25

Acceso a bases de datos con JDBC

Es un error capital especular antes de tener datos.

—Arthur Conan Doyle

Ahora ven, escríbelo en una tablilla, grábalo en un libro, y que dure hasta el último día, para testimonio.

—La Santa Biblia, Isaías 30:8

Primero consiga los hechos, y después podrá distorsionarlos según le parezca.

—Mark Twain

Me gustan dos tipos de hombres: domésticos y foráneos.

—Mae West

OBJETIVOS

En este capítulo aprenderá a:

- Utilizar los conceptos acerca de las bases de datos relacionales.
- Utilizar el Lenguaje de Consulta Estructurado (SQL) para obtener y manipular los datos de una base de datos.
- Utilizar la API JDBC™ del paquete `java.sql` para acceder a las bases de datos.
- Utilizar la interfaz `RowSet` del paquete `javax.sql` para manipular bases de datos.
- Utilizar el descubrimiento de controladores de JDBC automático de JDBC 4.0.
- Utilizar objetos `PreparedStatement` para crear instrucciones de SQL precompiladas con parámetros.
- Conocer cómo el procesamiento de transacciones hace que las aplicaciones de datos sea más robusto.

Plan general

25.1 Introducción

Una **base de datos** es una colección organizada de datos. Existen diversas estrategias para organizar datos y facilitar el acceso y la manipulación. Un **sistema de administración de bases de datos (DBMS)** proporciona los mecanismos para almacenar, organizar, obtener y modificar datos para muchos usuarios. Los sistemas de administración de bases de datos permiten el acceso y almacenamiento de datos sin necesidad de preocuparse por su representación interna.

En la actualidad, los sistemas de bases de datos más populares son las bases de datos relacionales, en donde los datos se almacenan sin considerar su estructura física (sección 25.2). Un lenguaje llamado **SQL** es el lenguaje estándar internacional que se utiliza casi universalmente con las bases de datos relacionales para realizar **consultas** (es decir, para solicitar información que satisfaga ciertos criterios) y para manipular datos.

Algunos **sistemas de administración de bases de datos relacionales (RDBMS)** populares son Microsoft SQL Server, Oracle, Sybase, IBM DB2, Informix, PostgreSQL y MySQL. El JDK incluye ahora un RDBMS puro de Java, conocido como Java DB (la versión de Sun de Apache Derby). En este capítulo presentaremos ejemplos utilizando MySQL y Java DB.[1]

1. MySQL es uno de los sistemas de administración de bases de datos de código fuente abierto más populares de la actualidad. Al momento de escribir este libro todavía no soportaba JDBC 4, que forma parte de Java SE 6 (Mustang). Sin embargo, el sistema Java DB de Sun, que está basado en el sistema de administración de bases de datos de código fuente abierto Apache Derby y se incluye con el JDK 1.6.0 de Sun, sí ofrece soporte para JDSBC 4. En las secciones 25.8 a 25.10 utilizamos MySQL y JDBC 3, y en la sección 25.11 utilizamos Java DB y JDBC 4.

Los programas en Java se comunican con las bases de datos y manipulan sus datos utilizando la **API JDBC™**. Un **controlador de JDBC** permite a las aplicaciones de Java conectarse a una base de datos en un DBMS específico, y nos permite manipular esa base de datos mediante la API JDBC.

Observación de ingeniería de software 25.1

Al utilizar la API JDBC, los desarrolladores pueden modificar el DBMS subyacente sin necesidad de modificar el código de Java que accede a la base de datos.

La mayoría de los sistemas de administración de bases de datos populares incluyen ahora controladores de JDBC. También hay muchos controladores de JDBC de terceros disponibles. En este capítulo presentaremos la tecnología JDBC y la emplearemos para manipular bases de datos de MySQL y Java DB. Las técnicas que demostraremos aquí también pueden usarse para manipular otras bases de datos que tengan controladores de JDBC. Consulte la documentación de su DBMS para determinar si incluye un controlador de JDBC. Incluso si su DBMS no viene con un controlador de JDBC, muchos distribuidores independientes proporcionan estos controladores para una amplia variedad de sistemas DBMS.

Para obtener más información acerca de JDBC, visite la página:

```
java.sun.com/javase/technologies/database/index.jsp
```

Este sitio contiene información relacionada con JDBC, incluyendo las especificaciones de JDBC, preguntas frecuentes (FAQs) acerca de JDBC, un centro de recursos de aprendizaje y descargas de software para buscar controladores de JDBC para su DBMS,

```
developers.sun.com/product/jdbc/drivers/
```

Este sitio proporciona un motor de búsqueda para ayudarlo a localizar los controladores apropiados para su DBMS.

25.2 Bases de datos relacionales

Una **base de datos relacional** es una representación lógica de datos que permite acceder a éstos sin necesidad de considerar su estructura física. Una base de datos relacional almacena los datos en **tablas**. En la figura 25.1 se muestra una tabla de ejemplo que podría utilizarse en un sistema de personal. El nombre de la tabla es `Empleado`, y su principal propósito es almacenar los atributos de un empleado. Las tablas están compuestas de **filas**, y las filas, de **columnas** en las que se almacenan los valores. Esta tabla consiste de seis filas. La columna `Numero` de cada fila en esta tabla es su **clave primaria**: una columna (o grupo de columnas) en una tabla que tiene un valor único, el cual no puede duplicarse en las demás filas. Esto garantiza que cada fila pueda identificarse por su clave primaria. Algunos buenos ejemplos de columnas con clave primaria son un número de seguro social, un número de identificación de empleado y un número de pieza en un sistema de inventario, ya que se garantiza que los valores en cada una de esas columnas serán únicos. Las filas de la figura 25.1 se muestran en orden, con base en la clave primaria. En este caso, las filas se muestran en orden ascendente; también podríamos utilizar el orden descendente.

	Número	Nombre	Departamento	Salario	Ubicación
	23603	Jones	413	1100	New Jersey
	24568	Kerwin	413	2000	New Jersey
Fila	34589	Larson	642	1800	Los Angeles
	35761	Myers	611	1400	Orlando
	47132	Neumann	413	9000	New Jersey
	78321	Stephens	611	8500	Orlando
	Clave primaria		Columna		

Figura 25.1 | Datos de ejemplo de la tabla `Empleado`.

No se garantiza que las filas en las tablas se almacenen en un orden específico. Como lo demostraremos en un ejemplo más adelante, los programas pueden especificar criterios de ordenamiento al solicitar datos de una base de datos.

Cada columna de la tabla representa un atributo de datos distinto. Las filas generalmente son únicas (por clave primaria) dentro de una tabla, pero los valores de columnas específicas pueden duplicarse entre filas. Por ejemplo, tres filas distintas en la columna `Departamento` de la tabla `Empleado` contienen el número `413`.

A menudo los distintos usuarios de una base de datos se interesan en datos diferentes, y en relaciones distintas entre esos datos. La mayoría de los usuarios requieren solamente de ciertos subconjuntos de las filas y columnas. Para obtener estos subconjuntos, utilizamos consultas para especificar cuáles datos se deben seleccionar de una tabla. Utilizamos SQL para definir consultas complejas que seleccionen datos de una tabla. Por ejemplo, podríamos seleccionar datos de la tabla `Empleado` para crear un resultado que muestre en dónde se ubican los departamentos, y presentar los datos ordenados en forma ascendente, por número de departamento. Este resultado se muestra en la figura 25.2. Hablaremos sobre las consultas de SQL en la sección 25.4.

```
Departamento      Ubicación

413               New Jersey
611               Orlando
642               Los Angeles
```

Figura 25.2 | Resultado de seleccionar distintos datos de `Departamento` y `Ubicacion` de la tabla `Empleado`.

25.3 Generalidades acerca de las bases de datos relacionales: la base de datos `libros`

Ahora veremos las generalidades sobre las bases de datos relacionales, y para ello emplearemos una base de datos llamada `libros`, misma que creamos para este capítulo. Antes de hablar sobre SQL, veremos una descripción general de las tablas de la base de datos `libros`. Utilizaremos esta base de datos para presentar varios conceptos de bases de datos, incluyendo el uso de SQL para obtener información de la base de datos y manipular los datos. Le proporcionaremos una secuencia de comandos (script) para crear la base de datos. En el directorio de ejemplos para este capítulo (en el CD que acompaña al libro) encontrará la secuencia de comandos. En la sección 25.5 le explicaremos cómo utilizar esta secuencia de comandos.

La base de datos consiste de tres tablas: `autores`, `isbnAutor` y `titulos`. La tabla `autores` (descrita en la figura 25.3) consta de tres columnas que mantienen el número único de identificación de cada autor, su nombre de pila y apellido. La figura 25.4 contiene datos de ejemplo de la tabla `autores` de la base de datos `libros`.

La tabla `isbnAutor` (descrita en la figura 25.5) consta de dos columnas que representan a cada ISBN y el correspondiente número de ID del autor. Esta tabla asocia a los autores con sus libros. Ambas columnas son claves externas que representan la relación entre las tablas `autores` y `titulos`; una fila en la tabla `autores` puede estar asociada con muchas filas en la tabla `titulos` y viceversa. La figura 25.6 contiene datos de ejemplo de la tabla `isbnAutor` de la base de datos `libros`. [*Nota:* para ahorrar espacio, dividimos el contenido de esta tabla en

Columna	Descripción
`idAutor`	El número de identificación (ID) del autor en la base de datos. En la base de datos `libros`, esta columna de enteros se define como **autoincrementada**; para cada fila insertada en esta tabla, la base de datos incrementa automáticamente el valor de `idAutor` por 1 para asegurar que cada fila tenga un `idAutor` único. Esta columna representa la clave primaria de la tabla.
`nombrePila`	El nombre de pila del autor (una cadena).
`apellidoPaterno`	El apellido paterno del autor (una cadena).

Figura 25.3 | La tabla `autores` de la base de datos `libros`.

idAutor	nombrePila	apellidoPaterno
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Figura 25.4 | Datos de ejemplo de la tabla `autores`.

Columna	Descripción
`idAutor`	El número de identificación (ID) del autor, una clave externa para la tabla `autores`.
`isbn`	El ISBN para un libro, una clave externa para la tabla `titulos`.

Figura 25.5 | La tabla `isbnAutor` de la base de datos `libros`.

idAutor	isbn	idAutor	isbn
1	0131869000	2	0131450913
2	0131869000	1	0131828274
1	0131483986	2	0131828274
2	0131483986	3	0131450913
1	0131450913	4	0131828274

Figura 25.6 | Datos de ejemplo de la tabla `isbnAutor` de `libros`.

dos columnas, cada una de las cuales contiene las columnas `idAutor` y `isbn`.] La columna `idAutor` es una **clave externa**; una columna en esta tabla que coincide con la columna de la clave primaria en otra tabla (por ejemplo, `idAutor` en la tabla `autores`). Las claves externas se especifican al crear una tabla. La clave externa ayuda a mantener la **Regla de integridad referencial**: todo valor de una clave externa debe aparecer como el valor de la clave primaria de otra tabla. Esto permite al DBMS determinar si el valor de `idAutor` para un libro específico es válido. Las claves externas también permiten seleccionar datos relacionados en varias tablas para fines analíticos; a esto se conoce como **unir** los datos.

La tabla `titulos` descrita en la figura 25.7 consiste en cuatro columnas que representan el ISBN, el título, el número de edición y el año de copyright. La tabla está en la figura 25.8.

Hay una relación de uno a varios entre una clave primaria y su correspondiente clave externa (por ejemplo, una editorial puede publicar muchos libros). Una clave externa puede aparecer varias veces en su propia tabla, pero sólo una vez (como clave primaria) en otra tabla. La figura 25.9 es un **diagrama de relación de entidades (ER)** para la base de datos `libros`. Este diagrama muestra las tablas en la base de datos, así como las relaciones entre ellas. El primer compartimiento en cada cuadro contiene el nombre de la tabla. Los nombres en cursiva son claves primarias. La clave primaria de una tabla identifica en forma única a cada fila. Cada fila debe tener un valor en la clave primaria, y éste debe ser único en la tabla. A esto se le conoce como **Regla de integridad de entidades**.

Error común de programación 25.1

Si no se proporciona un valor para cada columna en una clave primaria, se quebranta la Regla de Integridad de Entidades y el DBMS reporta un error.

Columna	Descripción
`isbn`	El número ISBN del libro (una cadena). La clave primaria de la tabla. ISBN son las siglas de "International Standard Book Number" (Número internacional normalizado para libros); un esquema de numeración utilizado por las editoriales en todo el mundo para dar a cada libro un número de identificación único.
`titulo`	Título del libro (una cadena).
`numeroEdicion`	Número de edición del libro (un entero).
`copyright`	Año de edición (copyright) del libro (una cadena).

Figura 25.7 | La tabla `titulos` de la base de datos `libros`.

isbn	titulo	numeroEdicion	copyright
0131869000	Visual Basic How to Program	3	2006
0131525239	Visual C# How to Program	2	2006
0132222205	Java How to Program	7	2007
0131857576	C++ How to Program	5	2005
0132404168	C How to Program	5	2007
0131450913	Internet & World Wide Web How to Program	3	2004

Figura 25.8 | Datos de ejemplo para la tabla `titulos` de la base de datos `libros`.

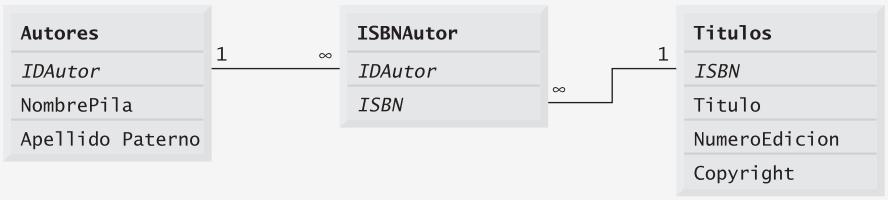

Figura 25.9 | Relaciones de las tablas en la base de datos `libros`.

Error común de programación 25.2

Al proporcionar el mismo valor para la clave primaria en varias filas, el DBMS reporta un error.

Las líneas que conectan las tablas en la figura 25.9 representan las relaciones entre las tablas. Considere la línea entre las tablas `isbnAutor` y `autores`. En el extremo de la línea que va a `autores` hay un `1`, y en el extremo que va a `isbnAutor` hay un símbolo de infinito (∞), el cual indica una **relación de uno a varios** en la que cualquier autor de la tabla `autores` puede tener un número arbitrario de libros en la tabla `isbnAutor`. Observe que la línea de relación enlaza a la columna `idAutor` en la tabla `autores` (su clave primaria) con la columna `idAutor` en la tabla `isbnAutor` (es decir, su clave externa). La columna `idAutor` en la tabla `isbnAutor` es una clave externa.

Error común de programación 25.3

Al proporcionar un valor de clave externa que no aparezca como valor de clave primaria en otra tabla, se quebranta la Regla de Integridad Referencial y el DBMS reporta un error.

La línea entre las tablas `titulos` e `isbnAutor` muestra otra relación de uno a varios; un título puede ser escrito por cualquier número de autores. De hecho, el único propósito de la tabla `isbnAutor` es proporcionar una relación de varios a varios entre las tablas `autores` y `titulos`; un autor puede escribir cualquier número de libros y un libro puede tener cualquier número de autores.

25.4 SQL

En esta sección veremos una descripción general acerca de SQL, en el contexto de nuestra base de datos `libros`. En los ejemplos posteriores y en los ejemplos de los capítulos 26 a 28, usted podrá utilizar las consultas de SQL que describimos aquí.

Las palabras clave de SQL enlistadas en la figura 25.10 se describen en las siguientes subsecciones, dentro del contexto de consultas e instrucciones de SQL completas. Las demás palabras clave de SQL se encuentran más allá del alcance de este libro. Para aprender acerca de las otras palabras clave, consulte la guía de referencia de SQL que proporciona el distribuidor del RDBMS que usted utilice. [*Nota:* para obtener más información acerca de SQL, consulte los recursos Web en la sección 25.15 y las lecturas recomendadas que se enlistan al final de este capítulo].

Palabra clave de SQL	Descripción
`SELECT`	Obtiene datos de una o más tablas.
`FROM`	Las tablas involucradas en la consulta. Se requiere para cada `SELECT`.
`WERE`	Los criterios de selección que determinan cuáles filas se van a recuperar, eliminar o actualizar. Es opcional en una consulta o instrucción de SQL.
`GROUP BY`	Criterios para agrupar filas. Es opcional en una consulta `SELECT`.
`ORDER BY`	Criterios para ordenar filas. Es opcional en una consulta `SELECT`.
`INNER JOIN`	Fusionar filas de varias tablas.
`INSERT`	Insertar filas en una tabla especificada.
`UPDATE`	Actualizar filas en una tabla especificada.
`DELETE`	Eliminar filas de una tabla especificada.

Figura 25.10 | Palabras clave para consultas de SQL.

25.4.1 Consulta básica SELECT

Veamos ahora varias consultas de SQL que extraen información de la base de datos `libros`. Una consulta de SQL "selecciona" filas y columnas de una o más tablas en una base de datos. Dichas selecciones se llevan a cabo mediante consultas con la palabra clave `SELECT`. La forma básica de una consulta `SELECT` es:

```
SELECT * FROM nombreDeTabla
```

en la consulta anterior, el **asterisco** (*) indica que deben obtenerse todas las columnas de la tabla *nombreDeTabla*. Por ejemplo, para obtener todos los datos en la tabla `autores`, podemos utilizar la siguiente consulta:

```
SELECT * FROM autores
```

La mayoría de los programas no requieren todos los datos en la tabla. Para obtener sólo ciertas columnas de una tabla, reemplace el asterisco (*) con una lista separada por comas de los nombres de las columnas. Por ejemplo, para obtener solamente las columnas `idAutor` y `apellidoPaterno` para todas las filas en la tabla `autores`, utilice la siguiente consulta:

```
SELECT idAutor, apellidoPaterno FROM autores
```

Esta consulta devuelve los datos que se muestran en la figura 25.11.

idAutor	apellidoPaterno
1	Deitel
2	Deitel
3	Nieto
4	Santry

Figura 25.11 | Datos de ejemplo para `idAutor` y `apellidoPaterno` de la tabla `autores`.

Observación de ingeniería de software 25.2

Para la mayoría de las consultas, no debe utilizarse el asterisco (*) para especificar nombres de columnas. En general, los programadores procesan los resultados sabiendo de antemano el orden de las columnas en el resultado; por ejemplo, al seleccionar `idAutor` y `apellidoPaterno` de la tabla `autores` nos aseguramos que las columnas aparezcan en el resultado con `idAutor` como la primera columna y `apellidoPaterno` como la segunda. Generalmente los programas procesan las columnas de resultado especificando el número de columna en el resultado (empezando con el número 1 para la primera columna). Al seleccionar las columnas por nombre, también evitamos devolver columnas innecesarias y nos protegemos contra los cambios en el orden actual de las columnas en la(s) tabla(s).

Error común de programación 25.4

Si un programador supone que las columnas siempre se devuelven en el mismo orden de una consulta que utiliza el asterisco (*), el programa podría procesar los resultados incorrectamente. Si cambia el orden de las columnas en la(s) tabla(s), o si se agregan más columnas posteriormente, el orden de las columnas en el resultado cambia de manera acorde.

25.4.2 La cláusula `WHERE`

En la mayoría de los casos es necesario localizar, en una base de datos, filas que cumplan con ciertos **criterios de selección**. Sólo se seleccionan las filas que cumplan con los criterios de selección (formalmente llamados **predicados**). SQL utiliza la **cláusula** `WHERE` opcional en una consulta para especificar los criterios de selección para la misma. La forma básica de una consulta con criterios de selección es:

```
SELECT nombreDeColumna1, nombreDeColumna2, ... FROM nombreDeTabla WHERE criterios
```

Por ejemplo, para seleccionar las columnas `titulo`, `numeroEdicion` y `copyright` de la tabla `titulos` para las cuales la fecha de `copyright` sea mayor que `2005`, utilice la siguiente consulta:

```
SELECT titulo, numeroEdicion, copyright
    FROM titulos
    WHERE copyright > '2005'
```

En la figura 25.12 se muestra el resultado de la consulta anterior. Los criterios de la cláusula `WHERE` pueden contener los operadores <, >, <=, >=, =, <> y `LIKE`. El operador `LIKE` se utiliza para **relacionar patrones** con los caracteres comodines **porcentaje (%)** y **guión bajo (_)**. El relacionar patrones permite a SQL buscar cadenas que coincidan con un patrón dado.

Un patrón que contenga un carácter de porcentaje (%) busca cadenas que tengan cero o más caracteres en la posición del carácter de porcentaje en el patrón. Por ejemplo, la siguiente consulta localiza las filas de todos los autores cuyo apellido paterno empiece con la letra `D`:

```
SELECT idAutor, nombrePila, apellidoPaterno
    FROM autores
    WHERE apellidoPaterno LIKE 'D%'
```

La consulta anterior selecciona las dos filas que se muestran en la figura 25.13, ya que dos de los cuatro autores en nuestra base de datos tienen un apellido paterno que empieza con la letra `D` (seguida de cero o más caracteres). El signo de `%` y el operador `LIKE` de la cláusula `WHERE` indican que puede aparecer cualquier número de caracteres después de la letra `D` en la columna `apellidoPaterno`. Observe que la cadena del patrón está encerrada entre caracteres de comilla sencilla.

titulo	numeroEdicion	copyright
Visual C# How to Program	2	2006
Visual Basic 2005 How to Program	3	2006
Java How to Program	7	2007
C How to Program	5	2005

Figura 25.12 | Ejemplos de títulos con fechas de copyright posteriores a 2005 de la tabla `titulos`.

idAutor	nombrePila	apellidoPaterno
1	Harvey	Deitel
2	Paul	Deitel

Figura 25.13 | Autores, cuyo apellido paterno empieza con `D` de la tabla `autores`.

Tip de portabilidad 25.1

Consulte la documentación de su sistema de bases de datos para determinar si SQL es susceptible al uso de mayúsculas en su sistema, y para determinar la sintaxis de las palabras clave de SQL (por ejemplo, ¿deben estar completamente en mayúsculas, completamente en minúsculas o puede ser una combinación de ambas?).

Tip de portabilidad 25.2

Lea cuidadosamente la documentación de su sistema de bases de datos para determinar si éste soporta el operador `LIKE`. El SQL que describimos es soportado por la mayoría de los RDBMS, pero siempre es buena idea comprobar las características de SQL que soporta su RDBMS.

Un guión bajo (_) en la cadena del patrón indica un carácter comodín individual en esa posición. Por ejemplo, la siguiente consulta localiza las filas de todos los autores cuyo apellido paterno empiece con cualquier carácter (lo que se especifica con _), seguido por la letra `o`, seguida por cualquier número de caracteres adicionales (lo que se especifica con `%`):

```
SELECT idAutor, nombrePila, apellidoPaterno
   FROM autores
   WHERE apellidoPaterno LIKE '_o%'
```

La consulta anterior produce la fila que se muestra en la figura 25.14, ya que sólo un autor en nuestra base de datos tiene un apellido paterno que contiene la letra `o` como su segunda letra.

idAutor	nombrePila	apellidoPaterno
3	Andrew	Goldberg

Figura 25.14 | El único autor de la tabla `autores` cuyo apellido paterno contiene `o` como la segunda letra.

25.4.3 La cláusula ORDER BY

Las filas en el resultado de una consulta pueden ordenarse en forma ascendente o descendente, mediante el uso de la **cláusula** `ORDER BY` opcional. La forma básica de una consulta con una cláusula `ORDER BY` es:

```
SELECT nombreDeColumna1, nombreDeColumna2 , ... FROM nombreDeTabla ORDER BY columna ASC
SELECT nombreDeColumna1, nombreDeColumna2 , ... FROM nombreDeTabla ORDER BY columna DESC
```

en donde `ASC` especifica el orden ascendente (de menor a mayor), `DESC` especifica el orden descendente (de mayor a menor) y *columna* especifica la columna en la cual se basa el ordenamiento. Por ejemplo, para obtener la lista de autores en orden ascendente por apellido paterno (figura 25.15), utilice la siguiente consulta:

```
SELECT idAutor, nombrePila, apellidoPaterno
    FROM autores
    ORDER BY apellidoPaterno ASC
```

Observe que el orden predeterminado es ascendente, por lo que `ASC` es opcional. Para obtener la misma lista de autores en orden descendente por apellido paterno (figura 25.16), utilice la siguiente consulta:

```
SELECT idAutor, nombrePila, apellidoPaterno
    FROM autores
    ORDER BY apellidoPaterno DESC
```

Pueden usarse varias columnas para ordenar mediante una cláusula `ORDER BY`, de la siguiente forma:

```
ORDER BY columna1 tipoDeOrden, columna2 tipoDeOrden, ...
```

en donde *tipoDeOrden* puede ser `ASC` o `DESC`. Observe que el *tipoDeOrden* no tiene que ser idéntico para cada columna. La consulta:

```
SELECT idAutor, nombrePila, apellidoPaterno
    FROM autores
    ORDER BY apellidoPaterno, nombrePila
```

idAutor	nombrePila	apellidoPaterno
4	David	Choffnes
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg

Figura 25.15 | Datos de ejemplo de la tabla `autores` ordenados en forma ascendente por la columna `apellidoPaterno`.

idAutor	nombrePila	apellidoPaterno
3	Andrew	Goldberg
1	Harvey	Deitel
2	Paul	Deitel
4	David	Choffnes

Figura 25.16 | Datos de ejemplo de la tabla `autores` ordenados en forma descendente por la columna `apellidoPaterno`.

ordena en forma ascendente todas las filas por apellido paterno, y después por nombre de pila. Si varias filas tienen el mismo valor de apellido paterno, se devuelven ordenadas por nombre de pila (figura 25.17).

Las cláusulas `WHERE` y `ORDER BY` pueden combinarse en una consulta. Por ejemplo, la consulta:

```
SELECT isbn, titulo, numeroEdicion, copyright, precio
    FROM titulos
    WHERE titulo LIKE '%How to Program'
    ORDER BY titulo ASC
```

devuelve el `isbn`, `titulo`, `numeroEdicion`, `copyright` y `precio` de cada libro en la tabla `titulos` que tenga un `titulo` que termine con `"How to Program"` y los ordena en forma ascendente, por `titulo`. El resultado de la consulta se muestra en la figura 25.18.

idAutor	nombrePila	apellidoPaterno
4	David	Choffner
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg

Figura 25.17 | Datos de ejemplo de autores de la tabla `autores` ordenados de manera ascendente, por las columnas `apellidoPaterno` y `nombrePila`.

isbn	titulo	numeroEdicion	copyright
0132404168	C How to Program	5	2007
0131857576	C++ How to Program	5	2005
0131450913	Internet & World Wide Web How to Program	3	2004
0132222205	Java How to Program	7	2007
0131869000	Visual Basic 2005 How to Program	3	2006
013152539	Visual C# How to Program	2	2006

Figura 25.18 | Ejemplos de libros de la tabla `titulos` cuyos títulos terminan con `How to Program`, y están ordenados en forma ascendente por medio de la columna `titulo`.

25.4.4 Cómo fusionar datos de varias tablas: `INNER JOIN`

Los diseñadores de bases de datos a menudo dividen los datos en tablas separadas para asegurarse de no guardar información redundante. Por ejemplo, la base de datos `libros` tiene las tablas `autores` y `titulos`. Utilizamos una tabla `isbnAutor` para almacenar los datos de la relación entre los autores y sus correspondientes títulos. Si no separáramos esta información en tablas individuales, tendríamos que incluir la información del autor con cada entrada en la tabla `titulos`. Esto implicaría almacenar información duplicada sobre los autores en la base de datos, para quienes hayan escrito varios libros. A menudo es necesario fusionar datos de varias tablas en un solo resultado. Este proceso, que se le conoce como unir las tablas, se especifica mediante un operador `INNER JOIN` en la consulta. Un operador `INNER JOIN` fusiona las filas de dos tablas al relacionar los valores en columnas que sean comunes para las dos tablas. La forma básica de un `INNER JOIN` es:

```
SELECT nombreDeColumna1, nombreDeColumna2, ...
FROM tabla1
```

```
INNER JOIN tabla2
    ON tabla1.nombreDeColumna = tabla2.nombreDeColumna
```

La **cláusula ON** de `INNER JOIN` especifica las columnas de cada tabla que se comparan para determinar cuáles filas se fusionan. Por ejemplo, la siguiente consulta produce una lista de autores acompañada por los ISBNs para los libros escritos por cada autor.

```
SELECT nombrePila, apellidoPaterno, isbn
FROM autores
INNER JOIN isbnAutor
    ON autores.idAutor = isbnAutor.idAutor
ORDER BY apellidoPaterno, nombrePila
```

La consulta fusiona los datos de las columnas `nombrePila` y `apellidoPaterno` de la tabla `autores` con la columna `isbn` de la tabla `isbnAutor`, ordenando el resultado en forma ascendente por `apellidoPaterno` y `nombrePila`. Observe el uso de la sintaxis *nombreDeTabla.nombreDeColumna* en la cláusula `ON`. Esta sintaxis (a la que se le conoce como **nombre calificado**) especifica las columnas de cada tabla que deben compararse para unir las tablas. La sintaxis "*nombreDeTabla*." es requerida si las columnas tienen el mismo nombre en ambas tablas. La misma sintaxis puede utilizarse en cualquier consulta para diferenciar entre columnas de tablas distintas que tengan el mismo nombre. En algunos sistemas pueden utilizarse nombres de tablas calificados con el nombre de la base de datos para realizar consultas entre varias bases de datos. Como siempre, la consulta puede contener una cláusula `ORDER BY`. En la figura 25.19 se muestra una parte de los resultados de la consulta anterior, ordenados por `apellidoPaterno` y `nombrePila`. [*Nota:* para ahorrar espacio dividimos el resultado de la consulta en dos columnas, cada una de las cuales contiene a las columnas `nombrePila`, `apellidoPaterno` e `isbn`].

Observación de ingeniería de software 25.3

Si una instrucción de SQL incluye columnas de varias tablas que tengan el mismo nombre, en la instrucción se debe anteponer a los nombres de esas columnas los nombres de sus tablas y el operador punto (por ejemplo, `autores.idAutor`*).*

Error común de programación 25.5

Si no se califican los nombres para las columnas que tengan el mismo nombre en dos o más tablas se produce un error.

nombrePila	apellidoPaterno	isbn	nombrePila	apellidoPaterno	isbn
David	Choffnes	0131828274	Paul	Deitel	0131525239
Harvey	Deitel	0131525239	Paul	Deitel	0132404168
Harvey	Deitel	0132404168	Paul	Deitel	0131869000
Harvey	Deitel	0131869000	Paul	Deitel	0132222205
Harvey	Deitel	0132222205	Paul	Deitel	0131450913
Harvey	Deitel	0131450913	Paul	Deitel	0131525239
Harvey	Deitel	0131525239	Paul	Deitel	0131857576
Harvey	Deitel	0131857576	Paul	Deitel	0131828274
Harvey	Deitel	0131828274	Andrew	Goldberg	0131450913

Figura 25.19 | Ejemplo de datos de autores e ISBNs para los libros que han escrito, en orden ascendente por `apellidoPaterno` y `nombrePila`.

25.4.5 La instrucción INSERT

La instrucción `INSERT` inserta una fila en una tabla. La forma básica de esta instrucción es:

```
INSERT INTO nombreDeTabla ( nombreDeColumna1, nombreDeColumna2, ..., nombreDeColumnaN )
   VALUES ( valor1, valor2, ..., valorN )
```

en donde *nombreDeTabla* es la tabla en la que se va a insertar la fila. El *nombreDeTabla* va seguido de una lista separada por comas de nombres de columnas entre paréntesis (esta lista no es requerida si la operación `INSERT` especifica un valor para cada columna de la tabla en el orden correcto). La lista de nombres de columnas va seguida por la palabra clave `VALUES` de SQL, y una lista separada por comas de valores entre paréntesis. Los valores especificados aquí deben coincidir con las columnas especificadas después del nombre de la tabla, tanto en orden como en tipo (por ejemplo, si *nombreDeColumna1* se supone que debe ser la columna `nombrePila`, entonces *valor1* debe ser una cadena entre comillas sencillas que represente el nombre de pila). Siempre debemos enlistar explícitamente las columnas al insertar filas. Si el orden de las columnas cambia en la tabla, al utilizar solamente `VALUES` se puede provocar un error. La instrucción `INSERT`:

```
INSERT INTO autores ( nombrePila, apellidoPaterno )
   VALUES ( 'Alejandra', 'Villarreal' )
```

inserta una fila en la tabla `autores`. La instrucción indica que se proporcionan valores para las columnas `nombrePila` y `apellidoPaterno`. Los valores correspondientes son `'Alejandra'` y `'Villarreal'`. No especificamos un `idAutor` en este ejemplo, ya que `idAutor` es una columna autoincrementada en la tabla `autores`. Para cada fila que se agrega a esta tabla, MySQL asigna un valor de `idAutor` único que es el siguiente valor en la secuencia autoincrementada (por ejemplo: 1, 2, 3 y así sucesivamente). En este caso, Alejandra Villarreal recibiría el número 5 para `idAutor`. En la figura 25.20 se muestra la tabla `autores` después de la operación `INSERT`. [*Nota:* no todos los sistemas de administración de bases de datos soportan las columnas autoincrementadas. Consulte la documentación de su DBMS para encontrar alternativas a las columnas autoincrementadas].

idAutor	nombrePila	apellidoPaterno
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes
5	Alejandra	Villarreal

Figura 25.20 | Datos de ejemplo de la tabla `autores` después de una operación `INSERT`.

Error común de programación 25.6

Por lo general, es un error especificar un valor para una columna que se autoincrementa.

Error común de programación 25.7

Las instrucciones de SQL utilizan el carácter de comilla sencilla (') como delimitador para las cadenas. Para especificar una cadena que contenga una comilla sencilla (como O'Malley) en una instrucción de SQL, la cadena debe tener dos comillas sencillas en la posición en la que aparezca el carácter de comilla sencilla en la cadena (por ejemplo, `'O''Malley'`). El primero de los dos caracteres de comilla sencilla actúa como carácter de escape para el segundo. Si no se utiliza el carácter de escape en una cadena que sea parte de una instrucción de SQL, se produce un error de sintaxis de SQL.

25.4.6 La instrucción UPDATE

Una instrucción `UPDATE` modifica los datos en una tabla. La forma básica de la instrucción `UPDATE` es:

```
UPDATE nombreDeTabla
    SET nombreDeColumna1 = valor1, nombreDeColumna2 = valor2, ..., nombreDeColumnaN = valorN
    WHERE criterios
```

en donde *nombreDeTabla* es la tabla que se va a actualizar. El *nombreDeTabla* va seguido por la palabra clave `SET` y una lista separada por comas de los pares nombre/valor de las columnas, en el formato *nombreDeColumna=valor*. La cláusula `WHERE` opcional proporciona los criterios que determinan cuáles filas se van a actualizar. Aunque no es obligatoria, la cláusula `WHERE` se utiliza comúnmente, a menos que se vaya a realizar un cambio en todas las filas. La instrucción `UPDATE`:

```
UPDATE autores
    SET apellidoPaterno = 'Garcia'
    WHERE apellidoPaterno = 'Villarreal' AND nombrePila = 'Alejandra'
```

actualiza una fila en la tabla `autores`. La instrucción indica que `apellidoPaterno` recibirá el valor `Garcia` para la fila en la que `apellidoPaterno` sea igual a `Villarreal` y `nombrePila` sea igual a `Alejandra`. [*Nota:* si hay varias filas con el mismo nombre de pila "Alejandra" y el apellido paterno "Villarreal", esta instrucción modificará a todas esas filas para que tengan el apellido paterno "Garcia"]. Si conocemos el `idAutor` desde antes de realizar la operación `UPDATE` (tal vez porque lo hayamos buscado con anterioridad), la cláusula `WHERE` se puede simplificar de la siguiente manera:

```
WHERE idAutor = 5
```

En la figura 25.21 se muestra la tabla `autores` después de realizar la operación `UPDATE`.

idAutor	nombrePila	apellidoPaterno
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes
5	Alejandra	Garcia

Figura 25.21 | Datos de ejemplo de la tabla `autores` después de una operación `UPDATE`.

25.4.7 La instrucción DELETE

Una instrucción `DELETE` de SQL elimina filas de una tabla. La forma básica de una instrucción `DELETE` es:

```
DELETE FROM nombreDeTabla WHERE criterios
```

en donde *nombreDeTabla* es la tabla de la que se van a eliminar filas. La cláusula `WHERE` opcional especifica los criterios utilizados para determinar cuáles filas eliminar. Si se omite esta cláusula, se eliminan todas las filas de la tabla. La instrucción `DELETE`:

```
DELETE FROM autores
    WHERE apellidoPaterno = 'Garcia' AND nombrePila = 'Alejandra'
```

elimina la fila de Alejandra Garcia en la tabla `autores`. Si conocemos el `idAutor` desde antes de realizar la operación `DELETE`, la cláusula `WHERE` puede simplificarse de la siguiente manera:

```
WHERE idAutor = 5
```

En la figura 25.22 se muestra la tabla `autores` después de realizar la operación `DELETE`.

idAutor	nombrePila	apellidoPaterno
1	Harvey	Deitel
2	Paul	Deitel
3	Andrew	Goldberg
4	David	Choffnes

Figura 25.22 | Datos de ejemplo de la tabla `autores` después de una operación `DELETE`.

25.5 Instrucciones para instalar MySQL y MySQL Connector/J

MySQL 5.0 Community Edition es un sistema de administración de bases de datos de código fuente abierto que se ejecuta en muchas plataformas, incluyendo Windows, Solaris, Linux y Macintosh. En el sitio `www.mysql.com` encontrará toda la información acerca de MySQL. Los ejemplos en las secciones 25.8 y 25.9 manipulan bases de datos de MySQL.

Instalación de MySQL

Para instalar MySQL Community Edition:

1. Para aprender acerca de los requerimientos de instalación para su plataforma, visite el sitio `dev.mysql.com/doc/refman/5.0/en/general-installation-issues.html` (para ver esta información en español, visite el sitio `dev.mysql.com/doc/refman/5.0/es/general-installation-issues.html`).
2. Visite `dev.mysql.com/downloads/mysql/5.0.html` y descargue el instalador para su plataforma. Para los ejemplos de MySQL en este capítulo, sólo necesita el paquete Windows Essentials en Microsoft Windows, o el paquete Standard en la mayoría de las otras plataformas. [*Nota:* para estas instrucciones, vamos a suponer que está utilizando Microsoft Windows. En el sitio `dev.mysql.com/doc/refman/5.0/en/installing.html` (o `dev.mysql.com/doc/refman/5.0/es/installing.html` en español) encontrará las instrucciones completas de instalación para las otras plataformas].
3. Haga doble clic en el archivo `mysql-essential-5.0.27-win32.msi` para iniciar el instalador. [*Nota:* el nombre de este archivo puede diferir, dependiendo de la versión actual de MySQL 5.0].
4. Seleccione la opción **Typical** (**Típica**) en **Setup Type** (**Tipo de instalación**) y haga clic en **Next >** (**Siguiente**). Después haga clic en **Install** (**Instalar**).

Cuando termine la instalación, el programa le pedirá que configure una cuenta en MySQL.com. Si no desea hacerlo, seleccione **Skip Sign-up** (**Omitir registro**) y haga clic en **Next >** (**Siguiente**). Después de completar el proceso de registro o de omitirlo, puede configurar el servidor de MySQL. Haga clic en **Finish** (**Terminar**) para iniciar el **MySQL Server Instance Configuration Wizard** (**Asistente para la configuración de una instancia de MySQL Server**). Para configurar el servidor:

1. Haga clic en **Next >** (**Siguiente**); después seleccione **Standard Configuration** (**Configuración estándar**) y haga clic en **Next >** otra vez.
2. Tiene la opción de instalar MySQL como servicio Windows, lo cual permite al servidor de MySQL empezar a ejecutarse automáticamente cada vez que su sistema se inicie. Para nuestros ejemplos esto no es necesario, por lo que puede desactivar la opción **Install as a Windows Service** (**Instalar como servicio Windows**), y después haga clic en la opción **Include Bin Directory in Windows PATH** (**Incluir directorio Bin en la ruta PATH de Windows**). Esto le permitirá usar los comandos de MySQL en el Símbolo del sistema de Windows.
3. Haga clic en **Next >** y después en **Execute** (**Ejecutar**) para llevar a cabo la configuración del servidor.
4. Haga clic en **Finish** (**Terminar**) para cerrar el asistente.

Instalación de MySQL Connector/J

Para usar MySQL con JDBC, también necesita instalar **MySQL Connector/J** (la J representa a Java): un controlador de JDBC que permite a los programas usar JDBC para interactuar con MySQL. Puede descargar MySQL Connector/J de

```
dev.mysql.com/downloads/connector/j/5.0.html
```

La documentación para Connector/J se encuentra en `dev.mysql.com/doc/connector/j/en/connector-j.html`. Al momento de escribir este libro, la versión actual disponible en forma general de MySQL Connector/J es 5.0.4. Para instalar MySQL Connector/J:

1. Descargue el archivo `mysql-connector-java-5.0.4.zip`.
2. Abra `mysql-connector-java-5.0.4.zip` con un extractor de archivos, como WinZip (`www.winzip.com`). Extraiga su contenido en la unidad C:\. Esto creará un directorio llamado `mysql-connector-java-5.0.4`. La documentación para MySQL Connector/J está en el archivo `connector-j.pdf` en el subdirectorio `docs` de `mysql-connector-java-5.0.4`, o puede verla en línea, en el sitio `dev.mysql.com/doc/connector/j/en/connector-j.html`.

25.6 Instrucciones para establecer una cuenta de usuario de MySQL

Para que los ejemplos de MySQL se ejecuten correctamente, necesita configurar una cuenta de usuario que permita a los usuarios crear, eliminar y modificar una base de datos. Una vez instalado MySQL, siga los pasos que se muestran a continuación para configurar una cuenta de usuario (en estos pasos asumimos que MySQL está instalado en su directorio predeterminado):

1. Abra una ventana de Símbolo del sistema e inicie el servidor de bases de datos, ejecutando el comando `mysqld-nt.exe`. Observe que este comando no tiene salida; simplemente inicia el servidor MySQL. No cierre esta ventana, de lo contrario el servidor dejará de ejecutarse.
2. A continuación, inicie el monitor de MySQL para que pueda configurar una cuenta de usuario; abra otra ventana de Símbolo del sistema y ejecute el comando

   ```
   mysql -h localhost -u root
   ```

   La opción `-h` indica el host (computadora) en el que se está ejecutando el servidor MySQL; en este caso, es su equipo local (`localhost`). La opción `-u` indica la cuenta de usuario que se utilizará para iniciar sesión en el servidor; `root` es la cuenta de usuario predeterminada que se crea durante la instalación, para que usted pueda configurar el servidor. Una vez que inicie sesión, aparecerá un indicador `mysql>` en el que podrá escribir comandos para interactuar con el servidor MySQL.
3. En el indicador `mysql>`, escriba

   ```
   USE mysql;
   ```

   para seleccionar la base de datos incrustada, llamada `mysql`, la cual almacena información relacionada con el servidor, como las cuentas de usuario y sus privilegios para interactuar con el servidor. Observe que cada comando debe terminar con punto y coma. Para confirmar el comando, MySQL genera el mensaje "`Database changed.`" (La base de datos cambió).
4. A continuación, agregue la cuenta de usuario `jhtp7` a la base de datos incrustada `mysql`. Esta base de datos contiene una tabla llamada `user`, con columnas que representan el nombre del usuario, su contraseña y varios privilegios. Para crear la cuenta de usuario `jhtp7` con la contraseña `jhtp7`, ejecute los siguientes comandos desde el indicador `mysql>`:

   ```
   create user 'jhtp7'@'localhost' identified by 'jhtp7';

   grant select, insert, update, delete, create, drop, references,
       execute on *.* to 'jhtp7'@'localhost';
   ```

Esto crea el usuario `jhtp7` con los privilegios necesarios para crear las bases de datos utilizadas en este capítulo, y para manipular esas bases de datos. Por último,

5. Escriba el comando

   ```
   exit;
   ```

   para terminar el monitor MySQL.

25.7 Creación de la base de datos `libros` en MySQL

Para cada una de las bases de datos MySQL que veremos en este libro, proporcionamos una secuencia de comandos SQL en un archivo con la extensión `.sql` que configura la base de datos y sus tablas. Puede ejecutar estas secuencias de comandos en el monitor de MySQL. En el directorio de ejemplos de este capítulo, encontrará la secuencia de comandos SQL `libros.sql` para crear la base de datos `libros`. Para los siguientes pasos, vamos a suponer que el servidor MySQL (`mysqld-nt.exe`) sigue ejecutándose. Para ejecutar la secuencia de comandos `libros.sql`:

1. Abra una ventana de Símbolo del sistema y utilice el comando `cd` para cambiar al directorio en el que se encuentra la secuencia de comandos `libros.sql`.
2. Inicie el monitor de MySQL, escribiendo

   ```
   mysql -h localhost -u jhtp7 -p
   ```

   La opción `-p` hará que el monitor le pida la contraseña para el usuario `jhtp7`. Cuando ocurra esto, escriba la contraseña `jhtp7`.
3. Ejecute la secuencia de comandos, escribiendo

   ```
   source libros.sql;
   ```

   Esto creará un nuevo directorio llamado `libros` en el directorio `data` del servidor (en Windows, se encuentra en `C:\Archivos de programa\MySQL\MySQL Server 5.0\data` de manera predeterminada). Este nuevo directorio contiene la base de datos `libros`.
4. Escriba el comando

   ```
   exit;
   ```

   para terminar el monitor de MySQL. Ahora está listo para continuar con el primer ejemplo de JDBC.

25.8 Manipulación de bases de datos con JDBC

En esta sección presentaremos dos ejemplos. El primero le enseñará cómo conectarse a una base de datos y hacer consultas en ella. El segundo ejemplo le demostrará cómo mostrar en pantalla el resultado de la consulta.

25.8.1 Cómo conectarse y realizar consultas en una base de datos

El ejemplo de la figura 25.23 realiza una consulta simple en la base de datos `libros` para obtener toda la tabla `autores` y mostrar los datos. El programa muestra cómo conectarse a la base de datos, hacer una consulta en la misma y procesar el resultado. En la siguiente discusión presentaremos los aspectos clave del programa relacionados con JDBC. [*Nota:* en las secciones 25.5 a 25.7 se muestra cómo iniciar el servidor MySQL, configurar una cuenta de usuario y crear la base de datos `libros`. Estos pasos *deben* realizarse antes de ejecutar el programa de la figura 25.23].

```
1  // Fig. 25.23: MostrarAutores.java
2  // Muestra el contenido de la tabla autores.
3  import java.sql.Connection;
4  import java.sql.Statement;
```

Figura 25.23 | Cómo mostrar el contenido de la tabla `autores`. (Parte 1 de 3).

```
5  import java.sql.DriverManager;
6  import java.sql.ResultSet;
7  import java.sql.ResultSetMetaData;
8  import java.sql.SQLException;
9
10 public class MostrarAutores
11 {
12    // nombre del controlador de JDBC y URL de la base de datos
13    static final String CONTROLADOR = "com.mysql.jdbc.Driver";
14    static final String URL_BASEDATOS = "jdbc:mysql://localhost/libros";
15
16    // inicia la aplicación
17    public static void main( String args[] )
18    {
19       Connection conexion = null; // maneja la conexión
20       Statement instruccion = null; // instrucción de consulta
21       ResultSet conjuntoResultados = null; // maneja los resultados
22
23       // se conecta a la base de datos libros y realiza una consulta
24       try
25       {
26          // carga la clase controlador
27          Class.forName( CONTROLADOR );
28
29          // establece la conexión a la base de datos
30          conexion =
31             DriverManager.getConnection( URL_BASEDATOS, "jhtp7", "jhtp7" );
32
33          // crea objeto Statement para consultar la base de datos
34          instruccion = conexion.createStatement();
35
36          // consulta la base de datos
37          conjuntoResultados = instruccion.executeQuery(
38             "SELECT IDAutor, nombrePila, apellidoPaterno FROM autores" );
39
40          // procesa los resultados de la consulta
41          ResultSetMetaData metaDatos = conjuntoResultados.getMetaData();
42          int numeroDeColumnas = metaDatos.getColumnCount();
43          System.out.println( "Tabla Autores de la base de datos Libros:\n" );
44
45          for ( int i = 1; i <= numeroDeColumnas; i++ )
46             System.out.printf( "%-8s\t", metaDatos.getColumnName( i ) );
47          System.out.println();
48
49          while ( conjuntoResultados.next() )
50          {
51             for ( int i = 1; i <= numeroDeColumnas; i++ )
52                System.out.printf( "%-8s\t", conjuntoResultados.getObject( i ) );
53             System.out.println();
54          } // fin de while
55       }  // fin de try
56       catch ( SQLException excepcionSql )
57       {
58          excepcionSql.printStackTrace();
59       } // fin de catch
60       catch ( ClassNotFoundException noEncontroClase )
61       {
62          noEncontroClase.printStackTrace();
63       } // fin de catch
```

Figura 25.23 | Cómo mostrar el contenido de la tabla `autores`. (Parte 2 de 3).

```
64         finally // asegura que conjuntoResultados, instruccion y conexion estén cerrados
65         {
66            try
67            {
68               conjuntoResultados.close();
69               instruccion.close();
70               conexion.close();
71            } // fin de try
72            catch ( Exception excepcion )
73            {
74               excepcion.printStackTrace();
75            } // fin de catch
76         } // fin de finally
77      } // fin de main
78   } // fin de la clase MostrarAutores
```

```
Tabla Autores de la base de datos Libros:

IDAutor        nombrePila        apellidoPaterno
1              Harvey            Deitel
2              Paul              Deitel
3              Andrew            Goldberg
4              David             Choffnes
```

Figura 25.23 | Cómo mostrar el contenido de la tabla `autores`. (Parte 3 de 3).

En las líneas 3 a 8 se importan las interfaces de JDBC y las clases del paquete `java.sql` que se utilizan en este programa. En la línea 13 se declara una constante de cadena para el controlador de la base de datos, y en la línea 14 se declara una constante de cadena para el URL de la base de datos. Estas constantes identifican el nombre de la base de datos a la que nos conectaremos, así como información acerca del protocolo utilizado por el controlador JDBC (que veremos en breve). El método `main` (líneas 17 a 77) se conecta a la base de datos `libros`, realiza una consulta en la base de datos, muestra el resultado de esa consulta y cierra la conexión a la base de datos.

El programa debe cargar el controlador de bases de datos para poder conectarse a la base de datos. En la línea 27 se utiliza el método `static` `forName` de la clase `Class` para cargar la clase para el controlador de bases de datos. Esta línea lanza una excepción verificada del tipo `java.lang.ClassNotFoundException` si el cargador de clases no puede localizar la clase del controlador. Para evitar esta excepción, necesitamos incluir el archivo `mysql-connector-java-5.0.4-bin.jar` (en el directorio `C:\mysql-connector-java-5.0.4`) en la ruta de clases del programa a la hora de ejecutarlo, como se muestra a continuación:

```
java -classpath
  .;c:\mysql-connector-java-5.0.4\mysql-connector-java-5.0.4-bin.jar
  MostrarAutores
```

En la ruta de clases del comando anterior, observe el punto (`.`) al principio de la información de la ruta de clases. Si se omite este punto, la JVM no buscará las clases en el directorio actual, y por ende, no encontrará el archivo de la clase `MostrarAutores`. También puede copiar el archivo `mysql-connector-java-5.0.4-bin.jar` al directorio `C:\Archivos de programa\Java\jdk1.6.0\jre\lib\ext`. Después de hacer esto, puede ejecutar la aplicación simplemente utilizando el comando `java MostrarAutores`.

Observación de ingeniería de software 25.4

La mayoría de los distribuidores de bases de datos proporcionan sus propios controladores de bases de datos JDBC, y muchos distribuidores independientes proporcionan también controladores JDBC. Para obtener más información acerca de los controladores de JDBC, visite el sitio Web sobre JDBC de Sun Microsystems en `servlet.java.sun.com/products/jdbc/drivers`.

En las líneas 30 y 31 de la figura 25.23 se crea un objeto `Connection` (paquete `java.sql`), el cual es referenciado mediante `conexion`. Un objeto que implementa a la interfaz `Connection` administra la conexión entre el programa de Java y la base de datos. Los objetos `Connection` permiten a los programas crear instrucciones de SQL para manipular bases de datos. El programa inicializa a `conexion` con el resultado de una llamada al método `static` `getConnection` de la clase `DriverManager` (paquete `java.sql`), el cual trata de conectarse a la base de datos especificada mediante su URL. El método `getConnection` recibe tres argumentos: un objeto `String` que especifica el URL de la base de datos, un objeto `String` que especifica el nombre de usuario y un objeto `String` que especifica la contraseña. El nombre de usuario y la contraseña se establecen en la sección 25.6. Si utilizó un nombre de usuario y contraseña distintos, necesita reemplazar el nombre de usuario (segundo argumento) y la contraseña (tercer argumento) que se pasan al método `getConnection` en la línea 31. El URL localiza la base de datos (posiblemente en una red o en el sistema de archivos local de la computadora). El URL `jdbc:mysql://localhost/libros` especifica el protocolo para la comunicación (`jdbc`), el **subprotocolo** para la comunicación (`mysql`) y la ubicación de la base de datos (`//localhost/libros`, en donde `localhost` es el host que ejecuta el servidor MySQL y `libros` es el nombre de la base de datos). El subprotocolo `mysql` indica que el programa utiliza un subprotocolo específico de MySQL para conectarse a la base de datos MySQL. Si el objeto `DriverManager` no se puede conectar a la base de datos, el método `getConnection` lanza una excepción `SQLException` (paquete `java.sql`). En la figura 25.24 se enlistan los nombres de los controladores de JDBC y los formatos de URL de base de datos de varios RDBMSs populares.

Observación de ingeniería de software 25.5

La mayoría de los sistemas de administración de bases de datos requieren que el usuario inicie sesión para poder administrar el contenido de la base de datos. El método `getConnection` *de* `DriverManager` *está sobrecargado con versiones que permiten al programa proporcionar el nombre de usuario y la contraseña para obtener acceso.*

En la línea 34 se invoca el método `createStatement` de `Connection` para obtener un objeto que implemente a la interfaz `Statement` (paquete `java.sql`). El programa utiliza al objeto `Statement` para enviar instrucciones de SQL a la base de datos.

En las líneas 37 y 38 se utiliza el método `executeQuery` del objeto `Statement` para enviar una consulta que seleccione toda la información sobre los autores en la tabla `autores`. Este método devuelve un objeto que implementa a la interfaz `ResultSet`, y contiene el resultado de esta consulta. Los métodos de `ResultSet` permiten al programa manipular el resultado de la consulta.

En las líneas 41 a 54 se procesa el objeto `ResultSet`. En la línea 41 se obtienen los metadatos para el objeto `ResultSet` en forma de un objeto `ResultSetMetaData` (paquete `java.sql`). Los **metadatos** describen el contenido del objeto `ResultSet`. Los programas pueden usar metadatos mediante la programación, para obtener información acerca de los nombres y tipos de las columnas del objeto `ResultSet`. En la línea 42 se utiliza el método `ResultSetMetaData` de `getColumnCount` para obtener el número de columnas para el objeto `ResultSet`. En las líneas 45 y 46 se anexan los nombres de las columnas.

RDBMS	Formato de URL de base de datos
MySQL	`jdbc:mysql://`*nombrehost:numeroPuerto/nombreBaseDatos*
ORACLE	`jdbc:oracle:thin:@`*nombrehost:numeroPuerto:nombreBaseDatos*
DB2	`jdbc:db2:`*nombrehost:numeroPuerto/nombreBaseDatos*
Java DB/Apache Derby	`jdbc:derby:`*nombreBaseDatos* (incrustado) `jdbc:derby://`*nombrehost:numeroPuerto/nombreBaseDatos* (red)
Microsoft SQL Server	`jdbc:sqlserver://`*nombrehost.numeroPuerto*;nombreBaseDatos=*nombreBaseDatos*
Sybase	`jdbc:sybase:`Tds:*nombrehost:numeroPuerto/nombreBaseDatos*

Figura 25.24 | Formatos populares de URL de base de datos.

Observación de ingeniería de software 25.6

Los metadatos permiten a los programas procesar el contenido de un objeto `ResultSet` *en forma dinámica, cuando no se conoce de antemano la información detallada acerca del objeto* `ResultSet`.

En las líneas 49 a 54 se muestran los datos en cada fila del objeto `ResultSet`. Primero, el programa posiciona el cursor de `ResultSet` (que apunta a la fila que se está procesando) en la primera fila en el objeto `ResultSet`, mediante el método `next` (línea 49). Este método devuelve el valor `boolean` `true` si el cursor puede posicionarse en la siguiente fila; en caso contrario el método devuelve `false`.

Error común de programación 25.8

Inicialmente, un cursor `ResultSet` *se posiciona antes de la primera fila. Si trata de acceder al contenido de un objeto* `ResultSet` *antes de posicionar el cursor* `ResultSet` *en la primera fila con el método* `next`*, se produce una excepción* `SQLException`.

Si hay filas en el objeto `ResultSet`, en la línea 52 se extrae el contenido de una columna en la fila actual. Al procesar un objeto `ResultSet`, es posible extraer cada columna de este objeto como un tipo de Java específico. De hecho, el método `getColumnType` de `ResultSetMetaData` devuelve un valor entero constante de la clase `Types` (paquete `java.sql`), indicando el tipo de una columna especificada. Los programas pueden utilizar estos valores en una estructura `switch` para invocar métodos de `ResultSet` que devuelvan los valores de las columnas como tipos de Java apropiados. Si el tipo de una columna es `Types.INTEGER`, el método `getInt` de `ResultSet` devuelve el valor de la columna como un `int`. Los métodos *obtener (get)* de `ResultSet` generalmente reciben como argumento un número de columna (como un valor `int`) o un nombre de columna (como un valor `String`), indicando cuál valor de la columna se va a obtener. Visite la página

```
java.sun.com/javase/6/docs/technotes/guides/jdbc/getstart/
   GettingStartedTOC.fm.html
```

para obtener las asociaciones detalladas de los tipos de datos SQL y los tipos de Java, y para determinar el método de `ResultSet` apropiado a llamar a cada tipo de datos SQL.

Tip de rendimiento 25.1

Si una consulta especifica las columnas exactas a seleccionar de la base de datos, entonces el objeto `ResultSet` *contendrá las columnas en el orden especificado. En este caso, es más eficiente utilizar el número de columna para obtener su valor que utilizar su nombre. El número de columna proporciona un acceso directo a la columna especificada. Si se usa el nombre, se requiere una búsqueda lineal de los nombres de columna para localizar la apropiada.*

Por cuestiones de simpleza, en este ejemplo trataremos a cada valor como un objeto `Object`. El programa obtiene el valor de cada columna con el método `getObject` de `ResultSet` (línea 52) e imprime la representación `String` del objeto `Object`. Observe que, a diferencia de los índices de arreglos que empiezan en 0, los números de columna de `ResultSet` empiezan en 1. El bloque `finally` (líneas 64 a 76) cierra los objetos `ResultSet` (línea 68), `Statement` (línea 69) y el objeto `Connection` (línea 70) de la base de datos. [*Nota:* en las líneas 68 a 70 se lanzarán excepciones `NullPointerException` si los objetos `ResultSet`, `Statement` o `Connection` no se crearon en forma apropiada. En código de producción, debemos comprobar las variables que se refieren a estos objetos, para ver si son `null` antes de llamar a `close`].

Error común de programación 25.9

Si se especifica el número de columna 0 cuando se van a obtener valores de un objeto `ResultSet`*, se produce una excepción* `SQLException`.

Error común de programación 25.10

Al tratar de manipular un objeto `ResultSet` *después de cerrar el objeto* `Statement` *que lo creó, se produce una excepción* `SQLException`. *El programa descarta el objeto* `ResultSet` *cuando se cierra el objeto* `Statement` *correspondiente.*

Observación de ingeniería de software 25.7

Cada objeto `Statement` puede abrir solamente un objeto `ResultSet` en un momento dado. Cuando un objeto `Statement` devuelve un nuevo objeto `ResultSet`, el objeto `Statement` cierra el objeto `ResultSet` anterior. Para utilizar varios objetos `ResultSet` en paralelo, se deben usar objetos `Statement` separados para devolver los objetos `ResultSet`.

25.8.2 Consultas en la base de datos `libros`

El siguiente ejemplo (figuras 25.25 y 25.28) permite al usuario introducir cualquier consulta en el programa. Este ejemplo muestra el resultado de una consulta en un objeto `JTable`, utilizando un objeto `TableModel` para proporcionar los datos del objeto `ResultSet` al objeto `JTable`. Un objeto `JTable` es un componente de la GUI de Swing que puede enlazarse a una base de datos para mostrar los resultados de una consulta. La clase `ResultSetTableModel` (figura 25.25) realiza la conexión a la base de datos por medio de un objeto `TableModel` y mantiene el objeto `ResultSet`. La clase `MostrarResultadosConsulta` (figura 25.28) crea la GUI y especifica una instancia de la clase `ResultSetTableModel` para proporcionar datos para el objeto `JTable`.

La clase `ResultSetTableModel`

La clase `ResultSetTableModel` (figura 25.25) extiende a la clase `AbstractTableModel` (paquete `javax.swing.table`), la cual implementa a la interfaz `TableModel`. La clase `ResultSetTableModel` sobrescribe a los métodos `getColumnClass`, `getColumnCount`, `getColumnName`, `getRowCount` y `getValueAt` de `TableModel`. Las implementaciones predeterminadas de los métodos `isCellEditable` y `setValueAt` de `TableModel` (proporcionados por `AbstractTableModel`) no se sobrescriben, ya que este ejemplo no soporta la capacidad de editar las celdas del objeto `JTable`. Tampoco se sobrescriben las implementaciones predeterminadas de los métodos `addTableModelListener` y `removeTableModelListener` de `TableModel` (proporcionados por `AbstractTableModel`), ya que las implementaciones de estos métodos de `AbstractTableModel` agregan y eliminan apropiadamente los componentes de escucha de eventos.

```
1  // Fig. 25.25: ResultSetTableModel.java
2  // Un objeto TableModel que suministra datos ResultSet a un objeto JTable.
3  import java.sql.Connection;
4  import java.sql.Statement;
5  import java.sql.DriverManager;
6  import java.sql.ResultSet;
7  import java.sql.ResultSetMetaData;
8  import java.sql.SQLException;
9  import javax.swing.table.AbstractTableModel;
10
11 // las filas y columnas del objeto ResultSet se cuentan desde 1 y
12 // las filas y columnas del objeto JTable se cuentan desde 0. Al procesar
13 // filas o columnas de ResultSet para usarlas en un objeto JTable, es
14 // necesario sumar 1 al número de fila o columna para manipular
15 // la columna apropiada del objeto ResultSet (es decir, la columna 0 de JTable
16 // es la columna 1 de ResultSet y la fila 0 de JTable es la fila 1 de ResultSet).
17 public class ResultSetTableModel extends AbstractTableModel
18 {
19    private Connection conexion;
20    private Statement instruccion;
21    private ResultSet conjuntoResultados;
22    private ResultSetMetaData metaDatos;
23    private int numeroDeFilas;
24
25    // lleva la cuenta del estado de la conexión a la base de datos
26    private boolean conectadoABaseDatos = false;
```

Figura 25.25 | Un objeto `TableModel` que suministra datos `ResultSet` a un objeto `JTable`. (Parte 1 de 4).

```
27
28      // el constructor inicializa conjuntoResultados y obtiene su objeto de metadatos;
29      // determina el número de filas
30      public ResultSetTableModel( String controlador,String url, String nombreusuario,
31           String contrasenia, String consulta )
32         throws SQLException, ClassNotFoundException
33      {
34         Class.forName( controlador );
35         // se conecta a la base de datos
36         conexion = DriverManager.getConnection( url, nombreusuario, contrasenia );
37
38         // crea objeto Statement para consultar la base de datos
39         instruccion = conexion.createStatement(
40            ResultSet.TYPE_SCROLL_INSENSITIVE,
41            ResultSet.CONCUR_READ_ONLY );
42
43         // actualiza el estado de la conexión a la base de datos
44         conectadoABaseDatos = true;
45
46         // establece consulta y la ejecuta
47         establecerConsulta( consulta );
48      } // fin del constructor ResultSetTableModel
49
50      // obtiene la clase que representa el tipo de la columna
51      public Class getColumnClass( int columna ) throws IllegalStateException
52      {
53         // verifica que esté disponible la conexión a la base de datos
54         if ( !conectadoABaseDatos )
55            throw new IllegalStateException( "No hay conexion a la base de datos" );
56
57         // determina la clase de Java de la columna
58         try
59         {
60            String nombreClase = metaDatos.getColumnClassName( columna + 1 );
61
62            // devuelve objeto Class que representa a nombreClase
63            return Class.forName( nombreClase );
64         } // fin de try
65         catch ( Exception excepcion )
66         {
67            excepcion.printStackTrace();
68         } // fin de catch
69
70         return Object.class; // si ocurren problemas en el código anterior, asume el tipo
           Object
71      } // fin del método getColumnClass
72
73      // obtiene el número de columnas en el objeto ResultSet
74      public int getColumnCount() throws IllegalStateException
75      {
76         // verifica que esté disponible la conexión a la base de datos
77         if ( !conectadoABaseDatos )
78            throw new IllegalStateException( "No hay conexión a la base de datos" );
79
80         // determina el número de columnas
81         try
82         {
83            return metaDatos.getColumnCount();
```

Figura 25.25 | Un objeto `TableModel` que suministra datos `ResultSet` a un objeto `JTable`. (Parte 2 de 4).

```
84          } // fin de try
85          catch ( SQLException excepcionSql )
86          {
87             excepcionSql.printStackTrace();
88          } // fin de catch
89
90          return 0; // si ocurren problemas en el código anterior, devuelve 0 para el
            número de columnas
91       } // fin del método getColumnCount
92
93       // obtiene el nombre de una columna específica en el objeto ResultSet
94       public String getColumnName( int columna ) throws IllegalStateException
95       {
96          // verifica que esté disponible la conexión a la base de datos
97          if ( !conectadoABaseDatos )
98             throw new IllegalStateException( "No hay conexion a la base de datos" );
99
100         // determina el nombre de la columna
101         try
102         {
103            return metaDatos.getColumnName( columna + 1 );
104         } // fin de try
105         catch ( SQLException excepcionSql )
106         {
107            excepcionSql.printStackTrace();
108         } // fin de catch
109
110         return ""; // si hay problemas, devuelve la cadena vacía para el nombre de la
            columna
111      } // fin del método getColumnName
112
113      // devuelve el número de filas en el objeto ResultSet
114      public int getRowCount() throws IllegalStateException
115      {
116         // verifica que esté disponible la conexión a la base de datos
117         if ( !conectadoABaseDatos )
118            throw new IllegalStateException( "No hay conexion a la base de datos" );
119
120         return numeroDeFilas;
121      } // fin del método getRowCount
122
123      // obtiene el valor en la fila y columna específicas
124      public Object getValueAt( int fila, int columna )
125         throws IllegalStateException
126      {
127         // verifica que esté disponible la conexión a la base de datos
128         if ( !conectadoABaseDatos )
129            throw new IllegalStateException( "No hay conexion a la base de datos" );
130
131         // obtiene un valor en una fila y columna especificadas del objeto ResultSet
132         try
133         {
134            conjuntoResultados.absolute( fila + 1 );
135            return conjuntoResultados.getObject( columna + 1 );
136         } // fin de try
137         catch ( SQLException excepcionSql )
138         {
139            excepcionSql.printStackTrace();
```

Figura 25.25 | Un objeto `TableModel` que suministra datos `ResultSet` a un objeto `JTable`. (Parte 3 de 4).

```
140         } // fin de catch
141
142         return ""; // si hay problemas, devuelve el objeto cadena vacía
143      } // fin del método getValueAt
144
145      // establece nueva cadena de consulta en la base de datos
146      public void establecerConsulta( String consulta )
147         throws SQLException, IllegalStateException
148      {
149         // verifica que esté disponible la conexión a la base de datos
150         if ( !conectadoABaseDatos )
151            throw new IllegalStateException( "No hay conexion a la base de datos" );
152
153         // especifica la consulta y la ejecuta
154         conjuntoResultados = instruccion.executeQuery( consulta );
155
156         // obtiene metadatos para el objeto ResultSet
157         metaDatos = conjuntoResultados.getMetaData();
158
159         // determina el número de filas en el objeto ResultSet
160         conjuntoResultados.last(); // avanza a la última fila
161         numeroDeFilas = conjuntoResultados.getRow(); // obtiene el número de fila
162
163         // notifica al objeto JTable que el modelo ha cambiado
164         fireTableStructureChanged();
165      } // fin del método establecerConsulta
166
167      // cierra objetos Statement y Connection
168      public void desconectarDeBaseDatos()
169      {
170         if ( conectadoABaseDatos )
171         {
172            // cierra objetos Statement y Connection
173            try
174            {
175               conjuntoResultados.close();
176               instruccion.close();
177               conexion.close();
178            } // fin de try
179            catch ( SQLException excepcionSql )
180            {
181               excepcionSql.printStackTrace();
182            } // fin de catch
183            finally  // actualiza el estado de la conexión a la base de datos
184            {
185               conectadoABaseDatos = false;
186            } // fin de finally
187         } // fin de if
188      } // fin del método desconectarDeBaseDatos
189   }  // fin de la clase ResultSetTableModel
```

Figura 25.25 | Un objeto `TableModel` que suministra datos `ResultSet` a un objeto `JTable`. (Parte 4 de 4).

El constructor de `ResultSetTableModel` (líneas 30 a 48) acepta cinco argumentos `String`: el nombre del controlador de MySQL, el URL de la base de datos, el nombre de usuario, la contraseña y la consulta predeterminada a realizar. El constructor lanza cualquier excepción que ocurra en su cuerpo, de vuelta a la aplicación que creó el objeto `ResultSetTableModel`, para que la aplicación pueda determinar cómo manejar esa excepción (por ejemplo, reportar un error y terminar la aplicación). En la línea 34 se carga el controlador. En la línea 36 se esta-

blece una conexión a la base de datos. En las líneas 39 a 41 se invoca el método `createStatement` de `Connection` para obtener un objeto `Statement`. En este ejemplo utilizamos una versión del método `createStatement` que recibe dos argumentos: el tipo de conjunto de resultados y la concurrencia del conjunto de resultados. El **tipo de conjunto de resultados** (figura 25.26) especifica si el cursor del objeto `ResultSet` puede desplazarse en ambas direcciones o solamente hacia delante, y si el objeto `ResultSet` es susceptible a los cambios. Los objetos `ResultSet` que son susceptibles a los cambios los reflejan inmediatamente después de que éstos se realizan con los métodos de la interfaz `ResultSet`. Si un objeto `ResultSet` no es susceptible a los cambios, la consulta que produjo ese objeto `ResultSet` debe ejecutarse de nuevo para reflejar cualquier cambio realizado. La **concurrencia del conjunto de resultados** (figura 25.27) especifica si el objeto `ResultSet` puede actualizarse con los métodos de actualización de `ResultSet`. En este ejemplo utilizamos un objeto `ResultSet` que puede desplazarse, que no es susceptible a los cambios y es de sólo lectura. En la línea 47 se invoca nuestro método `establecerConsulta` de `ResultSetTableModel` (líneas 146 a 165) para ejecutar la consulta predeterminada.

Constante de tipo `static` de `ResultSet`	Descripción
`TYPE_FORWARD_ONLY`	Especifica que el cursor de un objeto `ResultSet` puede desplazarse solamente en dirección hacia delante (es decir, desde la primera hasta la última fila en el objeto `ResultSet`).
`TYPE_SCROLL_INSENSITIVE`	Especifica que el cursor de un objeto `ResultSet` puede desplazarse en cualquier dirección y que los cambios realizados al objeto `ResultSet` durante su procesamiento no se reflejarán en este objeto, a menos que el programa consulte la base de datos otra vez.
`TYPE_SCROLL_SENSITIVE`	Especifica que el cursor de un objeto `ResultSet` puede desplazarse en cualquier dirección y que los cambios realizados al objeto `ResultSet` durante su procesamiento se reflejarán inmediatamente en este objeto.

Figura 25.26 | Constantes de `ResultSet` para especificar el tipo del objeto `ResultSet`.

Constante de concurrencia `static` de `ResultSet`	Descripción
`CONCUR_READ_ONLY`	Especifica que un objeto `ResultSet` no puede actualizarse (es decir, los cambios en el contenido del objeto `ResultSet` no pueden reflejarse en la base de datos con los métodos *update* de `ResultSet`).
`CONCUR_UPDATABLE`	Especifica que un objeto `ResultSet` puede actualizarse (es decir, los cambios en el contenido del objeto `ResultSet` pueden reflejarse en la base de datos con los métodos *update* de `ResultSet`).

Figura 25.27 | Constantes de `ResultSet` para especificar las propiedades de los resultados.

Tip de portabilidad 25.3

Algunos controladores JDBC no soportan objetos `ResultSet` desplazables. En dichos casos, generalmente el controlador devuelve un objeto `ResultSet` en el que el cursor puede moverse sólo hacia adelante. Para obtener más información, consulte la documentación de su controlador de bases de datos.

Tip de portabilidad 25.4

Algunos controladores JDBC no soportan objetos `ResultSet` actualizables. En dichos casos, generalmente el controlador devuelve un objeto `ResultSet` de sólo lectura. Para obtener más información, consulte la documentación de su controlador de bases de datos.

Error común de programación 25.11

Al intentar actualizar un objeto `ResultSet` *cuando el controlador de base de datos no soporta objetos* `ResultSet` *actualizables se producen excepciones* `SQLException`.

Error común de programación 25.12

Al intentar mover el cursor hacia atrás mediante un objeto `ResultSet`, *cuando el controlador de base de datos no soporta el desplazamiento hacia atrás, se produce una excepción* `SQLException`.

El método `getColumnClass` (líneas 51 a 71) devuelve un objeto `Class` que representa a la superclase de todos los objetos en una columna específica. El objeto `JTable` utiliza esta información para configurar el desplegador de celdas y el editor de celdas predeterminados para esa columna en el objeto `JTable`. En la línea 60 se utiliza el método `getColumnClassName` de `ResultSetMetaData` para obtener el nombre de clase completamente calificado para la columna especificada. En la línea 63 se carga la clase y se devuelve el objeto `Class` correspondiente. Si ocurre una excepción, el bloque `catch` en las líneas 65 a 68 imprime un rastreo de la pila y en la línea 70 se devuelve `Object.class` (la instancia de `Class` que representa a la clase `Object`) como el tipo predeterminado. [*Nota:* en la línea 60 se utiliza el argumento `columna + 1`. Al igual que los arreglos, los números de fila y columna del objeto `JTable` se cuentan desde 0. Sin embargo, los números de fila y columna del objeto `ResultSet` se cuentan desde 1. Por lo tanto, al procesar filas o columnas de un objeto `ResultSet` para utilizarlas en un objeto `JTable`, es necesario sumar 1 al número de fila o de columna para manipular la fila o columna apropiada del objeto `ResultSet`].

El método `getColumnCount` (líneas 74 a 91) devuelve el número de columnas en el objeto `ResultSet` subyacente del modelo. En la línea 83 se utiliza el método `getColumnCount` de `ResultSetMetaData` para obtener el número de columnas en el objeto `ResultSet`. Si ocurre una excepción, el bloque `catch` en las líneas 85 a 88 imprime un rastreo de la pila y en la línea 93 se devuelve 0 como el número predeterminado de columnas.

El método `getColumnName` (líneas 94 a 111) devuelve el nombre de la columna en el objeto `ResultSet` subyacente del modelo. En la línea 103 se utiliza el método `getColumnName` de `ResultSetMetaData` para obtener el nombre de la columna del objeto `ResultSet`. Si ocurre una excepción, el bloque `catch` en las líneas 105 a 108 imprime un rastreo de la pila y en la línea 110 se devuelve la cadena vacía como el nombre predeterminado de la columna.

El método `getRowCount` (líneas 114 a 121) devuelve el número de filas en el objeto `ResultSet` subyacente del modelo. Cuando el método `establecerConsulta` (líneas 146 a 165) realiza una consulta, almacena el número de filas en la variable `numeroDeFilas`.

El método `getValueAt` (líneas 124 a 143) devuelve el objeto `Object` en una fila y columna específicas del objeto `ResultSet` subyacente del modelo. En la línea 134 se utiliza el método `absolute` de `ResultSet` para posicionar el cursor del objeto `ResultSet` en una fila específica. En la línea 135 se utiliza el método `getObject` de `ResultSet` para obtener el objeto `Object` en una columna específica de la fila actual. Si ocurre una excepción, el bloque `catch` en las líneas 137 a 140 imprime un rastreo de la pila y en la línea 142 se devuelve una cadena vacía como el valor predeterminado.

El método `establecerConsulta` (líneas 146 a 165) ejecuta la consulta que recibe como argumento para obtener un nuevo objeto `ResultSet` (línea 154). En la línea 157 se obtiene el objeto `ResultSetMetaData` para el nuevo objeto `ResultSet`. En la línea 160 se utiliza el método `last` de `ResultSet` para posicionar el cursor de `ResultSet` en la última fila del objeto `ResultSet`. [*Nota:* esto puede ser lento si la tabla contiene muchas filas]. En la línea 161 se utiliza el método `getRow` de `ResultSet` para obtener el número de fila de la fila actual en el objeto `ResultSet`. En la línea 164 se invoca el método `fireTableStructureChanged` (heredado de la clase `AbstractTableModel`) para notificar a cualquier objeto `JTable` que utilice a este objeto `ResultSetTableModel` como su modelo, que la estructura del modelo ha cambiado. Esto hace que el objeto `JTable` vuelva a llenar sus filas y columnas con los datos del nuevo objeto `ResultSet`. El método `establecerConsulta` lanza cualquier excepción que ocurra en su cuerpo, de vuelta a la aplicación que invocó a `establecerConsulta`.

El método `desconectarDeBaseDatos` (líneas 168 a 188) implementa un método de terminación apropiado para la clase `ResultSetTableModel`. Un diseñador de clases debe proporcionar un método `public` que los clientes de la clase deban invocar en forma explícita para liberar los recursos que haya utilizado un objeto. En este caso, el método `desconectarDeBaseDatos` cierra los objetos `ResultSet`, `Statement` y `Connection` (líneas 175 a 177),

los cuales se consideran recursos limitados. Los clientes de la clase `ResultSetTableModel` deben siempre invocar a este método cuando la instancia de esta clase ya no se necesite. Antes de liberar los recursos, en la línea 170 se verifica si la conexión ya está terminada. De no ser así, el método continúa. Observe que cada uno de los otros métodos en la clase lanzan una excepción `IllegalStateException` si el campo booleano `conectadoABaseDatos` es `false`. El método `desconectarDeBaseDatos` establece a `conectadoABaseDatos` en `false` (línea 185) para asegurarse que los clientes no utilicen una instancia de `ResultSetTableModel` después de que ésta haya sido eliminada. `IllegalStateExcepcion` es una excepción de las bibliotecas de Java que es apropiada para indicar esta condición de error.

La clase `MostrarResultadosConsulta`

La clase `MostrarResultadosConsulta` (figura 25.28) implementa la GUI de la aplicación e interactúa con el objeto `ResultSetTableModel` a través de un objeto `JTable`. Esta aplicación también demuestra las nuevas herramientas de ordenamiento y filtrado de `JTable`, que se introdujeron en Java SE 6.

En las líneas 27 a 30 y 33 se declaran el controlador de la base de datos, el URL, el nombre de usuario, la contraseña y la consulta predeterminada que se pasan al constructor de `ResultSetTableModel` para realizar la conexión inicial a la base de datos y ejecutar la consulta predeterminada. El constructor de `MostrarResultadosConsulta` (líneas 39 a 198) crea un objeto `ResultSetTableModel` y la GUI para la aplicación. En la línea 69 se crea el objeto `JTable` y se pasa un objeto `ResultSetTableModel` al constructor de `JTable`, el cual a su vez registra el objeto `JTable` como un componente de escucha para los eventos `TableModelEvent` que sean generados por el objeto `ResultSetTableModel`.

```
1  // Fig. 25.28: MostrarResultadosConsulta.java
2  // Muestra el contenido de la tabla Autores en la base de datos libros.
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.WindowAdapter;
7  import java.awt.event.WindowEvent;
8  import java.sql.SQLException;
9  import java.util.regex.PatternSyntaxException;
10 import javax.swing.JFrame;
11 import javax.swing.JTextArea;
12 import javax.swing.JScrollPane;
13 import javax.swing.ScrollPaneConstants;
14 import javax.swing.JTable;
15 import javax.swing.JOptionPane;
16 import javax.swing.JButton;
17 import javax.swing.Box;
18 import javax.swing.JLabel;
19 import javax.swing.JTextField;
20 import javax.swing.RowFilter;
21 import javax.swing.table.TableRowSorter;
22 import javax.swing.table.TableModel;
23
24 public class MostrarResultadosConsulta extends JFrame
25 {
26    // URL de la base de datos, nombre de usuario y contraseña para JDBC
27    static final String CONTROLADOR = "com.mysql.jdbc.Driver";
28    static final String URL_BASEDATOS = "jdbc:mysql://localhost/libros";
29    static final String NOMBREUSUARIO = "jhtp7";
30    static final String CONTRASENIA = "jhtp7";
31
32    // la consulta predeterminada obtiene todos los datos de la tabla autores
33    static final String CONSULTA_PREDETERMINADA = "SELECT * FROM autores";
34
```

Figura 25.28 | Visualización del contenido de la base de datos `libros`. (Parte 1 de 5).

```
35     private ResultSetTableModel modeloTabla;
36     private JTextArea areaConsulta;
37
38     // crea objeto ResultSetTableModel y GUI
39     public MostrarResultadosConsulta()
40     {
41        super( "Visualizacion de los resultados de la consulta" );
42
43        // crea objeto ResultSetTableModel y muestra la tabla de la base de datos
44        try
45        {
46           // crea objeto TableModel para los resultados de la consulta SELECT * FROM
          autores
47           modeloTabla = new ResultSetTableModel( CONTROLADOR, URL_BASEDATOS,
48              NOMBREUSUARIO, CONTRASENIA, CONSULTA_PREDETERMINADA );
49
50           // establece objeto JTextArea en el que el usuario escribe las consultas
51           areaConsulta = new JTextArea( CONSULTA_PREDETERMINADA, 3, 100 );
52           areaConsulta.setWrapStyleWord( true );
53           areaConsulta.setLineWrap( true );
54
55           JScrollPane scrollPane = new JScrollPane( areaConsulta,
56              ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
57              ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );
58
59           // establece objeto JButton para enviar las consultas
60           JButton botonEnviar = new JButton( "Enviar consulta" );
61
62           // crea objeto Box para manejar la colocación de areaConsulta y
63           // botonEnviar en la GUI
64           Box boxNorte = Box.createHorizontalBox();
65           boxNorte.add( scrollPane );
66           boxNorte.add( botonEnviar );
67
68           // crea delegado de JTable para modeloTabla
69           JTable tablaResultados = new JTable( modeloTabla );
70
71           JLabel etiquetaFiltro = new JLabel( "Filtro:" );
72           final JTextField textoFiltro = new JTextField();
73           JButton botonFiltro = new JButton( "Aplicar filtro" );
74           Box boxSur = boxNorte.createHorizontalBox();
75
76           boxSur.add( etiquetaFiltro );
77           boxSur.add( textoFiltro );
78           boxSur.add( botonFiltro );
79
80           // coloca los componentes de la GUI en el panel de contenido
81           add( boxNorte, BorderLayout.NORTH );
82           add( new JScrollPane( tablaResultados ), BorderLayout.CENTER );
83           add( boxSur, BorderLayout.SOUTH );
84
85           // crea componente de escucha de eventos para botonEnviar
86           botonEnviar.addActionListener(
87
88              new ActionListener()
89              {
90                 // pasa la consulta al modelo de la tabla
91                 public void actionPerformed( ActionEvent evento )
92                 {
```

Figura 25.28 | Visualización del contenido de la base de datos `libros`. (Parte 2 de 5).

```
93                   // realiza una nueva consulta
94                   try
95                   {
96                      modeloTabla.establecerConsulta( areaConsulta.getText() );
97                   } // fin de try
98                   catch ( SQLException excepcionSql )
99                   {
100                     JOptionPane.showMessageDialog( null,
101                        excepcionSql.getMessage(), "Error en base de datos",
102                        JOptionPane.ERROR_MESSAGE );
103
104                     // trata de recuperarse de una consulta inválida del usuario
105                     // ejecutando la consulta predeterminada
106                     try
107                     {
108                        modeloTabla.establecerConsulta( CONSULTA_PREDETERMINADA );
109                        areaConsulta.setText( CONSULTA_PREDETERMINADA );
110                     } // fin de try
111                     catch ( SQLException excepcionSql2 )
112                     {
113                        JOptionPane.showMessageDialog( null,
114                           excepcionSql2.getMessage(), "Error en base de datos",
115                           JOptionPane.ERROR_MESSAGE );
116
117                        // verifica que esté cerrada la conexión a la base de datos
118                        modeloTabla.desconectarDeBaseDatos();
119
120                        System.exit( 1 ); // termina la aplicación
121                     } // fin de catch interior
122                  } // fin de catch exterior
123               } // fin de actionPerformed
124            } // fin de la clase interna ActionListener
125         ); // fin de la llamada a addActionListener
126
127         final TableRowSorter< TableModel > sorter =
128            new TableRowSorter< TableModel >( modeloTabla );
129         tablaResultados.setRowSorter( sorter );
130         setSize( 500, 250 ); // establece el tamaño de la ventana
131         setVisible( true ); // muestra la ventana
132
133         // crea componente de escucha para botonFiltro
134         botonFiltro.addActionListener(
135            new ActionListener()
136            {
137               // pasa el texto del filtro al componente de escucha
138               public void actionPerformed( ActionEvent e )
139               {
140                  String texto = textoFiltro.getText();
141
142                  if ( texto.length() == 0 )
143                     sorter.setRowFilter( null );
144                  else
145                  {
146                     try
147                     {
148                        sorter.setRowFilter(
149                           RowFilter.regexFilter( texto ) );
150                     } // fin de try
151                     catch ( PatternSyntaxException pse )
```

Figura 25.28 | Visualización del contenido de la base de datos `libros`. (Parte 3 de 5).

```
                     {
                        JOptionPane.showMessageDialog( null,
                           "Patron de exp reg incorrecto", "Patron de exp reg incorrecto",
                           JOptionPane.ERROR_MESSAGE );
                     } // fin de catch
                  } // fin de else
               } // fin del método actionPerfomed
            } // fin de la clase interna anónima
         ); // fin de la llamada a addActionLister
      } // fin de try
      catch ( ClassNotFoundException noEncontroClase )
      {
         JOptionPane.showMessageDialog( null,
            "No se encontro controlador de base de datos", "No se encontro el
            controlador",
            JOptionPane.ERROR_MESSAGE );

         System.exit( 1 ); // termina la aplicación
      } // fin de catch
      catch ( SQLException excepcionSql )
      {
         JOptionPane.showMessageDialog( null, excepcionSql.getMessage(),
            "Error en base de datos", JOptionPane.ERROR_MESSAGE );

         // verifica que esté cerrada la conexión a la base de datos
         modeloTabla.desconectarDeBaseDatos();

         System.exit( 1 ); // termina la aplicación
      } // fin de catch

      // cierra la ventana cuando el usuario sale de la aplicación (se sobrescribe
      // el valor predeterminado de HIDE_ON_CLOSE)
      setDefaultCloseOperation( DISPOSE_ON_CLOSE );

      // verifica que esté cerrada la conexión a la base de datos cuando el usuario sale
      de la aplicación
      addWindowListener(

         new WindowAdapter()
         {
            // se desconecta de la base de datos y sale cuando se ha cerrado la ventana
            public void windowClosed( WindowEvent evento )
            {
               modeloTabla.desconectarDeBaseDatos();
               System.exit( 0 );
            } // fin del método windowClosed
         } // fin de la clase interna WindowAdapter
      ); // fin de la llamada a addWindowListener
   } // fin del constructor de MostrarResultadosConsulta

   // ejecuta la aplicación
   public static void main( String args[] )
   {
      new MostrarResultadosConsulta();
   } // fin de main
} // fin de la clase MostrarResultadosConsulta
```

Figura 25.28 | Visualización del contenido de la base de datos `libros`. (Parte 4 de 5).

Figura 25.28 | Visualización del contenido de la base de datos `libros`. (Parte 5 de 5).

En las líneas 86 a 125 se registra un manejador de eventos para el `botonEnviar` en el que el usuario hace clic para enviar una consulta a la base de datos. Cuando el usuario hace clic en el botón, el método `actionPerformed` (líneas 91 a 123) invoca al método `establecerConsulta` de la clase `ResultSetTableModel` para ejecutar la nueva consulta. Si la consulta del usuario falla (por ejemplo, debido a un error de sintaxis en la entrada del usuario), en las líneas 108 y 109 se ejecuta la consulta predeterminada. Si la consulta predeterminada también falla, podría haber un error más grave, por lo que en la línea 118 se asegura que se cierre la conexión a la base de datos y en la línea 120 se sale del programa. Las capturas de pantalla de la figura 25.28 muestran los resultados de dos consultas. La primera captura de pantalla muestra la consulta predeterminada, en la que se recuperan todos los datos de la tabla `autores` de la base de datos `libros`. La segunda captura de pantalla muestra una consulta que selecciona el nombre de pila y el apellido paterno de cada autor en la tabla `autores`, y combina esa información con el título y número de edición de la tabla `titulos`. Pruebe a introducir sus propias consultas en el área de texto y haga clic en el botón **Enviar consulta** para enviar la consulta.

A partir de Java SE 6, los objetos `JTable` ahora permiten a los usuarios ordenar filas en base a los datos en una columna específica. En las líneas 127 y 128 se utiliza la clase `TableRowSorter` (del paquete `javax.swing.table`) para crear un objeto que utilice nuestro objeto `ResultSetTableModel` para ordenar filas en el objeto `JTable` que muestre los resultados de la consulta. Cuando el usuario hace clic en el título de una columna

específica del objeto JTable, el objeto TableRowSorter interactúa con el objeto TableModel subyacente para reordenar las filas, con base en los datos en esa columna. En la línea 129 se utiliza el método **setRowSorter** del método JTable para especificar el objeto TableRowSorter para tablaResultados.

Otra de las nuevas características de los objetos JTable es la habilidad de ver subconjuntos de los datos del objeto TableModel subyacente. A esto se le conoce como filtrar los datos. En las líneas 134 a 160 se registra un manejador de eventos para el botonFiltro que el usuario oprime para filtrar los datos. En el método actionPerformed (líneas 138 a 158), en la línea 140 se obtiene el texto de filtro. Si el usuario no especificó un texto de filtro, en la línea 143 se utiliza el método **setRowFilter** de JTable para eliminar cualquier filtro anterior, estableciendo el filtro en null. En caso contrario, en las líneas 148 y 149 se utiliza setRowFilter para especificar un objeto **RowFilter** (del paquete javax.swing) con base en la entrada del usuario. La clase RowFilter cuenta con varios métodos para crear filtros. El método static **regexFilter** recibe un objeto String que contiene un patrón de expresión regular como argumento, y un conjunto opcional de índices que especifican cuáles columnas se van a filtrar. Si no se especifican índices, entonces se busca en todas las columnas. En este ejemplo, el patrón de expresión regular es el texto que escribió el usuario. Una vez establecido el filtro, se actualizan los datos mostrados en el objeto JTable con base en el objeto TableModel filtrado.

25.9 La interfaz RowSet

En los ejemplos anteriores, aprendido a realizar consultas en una base de datos al establecer en forma explícita una conexión (Connection) a la base de datos, preparar una instrucción (Statement) para consultar la base de datos y ejecutar la consulta. En esta sección demostraremos la interfaz RowSet, la cual configura la conexión a la base de datos y prepara instrucciones de consulta en forma automática. La interfaz RowSet proporciona varios métodos *establecer (get)* que nos permiten especificar las propiedades necesarias para establecer una conexión (como el URL de la base de datos, el nombre de usuario y la contraseña) y crear un objeto Statement (como una consulta). RowSet también cuenta con varios métodos *obtener (get)* para devolver estas propiedades.

Hay dos tipos de objetos RowSet: conectados y desconectados. Un objeto **RowSet conectado** se conecta a la base de datos una sola vez, y permanece conectado hasta que termina la aplicación. Un objeto **RowSet desconectado** se conecta a la base de datos, ejecuta una consulta para obtener los datos de la base de datos y después cierra la conexión. Un programa puede cambiar los datos en un objeto RowSet desconectado, mientras éste se encuentre desconectado. Los datos modificados pueden actualizarse en la base de datos, después de que un objeto RowSet desconectado reestablece la conexión a la base de datos.

El paquete **javax.sql.rowset** contiene dos subinterfaces de RowSet: JdbcRowSet y CachedRowSet. **JdbcRowSet**, un objeto RowSet conectado, actúa como una envoltura alrededor de un objeto ResultSet y nos permite desplazarnos a través de las filas en el objeto ResultSet, y también actualizarlas. Recuerde que, de manera predeterminada, un objeto ResultSet no es desplazable y es de sólo lectura; debemos establecer explícitamente la constante de tipo del conjunto de resultados a TYPE_SCROLL_INSENSITIVE y establecer la constante de concurrencia del conjunto de resultados CONCUR_UPDATABLE para hacer que un objeto ResultSet sea desplazable y pueda actualizarse. Un objeto JdbcRowSet es desplazable y puede actualizarse de manera predeterminada. **CachedRowSet**, un objeto RowSet desconectado, coloca los datos de un objeto ResultSet en caché de memoria y los desconecta de la base de datos. Al igual que un objeto JdbcRowSet, un objeto CachedRowSet es desplazable y puede actualizarse de manera predeterminada. Un objeto CachedRowSet también es serializable, por lo que puede pasarse de una aplicación de Java a otra mediante una red, como Internet. Sin embargo, CachedRowSet tiene una limitación: la cantidad de datos que pueden almacenarse en memoria es limitada. El paquete javax.sql.rowset contiene otras tres subinterfaces de RowSet. Para obtener detalles de estas interfaces, visite java.sun.com/javase/6/docs/technotes/guides/jdbc/getstart/rowsetImpl.html.

Tip de portabilidad 25.5

Un objeto RowSet puede proporcionar capacidad de desplazamiento a los controladores que no tienen soporte para objetos ResultSet desplazables.

En la figura 25.29 se reimplementa el ejemplo de la figura 25.23, usando un objeto RowSet. En vez de establecer la conexión y crear un objeto Statement de manera explícita, en la figura 25.29 utilizamos un objeto JdbcRowSet para crear los objetos Connection y Statement de manera automática.

```
1  // Fig. 25.29: PruebaJdbcRowSet.java
2  // Visualización del contenido de la tabla autores, mediante el uso de JdbcRowSet.
3  import java.sql.ResultSetMetaData;
4  import java.sql.SQLException;
5  import javax.sql.rowset.JdbcRowSet;
6  import com.sun.rowset.JdbcRowSetImpl; // implementación de JdbcRowSet de Sun
7
8  public class PruebaJdbcRowSet
9  {
10    // nombre del controlador de JDBC y URL de la base de datos
11    static final String CONTROLADOR = "com.mysql.jdbc.Driver";
12    static final String URL_BASEDATOS = "jdbc:mysql://localhost/libros";
13    static final String NOMBREUSUARIO = "jhtp7";
14    static final String CONTRASENIA = "jhtp7";
15
16    // el constructor se conecta a la base de datos, la consulta, procesa
17    // los resultados y los muestra en la ventana
18    public PruebaJdbcRowSet()
19    {
20       // se conecta a la base de datos libros y la consulta
21       try
22       {
23          Class.forName( CONTROLADOR );
24
25          // especifica las propiedades del objeto JdbcRowSet
26          JdbcRowSet rowSet = new JdbcRowSetImpl();
27          rowSet.setUrl( URL_BASEDATOS ); // establece URL de la base de datos
28          rowSet.setUsername( NOMBREUSUARIO ); // establece el nombre de usuario
29          rowSet.setPassword( CONTRASENIA ); // establece contraseña
30          rowSet.setCommand( "SELECT * FROM autores" ); // establece la consulta
31          rowSet.execute(); // ejecuta la consulta
32
33          // procesa los resultados de la consulta
34          ResultSetMetaData metaDatos = rowSet.getMetaData();
35          int numeroDeColumnas = metaDatos.getColumnCount();
36          System.out.println( "Tabla Autores de la base de datos Libros:\n" );
37
38          // muestra el encabezado del objeto RowSet
39          for ( int i = 1; i <= numeroDeColumnas; i++ )
40             System.out.printf( "%-8s\t", metaDatos.getColumnName( i ) );
41          System.out.println();
42
43          // muestra cada fila
44          while ( rowSet.next() )
45          {
46             for ( int i = 1; i <= numeroDeColumnas; i++ )
47                System.out.printf( "%-8s\t", rowSet.getObject( i ) );
48             System.out.println();
49          } // fin de while
50
51          // cierra el objeto ResultSet subyacente, y los objetos Statement y Connection
52          rowSet.close();
53       } // fin de try
54       catch ( SQLException excepcionSql )
55       {
56          excepcionSql.printStackTrace();
57          System.exit( 1 );
58       } // fin de catch
59       catch ( ClassNotFoundException noEncontroClase )
```

Figura 25.29 | Visualización de la tabla `autores` mediante `JdbcRowSet`. (Parte 1 de 2).

```
60            {
61               noEncontroClase.printStackTrace();
62               System.exit( 1 );
63            } // fin de catch
64         } // fin del constructor de mostrarAutores
65
66      // inicia la aplicación
67      public static void main( String args[] )
68      {
69         PruebaJdbcRowSet aplicacion = new PruebaJdbcRowSet();
70      } // fin de main
71   } // fin de la clase PruebaJdbcRowSet
```

```
Tabla Autores de la base de datos Libros:

IDAutor        nombrePila        apellidoPaterno
1              Harvey            Deitel
2              Paul              Deitel
3              Andrew            Goldberg
4              David             Choffnes
```

Figura 25.29 | Visualización de la tabla `autores` mediante `JdbcRowSet`. (Parte 2 de 2).

El paquete `com.sun.rowset` proporciona las implementaciones de referencia de Sun de las interfaces en el paquete `javax.sql.rowset`. En la línea 26 se utiliza la implementación de referencia de Sun de la interfaz `JdbcRowSet` (`JdbcRowSetImpl`) para crear un objeto `JdbcRowSet`. Utilizamos la clase `JdbcRowSetImpl` aquí para demostrar la capacidad de la herramienta `JdbcRowSet`. Otras bases de datos pueden proporcionar sus propias implementaciones de `RowSet`.

En las líneas 27 a 29 se establecen las propiedades de `RowSet` que utiliza el objeto `DriverManager` para establecer una conexión a la base de datos. En la línea 27 se invoca el método `setUrl` de `JdbcRowSet` para especificar el URL de la base de datos. En la línea 28 se invoca el método `setUsername` de `JdbcRowSet` para especificar el nombre de usuario. En la línea 29 se invoca el método `setPassword` de `JdbcRowSet` para especificar la contraseña. En la línea 30 se invoca el método `setCommand` de `JdbcRowSet` para especificar la consulta SQL que se utilizará para llenar el objeto `RowSet`. En la línea 31 se invoca el método `execute` de `JdbcRowSet` para ejecutar la consulta SQL. El método `execute` realiza cuatro acciones: establece una conexión (`Connection`), prepara la instrucción (`Statement`) de consulta, ejecuta la consulta y almacena el objeto `ResultSet` devuelto por la consulta. Los objetos `Connection`, `Statement` y `ResultSet` se encapsulan en el objeto `JdbcRowSet`.

El resto del código es casi idéntico al de la figura 25.23, excepto que en la línea 34 se obtiene un objeto `ResultSetMetaData` del objeto `JdbcRowSet`, en la línea 44 se utiliza el método `next` de `JdbcRowSet` para obtener la siguiente fila del resultado, y en la línea 47 se utiliza el método `getObject` de `JdbcRowSet` para obtener el valor de una columna. En la línea 52 se invoca el método `close` de `JdbcRowSet`, el cual cierra los objetos `ResultSet`, `Statement` y `Connection` de `RowSet`. En un objeto `CachedRowSet`, al invocar `close` también se libera la memoria ocupada por ese objeto `RowSet`. Observe que la salida de esta aplicación es igual que la de la figura 25.23.

25.10 Java DB/Apache Derby

A partir del JDK 6, Sun Microsystems está incluyendo la base de datos basada exclusivamente en Java de código fuente abierto, llamada **Java DB** (la versión de Apache Derby producida por Sun) junto con el JDK. En la sección 25.11 utilizaremos Java DB para demostrar una nueva característica de JDBC 4.0, y demostraremos las instrucciones denominadas `PreparedStatements`. Para que pueda ejecutar la aplicación de la siguiente sección, debe configurar la base de datos `LibretaDirecciones` en Java DB. En la sección 25.11 usaremos la versión incrustada de Java DB. También hay una versión en red, que se ejecuta de manera similar al DBMS MySQL que presentamos en secciones anteriores. Para los siguientes pasos, vamos a suponer que usted está ejecutando Microsoft Windows con Java instalado en su ubicación predeterminada.

1. Java DB incluye varios archivos de procesamiento por lotes para su configuración y ejecución. Antes de ejecutar estos archivos por lotes desde un símbolo del sistema, debe establecer la variable de entorno `JAVA_HOME` para que haga referencia al directorio de instalación `C:\Archivos de programa\Java\jdk1.6.0` del JDK. Para obtener información acerca de cómo establecer el valor de una variable de entorno, consulte la sección *Antes de empezar* de este libro.
2. Abra el archivo de procesamiento por lotes `setEmbeddedCP.bat` (ubicado en `C:\Archivos de programa\Java\jdk1.6.0\db\frameworks\embedded\bin`) en un editor de texto, como el Bloc de notas. Localice la línea

   ```
   rem set DERBY_INSTALL=
   ```

   y cámbiela por

   ```
   set DERBY_INSTALL=C:\Archivos de programa\Java\jdk1.6.0\db
   ```

   Después, convierta en comentario la siguiente línea:

   ```
   @FOR %%X in ("%DERBY_HOME%") DO SET DERBY_HOME=%%~sX
   ```

   a la cual debe anteponer la palabra clave `REM`, de la siguiente manera:

   ```
   REM @FOR %%X in ("%DERBY_HOME%") DO SET DERBY_HOME=%%~sX
   ```

   Guarde sus cambios y cierre este archivo.
3. Abra una ventana de Símbolo del sistema y cambie al directorio `C:\Archivos de programa\Java\jdk1.6.0\db\frameworks\embedded\bin`. Después, escriba `setEmbeddedCP.bat` y oprima *Intro* para establecer las variables de entorno requeridas por Java DB.
4. Una base de datos Java DB incrustada debe residir en la misma ubicación que la aplicación que manipula a la base de datos. Por esta razón, cambie al directorio que contiene el código para las figuras 25.30 a 25.32. Este directorio contiene un archivo de secuencia de comandos SQL llamado `direccion.sql`, el cual crea la base de datos `LibretaDirecciones`.
5. Ejecute el comando

   ```
   "C:\Archivos de programa\Java\jdk1.6.0\db\frameworks\embedded\bin\ij"
   ```

   para iniciar la herramienta de línea de comandos para interactuar con Java DB. Las comillas dobles son necesarias, ya que la ruta contiene un espacio. Esto mostrará el indicador `ij>`.
6. En el indicador `ij>`, escriba

   ```
   connect 'jdbc:derby:LibretaDirecciones;create=true;user=jhtp7;password=jhtp7';
   ```

   para crear la base de datos `LibretaDirecciones` en el directorio actual. Este comando también crea el usuario `jhtp7` con la contraseña `jhtp7` para acceder a la base de datos.
7. Para crear la tabla de la base de datos e insertar los datos de ejemplo, escriba

   ```
   run 'direccion.sql';
   ```

8. Para terminar la herramienta de línea de comandos de Java DB, escriba

   ```
   exit;
   ```

   Ahora está listo para ejecutar la aplicación `LibretaDirecciones` en la sección 25.12.

25.11 Objetos `PreparedStatement`

La interfaz **`PreparedStatement`** nos permite crear instrucciones SQL compiladas, que se ejecutan con más eficiencia que los objetos `Statement`. Las instrucciones `PreparedStatement` también pueden especificar parámetros, lo cual las hace más flexibles que las instrucciones `Statement`. Los programas pueden ejecutar la misma

consulta varias veces, con distintos valores para los parámetros. Por ejemplo, en la base de datos `libros`, tal vez sea conveniente localizar todos los libros para un autor con un apellido paterno y primer nombre específicos, y ejecutar esa consulta para varios autores. Con un objeto `PreparedStatement`, esa consulta se define de la siguiente manera:

```
PreparedStatement librosAutor = connection.prepareStatement(
    "SELECT apellidoPaterno, primerNombre, titulo " +
    "FROM autores INNER JOIN isbnAutor " +
        "ON autores.idAutor=isbnAutor.idAutor " +
    "INNER JOIN titulos " +
        "ON isbnAutor.isbn=titulos.isbn " +
    "WHERE apellidoPaterno = ? AND primerNombre = ?" );
```

Los dos signos de interrogación (?) en la última línea de la instrucción SQL anterior son receptáculos para valores que se pasarán como parte de la consulta en la base de datos. Antes de ejecutar una instrucción `PreparedStatement`, el programa debe especificar los valores de los parámetros mediante el uso de los métodos *establecer (set)* de la interfaz `PreparedStatement`.

Para la consulta anterior, ambos parámetros son cadenas que pueden establecerse con el método **`setString`** de `PreparedStatement`, como se muestra a continuación:

```
librosAutor.setString( 1, "Deitel" );
librosAutor.setString( 2, "Paul" );
```

El primer argumento del método `setString` representa el número del parámetro que se va a establecer, y el segundo argumento es el valor de ese parámetro. Los números de los parámetros se cuentan a partir de 1, empezando con el primer signo de interrogación (?). Cuando el programa ejecuta la instrucción `PreparedStatement` anterior con los valores de los parámetros que se muestran aquí, la instrucción SQL que se pasa a la base de datos es

```
SELECT apellidoPaterno, primerNombre, titulo
FROM autores INNER JOIN isbnAutor
    ON autores.idAutor=isbnAutor.idAutor
INNER JOIN titulos
    ON isbnAutor.isbn=titulos.isbn
WHERE apellidoPaterno = 'Deitel' AND primerNombre = 'Paul'
```

El método `setString` escapa de manera automática los valores de los parámetros `String` según sea necesario. Por ejemplo, si el apellido paterno es O'Brien, la instrucción

```
librosAutor.setString( 1, "O'Brien" );
```

escapa el carácter `'` en O'Brien, sustituyéndolo con dos caracteres de comilla sencilla.

Tip de rendimiento 25.2

Las instrucciones `PreparedStatement` son más eficientes que las instrucciones `Statement` al ejecutar instrucciones SQL varias veces, y con distintos valores para los parámetros.

Tip para prevenir errores 25.1

Use las instrucciones `PreparedStatement` con parámetros para las consultas que reciban valores `String` como argumentos, para asegurar que los objetos `String` utilicen comillas de manera apropiada en la instrucción SQL.

La interfaz `PreparedStatement` proporciona métodos *establecer (set)* para cada tipo de SQL soportado. Es importante utilizar el método *establecer* que sea apropiado para el tipo de SQL del parámetro en la base de datos; las excepciones `SQLException` ocurren cuando un programa trata de convertir el valor de un parámetro en un tipo incorrecto. Para una lista completa de los métodos *establecer (set)* de la interfaz `PreparedStatement`, consulte la página Web `java.sun.com/javase/6/docs/api/java/sql/PreparedStatement.html`.

Aplicación "libreta de direcciones" que utiliza instrucciones `PreparedStatement`

Ahora presentaremos una aplicación "libreta de direcciones" que nos permite explorar las entradas existentes, agregar nuevas entradas y buscar entradas con un apellido paterno específico. Nuestra base de datos `LibretaDirecciones` de JavaDB contiene una tabla `Direcciones` con las columnas `idDireccion`, `primerNombre`, `apellidoPaterno`, `email` y `numeroTelefonico`. La columna `idDireccion` se denomina columna de identidad. Ésta es la manera estándar de SQL para representar una columna autoincrementada. La secuencia de comandos SQL que proporcionamos para esta base de datos utiliza la palabra clave `IDENTITY` de SQL para marcar la columna `idDireccion` como una columna de identidad. Para obtener más información acerca de la palabra `IDENTITY` y crear bases de datos, consulte la Guía para el desarrollador de Java DB en `developers.sun.com/prodtech/javadb/reference/docs/10.2.1.6/devguide/index.html`.

Nuestra aplicación de libro de direcciones consiste en tres clases: `Persona` (figura 25.30), `ConsultasPersona` (figura 25.31) y `MostrarLibretaDirecciones` (figura 25.32). La clase `Persona` es una clase simple que representa a una persona en la libreta de direcciones. Esta clase contiene campos para el ID de dirección, primer nombre, apellido paterno, dirección de e-mail y número telefónico, así como métodos *establecer* y *obtener* para manipular esos campos.

```
1  // Fig. 25.30: Persona.java
2  // La clase Persona representa una entrada en una libreta de direcciones.
3  public class Persona
4  {
5     private int idDireccion;
6     private String primerNombre;
7     private String apellidoPaterno;
8     private String email;
9     private String numeroTelefonico;
10
11    // constructor sin argumentos
12    public Persona()
13    {
14    } // fin del constructor de Persona sin argumentos
15
16    // constructor
17    public Persona( int id, String nombre, String apellido,
18       String direccionEmail, String telefono )
19    {
20       establecerIDDireccion( id );
21       establecerPrimerNombre( nombre );
22       establecerApellidoPaterno( apellido );
23       establecerEmail( direccionEmail );
24       establecerNumeroTelefonico( telefono );
25    } // fin del constructor de Persona con cinco argumentos
26
27    // establece el objeto idDireccion
28    public void establecerIDDireccion( int id )
29    {
30       idDireccion = id;
31    } // fin del método establecerIDDireccion
32
33    // devuelve el valor de idDireccion
34    public int obtenerIDDireccion()
35    {
36       return idDireccion;
37    } // fin del método obtenerIDDireccion
38
39    // establece el primerNombre
40    public void establecerPrimerNombre( String nombre )
```

Figura 25.30 | La clase `Persona` representa una entrada en un objeto `LibretaDirecciones`. (Parte 1 de 2).

```
41      {
42         primerNombre = nombre;
43      } // fin del método establecerPrimerNombre
44
45      // devuelve el primer nombre
46      public String obtenerPrimerNombre()
47      {
48         return primerNombre;
49      } // fin del método obtenerPrimerNombre
50
51      // establece el apellidoPaterno
52      public void establecerApellidoPaterno( String apellido )
53      {
54         apellidoPaterno = apellido;
55      } // fin del método establecerApellidoPaterno
56
57      // devuelve el apellido paterno
58      public String obtenerApellidoPaterno()
59      {
60         return apellidoPaterno;
61      } // fin del método obtenerApellidoPaterno
62
63      // establece la dirección de email
64      public void establecerEmail( String direccionEmail )
65      {
66         email = direccionEmail;
67      } // fin del método establecerEmail
68
69      // devuelve la dirección de email
70      public String obtenerEmail()
71      {
72         return email;
73      } // fin del método obtenerEmail
74
75      // establece el número telefónico
76      public void establecerNumeroTelefonico( String telefono )
77      {
78         numeroTelefonico = telefono;
79      } // fin del método establecerNumeroTelefonico
80
81      // devuelve el número telefónico
82      public String obtenerNumeroTelefonico()
83      {
84         return numeroTelefonico;
85      } // fin del método obtenerNumeroTelefonico
86   } // fin de la clase Persona
```

Figura 25.30 | La clase `Persona` representa una entrada en un objeto `LibretaDirecciones`. (Parte 2 de 2).

La clase `ConsultasPersona`

La clase `ConsultasPersona` (figura 25.31) maneja la conexión a la base de datos de la aplicación libreta de direcciones y crea las instrucciones `PreparedStatement` que utiliza la aplicación para interactuar con la base de datos. En las líneas 18 a 20 se declaran tres variables `PreparedStatement`. El constructor (líneas 23 a 49) se conecta con la base de datos en las líneas 27 y 28. Observe que no utilizamos `Class.forName` para cargar el controlador de la base de datos para Java DB, como hicimos en los ejemplos que utilizan MySQL en secciones anteriores de este capítulo. JDBC 4.0, que forma parte de Java SE 6, soporta el **descubrimiento automático de controladores**; ya no tenemos que cargar el controlador de la base de datos por adelantado. Al momento de escribir este libro, esta característica se encuentra en proceso de implementarse en MySQL.

```
1  // Fig. 25.31: ConsultasPersona.java
2  // Instrucciones PreparedStatement utilizadas por la aplicación Libreta de direcciones
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.PreparedStatement;
6  import java.sql.ResultSet;
7  import java.sql.SQLException;
8  import java.util.List;
9  import java.util.ArrayList;
10
11 public class ConsultasPersona
12 {
13    private static final String URL = "jdbc:derby:LibretaDirecciones";
14    private static final String NOMBREUSUARIO = "jhtp7";
15    private static final String CONTRASENIA = "jhtp7";
16
17    private Connection conexion = null; // maneja la conexión
18    private PreparedStatement seleccionarTodasLasPersonas = null;
19    private PreparedStatement seleccionarPersonasPorApellido = null;
20    private PreparedStatement insertarNuevaPersona = null;
21
22    // constructor
23    public ConsultasPersona()
24    {
25       try
26       {
27          conexion =
28             DriverManager.getConnection( URL, NOMBREUSUARIO, CONTRASENIA );
29
30          // crea una consulta que selecciona todas las entradas en la LibretaDirecciones
31          seleccionarTodasLasPersonas =
32             conexion.prepareStatement( "SELECT * FROM Direcciones" );
33
34          // crea una consulta que selecciona las entradas con un apellido específico
35          seleccionarPersonasPorApellido = conexion.prepareStatement(
36             "SELECT * FROM Direcciones WHERE ApellidoPaterno = ?" );
37
38          // crea instrucción insert para agregar una nueva entrada en la base de 39
            datos
39          insertarNuevaPersona = conexion.prepareStatement(
40             "INSERT INTO Direcciones " +
41             "( PrimerNombre, ApellidoPaterno, Email, NumeroTelefonico ) " +
42             "VALUES ( ?, ?, ?, ? )" );
43       } // fin de try
44       catch ( SQLException excepcionSql )
45       {
46          excepcionSql.printStackTrace();
47          System.exit( 1 );
48       } // fin de catch
49    } // fin del constructor de ConsultasPersona
50
51    // selecciona todas las direcciones en la base de datos
52    public List< Persona > obtenerTodasLasPersonas()
53    {
54       List< Persona > resultados = null;
55       ResultSet conjuntoResultados = null;
56
57       try
```

Figura 25.31 | Una interfaz que almacena todas las consultas para que las utilice un objeto `LibretaDirecciones`. (Parte 1 de 4).

```
58         {
59            // executeQuery devuelve ResultSet que contiene las entradas que coinciden
60            conjuntoResultados = seleccionarTodasLasPersonas.executeQuery();
61            resultados = new ArrayList< Persona >();
62
63            while ( conjuntoResultados.next() )
64            {
65               resultados.add( new Persona(
66                  conjuntoResultados.getInt( "idDireccion" ),
67                  conjuntoResultados.getString( "primerNombre" ),
68                  conjuntoResultados.getString( "apellidoPaterno" ),
69                  conjuntoResultados.getString( "email" ),
70                  conjuntoResultados.getString( "numeroTelefonico" ) ) );
71            } // fin de while
72         } // fin de try
73         catch ( SQLException excepcionSql )
74         {
75            excepcionSql.printStackTrace();
76         } // fin de catch
77         finally
78         {
79            try
80            {
81               conjuntoResultados.close();
82            } // fin de try
83            catch ( SQLException excepcionSql )
84            {
85               excepcionSql.printStackTrace();
86               close();
87            } // fin de catch
88         } // fin de finally
89
90         return resultados;
91      } // fin del método obtenerTodasLasPersonaas
92
93      // selecciona persona por apellido paterno
94
95      public List< Persona > obtenerPersonasPorApellido( String nombre )
96      {
97         List< Persona > resultados = null;
98         ResultSet conjuntoResultados = null;
99
100        try
101        {
102           seleccionarPersonasPorApellido.setString( 1, nombre ); // especifica el
              apellido paterno
103
104           // executeQuery devuelve ResultSet que contiene las entradas que coinciden
105           conjuntoResultados = seleccionarPersonasPorApellido.executeQuery();
106
107           resultados = new ArrayList< Persona >();
108
109           while ( conjuntoResultados.next() )
110           {
111              resultados.add( new Persona(
112                 conjuntoResultados.getInt( "idDireccion" ),
113                 conjuntoResultados.getString( “primerNombre" ),
114                 conjuntoResultados.getString( "apellidoPaterno" ),
```

Figura 25.31 | Una interfaz que almacena todas las consultas para que las utilice un objeto `LibretaDirecciones`. (Parte 2 de 4).

```
115                conjuntoResultados.getString( "email" ),
116                conjuntoResultados.getString( "numeroTelefonico" ) ) );
117             } // fin de while
118          } // fin de try
119          catch ( SQLException excepcionSql )
120          {
121             excepcionSql.printStackTrace();
122          } // fin de catch
123          finally
124          {
125             try
126             {
127                conjuntoResultados.close();
128             } // fin de try
129             catch ( SQLException excepcionSql )
130             {
131                excepcionSql.printStackTrace();
132                close();
133             } // fin de catch
134          } // fin de finally
135
136          return resultados;
137       } // fin del método obtenerPersonasPorApellido
138
139       // agrega una entrada
140       public int agregarPersona(
141          String pnombre, String apaterno, String email, String num )
142       {
143          int resultado = 0;
144
145          // establece los parámetros, después ejecuta insertarNuevaPersona
146          try
147          {
148             insertarNuevaPersona.setString( 1, pnombre );
149             insertarNuevaPersona.setString( 2, apaterno );
150             insertarNuevaPersona.setString( 3, email );
151             insertarNuevaPersona.setString( 4, num );
152
153             // inserta la nueva entrada; devuelve # de filas actualizadas
154             resultado = insertarNuevaPersona.executeUpdate();
155          } // fin de try
156          catch ( SQLException excepcionSql )
157          {
158             excepcionSql.printStackTrace();
159             close();
160          } // fin de catch
161
162          return resultado;
163       } // fin del método agregarPersona
164
165       // cierra la conexión a la base de datos
166       public void close()
167       {
168          try
169          {
170             conexion.close();
171          } // fin de try
172          catch ( SQLException excepcionSql )
```

Figura 25.31 | Una interfaz que almacena todas las consultas para que las utilice un objeto `LibretaDirecciones`. (Parte 3 de 4).

```
173         {
174            excepcionSql.printStackTrace();
175         } // fin de catch
176      } // fin del método close
177   } // fin de la interfaz ConsultasPersona
```

Figura 25.31 | Una interfaz que almacena todas las consultas para que las utilice un objeto `LibretaDirecciones`. (Parte 4 de 4).

En las líneas 31 y 32 se invoca el método **`prepareStatement`** de `Connection` para crear la instrucción `PreparedStatement` llamada `seleccionarTodasLasPersonas`, la cual selecciona todas las filas en la tabla `Direcciones`. En las líneas 35 y 36 se crea la instrucción `PreparedStatement` llamada `seleccionarPersonasPorApellido` con un parámetro. Esta instrucción selecciona todas las filas en la tabla `Direcciones` que coincidan con un apellido específico. Observe el carácter `?` que se utiliza para especificar el parámetro apellido. En las líneas 39 a 42 se crea la instrucción `PreparedStatement` llamada `insertarNuevaPersona`, concuatro parámetros que representan el primer nombre, apellido paterno, dirección de e-mail y número telefónico para una nueva entrada. De nuevo, observe los caracteres `?` que se utilizan para representar estos parámetros.

El método `obtenerTodasLasPersonas` (líneas 52 a 91) ejecuta la instrucción `PreparedStatement` `seleccionarTodasLasPersonas` (línea 60) mediante una llamada al método **`executeQuery`**, el cual devuelve un objeto `ResultSet` que contiene las filas que coinciden con la consulta (en este caso, todas las filas en la tabla `Direcciones`). En las líneas 61 a 71 se colocan los resultados de la consulta en un objeto `ArrayList` de objetos `Persona`, el cual se devuelve en la línea 90 al método que hizo la llamada. El método `obtenerPersonasPorApellido` (líneas 95 a 137) utiliza el método **`setString`** de `PreparedStatement` para establecer el parámetro en `seleccionarPersonasPorApellido`. Después, en la línea 105 se ejecuta la consulta y en las líneas 107 a 117 se colocan los resultados de la consulta en un objeto `ArrayList` de objetos `Persona`. En la línea 136 se devuelve el objeto `ArrayList` al método que hizo la llamada.

El método `agregarPersona` (líneas 140 a 163) utiliza el método `setString` de `PreparedStatement` (líneas 148 a 151) para establecer los parámetros para la instrucción `PreparedStatement` llamada `insertarNuevaPersona`. La línea 154 utiliza el método **`executeUpdate`** de `PreparedStatement` para insertar un nuevo registro. Este método devuelve un entero, el cual indica el número de filas que se actualizaron (o insertaron) en la base de datos. El método `close` (líneas 166 a 176) simplemente cierra la conexión a la base de datos.

La clase `MostrarLibretaDirecciones`

La aplicación `MostrarLibretaDirecciones` (figura 25.32) utiliza un objeto de la clase `ConsultasPersona` para interactuar con la base de datos. En la línea 59 se crea el objeto `ConsultasPersona` que se utiliza en la clase `MostrarLibretaDirecciones`. Cuando el usuario oprime el objeto `JButton` llamado **Explorar todas las entradas**, se hace una llamada al manejador `botonExplorarActionPerformed` (líneas 309 a 335). En la línea 313 se hace una llamada al método `obtenerTodasLasPersonas` en el objeto `ConsultasPersona` para obtener todas las entradas en la base de datos. Después, el usuario puede desplazarse a través de las entradas, usando los objetos `JButton` **Anterior** y **Siguiente**. Cuando el usuario oprime el objeto `JButton` **Buscar**, se hace una llamada al manejador `botonConsultaActionPerformed` (líneas 265 a 287). En las líneas 267 y 268 se hace una llamada al método `obtenerPersonasPorApellido` en el objeto `ConsultasPersona`, para obtener las entradas en la base de datos que coincidan con el apellido paterno especificado. Si hay varias entradas de este tipo, el usuario puede desplazarse de una entrada a otra mediante los objetos `JButton` **Anterior** y **Siguiente**.

Para agregar una nueva entrada a la base de datos `LibretaDirecciones`, el usuario puede escribir el primer nombre, apellido paterno, email y número telefónico (el valor de `IdDireccion` se autoincrementará) en los objetos `JTextField` y oprimir el objeto `JButton` **Insertar nueva entrada**. Cuando el usuario oprime este botón, se hace una llamada al manejador `botonInsertarActionPerformed` (líneas 338 a 352). En las líneas 340 a 342 se hace una llamada al método `agregarPersona` en el objeto `ConsultasPersona`, para agregar una nueva entrada a la base de datos.

Después, el usuario puede ver distintas entradas oprimiendo los objetos `JButton` **Anterior** o **Siguiente**, lo cual produce llamadas a los métodos `botonAnteriorActionPerformed` (líneas 241 a 250) o `botonSiguien-`

```
1  // Fig. 25.32: MostrarLibretaDirecciones.java
2  // Una libreta de direcciones simple
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.awt.event.WindowAdapter;
6  import java.awt.event.WindowEvent;
7  import java.awt.FlowLayout;
8  import java.awt.GridLayout;
9  import java.util.List;
10 import javax.swing.JButton;
11 import javax.swing.Box;
12 import javax.swing.JFrame;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.JTextField;
16 import javax.swing.WindowConstants;
17 import javax.swing.BoxLayout;
18 import javax.swing.BorderFactory;
19 import javax.swing.JOptionPane;
20
21 public class MostrarLibretaDirecciones extends JFrame
22 {
23    private Persona entradaActual;
24    private ConsultasPersona consultasPersona;
25    private List< Persona > resultados;
26    private int numeroDeEntradas = 0;
27    private int indiceEntradaActual;
28
29    private JButton botonExplorar;
30    private JLabel etiquetaEmail;
31    private JTextField campoTextoEmail;
32    private JLabel etiquetaPrimerNombre;
33    private JTextField campoTextoPrimerNombre;
34    private JLabel etiquetaID;
35    private JTextField campoTextoID;
36    private JTextField campoTextoIndice;
37    private JLabel etiquetaApellidoPaterno;
38    private JTextField campoTextoApellidoPaterno;
39    private JTextField campoTextoMax;
40    private JButton botonSiguiente;
41    private JLabel etiquetaDe;
42    private JLabel etiquetaTelefono;
43    private JTextField campoTextoTelefono;
44    private JButton botonAnterior;
45    private JButton botonConsulta;
46    private JLabel etiquetaConsulta;
47    private JPanel panelConsulta;
48    private JPanel panelNavegar;
49    private JPanel panelMostrar;
50    private JTextField campoTextoConsulta;
51    private JButton botonInsertar;
52
53    // constructor sin argumentos
54    public MostrarLibretaDirecciones()
55    {
56       super( "Libreta de direcciones" );
57
58       // establece la conexión a la base de datos y las instrucciones PreparedStatement
59       consultasPersona = new ConsultasPersona();
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 1 de 7).

```
60
61         // crea la GUI
62         panelNavegar = new JPanel();
63         botonAnterior = new JButton();
64         campoTextoIndice = new JTextField( 2 );
65         etiquetaDe = new JLabel();
66         campoTextoMax = new JTextField( 2 );
67         botonSiguiente = new JButton();
68         panelMostrar = new JPanel();
69         etiquetaID = new JLabel();
70         campoTextoID = new JTextField( 10 );
71         etiquetaPrimerNombre = new JLabel();
72         campoTextoPrimerNombre = new JTextField( 10 );
73         etiquetaApellidoPaterno = new JLabel();
74         campoTextoApellidoPaterno = new JTextField( 10 );
75         etiquetaEmail = new JLabel();
76         campoTextoEmail = new JTextField( 10 );
77         etiquetaTelefono = new JLabel();
78         campoTextoTelefono = new JTextField( 10 );
79         panelConsulta = new JPanel();
80         etiquetaConsulta = new JLabel();
81         campoTextoConsulta = new JTextField( 10 );
82         botonConsulta = new JButton();
83         botonExplorar = new JButton();
84         botonInsertar = new JButton();
85
86         setLayout( new FlowLayout( FlowLayout.CENTER, 10, 10 ) );
87         setSize( 400, 300 );
88         setResizable( false );
89
90         panelNavegar.setLayout(
91            new BoxLayout( panelNavegar, BoxLayout.X_AXIS ) );
92
93         botonAnterior.setText( "Anterior" );
94         botonAnterior.setEnabled( false );
95         botonAnterior.addActionListener(
96            new ActionListener()
97            {
98               public void actionPerformed( ActionEvent evt )
99               {
100                 botonAnteriorActionPerformed( evt );
101              } // fin del método actionPerformed
102           } // fin de la clase interna anónima
103        ); // fin de la llamada a addActionListener
104
105        panelNavegar.add( botonAnterior );
106        panelNavegar.add( Box.createHorizontalStrut( 10 ) );
107
108        campoTextoIndice.setHorizontalAlignment(
109           JTextField.CENTER );
110        campoTextoIndice.addActionListener(
111           new ActionListener()
112           {
113              public void actionPerformed( ActionEvent evt )
114              {
115                 campoTextoIndiceActionPerformed( evt );
116              } // fin del método actionPerformed
117           } // fin de la clase interna anónima
118        ); // fin de la llamada a addActionListener
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 2 de 7).

```
119
120         panelNavegar.add( campoTextoIndice );
121         panelNavegar.add( Box.createHorizontalStrut( 10 ) );
122
123         etiquetaDe.setText( "de" );
124         panelNavegar.add( etiquetaDe );
125         panelNavegar.add( Box.createHorizontalStrut( 10 ) );
126
127         campoTextoMax.setHorizontalAlignment(
128            JTextField.CENTER );
129         campoTextoMax.setEditable( false );
130         panelNavegar.add( campoTextoMax );
131         panelNavegar.add( Box.createHorizontalStrut( 10 ) );
132
133         botonSiguiente.setText( "Siguiente" );
134         botonSiguiente.setEnabled( false );
135         botonSiguiente.addActionListener(
136            new ActionListener()
137            {
138               public void actionPerformed( ActionEvent evt )
139               {
140                  botonSiguienteActionPerformed( evt );
141               } // fin del método actionPerformed
142            } // fin de la clase interna anónima
143         ); // fin de la llamada a addActionListener
144
145         panelNavegar.add( botonSiguiente );
146         add( panelNavegar );
147
148         panelMostrar.setLayout( new GridLayout( 5, 2, 4, 4 ) );
149
150         etiquetaID.setText( "ID Direccion:" );
151         panelMostrar.add( etiquetaID );
152
153         campoTextoID.setEditable( false );
154         panelMostrar.add( campoTextoID );
155
156         etiquetaPrimerNombre.setText( "Primer nombre:" );
157         panelMostrar.add( etiquetaPrimerNombre );
158         panelMostrar.add( campoTextoPrimerNombre );
159
160         etiquetaApellidoPaterno.setText( "Apellido paterno:" );
161         panelMostrar.add( etiquetaApellidoPaterno );
162         panelMostrar.add( campoTextoApellidoPaterno );
163
164         etiquetaEmail.setText( "Email:" );
165         panelMostrar.add( etiquetaEmail );
166         panelMostrar.add( campoTextoEmail );
167
168         etiquetaTelefono.setText( "Telefono:" );
169         panelMostrar.add( etiquetaTelefono );
170         panelMostrar.add( campoTextoTelefono );
171         add( panelMostrar );
172
173         panelConsulta.setLayout(
174            new BoxLayout( panelConsulta, BoxLayout.X_AXIS) );
175
176         panelConsulta.setBorder( BorderFactory.createTitledBorder(
177            "Buscar una entrada por apellido" ) );
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 3 de 7).

```
178         etiquetaConsulta.setText( "Apellido paterno:" );
179         panelConsulta.add( Box.createHorizontalStrut( 5 ) );
180         panelConsulta.add( etiquetaConsulta );
181         anelConsulta.add( Box.createHorizontalStrut( 10 ) );
182         panelConsulta.add( campoTextoConsulta );
183         panelConsulta.add( Box.createHorizontalStrut( 10 ) );
184
185         botonConsulta.setText( "Buscar" );
186         botonConsulta.addActionListener(
187            new ActionListener()
188            {
189               public void actionPerformed( ActionEvent evt )
190               {
191                  botonConsultaActionPerformed( evt );
192               } // fin del método actionPerformed
193            } // fin de la clase interna anónima
194         ); // fin de la llamada a addActionListener
195
196         panelConsulta.add( botonConsulta );
197         panelConsulta.add( Box.createHorizontalStrut( 5 ) );
198         add( panelConsulta );
199
200         botonExplorar.setText( "Explorar todas las entradas" );
201         botonExplorar.addActionListener(
202            new ActionListener()
203            {
204               public void actionPerformed( ActionEvent evt )
205               {
206                  botonExplorarActionPerformed( evt );
207               } // fin del método actionPerformed
208            } // fin de la clase interna anónima
209         ); // fin de la llamada a addActionListener
210
211         add( botonExplorar );
212
213         botonInsertar.setText( "Insertar nueva entrada" );
214         botonInsertar.addActionListener(
215            new ActionListener()
216            {
217               public void actionPerformed( ActionEvent evt )
218               {
219                  botonInsertarActionPerformed( evt );
220               } // fin del método actionPerformed
221            } // fin de la clase interna anónima
222         ); // fin de la llamada a addActionListener
223
224         add( botonInsertar );
225
226          addWindowListener(
227            new WindowAdapter()
228            {
229               public void windowClosing( WindowEvent evt )
230               {
231                  consultasPersona.close(); // cierra la conexión a la base de datos
232                  System.exit( 0 );
233               } // fin del método windowClosing
234            } // fin de la clase interna anónima
235         ); // fin de la llamada a addWindowListener
236
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 4 de 7).

```
237         setVisible( true );
238      } // fin del constructor sin argumentos
239
240      // maneja la llamada cuando se hace clic en botonAnterior
241      private void botonAnteriorActionPerformed( ActionEvent evt )
242      {
243         indiceEntradaActual--;
244
245         if ( indiceEntradaActual < 0 )
246            indiceEntradaActual = numeroDeEntradas - 1;
247
248         campoTextoIndice.setText( "" + ( indiceEntradaActual + 1 ) );
249         campoTextoIndiceActionPerformed( evt );
250      } // fin del método botonAnteriorActionPerformed
251
252      // maneja la llamada cuando se hace clic en botonSiguiente
253      private void botonSiguienteActionPerformed( ActionEvent evt )
254      {
255         indiceEntradaActual++;
256
257         if ( indiceEntradaActual >= numeroDeEntradas )
258            indiceEntradaActual = 0;
259
260         campoTextoIndice.setText( "" + ( indiceEntradaActual + 1 ) );
261         campoTextoIndiceActionPerformed( evt );
262      } // fin del método botonSiguienteActionPerformed
263
264      // maneja la llamada cuando se hace clic en botonConsulta
265      private void botonConsultaActionPerformed( ActionEvent evt )
266      {
267         resultados =
268            consultasPersona.obtenerPersonasPorApellido( campoTextoConsulta.getText() );
269         numeroDeEntradas = resultados.size();
270
271         if ( numeroDeEntradas != 0 )
272         {
273            indiceEntradaActual = 0;
274            entradaActual = resultados.get( indiceEntradaActual );
275            campoTextoID.setText( "" + entradaActual.obtenerIDDireccion() );
276            campoTextoPrimerNombre.setText( entradaActual.obtenerPrimerNombre() );
277            campoTextoApellidoPaterno.setText( entradaActual.obtenerApellidoPaterno() );
278            campoTextoEmail.setText( entradaActual.obtenerEmail() );
279            campoTextoTelefono.setText( entradaActual.obtenerNumeroTelefonico() );
280            campoTextoMax.setText( "" + numeroDeEntradas );
281            campoTextoIndice.setText( "" + ( indiceEntradaActual + 1 ) );
282            botonSiguiente.setEnabled( true );
283            botonAnterior.setEnabled( true );
284         } // fin de if
285         else
286            botonExplorarActionPerformed( evt );
287      } // fin del método botonConsultaActionPerformed
288
289      // maneja la llamada cuando se introduce un nuevo valor en campoTextoIndice
290      private void campoTextoIndiceActionPerformed( ActionEvent evt )
291      {
292         indiceEntradaActual =
293            ( Integer.parseInt( campoTextoIndice.getText() ) - 1 );
294
295         if ( numeroDeEntradas != 0 && indiceEntradaActual < numeroDeEntradas )
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 5 de 7).

```
296         {
297            entradaActual = resultados.get( indiceEntradaActual );
298            campoTextoID.setText("" + entradaActual.obtenerIDDireccion() );
299            campoTextoPrimerNombre.setText( entradaActual.obtenerPrimerNombre() );
300            campoTextoApellidoPaterno.setText( entradaActual.obtenerApellidoPaterno() );
301            campoTextoEmail.setText( entradaActual.obtenerEmail() );
302            campoTextoTelefono.setText( entradaActual.obtenerNumeroTelefonico() );
303            campoTextoMax.setText( "" + numeroDeEntradas );
304            campoTextoIndice.setText( "" + ( indiceEntradaActual + 1 ) );
305         } // fin de if
306      } // fin del método campoTextoIndiceActionPerformed
307
308      // maneja la llamada cuando se hace clic en botonExplorar
309      private void botonExplorarActionPerformed( ActionEvent evt )
310      {
311         try
312         {
313            resultados = consultasPersona.obtenerTodasLasPersonas();
314            numeroDeEntradas = resultados.size();
315
316            if ( numeroDeEntradas != 0 )
317            {
318               indiceEntradaActual = 0;
319               entradaActual = resultados.get( indiceEntradaActual );
320               campoTextoID.setText( "" + entradaActual.obtenerIDDireccion() );
321               campoTextoPrimerNombre.setText( entradaActual.obtenerPrimerNombre() );
322               campoTextoApellidoPaterno.setText( entradaActual.obtenerApellidoPaterno() );
323               campoTextoEmail.setText( entradaActual.obtenerEmail() );
324               campoTextoTelefono.setText( entradaActual.obtenerNumeroTelefonico() );
325               campoTextoMax.setText( "" + numeroDeEntradas );
326               campoTextoIndice.setText( "" + ( indiceEntradaActual + 1 ) );
327               botonSiguiente.setEnabled( true );
328               botonAnterior.setEnabled( true );
329            } // fin de if
330         } // fin de try
331         catch ( Exception e )
332         {
333            e.printStackTrace();
334         } // fin de catch
335      } // fin del método botonExplorarActionPerformed
336
337      // maneja la llamada cuando se hace clic en botonInsertar
338      private void botonInsertarActionPerformed( ActionEvent evt )
339      {
340         int resultado = consultasPersona.agregarPersona( campoTextoPrimerNombre.getText(),
341            campoTextoApellidoPaterno.getText(), campoTextoEmail.getText(),
342            campoTextoTelefono.getText() );
343
344         if ( resultado == 1 )
345            JOptionPane.showMessageDialog( this, "Se agrego persona!",
346               "Se agrego persona", JOptionPane.PLAIN_MESSAGE );
347         else
348            JOptionPane.showMessageDialog( this, "No se agrego persona!",
349               "Error", JOptionPane.PLAIN_MESSAGE );
350
351         botonExplorarActionPerformed( evt );
352      } // fin del método botonInsertarActionPerformed
353
354      // método main
```

Figura 25.32 | Una libreta de direcciones simple. (Parte 6 de 7).

```
355     public static void main( String args[] )
356     {
357        new MostrarLibretaDirecciones();
358     } // fin del método main
359  } // fin de la clase MostrarLibretaDireccione
```

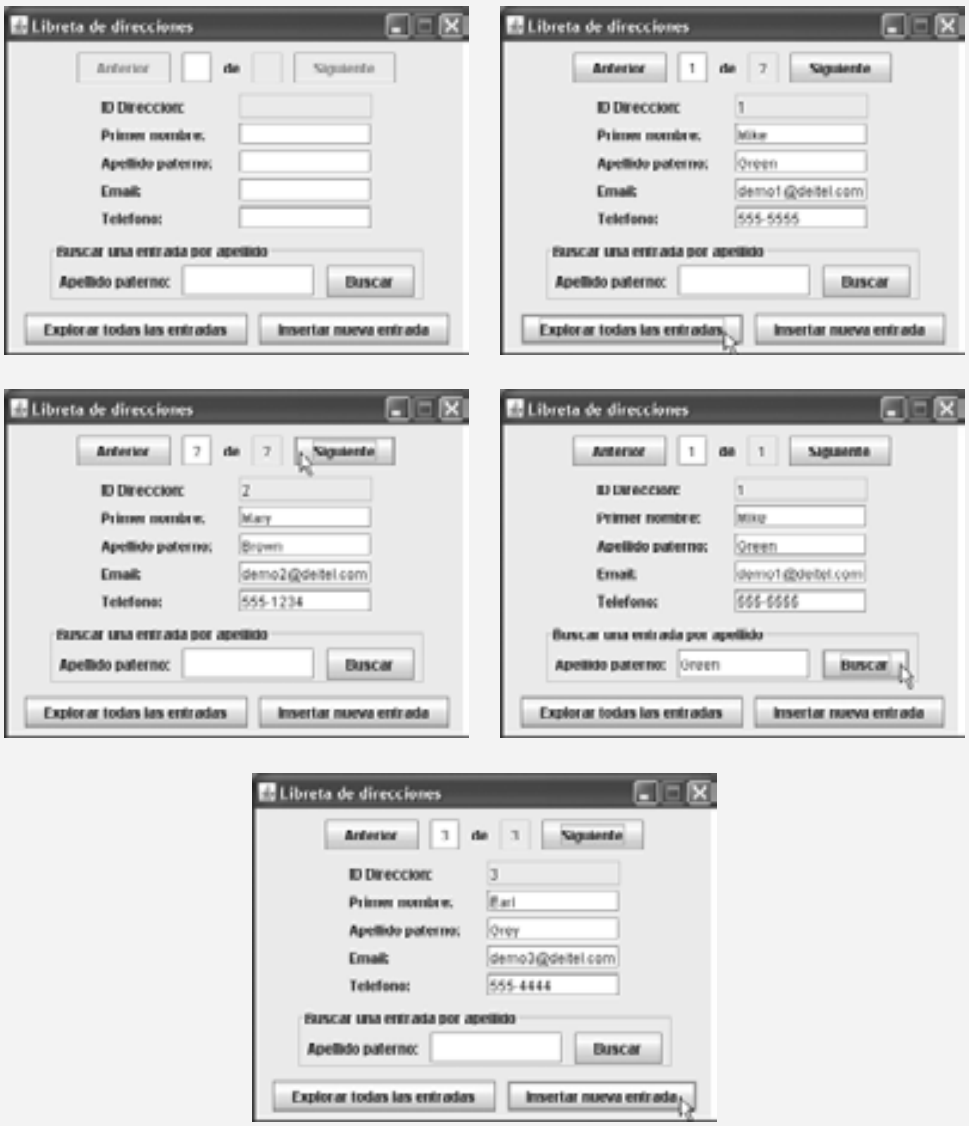

Figura 25.32 | Una libreta de direcciones simple. (Parte 7 de 7).

`teActionPerformed` (líneas 253 a 262), respectivamente. De manera alternativa, el usuario puede escribir un número en el objeto `campoTextoIndice` y oprimir *Intro* para ver una entrada específica.

25.12 Procedimientos almacenados

Muchos sistemas de administración de bases de datos pueden almacenar instrucciones de SQL individuales o conjuntos de instrucciones de SQL en una base de datos, para que los programas que tengan acceso a esa base de datos puedan invocar esas instrucciones. A dichas instrucciones de SQL se les conoce como **procedimientos**

almacenados. JDBC permite a los programas invocar procedimientos almacenados mediante el uso de objetos que implementen a la interfaz `CallableStatement`. Los objetos `CallableStatement` pueden recibir argumentos especificados con los métodos heredados de la interfaz `PreparedStatement`. Además, los objetos `CallableStatement` pueden especificar **parámetros de salida**, en los cuales un procedimiento almacenado puede colocar valores de retorno. La interfaz `CallableStatement` incluye métodos para especificar cuáles parámetros en un procedimiento almacenado son parámetros de salida. La interfaz también incluye métodos para obtener los valores de los parámetros de salida devueltos de un procedimiento almacenado.

Tip de portabilidad 25.6

Aunque la sintaxis para crear procedimientos almacenados difiere entre los diversos sistemas de administración de bases de datos, la interfaz `CallableStatement` proporciona una interfaz uniforme para especificar los parámetros de entrada y salida de los procedimientos almacenados, así como para invocarlos.

Tip de portabilidad 25.7

De acuerdo con la documentación de la API para la interfaz `CallableStatement`, para lograr una máxima portabilidad entre los sistemas de bases de datos, los programas deben procesar las cuentas de actualización o los objetos `ResultSet` devueltos de un objeto `CallableStatement` antes de obtener los valores de cualquier parámetro de salida.

25.13 Procesamiento de transacciones

Muchas aplicaciones de bases de datos requieren garantías de que una serie de operaciones de inserción, actualización y eliminación se ejecuten de manera apropiada, antes de que la aplicación continúe procesando la siguiente operación en la base de datos. Por ejemplo, al transferir dinero por medios electrónicos entre dos cuentas de banco, varios factores determinan si la transacción fue exitosa. Empezamos por especificar la cuenta de origen y el monto que deseamos transferir de esa cuenta hacia una cuenta de destino. Después, especificamos la cuenta de destino. El banco comprueba la cuenta de origen para determinar si hay suficientes fondos en ella como para poder completar la transferencia. De ser así, el banco retira el monto especificado de la cuenta de origen y, si todo sale bien, deposita el dinero en la cuenta de destino para completar la transferencia. ¿Qué ocurre si la transferencia falla después de que el banco retira el dinero de la cuenta de origen? En un sistema bancario apropiado, el banco vuelve a depositar el dinero en la cuenta de origen. ¿Cómo se sentiría usted si el dinero se restara de su cuenta de origen y el banco *no* depositara el dinero en la cuenta de destino?

El **procesamiento de transacciones** permite a un programa que interactúa con una base de datos tratar una operación en la base de datos (o un conjunto de operaciones) como una sola operación. Dicha operación también se conoce como **operación atómica** o **transacción**. Al final de una transacción, se puede tomar una de dos decisiones: **confirmar (commit) la transacción** o **rechazar (roll back) la transacción**. Al confirmar la transacción se finaliza(n) la(s) operación(es) de la base de datos; todas las inserciones, actualizaciones y eliminaciones que se llevaron a cabo como parte de la transacción no pueden invertirse sin tener que realizar una nueva operación en la base de datos. Al rechazar la transacción, la base de datos queda en el estado anterior a la operación. Esto es útil cuando una parte de una transacción no se completa en forma apropiada. En nuestra discusión acerca de la transferencia entre cuentas bancarias, la transacción se rechazaría si el depósito no pudiera realizarse en la cuenta de destino.

Java ofrece el procesamiento de transacciones a través de varios métodos de la interfaz `Connection`. El método `setAutoCommit` especifica si cada instrucción SQL se confirma una vez completada (un argumento `true`), o si deben agruparse varias instrucciones SQL para formar una transacción (un argumento `false`). Si el argumento para `setAutoCommit` es `false`, el programa debe colocar después de la última instrucción SQL en la transacción una llamada al método `commit` de `Connection` (para confirmar los cambios en la base de datos) o al método `rollback` de `Connection` (para regresar la base de datos al estado anterior a la transacción). La interfaz `Connection` también cuenta con el método `getAutoCommit` para determinar el estado de autoconfirmación para el objeto `Connection`.

25.14 Conclusión

En este capítulo aprendió acerca de los conceptos básicos de las bases de datos, cómo interactuar con los datos en una base de datos mediante el uso de SQL y cómo usar JDBC para permitir que las aplicaciones de Java manipu-

len bases de datos MySQL y Java DB. Aprendió acerca de los comandos `SELECT`, `INSERT`, `UPDATE` y `DELETE` de SQL, y también acerca de las cláusulas como `WHERE`, `ORDER BY` e `INNER JOIN`. Conoció los pasos explícitos para obtener una conexión (`Connection`) a la base de datos, crear una instrucción (`Statement`) para interactuar con los datos de la base de datos, ejecutar la instrucción y procesar los resultados. Después, vio cómo usar un objeto `RowSet` para simplificar el proceso de conectarse a una base de datos y crear instrucciones. Utilizó instrucciones `PreparedStatement` para crear instrucciones de SQL precompiladas. Aprendió también a crear y configurar bases de datos, tanto en MySQL como en Java DB. También vimos las generalidades acerca de las instrucciones `CallableStatement` y el procesamiento de transacciones. En el siguiente capítulo, aprenderá acerca del desarrollo de aplicaciones Web con JavaServer Faces. Las aplicaciones basadas en Web crean contenido que, por lo general, se muestra en los clientes exploradores Web. Como veremos en el capítulo 27, las aplicaciones Web también pueden usar la API JDBC para acceder a las bases de datos y crear contenido Web más dinámico.

25.15 Recursos Web y lecturas recomendadas

`java.sun.com/products/jdbc`
La página inicial sobre JDBC de Sun Microsystems, Inc.

`java.sun.com/docs/books/tutorial/jdbc/index.html`
La trayectoria del *Tutorial de Java sobre JDBC*.

`industry.java.sun.com/products/jdbc/drivers`
El motor de búsqueda de Sun Microsystems para localizar controladores de JDBC.

`www.sql.org`
Este portal de SQL proporciona vínculos hacia muchos recursos, incluyendo la sintaxis de SQL, tips, tutoriales, libros, revistas, grupos de discusión, compañías con servicios de SQL, consultores de SQL y software gratuito.

`www.datadirect.com/developer/jdbc/topics/perfoptjdbc/index.ssp`
Informe que describe cómo diseñar una buena aplicación de JDBC.

`java.sun.com/javase/6/docs/technotes/guides/jdbc/index.html`
La documentación de la API JDBC de Sun Microsystems, Inc.

`java.sun.com/products/jdbc/faq.html`
Las preguntas frecuentes acerca de JDBC de Sun Microsystems, Inc.

`www.jguru.com/faq/JDBC`
Las FAQs sobre JDBC de JGuru.

`www.mysql.com`
Este sitio es la página inicial de la base de datos MySQL. Puede descargar las versiones más recientes de MySQL y MySQL Connector/J, y acceder a su documentación en línea.

`www.mysql.com/products/enterprise/server.html`
Introducción al servidor de bases de datos MySQL y vínculos a sus sitios de documentación y descargas.

`dev.mysql.com/doc/mysql/en/index.html`
`dev.mysql.com/doc/refman/5.0/es/index.html`
Manual de referencia de MySQL, en inglés y en español.

`dev.mysql.com/doc/refman/5.1/en/connector-mxj.html`
Documentación sobre MySQL Connector/J, incluyendo instrucciones de instalación y ejemplos.

`developers.sun.com/prodtech/javadb/reference/docs/10.2.1.6/devguide/index.html`
La *Guía para el desarrollador de Java DB*.

`java.sun.com/javase/6/docs/technotes/guides/jdbc/getstart/rowsetImpl.html`
Presenta las generalidades acerca de la interfaz `RowSet` y sus subinterfaces. Este sitio también habla sobre las implementaciones de referencia de estas interfaces de Sun y su uso.

`developer.java.sun.com/developer/Books/JDBCTutorial/chapter5.html`
El capítulo 5 (tutorial de `RowSet`) del libro *The JDBC 2.0 API Tutorial and Reference, Segunda edición*.

Lecturas recomendadas

Ashmore, D. C., "Best Practices for JDBC Programming", *Java Developers Journal*, 5: núm. 4 (2000), 42 a 54.

Blaha, M. R., W. J. Premerlani y J. E. Rumbaugh, "Relational Database Design Using an Object-Oriented Methodology", *Communications of the ACM*, 31: núm. 4 (1988): 414 a 427.

Brunner, R. J., "The Evolution of Connecting", *Java Developers Journal*, 5: núm. 10 (2000): 24 a 26.