas a objetos?

La aparición de programación orientada a objetos, avances en tecnologías de hardware y software, y las crecientes complejidades de las aplicaciones.

10. ¿Qué nivel de conformidad debe soportar un RDBMS para poder cumplir con SQL:2006?

- A** Entrada
- B** Core
- C** Pleno
- D** Intermedio

La respuesta correcta es **B**.

11. ¿Cuáles son las diferencias entre las instrucciones DDL y DML?

Las instrucciones DDL se utilizan para crear, modificar y eliminar objetos de la base de datos como tablas, vistas, esquemas, dominios, activadores y procedimientos almacenados. Las instrucciones DML se utilizan para ver, agregar, modificar o eliminar datos almacenados en los objetos de la base de datos.

12. ¿Qué método de ejecución de instrucciones SQL usaría si deseara comunicarse directamente con una base de datos SQL desde una aplicación de usuario?

Invocación directa.

13. ¿Cuáles son los cuatro métodos que soporta el estándar SQL:2006 para la ejecución de las instrucciones SQL?

Invocación directa, SQL incrustado, unión de módulos y CLI.

14. ¿Qué es un sistema de gestión de base de datos relacional?

Un RDBMS (por sus siglas en inglés) es un programa o conjunto de programas que almacenan, administran, recuperan, modifican y manipulan los datos en una o más bases de datos relacionales.

15. ¿Cuál es un ejemplo de un RDBMS?

Cualquiera de los siguientes son ejemplos de RDBMS: DB2, MySQL, Oracle, SQL Server, PostgreSQL, Sybase, Informix, Ocelot o cualquier otro RDBMS en el mercado.

Capítulo 2: Trabajo con el entorno SQL

1. ¿Cuáles son las diferencias entre un agente SQL y una implementación SQL?

Un agente SQL es cualquier estructura que provoque que las instrucciones SQL sean ejecutadas. El agente SQL está ligado al cliente SQL dentro de la implementación SQL. Una implementación SQL es un procesador que ejecuta instrucciones SQL de acuerdo con los requerimientos del agente SQL. La implementación SQL incluye un cliente SQL y uno o más servidores SQL. El cliente SQL establece las conexiones SQL con los servidores SQL y mantiene los datos relacionados a las interacciones con el agente SQL y los servidores SQL. Un servidor SQL administra la sesión SQL que toma lugar sobre la conexión SQL y ejecuta las instrucciones SQL recibidas desde el cliente SQL.

2. ¿Qué componente del entorno SQL representa a un usuario o rol que concede privilegios específicos para acceder a los objetos y datos?

- A** Catálogo
- B** Identificador de autorización
- C** Módulo de cliente SQL
- D** Agente SQL

La respuesta correcta es **B**.

3. Un(a) _____ es una colección de esquemas que forman un nombre dentro del entorno SQL.

Catálogo

4. ¿Qué es un esquema?

Un *esquema* es un conjunto de objetos relacionados que son recolectados bajo una asignación común. El esquema actúa como un contenedor para esos objetos, los cuales a su vez almacenan los datos SQL o realizan otras funciones relacionadas con los datos.

5. ¿Qué instrucción se utiliza para agregar un esquema en el entorno SQL?

- A** ADD SCHEMA
- B** INSERT SCHEMA
- C** CREATE SCHEMA

La respuesta correcta es **C**.

6. ¿Cuál es el nombre del esquema que contiene las definiciones para los objetos de esquema en un catálogo?

INFORMATION_SCHEMA

7. ¿Cuáles son los 11 tipos de objetos de esquema que pueden estar contenidos en un esquema?

Tablas base, vistas, dominios, UDT, restricciones, módulos de servidor de SQL, activadores, rutinas invocadas por SQL, conjuntos de caracteres, cotejos y traducciones.

8. ¿Qué es una vista?

Una vista es una tabla virtual que es creada cuando se invoca la vista (al convocar su nombre). La tabla no existe en realidad, solamente la instrucción SQL que define a la tabla.

9. ¿Cuáles objetos de esquema proporcionan la unidad básica de gestión de datos en el entorno SQL?

- A** Vistas
- B** Dominios
- C** Tablas base
- D** Conjuntos de caracteres

La respuesta correcta es **C**.

10. ¿Cómo define el estándar SQL:2006 a una base de datos?

El estándar SQL:2006 no define una base de datos.

11. Un(a) _____ es el nombre dado a un objeto SQL.

Identificador.

12. ¿Cómo se distingue un identificador regular de un identificador delimitado en una instrucción SQL?

Un identificador delimitado está encerrado entre comillas, mientras que un identificador regular no.

13. ¿Qué tipo de identificador permite que se utilicen espacios como parte del nombre de un objeto?

Un identificador delimitado.

14. El entorno SQL incluye un catálogo denominado INVENTARIO. En ese catálogo se encuentra el esquema denominado DISCOS_COMPACTOS, y en ese esquema se encuentra una tabla denominada ARTISTAS. ¿Cuál es el nombre cualificado de esa tabla?

INVENTARIO.DISCOS_COMPACTOS.ARTISTAS

15. ¿Cuáles son las tres formas que puede tomar el componente <cláusula de nombre> de una instrucción CREATE SCHEMA?

Una <cláusula de nombre> en una instrucción CREATE SCHEMA puede tomar cualquiera de las siguientes tres formas:

<nombre del esquema>

AUTHORIZATION <identificador de autorización>

<nombre del esquema> AUTHORIZATION <identificador de autorización>

16. ¿Cuáles son las diferencias entre la opción CASCADE y la opción RESTRICT en la instrucción DROP SCHEMA?

Si se especifica la opción CASCADE, todos los objetos de esquema y los datos SQL dentro de esos objetos son eliminados del sistema. Si se utiliza la opción RESTRICT, el esquema es eliminado solamente si no existe ningún objeto de esquema.

17. Dentro de la jerarquía del entorno SQL, ¿cómo está relacionado un dominio con un catálogo?

Un dominio es un objeto de esquema, que es un objeto o hijo del esquema. El esquema es un objeto secundario de un catálogo.

18. ¿Qué tipo de identificador permite utilizar una palabra clave reservada?

Un identificador delimitado.

Capítulo 3: Creación y modificación de tablas

1. ¿Qué tipos de tablas base se pueden crear utilizando una instrucción CREATE TABLE?

A Tablas base persistentes

B Tablas base temporales globales

- C** Tablas temporales locales creadas
- D** Tablas temporales locales declaradas

Las respuestas correctas son **A**, **B** y **C**.

2. ¿Cuál es la principal diferencia entre una tabla temporal global y una tabla temporal local creada?

Una tabla temporal global puede ser accedida desde cualquier lugar dentro de la sesión SQL asociada, mientras que una tabla temporal local creada puede ser accedida solamente dentro del módulo asociado.

3. Está creando una tabla llamada AGENTES. La tabla incluye la columna ID_AGENTE, que tiene un tipo de datos INT, y la columna NOMBRE_AGENTE, que tiene un tipo de datos CHAR(60). ¿Qué instrucción SQL utilizaría?

Deberá utilizarse la siguiente instrucción SQL:

```
CREATE TABLE AGENTES
( ID_AGENTE INT,
  NOMBRE_AGENTE CHAR(60) );
```

4. ¿Cuáles son los tres tipos de datos que soporta SQL?

Predefinidos, construidos y definidos por el usuario.

5. ¿Cuáles son los cuatro tipos de datos de cadena?

Cadenas de caracteres, cadenas de caracteres nacionales, cadenas de bits y cadenas binarias.

6. Un(a) _____ es un tipo de datos que permite valores que se basan en bits de datos, en lugar de conjuntos de caracteres o cotejos. Este tipo de datos permite sólo valores de 0 y 1.

Cadenas de bits

7. ¿Cuál es la precisión y la escala del número 5293.472?

La precisión es 7 y la escala es 3.

8. ¿Cuáles son las diferencias entre los tipos de datos numéricos exactos y los tipos de datos numéricos aproximados?

Con los tipos de datos numéricos exactos, los valores permitidos tienen una precisión y escala. Con los tipos de datos numéricos aproximados, los valores permitidos tienen una precisión pero no escala.

9. ¿Cuáles tipos de datos son tipos de datos numéricos exactos?

- A** DOBLE PRECISION
- B** DECIMAL
- C** REAL
- D** SMALLINT

Las respuestas correctas son **B** y **D**.

- 10.** Un tipo de datos _____ especifica los valores de una fecha por año, mes y día.

DATE

- 11.** ¿Cuáles son las dos formas de tipos de datos de intervalo que soporta SQL?

Intervalos año-mes e intervalos día-hora.

- 12.** ¿Qué tipo de datos debe utilizarse para soportar una construcción verdadero/falso que pueda ser utilizada para comparar valores?

BOOLEAN

- 13.** Está creando un tipo definido por el usuario distinto llamado CIUDAD. El tipo de usuario se basa en el tipo de datos CHAR(40). ¿Qué instrucción SQL utilizaría?

Deberá utilizarse la siguiente instrucción SQL:

```
CREATE TYPE CIUDAD AS CHAR(40)
FINAL;
```

- 14.** Se crea una tabla llamada CLIENTES. La tabla incluye la columna NOMBRE_CLIENTE y la columna CIUDAD_CLIENTE. Ambas columnas tienen un tipo de datos VARCHAR(60). La columna CIUDAD_CLIENTE también tiene el valor predeterminado *Seattle*. ¿Qué instrucción SQL utilizaría?

Deberá utilizarse la siguiente instrucción SQL:

```
CREATE TABLE CLIENTES
( NOMBRE_CLIENTE VARCHAR(60),
  CIUDAD_CLIENTE VARCHAR(60) DEFAULT 'Seattle' );
```

- 15.** ¿Qué instrucción SQL deberá utilizarse para eliminar una columna de una tabla existente?

ALTER TABLE

- 16.** ¿Qué instrucción SQL deberá utilizarse para eliminar la definición de una tabla y todos los datos de SQL de una base de datos?

DROP TABLE

- 17.** Una base de datos incluye una tabla llamada CANTANTES_OPERA. Se quiere agregar una columna llamada NACIONALIDAD a esa tabla. La columna debe tener el tipo de datos VARCHAR(40). ¿Qué instrucción SQL utilizaría?

Deberá utilizarse la siguiente instrucción SQL:

```
ALTER TABLE CANTANTES_OPERA
ADD COLUMN NACIONALIDAD VARCHAR(40);
```

- 18.** Se desea eliminar la definición de la tabla CANTANTES_OPERA de la base de datos. También se quiere eliminar todos los datos y cualquier dependencia de la tabla. ¿Qué instrucción SQL utilizaría?

Deberá utilizarse la siguiente instrucción SQL:

```
DROP TABLE CANTANTES_OPERA CASCADE;
```

Capítulo 4: Implementación de la integridad de datos

1. ¿Cuáles son las tres categorías de las restricciones de integridad?

Restricciones relacionadas con la tabla (a veces simplemente denominadas restricciones de tabla), afirmaciones y restricciones de dominio.

2. ¿Cuáles son las diferencias entre una restricción de columna y una restricción de tabla?

Ambos tipos de restricciones se encuentran definidos en la definición de la tabla. Una restricción de columna se incluye con la definición de la columna, y una restricción de tabla se incluye como un elemento de la tabla, similar a la forma en que las columnas son definidas como elementos de la tabla.

3. ¿Qué tipos de restricciones se pueden incluir en una definición de columna?

NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY y CHECK.

4. ¿Cuál es la diferencia entre una restricción de tabla y una afirmación?

Una restricción de tabla está definida dentro de una definición de tabla y aplica solamente a esa tabla. Una afirmación es un tipo de restricción que está definida dentro de una definición de afirmación (separada de la definición de la tabla). Una afirmación puede estar asociada con una o más tablas.

5. ¿Qué significa un valor nulo?

Un valor nulo es aquel que es desconocido o no definido. Esto no es lo mismo que un cero, un espacio en blanco o un valor por defecto. En su lugar, éste indica que un valor de datos está ausente.

6. ¿Cuál de los siguientes tipos de restricciones soporta restricciones NOT NULL?

- A** Restricciones de tabla
- B** Restricciones de columna
- C** Restricciones de dominio
- D** Afirmaciones

La respuesta correcta es **B**.

7. Se crea una tabla que incluye una columna que acepta valores nulos, pero cuyos valores no nulos deben ser únicos. ¿Qué tipo de restricción se debe utilizar?

UNIQUE.

8. Se crea una tabla que incluye la columna NOMBRE_TIPO. La columna se define con el tipo de datos CHAR(10) y requiere una restricción UNIQUE, que se define como una restricción de columna. ¿Qué código SQL debe utilizarse para la definición de la columna?

Deberá utilizarse el siguiente código:

```
TYPE_NAME CHAR(10) UNIQUE
```

9. ¿Cuáles dos restricciones se aplican a las restricciones PRIMARY KEY pero no aplican a las restricciones UNIQUE?

Una columna que es definida con una restricción PRIMARY KEY no puede contener valores nulos, y solamente una restricción PRIMARY KEY puede ser definida para cada tabla.

- 10.** Se crea una restricción de PRIMARY KEY llamada PK_TIPOS_MUSICA_ARTISTA en la tabla TIPOS_MUSICA_ARTISTA. La clave primaria incluye las columnas NOMBRE_ARTISTA y FDN_ARTISTA. ¿Qué código SQL deberá utilizarse para la restricción de la tabla?

Deberá utilizarse el siguiente código:

```
CONSTRAINT PK_TIPOS_MUSICA_ARTISTA
PRIMARY KEY ( NOMBRE_ARTISTA, FDN_ARTISTA )
```

- 11.** ¿Cómo difiere una restricción referencial de una restricción única?

Las restricciones referenciales están relacionadas con cómo se relacionan los datos en una tabla con los datos en otra tabla, asegurando la integridad referencial entre las dos tablas. Las restricciones únicas aseguran la integridad dentro de una tabla al bloquear valores duplicados.

- 12.** Una restricción _____ impone la integridad referencial entre dos tablas garantizando que no se lleve a cabo ninguna acción en ninguna tabla que pueda afectar a los datos protegidos por la restricción.

FOREIGN KEY

- 13.** Se crea una tabla que incluye la columna llamada ID_TIPO_NEGOCIO, con un tipo de datos INT. La columna se define con una restricción FOREIGN KEY que hace referencia a la clave primaria en la tabla llamada TIPOS_NEGOCIO. La clave foránea se añade como una restricción de columna. ¿Qué código SQL se debe usar para la definición de columna?

Deberá utilizarse el siguiente código:

```
ID_TIPO_NEGOCIO INT REFERENCES TIPOS_NEGOCIO
```

- 14.** ¿Cuáles tres opciones se pueden utilizar en la cláusula MATCH de una restricción FOREIGN KEY?

FULL, PARTIAL y SIMPLE.

- 15.** ¿Cuáles son los dos tipos de acciones referenciales desencadenadas que se pueden definir en una restricción FOREIGN KEY?

ON UPDATE y ON DELETE.

- 16.** Se crea una restricción FOREIGN KEY y se desea que los valores en la columna de referencia se actualicen si los valores en la columna referenciada se actualizan. ¿Qué cláusula <acción referencial desencadenada> utilizaría?

- A** ON UPDATE RESTRICT
- B** ON UPDATE NO ACTION
- C** ON UPDATE CASCADE
- D** ON UPDATE SET DEFAULT

La respuesta correcta es C.

- 17.** ¿Qué sintaxis debe utilizar para una restricción CHECK que se define como una restricción de tabla?

```
[ CONSTRAINT <nombre de la restricción> ] CHECK ( <condición de búsqueda> )
```

18. ¿Qué tipo de restricciones se pueden definir dentro en una afirmación?

Restricciones CHECK.

19. Se crea una restricción CHECK en la columna NUMERO_EN_EXISTENCIA. Se desea limitar los valores que se pueden introducir en la columna en un rango de 11 a 29. ¿Qué debe utilizar para la cláusula <condición de búsqueda> de la restricción?

(NUMERO_EN_EXISTENCIA BETWEEN 11 AND 29)

Capítulo 5: Crear vistas SQL

1. ¿Cuáles son las dos ventajas de utilizar vistas?

Se pueden definir consultas complejas y almacenarlas dentro de la definición de la vista. De esa manera, en lugar de tener que volver a crear esas consultas cada vez que sean necesarias, se puede simplemente invocar la vista. También es posible presentar información a los usuarios sin proporcionarles más información de la que necesitan o información que no deben ver.

2. ¿Cuáles son los tres tipos de tablas almacenadas, soportadas por SQL?

Tablas base, tablas derivadas y tablas vistas (vistas).

3. ¿Qué sucede si no se asignan nombres de columna a una vista?

Las columnas de la vista heredan los nombres de la tabla que la origina.

4. ¿Cómo se asignan los tipos de datos a las columnas de una vista?

No se asignan tipos de datos a las columnas de la vista. Las columnas de la vista heredan sus tipos de datos de sus respectivas columnas de la tabla.

5. ¿En qué circunstancias se deben proporcionar nombres a las columnas de una vista en una definición de vista?

Se deben proporcionar nombres si cualquiera de las columnas que llegan lo hacen mediante algún tipo de operación que calcule el valor que habrá de insertarse en la columna, en lugar del valor que viene directamente de la tabla. También se deben proporcionar nombres si existen nombres de columna duplicados en la tabla, que es algo que sucede cuando se fusionan dos o más tablas.

6. Se crea la vista llamada EMP_CUMPLEAÑOS. La vista se basa en las columnas EMP_NOMBRE y CUMPLEAÑOS de la tabla EMPLEADOS. Los nombres de columna de la vista serán los mismos que los nombres de columna de la tabla. ¿Qué código SQL se utilizaría para crear la vista?

Deberá utilizarse el siguiente código:

```
CREATE VIEW EMP_CUMPLEAÑOS AS
  SELECT EMP_NOMBRE, CUMPLEAÑOS
  FROM EMPLEADOS;
```

7. Se crea una vista basada en la tabla DISCOS_COMPACTOS en la base de datos INVENTARIO. Se desea que la vista incluya sólo las filas cuyo valor en la columna ID_DISQUERA sea 546. ¿Qué cláusula (además de las cláusulas SELECT y FROM) deberá incluirse en la instrucción SELECT para la vista?

Deberá utilizarse la siguiente cláusula WHERE:

```
WHERE ID_DISQUERA = 546
```

- 8. Se crea una vista que hace referencia a las tablas EMPLEADOS y TITULO_TRABAJO. Los datos en las dos tablas coinciden a través de la columna ID_TITULO_TRABAJO en cada tabla. ¿Cómo se debe escribir la cláusula WHERE en la instrucción SELECT de la vista?**

Deberá escribirse la siguiente cláusula WHERE:

```
WHERE EMPLEADOS.ID_TITULO_TRABAJO = TITULO_TRABAJO.ID_TITULO_TRABAJO
```

- 9. Se crea una vista que hace referencia a las tablas EMPLEADOS y TITULO_TRABAJO. Los datos en las dos tablas coinciden a través de la columna ID_TITULO_TRABAJO en cada tabla. Se desea que la vista muestre sólo las filas cuyo valor en la columna ID_TITULO_TRABAJO de la tabla TITULO_TRABAJO sea 109. ¿Cómo se debe escribir la cláusula WHERE en la instrucción SELECT de la vista?**

Deberá escribirse la siguiente cláusula WHERE:

```
WHERE EMPLEADOS.ID_TITULO_TRABAJO = TITULO_TRABAJO.ID_TITULO_TRABAJO
AND TITULO_TRABAJO.ID_TITULO_TRABAJO = 109
```

- 10. ¿Qué es una especificación de consulta?**

Una especificación de consulta es una expresión SQL que comienza con la palabra clave SELECT e incluye varios elementos que forman esa expresión.

- 11. ¿Qué directrices deben seguirse si desea crear una vista actualizable?**

- A** Los datos dentro de la vista no se pueden resumir, agrupar o eliminar automáticamente.
- B** Por lo menos una columna en la tabla fuente debe ser actualizable.
- C** Cada columna en la vista se debe rastrear exactamente a una columna fuente en una tabla.
- D** Cada fila en la vista se debe rastrear exactamente a una fila fuente en una tabla.

Las respuestas correctas son A, B, C y D. Una vista debe cumplir con las cuatro directrices para ser actualizable.

- 12. Se crea la siguiente vista basada en la tabla DISCOS_COMPACTOS en la base de datos INVENTARIO:**

```
CREATE VIEW PROMEDIO_EN_EXISTENCIA AS
SELECT AVG(EN_EXISTENCIA)
FROM DISCOS_COMPACTOS;
```

¿Cómo se insertan datos a través de esta vista?

No es posible insertar datos a través de esta vista debido a que los datos están siendo resumidos (utilizando la función AVG), lo que significa que la fila en la vista no es rastreable para exactamente una fila fuente en una tabla.

- 13. ¿A qué tipo de vista aplica la cláusula WITH CHECK OPTION?**

La cláusula WITH CHECK OPTION aplica a las vistas actualizables que incluyen una cláusula WHERE en la instrucción SELECT.

14. Se crea la siguiente definición de vista:

```
CREATE VIEW COM_EMP AS
SELECT ID_EMPLEADO, AÑO_1999, AÑO_2000
FROM COMISION_EMPLEADO
WHERE AÑO_1999 > 100;
```

Se quiere utilizar la vista para actualizar los datos. ¿Qué sucede si se cambia el valor AÑO_1999 a una cantidad inferior o igual a 100?

La fila es cambiada, pero ya no podrá utilizarse la vista para desplegar la fila o actualizarla. Para evitar que esto ocurra, se puede utilizar la cláusula WITH CHECK OPTION en la instrucción CREATE VIEW.

15. Se desea modificar la definición de la vista COM_EMP en la base de datos. ¿Cómo se modifica esa definición?

Deberá eliminarse la vista y luego volver a crearla.

16. Se desea eliminar la definición de la vista EMP_CUMPLEAÑOS de la base de datos. ¿Qué instrucción SQL deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
DROP VIEW EMP_CUMPLEAÑOS;
```

17. ¿Qué les sucede a los datos de SQL cuando se elimina una vista de la base de datos?

Ninguno de los datos subyacentes (que esté almacenado en las tablas base) se ven afectados cuando se elimina una vista. Solamente la definición de la vista es eliminada.

Capítulo 6: La gestión de la seguridad en la base de datos

1. ¿Cuál es la diferencia entre un identificador de usuario y un nombre de rol?

Un identificador de usuario es una cuenta de seguridad individual que puede representar a un individuo, a una aplicación o a un servicio del sistema. Un nombre de rol es un conjunto definido de privilegios que puede ser asignado a un usuario o a otro rol.

2. ¿Cuál es el nombre del identificador de autorización especial que otorga el acceso a todos los usuarios de la base de datos?

PUBLIC

3. Cada _____ se asocia con un identificador de usuario y un nombre de rol.

Sesión SQL

4. ¿Con cuál de los siguientes se asocia una sesión SQL?

- A** Privilegio
- B** Identificador de usuario
- C** PUBLIC
- D** Nombre de rol

Las respuestas correctas son **B** y **D**.

- 5. Cuando se establece por primera vez una sesión SQL, el identificador de usuario siempre es el _____.**

Identificador de usuario de sesión de SQL.

- 6. ¿Cuál es el valor del nombre de rol actual cuando se establece por primera vez una sesión SQL?**

Un valor nulo.

- 7. ¿Qué es un identificador de autorización?**

Un identificador de autorización es un objeto en el entorno SQL que representa a un usuario o un grupo de usuarios a quienes se les otorgan privilegios específicos de acceso a objetos y datos dentro del entorno SQL.

- 8. ¿Cuáles son los dos tipos de identificadores de autorización que soporta SQL?**

Identificadores de usuario y nombres de rol.

- 9. ¿Qué privilegios se le deben otorgar a un objeto si desea permitir que un identificador de autorización consulte los datos de ese objeto?**

SELECT

- 10. Se establece una sesión SQL con la base de datos. El identificador de usuario actual es EthanW. El nombre de rol actual es nulo. ¿Cuál es el identificador de autorización actual?**

EthanW.

- 11. ¿En qué objetos de esquema se pueden definir privilegios de acceso?**

Tablas base, vistas, columnas, dominios, conjuntos de caracteres, cotejos, traducciones, tipos definidos por el usuario, activadores y rutinas invocadas por SQL.

- 12. ¿En qué tipos de objetos de base de datos se puede asignar el privilegio DELETE?**

- A** Tablas
- B** Vistas
- C** Columnas
- D** Dominios

Las respuestas correctas son A y B.

- 13. ¿En qué tipos de objetos de base de datos se puede asignar el privilegio TRIGGER?**

- A** Tablas
- B** Vistas
- C** Columnas
- D** Dominios

La respuesta correcta es A.

14. Se crea un rol llamado CONTABILIDAD. ¿Qué instrucción SQL deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
CREATE ROLE CONTABILIDAD;
```

15. Se otorgan todos los privilegios en la vista NOMBRE_CD a todos los que utilizan la base de datos. ¿Qué instrucción SQL deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
GRANT ALL PRIVILEGES ON TABLE NOMBRES_CD TO PUBLIC;
```

16. Se otorga el privilegio SELECT al rol EMPLEADO_VENTAS a una tabla de la base de datos. Se desea que el rol EMPLEADO_VENTAS sea capaz de asignar el privilegio SELECT a otros usuarios. ¿Qué cláusula deberá incluirse en la instrucción GRANT?

WITH GRANT OPTION

17. Se desea conceder el rol ACCT a la autorización de usuario MaxN. No se quiere que el usuario pueda otorgar el rol a otros usuarios. ¿Qué instrucción SQL deberá utilizarse para otorgar el rol?

Deberá utilizarse la siguiente instrucción SQL:

```
GRANT ACCT TO MaxN;
```

Capítulo 7: Consulta de datos de SQL

1. ¿Cuáles cláusulas en una instrucción SELECT son parte de la expresión de la tabla?

- A** SELECT
- B** FROM
- C** WHERE
- D** ORDER BY

Las respuestas correctas son **B** y **C**.

2. ¿En qué orden se aplican las cláusulas de una instrucción SELECT?

Las cláusulas son aplicadas en el siguiente orden: FROM, WHERE, GROUP BY, HAVING, SELECT y ORDER BY.

3. Se está escribiendo una instrucción SELECT que recupera la columna TITULO_CD y todas las filas de la tabla INVENTARIO. ¿Cuál instrucción SELECT se debe utilizar?

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT TITULO_CD  
FROM INVENTARIO;
```

4. Se está escribiendo una instrucción SELECT que recupera la columna TITULO_CD y todas las filas de la tabla INVENTARIO. Se desea que la columna en los resultados de la consulta sea nombrada DISCO_COMPACTO. ¿Cuál instrucción SELECT deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT TITULO_CD AS DISCO_COMPACTO
FROM INVENTARIO;
```

5. ¿Cuáles cláusulas se requieren en una instrucción SELECT?

- A** SELECT
- B** FROM
- C** WHERE
- D** GROUP BY

Las respuestas correctas son **A** y **B**.

6. ¿Cuál palabra clave se debe añadir a la cláusula SELECT para asegurarse que cada fila de los resultados de la consulta sea única?

- A** ALL
- B** ROLLUP
- C** DISTINCT
- D** CUBE

La respuesta correcta es **C**.

7. Se está creando una instrucción SELECT para la tabla INVENTARIO y desea asegurarse que sólo las filas con un valor PRECIO_MENUDEO menor a \$16.00 sean incluidas en los resultados de la consulta. ¿Qué cláusula WHERE deberá utilizarse?

Deberá utilizarse la siguiente cláusula WHERE:

```
WHERE PRECIO_MENUDEO < 16.00
```

8. Se está creando una instrucción SELECT que incluye una cláusula WHERE. La cláusula WHERE contiene dos predicados. Se desea que la condición de uno de los predicados se cumpla, pero no es necesario que ambas condiciones se cumplan. ¿Qué palabra clave deberá utilizarse para conectar los dos predicados?

La palabra clave OR.

9. ¿Cada predicado en una cláusula WHERE se evalúa con cuál de las siguientes?

- A** Verdadero
- B** No
- C** Falso
- D** Desconocido

Las respuestas correctas son **A**, **C** y **D**.

10. ¿Cuál cláusula permite agrupar valores en una columna específica?

- A** ROLLUP
- B** HAVING
- C** ORDER BY
- D** GROUP BY

La respuesta correcta es **D**.

11. ¿Cuáles dos operadores pueden utilizarse en una cláusula GROUP BY para arrojar datos de resumen adicionales en los resultados de una consulta?

- A** ROLLUP
- B** HAVING
- C** CUBE
- D** DISTINCT

Las respuestas correctas son **A** y **C**.

12. Se está escribiendo la instrucción SELECT que recupera las columnas CATEGORIA y PRECIO de la tabla EXISTENCIA_DISCO_COMPACTO. Quiere agrupar los datos primero por la columna CATEGORIA y luego por la columna PRECIO. ¿Cuál instrucción SELECT deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT CATEGORIA, PRECIO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CATEGORIA, PRECIO;
```

13. Se está escribiendo la instrucción SELECT que recupera las columnas CATEGORIA y PRECIO de la tabla EXISTENCIA_DISCO_COMPACTO. Quiere agrupar los datos primero por la columna CATEGORIA y luego por la columna PRECIO. A continuación desea filtrar cualquier grupo que tenga un valor PRECIO superior a \$15.99. ¿Cuál instrucción SELECT deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT CATEGORIA, PRECIO
FROM EXISTENCIA_DISCO_COMPACTO
GROUP BY CATEGORIA, PRECIO
HAVING PRECIO < 16.00;
```

14. Se crea una instrucción SELECT que incluye una cláusula SELECT, una cláusula FROM, una cláusula WHERE, una cláusula GROUP BY y una cláusula HAVING. ¿Desde cuál cláusula recibirá resultados la cláusula HAVING?

- A** SELECT
- B** FROM

- C** WHERE
- D** GROUP BY

La respuesta correcta es **D**.

15. ¿En qué aspecto la cláusula HAVING es diferente de la cláusula WHERE?

La cláusula HAVING es similar a la cláusula WHERE en que define una condición de búsqueda. Sin embargo, a diferencia de la cláusula WHERE, la cláusula HAVING está relacionada con grupos, y no con filas individuales.

16. ¿De cuál cláusula recibe resultados la cláusula ORDER BY?

De la cláusula SELECT.

17. ¿Cuál palabra clave deberá agregarse a una cláusula ORDER BY para clasificar los datos en orden descendente?

DESC.

Capítulo 8: Modificar datos SQL

1. ¿Cuál instrucción SQL deberá utilizarse para agregar datos a una tabla?

- A** SELECT
- B** INSERT
- C** UPDATE
- D** DELETE

La respuesta correcta es **B**.

2. ¿Cuáles dos cláusulas son obligatorias en una instrucción INSERT?

INSERT INTO y VALUES

3. ¿En cuál cláusula dentro de la instrucción INSERT se identifica la tabla que recibirá los nuevos datos?

INSERT INTO

4. Se crea la siguiente instrucción INSERT para agregar datos a la tabla ARTISTAS_INTERPRETES:

```
INSERT INTO ARTISTAS_INTERPRETES VALUES ( 12, 'Frank Sinatra' );
```

La tabla ARTISTAS_INTERPRETES incluye tres columnas. ¿Qué sucederá cuando se intente ejecutar esta instrucción?

Se recibirá un error debido a que no existen suficientes valores definidos para la tabla y ninguna columna ha sido especificada para distinguir dónde deben ser insertados los dos valores.

5. ¿Qué información se debe especificar en la cláusula VALUES de una instrucción INSERT?

Se debe especificar uno o más valores a ser insertados en la tabla.

6. ¿Qué requerimientos deberán ser cumplidos por los valores en una cláusula VALUES?

Los valores deben estar encerrados en paréntesis y, si se especifica más de uno, éstos deben estar separados por comas. Si no se especifican los nombres de las columnas en la cláusula INSERT INTO, entonces debe haber un valor para cada columna en la tabla y los valores deben estar en el mismo orden en que están definidos en la tabla. Si los nombres de las columnas están especificados en la cláusula INSERT INTO, entonces debe existir exactamente un valor por cada columna especificada y esos valores deben estar en el mismo orden en que están definidos en la cláusula INSERT INTO. Cada valor con un tipo de datos de cadena debe estar encerrado en comillas simples.

7. Se está creando una instrucción INSERT para introducir datos en la tabla TIPOS_ARTISTA. La tabla incluye solamente dos columnas: ID_ART y NOMBRE_TIPO. Se quiere insertar una fila que incluye el valor ID_ART de 27 y el valor NOMBRE_TIPO de Gospel. ¿Cuál instrucción SQL deberá ser utilizada?

Deberá utilizarse la siguiente instrucción SQL:

```
INSERT INTO TIPOS_ARTISTA VALUES ( 27, 'Gospel' );
```

8. Se está creando una instrucción INSERT que inserta valores tomados desde otra tabla. ¿Qué tipo de instrucción o cláusula se puede utilizar en lugar de la cláusula VALUES para tomar datos desde la otra tabla?

- A** SELECT
- B** SET
- C** SELECT
- D** WHERE

La respuesta correcta es C.

9. ¿Cuál instrucción deberá utilizarse para modificar los datos existentes en una o más filas en una tabla?

- A** SELECT
- B** INSERT
- C** UPDATE
- D** DELETE

La respuesta correcta es C.

10. ¿Cuáles cláusulas son obligatorias en una instrucción UPDATE?

UPDATE y SET

11. ¿Cuál es el propósito de la cláusula WHERE en una instrucción UPDATE?

La cláusula WHERE especificó una condición o un conjunto de condiciones que actúan como un filtro para las filas que son actualizadas. Solamente se actualizan las filas que cumplen con la condición o condiciones especificadas.

12. Se está creando una instrucción UPDATE para actualizar los datos en la tabla ARTISTAS_INTERPRETES. Se quiere actualizar el valor ID_ART en la fila que contenga el valor ID_ART_INTER de 139. El nuevo valor ID_ART es 27. ¿Cuál instrucción SQL deberá ser utilizada?

Deberá utilizarse la siguiente instrucción SQL:

```
UPDATE ARTISTAS_INTERPRETES
    SET ID_ART = 27
    WHERE ID_ART_INTER = 139;
```

- 13.** Se está creando una instrucción UPDATE para actualizar los datos en la tabla ARTISTAS_INTERPRETES. Se quiere actualizar el valor ID_ART de cada fila a 27. ¿Cuál instrucción SQL deberá ser utilizada?

Deberá utilizarse la siguiente instrucción SQL:

```
UPDATE ARTISTAS_INTERPRETES
    SET ID_ART = 27;
```

- 14.** Se están actualizando dos columnas en la tabla INVENTARIO_CD. Se quiere cambiar el valor EDITOR a MCA Records y se quiere duplicar el valor EN_EXISTENCIA. ¿Cuál cláusula SET deberá ser utilizada?

Deberá utilizarse la siguiente instrucción SQL:

```
SET PUBLISHER = 'MCA Records',
    EN_EXISTENCIA = (EN_EXISTENCIA * 2);
```

- 15.** Se está creando una instrucción UPDATE que incluye una cláusula SET con una expresión de valor. Se requiere que la expresión de valor tome un valor desde otra tabla en la base de datos. ¿Cuál instrucción o cláusula se puede utilizar como una expresión de valor para seleccionar datos desde otra tabla?

- A** SELECT
- B** WHERE
- C** UPDATE
- D** INSERT

La respuesta correcta es **A**.

- 16.** ¿Cuál cláusula es requerida en una instrucción DELETE?

```
DELETE FROM
```

- 17.** ¿Cuál instrucción o cláusula se utiliza en una instrucción DELETE para especificar cuáles filas serán eliminadas en una tabla?

- A** SELECT
- B** WHERE
- C** UPDATE
- D** INSERT

La respuesta correcta es **B**.

Capítulo 9: Utilizar predicados

1. ¿En cuál cláusula de la instrucción **SELECT** se incluyen predicados?

En la cláusula **WHERE**

2. ¿Cuál símbolo de operador de comparación deberá utilizarse para expresar una condición desigual?

A <=

B >=

C <>

D =<

La respuesta correcta es **C**.

3. ¿Cuáles palabras clave pueden utilizarse para combinar predicados en una cláusula **WHERE**?

La palabra clave **AND** y la palabra clave **OR**.

4. Se quiere consultar una tabla que incluye la columna **PRECIO**. Es necesario asegurarse que todas las filas arrojadas tengan un valor **PRECIO** de 13.99. ¿Cuál predicado deberá utilizarse?

PRECIO = 13.99

5. Se crea la siguiente instrucción **SQL**:

```
SELECT TITULO_CD, PRECIO_MENUDEO
FROM CDS_A_LA_MANO
WHERE PRECIO_MENUDEO >= 14
AND PRECIO_MENUDEO <= 16;
```

¿Qué predicado puede utilizarse en lugar de los dos predicados mostrados en esta instrucción?

PRECIO_MENUDEO BETWEEN 14 AND 16

6. ¿Qué palabra clave puede agregarse a un predicado **BETWEEN** para encontrar el inverso de la condición especificada por el predicado?

NOT

7. ¿Cuándo se utiliza un valor nulo en una columna?

Se utiliza un valor nulo en lugar de un valor regular cuando éste es desconocido o no identificado. Un valor nulo indica que el valor está ausente. Esto no es lo mismo que un cero, un espacio en blanco o un valor predeterminado.

8. Se quiere consultar una tabla para determinar cuáles valores son nulos. ¿Qué tipo de predicado deberá utilizarse?

El predicado **NULL**

9. Se está creando una instrucción **SELECT** que consulta la tabla **BIO_ARTISTAS**. Se quiere arrojar todas las columnas en la tabla, pero arrojar sólo aquellas columnas que no contengan valores nulos en la columna **LUGAR_DE_NACIMIENTO**. ¿Cuál instrucción **SELECT** deberá utilizarse?

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT *
  FROM BIO_ARTISTAS
 WHERE LUGAR_DE_NACIMIENTO IS NOT NULL;
```

- 10. Se está consultando la tabla INVENTARIO_CD. Se quiere ver todas las columnas, pero se requiere ver sólo aquellas columnas que contengan la palabra Christmas en el nombre del CD. Los nombres están almacenados en la columna TITULO_CD. ¿Cuál instrucción SELECT deberá utilizarse?**

Deberá utilizarse la siguiente instrucción SQL:

```
SELECT *
  FROM INVENTARIO_CD
 WHERE TITULO_CD LIKE ('%Christmas%');
```

- 11. ¿Cuál es la diferencia entre un signo de porcentaje y un guión bajo cuando se usan en un predicado LIKE?**

El signo de porcentaje representa cero o más caracteres desconocidos, mientras que el guión bajo representa exactamente un carácter desconocido.

- 12. ¿Cuáles dos tipos de fuentes de datos pueden utilizarse en un predicado IN?**

Una lista definida o una subconsulta.

- 13. ¿Cuál tipo de predicado se ocupa solamente de determinar si una subconsulta arroja cualquier fila o no?**

El predicado EXISTS.

- 14. ¿Cuáles nombres de columna deben ser especificados en un predicado EXISTS?**

No tiene importancia cuáles columnas o cuántas columnas se especifiquen en la cláusula SELECT de la subconsulta en un predicado EXISTS. Este tipo de predicado está únicamente relacionado con la condición de que las filas sean arrojadas, no con el contenido de esas filas. Por lo tanto, se puede especificar cualquier nombre para las columnas o solamente un asterisco.

- 15. Se está creando una instrucción SELECT que incluye un predicado en la cláusula WHERE. Se quiere utilizar un operador de comparación para comparar los valores en una de las columnas con los resultados de una subconsulta. Se quiere que el predicado se evalúe como verdadero para cualquiera de los resultados de la subconsulta. ¿Qué tipo de predicado deberá utilizarse?**

- A** EXISTS
- B** ANY
- C** ALL
- D** IN

La respuesta correcta es **B**.

- 16. ¿Cuál es la diferencia entre un predicado SOME y un predicado ANY?**

No hay diferencia. Los dos predicados arrojan resultados idénticos.

17. ¿Cómo difiere el predicado ALL del predicado SOME?

En muchos aspectos, el predicado ALL funciona de la misma forma que el predicado SOME. El predicado ALL compara los valores de columna con los resultados de la subconsulta. Sin embargo, en lugar de que los valores de la columna tengan que evaluarse como verdaderos para cualquiera de los valores resultantes, los valores de la columna deben evaluarse como verdaderos para todos los valores resultantes; de otra manera, la fila no es arrojada.

Capítulo 10: Trabajar con funciones y expresiones de valor

1. ¿Qué es una función set?

Una función set es un tipo de función que procesa o calcula los datos y arroja los valores apropiados.

2. Se está creando una instrucción SELECT que consulta la tabla CDS_ARTISTA. La tabla incluye las columnas NOMBRE_ARTISTA y NOMBRE_CD. Se requiere que la instrucción arroje el número total de filas en la tabla. ¿Cuál función COUNT deberá incluirse en la cláusula SELECT?

- A** COUNT(*)
- B** COUNT(NOMBRE_ARTISTA)
- C** COUNT(NOMBRE_CD)
- D** COUNT(NOMBRE_ARTISTA, NOMBRE_CD)

La respuesta correcta es A.

3. ¿Cuál función set deberá utilizarse para sumar los valores en una columna?

- A** MAX
- B** COUNT
- C** SUM
- D** AVG

La respuesta correcta es C.

4. Las funciones set requieren que los datos estén_____ de alguna manera.

Agrupados.

5. ¿Qué son las funciones de valor?

Las funciones de valor son un tipo de función que permite arrojar un valor que de alguna manera calcule o derive información desde los datos almacenados dentro de las tablas o desde la misma implementación SQL.

6. Se está utilizando la función SUBSTRING para extraer caracteres de la columna DISCO_COMPACTO de la tabla FECHAS_VENTAS. Se quiere iniciar con el tercer carácter y extraer ocho caracteres. ¿Qué parámetros deberán utilizarse en la función SUBSTRING?

(DISCO_COMPACTO FROM 3 FOR 8)

- 7.** Se está utilizando la función LOWER en el valor Past Light de la columna NOMBRE_CD. ¿Qué valor será arrojado?

El valor arrojado será el siguiente: past light.

- 8.** ¿Qué función arroja un valor que represente la fecha y la hora actuales al igual que la información relacionada con UCT?

- A** LOCALTIMESTAMP
- B** CURRENT_DATE
- C** LOCALTIME
- D** CURRENT_TIMESTAMP

La respuesta correcta es **D**.

- 9.** ¿Cuáles son los cuatro tipos de operadores que se utilizan en una expresión de valor numérica?

Suma, resta, multiplicación y división.

- 10.** Se están consultando datos de la tabla RASTREO_CD. Se quieren agregar valores en la columna EN_EXISTENCIA a los valores en la columna EN_PEDIDO. Luego se quiere duplicar los totales de la columna. ¿Cómo se establece la expresión de valor numérica?

$(EN_EXISTENCIA + EN_ORDEN) * 2$

- 11.** ¿Cuál expresión de valor se utiliza para establecer una serie de condiciones que modifiquen valores?

La expresión CASE.

- 12.** Se está creando una instrucción SELECT que incluye una expresión de valor CASE. Se requiere que una de las condiciones especifique que cualquier valor EN_PEDIDO mayor a 10 se incremente en 5. ¿Cómo deberá establecerse la cláusula WHEN/THEN?

WHEN EN_PEDIDO > 10 THEN EN_PEDIDO + 5

- 13.** ¿Cuál es la última palabra en una expresión de valor CASE?

La palabra END.

- 14.** ¿Qué es la expresión de valor CAST?

Una expresión de valor CAST es un tipo de expresión que permite cambiar un tipo de datos de valor cuando se recupera ese valor desde la base de datos.

- 15.** Se está consultando la columna FECHA_VENTA en la tabla FECHAS_VENTAS. Se requiere convertir los valores a un tipo de datos CHAR(25), y que los datos sean desplegados en la columna CHAR_FECHA en los resultados de la consulta. ¿Cómo se define la expresión de valor CAST?

CAST(FECHA_VENTA AS CHAR (25)) AS CHAR_FECHA

- 16.** ¿Qué valor especial puede utilizarse para identificar al identificador de usuario de sesión SQL actual?

SESSION_USER

Capítulo 11: Acceder a múltiples tablas

- 1. Se está utilizando una operación join separada por comas para unir dos tablas. La primera tabla contiene cinco filas y la segunda tabla contiene tres filas. ¿Cuántas filas contendrá la tabla de producto cartesiano?**

15 filas.

- 2. ¿Qué constituye una condición equi-join en una cláusula WHERE?**

Los valores en una o más columnas en la primera tabla son igualados con los valores en una o más columnas correspondientes en la segunda tabla.

- 3. ¿Cuál cláusula contiene la condición equi-join en una operación join separada por comas?**

La cláusula WHERE.

- 4. ¿Qué lineamientos básicos deberán seguirse cuando se crea una operación join separada por comas?**

La cláusula FROM deberá incluir todos los nombres de las tablas, la cláusula WHERE deberá definir una condición equi-join, y las referencias de columna deberán ser cualificadas cuando los nombres de columna sean compartidos entre las tablas.

- 5. Se está creando una operación join sobre dos tablas. Se asignan nombres de correlación para cada una de estas tablas. ¿Cuáles nombres deberán utilizarse en la cláusula SELECT: los nombres de correlación o los nombres reales de las tablas?**

Los nombres de correlación.

- 6. ¿Qué tipo de operación join es prácticamente idéntica a la operación join separada por comas?**

- A** Join de condición
- B** Join natural
- C** Cross join
- D** Join de columna nombrada

La respuesta correcta es C.

- 7. ¿Cuántas tablas están contenidas en una operación self-join?**

Una.

- 8. ¿Qué lineamientos deberán seguirse cuando se crean operaciones join naturales o de columna nombrada?**

Las columnas unidas deben compartir el mismo nombre y tener tipos de datos compatibles. Y los nombres de las columnas unidas no pueden ser cualificados con los nombres de las tablas.

- 9. ¿Cuál es la diferencia entre una operación join natural y una de columna nombrada?**

La operación join natural automáticamente hace coincidir las filas para aquellas columnas con el mismo nombre. No es necesario especificar ningún tipo de condición equi-join para las operaciones join naturales. La implementación SQL determina cuáles columnas tienen los mismos nombres y luego intenta formar una coincidencia. En una operación join de columna nombrada, se debe especificar la columna coincidente. Las columnas coincidentes no se determinan automáticamente.

- 10. ¿Qué tipo de operación join contiene una cláusula USING para especificar la condición equi-join?**

Join de columna nombrada.

- 11. ¿Cuáles son los dos tipos de operaciones join de condición?**

Inner joins y outer joins

- 12. ¿Cuáles son los tres tipos de operación outer join?**

Left, right y full.

- 13. ¿Cuál tipo de operación join de condición deberá utilizarse si se quiere arrojar solamente filas coincidentes?**

- A** Inner join
- B** Left outer join
- C** Right outer join
- D** Full outer join

La respuesta correcta es **A**.

- 14. ¿Qué tipo de operación join de condición arroja todas las filas coincidentes y no coincidentes?**

- A** Inner join
- B** Left outer join
- C** Right outer join
- D** Full outer join

La respuesta correcta es **D**.

- 15. ¿Qué tipo de operación join contiene una cláusula ON?**

- A** Cross join
- B** Join separada por comas
- C** Join natural
- D** Join de condición

La respuesta correcta es **D**.

- 16. Un operador _____ permite combinar instrucciones SELECT separadas en una sola instrucción para unir los datos en un solo resultado de consulta.**

UNION

- 17. ¿Qué palabra clave puede utilizarse con un operador UNION para arrojar todas las filas en los resultados de la consulta, sin importar si existen valores duplicados?**

La palabra clave ALL.

Capítulo 12: Utilizar subconsultas para acceder y modificar datos

1. ¿En cuál tipo de instrucción se pueden incluir subconsultas?

- A** SELECT
- B** INSERT
- C** UPDATE
- D** DELETE

Las respuestas correctas son **A, B, C y D**.

2. ¿Qué es una subconsulta?

Una subconsulta es una instrucción SELECT incrustada que actúa como una puerta de comunicación hacia los datos en una segunda tabla. Los datos arrojados por la subconsulta se utilizan por la instrucción primaria para cumplir cualquier condición que haya sido definida para esa instrucción.

3. ¿En cuáles cláusulas de una instrucción SELECT se puede incluir una subconsulta?

- A** SELECT
- B** WHERE
- C** GROUP BY
- D** HAVING

Las respuestas correctas son **A, B y D**.

4. ¿En cuáles dos categorías generales se pueden dividir las subconsultas de una cláusula WHERE?

Las subconsultas que pueden arrojar múltiples filas y aquellas que pueden arrojar solamente un valor.

5. ¿Cuáles tipos de predicados se debe evitar utilizar con subconsultas que arrojen múltiples filas?

- A** Predicados IN y EXISTS
- B** Predicados SOME, ANY y ALL
- C** Predicados de comparación
- D** Predicados de comparación cuantificados

La respuesta correcta es **C**.

6. ¿Cuándo se evalúa una condición EXISTS como verdadera?

Una condición EXISTS se evalúa como verdadera si una o más filas son arrojadas por la subconsulta; de otra manera, se evalúa como falsa.

7. ¿Qué deberá incluirse en la condición de búsqueda de una subconsulta cuando se utiliza un predicado EXISTS?

La subconsulta de un predicado EXISTS deberá incluir una condición de búsqueda que coincida con los valores en las dos tablas que están siendo vinculadas a través de la subconsulta.

- 8. Además de números, los datos _____ pueden ser comparados en los predicados de comparación.**

De cadena de caracteres.

- 9. ¿Cuáles son los tres predicados de comparación cuantificados?**

SOME, ANY y ALL

- 10. ¿Qué tipos de predicados permiten utilizar subconsultas que arrojen múltiples filas?**

- A** Predicados IN y EXISTS
- B** Predicados SOME, ANY y ALL
- C** Predicados de comparación
- D** Predicados de comparación cuantificados

Las respuestas correctas son **A, B y D**.

- 11. ¿Qué es una subconsulta correlacionada?**

Una subconsulta correlacionada es aquella que es dependiente en alguna manera de la instrucción outer.

- 12. ¿Con qué constancia es evaluada una subconsulta correlacionada cuando se procesa una instrucción SELECT?**

La subconsulta correlacionada debe ser evaluada para cada fila arrojada por la instrucción SELECT outer.

- 13. Un(a) _____ es una subconsulta que es un componente de otra subconsulta.**

Subconsulta anidada.

- 14. ¿Cuántas subconsultas pueden ser incluidas en una instrucción SELECT, según especifica el estándar SQL?**

El estándar SQL:2006 no limita el número de subconsultas que pueden ser incluidas en una instrucción.

- 15. ¿Cuál cláusula en una instrucción INSERT puede contener una subconsulta?**

La cláusula VALUES.

- 16. ¿Cuántos valores puede arrojar una subconsulta si es utilizada dentro de una instrucción INSERT?**

Uno.

- 17. ¿Cuáles cláusulas en una instrucción UPDATE pueden contener una subconsulta?**

WHERE y SET

Capítulo 13: Crear rutinas invocadas por SQL

- 1. ¿Cuáles son los tipos de rutinas invocadas por SQL soportados por el estándar SQL?**

- A** CHECK Constraint
- B** Function

- C** Trigger
- D** Procedimiento invocado por SQL

Las respuestas correctas son **B** y **D**.

2. ¿Cuáles tipos de parámetros pueden utilizarse en una función invocada por SQL?

- A** De entrada
- B** De salida
- C** De entrada/salida
- D** Variables

Las respuestas correctas son **A**, **B** y **C**.

3. ¿Cuál instrucción se utiliza para invocar un procedimiento invocado por SQL?

- A** RETURN
- B** CALL
- C** SET
- D** DECLARE

La respuesta correcta es **B**.

4. Un(a) _____ es un valor pasado a una instrucción en un procedimiento cuando se invoca ese procedimiento.

Parámetro.

5. ¿Cuáles tipos de parámetros pueden utilizarse en una función invocada por SQL?

- A** De entrada
- B** De salida
- C** De entrada/salida
- D** Variables

La respuesta correcta es **A**.

6. ¿Cuál es otro nombre para un procedimiento invocado por SQL?

Procedimiento almacenado.

7. ¿Cuáles son las dos diferencias principales entre procedimientos y funciones?

Los procedimientos son invocados utilizando una instrucción CALL, y soportan parámetros de entrada y de salida. Las funciones son invocadas como un valor en una expresión, y solamente soportan parámetros de entrada.

8. ¿Qué información debe incluirse en una instrucción CALL cuando se invoca un procedimiento?

El nombre del procedimiento y los valores que son pasados a los parámetros.

9. ¿Qué tipo de instrucciones pueden incluirse en un procedimiento?

- A** SELECT
- B** INSERT

- C** UPDATE
- D** DELETE

Las respuestas correctas son **A, B, C y D**.

10. ¿Cuál instrucción se utiliza para asignar un valor inicial a una variable?

- A** DECLARE
- B** RETURN
- C** SET
- D** CALL

La respuesta correcta es **C**.

11. Una instrucción _____ permite agrupar las instrucciones SQL en bloques.

Control.

12. ¿Qué palabra clave se utiliza para comenzar una instrucción condicional?

- A** IF
- B** BEGIN
- C** THEN
- D** ELSE

La respuesta correcta es **A**.

13. ¿Qué palabra clave se utiliza en una instrucción LOOP para terminar la repetición?

LEAVE.

14. ¿Cuál es la diferencia entre una instrucción condicional y una instrucción compuesta?

Una instrucción condicional determina si una instrucción es ejecutada basada en si una condición específica se evalúa como verdadera. Una instrucción compuesta agrupa las instrucciones en un bloque.

15. ¿Cuáles son los dos tipos de instrucciones de repetición?

- A** BEGIN...END
- B** IF...END IF
- C** LOOP...END LOOP
- D** WHILE...END WHILE

Las respuestas correctas son **C y D**.

16. ¿Qué tipo de parámetro puede arrojar un valor cuando se invoca un procedimiento?

De salida.

17. ¿Qué paso debe tomarse cuando se convoque un procedimiento que incluya un parámetro de salida?

Se debe declarar una variable que sea entonces utilizada en la instrucción CALL como un valor de parámetro.

18. ¿Qué tanto difiere una instrucción CREATE FUNCTION de una instrucción CREATE PROCEDURE?

En una instrucción CREATE FUNCTION, las definiciones de parámetro de entrada no pueden incluir la palabra clave IN. Además, una cláusula RETURNS debe seguir a las definiciones de parámetro, y el cuerpo de la rutina debe incluir una instrucción RETURN.

19. Se está convocando un procedimiento llamado OBTENER_TOTALES. El procedimiento no incluye ningún parámetro, pero sí incluye una instrucción SELECT que consulta la tabla INVENTARIO_CD. ¿Cuál instrucción SQL deberá utilizarse para invocar este parámetro?

Se deberá utilizar la siguiente instrucción SQL:

```
CALL OBTENER_TOTALES( );
```

20. Se crea un procedimiento llamado OBTENER_INFO_CD que selecciona datos acerca de un artista de la tabla INFO_CD. El procedimiento incluye un parámetro de entrada. Se quiere convocar ese procedimiento con el valor Bonnie Raitt. ¿Cuál instrucción SQL deberá utilizarse para invocar el procedimiento?

Se deberá utilizar la siguiente instrucción SQL:

```
CALL OBTENER_INFO_CD ('Bonnie Raitt');
```

21. ¿Cuáles son los dos tipos de objetos de esquema que pueden utilizarse para almacenar una instrucción SELECT?

Vistas y procedimientos invocados por SQL.

Capítulo 14: Crear activadores SQL

1. ¿Qué es un activador?

Un activador es un objeto de esquema que es invocado automáticamente cuando se realiza una modificación de datos específica. Una vez invocado, el activador toma una acción predefinida, la cual se encuentra especificada en una o más instrucciones SQL.

2. ¿Cuáles son los tres tipos de activadores?

Insert, update y delete.

3. ¿Qué tipo de acciones pueden ser realizadas por las instrucciones SQL activadas?

Las instrucciones SQL activadas pueden realizar acciones tales como actualizar tablas, eliminar datos, invocar procedimientos, o realizar la mayoría de las tareas que pueden llevarse a cabo con instrucciones SQL.

4. ¿Cuáles acciones puede invocar un activador?

A Actualización de datos

B Consulta de datos

- C** Eliminación de datos
- D** Inserción de datos

Las respuestas correctas son **A**, **C** y **D**.

5. ¿Cuándo es invocado un activador de inserción?

Cuando se insertan datos en la tabla en la cual se encuentra definido el activador.

6. ¿En cuántas tablas puede ser definido un activador?

- A** Solamente una
- B** Una o más
- C** De una a tres
- D** Cualquier número de tablas

La respuesta correcta es **A**.

7. Un(a) _____ es un espacio creado en la memoria que alberga un proceso de activador durante la ejecución de ese activador.

Contexto de ejecución del activador.

8. Se insertan datos en la tabla 1, que invoca un activador de inserción definido en esa tabla. El activador actualiza la información en la tabla 2, que invoca un activador de actualización definido en esa tabla. El activador de actualización elimina información en la tabla 3, que invoca un activador de eliminación definido en esa tabla. ¿Cuál contexto de ejecución de activador está activo en este punto?

- A** El contexto de ejecución de activador para el activador de inserción
- B** El contexto de ejecución de activador para el activador de actualización
- C** El contexto de ejecución de activador para el activador de eliminación

La respuesta correcta es **C**.

9. Si tres activadores son invocados durante una sesión, ¿cuántos contextos de ejecución de activador son creados en esa sesión?

Tres.

10. ¿Qué información es incluida en un contexto de ejecución de activador?

Un contexto de ejecución del activador contiene la información necesaria para que el activador sea ejecutado correctamente. Esta información incluye detalles acerca del mismo activador y de la tabla sujeto. Además, el contexto de ejecución incluye una o dos tablas de transición.

11. ¿En cuál cláusula de la instrucción CREATE TRIGGER se asignan nombres de correlación a los datos antiguos y nuevos?

- A** FOR EACH
- B** ON
- C** REFERENCING
- D** WHEN

La respuesta correcta es **C**.

- 12.** ¿En cuál cláusula de la instrucción **CREATE TRIGGER** se especifica si las instrucciones SQL activadas serán ejecutadas una vez para cada fila o una vez para cada instrucción?

A FOR EACH
B ON
C REFERENCING
D WHEN

La respuesta correcta es **A**.

- 13.** Se está creando una definición de activador para un activador de inserción. ¿Cuáles cláusulas **REFERENCING** se pueden incluir en la instrucción **CREATE TRIGGER**?

A REFERENCING OLD ROW AS Old
B REFERENCING NEW ROW AS New
C REFERENCING OLD TABLE AS Old
D REFERENCING NEW TABLE AS New

Las respuestas correctas son **B** y **D**.

- 14.** Un activador _____ permite especificar los nombres de columna de una tabla en cuestión.

Update.

- 15.** ¿Cuáles palabras clave pueden utilizarse para designar si las instrucciones SQL activadas serán ejecutadas antes o después de que la instrucción de modificación de datos sea aplicada a la tabla en cuestión?

Las palabras clave **BEFORE** o **AFTER**.

- 16.** Se está creando un activador de actualización en la tabla **INVENTARIO_CD**. La tabla incluye una columna llamada **EN_EXISTENCIA**. Se requiere que las instrucciones SQL activadas sean ejecutadas solamente cuando el valor **EN_EXISTENCIA** de la fila actualizada exceda 20. ¿Cuál cláusula deberá incluirse en la instrucción **CREATE TRIGGER** para restringir cuándo son ejecutadas las instrucciones?

A WHERE
B HAVING
C FOR EACH
D WHEN

La respuesta correcta es **D**.

- 17.** ¿Qué instrucción debe incluirse en la instrucción **CREATE TRIGGER** si la definición del activador incluye más de una instrucción SQL activada?

La instrucción **BEGIN...END**.

- 18.** ¿Cuál instrucción puede utilizarse para eliminar un activador del esquema?

La instrucción **DROP TRIGGER**.

19. ¿Cuál instrucción SQL se utiliza para alterar una definición de activador?

SQL no soporta una instrucción que permita alterar una definición de activador. Primero debe utilizarse una instrucción DROP TRIGGER para eliminar al activador de la base de datos, y luego utilizar la instrucción CREATE TRIGGER para volver a crear al activador.

Capítulo 15: Utilizar cursores SQL

1. ¿Qué es un cursor?

Un cursor funciona como un señalador que permite al lenguaje de programación de aplicación tratar con los resultados de la consulta una fila a la vez. A pesar de que el cursor puede recorrer todas las filas de los resultados de la consulta, se enfoca solamente en una fila a la vez.

2. ¿Cuáles métodos de invocación soportan el uso de cursores?

SQL incrustado y los módulos cliente SQL.

3. ¿Cuál forma de incongruencia en la impedancia es cubierta a través del uso de cursores?

El hecho de que SQL arroje datos en conjuntos pero otros lenguajes de programación solamente puedan procesar muy pocos segmentos de datos al mismo tiempo.

4. Un(a) _____ funciona como un señalador que permite al lenguaje de programación de aplicación tratar con los resultados de la consulta una fila a la vez.

Cursor.

5. Cuando se utilizan cursores en SQL incrustado, ¿cuál es el primer paso que se debe tomar antes de que se puedan recuperar datos a través de ese cursor?

- A** Buscar el cursor.
- B** Declarar el cursor.
- C** Cerrar el cursor.
- D** Abrir el cursor.

La respuesta correcta es **B**.

6. ¿Cuáles son las cuatro instrucciones relacionadas con el cursor que pueden ser incrustadas en un lenguaje host?

DECLARE CURSOR, OPEN, FETCH y CLOSE.

7. ¿Cuáles opciones pueden ser utilizadas en las instrucciones de cursor de sólo lectura?

- A** SCROLL
- B** WITH HOLD
- C** ORDER BY
- D** INSENSITIVE

Las respuestas correctas son **A, C y D**.

8. ¿Cuáles son los elementos obligatorios de una instrucción DECLARE CURSOR?

DECLARE <nombre del cursor> CURSOR FOR <expresión de la consulta>

9. ¿Qué tipo de cursor no ve los cambios hechos por instrucciones fuera del cursor?

Cursor no sensitivo.

10. ¿Cuál opción deberá utilizarse en una instrucción de cursor para extender las capacidades de recuperación de una instrucción FETCH?

A WITHOUT HOLD

B ASENSITIVE

C SCROLL

D FOR UPDATE

La respuesta correcta es C.

11. La capacidad para _____ el cursor hace referencia a una característica en los cursores que está relacionada con la condición de cerrar o no un cursor automáticamente cuando la transacción en la que el cursor fue abierto es completada.

Mantener abierto.

12. Se está creando una instrucción de cursor. La instrucción SELECT incluye una cláusula ORDER BY. ¿Cuáles cláusulas no pueden ser incluidas en la instrucción SELECT?

A SELECT

B HAVING

C GROUP BY

D WHERE

Las respuestas correctas son B y C.

13. ¿Cuál opción deberá incluirse en una instrucción de cursor para definir que ese cursor tiene capacidades para mantenerse abierto?

WITH HOLD

14. La instrucción de cursor incluye una cláusula FOR UPDATE que no especifica ninguna columna. ¿Cuáles columnas en la tabla subyacente pueden ser actualizadas?

Todas las columnas.

15. ¿Cuál instrucción SQL deberá utilizarse si se quiere abrir el cursor ARTISTAS_CD?

Deberá utilizarse la siguiente instrucción:

OPEN ARTISTAS_CD;

16. ¿Cuál instrucción SQL ejecuta la instrucción SELECT en un cursor?

La instrucción OPEN.

- 17.** Una instrucción _____ recupera filas desde los resultados de la consulta del cursor una vez que se abre ese cursor.

FETCH.

- 18.** ¿Qué tipo de cursor permite utilizar todas las opciones de orientación para búsqueda en la instrucción FETCH?

El cursor con capacidad de desplazamiento.

- 19.** ¿Cuál opción de orientación para búsqueda deberá utilizarse en una instrucción FETCH si se requiere asegurarse de recuperar la primera fila en los resultados de la consulta del cursor?

- A** PRIOR
- B** NEXT
- C** ABSOLUTE -1
- D** FIRST

La respuesta correcta es **D**.

- 20.** ¿Cuál cláusula es requerida en una instrucción UPDATE posicionada para poder actualizar una fila arrojada por la instrucción FETCH más reciente?

WHERE CURRENT OF <nombre del cursor>

Capítulo 16: Manejar transacciones SQL

- 1.** ¿Cuál característica de una transacción se refiere a la naturaleza todo-o-nada de una transacción?

- A** Atómica
- B** Consistente
- C** Aislada
- D** Durable

La respuesta correcta es **A**.

- 2.** Un(a) _____ es una unidad de trabajo que consta de una o más instrucciones SQL que realizan un conjunto relacionado de acciones.

Transacción.

- 3.** ¿Qué instrucción puede utilizarse para iniciar explícitamente una transacción?

START TRANSACTION.

- 4.** ¿Cuáles instrucciones SQL finalizarán una transacción?

- A** SAVEPOINT
- B** SET TRANSACTION
- C** ROLLBACK
- D** COMMIT

Las respuestas correctas son **C** y **D**.

- 5. ¿Cuáles son los tres tipos de modo de transacción que pueden especificarse en una instrucción SET TRANSACTION?**

Nivel de acceso, nivel de aislamiento y tamaño de diagnóstico.

- 6. ¿Cuáles opciones del nivel de acceso pueden incluirse en una instrucción START TRANSACTION?**

- A** READ ONLY
- B** UPDATE
- C** LOCAL
- D** READ WRITE

Las respuestas correctas son **A** y **D**.

- 7. Dos transacciones simultáneas están activas en el sistema. La primera transacción modifica los datos en una tabla. La segunda transacción ve esas modificaciones antes de que sean completadas realmente en la base de datos. Luego, la primera transacción reinvierte las modificaciones. ¿Qué tipo de irregularidad de datos ha ocurrido?**

- A** Lectura fantasma
- B** Lectura repetible
- C** Lectura sucia
- D** Lectura no repetible

La respuesta correcta es **C**.

- 8. Una lectura _____ puede ocurrir cuando una transacción lee una tabla basada en algún tipo de condición de búsqueda, después una segunda transacción actualiza los datos en la tabla, y la primera transacción intenta volver a leer los datos, sólo que esta vez se arrojan diferentes filas debido a como está definida la condición de búsqueda.**

Fantasma.

- 9. ¿Qué tipo de restricción puede especificarse en una instrucción SET CONSTRAINTS?**

Restricción aplazable.

- 10. ¿Cuál de los niveles de aislamiento aísla por completo una transacción de otra transacción?**

SERIALIZABLE.

- 11. Se está utilizando una instrucción SET TRANSACTION para configurar los modos de transacción. Se requiere asegurarse de que no puedan ocurrir lecturas no repetibles ni lecturas sucias dentro de la transacción. Sin embargo, no es necesario preocuparse acerca de las lecturas fantasma. ¿Cuál nivel de aislamiento deberá utilizarse?**

- A** READ UNCOMMITTED
- B** READ COMMITTED

- C** REPEATABLE READ
- D** SERIALIZABLE

La respuesta correcta es **C**.

- 12.** Se está configurando una transacción que aplase la aplicación de la restricción REST_CD_EXISTENCIA hasta que se ejecuten varias instrucciones SQL. Después de ejecutar las instrucciones, se requiere aplicar explícitamente la restricción a los cambios que se hicieron a la base de datos. ¿Qué instrucción SQL deberá utilizarse para aplicar las restricciones?

Deberá utilizarse la siguiente instrucción:

```
SET CONSTRAINTS REST_CD_EXISTENCIA IMMEDIATE;
```

- 13.** Un(a) _____ es un marcador designado dentro de la transacción que actúa como un punto para reinvertir una porción de la transacción.

Punto de recuperación.

- 14.** Se necesita crear un punto de recuperación llamado svpt_Seccion2. ¿Qué instrucción SQL deberá utilizarse?

Deberá utilizarse la siguiente instrucción:

```
SAVEPOINT svpt_Seccion2;
```

- 15.** Se crea una transacción que incluye cuatro puntos de recuperación: Seccion1, Seccion2, Seccion3 y Seccion4. Cerca del final de la transacción, posterior a los cuatro puntos de recuperación, se define RELEASE SAVEPOINT que especifica el punto de recuperación Seccion2. ¿Cuál punto o cuáles puntos de recuperación son eliminados de la transacción cuando se ejecuta la instrucción RELEASE SAVEPOINT?

- A** Seccion1
- B** Seccion2
- C** Seccion3
- D** Seccion4

Las respuestas correctas son **B, C y D**.

- 16.** ¿Qué circunstancias finalizarán una transacción?

Que una instrucción ROLLBACK sea definida explícitamente en la transacción; que una instrucción COMMIT sea definida explícitamente en la transacción; que el programa que inició la transacción sea interrumpido, causando que el programa aborte, o que el programa complete exitosamente su ejecución.

- 17.** Se está creando una instrucción ROLLBACK en la transacción. Se necesita que la reinversión deshaga los cambios hasta el punto de recuperación svpt_Seccion2. ¿Qué instrucción SQL deberá utilizarse?

Deberá utilizarse la siguiente instrucción:

```
ROLLBACK TO SAVEPOINT svpt_Seccion2;
```


- 18.** Se está creando una instrucción COMMIT en la transacción. Después de que la transacción es finalizada, se requiere que inicie una nueva transacción. La nueva transacción deberá estar configurada con los mismos modos de transacción que la primera transacción. ¿Cómo deberá crearse la instrucción COMMIT?

Deberá utilizarse la siguiente instrucción:

```
COMMIT AND CHAIN;
```

Capítulo 17: Acceder a datos SQL desde un programa host

- 1.** ¿Cuál método de acceso de datos deberá utilizarse si se quiere crear y ejecutar instrucciones SQL *ad hoc*?

- A** CLI
- B** De módulos cliente de SQL
- C** Invocación directa
- D** SQL incrustado

La respuesta correcta es C.

- 2.** ¿Qué es SQL incrustado?

SQL incrustado se refiere a las instrucciones SQL que están entremezcladas con algún tipo de lenguaje de programación de aplicación. Las instrucciones SQL se fusionan dentro del lenguaje host para permitir al programa fuente ser capaz de acceder y modificar tanto los datos SQL como la estructura subyacente de la base de datos.

- 3.** ¿Qué hace el precompilador con el archivo del programa?

El precompilador retira las instrucciones SQL del código del lenguaje host y las reemplaza con llamadas a las instrucciones SQL.

- 4.** ¿Cuáles archivos son creados por un precompilador SQL?

- A** Un archivo para las funciones CLI
- B** Un archivo para el lenguaje host
- C** Un archivo para las llamadas CLI
- D** Un archivo para las instrucciones SQL incrustadas

Las respuestas correctas son B y D.

- 5.** ¿Qué cláusula se utiliza en una instrucción MODULE para especificar el lenguaje de programación host?

La cláusula LANGUAGE.

- 6.** ¿Qué prefijo deberán utilizar las instrucciones SQL incrustadas cuando esas instrucciones son incrustadas en el lenguaje de programación MUMPS?

- A** &SQL(
- B** EXEC SQL

C START-EXEC

D Las instrucciones incrustadas en MUMPS no requieren un prefijo

La respuesta correcta es **A**.

- 7.** Un(a) _____ es un tipo de parámetro que es declarado dentro del lenguaje host y luego es referenciado dentro de la instrucción SQL incrustada.

Variable host.

- 8.** ¿Qué instrucción deberá utilizarse al principio de la sección de instrucción para las variables host?

BEGIN DECLARE SECTION.

- 9.** ¿Qué prefijo deberá proporcionarse para una variable host cuando esté incluida en una instrucción SQL?

A Signo de interrogación

B Signo de unión &

C Punto y coma

D Dos puntos

La respuesta correcta es **D**.

- 10.** Se planea incrustar instrucciones SQL en el programa host. Se quieren declarar diferentes variables host para ser utilizadas en las instrucciones SQL. ¿Qué instrucción SQL deberá utilizarse para finalizar la sección de instrucción del programa?

A TERMINATE DECLARE SECTION

B END DECLARE SECTION

C TERMINATE DECLARATIONS

D END DECLARATIONS

La respuesta correcta es **B**.

- 11.** ¿Qué puede provocar que ocurra una incongruencia en la impedancia cuando se pasa una variable de un programa host a una instrucción SQL?

Las diferencias entre el tipo de datos del lenguaje host y el tipo de datos SQL.

- 12.** ¿Qué tipo de instrucción SELECT puede utilizarse en SQL incrustado cuando se recupera solamente una fila de datos?

Una instrucción SELECT de instancia única.

- 13.** Un(a) _____ es un tipo de variable que especifica si una variable de datos asociada contiene un valor nulo.

Una variable host de indicador.

- 14. ¿Qué instrucción puede utilizarse en SQL incrustado para proporcionarle al programa host la información de excepción y de advertencia?**

A WHENEVER
B INTO
C CAST
D PROCEDURE

La respuesta correcta es **A**.

- 15. Un(a) _____ es una colección de instrucciones SQL autocontenidas que están separadas de un lenguaje de programación host pero que pueden ser convocadas desde dentro de ese lenguaje.**

Módulo cliente SQL.

- 16. ¿Qué indicadores de asignación deben establecerse para poder ejecutar una instrucción SQL a través de una interfaz de programación CLI?**

De ambiente, de conexión y de instrucción.

- 17. ¿Cuántas instrucciones SQL pueden incluirse en un procedimiento en un módulo cliente de SQL?**

Una.

- 18. ¿Qué función deberá utilizarse para establecer un indicador de conexión CLI?**

A ExecDirect()
B Connect()
C Prepare()
D AllocHandle()

La respuesta correcta es **D**.

- 19. Se está asignando un indicador de ambiente dentro de un programa C y asignando el indicador con la variable host henv. ¿Qué instrucción de función deberá utilizarse?**

Deberá utilizarse la siguiente instrucción de función:

```
SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
```

- 20. Se está creando la siguiente instrucción de función Prepare() en el programa host:**

```
SQLPrepare ( hstmt, "SELECT ID_CD, TITULO_CD, EN_EXISTENCIA FROM  
DISCOS_COMPACTOS  
WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
```

¿Cuántas instrucciones de función BindCol() deberán crearse?

A Una
B Dos

- C** Tres
- D** Cuatro

La respuesta correcta es **C**.

- 21. ¿Qué función CLI deberá utilizarse si se quiere ejecutar una instrucción SQL en un solo paso?**

ExecDirect()

Capítulo 18: Trabajar con datos XML

- 1. ¿Qué es XML?**

XML es un lenguaje de marcado de propósito general utilizado para describir documentos.

- 2. ¿Cuáles de los siguientes son usos comunes de XML?**

- A** Desplegar datos de la base de datos en una página Web
- B** La creación de páginas Web estáticas
- C** La transmisión de datos de una base de datos a otro partido
- D** El reforzamiento de reglas de negocios en documentos

Las respuestas correctas son **A** y **C**.

- 3. ¿Qué tanto varían las bases de datos SQL y los documentos XML en términos de estructura de datos?**

XML define una secuencia y una estructura jerárquica de árbol, no así SQL.

- 4. Si dos organizaciones están usando XML, ¿significa eso que ellos tienen una forma estándar de intercambiar datos sin tener que crear software de interfaz?**

No necesariamente. Las dos organizaciones deben utilizar una definición común para los elementos dentro de los documentos XML a ser intercambiados antes de que puedan procesarlos sin la necesidad de realizar ninguna interpretación o conversión.

- 5. ¿Cuáles de los siguientes son modificadores de tipo válidos para el tipo de datos XML?**

- A** DOCUMENT
- B** SEQUENCE
- C** SQLXML
- D** CONTENT

Las respuestas correctas son **A**, **B** y **D**.

- 6. ¿Cuáles son los modificadores secundarios de tipo válidos para el modificador de tipo SEQUENCE?**

El modificador de tipo SEQUENCE no puede tener un modificador secundario de tipo.

7. ¿Cuál de las siguientes funciones SQL/XML crea un elemento basado en una columna de una tabla?

- A** XMLQUERY
- B** XMLELEMENT
- C** XMLFOREST
- D** XMLDOCUMENT
- E** XMLPARSE

La respuesta correcta es **C**.

8. ¿Cuál tipo de esquema XML es trazado desde el tipo de datos SQL NUMERIC?

- A** xs:integer
- B** xs:float
- C** xs:decimal
- D** xs:double

La respuesta correcta es **C**.

9. ¿Cuál tipo de esquema XML es trazado desde el tipo de datos SQL DATE?

- A** xs:dateTime
- B** xdt:yearMonthDuration
- C** xs:time
- D** xs:date
- E** xdt:dateTimeDuration

La respuesta correcta es **D**.

10. ¿Cuáles son las dos formas en que los valores nulos de la base de datos pueden ser representados por SQL/XML?

- A** Absent document
- B** Absent element
- C** xsi:null= "true"
- D** xsi:nil= "true"
- E** <elementname=nil>

La respuesta correcta es **D**.



Apéndice **B**

Palabras clave
de SQL:2006

El estándar SQL:2006 define un conjunto de palabras clave reservadas y palabras clave no reservadas que se utilizan dentro de las declaraciones SQL. No es posible utilizar palabras clave reservadas como identificadores. Además, generalmente es una buena idea evitar utilizar palabras clave no reservadas. Nótese que el estándar SQL advierte que no garantiza aquellas palabras que pudieran ser agregadas al estándar en el futuro. Como resultado, un identificador que se utilice en una base de datos actual pudiera no ser utilizable en versiones futuras de SQL. Se pueden evitar los conflictos con palabras clave reservadas en el futuro al agregar un dígito o un guión bajo a los identificadores, especialmente a aquellos comprimidos en una sola palabra, y al no comenzar un identificador con `current_`, `session_`, `system_`, or `timezone_` o terminar el identificador con `_length`.

NOTA

Varias implementaciones SQL incluyen palabras clave adicionales y palabras reservadas que no pueden ser utilizadas como identificadores. Como siempre, revise la documentación de su producto.

Palabras clave reservadas de SQL

La tabla B-1 lista las palabras clave reservadas de SQL.

ABS	ALL	ALLOCATE	ALTER
AND	ANY	ARE	ARRAY
AS	ASENSITIVE	ASYMMETRIC	AT
ATOMIC	AUTHORIZATION	AVG	BEGIN
BETWEEN	BIGINT	BINARY	BLOB
BOOLEAN	BOTH	BY	CALL
CALLED	CARDINALITY	CASCADE	CASE
CAST	CEIL	CEILING	CHAR
CHAR_LENGTH	CHARACTER	CHARACTER_LENGTH	CHECK
CLOB	CLOSE	COALESCE	COLLATE
COLLECT	COLUMN	COMMIT	CONDITION
CONNECT	CONSTRAINT	CONVERT	CORR
CORRESPONDING	COUNT	COVAR_POP	COVAR_SAMP
CREATE	CROSS	CUBE	CUME_DIST
CURRENT	CURRENT_DATE	CURRENT_DEFAULT_TRANSFORM_GROUP	CURRENT_PATH
CURRENT_ROLE	CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_TRANSFORM_GROUP_FOR_TYPE

Tabla B-1 Palabras clave reservadas de SQL.

CURRENT_USER	CURSOR	CYCLE	DATE
DAY	DEALLOCATE	DEC	DECIMAL
DECLARE	DEFAULT	DELETE	DENSE_RANK
DEREF	DESCRIBE	DETERMINISTIC	DISCONNECT
DISTINCT	DOUBLE	DROP	DYNAMIC
EACH	ELEMENT	ELSE	END
END-EXEC	ESCAPE	EVERY	EXCEPT
EXEC	EXECUTE	EXISTS	EXP
EXTERNAL	EXTRACT	FALSE	FETCH
FILTER	FLOAT	FLOOR	FOR
FOREIGN	FREE	FROM	FULL
FUNCTION	FUSION	GET	GLOBAL
GRANT	GROUP	GROUPING	HAVING
HOLD	HOURL	IDENTITY	IN
INDICATOR	INNER	INOUT	INSENSITIVE
INSERT	INT	INTEGER	INTERSECT
INTERSECTION	INTERVAL	INTO	IS
JOIN	LANGUAGE	LARGE	LATERAL
LEADING	LEFT	LIKE	LN
LOCAL	LOCALTIME	LOCALTIMESTAMP	LOWER
MATCH	MAX	MEMBER	MERGE
METHOD	MIN	MINUTE	MOD
MODIFIES	MODULE	MONTH	MULTISET
NATIONAL	NATURAL	NCHAR	NCLOB
NEW	NO	NONE	NORMALIZE
NOT	NULL	NULLIF	NUMERIC
OCTET_LENGTH	OF	OLD	ON
ONLY	OPEN	OR	ORDER
OUT	OUTER	OVER	OVERLAPS
OVERLAY	PARAMETER	PARTITION	PERCENT_RANK
PERCENTILE_CONT	PERCENTILE_DISC	POSITION	POWER
PRECISION	PREPARE	PRIMARY	PROCEDURE
RANGE	RANK	READS	REAL

Tabla B-1 Palabras clave reservadas de SQL (continuación).

RECURSIVE	REF	REFERENCES	REFERENCING
REGR_AVGX	REGR_AVGY	REGR_COUNT	REGR_INTERCEPT
REGR_R2	REGR_SLOPE	REGR_SXX	REGR_SXY
REGR_SYY	RELEASE	RESULT	RETURN
RETURNS	REVOKE	RIGHT	ROLLBACK
ROLLUP	ROW	ROW_NUMBER	ROWS
SAVEPOINT	SCOPE	SCROLL	SEARCH
SECOND	SELECT	SENSITIVE	SESSION_USER
SET	SIMILAR	SMALLINT	SOME
SPECIFIC	SPECIFICTYPE	SQL	SQLEXCEPTION
SQLSTATE	SQLWARNING	SQRT	START
STATIC	STDDEV_POP	STDDEV_SAMP	SUBMULTISET
SUBSTRING	SUM	SYMMETRIC	SYSTEM
SYSTEM_USER	TABLE	TABLESAMPLE	THEN
TIME	TIMESTAMP	TIMEZONE_HOUR	TIMEZONE_MINUTE
TO	TRAILING	TRANSLATE	TRANSLATION
TREAT	TRIGGER	TRIM	TRUE
UESCAPE	UNION	UNIQUE	UNKNOWN
UNNEST	UPDATE	UPPER	USER
USING	VALUE	VALUES	VAR_POP
VAR_SAMP	VARCHAR	VARYING	WHEN
WHENEVER	WHERE	WIDTH_BUCKET	WINDOW
WITH	WITHIN	WITHOUT	YEAR

Tabla B-1 Palabras clave reservadas de SQL (continuación).

Palabras clave no reservadas de SQL

La tabla B-2 lista las palabras clave no reservadas de SQL.

ABSOLUTE	ACTION	ADA
ADD	ADMIN	AFTER
ALWAYS	ASC	ASSERTION
ASSIGNMENT	ATTRIBUTE	ATTRIBUTES
BEFORE	BERNOULLI	BREADTH

Tabla B-2 Palabras clave no reservadas de SQL.

CASCADE	CATALOG	CATALOG_NAME
CHAIN	CHARACTER_SET_CATALOG	CHARACTER_SET_NAME
CHARACTER_SET_SCHEMA	CHARACTERISTICS	CHARACTERS
CLASS_ORIGIN	COBOL	COLLATION
COLLATION_CATALOG	COLLATION_NAME	COLLATION_SCHEMA
COLUMN_NAME	COMMAND_FUNCTION	COMMAND_FUNCTION_CODE
COMMITTED	CONDITION_NUMBER	CONNECTION
CONNECTION_NAME	CONSTRAINT_CATALOG	CONSTRAINT_NAME
CONSTRAINT_SCHEMA	CONSTRAINTS	CONSTRUCTOR
CONTAINS	CONTINUE	CURSOR_NAME
DATA	DATETIME_INTERVAL_CODE	DATETIME_INTERVAL_PRECISION
DEFAULTS	DEFERRABLE	DEFERRED
DEFINED	DEFINER	DEGREE
DEPTH	DERIVED	DESC
DESCRIPTOR	DIAGNOSTICS	DISPATCH
DOMAIN	DYNAMIC_FUNCTION	DYNAMIC_FUNCTION_CODE
EQUALS	EXCEPTION	EXCLUDE
EXCLUDING	FINAL	FIRST
FOLLOWING	FORTRAN	FOUND
GENERAL	GENERATED	GO
GOTO	GRANTED	IMMEDIATE
IMPLEMENTATION	INCLUDING	INCREMENT
INITIALLY	INPUT	INSTANCE
INSTANTIABLE	INVOKER	ISOLATION
KEY	KEY_MEMBER	KEY_TYPE
LAST	LENGTH	LEVEL
LOCATOR	MAP	MATCHED
MAXVALUE	MESSAGE_LENGTH	MESSAGE_OCTET_LENGTH
MESSAGE_TEXT	MINVALUE	MORE
MUMPS	NAME	NAMES
NESTING	NEXT	NORMALIZED
NULLABLE	NULLS	NUMBER
OBJECT	OCTETS	OPTION

Tabla B-2 Palabras clave no reservadas de SQL (*continuación*).

OPTIONS	ORDERING	ORDINALITY
OTHERS	OUTPUT	OVERRIDING
PAD	PARAMETER_MODE	PARAMETER_NAME
PARAMETER_ORDINAL_POSITION	PARAMETER_SPECIFIC_CATALOG	PARAMETER_SPECIFIC_NAME
PARAMETER_SPECIFIC_SCHEMA	PARTIAL	PASCAL
PATH	PLACING	PLI
PRECEDING	PRESERVE	PRIOR
PRIVILEGES	PUBLIC	READ
RELATIVE	REPEATABLE	RESTART
RESTRICT	RETURNED_CARDINALITY	RETURNED_LENGTH
RETURNED_OCTET_LENGTH	RETURNED_SQLSTATE	ROLE
ROUTINE	ROUTINE_CATALOG	ROUTINE_NAME
ROUTINE_SCHEMA	ROW_COUNT	SCALE
SCHEMA	SCHEMA_NAME	SCOPE_CATALOG
SCOPE_NAME	SCOPE_SCHEMA	SECTION
SECURITY	SELF	SEQUENCE
SERIALIZABLE	SERVER_NAME	SESSION
SETS	SIMPLE	SIZE
SOURCE	SPACE	SPECIFIC_NAME
STATE	STATEMENT	STRUCTURE
STYLE	SUBCLASS_ORIGIN	TABLE_NAME
TEMPORARY	TIES	TOP_LEVEL_COUNT
TRANSACTION	TRANSACTION_ACTIVE	TRANSACTIONS_COMMITTED
TRANSACTIONS_ROLLED_BACK	TRANSFORM	TRANSFORMS
TRIGGER_CATALOG	TRIGGER_NAME	TRIGGER_SCHEMA
TYPE	UNBOUNDED	UNCOMMITTED
UNDER	UNNAMED	USAGE
USER_DEFINED_TYPE_CATALOG	USER_DEFINED_TYPE_CODE	USER_DEFINED_TYPE_NAME
USER_DEFINED_TYPE_SCHEMA	VIEW	WORK
WRITE	ZONE	

Tabla B-2 Palabras clave no reservadas de SQL (continuación).

Apéndice C

Código SQL utilizado
en los ejercicios
Pruebe esto

En los ejercicios que se encuentran a lo largo de todo el libro se crearon muchas instrucciones SQL que permitieron definir los objetos de la base de datos en la base de datos INVENTARIO, modificar esos objetos, insertar datos en las tablas que se crearon, recuperar esos datos, y actualizar y eliminar los datos. Las instrucciones se incluyen aquí para que pueda observarse la progresión de esas instrucciones según se iba avanzando en los ejercicios, y también para que sirvan de referencia según sea necesario en caso de requerir la repetición de ciertos elementos de un ejercicio. Además, se han proporcionado las instrucciones SQL (en forma consolidada) utilizadas para crear la base de datos INVENTARIO y llenar las tablas con datos. Puede encontrarse que, en la forma en que cada usuario trabaje en el libro, se necesitará ser capaz de volver a crear la base de datos y de dejarla en un estado consistente. Al utilizar el código consolidado, simplemente es posible crear los objetos de la base de datos y llenar las tablas tan frecuentemente como sea necesario, sin tener que juntar los elementos desde los diferentes ejercicios. Puede descargar los archivos desde www.mcgraw-hill-educacion.com. Haga una búsqueda por autor, título o ISBN. También los puede descargar del sitio en inglés: <http://www.mhprofessional.com>. Diríjase a Computing y luego Downloads.

NOTA

Las instrucciones SQL están escritas en SQL puro. Por lo tanto, algunas implementaciones SQL pudieran requerir que se modifiquen las instrucciones para adecuarse a las convenciones de esa implementación en particular. Asegúrese de revisar la documentación de su producto.

Código SQL por cada ejercicio

Las instrucciones SQL se presentan aquí de acuerdo con el orden en que los proyectos fueron presentados en el libro. Puede referirse a esas instrucciones según sea necesario para utilizarlas para volver a crear los ejercicios o para utilizarlas como una base para otros ejercicios. Si desea volver a crear la base de datos INVENTARIO y quiere asegurarse de que se están incluyendo todos los elementos necesarios, véase la sección “La base de datos INVENTARIO”, más adelante en este apéndice.

Pruebe esto 3-1 Creación de tablas en SQL

```
CREATE TABLE DISCOS_COMPACTOS
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) ,
  ID_DISQUERA        INT );

CREATE TABLE DISQUERAS_CD
( ID_DISQUERA        INT,
  NOMBRE_COMPANIA    VARCHAR(60) );

CREATE TABLE TIPOS_MUSICA
( ID_TIPO            INT,
  NOMBRE_TIPO        VARCHAR(20) );
```

Pruebe esto 3-2 Modifique y elimine tablas en SQL

```
CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  ID_TIPO            INT );

DROP TABLE TIPOS_DISCO_COMPACTO CASCADE;

CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60),
  ID_TIPO            INT );

ALTER TABLE TIPOS_DISCO_COMPACTO
  DROP COLUMN TITULO_CD CASCADE;
```

Pruebe esto 4-1 Añada restricciones NOT NULL, únicas y referenciales

```
DROP TABLE DISCOS_COMPACTOS      CASCADE;
DROP TABLE TIPOS_DISCO_COMPACTO  CASCADE;
DROP TABLE TIPOS_MUSICA          CASCADE;
DROP TABLE DISQUERAS_CD         CASCADE;

CREATE TABLE TIPOS_MUSICA
( ID_TIPO            INT,
  NOMBRE_TIPO        VARCHAR(20)      NOT NULL,
  CONSTRAINT UN_NOMBRE_TIPO  UNIQUE (NOMBRE_TIPO),
  CONSTRAINT PK_TIPOS_MUSICA PRIMARY KEY (ID_TIPO) );

CREATE TABLE DISQUERAS_CD
( ID_DISQUERA        INT,
  NOMBRE_COMPañIA    VARCHAR(60)      DEFAULT 'Independiente' NOT NULL,
  CONSTRAINT PK_DISQUERAS_CD PRIMARY KEY (ID_DISQUERA) );

CREATE TABLE DISCOS_COMPACTOS
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60)      NOT NULL,
  ID_DISQUERA        INT              NOT NULL,
  CONSTRAINT PK_DISCOS_COMPACTOS PRIMARY KEY (ID_DISCO_COMPACTO),
  CONSTRAINT FK_ID_DISQUERA FOREIGN KEY (ID_DISQUERA) REFERENCES
DISQUERAS_CD );

CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,
  ID_TIPO_MUSICA     INT,
  CONSTRAINT PK_TIPOS_DISCO_COMPACTO
```

```
PRIMARY KEY ( ID_DISCO_COMPACTO, ID_TIPO_MUSICA),
CONSTRAINT FK_ID_DISCO_COMPACTO_01
FOREIGN KEY (ID_DISCO_COMPACTO) REFERENCES DISCOS_COMPACTOS,
CONSTRAINT FK_ID_TIPO_MUSICA
FOREIGN KEY (ID_TIPO_MUSICA) REFERENCES TIPOS_MUSICA );

CREATE TABLE ARTISTAS
( ID_ARTISTA INT,
  NOMBRE_ARTISTA          VARCHAR(60)                NOT NULL,
  LUGAR_DE_NACIMIENTO     VARCHAR(60) DEFAULT 'Desconocido' NOT NULL,
  CONSTRAINT PK_ARTISTAS PRIMARY KEY (ID_ARTISTA) );

CREATE TABLE CDS_ARTISTA
( ID_ARTISTA      INT,
  ID_DISCO_COMPACTO INT,
  CONSTRAINT PK_CDS_ARTISTA PRIMARY KEY ( ID_ARTISTA, ID_DISCO_COMPACTO ),
  CONSTRAINT FK_ID_ARTISTA FOREIGN KEY (ID_ARTISTA) REFERENCES ARTISTAS,
  CONSTRAINT FK_ID_DISCO_COMPACTO_02 FOREIGN KEY (ID_DISCO_COMPACTO)
REFERENCES DISCOS_COMPACTOS );
```

Pruebe esto 4-2 Añada una restricción CHECK

```
ALTER TABLE DISCOS_COMPACTOS
ADD COLUMN EN_EXISTENCIA INT NOT NULL;

ALTER TABLE DISCOS_COMPACTOS
ADD CONSTRAINT CK_EN_EXISTENCIA CHECK ( EN_EXISTENCIA > 0 AND EN_
EXISTENCIA < 50 );
```

Pruebe esto 5-1 Añada vistas a su base de datos

```
CREATE VIEW CDS_EN_EXISTENCIA AS
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA > 10 WITH CHECK OPTION;

CREATE VIEW EDITORES_CD ( TITULO_CD, EDITOR ) AS
SELECT DISCOS_COMPACTOS.TITULO_CD, DISQUERAS_CD.NOMBRE_COMPañIA
FROM DISCOS_COMPACTOS, DISQUERAS_CD
WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA
AND DISQUERAS_CD.ID_DISQUERA = 5403 OR DISQUERAS_CD.ID_DISQUERA =
5402;

DROP VIEW EDITORES_CD;

CREATE VIEW EDITORES_CD ( TITULO_CD, EDITOR ) AS
SELECT DISCOS_COMPACTOS.TITULO_CD, DISQUERAS_CD.NOMBRE_COMPañIA
FROM DISCOS_COMPACTOS, DISQUERAS_CD
WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA;
```

Pruebe esto 6-1 Gestión de roles y privilegios

```
CREATE ROLE MRKT;

CREATE ROLE PERSONAL_VENTAS;

GRANT SELECT ON CDS_EN_EXISTENCIA TO PUBLIC;

GRANT SELECT, INSERT, UPDATE(TITULO_CD) ON DISCOS_COMPACTOS
  TO PERSONAL_VENTAS WITH GRANT OPTION;

GRANT PERSONAL_VENTAS TO MRKT;

REVOKE SELECT ON CDS_EN_EXISTENCIA FROM PUBLIC CASCADE;

REVOKE ALL PRIVILEGES ON DISCOS_COMPACTOS FROM PERSONAL_VENTAS CASCADE;

REVOKE PERSONAL_VENTAS FROM MRKT CASCADE;

DROP ROLE MRKT;

DROP ROLE PERSONAL_VENTAS;
```

Pruebe esto 7-1 Consultar la base de datos INVENTORY

NOTA

Las instrucciones INSERT utilizadas para este proyecto están enlistadas al final de la sección “La base de datos INVENTARIO”, más adelante en este apéndice.

```
SELECT *
  FROM ARTISTAS;

SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS;

SELECT *
  FROM CDS_EN_EXISTENCIA;

SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE EN_EXISTENCIA > 10 AND EN_EXISTENCIA < 30;

SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL_EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 GROUP BY ID_DISQUERA;

SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL_EN_EXISTENCIA
```



```
FROM DISCOS_COMPACTOS
GROUP BY ID_DISQUERA
HAVING SUM(EN_EXISTENCIA) > 10;
```

```
SELECT *
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA > 10
ORDER BY TITULO_CD DESC;
```

Pruebe esto 8-1 Modificar datos SQL

```
INSERT INTO DISQUERAS_CD
VALUES ( 837, 'DRG Records' );
```

```
INSERT INTO DISCOS_COMPACTOS
VALUES ( 116, 'Ann Hampton Callaway', 836, 14 );
```

```
INSERT INTO DISCOS_COMPACTOS
( ID_DISCO_COMPACTO, TITULO_CD, ID_DISQUERA, EN_EXISTENCIA )
VALUES ( 117, 'Rhythm Country and Blues', 832, 21 );
```

```
UPDATE DISCOS_COMPACTOS
SET EN_EXISTENCIA = 25
WHERE ID_DISCO_COMPACTO = 117;
```

```
UPDATE DISCOS_COMPACTOS
SET ID_DISQUERA =
( SELECT ID_DISQUERA
  FROM DISQUERAS_CD
  WHERE NOMBRE_COMPAÑIA = 'DRG Records' )
WHERE ID_DISCO_COMPACTO = 116;
```

```
SELECT *
FROM DISCOS_COMPACTOS
WHERE ID_DISCO_COMPACTO = 116
OR ID_DISCO_COMPACTO = 117;
```

```
DELETE FROM DISCOS_COMPACTOS
WHERE ID_DISCO_COMPACTO = 116
OR ID_DISCO_COMPACTO = 117;
```

```
DELETE FROM DISQUERAS_CD
WHERE ID_DISQUERA = 837;
```

Pruebe esto 9-1**Utilizar predicados
en instrucciones SQL**

```

SELECT ID_TIPO, NOMBRE_TIPO
FROM TIPOS_MUSICA
WHERE ID_TIPO = 11
      OR ID_TIPO = 12;

SELECT NOMBRE_ARTISTA, LUGAR_DE_NACIMIENTO
FROM ARTISTAS
WHERE NOMBRE_ARTISTA <> 'Patsy Cline'
      AND NOMBRE_ARTISTA <> 'Bing Crosby';

SELECT ID_ARTISTA, NOMBRE_ARTISTA
FROM ARTISTAS
WHERE ID_ARTISTA > 2004
      AND ID_ARTISTA < 2014;

SELECT ID_ARTISTA, NOMBRE_ARTISTA
FROM ARTISTAS
WHERE ID_ARTISTA BETWEEN 2004 AND 2014;

SELECT *
FROM ARTISTAS
WHERE LUGAR_DE_NACIMIENTO IS NULL;

SELECT *
FROM ARTISTAS
WHERE LUGAR_DE_NACIMIENTO IS NOT NULL;

SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE TITULO_CD LIKE ('%Greatest%')
      OR TITULO_CD LIKE ('%Best%');

SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE TITULO_CD NOT LIKE ('%Greatest%')
      AND TITULO_CD NOT LIKE ('%Best%');

```

Pruebe esto 9-2**Utilizar subconsultas en predicados**

```

SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE ID_DISQUERA IN
      ( SELECT ID_DISQUERA
        FROM DISQUERAS_CD
        WHERE NOMBRE_COMPANIA = 'Decca Record Company' );

```

```
SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_COMPACTOS
WHERE EXISTS
  ( SELECT ID_DISQUERA FROM DISQUERAS_CD
    WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA
      AND ID_DISQUERA > 830 );

SELECT ID_DISQUERA, NOMBRE_COMPañIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ANY
  ( SELECT ID_DISQUERA
    FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 20 );

SELECT ID_DISQUERA, NOMBRE_COMPañIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ALL
  ( SELECT ID_DISQUERA
    FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 20 );

SELECT ID_DISQUERA, NOMBRE_COMPañIA
FROM DISQUERAS_CD
WHERE ID_DISQUERA = ALL
  ( SELECT ID_DISQUERA
    FROM DISCOS_COMPACTOS
      WHERE EN_EXISTENCIA > 40 );
```

Pruebe esto 10-1

Utilizar funciones y expresiones de valor

```
SELECT COUNT(DISTINCT NOMBRE_ARTISTA) AS ARTISTAS
FROM ARTISTAS;

SELECT MIN(EN_EXISTENCIA) AS MIN_EXISTENCIA
FROM DISCOS_COMPACTOS;

SELECT ID_DISQUERA, SUM(EN_EXISTENCIA) AS TOTAL
FROM DISCOS_COMPACTOS
GROUP BY ID_DISQUERA;

SELECT NOMBRE_ARTISTA,
       SUBSTRING(LUGAR_DE_NACIMIENTO FROM 1 FOR 8) AS LUGAR_NACIMIENTO
FROM ARTISTAS;

SELECT UPPER(TITULO_CD) AS NOMBRE_CD
FROM DISCOS_COMPACTOS;

SELECT TITULO_CD, EN_EXISTENCIA,
```

```
(EN_EXISTENCIA * 2) AS DOUBLED, (EN_EXISTENCIA * 3) AS TRIPLED
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA < 25;
```

```
SELECT TITULO_CD, EN_EXISTENCIA, EN_PEDIDO =
CASE
  WHEN EN_EXISTENCIA < 10 THEN EN_EXISTENCIA * 2
  WHEN EN_EXISTENCIA BETWEEN 10 AND 15 THEN EN_EXISTENCIA + 3
  ELSE EN_EXISTENCIA
END
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA < 20;
```

```
SELECT ID_TIPO, CAST(NOMBRE_TIPO AS CHAR(20)) AS CHAR_TYPE
FROM TIPOS_MUSICA;
```

Pruebe esto 11-1

Consultar múltiples tablas

```
SELECT * FROM ARTISTAS a, CDS_ARTISTA c
WHERE a.ID_ARTISTA = c.ID_ARTISTA;
```

```
SELECT d.TITULO_CD, a.NOMBRE_ARTISTA, a.LUGAR_DE_NACIMIENTO
FROM ARTISTAS a, CDS_ARTISTA c, DISCOS_COMPACTOS d
WHERE a.ID_ARTISTA = c.ID_ARTISTA
AND d.ID_DISCO_COMPACTO = c.ID_DISCO_COMPACTO;
```

```
SELECT d.TITULO_CD, a.NOMBRE_ARTISTA, a.LUGAR_DE_NACIMIENTO
FROM ARTISTAS a CROSS JOIN CDS_ARTISTA c CROSS JOIN DISCOS_COMPACTOS d
WHERE a.ID_ARTISTA = c.ID_ARTISTA
AND d.ID_DISCO_COMPACTO = c.ID_DISCO_COMPACTO;
```

```
SELECT d.TITULO_CD, t.NOMBRE_TIPO
FROM DISCOS_COMPACTOS d JOIN TIPOS_DISCO_COMPACTO dt
ON d.ID_DISCO_COMPACTO = dt.ID_DISCO_COMPACTO
JOIN TIPOS_MUSICA t
ON dt.ID_TIPO_MUSICA = t.ID_TIPO;
```

```
SELECT d.TITULO_CD, t.NOMBRE_TIPO
FROM DISCOS_COMPACTOS d FULL JOIN TIPOS_DISCO_COMPACTO dt
ON d.ID_DISCO_COMPACTO = dt.ID_DISCO_COMPACTO
FULL JOIN TIPOS_MUSICA t
ON dt.ID_TIPO_MUSICA = t.ID_TIPO;
```

Pruebe esto 12-1

Trabajar con subconsultas

```
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE ID_DISQUERA IN
```

```
( SELECT ID_DISQUERA
  FROM DISQUERAS_CD
 WHERE NOMBRE_COMPañIA = 'MCA Records' );

SELECT NOMBRE_COMPañIA
  FROM DISQUERAS_CD l
 WHERE EXISTS
    ( SELECT *
      FROM DISCOS_COMPACTOS d
     WHERE l.ID_DISQUERA = d.ID_DISQUERA
       AND TITULO_CD = 'Out of Africa' );

SELECT NOMBRE_COMPañIA
  FROM DISQUERAS_CD
 WHERE ID_DISQUERA = ANY
    ( SELECT ID_DISQUERA
      FROM DISCOS_COMPACTOS
     WHERE EN_EXISTENCIA > 30 );

SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS
 WHERE ID_DISQUERA =
    ( SELECT ID_DISQUERA
      FROM DISQUERAS_CD
     WHERE NOMBRE_COMPañIA = 'Capitol Records' );

SELECT TITULO_CD, EN_EXISTENCIA
  FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
 WHERE d.ID_DISQUERA = l.ID_DISQUERA
    AND NOMBRE_COMPañIA = 'Capitol Records';

SELECT NOMBRE_ARTISTA
  FROM ARTISTAS
 WHERE ID_ARTISTA IN
    ( SELECT ID_ARTISTA
      FROM CDS_ARTISTA
     WHERE ID_DISCO_COMPACTO IN
        ( SELECT ID_DISCO_COMPACTO
          FROM DISCOS_COMPACTOS
         WHERE TITULO_CD = 'Past Light' ) );

SELECT TITULO_CD, NOMBRE_TIPO
  FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
 WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
    AND t.ID_TIPO_MUSICA = m.ID_TIPO
    AND TITULO_CD = 'Kojiki';

UPDATE TIPOS_DISCO_COMPACTO
  SET ID_TIPO_MUSICA =
```

```

( SELECT ID_TIPO
  FROM TIPOS_MUSICA
  WHERE NOMBRE_TIPO = 'Classical' )
WHERE ID_DISCO_COMPACTO =
( SELECT ID_DISCO_COMPACTO
  FROM DISCOS_COMPACTOS
  WHERE TITULO_CD = 'Kojiki' )
AND ID_TIPO_MUSICA =
( SELECT ID_TIPO
  FROM TIPOS_MUSICA
  WHERE NOMBRE_TIPO = 'New Age' );

SELECT TITULO_CD, NOMBRE_TIPO
FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
AND t.ID_TIPO_MUSICA = m.ID_TIPO
AND TITULO_CD = 'Kojiki';

UPDATE TIPOS_DISCO_COMPACTO
SET ID_TIPO_MUSICA =
( SELECT ID_TIPO
  FROM TIPOS_MUSICA
  WHERE NOMBRE_TIPO = 'New Age' )
WHERE ID_DISCO_COMPACTO =
( SELECT ID_DISCO_COMPACTO
  FROM DISCOS_COMPACTOS
  WHERE TITULO_CD = 'Kojiki' )
AND ID_TIPO_MUSICA =
( SELECT ID_TIPO
  FROM TIPOS_MUSICA
  WHERE NOMBRE_TIPO = 'Classical' );

SELECT TITULO_CD, NOMBRE_TIPO
FROM DISCOS_COMPACTOS d, TIPOS_DISCO_COMPACTO t, TIPOS_MUSICA m
WHERE d.ID_DISCO_COMPACTO = t.ID_DISCO_COMPACTO
AND t.ID_TIPO_MUSICA = m.ID_TIPO
AND TITULO_CD = 'Kojiki';

```

Pruebe esto 13-1

Crear procedimientos invocados por SQL

```

CREATE PROCEDURE OBTENER_CD_ARTISTAS ( )
SELECT cd.TITULO_CD, a.NOMBRE_ARTISTA
FROM DISCOS_COMPACTOS cd, CDS_ARTISTA ac, ARTISTAS a
WHERE cd.ID_DISCO_COMPACTO = ac.ID_DISCO_COMPACTO
AND ac.ID_ARTISTA = a.ID_ARTISTA;

```

```
CALL OBTENER_CD_ARTISTAS( );

DROP PROCEDURE OBTENER_CD_ARTISTAS CASCADE;

SELECT cd.TITULO_CD, a.NOMBRE_ARTISTA
  FROM DISCOS_COMPACTOS cd, CDS_ARTISTA ac, ARTISTAS a
 WHERE cd.ID_DISCO_COMPACTO = ac.ID_DISCO_COMPACTO
       AND ac.ID_ARTISTA      = a.ID_ARTISTA
       AND cd.TITULO_CD       = p_CD;

CALL OBTENER_CD_ARTISTAS('Fundamental');

CREATE PROCEDURE OBTENER_CANTIDAD_CD( )
BEGIN
  DECLARE v_En_Existencia INT;
  SET v_En_Existencia = ( SELECT AVG(EN_EXISTENCIA)
                        FROM DISCOS_COMPACTOS );
  SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
   WHERE EN_EXISTENCIA < v_En_Existencia;
END;

CALL OBTENER_CANTIDAD_CD( );
```

Pruebe esto 13-2 Crear funciones invocadas por SQL

```
CREATE FUNCTION DISQUERA_CD ( p_CD VARCHAR(60) )
  RETURNS VARCHAR(60)
  BEGIN
    RETURN ( SELECT NOMBRE_COMPañIA
              FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
             WHERE d.ID_DISQUERA = l.ID_DISQUERA
                   AND TITULO_CD = p_CD );
  END;

SELECT TITULO_CD, NOMBRE_COMPañIA
  FROM DISCOS_COMPACTOS d, DISQUERAS_CD l
 WHERE d.ID_DISQUERA = l.ID_DISQUERA
       AND NOMBRE_COMPañIA = DISQUERA_CD ('Blues on the Bayou');

DROP FUNCTION DISQUERA_CD CASCADE;
```

Pruebe esto 14-1 Crear activadores SQL

```
CREATE TABLE ARTIST_LOG
  ( ACTION_TYPE CHAR(6),
    ID_ARTISTA INT,
```

```

MOD_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP );

CREATE TRIGGER INSERT_LOG
AFTER INSERT ON ARTISTAS
REFERENCING NEW ROW AS Nueva
FOR EACH ROW
BEGIN ATOMIC
    INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
    VALUES ( 'INSERT', Nueva.ID_ARTISTA );
END;

CREATE TRIGGER ACTUALIZAR_REGISTRO
AFTER UPDATE ON ARTISTAS
REFERENCING NEW ROW AS Nueva
FOR EACH ROW
BEGIN ATOMIC
    INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
    VALUES ( 'UPDATE', Nueva.ID_ARTISTA );
END;

CREATE TRIGGER ELIMINAR_REGISTRO
AFTER DELETE ON ARTISTAS
REFERENCING OLD ROW AS Antigua
FOR EACH ROW
BEGIN ATOMIC
    INSERT INTO REGISTRO_ARTISTA ( TIPO_ACCION, ID_ARTISTA )
    VALUES ( 'DELETE', Antigua.ID_ARTISTA );
END;

INSERT INTO ARTISTAS ( ID_ARTISTA, NOMBRE_ARTISTA )
VALUES ( 2019, 'John Lee Hooker' );

UPDATE ARTISTAS
SET LUGAR_DE_NACIMIENTO = 'Clarksdale, Mississippi, Estados Unidos'
WHERE ID_ARTISTA = 2019;

DELETE ARTISTAS
WHERE ID_ARTISTA = 2019;

SELECT * FROM REIGSTRO_ARTISTA;

DROP TRIGGER INSTERTAR_REGISTRO;

DROP TRIGGER ACTUALIZAR_REGISTRO;

DROP TRIGGER ELIMINAR_REGISTRO;

DROP TABLE REGISTRO_ARTISTA;

```


Prueba esto 15-1 Trabajar con cursores SQL

```
DECLARE v_NOMBRE_CD VARCHAR (60);

DECLARE CD_cursor_1 CURSOR
FOR
    SELECT TITULO_CD
    FROM DISCOS_COMPACTOS
    ORDER BY TITULO_CD ASC;

OPEN CD_cursor_1;

FETCH CD_cursor_1 INTO v_NOMBRE_CD;

CLOSE CD_cursor_1;

DECLARE v_NOMBRE_CD VARCHAR(60);

DECLARE CD_cursor_2 SCROLL INSENSITIVE CURSOR
FOR
    SELECT TITULO_CD
    FROM DISCOS_COMPACTOS
    ORDER BY TITULO_CD ASC
    FOR READ ONLY;

OPEN CD_cursor_2;

FETCH LAST FROM CD_cursor_2 INTO v_NOMBRE_CD;

CLOSE CD_cursor_2;

DECLARE v_NOMBRE_CD VARCHAR(60);

DECLARE CD_cursor_3 CURSOR
FOR
    SELECT TITULO_CD
    FROM DISCOS_COMPACTOS
    FOR UPDATE;

OPEN CD_cursor_3;

FETCH CD_cursor_3 INTO v_NOMBRE_CD;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = EN_EXISTENCIA * 2
    WHERE CURRENT OF CD_cursor_3;
```

```
CLOSE CD_cursor_3;

SELECT * FROM DISCOS_COMPACTOS;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = 13
    WHERE ID_DISCO_COMPACTO = 101;
```

Pruebe esto 16-1 Trabajar con transacciones

```
START TRANSACTION
    ISOLATION LEVEL READ UNCOMMITTED;

SELECT *
    FROM ARTISTAS;

COMMIT;

START TRANSACTION
    ISOLATION LEVEL SERIALIZABLE;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = EN_EXISTENCIA + 2
    WHERE ID_DISQUERA = 832;

ROLLBACK;

SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
    WHERE ID_DISQUERA = 832;

START TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT TITULO_CD, EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
    WHERE ID_DISQUERA = 832;

SAVEPOINT SECCION_1;

UPDATE DISCOS_COMPACTOS
    SET EN_EXISTENCIA = EN_EXISTENCIA + 2
    WHERE ID_DISQUERA = 832;

ROLLBACK TO SAVEPOINT SECCION_1;
```



```

END DECLARE SECTION;
EXEC SQL
  WHENEVER SQLLEXCEPTION GOTO Error1;
EXEC SQL
  SELECT TITULO_CD, EN_EXISTENCIA
    INTO :v_TITULO_CD :ind_TITULO_CD, :v_EN_EXISTENCIA :ind_EN_EXISTENCIA
    FROM DISCOS_COMPACTOS
   WHERE ID_DISCO_COMPACTO = :v_CD_ID;

```

Pruebe esto 17-2 Utilizar la interfaz de nivel de llamada de SQL

```

SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
SQLAllocHandle ( SQL_HANDLE_DBC, henv, &hdbc );
SQLConnect ( hdbc, ServidorBD, SQL_NTS, AdminBD, SQL_NTS, CAdmin,
SQL_NTS );
SQLAllocHandle ( SQL_HANDLE_STMT, hdbc, &hstmt );
SQLExecDirect ( hstmt, "DELETE FROM DISCOS_COMPACTOS
  WHERE ID_DISCO_COMPACTO = 122", SQL_NTS );
SQLPrepare ( hstmt, "SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_
COMPACTOS
  WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
SQLBindParameter ( hstmt, 1, SQL_PARAMETER_MODE_IN, SQL_INT,
  SQL_INT, 3, 0, &v_CD_ID, 4, &ind_CD_ID );
SQLExecute ( hstmt );
SQLBindCol ( hstmt, 1, SQL_CHAR, &v_TITULO_CD, 60, &ind_TITULO_CD );
SQLBindCol ( hstmt, 2, SQL_INT, &v_EN_EXISTENCIA, 5, &ind_EN_EXISTENCIA );
SQLAllocHandle ( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
SQLAllocHandle ( SQL_HANDLE_DBC, henv, &hdbc );
SQLConnect ( hdbc, ServidorBD, SQL_NTS, AdminBD, SQL_NTS, CAdmin,
SQL_NTS );
SQLAllocHandle ( SQL_HANDLE_STMT, hdbc, &hstmt );
SQLExecDirect ( hstmt, "DELETE FROM DISCOS_COMPACTOS
  WHERE ID_DISCO_COMPACTO = 122", SQL_NTS );
SQLPrepare ( hstmt, "SELECT TITULO_CD, EN_EXISTENCIA FROM DISCOS_
COMPACTOS
  WHERE ID_DISCO_COMPACTO = ?", SQL_NTS );
SQLBindParameter ( hstmt, 1, SQL_PARAMETER_MODE_IN, SQL_INT,
  SQL_INT, 3, 0, &v_CD_ID, 4, &ind_CD_ID );
SQLExecute ( hstmt );
SQLBindCol ( hstmt, 1, SQL_CHAR, &v_TITULO_CD, 60, &ind_TITULO_CD );
SQLBindCol ( hstmt, 2, SQL_INT, &v_EN_EXISTENCIA, 5, &ind_EN_EXISTENCIA );

```

Pruebe esto 18-1 Utilizar funciones SQL/XML

```

SELECT XMLELEMENT(NOMBRE CD,
    XMLATTRIBUTE(ID_DISCO_COMPACTO AS ID),
    XMLFOREST(TITULO_CD as Titulo))
FROM DISCOS_COMPACTOS
ORDER BY ID_DISCO_COMPACTO;
<CD ID='101'>
  <Titulo>Famous Blue Raincoat</Titulo>
</CD>
<CD ID='102'>
  <Titulo>Blue</Titulo>
</CD>

```

La base de datos INVENTARIO

Es posible que mientras se trabaja en los proyectos de este libro se necesite volver a crear la base de datos INVENTARIO. Ésta pudiera ser el resultado de cambiar a una implementación SQL diferente, reinstalar la implementación SQL, o incluso querer empezar con datos frescos y una base de datos nueva. Las siguientes instrucciones SQL permitirán volver a crear los objetos de la base de datos INVENTARIO (tablas y vistas) en su totalidad. Una vez que se creen las tablas necesarias, se pueden utilizar instrucciones INSERT para agregarles datos. Si se desea copiar las instrucciones SQL directamente desde un archivo, puede descargar el archivo AppC.txt, que contiene las instrucciones de definición de datos y las instrucciones INSERT.

```

CREATE TABLE TIPOS_MUSICA
( ID_TIPO          INT,
  NOMBRE_TIPO VARCHAR(20) NOT NULL,
  CONSTRAINT UN_NOMBRE_TIPO UNIQUE (NOMBRE_TIPO),
  CONSTRAINT PK_TIPOS_MUSICA PRIMARY KEY (ID_TIPO) );

CREATE TABLE DISQUERAS_CD
( ID_DISQUERA      INT,
  NOMBRE_COMPañIA VARCHAR(60) DEFAULT 'Independiente' NOT NULL,
  CONSTRAINT PK_DISQUERAS_CD PRIMARY KEY (ID_DISQUERA) );

CREATE TABLE DISCOS_COMPACTOS
( ID_DISCO_COMPACTO INT,
  TITULO_CD          VARCHAR(60) NOT NULL,
  ID_DISQUERA        INT          NOT NULL,
  EN_EXISTENCIA      INT          NOT NULL,
  CONSTRAINT PK_DISCOS_COMPACTOS PRIMARY KEY (ID_DISCO_COMPACTO),
  CONSTRAINT FK_ID_DISQUERA FOREIGN KEY (ID_DISQUERA) REFERENCES
DISQUERAS_CD,
  CONSTRAINT CK_EN_EXISTENCIA CHECK ( EN_EXISTENCIA > 0 AND EN_EXISTENCIA
< 50 ) );

CREATE TABLE TIPOS_DISCO_COMPACTO
( ID_DISCO_COMPACTO INT,

```

```

ID_TIPO_MUSICA INT,
CONSTRAINT PK_TIPOS_DISCO_COMPACTO
    PRIMARY KEY ( ID_DISCO_COMPACTO, ID_TIPO_MUSICA ),
CONSTRAINT FK_ID_DISCO_COMPACTO_01
    FOREIGN KEY (ID_DISCO_COMPACTO) REFERENCES DISCOS_COMPACTOS,
CONSTRAINT FK_ID_TIPO_MUSICA
    FOREIGN KEY (ID_TIPO_MUSICA) REFERENCES TIPOS_MUSICA );

CREATE TABLE ARTISTAS
( ID_ARTISTA INT,
  NOMBRE_ARTISTA          VARCHAR(60)                NOT NULL,
  LUGAR_DE_NACIMIENTO VARCHAR(60) DEFAULT 'Desconocido' NOT NULL,
  CONSTRAINT PK_ARTISTAS PRIMARY KEY (ID_ARTISTA) );

CREATE TABLE CDS_ARTISTA
( ID_ARTISTA          INT,
  ID_DISCO_COMPACTO INT,
  CONSTRAINT PK_CDS_ARTISTA PRIMARY KEY ( ID_ARTISTA, ID_DISCO_COMPACTO ),
  CONSTRAINT FK_ID_ARTISTA FOREIGN KEY (ID_ARTISTA) REFERENCES ARTISTAS,
  CONSTRAINT FK_ID_DISCO_COMPACTO_02 FOREIGN KEY (ID_DISCO_COMPACTO)
    REFERENCES DISCOS_COMPACTOS );

CREATE VIEW CDS_EN_EXISTENCIA AS
SELECT TITULO_CD, EN_EXISTENCIA
FROM DISCOS_COMPACTOS
WHERE EN_EXISTENCIA > 10 WITH CHECK OPTION;

CREATE VIEW EDITORES_CD ( TITULO_CD, EDITOR ) AS
SELECT DISCOS_COMPACTOS.TITULO_CD, DISQUERAS_CD.NOMBRE_COMPAÑIA
FROM DISCOS_COMPACTOS, DISQUERAS_CD
WHERE DISCOS_COMPACTOS.ID_DISQUERA = DISQUERAS_CD.ID_DISQUERA;

--Inserta datos en la tabla DISQUERAS_CD
INSERT INTO DISQUERAS_CD VALUES ( 827, 'Private Music' );
INSERT INTO DISQUERAS_CD VALUES ( 828, 'Reprise Records' );
INSERT INTO DISQUERAS_CD VALUES ( 829, 'Asylum Records' );
INSERT INTO DISQUERAS_CD VALUES ( 830, 'Windham Hill Records' );
INSERT INTO DISQUERAS_CD VALUES ( 831, 'Geffen' );
INSERT INTO DISQUERAS_CD VALUES ( 832, 'MCA Records' );
INSERT INTO DISQUERAS_CD VALUES ( 833, 'Decca Record Company' );
INSERT INTO DISQUERAS_CD VALUES ( 834, 'CBS Records' );
INSERT INTO DISQUERAS_CD VALUES ( 835, 'Capitol Records' );
INSERT INTO DISQUERAS_CD VALUES ( 836, 'Sarabande Records' );
--Fin de inserción de datos para la tabla DISQUERAS_CD

--Inserta datos en la tabla DISCOS_COMPACTOS
INSERT INTO DISCOS_COMPACTOS VALUES
    ( 101, 'Famous Blue Raincoat', 827, 13 );
INSERT INTO DISCOS_COMPACTOS VALUES
    ( 102, 'Blue', 828, 42 );
INSERT INTO DISCOS_COMPACTOS VALUES
    ( 103, 'Court and Spark', 829, 22 );
INSERT INTO DISCOS_COMPACTOS VALUES

```

```
( 104, 'Past Light', 830, 17 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 105, 'Kojiki', 831, 6 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 106, 'That Christmas Feeling', 832, 8 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 107, 'Patsy Cline: 12 Greatest Hits', 832, 32 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 108, 'Carreras Domingo Pavarotti in Concert', 833, 27 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 109, 'After the Rain: The Soft Sounds of Erik Satie', 833, 21 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 110, 'Out of Africa', 832, 29 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 111, 'Leonard Cohen The Best Of', 834, 12 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 112, 'Fundamental', 835, 34 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 113, 'Bob Seger and the Silver Bullet Band Greatest Hits',
835, 16 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 114, 'Blues on the Bayou', 832, 27 );
INSERT INTO DISCOS_COMPACTOS VALUES
( 115, 'Orlando', 836, 5 );
--Fin de inserción de datos para la tabla DISCOS_COMPACTOS

--Inserta datos en la tabla TIPOS_MUSICA
INSERT INTO TIPOS_MUSICA VALUES ( 11, 'Blues' );
INSERT INTO TIPOS_MUSICA VALUES ( 12, 'Jazz' );
INSERT INTO TIPOS_MUSICA VALUES ( 13, 'Pop' );
INSERT INTO TIPOS_MUSICA VALUES ( 14, 'Rock' );
INSERT INTO TIPOS_MUSICA VALUES ( 15, 'Classical' );
INSERT INTO TIPOS_MUSICA VALUES ( 16, 'New Age' );
INSERT INTO TIPOS_MUSICA VALUES ( 17, 'Country' );
INSERT INTO TIPOS_MUSICA VALUES ( 18, 'Folk' );
INSERT INTO TIPOS_MUSICA VALUES ( 19, 'International' );
INSERT INTO TIPOS_MUSICA VALUES ( 20, 'Soundtracks' );
INSERT INTO TIPOS_MUSICA VALUES ( 21, 'Christmas' );
--Fin de inserción de datos para la tabla TIPOS_MUSICA

--Inserta datos en la tabla TIPOS_DISCO_COMPACTO
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 101, 18 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 101, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 102, 11 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 102, 18 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 102, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 103, 18 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 103, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 104, 16 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 105, 16 );
```

```

INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 106, 21 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 107, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 107, 17 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 108, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 108, 15 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 109, 15 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 110, 20 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 111, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 111, 18 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 112, 11 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 112, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 113, 13 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 113, 14 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 114, 11 );
INSERT INTO TIPOS_DISCO_COMPACTO VALUES ( 115, 20 );
--Fin de inserción de datos para la tabla TIPOS_DISCO_COMPACTO

--Inserta datos en la tabla ARTISTAS
INSERT INTO ARTISTAS VALUES
    ( 2001, 'Jennifer Warnes', 'Seattle, Washington, Estados Unidos' );
INSERT INTO ARTISTAS VALUES
    ( 2002, 'Joni Mitchell', 'Fort MacLeod, Alberta, Canadá' );
INSERT INTO ARTISTAS VALUES
    ( 2003, 'William Ackerman', 'Alemania' );
INSERT INTO ARTISTAS VALUES
    ( 2004, 'Kitaro', 'Toyohashi, Japón' );
INSERT INTO ARTISTAS VALUES
    ( 2005, 'Bing Crosby', 'Tacoma, Washington, Estados Unidos' );
INSERT INTO ARTISTAS VALUES
    ( 2006, 'Patsy Cline', 'Winchester, Virginia, Estados Unidos' );
INSERT INTO ARTISTAS VALUES
    ( 2007, 'Jose Carreras', 'Barcelona, España' );
INSERT INTO ARTISTAS VALUES
    ( 2008, 'Luciano Pavarotti', 'Modena, Italia' );
INSERT INTO ARTISTAS VALUES
    ( 2009, 'Placido Domingo', 'Madrid, España' );
INSERT INTO ARTISTAS VALUES
    ( 2010, 'Pascal Roge', 'Desconocido' );
INSERT INTO ARTISTAS VALUES
    ( 2011, 'John Barry', 'Desconocido' );
INSERT INTO ARTISTAS VALUES
    ( 2012, 'Leonard Cohen', 'Montreal, Quebec, Canadá' );
INSERT INTO ARTISTAS VALUES
    ( 2013, 'Bonnie Raitt', 'Burbank, California, Estados Unidos' );
INSERT INTO ARTISTAS VALUES
    ( 2014, 'Bob Seger', 'Dearborn, Michigan, Estados Unidos' );
INSERT INTO ARTISTAS VALUES
    ( 2015, 'Silver Bullet Band', 'No aplica' );
INSERT INTO ARTISTAS VALUES
    ( 2016, 'B.B. King', 'Indianola, Mississippi, Estados Unidos' );

```



```
INSERT INTO ARTISTAS VALUES
    ( 2017, 'David Motion', 'Desconocido' );
INSERT INTO ARTISTAS VALUES
    ( 2018, 'Sally Potter', 'Desconocido' );
--Fin de inserción de datos para la tabla ARTISTAS

--Inserta datos en la tabla CDS_ARTISTA
INSERT INTO CDS_ARTISTA VALUES ( 2001, 101 );
INSERT INTO CDS_ARTISTA VALUES ( 2002, 102 );
INSERT INTO CDS_ARTISTA VALUES ( 2002, 103 );
INSERT INTO CDS_ARTISTA VALUES ( 2003, 104 );
INSERT INTO CDS_ARTISTA VALUES ( 2004, 105 );
INSERT INTO CDS_ARTISTA VALUES ( 2005, 106 );
INSERT INTO CDS_ARTISTA VALUES ( 2006, 107 );
INSERT INTO CDS_ARTISTA VALUES ( 2007, 108 );
INSERT INTO CDS_ARTISTA VALUES ( 2008, 108 );
INSERT INTO CDS_ARTISTA VALUES ( 2009, 108 );
INSERT INTO CDS_ARTISTA VALUES ( 2010, 109 );
INSERT INTO CDS_ARTISTA VALUES ( 2011, 110 );
INSERT INTO CDS_ARTISTA VALUES ( 2012, 111 );
INSERT INTO CDS_ARTISTA VALUES ( 2013, 112 );
INSERT INTO CDS_ARTISTA VALUES ( 2014, 113 );
INSERT INTO CDS_ARTISTA VALUES ( 2015, 113 );
INSERT INTO CDS_ARTISTA VALUES ( 2016, 114 );
INSERT INTO CDS_ARTISTA VALUES ( 2017, 115 );
INSERT INTO CDS_ARTISTA VALUES ( 2018, 115 );
--Fin de inserción de datos para la tabla CDS_ARTISTA
```

Índice

Símbolos

-- (guiones) con instrucciones INSERT, 170
- (sustracción) operador de, 239
% (signo de porcentaje) con LIKE, 203-205
& (símbolo de conjunción) con indicadores de asignación, 422
() (paréntesis)
 con la función COUNT, 227
 con la instrucción CALL, 308
 con las palabras clave AND y OR, 155
 con predicados, 158-159
 organización en código, 96-97
* (asterisco)
 con cursores, 361
 con resultados de la consulta, 148
 en la función COUNT, 227-228
* (multiplicación) operador de, 239
/ (división) operador de, 239
/ (diagonal) en XML, 436
: (dos puntos)
 con etiquetas de instrucción, 317
 con variables host, 408-409
::= (símbolo), 90
; (punto y coma) con instrucciones compuestas, 313
? (signo de interrogación) marcador de posición, 425
[] (corchetes) con la cláusula MATCH, 88
_ (guiones bajos)
 con el predicado LIKE, 203-204
 utilización de, 41
{ } (llaves), 51
+ (adición) operador de, 239
< (menor que) operador de, 195, 197
<!-- y --> en XML, 436
<= (menor que o igual a) operador de, 195, 198-199
<> (corchetes angulares)
 en XML, 436
 utilización de, 42
<> (desigual a) operador de, 195-196
<::> separador con la palabra clave class, 133

= (igual a) operador
 con equi-joins, 256
 con la cláusula WHERE, 196
 ejemplo de, 195
> (predicados de comparación)
 combinación, 197, 207
 con restricciones, 95
 definir subconsultas con, 292
 descripción de, 194
> (mayor que) operador, 195, 197
>= (mayor que o igual a) operador, 195, 198-199
' (comillas sencillas) con INSERT INTO, 177-178
, (coma)
 con columnas, 53
 con identificadores de autorización, 137-138
 con la cláusula INSERT INTO, 177
 con la instrucción CALL, 308
 con referencias de tabla, 149

A

acción CONTINUE con WHENEVER, 413
acción referencial CASCADE, 90
acción referencial NO ACTION, 90
acción referencial RESTRICT, 90
acción referencial SET DEFAULT, 90
acción referencial SET NULL, 90
activadores. *Véase también* activadores de eliminación; activadores de inserción; activadores de actualización
 como objetos de esquema, 330-331
 contexto de ejecución para, 331-332
 creación de, 345-348, 508-509
 definidos, 17
 descripción de, 36
 disparar al mismo tiempo, 342
 invocar, 342
 nombres de correlación para, 334-335

probar, 346
quitar, 335, 341, 347
sintaxis, 333
tipos de, 335
vistazo general de, 330-331
activadores de actualización. *Véase también* activadores
 creación de, 338-341, 345-346
 quitar, 347
activadores de eliminación. *Véase también* activadores
 creación de, 343-344, 346
 quitar, 347
activadores de inserción, creación de, 336-338, 345. *Véase también* activadores
actualizar datos en columnas referenciadas, 89-91
Ada, instrucciones SQL en, 408
adición (+)
 operador de, 239
afirmaciones
 definición, 97-98
 definidas, 74
 limitación de, 75
agente SQL, 31
almacenaje en tuplas, definición de, 6
asignación de nombres, definida, 33
asterisco (*)
 con cursores, 361
 con resultados de la consulta, 148
 en la función COUNT, 227-228
atributos
 definidos, 6
 dependencia funcional de, 9
 en XML, 436
 identificadores únicos, 8-9
 terminología, 7
atributos XML, 436

B

bases de datos
 agregar vistas a, 119-120
 concepto de, 37-39
 conectar a, 25-26
 creación de, 44-45
 definidas, 4
 quitar vistas de, 117

bloque BEGIN
 con instrucciones compuestas,
 313-314
 con repeticiones, 318
 bosques en XML, 437
 bytes
 almacenaje, 57
 tamaño de, 55

C

cadenas binarias, 55, 57
 cadenas de bits, 55-56
 cadenas de caracteres. *Véase también*
 funciones de valor de cadena
 comparación, 230
 tipos de, 55
 cadenas de caracteres nacionales, 55-56
 calidad aislada, aplicar a transacciones,
 379
 calidad consistente,
 aplicar a transacciones, 379
 capacidad de actualización del cursor
 definición de, 362, 373
 vistazo general de, 359-360
 capacidad de desplazamiento del cursor
 definición de, 361
 vistazo general de, 357
 capacidad del cursor para arrojar
 resultados, vistazo general de,
 358-359
 capacidad para aceptar valores nulos por
 defecto, anular, 76-77
 capacidad para mantener abierto el
 cursor, vistazo general de, 358
 característica durable, aplicar a
 transacciones, 379
 cardinalidad mínima, definida, 13. *Véase*
 también tuplas
 catálogos. *Véase también* esquemas
 componentes de, 33
 descripción de, 32
 esquemas en, 34-35
 estructura jerárquica de, 33-34
 frente a esquemas, 34
 relaciones de objeto en, 34
 catálogos SQL. *Véase* catálogos
 CD en existencias,
 determinar, 247-248
 cláusulas
 procesar en orden, 165, 228
 uso opcional de, 88
 clave candidato
 con la restricción PRIMARY
 KEY, 80
 definición, 7
 clave primaria. *Véase* identificador único

clave sustituto, definida, 7
 claves en normalización, 7
 claves foráneas
 columnas múltiples en, 86-87
 crear cómo restricciones de tabla,
 86
 en la tabla DISCOS_
 COMPACTOS, 93
 en la tabla TIPOS_DISCO_
 COMPACTO, 94
 para unir tablas, 108
 representar como [FK], 91
 claves primarias
 para la tabla DISQUERAS_CD,
 93
 para la tabla TIPOS_MUSICA, 93
 versus restricciones FOREIGN
 KEY, 87-88
 CLI (interfaz de nivel de llamada)
 ejecutar instrucciones SQL en,
 423-424
 indicadores de asignación en, 421-
 423
 la función ExecDirect() en, 423-
 424, 428
 la función Execute() en, 424-425
 la función Prepare() en, 424
 recuperar datos en, 426
 utilización de, 427-429, 513
 variables host en, 424-425
 vistazo general de, 419-420
 COBOL, instrucciones SQL en, 408
 Codd, E.F. (Edgar Frank Codd), 5, 7, 15
 código, separar en líneas, 96-97
 columnas. *Véase también* marcador de
 posición <columna derivada>;
 columnas de vista
 agregar, 239
 arrojar desde tablas, 361
 calcular, 240
 combinar en restricciones, 79
 consultar, 149-151, 153
 crear y eliminar, 70-71
 definición de, 52-53
 en claves foráneas, 86-87
 en conjuntos de resultados, 411
 en el predicado NULL, 202
 en INSERT INTO, 180
 en la cláusula HAVING, 165
 en los resultados de la consulta,
 148
 en vistas, 116
 enlistar valores para, 96
 especificar con GROUP BY, 162
 especificar para columnas, 95-97
 nombrar para vistas, 110-111
 para INSERT INTO, 178-179

promediar valores en, 148-149
 utilizar coma (,) con, 53
 utilizar la opción RESTRICT con,
 68
 valores para, 95-97
 valores únicos en, 77-79
 visualización, 105
 columnas de vista, asignar nombres para,
 105. *Véase también* columnas
 coma (,)
 con columnas, 53
 con identificadores de
 autorización, 137-138
 con la cláusula INSERT INTO,
 177
 con la instrucción CALL, 308
 con referencias de tabla, 149
 comentarios,
 agregar en XML, 436
 comillas simples (') con INSERT INTO,
 177-178
 complejidad de la consulta, ocultar con
 vistas, 105-107
 componentes del modelo de seguridad
 identificadores de autorización,
 124
 identificadores de usuario, 124
 identificadores PUBLIC, 125-
 126
 nombres de rol, 125
 sesiones, 126-128
 condiciones
 inverso de, 200, 212, 214
 y tamaño de diagnóstico, 387
 condiciones de búsqueda. *Véase también*
 predicados
 combinar, 159
 definir con la cláusula WHERE,
 152-159
 definir para el predicado LIKE,
 203-204
 formatear desde predicados,
 198
 operadores para, 156
 condiciones de búsqueda de grupo,
 especificar, 164-166
 conectividad,
 verificar, 26
 conjuntos de caracteres, 36
 conjuntos de caracteres de SQL, trazado
 para, 443
 conjuntos de resultados. *Véase también*
 resultados de la consulta
 arrojar con cursores, 353
 definidos, 352
 filas y columnas en, 411
 consultar tablas múltiples, 505

consultas. *Véase también* subconsultas correlacionadas
 almacenar en definiciones de vista, 104-105
 para columnas en resultados de la consulta, 170-171
 contexto de ejecución, aplicar a activadores, 331-332
 corchetes ([])
 con cláusula MATCH, 88
 corchetes angulares (<>)
 en XML, 436
 utilizar, 42
 cotejo por defecto, 36
 cotejos, 36
 cross join,
 creación de, 259, 273
 cualidad atómica,
 aplicación a transacciones, 379
 cuerpo de la rutina, variables en, 311-312
 cursor de sólo lectura, creación del, 372-373
 cursor desplazable, definición de, 361
 cursores
 abrir y cerrar, 354, 363, 369
 acceder tablas con, 353-354
 características de, 364
 como señaladores, 353, 411
 con DELETE posicionada, 370-371
 con instrucciones SELECT, 368
 con UPDATE posicionada, 368-370
 de sólo lectura, 372-373
 declarar, 354-356, 371-372
 declarar en módulos, 417
 declarar y acceder, 373
 definición de, 369
 instrucciones utilizadas con, 354
 recuperar datos desde, 363-367
 utilización de, 510-511
 vistazo general de, 353-355
 cursores de la instrucción OPEN, 354, 364

D

datos
 actualizar, 182-186, 290
 actualizar desde columnas referenciadas, 89-91
 almacenaje en tuplas, 6
 comparar con predicados, 194-198
 eliminar con subconsultas, 291
 eliminar de columnas referenciado, 89-91
 en nodos, 5-6

en tipos de registro, 6
 insertar, 288-289
 modificar, 188-190, 502
 modificar con procedimientos, 309-310
 modificar con vistas, 108
 proteger entre tablas, 83
 recuperar, 411-412
 recuperar en CLI, 426
 resumir, 164
 unir con vistas, 108
 utilizar la instrucción DELETE con, 69
 datos SQL. *Véase* datos
 DBMS de Oracle, arquitectura de, 39
 DCL (Lenguaje de Control de Datos), 19
 DDL (Lenguaje de Definición de Datos), 18
 definición de, 358
 definiciones de vista
 almacenar instrucciones SELECT en, 304
 la instrucción SELECT en, 153
 dependencia funcional, definida, 9
 desnormalizar datos, 11
 diagonal (/) en XML, 436
 diseño relacional, 11
 DML (Lenguaje de Manipulación de Datos), 19
 documentos XML
 ejemplo de, 434
 trazado para, 441, 443
 dominios
 creación de, 98
 definidos, 6
 descripción de, 35, 37
 dos puntos (:)
 con la etiqueta de instrucción, 317
 con variables host, 408-409

E

el activador INSERT, definición de, 336
 el activador INSERT_LOG, creación de, 336-337
 el cursor WITH HOLD, definición de, 358
 el esquema INVENTARIO_CD, creación de, 45
 el esquema INVENTARIO, creación de, 43
 el estándar ANSI para SQL-86, 16
 el estándar SQL, partes de, 16
 el estándar SQL-86, liberación de, 16
 el estándar SQL:2006
 conformar con, 20
 conformidad para, 17-18

terminología, 4
 ver información acerca de, 18
 el identificador de autorización App_User, 127-128
 el lenguaje host, SQL incrustado en, 407-408
 el marcador de posición <acción referencial desencadenada> con FOREIGN KEY, 89-91
 el marcador de posición <condición de búsqueda>
 con la restricción CHECK, 95-96
 utilización de, 153
 el marcador de posición <cuerpo de la rutina>, 303
 el marcador de posición <definición de la columna>, 68
 el marcador de posición <determinar expresión de la cláusula>, 183
 el marcador de posición <especificación de agrupación>, 159-160
 el marcador de posición <expresión de la consulta> con vistas, 109-110
 el marcador de posición <lista de privilegios> con la cláusula GRANT, 132
 el marcador de posición <lista selecta> con resultados de la consulta, 148
 el marcador de posición <modo> con SET TRANSACTION, 382
 el marcador de posición <nombre de objeto> con la cláusula ON, 132-133
 el marcador de posición <orientación para búsqueda>, 364
 el marcador de posición <tipo de objeto> con la cláusula ON, 132-133
 el marcador de posición signo de interrogación (?), 425
 el método CLI, 20
 el modelo CLI,
 implementaciones de, 419
 el operador BETWEEN con restricciones, 97
 el operador CUBE con GROUP BY, 163-164
 el operador EXCEPT, efecto de, 272
 el operador igual a (=)
 con equi-joins, 256
 con la cláusula WHERE, 196
 ejemplo de, 195
 el operador INTERSECT *versus* UNION, 272
 el operador IS FALSE, 156
 el operador IS TRUE, 156
 el operador mayor que (>), 195, 197

- el operador mayor que o igual a (\geq), 195, 198-199
 - el operador menor que ($<$), 195, 197
 - el operador menor que o igual a (\leq), 195, 198-199
 - el operador UNION
 - utilización de, 269-271
 - versus* full outer join, 271
 - el paquete PKG008, activadores en, 331
 - el parámetro FOR con SUBSTRING, 234-235
 - el parámetro FROM con SUBSTRING, 234-235
 - el parámetro SQLSTATE con módulos, 417-418
 - el predicado ALL, 218-219, 221
 - el predicado ANY, 216-218, 221, 282
 - el predicado BETWEEN, 199-200, 207
 - el predicado EXISTS
 - con subconsultas, 215, 220, 281
 - descripción de, 213
 - utilización de, 284-285, 292
 - el predicado IN
 - actualizar datos con, 290
 - con subconsultas, 284
 - definir listas con, 210-211
 - descripción de, 209
 - subconsulta en, 211-212
 - utilización de, 220, 279-280
 - el predicado NULL
 - agregar la palabra clave NOT a, 208
 - utilización de, 200-203
 - el predicado SOME, 216-218, 282
 - el prefijo sqlxml con el tipo XMLSCHEMA, 438
 - el prefijo Xs con el tipo XMLSCHEMA, 438
 - el privilegio EXECUTE con objetos de esquema, 130
 - el privilegio TRIGGER con objetos de esquema, 130
 - el privilegio UNDER con objetos de esquema, 130
 - el privilegio USAGE con objetos de esquema, 130
 - el procedimiento CDS_NEW_AGE
 - creación de, 305
 - descripción de, 306
 - el procedimiento OBTENER_ARTISTAS_CD
 - convocar, 320
 - quitar, 319
 - el programa C
 - declarar variables host en, 409-410
 - incrustar instrucciones en, 409
 - instrucciones SQL en, 408
 - la función AllocHandle() en, 422
 - la función BindParameter() en, 425
 - módulos de llamada en, 418
 - el rol ADMINISTRADORES, creación de, 138
 - el rol CONTABILIDAD, otorgar privilegios a, 134
 - el rol MRKT
 - creación de, 139
 - otorgar el rol PERSONAL_VENTAS hacia, 140
 - quitar, 140
 - revocar el rol PERSONAL_VENTAS desde, 140
 - el rol PERSONAL_VENTAS
 - creación de, 139
 - otorgar para el rol MRKT, 140
 - otorgar privilegios para, 130-140
 - quitar, 140
 - revocar desde el rol MRKT, 140
 - el rol VENTAS, otorgar privilegios a, 134
 - el tipo SALARIO, creación de, 64
 - el valor SESSION_USER, 246
 - el valor USER, 246
 - elementos de agrupación, 51
 - elementos XML, 436
 - eliminar
 - columnas, 71
 - datos desde columnas referenciadas, 89-91
 - datos desde tablas, 69
 - esquemas, 34, 43-44
 - objetos de esquema, 43
 - roles, 130-131, 140
 - tablas, 69-71, 499
 - entidad, definida, 6
 - entorno de conexión, entorno SQL
 - acceder al, 25-26
 - componentes de, 30-31
 - objetos de nombre en, 40-41
 - equi-join
 - creación de, 256
 - definir en join de condición, 263, 272
 - escala, aplicar a números, 57
 - especificación de la consulta, definida, 114. *Véase también* instrucciones SELECT
 - esquema
 - acceder a, 128-130
 - organización de, 44
 - otorgar privilegios en, 130
 - esquemas. *Véase también* catálogos características de, 34-35
 - creación de, 34, 42-45
 - eliminar, 34, 43-44
 - INFORMATION_SCHEMA, 34
 - instrucciones SELECT en, 304-305
 - INVENTARIO, 43
 - nombrado, 42
 - tratamiento de, 34
 - versus* catálogos, 34
 - establecer en CLI, 427
 - estilos de enlace. *Véase* métodos de ejecución
 - etiqueta de instrucción con repeticiones, 317
 - etiquetas, valores nulos como, 76
 - EXEC SQL
 - ejemplo de, 409-410
 - significado de, 355, 408
 - expresión de la tabla con la instrucción SELECT, 147
 - expresiones de consulta
 - componentes de, 114
 - con vistas, 110-111
 - instrucciones SELECT como, 146
 - expresiones de valor
 - CASE, 241-243, 249
 - CAST, 244-245, 249
 - numéricas, 238-241, 248
 - utilización de, 247-249, 504-505
 - expresiones de valor numéricas
 - combinar, 240
 - utilización de, 238-241, 248-249
 - extensiones del fabricante, definidas, 17
-
- ## F
-
- factor determinante, definido, 9. *Véase también* identificador único
 - falsas, evaluar filas como, 157
 - fecha, recuperar, 237
 - filas
 - actualizar, 293-294
 - actualizar para activadores, 346
 - agrupar, 160
 - arrojado limitado de, 256-257
 - arrojar con subconsultas, 213, 278-283
 - consultar, 149-150
 - contar, 227-228
 - eliminar, 187, 190
 - eliminar para activadores, 346
 - en conjuntos de resultados, 411
 - en resultados de la consulta, 153
 - evaluación de, 213-214
 - evaluar como verdaderas, 155
 - hacer coincidir con la cláusula WHERE, 119

insertar en la tabla DISCOS_ COMPACTOS, 188
 insertar en tablas, 178
 tuplas como, 6
 [FK], significado de, 91
 Fortran, instrucciones SQL en, 408
 frente al operador UNION, 271
 full outer join
 definición de, 268-269
 descripción de, 266
 función de valor de cadena LOWER, 235-236
 funciones. *Véase también* funciones
 set; funciones definidas por el usuario; funciones de valor
 capacidades de las, 36
 con rutinas CLI, 419
 concepto, 301
 creación de, 303, 322
 definidas, 63, 226
 frente a procedimientos, 301-302, 321
 invocar, 322-323
 llamadas en CLI (interfaz de nivel de llamada), 419
 quitar, 324
 sintaxis, 301-303
 utilización de, 247-249, 504-505
 utilizar en SQL/XML, 439-440
 y rutinas invocadas por SQL, 300
 funciones de agregado. *Véase* alias de las funciones set
 con activadores, 334-335
 utilizar, 257-258
 funciones de valor. *Véase también* funciones
 descripción de, 232-233
 para cadenas, 233-236
 SUBSTRING, 233-235, 248
 UPPER, 248
 UPPER y LOWER, 235-236
 utilizar en SQL/XML, 439-440
 funciones de valor CURRENT_*, 236-237
 funciones de valor de cadena. *Véase también* cadenas de caracteres
 SUBSTRING, 233-235, 248
 UPPER, 248
 UPPER y LOWER, 235-236
 funciones de valor de fecha y hora, disponibilidad de, 236
 funciones definidas por el usuario. *Véase también* funciones
 creación de, 63-64, 321-325, 508
 descripción de, 36, 55
 frente a procedimientos, 301
 soporte para, 301

tipos definidos por el usuario (UDT)
 utilización de, 55
 funciones invocadas por SQL. *Véase también* funciones
 creación de, 321-325, 508
 versus procedimientos, 301
 funciones set. *Véase también* funciones
 AVG, 232
 con la cláusula GROUP BY, 230-231
 COUNT, 227-228
 definidas, 226
 en instrucciones, 229
 MAX, 229-231
 MIN, 229-231
 SUM, 231-232
 utilización de, 247-248

G

generadores de secuencia, 36
 guiones (--) con instrucciones INSERT, 170
 guiones bajos (_) con el predicado LIKE, 203-204
 utilización de, 41

I

<identificador de autorización> valor, 43
 identificador único, elegir, 7. *Véase también* determinante
 identificadores. *Véase también* identificadores de autorización
 calificación de, 41-42
 casos de, 41
 convenciones para, 40
 identificadores de autorización. *Véase también* identificadores
 asociar privilegios con, 131-132
 con coma (,), 137-138
 descripción de, 32
 identificadores de usuario, 124
 nombres de rol, 125
 identificadores de autorización PUBLIC
 descripción de, 125-126
 especificar para otorgar roles, 138
 otorgar privilegios a, 139-140
 otorgar privilegios SELECT a, 134-135
 revocar privilegios para, 136
 identificadores de objeto de esquema, calificación de, 41-42
 identificadores de objeto, definidos, 40

identificadores de usuario
 definidos, 124
 designar, 131
 otorgar roles para, 138
 y el nombre de rol para sesión, 126-128
 identificadores delimitados, convenciones para, 40-41
 identificadores SQL, trazado para, 443
 implementaciones SQL
 descripción de, 31
 diferencias en, 404-405
 incongruencia en la impedancia
 aparición de, 364
 definida, 352-353
 eliminación de, 405
 utilizar cursores para, 411
 incrustar, 413-414
 indexar, soporte para, 70
 indicador de ambiente
 en CLI, 421-422
 establecer en CLI, 427
 indicador de conexión en CLI, 421-423
 indicador de instrucción
 crear en CLI, 427
 utilizar en CLI, 421, 423
 indicadores de asignación en CLI, 421-423, 427
 la función AllocHandle(), 422-423
 INFORMATION_SCHEMA, contenido de, 34
 inner joins
 creación de, 264-266, 273
 frente a outer join, 264
 instrucción DECLARE con variables, 311-312
 instrucción del cursor, creación de, 360-363
 instrucciones. *Véase* instrucciones SQL
 instrucciones activadas, ejecución limitada de las, 340
 instrucciones compuestas, creación de, 312-313
 instrucciones condicionales de control
 condiciones en, 316
 utilización de, 314-315
 instrucciones DDL
 palabras clave SQL para, 18
 utilizar en transacciones, 390
 instrucciones de control. *Véase también* instrucciones SQL
 compuestas, 312-313
 condicionales, 314-315
 de repetición, 316-318
 definidas, 312
 instrucciones de repetición, creación de, 316-318

instrucciones DML en transacciones, 390
instrucciones INSERT

- con activadores de eliminación, 344
- con activadores de inserción, 337-338
- con funciones de fecha y hora, 237-238
- con la instrucción SELECT, 181-182
- con procedimientos, 310
- ejecutar, 180
- la cláusula VALUES en, 177
- sintaxis, 176
- utilización de, 169-170, 178, 288-289

instrucciones SELECT. *Véase también*
la instrucción SELECT
incrustada; especificación de la consulta

- activadores en, 338
- actualizar valores desde, 185-186
- almacenar en esquemas, 304-305
- cláusulas necesarias para, 146
- combinar resultados de, 269-271
- como expresiones de consulta, 146
- con cursores, 368
- con el predicado EXISTS, 213
- con la vista DESCUENTOS_CD, 113
- de instancia única, 411
- el predicado ALL en, 218-219
- el predicado ANY en, 217
- el predicado IN en, 211-212
- insertar valores desde, 180-182
- la cláusula FROM en, 112
- la cláusula GROUP BY en, 159-164
- la cláusula HAVING en, 171-172
- la cláusula ORDER BY en, 166-169
- la cláusula WHERE en, 112-113, 152-159, 171, 182
- la expresión de tabla para, 147
- la función COUNT en, 227-228
- la función UPPER en, 235-236
- orden de las cláusulas para, 147
- orden de procesamiento de, 228, 258
- orden de procesamiento de cláusulas en, 165
- predicados LIKE en, 205
- sintaxis, 26, 146
- SUBSTRING en, 234
- unir tablas en, 259
- vistas en, 110-111

instrucciones SQL. *Véase también*

- instrucciones de control
- clasificación, 19-20
- comienzo, 42
- DCL (Lenguaje de Control de Datos), 19
- DDL (Lenguaje de Definición de Datos), 18
- decidir en alternativas para, 215
- DML (Lenguaje de Manipulación de Datos), 19
- ejecutar, 19-20, 423-424, 428
- frente a implementaciones de productos, 21-24
- incluir funciones set en, 229
- pasar datos hacia, 410
- variables host en, 408-410
- instrucciones WHENEVER
- en manejo de errores, 415
- integridad de los datos
- asegurar la, 13
- definidos, 5
- integridad referencial, definida, 83
- interfaz de nivel de llamada de SQL. *Véase* CLI (interfaz de nivel de llamada)
- intervalo año-mes, 60
- intervalo día-hora, 60
- irregularidades de los datos
- lecturas fantasma, 384-385
- lecturas no repetibles, 384
- lecturas sucias, 383-384

J

- join de columna nombrada, creación de, 263
- join de condición
- como join de entrada y de salida, 264
- descripción de, 263
- utilización de, 273
- join natural, creación de, 262-263
- join separada por comas
- frente a cross join, 259
- implementar, 254
- lineamientos para, 257
- realizar, 272
- utilización de, 258-259
- joints
- con vistas, 108
- crear con tablas múltiples, 258-259
- cross-join, 259
- equi-join, 256
- evitar filas duplicadas en, 260

- frente a subconsultas
- correlacionadas, 286
- full outer join, 266, 268-269, 273
- inner join, 264-266, 273
- join de columna nombrada, 263
- join de condición, 263-264, 273
- natural, 262-263
- outer join, 266-269
- self-join, 260-261
- separados por comas, 254, 257-259, 272
- theta-join, 257
- tipos de, 263

L

- la acción GOTO con WHENEVER, 413-414
- la base de datos INVENTARIO
- conectarse a, 66, 70
- consultar, 169-172, 501-502
- creación de, 45
- crear roles en, 139-140
- modelo de datos para, 92
- tablas en, 71
- volver a crear, 514-518
- la cláusula ADMIN OPTION FOR
- con roles, 138
- la cláusula ALTER [COLUMN], 68
- la cláusula AND CHAIN
- con COMMIT, 396
- con ROLLBACK, 397
- la cláusula AND NO CHAIN
- con COMMIT, 396
- con ROLLBACK, 397
- la cláusula AUTHORIZATION, 417
- la cláusula <conjunto o ruta de caracteres>, 43
- la cláusula DELETE FROM, 186
- la cláusula DROP [COLUMN], 68
- la cláusula <elemento de tabla>, 52
- la cláusula <elementos de esquema>, 43
- la cláusula ELSE con CASE, 242
- la cláusula ELSE IF en instrucciones condicionales, 316
- la cláusula ELSEIF en instrucciones condicionales, 316
- la cláusula FOR EACH
- con activadores, 333, 335, 340
- con activadores de eliminación, 344
- con activadores de inserción, 337
- la cláusula FOR READ ONLY con cursores, 361
- la cláusula FOR UPDATE con cursores, 362-363

- ul style="list-style-type: none;">
- la cláusula FROM
 - con la cláusula SELECT, 149, 152-153
 - con vistas, 110-111
 - en la instrucción REVOKE, 136
 - evaluación de, 147
- la cláusula GRANT OPERATION FOR, 135
- la cláusula GRANT, opciones para, 132
- la cláusula GRANT OPTION FOR
 - para revocar privilegios, 135, 137
 - para revocar roles, 138
- la cláusula GRANTED BY
 - con la instrucción REVOKE, 135
 - utilización de, 132, 138
- la cláusula GROUP BY
 - con funciones set, 230-231, 247
 - con la cláusula HAVING, 166
 - con la cláusula WHERE, 162
 - con resultados de la consulta, 159-164
 - consideraciones respecto al rendimiento, 161
- la cláusula HAVING
 - agregar a la instrucción SELECT, 171-172
 - con GROUP BY, 166
 - incluir columnas en, 165
 - subconsultas en, 279
 - utilización de, 165-166
 - versus* WHERE, 164-165
- la cláusula INSERT INTO
 - especificar columnas en, 180
 - utilización de, 176-177
- la cláusula INTO
 - con SELECT de instancia única, 411
 - con SELECT incrustada, 415-416
 - con variables de indicador, 412
- la cláusula LANGUAGE, 417
- la cláusula MATCH con restricciones FOREIGN KEY, 88-89
- la cláusula NAMES ARE, 417
- la cláusula ON
 - agregar la condición de unión después de, 265
 - con activadores, 333, 339
 - con la instrucción REVOKE, 135
 - definir la condición de unión en, 263
 - marcadores de posición para, 132-133
- la cláusula ON DELETE, 90
- la cláusula ON UPDATE, 90
- la cláusula ORDER BY
 - con cursores, 359, 361-362
 - con XML, 435
- consideraciones respecto al rendimiento, 161
- en instrucciones SELECT, 166-169
- evaluación de, 147
- utilización de, 178
- la cláusula REFERENCING
 - con activadores, 334-335, 345-346
 - con activadores de actualización, 339
 - con activadores de eliminación, 343-344
 - soporte para, 340
- la cláusula RETURNS con funciones, 303, 322
- la cláusula SCHEMA, 417
- la cláusula SELECT
 - con expresiones de valor numéricas, 239
 - con la cláusula FROM, 149, 152-153
 - con la cláusula WHERE, 153-154
 - con la palabra clave ALL, 147-148
 - con la palabra clave DISTINCT, 147-148
 - con vistas, 112
 - subconsultas en, 279, 283
- la cláusula SET
 - combinar con la cláusula WHERE, 186
 - con UPDATE, 183, 186
 - especificar expresiones en, 184
 - subconsultas en, 290, 294
- la cláusula STATIC DISPATCH con funciones, 303
- la cláusula TO
 - con la cláusula GRANT, 137
 - opciones para, 133
- la cláusula TO SAVEPOINT, 397. *Véase también* puntos de recuperación
- la cláusula USING con joins, 263
- la cláusula VALUES
 - alternativa para, 180-182
 - con INSERT, 177-178
 - especificar valores en, 180
- la cláusula WHEN
 - con activadores, 333-334, 339-341
 - con activadores de eliminación, 344
- la cláusula WHERE
 - AND y OR en, 155
 - capacidades de, 194
 - combinar la cláusula SET con, 186
 - con activadores, 339, 341
 - con equi-join, 256
 - con funciones set, 230
 - con GROUP BY, 162
 - con la cláusula FROM, 147
 - con la instrucción DELETE, 186-187
 - con la instrucción SELECT, 153-154, 156, 171, 182
 - con predicado EXISTS, 215
 - con WITH CHECK OPTION, 111-113, 117
 - definición de, 156-159
 - definir condiciones de búsqueda con, 152-159
 - el operador igual a (=) en, 196
 - el predicado LIKE en, 204
 - en la instrucción DELETE, 198
 - en la instrucción SELECT, 113
 - en la instrucción UPDATE, 183-186, 198
 - frente a HAVING, 164-165
 - para DELETE posicionadas, 370-371
 - para hacer coincidir filas, 119
 - para UPDATE posicionadas, 368-369
 - predicados en, 157, 194
 - subconsultas en, 278
- la cláusula WHEN/THEN con CASE, 242
- la cláusula WITH ADMIN con roles, 131, 138
- la cláusula WITH CHECK OPTION, 116-117
 - para promedios, 115
 - utilización de, 110-111, 113
- la cláusula WITH GRANT OPTION, 133-135
- la columna CATEGORIA
 - valores en, 160, 163
- la columna PRECIO
 - especificar en ORDER BY, 167
 - valor nulo en, 162-163
- la columna PROM_PRECIO
 - arrojar, 165
- la columna TIPO_MUSICA, valor nulo en, 178-179
- la condición NOT FOUND, definida, 413
- la condición SQLEXCEPTION, definida, 413-414
- la condición SQLWARNING, definida, 413
- la denominación SEQUEL, posesión de, 15
- la función ARTISTA_CD
 - arrojar valores desde, 323
 - definición de, 322

- la función AVG
 - en consultas, 148-149
 - en vistas, 115
 - utilización de, 232
- la función BindCol() en CLI, 426
- la función BindParameter() en CLI, 425
- la función COUNT, 227-228, 247
- la función de valor de cadena UPPER, 235-236, 248
- la función de valor LOCALTIME, 236
- la función de valor
 - LOCALTIMESTAMP, 236
- la función de valor SUBSTRING, 233-235, 248
- la función ExecDirect(), 423-424
- la función Execute() en CLI, 424-425, 428
- la función getdate, 237
- la función MAX, 229-231
- la función MIN, 229-231, 247
- la función Prepare() en CLI, 424
- la función SQLConnect(), 423
- la función SUBSTR, 234
- la función SUM
 - con GROUP BY, 160
 - utilización de, 231-232
- la función XMLELEMENT, 439
- la función XMLFOREST, 439
- la instrucción ALTER FUNCTION, 311
- la instrucción ALTER PROCEDURE, 310
- la instrucción ALTER TABLE
 - con restricciones CHECK, 99
 - utilizar, 50, 67-68
- la instrucción ALTER VIEW, 118
- la instrucción BEGIN...END
 - con activadores, 334, 340
 - con activadores de eliminación, 344
 - con activadores de inserción, 337
 - con instrucciones condicionales, 314
 - utilización de, 313-314
- la instrucción CALL
 - con instrucciones condicionales, 315
 - con procedimientos, 305-306, 308-310, 312, 320
 - con repeticiones, 317
- la instrucción CLOSE con cursores, 354, 363
- la instrucción COMMIT
 - con transacciones, 395-396
 - definida, 380
 - finalizar transacciones con, 395
 - utilización de, 52, 381
- la instrucción CREATE DATABASE, 44
- la instrucción CREATE FUNCTION, 303, 322
- la instrucción CREATE INDEX, variaciones de, 70
- la instrucción CREATE PROCEDURE
 - con parámetros de salida, 321
 - definir parámetros en, 308
 - utilización de, 302-303, 305
- la instrucción CREATE ROLE
 - sintaxis, 130
 - utilización de, 131
- la instrucción CREATE SCHEMA
 - sintaxis, 42-43
 - utilización de, 34
- la instrucción CREATE TABLE
 - ejecutar, 53
 - para tablas base, 51
 - sintaxis, 51-52
 - utilización de, 43, 50, 65
- la instrucción CREATE TRIGGER
 - la semántica cambiante de, 337
 - utilización de, 334, 336, 338-339
- la instrucción CREATE TYPE, 64
- la instrucción CREATE VIEW
- la instrucción DECLARE CURSOR
 - capacidad de actualización del cursor parte de, 359-360
 - capacidad de desplazamiento del cursor parte de, 357
 - capacidad de ordenamiento del cursor parte de, 359
 - capacidad del cursor para arrojar resultados parte de, 358-359
 - capacidad para mantener abierto el cursor parte de, 358
 - sensibilidad del cursor parte de, 357
 - utilización de, 354-356, 361
- la instrucción DELETE
 - agregar la cláusula WHERE a, 198
 - frente a DROP TABLE, 187
 - para activadores, 343-344
 - para cursores, 359, 370-371
 - para datos, 69, 291
 - para filas, 190
 - para procedimientos, 310
 - utilización de, 186-187
- la instrucción DELETE posicionada, 370-371
- la instrucción DROP FUNCTION, 311, 324
- la instrucción DROP PROCEDURE, 311, 319
- la instrucción DROP ROLE, 131, 140
- la instrucción DROP SCHEMA, 34, 43-44
- la instrucción DROP TABLE
 - para activadores, 347
 - utilización de, 50, 119
 - versus* DELETE, 69, 187
- la instrucción DROP TRIGGER, 335, 341, 347
- la instrucción DROP VIEW, 117
- la instrucción FETCH
 - con cursores, 354-355, 361, 364-367, 373
 - y la capacidad de desplazamiento del cursor, 357
- la instrucción GRANT
 - con objetos, 130
 - con roles, 137
 - ejemplo de, 134
 - utilización de, 132
 - versus* la cláusula FROM, 136
- la instrucción IF con repeticiones, 317
- la instrucción LOOP
 - frente a la instrucción WHILE, 317-318
 - utilización de, 316-318
- la instrucción MODULE, 418
- la instrucción RELEASE SAVEPOINT
 - definida, 380
 - utilización de, 394
- la instrucción RETURN con funciones, 303, 322
- la instrucción REVOKE
 - con objetos, 130, 137
 - con roles, 138-139
 - la cláusula FROM en, 136
 - sintaxis, 135
- la instrucción ROLLBACK
 - con transacciones, 396-397, 399-400
 - definida, 380
 - finalizar transacciones con, 395
 - utilización de, 381
- la instrucción SAVEPOINT, definida, 380
- la instrucción SELECT de instancia única, 411
- la instrucción SELECT incrustada, 415-416. *Véase también* instrucciones SELECT
- la instrucción SET
 - con parámetros de salida, 321
 - con repeticiones, 317-318
 - con variables, 311-312
- la instrucción SET CONSTRAINTS
 - con transacciones, 390-392
 - definida, 380
- la instrucción SET TRANSACTION
 - creación de, 388
 - definida, 380

- frente a START TRANSACTION, 389
- uso limitado de, 381
- utilización de, 381-382
- la instrucción START TRANSACTION
 - definida, 380
 - uso limitado de, 381
 - utilización de, 381, 389, 398-399
- la instrucción TRUNCATE, 69
- la instrucción UPDATE
 - activadores en, 338-341
 - creación de, 185
 - cursores en, 359-360, 368-370, 373-374
 - impacto sobre activadores, 332
 - instrucciones condicionales en, 315
 - la cláusula SET en, 183
 - la cláusula WHERE en, 183-186, 198
 - procedimientos en, 310
 - repeticiones en, 317-318
 - sintaxis, 182
 - subconsultas en, 290, 293
 - utilización de, 290
- la instrucción UPDATE posicionada, 368-370
- la instrucción WHILE con repeticiones, 317-318
- la interfaz de línea de comandos de MySQL
 - sensibilidad al uso de mayúsculas/minúsculas de, 41
 - utilización de, 22, 24, 26
- la interfaz de nivel de llamada (CLI)
 - ejecutar instrucciones SQL en, 423-424
 - indicadores de asignación en, 421-423
 - la función ExecDirect() en, 423-424, 428
 - la función Execute() en, 424-425
 - la función Prepare() en, 424
 - recuperar datos en, 426
 - utilización de, 427-429, 513
 - variables host en, 424-425
 - vistazo general de, 419-420
- la interfaz GUI, 22
- la opción FIRST con instrucción FETCH, 365
- la opción FULL con la cláusula MATCH, 89
- la opción INSENSITIVE con cursores, 357, 362, 373
- la opción LAST con la instrucción FETCH, 365
- la opción NEW ROW, omitir desde activadores, 335
- la opción NEW TABLE, omitir desde activadores, 335
- la opción NEXT con la instrucción FETCH, 364, 367
- la opción NO SCROLL con cursores, 357
- la opción OLD ROW
 - con activadores, 343
 - omitir desde activadores, 335
- la opción OLD TABLE con activadores, 343
- la opción PARTIAL con la cláusula MATCH, 89
- la opción PRIOR con la instrucción FETCH, 364
- la opción PUBLIC con la cláusula TO, 133
- la opción READ ONLY, 382, 386, 388-389
- la opción READ WRITE, 382, 386
- la opción RELATIVE con la instrucción FETCH, 365-367
- la opción RESTRICT
 - con columnas, 68
 - con esquemas, 43
 - con procedimientos, 311
 - con REVOKE, 136
 - con tablas, 69
- la opción ROW
 - con la cláusula FOR EACH, 333
 - para activadores, 340
- la opción SCROLL con cursores, 357, 361-362, 373
- la opción SENSITIVE con cursores, 357
- la opción SIMPLE con la cláusula MATCH, 89
- la opción STATEMENT con la cláusula FOR EACH, 333-334
- la palabra clave AFTER
 - con activadores, 333, 339
 - con activadores de inserción, 337
- la palabra clave AFTER DELETE
 - con activadores, 343
- la palabra clave ALL
 - con la cláusula SELECT, 147-148, 152
 - después del operador UNION, 271
- la palabra clave AND
 - con el predicado BETWEEN, 199
 - con joins, 257
 - con predicados, 153-155, 157, 197-198, 208, 219
- la palabra clave AS
 - con CAST, 244-245
 - con nombres de correlación, 258
 - con vistas, 109-111
- la palabra clave ASC con ORDER BY, 166
- la palabra clave ATOMIC
 - con activadores, 334
 - con activadores de inserción, 337
- la palabra clave AUTHORIZATION, 42
- la palabra clave class con separador <::>, 133
- la palabra clave CONSTRAINT, 86
- la palabra clave DEFAULT con
 - columnas, 64-65
- la palabra clave DEFERRED con
 - restricciones, 392
- la palabra clave DESC
 - con la columna A_LA_MANO, 167
 - con ORDER BY, 166
- la palabra clave DISTINCT
 - con joins, 260
 - con la cláusula SELECT, 147-148, 151-152
 - con la función COUNT, 228
 - consideraciones respecto al rendimiento, 161
- la palabra clave ELSE en instrucciones condicionales, 314-316
- la palabra clave END
 - con CASE, 242
 - con instrucciones compuestas, 313-314
- la palabra clave END IF con repeticiones, 317
- la palabra clave END LOOP, 316-317
- la palabra clave EXIT con repeticiones, 316
- la palabra clave FOR con cursores, 356
- la palabra clave IN
 - evitar el uso de, 322
 - utilización de, 212
- la palabra clave IS
 - con cursores, 356
 - con el predicado NULL, 201
- la palabra clave ISOLATION LEVEL, 388
- la palabra clave JOIN, 264
- la palabra clave LOCAL con SET TRANSACTION, 381-382
- la palabra clave NOT
 - agregar a predicados LIKE, 205
 - agregar al predicado NULL, 208
 - con el predicado BETWEEN, 200
 - con el predicado IN, 212
 - con el predicado NULL, 202
 - con predicados, 158, 195
 - incluir en predicados, 208
 - utilización de, 156
- la palabra clave NULL con INSERT, 177

- la palabra clave OPEN con cursores, 363
- la palabra clave OR con predicados, 153-155, 158, 198, 200, 208, 219
- la palabra clave OUT con parámetros, 321
- la palabra clave REFERENCES
 - con objetos de esquema, 130
 - con restricciones de columna, 85
- la palabra clave ROLLUP con GROUP BY, 160, 163-164
- la palabra clave TABLE
 - con privilegios, 137
 - vistas para, 133
- la palabra clave THEN con instrucciones condicionales, 314-315
- la palabra clave VALUE con restricciones de dominio, 98
- la palabra LEAVE con repeticiones, 316
- la pila de autorización
 - creación de, 127
- la prueba ACID,
 - aplicada a transacciones, 379
 - base de datos activa, definida, 331
- la publicación PSM-96, 300
- la publicación SQL/PSM, 300
- la restricción CHECK de columna
 - creación de, 95
- la subcláusula AS con la instrucción SELECT, 151
- la tabla ARTISTAS
 - alterar, 68
 - consultar, 170
 - creación, 53, 62-63, 65
 - crear un activador de inserción en, 345
 - restricciones en, 94
 - transacciones en, 398-400
- la tabla ARTISTAS_CD
 - consultar, 281
- la tabla ARTISTAS_DISCO, consultar, 286-288
- la tabla CDS_ARTISTA,
 - creación, 94
 - establecer funciones en, 226-227
- la tabla DISCOS_COMPACTOS
 - agregar CD a, 188
 - con la restricción CHECK, 99
 - con restricciones, 93
 - consultar, 171
 - creación de, 66
 - insertar filas en, 188
 - otorgar privilegios en, 130-140
- la tabla DISQUERAS_CD
 - agregar compañía a, 188
 - creación de, 67
 - utilizar restricciones con, 93
- la tabla EMPLEADO_COMISIONES
 - creación de, 114-115
 - extraer datos desde, 116
- la tabla EMPLEADOS, usar la operación self-join, 260-261
- la tabla EXISTENCIA_CD
 - consultar, 279-280
 - crear activador de eliminación en, 343
- la tabla EXISTENCIA_DISCO_COMPACTO
 - con la cláusula HAVING, 165
 - utilización de, 160-161
- la tabla FECHAS_VENTAS, funciones de valor en, 233
- la tabla INTERPRETES
 - acceder con cursores, 353-354
 - columnas en, 150
- la tabla INVENTARIO, columnas en, 156-157
- la tabla INVENTARIO_CD
 - consultar, 181-182
 - declarar cursores en, 360-363
 - insertar valores en, 177-182
 - otorgar acceso a, 134
- la tabla INVENTARIO_DISCO,
 - consultar, 286-288
- la tabla PRECIOS_MENUDEO,
 - consultar, 282
- la tabla PRECIOS_VENTAS, consultar, 282
- la tabla RASTREO_CD
 - consultar, 241
 - expresiones de valor en, 238
- la tabla REGISTRO_ARTISTA con activadores, 345-348
- la tabla TIPO_INTERPRETE
 - filas en, 152
- la tabla TIPOS_DISCO, consultar, 286-288
- la tabla TIPOS_DISCO_COMPACTO
 - consultar, 294
 - creación de, 70
 - restricciones en, 93-94
- la tabla TIPOS_MUSICA
 - creación de, 67
 - utilizar restricciones con, 93
- la tabla TIPOS_TITULO, modificar, 288-289
- la tabla TITULOS_CD
 - creación de, 85
- la vista DESCUENTOS_CD
 - columnas en, 113
 - desplegar, 108-109
- la vista EDITORES_CD
 - creación de, 119
- la vista EMP_COMM
 - columnas en, 115
 - eliminar, 117
- LAMP (Linux, Apache, MySQL, PHP), 25
- las definiciones de columna
 - agregar restricciones PRIMARY KEY a, 81
 - con cadenas, 56
 - con NOT NULL, 76
 - sin valores por defecto, 69
 - tipos de datos de intervalo en, 61
- las palabras clave AND NOT, 157-158
- las palabras clave DEFAULT
 - CHARACTER SET, 43
- las restricciones NOT NULL
 - agregar, 76-77, 93, 499-500
 - con definiciones de columna, 79, 82-83
 - con UNIQUE, 79
 - efecto de, 200
 - utilización de, 412
- las variables host. *Véase también*
 - variables
 - como parámetro, 368
 - creación de, 414-415
 - declaración de, 422
 - definidas, 364
 - en CLI (interfaz de nivel de llamada), 424-425
 - en instrucciones SQL, 408-410
 - indicadores, 412
- lecturas fantasma, aparición de, 384, 386-387
- lecturas no repetibles, 384-385, 387
- lecturas sucias, aparición de, 383-384, 387
- left outer join
 - creación de, 267-268
 - descripción de, 266
- Lenguaje de Consulta Estructurado (SQL). *Véase* SQL (Lenguaje de Consulta Estructurado)
- Lenguaje de Control de Datos (DCL), 19
- Lenguaje de Definición de Datos (DDL), 18
- Lenguaje de Manipulación de Datos (DML), 19
- lenguaje de marcado, definido, 434
- Lenguaje de Marcado Extensible (XML)
 - agregar comentarios en, 436
 - vistazo general de, 434-437
- lenguaje procesal, definido, 15
- lenguaje relacional a objetos, SQL como, 17
- ligar los resultados de la consulta con, 428

líneas, separar el código en, 96-97
 Linux, Apache, MySQL, PHP (LAMP), 25
 <lista del identificador de autorización>
 opción, 133
 listas, definir con el predicado IN, 210-211
 literal, definido, 62, 64-65
 llaves ({}), 51
 los nombres de columna
 agregar después de los privilegios, 134
 compartir, 261-263
 con operadores de comparación, 195
 especificar en INSERT INTO, 177
 ordenar en la cláusula ORDER BY, 168
 proporcionar para columnas derivadas, 149
 proporcionar para vistas, 110
 los tipos de datos CHARACTER
 descripciones de, 56
 trazado para los tipos de esquema XML, 442
 los valores de columna
 agregar juntos, 231-232
 combinar, 240-241
 contar, 227-228
 eliminar, 187

M

manejo de errores, proporcionar, 413-415
 marcador de posición <columna derivada>, 148-149. *Véase también* columnas
 marcadores de posición, separar, 90
 metadatos, definidos, 4
 método de invocación directa
 con cursores, 372
 definido, 352
 soporte para, 22, 25
 utilización del, 19-20, 404
 ventajas del, 405
 método de vinculación de módulos, 20
 métodos
 con rutinas invocadas por SQL, 301
 definidos, 63
 métodos de acceso a los datos, elegir, 420
 métodos de ejecución
 CLI (Interfaz de Nivel de Llamada), 20
 invocación directa, 19
 SQL incrustado, 20
 unión del módulo, 20

modelo de datos de red, 5
 modelo de datos jerárquico, 4-5
 modelo de datos relacional, 4. *Véase también* normalizar datos
 entidades en, 6
 relaciones en, 5-6
 vistazo general de, 5
 modelos de bases de datos
 de red, 5
 jerárquicos, 4-5
 relacionales, 5
 modelos de datos
 con restricciones, 91-94
 definidos, 13
 para la base de datos INVENTARIO, 91-92
 modificador de tipo en SQL/XML, 438
 modificador del tipo CONTENT en SQL/XML, 438
 modificador del tipo DOCUMENT en SQL/XML, 438
 modificador del tipo SEQUENCE en SQL/XML, 438
 módulo de servidor de SQL, 36
 módulos. *Véase también* SQL módulos
 cliente
 declarar cursores en, 417
 declarar tablas temporales en, 417
 definidos, 37, 51, 301
 módulos cliente. *Véase* módulos cliente de SQL
 módulos cliente de SQL. *Véase también* módulos
 declarar parámetros para, 418
 definición de, 418-419
 descripción de, 32
 sintaxis, 417
 vistazo general de, 417-418
 muestras de código
 activadores, 508-509
 alterar y eliminar tablas, 499
 CLI (interfaz de nivel de llamada), 513
 consultar la base de datos INVENTARIO, 501-502
 consultar tablas múltiples, 505
 crear tablas SQL, 498
 cursores, 510-511
 expresiones de valor, 504-505
 funciones, 504-505
 funciones invocadas SQL, 508
 funciones SQL/XML, 513-514
 instrucciones SQL incrustadas, 512
 modificar datos SQL, 502
 predicados en instrucciones SQL, 503

procedimientos invocados por SQL, 507-508
 restricciones CHECK, 500
 restricciones NOT NULL, 499-500
 restricciones referenciales, 499-500
 restricciones UNIQUE, 499-500
 roles y privilegios, 501
 subconsultas, 505-507
 subconsultas en predicados, 503-504
 transacciones, 511
 vistas, 500
 MUMPS, instrucciones SQL en, 408
 MySQL Community Server, descarga, 25

N

nivel de aislamiento READ COMMITTED, 386-388
 nivel de aislamiento READ UNCOMMITTED, 385-389, 398-399
 nivel de aislamiento REPEATABLE READ, 386-387
 nivel de aislamiento SERIALIZABLE, 386-388, 399
 niveles de aislamiento
 anomalías de datos para, 387
 establecer para transacciones, 382-383
 READ COMMITTED, 386, 388
 READ UNCOMMITTED, 385-386, 388
 REPEATABLE READ, 386
 SERIALIZABLE, 386-388, 399
 y las lecturas sucias, 383-384
 nodos, datos en, 5-6
 nodos primarios frente a nodos secundarios, 5-6
 nodos secundarios frente a nodos primarios, 5-6
 nombres calificados, 41-42
 nombres completos calificados, 41-42
 <nombres de columna de vista>, 109-110
 nombres de correlación
 con activadores, 334-335
 utilización de, 257-258
 nombres de rol
 definidos, 126
 designar, 131
 determinar, 128
 revocar privilegios para, 137
 y el identificador de usuario para sesión, 126-128
 nombres XML, trazado para, 441

normalizar datos, 14. *Véase también*
 modelo de datos relacional
 elegir identificador único, 7
 primera forma normal, 8-9
 segunda forma normal, 9
 tercera forma normal, 9-11
 normalizar el lenguaje de datos, definido,
 15
 número de punto flotante, definido, 57
 números enteros, generación de, 36

O

objetos. *Véase también* objetos de la base
 de datos; objetos de esquema
 en catálogos, 34
 identificadores para, 40
 nombrado de, 41
 otorgar acceso a, 19
 tomar posesión de, 132
 objetos de esquema. *Véase también*
 objetos de la base de datos;
 objetos; vistas
 activadores, 36
 activadores como, 330-331
 conjuntos de caracteres, 36
 cotejos, 36
 dominios, 35
 eliminar, 43
 generadores de secuencia, 36
 módulo de servidor de SQL, 36
 privilegios de acceso para, 128-
 129
 restricciones, 36
 rutinas invocadas por SQL, 36
 tablas base, 35
 transliteraciones, 36
 UDT (tipos definidos por el
 usuario), 36
 vistas, 35
 objetos de la base de datos. *Véase*
también objetos; objetos de
 octeto, definido, 55
 objetos SQL. *Véase* objetos
 ODBC API, desarrollo de, 419-420
 opción ABSOLUTE con la instrucción
 FETCH, 365-367
 opción ALL PRIVILEGES
 con la cláusula GRANT, 132
 utilizar, 136
 opción ASENSITIVE con cursores, 357
 opción CASCADE
 con esquemas, 43-44
 con procedimientos, 311
 con REVOKE, 136
 con tablas, 68-69
 opción CASCADE CONSTRAINTS, 93

opción CURRENT OF con UPDATE,
 369, 374
 operador de división (/), ejemplo de,
 239
 operador de multiplicación (*), 239
 operador de resta (-), 239
 operador desigual a (<>), 195-196
 operadores de comparación
 con ANY y SOME, 217-218
 con subconsultas, 284
 símbolos y ejemplos de, 195
 Oracle Express, descarga, 25
 Oracle, ventajas de, 22-23
 ordenamiento del cursor, vistazo general
 de, 359
 outer joins
 creación de, 266-269
 frente a inner joins, 264
 tipos de, 266-269

P

palabras clave
 AUTHORIZATION, 42
 con instrucciones DDL, 18
 DEFAULT CHARACTER SET,
 43
 definidas, 40
 no reservadas, 494-496
 reservadas, 492-494
 palabra clave BEFORE con activadores,
 333
 palabras clave no reservadas
 lista de, 494-496
 uso de, 40
 palabras clave reservadas
 lista de, 492-494
 uso de, 40
 parámetros. *Véase también* parámetros de
 entrada; parámetros de salida
 agregar a procedimientos, 307-
 310, 320-321
 como literales, 62
 con procedimientos, 319
 con rutinas invocadas por SQL,
 300
 declarar para módulos, 418
 definir en CREATE
 PROCEDURE, 308
 descripción de, 307
 instrucciones para, 307
 separar en SUBSTRING, 234
 tipos de, 307
 utilización de, 307
 variables host como, 368
 parámetros de entrada. *Véase también*
 parámetros de salida
 parámetros de salida. *Véase también*
 parámetros de entrada
 paréntesis (())
 con la función COUNT, 227
 con la instrucción CALL, 308
 con las palabras clave AND y OR,
 155
 con predicados, 158-159
 organizar en códigos, 96-97
 Parte 14. *Véase* SQL/XML
 Pascal, instrucciones SQL en, 408
 PL/I, instrucciones SQL en, 408
 precisión
 aplicar a números, 57
 especificar para intervalos, 61
 predicados. *Véase también* predicados
 de comparación cuantificados;
 condiciones de búsqueda
 ALL, 221
 ANY, 221
 BETWEEN, 199-200, 207
 combinar, 159, 195, 198
 con el marcador de posición
 <condición de búsqueda>, 153-
 154
 con la instrucción DELETE,
 187
 con paréntesis (()), 158-159
 con restricciones CHECK, 95-97
 en instrucciones SQL, 503
 evaluación de, 155
 evaluación en la cláusula WHERE,
 194
 EXISTS, 213-214, 220, 281, 292
 IN, 209-213, 220, 279-280
 incluir en la cláusula WHERE,
 157
 invertir condiciones para, 195
 LIKE, 203-206
 NULL, 200-201, 208
 para comparar datos, 194-198
 para la instrucción UPDATE,
 184
 subconsultas en, 209, 220-221
 predicados de comparación (>)
 combinar, 197, 207
 con restricciones, 95
 definir subconsultas con, 292
 descripción de, 194
 predicados de comparación
 cuantificados. *Véase también*
 predicados
 ALL, 218-219
 ANY, 216-218
 descripción de, 216
 SOME, 216-218
 utilización de, 219, 282-283

predicados LIKE
 combinar, 206
 en instrucciones SELECT, 205
 en la cláusula WHERE, 204
 palabra clave NOT en, 205
 símbolos utilizados en, 203-204
 utilización de, 208

primera forma normal, 8-9

privilegio DELETE con objetos de esquema, 129

privilegios
 agregar nombres de columna a, 134
 asociar con identificadores de autorización, 131-132
 definir para objetos de esquema, 128-129
 manejar, 139-140, 501
 nombres de rol, 126
 otorgar, 130
 otorgar a roles, 134
 otorgar en objetos, 131-132
 revocar, 130, 135-137, 139-140
 tipos para objetos de esquema, 129-130

privilegios de acceso
 definición, 128-129
 otorgar y restringir, 19

privilegios de seguridad. *Véase* privilegios

privilegios INSERT
 con objetos de esquema, 129
 otorgar a roles, 134
 otorgar en la tabla DISCOS_ COMPACTOS, 130-140
 revocar, 137

privilegios SELECT
 con objetos de esquema, 129
 otorgar a identificadores PUBLIC, 134-135
 otorgar a roles, 134
 otorgar en la tabla DISCOS_ COMPACTOS, 130-140
 otorgar en vistas, 139
 revocar, 136, 140

privilegios UPDATE
 con objetos de esquema, 129
 otorgar a roles, 134
 otorgar en la tabla DISCOS_ COMPACTOS, 130-140
 revocar, 137

procedimientos. *Véase también* rutinas invocadas por SQL
 agregar parámetros de entrada a, 307-310
 agregar parámetros de salida a, 320-321

agregar variables locales a, 311-313

capacidades de, 36

con módulos, 417

con repeticiones, 317

concepto, 301

creación de, 302-303, 319-320

crear y reemplazar, 310

definidos, 62

ejecutar con la instrucción CALL, 312

frente a funciones, 301-302, 321

instrucciones de los lenguajes de programación en, 407-408

invocados externamente, 62

invocar, 308

parámetros múltiples en, 308

parámetros para, 319

quitar, 311, 319

sintaxis, 301-303

utilización de, 303-305

utilizar para modificar datos, 309-310

y rutinas invocadas por SQL, 300

procedimientos almacenados. *Véase* procedimientos invocados por SQL

procedimientos invocados por SQL
 almacenar instrucciones SELECT como, 305
 creación de, 318-320, 507-508
 definidos, 17
 invocar, 305-306
 modificar, 310-311

programación orientada a objetos, definida, 17

programas, incrustar instrucciones en, 406-407

promedios,
 crear vistas de, 115

punto y coma (;) con instrucciones compuestas, 313

puntos de recuperación. *Véase también* la cláusula TO SAVEPOINT
 crear en transacciones, 392-394, 400
 liberar, 394

R

rangos, incluir puntos finales en, 97

RDBMS
 conectarse a, 25-26
 ejemplos de, 4
 utilización de, 21

referencias de tabla, separar, 149

registros de datos, representar, 6

registros DRG
 agregar a la tabla DISQUERAS_CD, 188
 eliminar de la tabla DISQUERAS_CD, 190

relación recursiva
 definida, 83

relación una-a-varias
 definida, 11
 ejemplo de, 12

relación una-a-una
 definida, 11
 ejemplos de, 11
 implementar, 12

relación varias-a-varias
 definida, 11
 ejemplos de, 12

relaciones
 cardinalidad mínima de, 13
 ejemplo de, 6
 entre objetos en catálogos, 34
 identificar, 14
 normalizar, 8
 relaciones entre, 13
 terminología, 7
 valores duplicados en, 8

repeticiones, nombrado, 317

resolver con self-join, 260-261

restricciones, 91

restricciones. *Véase también* restricciones de integridad
 aplazar en transacciones, 390-393
 CHECK, 95-97
 combinar columnas en, 79
 definición de, 52
 definidas, 68
 descripción de, 36
 dominio de, 98
 FOREIGN KEY, 83-87
 modelo de datos para, 91-94
 NOT NULL, 76-77, 93
 PRIMARY KEY, 79-82
 tipos de datos como, 6
 UNIQUE, 77-79, 93
 y sus relaciones, 91

restricciones CHECK
 afirmaciones como, 97-98
 agregar, 99
 definición de, 95-97, 500
 dominios como, 98

restricciones de columna
 agregar, 85
 creación de, 78
 definidas, 74-75

restricciones de dominio
 creación de, 98
 definidas, 74
 limitación de, 75

restricciones de integridad. *Véase también* restricciones
 categorías de, 74-75
 definidas, 74

restricciones de tabla
 claves foráneas como, 84, 86
 restricciones PRIMARY KEY como, 85

restricciones FOREIGN KEY
 agregar como restricciones de columna, 85
 cláusula <acción referencial activada> de, 89-91
 cláusula MATCH de, 88-89
 frente a claves primarias, 87-88
 lineamientos para, 84

restricciones FOREIGN KEY como, 84-85

restricciones para, 79-80

restricciones PRIMARY KEY
 como restricciones de tabla, 85
 con restricciones UNIQUE, 81-82
 frente a restricciones UNIQUE, 80-82

restricciones referenciales
 agregar, 499-500
 ejemplos de, 75
 FOREIGN KEY, 83-87

restricciones relacionadas con la tabla,
 tipos de, 74

restricciones UNIQUE
 agregar, 77-79, 93, 499-500
 con claves candidato, 80
 con restricciones PRIMARY KEY, 81-82
 ejemplos de, 75
 frente a PRIMARY KEY, 80-82

resultados de la consulta. *Véase también*
 conjuntos de resultados
 agrupar, 159-164
 convertir a XML, 435-436
 incluir columnas en, 148
 ligar con las variables host, 428
 orden de resultados arrojados, 163
 ordenamiento, 172
 organizar con la cláusula ORDER BY, 169

right outer join, 266

roles
 administración de, 139-140, 501
 crear y eliminar, 131
 otorgar, 137, 140

otorgar a identificadores de
 usuario, 138
 revocar, 138-140

ROUTINE_ROLE, autorizar, 127

rutinas invocadas por SQL. *Véase también* procedimientos
 descripción de, 36
 tipos de, 301
 vistazo general de, 300-301

S

SECOND con intervalos, 61

segunda forma normal, 9

self-join, creación de, 260-261

sensibilidad del cursor, opciones para, 357

sesión, definida, 51

sesión SQL, definida, 51

sesiones, vistazo general de, 126-128

signo de porcentaje (%) con LIKE, 203-205

símbolo de unión (&) con los indicadores
 de asignación, 422

sitio, descripción de, 32

sitios Web
 descarga de Oracle Express, 25
 MySQL Community Server, 25
 SQL Server Express, 25

software relacional, 16

SQL (Lenguaje de Consulta Estructurado)
 como lenguaje relacional de
 objetos, 17
 como lenguaje sin procedimiento, 15
 como sublenguaje, 15
 evolución de, 15-18
 frente a Oracle y SQL Server, 21

SQL incrustado
 crear instrucciones en, 407-408
 descripción de, 20, 352
 instrucciones, 512
 soporte para, 406-407
 utilización de, 414-416

SQL interactivo. *Véase* método de
 invocación directa

SQL Server
 ventajas de, 22-23
 utilización de, 26

SQL Server Express, descarga, 25

SQL/XML
 bases de, 434-436
 funciones, 439-440, 444, 513-514
 reglas de trazado para, 441-443
 soporte del fabricante para, 439
 tipo de datos, 437-439

subconsultas
 anidar, 286-288, 293
 arrojar filas con, 213, 278-283
 arrojar un valor con, 283-284
 con el predicado EXISTS, 281
 con el predicado IN, 279-280
 definir con predicados de
 comparación, 292
 descripción de, 278
 en el predicado IN, 211-212
 en instrucciones UPDATE, 293
 en la cláusula HAVING, 279
 en la cláusula SELECT, 279
 en la cláusula SET, 290, 294
 en la cláusula WHERE, 279
 en la instrucción UPDATE, 290
 en predicados, 209, 220-221, 503-504
 incluir en la cláusula SELECT, 283
 para eliminar datos, 291
 para insertar datos, 288-289
 requerimientos de, 280
 utilización de, 291-295, 505-507

subconsultas anidadas, 286-288, 293

subconsultas correlacionadas. *Véase también* consultas
 definidas, 213
 frente a joins, 286
 utilización de, 284-285

T

tabla de producto cartesiano
 producir la, 255

tablas. *Véase también* tablas base; tablas
 derivadas;
 alteración de tablas temporales, 67-68
 alterar y eliminar, 70-71, 499
 componentes de, 35
 consultar, 272-273, 505
 creación de, 50-53, 66-68, 498
 determinar la capacidad de
 actualización de, 116
 eliminar, 69
 nombradas, 52
 quitar restricciones ejemplo, 92
 soporte para, 37
 tipos de, 50
 unir, 254, 257-259, 264-265, 269-271, 304
 unir con nombres de columna
 compartidos, 261-263

- tablas base. *Véase también* tablas
 - creación de, 51
 - creadas como temporales locales, 51
 - declaradas como temporales locales, 51
 - descripción de, 35, 50
 - especificar partes de, 52
 - nombrado de, 52
 - persistentes, 50
 - temporales globales, 51
 - tipos de, 104
- tablas base como, 50-51
 - vaciar, 52
- tablas base persistentes
 - frente a tablas derivadas, 104
 - vistazo general de, 50
- tablas de transición con activadores, 332
- tablas derivadas. *Véase también* tablas
 - descripción de, 50
 - frente a tablas base persistentes, 104
- tablas SQL. *Véase* tablas
- tablas temporales. *Véase también* tablas
 - declarar en módulos, 417
 - designar, 52
 - propósito de, 54
- tablas temporales locales creadas, 51
- tablas temporales locales declaradas, 51
- tablas vistas, 50
- tamaño de diagnóstico, especificar para transacciones, 387-388
- tercera forma normal, 9-11
- terminales en rangos, 97
- theta-join, definida, 257
- tiempo, recuperar el valor, 237
- tipo de colección SQL, tipo de datos,
 - mapeo para los tipos de esquema XML, 442
- tipo de datos DECIMAL, 58
- tipo de datos INT
 - basar el dominio en, 98
 - con procedimientos, 312
- tipo de datos XML, 57
- tipos. *Véase* tipos de datos
- tipos de datos
 - asignar, 62
 - booleanos, 61
 - cambiar valores de, 244-245
 - construidos, 54
 - de fecha y hora, 58-59
 - de intervalo, 60-61
 - definidos por el usuario, 55, 63-64
 - en SQL/XML, 437-439
 - especificar por columnas, 54-55
 - frente a dominios, 6
 - herencia por columnas de vista, 105
 - numéricos, 57-58
 - para cadenas, 55-57
 - predefinidos, 54, 62
 - utilización de, 62-63
- tipos de datos BIGINT,
 - descripción de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos BOOLEAN
 - descripción de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos construidos, 54
- tipos de datos DATE
 - descripción de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos de cadena, 55-57
- tipos de datos de columna,
 - especificar, 54-55
- tipos de datos DECIMAL,
 - descripción de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos de dominio, trazado para los tipos de esquema XML, 442
- tipos de datos de esquema XML, trazado para, 441-442
 - trazado para los tipos de esquema XML, 442
- tipos de datos de fecha y hora
 - descripciones de, 58-59
 - frente a intervalos, 60
- tipos de datos de fila, trazado para los tipos de esquema XML, 442
- tipos de datos de intervalo
 - trazado para los tipos de esquema XML, 442
 - utilización de, 60-61
- tipos de datos de SQL. *Véase* tipos de datos
- tipos de datos distintivos de SQL, trazado para los tipos de esquema XML, 442
- tipos de datos DOUBLE PRECISION
 - descripción de, 58
 - trazado para los tipos de esquema XML, 442
- tipos de datos FLOAT
 - descripción de, 58
 - trazado para los tipos de esquema XML, 442
- tipos de datos INTEGER
 - descripción de, 58
 - trazado para los tipos de esquema XML, 442
- tipos de datos NUMERIC
 - descripciones de, 57-58
 - reglas para, 64
 - trazado para los tipos de esquema XML, 442
 - utilización de, 64
- tipos de datos predefinidos, 54, 62
- tipos de datos REAL
 - descripción de, 58
 - trazado para los tipos de esquema XML, 442
- tipos de datos SMALLINT
 - descripción de, 58
 - trazado para los tipos de esquema XML, 442
- tipos de datos TIME
 - descripciones de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos TIMESTAMP
 - descripciones de, 59
 - trazado para los tipos de esquema XML, 442
- tipos de datos ZONE, trazado para los tipos de esquema XML, 442
- tipos de registros, datos en, 6
- tipos de relaciones
 - una-a-una, 11
 - una-a-varias, 11
 - varias-a-varias, 11
- tipos distintos frente a tipos
 - estructurados, 63
- tipos estructurados frente a tipos
 - distintos, 63
- tipos XML, trazado para los tipos de esquema XML, 442
- tipos XMLSCHEMA, 438
- totales, visualización de los, 162
- transacciones
 - aplazar restricciones en, 390-393
 - consideraciones respecto al rendimiento, 397
 - creación de, 388
 - crear puntos de recuperación en, 392-394
 - definidas, 379
 - ejecución de, 395-396
 - finalizar, 395
 - iniciar, 389
 - instrucciones DDL en, 390
 - instrucciones relacionadas con, 380-381
 - la opción READ ONLY para, 382
 - la opción READ WRITE para, 382
 - lineamientos para, 398
 - modos para, 382
 - niveles de aislamiento para, 382-387
 - reinvertir, 396-397, 399-400
 - tamaño de diagnóstico para, 387-388

- utilización de, 398-400, 511
- y lecturas fantasma, 384-386
- y lecturas no repetibles, 384-385
- transliteraciones, 36
- trazado de la rutina, 32
- trazado de usuario, 32
- trazado para los tipos de esquema XML, 442
 - utilización de, 61
- tuplas. *Véase también* cardinalidad
 - mínima
 - almacenaje de datos en, 6
 - terminología, 7
 - violación de la primera forma normal, 8

U

- UDT (tipos definidos por el usuario)
 - creación de, 63-64
 - descripción de, 36, 55
 - soporte para, 301
 - utilización de, 55
- Unicode, trazado para, 441, 443
- usuarios de autorización,
 - determinado de, 128
- usuarios, determinar valores de, 128

V

- valor CURRENT_ROLE, 246
- valor CURRENT_USER
 - con sesiones, 128
 - descripción de, 246
- valor de expresión CASE, 241-244, 249
- valor de expresión CAST, 244-245, 249, 410
- valor SYSDATE, 237
- valor SYSTEM_USER, 246
- valores. *Véase también* valores especiales; valores SQL
 - actualizar, 189
 - actualizar desde la instrucción SELECT, 185-186
 - agregar juntos, 160
 - arrojar, 203-206
 - arrojar con MIN y MAX, 229-231
 - asignar a variables, 311
 - enlistar para columnas, 95-97

- insertar desde la instrucción SELECT, 180-182
- referenciar con activadores, 334-335
- renombrado de, con CASE, 242
- valores A_LA_MANO
 - en orden ascendente, 169
 - utilizar la cláusula HAVING con, 165-166
- valores de datos numéricos, generación de, 36
- valores de fecha y hora, conversión de, 245
- valores duplicados,
 - prevenir, 79
- valores EN_EXISTENCIA
 - arrojar, 158-159
 - basar filas en, 156-157
 - promediar, 185
- valores especiales, 245-247. *Véase también* valores
- valores nulos
 - arrojar, 200-203
 - autorizar, 79
 - como marcas, 76
 - comparaciones con, 197
 - en columnas, 69
 - en la columna PRECIO, 162-163
 - especificar para INSERT INTO, 178-179
 - manejo de, 412
 - nombres de rol como, 126, 128
 - utilización de, 61
 - y las restricciones NOT NULL, 76-77
- valores por defecto de columna
 - especificar, 64-65
- valores similares, arrojar, 203-206
- valores SQL, representar en XML, 441. *Véase también* valores
- valores XML, representar en SQL, 441
- variables. *Véase también* variables host
 - agregar a procedimientos, 311-313
 - asignar valores a, 311
 - declarar para cursores, 372
 - pasar datos desde, 410
- variables host de indicador, declarar, 412, 415
- variables locales, agregar a procedimientos, 311-313

- verdaderas, evaluar filas como, 155
- vistas. *Véase también* objetos de esquema; vistas actualizables
 - agregar, 500
 - agregar a las bases de datos, 119-120
 - alterar, 118
 - basar en tablas múltiples, 105-107, 118
 - capacidades de, 105
 - con columnas, 105
 - con expresiones de consulta, 110-111
 - convocar con instrucciones SELECT, 304
 - cortes de línea en, 110-111
 - creación de, 112, 118
 - de promedios, 115
 - definidas, 34, 104
 - descripción de, 35
 - desplegar columnas en, 116
 - en INFORMATION_SCHEMA, 34
 - espaciado en, 110-111
 - especificar operaciones en, 108
 - insertos en, 176
 - limitaciones de las, 305
 - nombrado, 109-111
 - ocultar la complejidad de la consulta con, 105-106
 - otorgar privilegios SELECT en, 139
 - para modificar datos, 108
 - quitar de las bases de datos, 117, 119
 - sintaxis, 109
 - unir datos con, 108
 - ventajas de, 104-105
- vistas actualizables. *Véase también* vistas CHECK OPTION con, 117
 - creación de, 114-116
- vistazo general sobre las tablas temporales globales, 51

X

- XML (Lenguaje de Marcado Extensible)
 - agregar comentarios en, 436
 - vistazo general de, 434-437