



Guía del desarrollador de aplicaciones de base de datos



VERSIÓN 8

Borland®
JBuilder®

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249
www.borland.com

En el archivo `deploy.html` ubicado en el directorio raíz del producto JBuilder encontrará una lista completa de archivos que se pueden distribuir de acuerdo con la licencia de JBuilder y la limitación de responsabilidad.

Borland Software Corporation puede tener patentes concedidas o en tramitación sobre los temas tratados en este documento. Diríjase al CD del producto o al cuadro de diálogo Acerca de para la lista de patentes. La modificación de este documento no le otorga derechos sobre las licencias de estas patentes.

COPYRIGHT © 1997-2002 Borland Software Corporation. Reservados todos los derechos. Todos los nombres de productos y marcas de Borland son marcas comerciales o registradas de Borland Software Corporation en Estados Unidos y otros países. Las otras marcas pertenecen de sus respectivos propietarios.

Si desea más información acerca de las condiciones de contrato de terceras partes y acerca de la limitación de responsabilidades, consulte las notas de esta versión en su CD de instalación de JBuilder.

Impreso en EE.UU.

JBE0080WW21002database 7E10R1002

0203040506-9 8 7 6 5 4 3 2 1

PDF

Índice de materias

Capítulo 1

Introducción

1-1

Resúmenes de los capítulos	1-2
Tutoriales de base de datos	1-4
Ejemplos de base de datos	1-5
Documentación relacionada	1-6
Convenciones de la documentación	1-8
Asistencia a los desarrolladores	1-10
Cómo ponerse en contacto con el servicio técnico de Borland	1-10
Recursos en línea	1-11
World Wide Web	1-11
Grupos de noticias de Borland	1-11
Usenet, grupos de noticias	1-12
Información sobre errores	1-12

Capítulo 2

Aplicaciones de base de datos

JBuilder

2-1

Arquitectura de las aplicaciones de base de datos	2-1
Componentes DataExpress	2-2
Principales funciones y ventajas	2-3
Descripción general de los componentes DataExpress	2-5
DataExpress para componentes EJB	2-10
InternetBeans Express	2-11
Componentes de bases de datos XML	2-11
dbSwing	2-11
Los módulos de datos y el modelador de datos	2-13
Explorador de bases de datos	2-13
Monitor JDBC	2-13
JDataStore y JBuilder	2-14
Diferencias entre el uso de los controladores JDataStore y JDBC	2-14
Ventajas adicionales de un JDataStore	2-15
El Explorador de JDataStore	2-16
Operaciones del explorador de JDataStore	2-16
InterBase y JBuilder	2-17

Capítulo 3

Importación y exportación de datos desde un archivo de texto

3-1

Incorporación de columnas a un TableDataSet en el editor	3-2
Importación de datos con formato desde un archivo de texto	3-3
Extracción de datos de una fuente de datos JDBC	3-3
Exportación de datos	3-4
Exportación de datos de TableDataSet a un archivo de texto	3-5
Almacenamiento de cambios desde un TableDataSet en una tabla SQL	3-5
Guardar los cambios cargados de un TextDataFile en una fuente de datos JDBC	3-5

Capítulo 4

Conexión con bases de datos

4-1

Conexión con bases de datos	4-2
Incorporación de componentes Data Access a la aplicación	4-3
Asignación de valores a las propiedades de conexión de una base de datos	4-4
Configuración de JDataStore	4-7
Configuración de InterBase e InterClient	4-7
Uso de InterBase e InterClient con JBuilder	4-8
Sugerencias para la utilización de las tablas de ejemplo de InterBase	4-9
Adición de un controlador JDBC a JBuilder	4-10
Creación de los archivos .library y .config	4-10
Adición del controlador JDBC a proyectos	4-11
Conexión con una base de datos mediante los controladores JDBC de InterClient	4-14
Utilización de componentes Database en las aplicaciones	4-16
Solicitud del nombre de usuario y la contraseña	4-17
Agrupación de conexiones JDBC	4-17
Optimización del rendimiento de JConnectionPool	4-20
Impresión del histórico de compresión	4-20
Ejemplo de agrupación	4-20
Resolución de problemas en conexiones de JDataStore e InterBase	4-23

Mensajes de error de conexión habituales . . . 4-23

Capítulo 5

Recuperación de datos de una fuente de datos 5-1

Consultas en bases de datos	5-2
Asignación de propiedades en el cuadro de diálogo de consulta.	5-4
Ficha Consulta	5-4
La ficha Parámetros.	5-6
Cómo colocar el texto SQL en un conjunto de recursos	5-7
Consultas en bases de datos: Sugerencias . . .	5-9
Cómo mejorar el rendimiento del DataSet	5-10
Metadatos persistentes de consulta.	5-11
Apertura y cierre de conjuntos de datos	5-12
Verificación de que una consulta es actualizable	5-12
Utilización de consultas parametrizadas para obtener datos	5-13
Parametrizar una consulta	5-13
Creación de la aplicación.	5-14
Cómo añadir filas de parámetros	5-14
Cómo añadir objetos QueryDataSet	5-15
Cómo añadir los componentes de la interfaz	5-16
Consultas parametrizadas: Sugerencias . . .	5-19
Utilización de parámetros	5-19
Ejecución de la consulta parametrizada con nuevos parámetros.	5-21
Consultas parametrizadas en relaciones maestro-detalle	5-22

Capítulo 6

Utilización de procedimientos almacenados 6-1

Procedimientos almacenados: sugerencias . . .	6-3
Secuencias de escape, sentencias SQL y llamadas a procedimientos específicos de servidor	6-3
Creación manual de tablas y procedimientos para el tutorial	6-4
Utilización de procedimientos específicos del fabricante	6-6

Utilización de los procedimientos almacenados de JdataStore y de las funciones definidas por el usuario.	6-6
Utilización de procedimientos almacenados de InterBase	6-7
Utilización de parámetros con procedimientos almacenados de Oracle PL/SQL	6-7
Utilización de procedimientos almacenados de Sybase	6-9
Aplicación de ejemplo con procedimientos almacenados específicos del servidor de base de datos	6-9
Creación de un proveedor de datos personalizado	6-9
Obtención de metadatos	6-10
Llamada a initData	6-11
Obtención de datos reales	6-12
Consejos para diseñar un proveedor de datos personalizado	6-12
Cómo funciona el método provideData() en un conjunto de datos maestro-detalle	6-12

Capítulo 7

Utilización de columnas 7-1

Propiedades de columnas y metadatos.	7-1
Propiedades Column distintas de metadatos	7-2
Visualización de información de columnas en el diseñador de columnas	7-2
El botón Generar clase RowIterator	7-3
Utilización del diseñador de columnas para convertir metadatos en persistentes	7-4
Cómo convertir metadatos en dinámicos con el Diseñador de columnas	7-5
Ver información de columnas en el Explorador de base de datos.	7-5
Optimización de una consulta.	7-7
Configuración de las propiedades de columna.	7-7
Configuración de las propiedades de columna mediante las herramientas de diseño visual de JBuilder	7-7
Definición de propiedades en el código.	7-7
Columnas persistentes	7-8
Combinación de metadatos dinámicos con columnas persistentes.	7-9
Eliminación de columnas persistentes	7-9

Utilización de columnas persistentes para añadir columnas vacías a un DataSet	7-10
Control del orden de columnas en un DataSet.	7-11

Capítulo 8

Almacenamiento de cambios en la fuente de datos 8-1

Almacenamiento de cambios desde un QueryDataSet.	8-3
Cómo añadir un botón para guardar los cambios de un QueryDataSet	8-3
Almacenamiento de cambios en las fuentes de datos con procedimientos almacenados	8-5
Guardar cambios por medio de QueryResolver	8-6
Codificación de procedimientos almacenados para gestionar el almacenamiento	8-8
Guardar cambios con un ProcedureResolver	8-9
Ejemplo: Utilización de procedimientos almacenados de InterBase con parámetros de devolución.	8-11
Almacenamiento de datos de varias tablas . .	8-12
Consideraciones sobre el tipo de vinculación entre tablas durante la consulta	8-12
Referencias a tablas y columnas (alias) en una cadena de consulta.	8-13
Control de la configuración de las propiedades de Column	8-14
¿Qué pasa cuando una tabla no se puede actualizar?	8-14
¿Cómo especificar que nunca debe actualizarse una tabla?	8-14
Utilización de conjuntos de datos con RMI (conjuntos de datos transportables)	8-15
Ejemplo: Utilización de conjuntos de datos transportables	8-15
Utilización de métodos DataSet transportables.	8-16
Personalización de la lógica del almacenador por defecto	8-17
Cómo funciona el almacenador por defecto	8-18
Incorporación de un componente QueryResolver	8-18

Intercepción de sucesos del almacenador	8-19
Utilización de sucesos de almacenador .	8-21
Cómo crear un almacenador de datos personalizado	8-21
Tratamiento de errores del almacenador	8-22
Almacenar relaciones maestro-detalle. .	8-23

Capítulo 9

Establecimiento de una relación maestro-detalle 9-1

Definición de una relación maestro-detalle . .	9-2
Creación de una aplicación con una relación maestro-detalle.	9-3
Captura de detalles	9-8
Captura de todos los detalles de una vez. .	9-8
Obtención de registros detalle a petición. .	9-8
Edición de datos en conjuntos de datos maestro-detalle	9-10
Pasos en la creación de una relación maestro-detalle	9-10
Almacenamiento de los cambios en una relación maestro-detalle.	9-11
Resolución de conjuntos de datos maestro-detalle en fuentes de datos JDBC	9-12

Capítulo 10

Utilización de módulos de datos para simplificar el acceso a los datos 10-1

Creación de un módulo de datos con las herramientas de diseño.	10-2
Creación del módulo de datos con el asistente.	10-2
Adición de componentes de datos al módulo de datos.	10-3
Cómo añadir la lógica empresarial al módulo de datos.	10-5
Utilización de un módulo de datos	10-5
Incorporación de una biblioteca necesaria a un proyecto	10-6
Hacer referencia a un módulo de datos en una aplicación	10-7
Cuadro de diálogo Usar módulo de datos	10-8
Creación de módulos de datos con el modelador de datos.	10-9

Creación de consultas con el modelador de datos	10-10	Creación de consultas	12-2
Apertura de URL	10-11	Introducción de datos con listas de selección	12-2
Inicio de una consulta	10-11	Cómo añadir un campo de lista de selección.	12-3
Adición de la cláusula Group by	10-14	Eliminación de un campo de lista de selección.	12-4
Selección de filas con valores de columna unívocos.	10-15	Creación de consultas mediante columnas calculadas.	12-5
Inclusión de una cláusula WHERE	10-15	Utilización de columnas calculadas.	12-8
Inclusión de una cláusula ORDER BY	10-16	Creación de una columna calculada en el diseñador.	12-9
Edición directa de la consulta	10-17	Totalización de datos con campos calculados	12-11
Comprobación de consultas	10-17	Ejemplo: Totalización de datos con campos calculados	12-12
Generación de consultas múltiples	10-17	Asignación de valores a las propiedades de AggDescriptor	12-15
Especificación de una relación maestro-detalle	10-17	Creación de un manejador personalizado de sucesos de totalización	12-16
Almacenamiento de consultas.	10-19	Incorporación de una plantilla de edición o visualización para formatear datos	12-17
Generación de aplicaciones de base de datos	10-20	Máscaras de visualización	12-18
Utilización de módulos de datos generados en el código	10-21	Máscaras de edición.	12-19
		Utilización de máscaras en la importación y exportación de datos	12-19
Capítulo 11		Modelos dependientes del tipo de datos	12-20
Filtrado, clasificación y localización de datos	11-1	Modelos para datos numéricos.	12-20
Suministro de datos de los ejemplos.	11-2	Modelos para datos de fecha y hora	12-21
Filtrado de datos	11-5	Modelos para datos de cadena	12-22
Añadir y eliminar filtros.	11-6	Modelos para datos booleanos	12-23
Clasificación de datos	11-9	Presentación de una vista alternativa de los datos	12-24
Clasificación de datos en una JdbTable	11-10	Persistencia de los datos	12-26
Clasificación de datos mediante las herramientas de diseño visual de JBuilder	11-11	Conversión de las columnas en persistentes.	12-26
Clasificación e indexación	11-12	Utilización de tipos de datos variantes	12-28
Clasificación de datos en el código	11-14	Almacenamiento de objetos Java	12-28
Localización de datos	11-15		
Búsqueda de datos con JdbNavField.	11-15	Capítulo 13	
Localización de datos mediante la escritura de código	11-17	Otros controles y sucesos	13-1
Localización de datos mediante una DataRow.	11-18	Sincronización de componentes visuales.	13-1
Opciones de localización	11-19	Acceso a la información de modelo y datos de un componente de la interfaz de usuario	13-2
Localización para gestionar cualquier tipo de datos	11-20	Visualización de la información de estado.	13-3
Orden de columnas en DataRow y en DataSet.	11-21	Creación de aplicaciones con componentes JdbStatusLabel	13-3
		Ejecución de la aplicación JdbStatusLabel	13-4
Capítulo 12			
Cómo añadir funcionalidad a las aplicaciones de base de datos	12-1		

Gestión de errores y excepciones.	13-5
Redefinición de la gestión de los controles por defecto de DataSetException.	13-6

Capítulo 14

Creación de una aplicación de base de datos distribuida con DataSetData 14-1

Ejemplo de aplicación de base de datos distribuida (con RMI Java y DataSetData). . .	14-2
Configuración del ejemplo de aplicación . .	14-3
¿Qué ocurre?	14-4
Paso de metadatos mediante DataSetData	14-4
Distribución de aplicaciones en tres niveles.	14-5

Capítulo 15

Tareas de administración de bases de datos 15-1

Explorar tablas de bases de datos y metadatos con el Explorador de bases de datos	15-1
Inspección de objetos esquema de una base de datos	15-2
Configuración de controladores para acceder a bases de datos locales y remotas	15-3
Ejecución de sentencias SQL	15-4
Utilización del Explorador para ver y editar los datos de la tabla.	15-6
Uso del Explorador de bases de datos para las tareas de administración	15-8
Creación de la fuente de datos SQL	15-8
Alimentación de una tabla SQL con datos mediante JBuilder	15-10
Eliminación de tablas en JBuilder	15-10
Seguimiento de conexiones de base de datos. .	15-11
Acercar del Monitor JDBC	15-11
Utilización del Monitor JDBC en una aplicación en ejecución.	15-12
Incorporación de MonitorButton a la paleta	15-12
Utilización de la clase MonitorButton en el código	15-13
Propiedades MonitorButton.	15-13

Capítulo 16

Tutorial: Importación y exportación de datos desde un archivo de texto 16-1

Paso 1: Creación de un proyecto	16-2
Paso 2: Crear el archivo de texto	16-3
Paso 3: Generación de una aplicación	16-3
Paso 4: Incorporación de componentes DataExpress a la aplicación	16-4
Paso 5: Adición de componentes dbSwing para crear una interfaz de usuario	16-6
Paso 6: Adición de un componente Swing JButton	16-8
Paso 7: Compilación y ejecución de la aplicación	16-10
Paso 8: Uso de modelos en la exportación de campos numéricos de fecha, hora y texto .	16-11

Capítulo 17

Tutorial: Creación de aplicaciones de base de datos distribuidas 17-1

Paso 1: Creación de un proyecto	17-3
Paso 2: Generación de una aplicación	17-4
Paso 3: Incorporación de componentes DataExpress a la aplicación	17-4
Paso 4: Diseño de las columnas de la aplicación	17-7
Añadir columnas y modificar propiedades de columnas	17-7
Especificación de cálculos para las columnas calculadas	17-9
Paso 5: Adición de componentes dbSwing para crear una interfaz de usuario	17-10
Paso 6: Totalización de datos con campos calculados	17-14

Capítulo 18

Tutorial: Recuperación de datos mediante procedimientos almacenados 18-1

Paso 1: Creación de tablas y procedimientos para el tutorial.	18-2
Paso 2: Incorporación de los componentes DataSet	18-2
Paso 3: Cómo añadir componentes visuales .	18-4

Índice I-1

Figuras

2.1	Diagrama de una aplicación de base de datos	2-2	11.4	Cuadro de diálogo Sort	11-12
2.2	Explorador de JDataStore	2-16	11.5	Aplicación ordenada durante la ejecución	11-12
4.1	Componente Database abierto en el panel de contenido	4-4	11.6	Aplicación de ejemplo con JdbNavField	11-15
4.2	Cuadro de diálogo Descriptor de conexiones	4-6	12.1	Aplicación de consultas	12-8
4.3	Cuadro de diálogo Conexión	4-15	12.2	Columnas calculadas	12-11
5.1	Editor de la propiedad Query	5-4	12.3	Diseñador de columnas	12-27
5.2	Ficha de Parámetros	5-7	15.1	Explorador de bases de datos	15-2
5.3	Cuadro de diálogo de recursos extraídos	5-7	15.2	Ficha Introducir SQL del Explorador de bases de datos	15-5
8.1	Interfaz de usuario para guardar cambios desde un QueryDataSet	8-4	15.3	Monitor JDBC	15-11
10.1	Modelador de datos	10-10	16.1	Aplicación de importación y exportación de base de datos	16-2
10.2	Selección de columnas	10-12	16.2	Aplicación Importar/Exportar en ejecución	16-8
10.3	Cuadro de diálogo Totalizar.	10-13	16.3	Aplicación de importación y exportación de base de datos con JButton	16-9
10.4	Ficha Group By	10-14	16.4	Aplicación de exportación de datos a un archivo de texto en ejecución	16-10
10.5	Ficha Where	10-15	17.1	Aplicación básica de base de datos	17-3
10.6	Ficha Order by	10-16	17.2	Cuadro de diálogo de la consulta	17-6
10.7	Cuadro de diálogo Enlazar consultas.	10-18	17.3	Nodo queryDataSet1 expandido	17-7
10.8	Flecha que muestra la relación entre columnas.	10-19	17.4	Columnas queryDataSet1 en el diseñador de columnas	17-9
10.9	El editor muestra el código generado por el modelador de datos	10-20	17.5	Componente JdbTable en el diseñador de interfaces de usuario.	17-11
10.10	Asistente para aplicaciones módulo de datos	10-21	17.6	Aplicación básica de base de datos con barra de navegación y etiqueta de estado	17-13
11.1	Ejecución de la aplicación de base de datos	11-5	17.7	Cuadro de diálogo Agg	17-15
11.2	Filtros de ejecución de aplicación	11-9	18.1	Recuperación de datos con la aplicación de procedimientos almacenados en ejecución	18-5
11.3	Haga clic en la cabecera de una columna para ordenar durante la ejecución	11-10			

Introducción

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

En *Desarrollo de aplicaciones de base de datos* se facilita información sobre el uso de las funciones de base de datos de DataExpress de JBuilder para el desarrollo de aplicaciones de base de datos. También se describe cómo utilizar componentes dbSwing para crear una interfaz de usuario (IU) para la aplicación. Se explican, mediante ejemplos, las características básicas incluidas normalmente en una aplicación de bases de datos, para facilitar así el aprendizaje práctico. A la información conceptual siguen inmediatamente ejemplos pertinentes, con referencias cruzadas para obtener información más detallada siempre que sea posible.

Visite la dirección <http://www.borland.com/techpubs/jbuilder> para obtener las actualizaciones y adiciones a la documentación. Consulte también la ayuda en línea de JBuilder. La información del sistema de ayuda es más reciente que la de los manuales impresos.

Si tiene alguna duda sobre cómo crear aplicaciones de base de datos a través de JBuilder, visite el grupo de noticias sobre bases de datos en [news://newsgroups.borland.com/borland.public.jbuilder.database](http://newsgroups.borland.com/borland.public.jbuilder.database). Este grupo de noticias, activamente supervisado por nuestros ingenieros del servicio técnico y por el equipo de desarrollo de JBuilder, se centra en temas relacionados con la escritura de aplicaciones de bases de datos en JBuilder. Si desea información sobre los componentes de dbSwing en el grupo de noticias [borland.public.jbuilder.dbSwing](http://newsgroups.borland.com/borland.public.jbuilder.dbSwing) puede obtener ayuda para la creación de interfaces de usuario de aplicaciones de base de datos. Puede encontrar una base de datos FAQ de DataExpress en la página web de la Comunidad Borland, en <http://community.borland.com/>.

Nota Todas las versiones de JBuilder proporcionan acceso directo a datos SQL mediante la API JDBC de Sun. JBuilder Enterprise proporciona componentes DataExpress adicionales que simplifican en gran medida el desarrollo de aplicaciones de base de datos, tal y como se describe en este

libro. Se puede acceder a muchos de estos componentes desde la pestaña DataExpress de la paleta de componentes.

DataExpress almacena los datos en memoria. La mayoría de las aplicaciones de ejemplo y tutoriales descritos en esta guía utilizan datos de ejemplo almacenados en un JDataStore y se accede a ellos por medio de un controlador JDBC. La sustitución de plug-ins de JDataStore para el almacenamiento en memoria proporciona un almacenamiento permanente de datos. JDataStore se puede gestionar como cualquier base de datos SQL: es posible conectarse con el componente como con cualquier otro servidor, ejecutar consultas SQL, etc. Si desea más información sobre JDataStore, consulte la *Guía del desarrollador de JDataStore*.

Si desea información sobre las convenciones de la documentación, consulte [“Convenciones de la documentación” en la página 1-8](#).

Si no está familiarizado con JBuilder, le sugerimos que comience con *Introducción a JBuilder*. Si no está familiarizado con Java, le sugerimos que comience con *Procedimientos iniciales con Java*.

Resúmenes de los capítulos

Este libro detalla cómo aparecen las tecnologías y las herramientas de base de datos en JBuilder y cómo trabajar con ellas en la IDE y en el editor. También explica cómo estas tecnologías se complementan entre sí en una aplicación de base de datos. Si desea obtener más información, seleccione uno de los siguientes temas:

- [Capítulo 2, “Aplicaciones de base de datos JBuilder”](#)

Se hace una introducción acerca de las tecnologías, componentes y herramientas que se utilizan para crear aplicaciones de base de datos en JBuilder, entre las que se incluyen elementos de la biblioteca de componentes de DataExpress, el explorador de bases de datos, el monitor JDBC, fuentes de datos, JDataStore e InterBase.

- [Capítulo 3, “Importación y exportación de datos desde un archivo de texto”](#)

Explica la forma de suministrar datos a la aplicación desde un archivo de texto y guardar los datos en un archivo de texto o en una fuente de datos SQL.

- [Capítulo 4, “Conexión con bases de datos”](#)

Describe cómo conectar los componentes de la base de datos con un servidor. Incluye información para la utilización de los controladores de bases de datos JDBC y ODBC e información específica para conectarse a las bases de datos de JDataStore e InterBase.

- **Capítulo 5, “Recuperación de datos de una fuente de datos”**

Describe la forma de crear una copia local de los datos desde la fuente de datos y qué componentes del paquete DataExpress deben utilizarse. Esta fase, llamada *providing* (suministro) pone los datos a disposición de la aplicación.

- **Capítulo 6, “Utilización de procedimientos almacenados”**

Describe cómo crear y utilizar procedimientos almacenados para ejecutar sentencias SQL para proporcionar o almacenar datos.

- **Capítulo 7, “Utilización de columnas”**

Explica cómo hacer persistentes las columnas, cómo controlar el aspecto y la modificación de los datos de las columnas, cómo obtener información de los metadatos, cómo añadir columnas a un conjunto de datos y cómo definir el orden de presentación de las columnas.

- **Capítulo 8, “Almacenamiento de cambios en la fuente de datos”**

Explica cómo guardar las actualizaciones de los datos realizadas en la aplicación JBuilder en una fuente de datos (este proceso se denomina almacenamiento). Trata múltiples métodos para almacenar, incluida la gestión de almacenamiento básico proporcionada por los componentes DataExpress, guardar cambios mediante procedimientos almacenados, almacenar datos de varias tablas, utilizar objetos *DataSet* con RMI y personalizar la lógica por defecto del almacenador.

- **Capítulo 9, “Establecimiento de una relación maestro-detalle”**

Proporciona información sobre cómo vincular dos o más conjuntos de datos para crear una relación maestro-detalle.

- **Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”**

Describe cómo se deben utilizar los módulos de datos para facilitar el acceso a datos de las aplicaciones. También ofrece información sobre la estandarización de la lógica de base de datos y las normas empresariales para todos los desarrolladores que acceden a esos datos. De igual forma, incluye información sobre la utilización del asistente del modelador de datos para crear módulos de datos.

- **Capítulo 11, “Filtrado, clasificación y localización de datos”**

Proporciona información sobre cómo implementar el filtrado, clasificación y localización de datos en aplicaciones de base de datos mediante los componentes estándar de DataExpress y las herramientas de diseño de JBuilder. Explica las diferencias entre estas funciones y se proporcionan tutoriales para todas ellas.

- [Capítulo 12, “Cómo añadir funcionalidad a las aplicaciones de base de datos”](#)

Incluye información acerca de las siguientes tareas:

- Formateo y análisis de datos mediante máscaras de edición o de visualización
 - Creación de columnas calculadas.
 - Datos de totalización (mínimo, máximo, suma, recuento).
 - Creación de campos de consulta.
 - Creación de una vista alternativa de los datos.
 - Creación de campos persistentes, o predefinidos.
- [Capítulo 13, “Otros controles y sucesos”](#)

Trata los distintos métodos adicionales para facilitar el desarrollo de la porción de aplicación correspondiente a la interfaz de usuario. Incluye información acerca de la forma de presentar la información de estado y de gestionar los errores en la aplicación.
 - [Capítulo 14, “Creación de una aplicación de base de datos distribuida con DataSetData”](#)

Trata la utilización de componentes DataExpress en un entorno de programación de objetos distribuidos (con Java RMI).
 - [Capítulo 15, “Tareas de administración de bases de datos”](#)

Proporciona información acerca de las tareas comunes de base de datos, entre las que se incluye:

 - Buscar y modificar datos, tablas y archivos schema de base de datos con el Explorador de bases de datos.
 - Crear y borrar tablas.
 - Rellenar las tablas con datos.
 - Seguir el tráfico de JDBC con el Monitor JDBC.

Tutoriales de base de datos

Los siguientes tutoriales sirven para ilustrar técnicas útiles para el desarrollo de aplicaciones de base de datos.

- [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#)

Muestra cómo utilizar el componente `TableDataSet` para importar y exportar datos desde un archivo de texto. Este tutorial también muestra cómo utilizar componentes `dbSwing` y las herramientas de diseño de `JBuilder` con el fin de construir una interfaz de usuario para la aplicación de base de datos.

- [Capítulo 17, “Tutorial: Creación de aplicaciones de base de datos distribuidas”](#)

Muestra cómo crear una aplicación sencilla que se conecta con una base de datos SQL. Aprenderá a configurar las propiedades de conexión con bases de datos, a añadir campos de búsqueda para localizar datos y a agregar campos calculados a valores totales de una columna.

- [Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados”](#)

Muestra cómo utilizar el componente `ProcedureDataSet` en una aplicación para recuperar datos de una base de datos. En este tutorial, aprenderá a crear procedimientos almacenados y a utilizarlos desde una aplicación.

Ejemplos de base de datos

Existen multitud de ejemplos que muestran diferentes técnicas y tecnologías específicas de aplicaciones de base de datos. La mayor parte de estos ejemplos se pueden encontrar en los siguientes directorios:

- `<jbuilder>/samples/DataExpress`: contiene una amplia variedad de proyectos que muestran técnicas útiles para utilizar componentes DataExpress en el desarrollo de aplicaciones de base de datos.
- `<jbuilder>/samples/dbSwing`: contiene proyectos que ilustran cómo se debe utilizar los componentes dbSwing para crear interfaces de usuario efectivas para las aplicaciones de base de datos.
- `<jbuilder>/samples/JDataStore`: contiene código de ejemplo, archivos de base de datos y proyectos de JBuilder para mostrar cómo se utilizan las bases de datos de JDataStore y los controladores de base de datos de JDataStore con JBuilder. Estos archivos de ejemplo complementan los tutoriales y ejemplos que se tratan en la *Guía del desarrollador de JDataStore*.

Muchas de las aplicaciones acceden a los datos de la base de datos de ejemplo de JDataStore, `employee.jds`, y de la base de datos de ejemplo de InterBase `employee.gdb`. Para obtener más información sobre JDataStore, consulte *Guía del desarrollador de JDataStore*. Si desea más información sobre el servidor InterBase consulte la documentación en línea.

A lo largo de esta guía, se hace referencia a ejemplos concretos para ilustrar mejor conceptos específicos que aparecen en el texto.

Nota Si desea ver las aplicaciones de ejemplo en el diseñador JBuilder, deberá generar el proyecto correspondiente a cada ejemplo antes de llevarlo al diseñador. Para generar un proyecto, seleccione Proyecto | Generar de nuevo el proyecto.

Documentación relacionada

La siguiente documentación de Borland contiene información de gran utilidad para el desarrollo de aplicaciones de base de datos:

- *DataExpress Component Library Reference* es la documentación API en línea para los paquetes DataExpress utilizados para el acceso a bases de datos. Incluye las siguientes referencias a los paquetes de componentes individuales:

- *Referencia de DataExpress:*

Contiene documentación API para los paquetes que proporcionan acceso básico a datos. El paquete `com.borland.dx.dataset` proporciona rutinas generales para la conexión, gestión y manipulación de datos. El paquete `com.borland.dx.sql.dataset` proporciona funciones específicas de JDBC para la conexión a datos. El paquete `com.borland.dx.text` contiene clases e interfaces que controlan la alineación y formato de objetos y el formato de datos y valores. Este paquete también gestiona las excepciones de formato y análisis y la validación de los datos introducidos.

- *Referencia de dbSwing:*

Contiene documentación API para el paquete `com.borland.dbswing`, que, a su vez, contiene componentes que hacen que los componentes Swing sean capaces de acceder a datos de la base de datos a través de los `DataSets` de DataExpress.

- *Referencia de JDataStore:*

Contiene documentación API para los paquetes que se utilizan para conectarse y realizar transacciones con las bases de datos de JDataStore. El paquete `com.borland.datastore` proporciona compatibilidad básica de conectividad y transacción para las bases de datos de JDataStore. El paquete `com.borland.datastore.jdbc` contiene la interfaz JDBC de DataStore, incluido el controlador JDBC en sí mismo, y clases para la implementación de su propio servidor DataStore para conexiones multi-usuario al mismo DataStore. El paquete `com.borland.datastore.javax.sql` proporciona funciones para la admisión de transacciones distribuidas (XA). Las clases de este paquete se utilizan internamente por otras clases de Borland. Nunca se debe utilizar las clases de este paquete directamente.

- *Referencia de las clases Javax:*

Contiene documentación API para el paquete `com.borland.javax.sql`, que proporciona implementación de JDBC 2.0 `DataSource` y componentes de agrupación de conexiones. Estas clases pueden utilizarse con controladores JDBC, pero poseen funciones adicionales específicas de los controladores JDBC de JDataStore.

- *Referencia de InternetBeans Express:*

Contiene documentación API para los paquetes

`com.borland.internetbeans` y `com.borland.internetbeans.taglib` que proporcionan componentes y una biblioteca de etiquetas JSP para crear y responder a la capa de presentación de una aplicación web.

- *Referencia de las clases del Adaptador SQL:*

Contiene documentación API para el paquete `com.borland.sql`. Este paquete contiene la interfaz del `AdaptadorSQL` que puede implementar cualquier clase JDBC que pueda adaptarse para obtener un rendimiento mejorado.

- *Referencia de las clases de las herramientas SQL:*

Contiene documentación API para el paquete `com.borland.sqltools`, que, a su vez, contiene clases para la recuperación de los informes de salida mediante consultas SQL realizadas en formato XML.

- *Referencia de CORBA Express:*

Contiene documentación API para el paquete `com.borland.cx`, que, a su vez, contiene clases de conexión CORBA para aplicaciones distribuidas basadas en CORBA.

- *Referencia de EJB DataExpress:*

Contiene documentación API para paquetes `com.borland.dx.ejb`. Este paquete contiene componentes DataExpress para EJB que posibilitan la utilización de beans entidad con DataSets de DataExpress para almacenar y suministrar datos. Se puede añadir algunos de estos componentes desde la ficha EJB de la paleta de componentes del diseñador de interfaces de usuario.

- *Referencia de los componentes XML de base de datos:*

Contiene documentación API para componentes XML de base de

datos de los paquetes `com.borland.jbuilder.xml.database.xmldbms`,

`com.borland.jbuilder.xml.database.template`, y

`com.borland.jbuilder.xml.database.common`. Se pueden añadir muchos de los componentes de estos paquetes a través de la ficha XML de la paleta de componentes del diseñador de interfaces de usuario.

- *Guía del desarrollador de aplicaciones web* contiene la información sobre la utilización de los componentes InternetBeans Express para crear aplicaciones web para el acceso a datos. La *Guía del desarrollador de aplicaciones web* incluye tutoriales que muestran cómo se han de utilizar los componentes InternetBeans Express con las páginas JSP y los servlets.

- *Guía del desarrollador de aplicaciones XML* explica cómo utilizar los componentes bean basados en modelos y plantillas XML para las consultas de bases de datos y transferencia de datos entre documentos

XML y bases de datos. La *Guía del desarrollador de XML* también incluye tutoriales que instruyen acerca del modo de utilización de los componentes XML de base de datos.

- La *Guía del desarrollador de Enterprise JavaBeans* describe cómo se debe utilizar componentes DataExpress para EJB con el fin de transferir datos desde los beans entidad distribuidos en un servidor a una aplicación cliente y viceversa.
- La *Guía del desarrollador de JDataStore* contiene amplia información de referencia que le servirá de gran ayuda para utilizar JDataStore con las aplicaciones de base de datos que se desarrollen.

Convenciones de la documentación

En la documentación de Borland para JBuilder, el texto con significado especial se identifica mediante la tipografía y los símbolos descritos en la siguiente tabla.

Tabla 1.1 Convenciones tipográficas y de símbolos

Tipo de letra	Significado
Letra monoespaciada	<p>El tipo monoespaciado representa lo siguiente:</p> <ul style="list-style-type: none"> • texto tal y como aparece en la pantalla • cualquier cosa que debe escribir, como “Escriba Hola a todos en el campo Título del Asistente para aplicaciones”. • nombres de archivos • nombres de vías de acceso • nombres de directorios y carpetas • comandos, como <code>SET PATH</code> • código Java • tipos de datos de Java, como <code>boolean</code>, <code>int</code> y <code>long</code>. • los identificadores de Java, como nombres de variables, clases, nombres de paquetes, interfaces, componentes, propiedades, métodos y sucesos. • nombres de argumentos • nombres de campos • palabras clave de Java, como <code>void</code> y <code>static</code>.
Negrita	La negrita se utiliza para las herramientas java, <code>bmj</code> (Borland Make for Java), <code>bcj</code> (Borland Compiler for Java) y opciones del compilador. Por ejemplo: <code>javac</code> , <code>bmj</code> , -vía de acceso a clases .
<i>Cursiva</i>	Las palabras en cursiva indican los términos nuevos que se definen y los títulos de libros; ocasionalmente se usan para indicar énfasis.
<i>Nombres de tecla</i>	Este tipo de letra indica una tecla, como “Pulse <i>Esc</i> para salir de un menú”.

Tabla 1.1 Convenciones tipográficas y de símbolos (continuación)

Tipo de letra	Significado
[]	Los corchetes, en las listas de texto o sintaxis, encierran elementos optativos. En estos casos no se deben escribir los corchetes.
< >	<p>Los corchetes angulares se utilizan para indicar variables en las vías de directorios, opciones de comando y ejemplos de código. Por ejemplo, <nombredearchivo> puede utilizarse para indicar dónde tiene que incluir el nombre de un archivo (incluida la extensión) y <usuario> indica normalmente que debe indicar su nombre de usuario.</p> <p>Cuando se sustituyen las variables en las vías de acceso a los directorios, comandos y ejemplos de código, se sustituye la variable completa, incluidos los corchetes (< >). Por ejemplo, reemplazaría <nombredearchivo> con el nombre de un archivo, como <code>employee.jds</code> y omitiría los corchetes.</p> <p>Nota: Los corchetes se utilizan en HTML, XML, JSP y otros archivos basados en etiquetas para demarcar los elementos del documento, como <color de fuente=red> y <ejb-jar>. Las siguientes convenciones describen cómo se especifican las cadenas de variables dentro del ejemplo de código que ya está utilizando corchetes como delimitadores.</p>
<i>Cursiva, serif</i>	Este formato se utiliza para indicar las cadenas de variables en los ejemplos de código que ya están usando corchetes como delimitadores. Por ejemplo, <url="jdbc:borland:jbuilder\samples\guestbook.jds">
...	En los ejemplos de código, los puntos suspensivos (...) indican código que se ha omitido en el ejemplo para ahorrar espacio y aumentar la claridad. Si están en un botón, los puntos suspensivos indican que éste conduce a un cuadro de diálogo de selección.

JBuilder se puede utilizar con diversas plataformas. Consulte la siguiente tabla para ver una descripción de las convenciones de plataforma utilizadas en la documentación.

Tabla 1.2 Convenciones de las plataformas

Elementos	Significado
Vías de acceso	Las vías de acceso a los directorios en la documentación se indican con una barra normal (/). Para la plataforma Windows se utiliza la barra invertida (\).
Directorio de inicio	La ubicación del directorio inicial varía según la plataforma y se indica con una variable <home>. <ul style="list-style-type: none"> • En UNIX y Linux, el directorio inicial puede variar. Por ejemplo, puede ser <code>/user/<nombre de usuario></code> o <code>/home/<nombre de usuario></code> • En Windows NT, el directorio inicial es <code>C:\Winnt\Profiles\<nombre de usuario></code> • En Windows 2000, el directorio inicial es <code>C:\Documents and Settings\<nombre de usuario></code>
Imágenes de pantalla	Las imágenes o capturas de pantalla utilizan el aspecto Metal en diversas plataformas.

Asistencia a los desarrolladores

Borland ofrece una amplia gama de opciones de asistencia técnica y recursos de información para ayudar a los desarrolladores a obtener lo máximo de sus productos Borland. Estas opciones incluyen un rango de programas de asistencia técnica de Borland, así como servicios gratuitos en Internet, donde es posible efectuar búsquedas en nuestra amplia base de información y ponerse en contacto con otros usuarios de productos Borland.

Cómo ponerse en contacto con el servicio técnico de Borland

Borland ofrece varios programas de asistencia para clientes y clientes potenciales. Se puede elegir entre varios tipos de asistencia, que van desde la asistencia para la instalación de los productos Borland hasta el asesoramiento de expertos y la asistencia pormenorizada (servicios no gratuitos).

Si desea más información sobre el servicio al desarrollador de Borland, visite nuestra página web, en <http://www.borland.com/devsupport>.

Cuando se ponga en contacto con el servicio técnico, tenga a mano la información completa sobre el entorno, la versión del producto utilizada y una descripción detallada del problema.

Si necesita más información sobre las herramientas o la documentación de otros proveedores, póngase en contacto con ellos.

Recursos en línea

También puede obtener información de los siguientes recursos en línea:

World Wide Web	http://www.borland.com/
FTP	ftp://ftp.borland.com/ Documentación técnica disponible por ftp anónimo.
Listserv	Para suscribirse a circulares electrónicas, rellene el formulario en línea que aparece en: http://info.borland.com/contact/listserv.html y para el servidor de listas internacional de Borland: http://info.borland.com/contact/intlist.html

World Wide Web

Visite periódicamente www.borland.com/jbuilder. El equipo de desarrollo de productos Java publica en esta página documentación técnica, análisis de competitividad, respuestas a preguntas frecuentes, aplicaciones de ejemplo, software actualizado e información sobre productos nuevos y antiguos.

En particular, pueden resultar interesantes las siguientes direcciones:

- <http://www.borland.com/jbuilder/> (actualizaciones de software y otros archivos)
- <http://www.borland.com/techpubs/jbuilder/> (actualizaciones de documentación y otros archivos)
- <http://community.borland.com/> (contiene nuestra revista de noticias para desarrolladores en formato web)

Grupos de noticias de Borland

Puede registrar JBuilder y participar en los grupos de debate sobre JBuilder, estructurados en hilos. Los grupos de noticias de Borland proporcionan los medios a todos los clientes de la comunidad Borland para intercambiar sugerencias y técnicas acerca de los productos, herramientas relacionadas y tecnologías Borland.

Puede encontrar grupos de noticias, moderados por los usuarios, sobre JBuilder y otros productos de Borland, en <http://www.borland.com/newsgroups>

Usenet, grupos de noticias

En Usenet existen los siguientes grupos dedicados a Java y temas relacionados:

- `news:comp.lang.java.advocacy`
- `news:comp.lang.java.announce`
- `news:comp.lang.java.beans`
- `news:comp.lang.java.databases`
- `news:comp.lang.java.gui`
- `news:comp.lang.java.help`
- `news:comp.lang.java.machine`
- `news:comp.lang.java.programmer`
- `news:comp.lang.java.security`
- `news:comp.lang.java.softwaretools`

Nota Se trata de grupos moderados por usuarios; no son páginas oficiales de Borland.

Información sobre errores

Si encuentra algún error en el software, comuníquelo en la página Support Programs, en <http://www.borland.com/devsupport/namerica/>. Pulse el enlace "Reporting Defects" para llegar al formulario Entry.

Cuando informe sobre un fallo, incluya todos los pasos necesarios para llegar a él, así como toda la información posible sobre la configuración, el entorno y las aplicaciones que se estaban utilizando junto con JBuilder. Intente explicar con la mayor claridad posible las diferencias entre el comportamiento esperado y el obtenido.

Si desea enviar felicitaciones, sugerencias o quejas al equipo de documentación de JBuilder, envíe un mensaje a jpgpubs@borland.com. Envíe únicamente comentarios sobre la documentación. Tenga en cuenta que los asuntos relacionados con el servicio técnico se deben enviar al departamento de asistencia técnica para programadores.

JBuilder es una herramienta creada por desarrolladores y para desarrolladores. Valoramos sumamente sus aportaciones.

Aplicaciones de base de datos JBuilder

El desarrollo de
aplicaciones de base de
datos es una función de
JBuilder Enterprise

Una aplicación de base de datos es aquella que accede a datos almacenados y permite visualizarlos y, posiblemente, modificarlos o manipularlos. En la mayoría de los casos, los datos se almacenan en bases de datos. Sin embargo, también se pueden almacenar en archivos de texto (o con otro formato). JBuilder permite acceder a esta información y manipularla utilizando las propiedades, los métodos y los sucesos definidos en los paquetes `DataSet` de la biblioteca de componentes `DataExpress`, en combinación con el paquete `dbSwing`.

Una aplicación que solicita información de una fuente de datos, como una base de datos, se denomina aplicación cliente. Un SGBD (Sistema de gestión de bases de datos) que gestiona solicitudes de datos de varios clientes se denomina aplicación servidor.

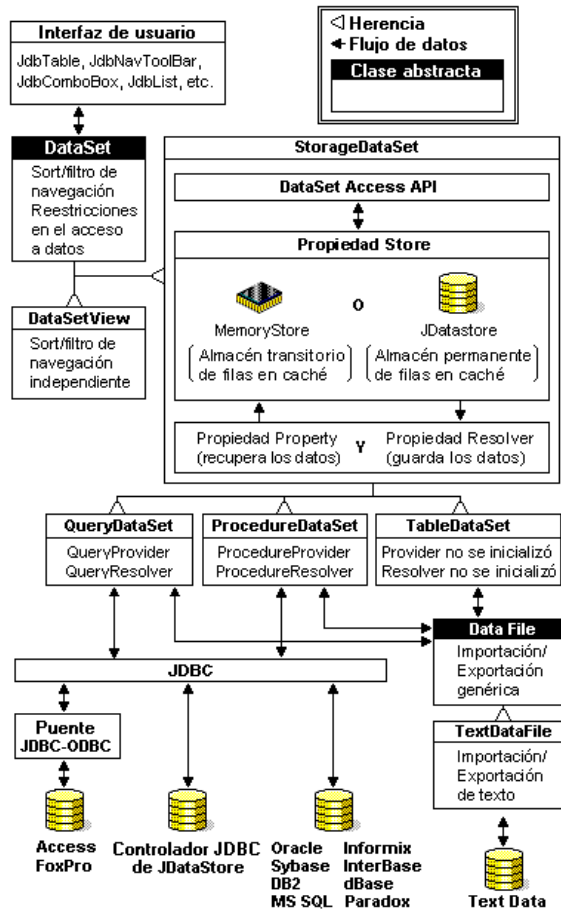
Arquitectura de las aplicaciones de base de datos

La Arquitectura `DataExpress` de JBuilder está centrada en la construcción de aplicaciones cliente/servidor totalmente escritas en Java, applets, servlets, y páginas `JavaServer (JSP)` para Internet o intranets. Dado que las aplicaciones que se construyen en JBuilder se ejecutan íntegramente en Java, son multiplataforma.

Las aplicaciones JBuilder se comunican con servidores de bases de datos a través de la API `JDBC`, la especificación de conectividad de bases de datos de Sun. `JDBC` es la API Java estándar para acceso y manipulación de datos de bases de datos. Las aplicaciones de bases de datos JBuilder pueden conectarse con cualquier base de datos que tenga un controlador `JDBC`.

El diagrama siguiente muestra una aplicación de bases de datos típica y las capas entre la aplicación cliente DataExpress de JBuilder y la fuente de datos:

Figura 2.1 Diagrama de una aplicación de base de datos



En el siguiente apartado, "[Componentes DataExpress](#)", se tratan con más profundidad los componentes de la arquitectura DataExpress.

Componentes DataExpress

DataExpress es un paquete, `com.borland.dx.dataset`, de clases e interfaces de Borland que proporcionan acceso básico a datos. Este paquete también define las clases básicas de proveedor y almacenador, así como una clase abstracta de `DataSet` que se extiende a otros objetos `DataSet`. Estas clases posibilitan el acceso a la información almacenada en bases de datos y otras

fuentes de datos. Este paquete incluye funciones que cubren las tres fases principales de la gestión de datos:

- **Suministro**

Función general que permite obtener datos y gestionar los conjuntos de datos locales. (Las clases del paquete `com.borland.dx.sql.dataset` gestionan las conexiones específicas de JDBC a servidores remotos.)

- **Manipulación**

Búsqueda y modificación de datos de forma local.

- **Almacenamiento**

Rutinas generales para que se actualicen los datos de la fuente original desde el `DataSet` local. (Las clases del paquete `com.borland.dx.sql.dataset` gestionan el almacenamiento de los cambios en los datos en servidores remotos a través de JDBC.)

Principales funciones y ventajas

Los componentes `DataExpress` están diseñados modularmente, para facilitar la separación de las principales funciones. Gracias a este diseño, pueden adaptarse a una amplia variedad de aplicaciones. He aquí algunas de las características modulares de la arquitectura `DataExpress`:

- Funcionalidad básica del `DataSet`

Se trata de un conjunto de funciones de manipulación de datos que todas las aplicaciones que utilicen `DataExpress` pueden usar. Muchas de estas funciones pueden aplicarse declarando valores de propiedades y sucesos. Por ejemplo, la navegación, el acceso a los datos o su actualización, la ordenación o el filtrado de datos, las relaciones maestro-detalle, consultas, restricciones, valores por defecto, etc.

- Independencia con respecto a la fuente de datos

La recuperación y actualización de datos de una fuente de datos, como un servidor Oracle o Sybase, se limita a dos interfaces clave: `Proveedor/Almacenador`. Al separar claramente la recuperación y actualización de datos con dos interfaces, resulta sencillo crear nuevos componentes `proveedor/almacenador` para nuevas fuentes de datos. Existen dos implementaciones `proveedor/almacenador` para controladores JDBC estándar que proporcionan acceso a conocidas bases de datos como Oracle, Sybase, Informix, InterBase, DB2, MS SQL Server, Paradox, dBASE, FoxPro, Access y otras. Es posible crear implementaciones personalizadas de componentes `proveedor/almacenador` para EJB, servidores de aplicaciones, SAP, BAAN, IMS, CICS, etc.

- Mecanismo de almacenamiento independiente

Cada vez que se extrae algún dato de un proveedor, queda almacenado en una caché dentro del DataSet. El sistema lleva el control de todas las modificaciones realizadas en la caché del DataSet, lo que permite a las implementaciones del almacenador determinar lo que deben actualizar en la fuente de los datos. DataExpress ofrece dos métodos de almacenamiento en caché: MemoryStore (el mecanismo por defecto) y JDataStore.

MemoryStore almacena en memoria caché todos los datos y sus modificaciones. JDataStore utiliza una base de datos integrable de alto rendimiento creada íntegramente en Java, que ocupa muy poca memoria, para almacenar en memoria caché los datos y sus modificaciones. El mecanismo JDataStore es idóneo para los ordenadores portátiles o no conectados a la red, así como para la replicación asíncrona de datos y las aplicaciones de base de datos compactas.

- Asociación de componentes visuales a datos

Los componentes DataSet de DataExpress proporcionan una potente interfaz de programación y permiten asociar directamente datos con componentes enlazados a datos, mediante parámetros de propiedades de ratón realizadas en un diseñador visual. JBuilder viene con una serie de componentes visuales basados en Swing, que se enlazan directamente con los componentes DataSet.

He aquí algunas ventajas que ofrece la utilización de la arquitectura modular de DataExpress:

- Ordenadores de red

Como se indicaba anteriormente, la filosofía proveedor/almacenador limita las interacciones con fuentes de datos arbitrarias a dos puntos claramente definidos. Este mecanismo ofrece otras dos ventajas:

- El par proveedor/almacenador puede subdividirse fácilmente para establecer un nivel intermedio. Puesto que la lógica proveedor/almacenador suele tener carácter transaccional, es idónea para la subdivisión en un nivel intermedio.
- Se trata de un modelo informático “sin estados”, que resulta idóneo para los ordenadores conectados en red. La conexión entre el cliente del componente DataSet y la fuente de los datos puede desconectarse una vez proporcionados los datos. Cuando se necesite volver a guardar los cambios en la fuente de datos, basta con restablecer la conexión durante el tiempo que dure la transacción de almacenamiento.

- Desarrollo rápido de interfaces de usuario

Como los componentes `DataSet` pueden asociarse a componentes enlazados a datos con sólo definir el valor de una propiedad, resultan idóneos para generar rápidamente las interfaces de usuario de las aplicaciones de base de datos.

- Informática móvil

Con la introducción del componente `DataStore`, las aplicaciones `DataExpress` tienen a su disposición una base de datos transportable y persistente. El `DataStore` (almacén de datos) puede contener varios componentes `DataSet`, archivos arbitrarios y objetos Java. Ello permite almacenar de manera permanente todo el estado de una aplicación en un solo archivo. Los componentes `DataSet` incorporan una tecnología de replicación de datos que permite guardar y consolidar las modificaciones realizadas en los datos replicados, volviéndolos a almacenar en la fuente.

- Aplicaciones incrustadas

Debido a su reducido consumo de memoria y su alta velocidad, la base de datos de `JDataStore` es idónea para las aplicaciones incrustadas y disfruta de toda la versatilidad funcional y la semántica del componente `DataSet`.

Para más información sobre la arquitectura `DataExpress`, visite la dirección web de Borland, <http://www.borland.com/jbuilder/>, donde existe un documento introductorio al respecto.

Descripción general de los componentes DataExpress

En los paquetes `com.borland.dx.dataset`, `com.borland.dx.sql.dataset`, y `com.borland.datastore` se encuentran las funciones fundamentales que se necesitan para la conexión a datos. Los componentes de estos paquetes encapsulan la conexión entre la aplicación y su origen de datos y el comportamiento necesario para el manejo de datos. Entre las funciones de estos paquetes se encuentran las de conectividad de la base de datos además de la funcionalidad del conjunto de datos.

La siguiente lista presenta las principales clases y componentes de los paquetes Borland relacionados con bases de datos, junto con una breve descripción del componente o clase. La columna de la derecha de la tabla enumera las propiedades de la clase o del componente más frecuentemente utilizadas. Algunas propiedades son en sí mismas objetos que agrupan varias propiedades. Estos objetos de propiedades complejos terminan con la palabra `Descriptor` y contienen propiedades clave a las que

(normalmente) hay que asignar un valor para poder utilizar el componente.

Componente/Clase	Descripción	Propiedades más utilizadas
Database	<p>Un componente necesario para acceder a datos almacenados en un servidor remoto, el componente <code>Database</code> gestiona la conexión JDBC con la base de datos del servidor SQL.</p> <p>Consulte el Capítulo 4, “Conexión con bases de datos”, donde encontrará más información para utilizar este componente.</p>	<p>El objeto <code>ConnectionDescriptor</code> almacena propiedades de conexión del controlador, el nombre de usuario, la contraseña y la conexión URL. Se accede a él mediante la propiedad <code>connection</code>.</p>
DataSet	<p>Es una clase abstracta que proporciona el comportamiento básico del conjunto de datos. <code>DataSet</code> también proporciona la infraestructura para el almacenamiento de datos, al mantener una matriz bidimensional organizada por filas y columnas. Posee el concepto de posición de fila actual, lo que permite recorrer las filas de datos y gestiona un pseudoregistro que contiene el registro nuevo o editado actual hasta que se lo envía a <code>DataSet</code>. Dado que <code>DataSet</code> deriva de <code>ReadWriteRow</code>, tiene métodos de asignación y obtención de valores de campo.</p>	<p>El objeto <code>SortDescriptor</code> contiene propiedades que influyen en el orden por el que se accede a los datos en los componentes de la interfaz de usuario. Se ajusta mediante la propiedad <code>sort</code>. Consulte “Clasificación de datos” en la página 11-9 para obtener información sobre su utilización.</p> <p>El objeto <code>MasterLinkDescriptor</code> contiene propiedades para la gestión de relaciones maestro-detalle entre dos componentes <code>DataSet</code>. Se accede a él mediante la propiedad <code>masterLink</code> en el <code>DataSet</code> de detalle. Consulte el Capítulo 9, “Establecimiento de una relación maestro-detalle” para obtener información sobre su utilización.</p>
StorageDataSet	<p>Una clase que se deriva de <code>DataSet</code> suministrando una implementación de almacenamiento de los datos y manipulación de la estructura de <code>DataSet</code>.</p> <p>Los componentes <code>StorageDataSet</code> se alimentan de datos mediante la extracción de información de una base de datos remota (tales como <code>InterBase</code> u <code>Oracle</code>) o mediante la importación de datos almacenados en un archivo de texto. Esto se consigue instanciando una de sus subclases: <code>QueryDataSet</code>, <code>ProcedureDataSet</code> o <code>TableDataSet</code>.</p>	<p>La propiedad <code>tableName</code> especifica la fuente de datos del componente <code>StorageDataSet</code>.</p> <p>La propiedad <code>maxRows</code> define el número máximo de filas que puede contener inicialmente <code>DataSet</code>.</p> <p>La propiedad <code>readOnly</code> controla la posibilidad de modificar los datos.</p>

Componente/Clase	Descripción	Propiedades más utilizadas
DataStore	<p>El componente <code>DataStore</code> proporciona un sustituto de <code>MemoryStore</code> que permite almacenar datos de forma permanente. Un <code>JDataStore</code> proporciona almacenamiento de datos en caché de alto rendimiento y persistencia compacta en los <code>DataSets</code> de <code>DataExpress</code>, archivos arbitrarios y objetos Java. El componente <code>DataStore</code> almacena en un mismo archivo varios flujos de datos. Los archivos <code>JDataStore</code> tienen una estructura jerarquizada de directorios que asocia un nombre y varios estados de directorio a un flujo de datos determinado. <code>JDataStore</code> se puede gestionar como cualquier base de datos SQL: es posible conectarse a él como a cualquier otro servidor, ejecutar consultas SQL, etc.</p> <p>En “JDataStore y JBuilder” en la página 2-14 y en la <i>Guía del desarrollador de JDataStore</i> puede encontrar más información sobre el componente <code>DataStore</code>.</p>	<p>El almacenamiento en caché y la persistencia de los componentes <code>StorageDataSet</code> de <code>DataStore</code> se consigue por medio de los valores de dos propiedades requeridas de <code>StorageDataSet</code>, llamadas <code>store</code> y <code>storeName</code>. Por defecto, todos los objetos <code>StorageDataSet</code> utilizan <code>MemoryStore</code>, si no se indica lo contrario, mediante la propiedad <code>store</code>. Actualmente, <code>MemoryStore</code> y <code>DataStore</code> son las únicas implementaciones de la propiedad <code>store</code>. La propiedad <code>storeName</code> contiene el nombre unívoco asociado a este <code>StorageDataSet</code> en el archivo <code>DataStore</code>.</p>
DataStoreDriver	<p><code>DataStoreDriver</code> es el controlador JDBC asociado al <code>JDataStore</code>. Este controlador permite tanto acceso local como remoto. En ambos tipos de accesos es necesario especificar un nombre de usuario (puede ser cualquier cadena, y no es necesario configurarla). Si <code>JDataStore</code> no está encriptado, el espacio destinado a la contraseña puede dejarse vacío. En caso contrario, será necesario introducir una contraseña no nula.</p>	
QueryDataSet	<p>El componente <code>QueryDataSet</code> almacena los resultados de una cadena de consultas ejecutada en una base de datos de servidor. Este componente funciona con el componente <code>Database</code> para conectar con las bases de datos del servidor SQL y ejecuta con parámetros (si los hay) la consulta especificada. Una vez almacenados los datos resultantes en el componente <code>QueryDataSet</code>, se pueden manipular con la API <code>DataSet</code>.</p> <p>Consulte el “Consultas en bases de datos” en la página 5-2, donde encontrará más información para utilizar este componente.</p>	<p>El objeto <code>QueryDescriptor</code> contiene la sentencia de consulta SQL, los parámetros de consulta y información sobre la conexión a base de datos. Se accede a él mediante la propiedad <code>query</code>.</p>

Componente/Clase	Descripción	Propiedades más utilizadas
ProcedureDataSet	<p>El componente <code>ProcedureDataSet</code> almacena los resultados de un procedimiento almacenado ejecutado en una base de datos de servidor. El componente <code>ProcedureDataSet</code> funciona con el componente <code>Database</code> de manera similar al componente <code>QueryDataSet</code>.</p> <p>Consulte el Capítulo 6, “Utilización de procedimientos almacenados” y el tutorial relacionado, Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados” donde encontrará más información e instrucciones de uso sobre este componente.</p>	<p>El objeto <code>ProcedureDescriptor</code> contiene la sentencia de consulta, los parámetros de consulta, el componente de la base de datos y otras propiedades. Se accede a él mediante la propiedad <code>procedure</code> del componente <code>ProcedureDataSet</code>.</p>
TableDataSet	<p>Utilice este componente cuando importe datos de un archivo de texto. Este componente se deriva de la clase <code>DataSet</code>. Reproduce las funciones del servidor SQL sin necesidad de una conexión con un servidor SQL.</p> <p>Consulte el Capítulo 3, “Importación y exportación de datos desde un archivo de texto” y el tutorial relacionado, Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto” donde encontrará más información e instrucciones de uso sobre este componente.</p>	<p>La propiedad (heredada) <code>dataFile</code> especifica el nombre de archivo desde el que se cargan los datos en <code>DataSet</code> y el nombre del archivo en el que se guardan.</p>
DataSetView	<p>Este componente presenta otra “vista” de los datos de un <code>StorageDataSet</code>. Dispone de su propia propiedad <code>sort</code> (heredada) que, con un valor nuevo, permite una presentación de los datos en distinto orden. También posee capacidades de filtro y de desplazamiento independientes de sus <code>StorageDataSet</code> asociados.</p> <p>Consulte “Presentación de una vista alternativa de los datos” en la página 12-24, donde encontrará más información para utilizar este componente.</p>	<p>La propiedad <code>storageDataSet</code> indica el componente que contiene los datos de los que presenta una vista <code>DataSetView</code>.</p>

Componente/Clase	Descripción	Propiedades más utilizadas
Column	<p>Una <code>Column</code> representa la colección de todas las filas de un elemento de datos concreto, por ejemplo, todos los valores <code>Nombre</code> de una tabla. Una <code>Column</code> obtiene su valor cuando un <code>DataSet</code> se instancia o como resultado de un cálculo.</p> <p>La <code>Column</code> está gestionada por su componente <code>StorageDataSet</code>.</p> <p>Consulte “Utilización de columnas” en la página 7-1, donde encontrará más información para utilizar este componente.</p>	<p>Se pueden establecer propiedades a nivel de <code>Column</code> como convenga para que las configuraciones que afecten a la columna de datos entera se puedan establecer en un lugar, por ejemplo, <code>font</code>. Las herramientas de diseño de JBuilder incluyen el acceso a propiedades a nivel de columna al hacer doble clic en una implementación <code>StorageDataSet</code> del panel de contenido y, a continuación, seleccionar la <code>Column</code> con la que se desea trabajar. Las propiedades y sucesos del componente <code>Column</code> seleccionado se muestran en el diseñador de columnas (sólo las propiedades) o en el Inspector, y se pueden modificar en cualquiera de los dos lugares.</p>
DataRow	<p>El componente <code>DataRow</code> es una colección de todos los datos <code>Column</code> de una única fila donde cada fila es un registro completo de información. El componente <code>DataRow</code> utiliza las mismas columnas que el <code>DataSet</code> con el que se construyó. Los nombres de las columnas de un componente <code>DataRow</code> son nombres de campo.</p> <p>Es conveniente trabajar con un <code>DataRow</code> cuando se comparan los datos de dos filas o se quieren localizar datos en un <code>DataSet</code>. Puede utilizarse en todos los métodos <code>DataSet</code> que requieran un <code>ReadRow</code> o <code>ReadWriteRow</code>.</p>	
ParameterRow	<p>El componente <code>ParameterRow</code> tiene un <code>Column</code> por cada columna del conjunto asociado de datos que desee consultar. Introduzca los valores que utilizará la consulta en el <code>ParameterRow</code> y asíócelos con la consulta mediante sus nombres de parámetro (que son los nombres de columna del <code>ParameterRow</code>).</p> <p>Consulte el “Utilización de consultas parametrizadas para obtener datos” en la página 5-13, donde encontrará más información para utilizar este componente.</p>	

Componente/Clase	Descripción	Propiedades más utilizadas
DataModule	<p>El módulo de datos es una interfaz del paquete <code>com.borland.dx.dataset</code>. El diseñador de JBuilder reconocerá una clase que implemente el módulo de datos como una clase que contiene varios componentes <code>dataset</code> agrupados en un modelo de datos. Es posible crear un modelo de datos vacío seleccionando el icono Módulo de datos del cuadro de diálogo Archivo Nuevo. A continuación, mediante la paleta y el árbol de componentes, sitúe en él varios objetos <code>DataSet</code> y establezca conexiones, consultas, clasificaciones y la lógica de reglas empresariales personalizadas. Los módulos de datos simplifican la reutilización y el uso múltiple de colecciones de componentes <code>dataset</code>. Por ejemplo, una o más clases de interfaz de la aplicación pueden utilizar una instancia compartida del <code>DataModule</code> de cliente.</p> <p>Consulte el Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”, donde encontrará más información para utilizar este componente.</p>	

Existen otras muchas clases y componentes en los paquetes `com.borland.dx.dataset`, `com.borland.dx.sql.dataset` y `com.borland.datastore` además de varias clases de apoyo en otros paquetes como `util` y `view`. Si desea información más detallada sobre los paquetes y las clases disponibles en la biblioteca de DataExpress, consulte la documentación de la *DataExpress Component Library Reference*.

DataExpress para componentes EJB

El paquete DataExpress para EJB, el paquete `com.borland.dx.ejb`, contiene los componentes DataExpress para EJB. Estos componentes permiten proporcionar datos de beans entidad EJB a DataSets de DataExpress y, a continuación, almacenar los cambios realizados en los DataSets en los beans entidad.

El paquete DataExpress para EJB no se trata en este manual. Si desea información adicional acerca de cómo utilizar el paquete DataExpress para EJB para desarrollar aplicaciones empresariales enlazadas a datos, consulte “Los componentes DataExpress para EJB” en la *Guía del desarrollador de Enterprise JavaBeans*. Si desea información de referencia, consulte la documentación API del paquete `com.borland.dx.ejb`.

InternetBeans Express

El paquete InternetBeans Express, `com.borland.internetbeans`, proporciona componentes y extensiones de etiquetas de páginas JSP para la generación de presentaciones y la interacción con aplicaciones web.

El paquete InternetBeans Express no se trata en este manual. Si desea más información acerca de la utilización de los componentes del paquete InternetBeans Express para desarrollar aplicaciones de páginas JSP y servlets enlazados a datos, consulte “InternetBeans Express” en la *Guía del desarrollador de aplicaciones web*. Si desea información de referencia, consulte la documentación API del paquete `com.borland.internetbeans`.

Componentes de bases de datos XML

Los componentes XML de base de datos de JBuilder admiten el desarrollo de aplicaciones XML de base de datos. Se pueden añadir componentes a la aplicación desde la ficha XML de la paleta de componentes del diseñador de interfaces de usuario. Existen componentes basados en modelos y componentes basados en plantillas. Los componentes basados en modelos utilizan un documento de mapa que determina cómo se transfieren los datos entre una estructura XML y los metadatos de la base de datos. Para utilizar componentes basados en plantillas, debe proporcionar una sentencia SQL y el componente generará el documento XML. Esa sentencia SQL funciona como la plantilla que se sustituye en el documento XML al aplicarla.

La utilización de componentes XML de base de datos no se trata en este manual. Si desea información más detallada, consulte “Utilización de los componentes de base de datos XML de JBuilder” en la *Guía del desarrollador de XML*. Si desea información de referencia, consulte la documentación API de los paquetes

`com.borland.jbuilder.xml.database.common`, `com.borland.jbuilder.xml.database.template`, y `com.borland.jbuilder.xml.database.xmldbms`.

dbSwing

El paquete `dbSwing` permite generar aplicaciones de base de datos que aprovechan la arquitectura de los componentes Swing de Java. Además de las subclases enlazadas a datos que incorporan de origen la mayoría de los componentes Swing, `dbSwing` incluye varias utilidades diseñadas específicamente para el desarrollo de aplicaciones basadas en DataExpress y JDataStore.

Para crear una aplicación de base de datos, es necesario, en primer lugar, conectarse con una base de datos y suministrar datos a un `DataSet`.

“[Suministro de datos de los ejemplos](#)” en la página 11-2 configura una consulta que se puede utilizar como punto de partida para la creación de aplicaciones de base de datos y una interfaz de usuario básica.

Para utilizar los componentes de dbSwing asociados a datos:

- 1 Abra el archivo Marco y seleccione la pestaña Diseño.
- 2 Seleccione una de las pestañas dbSwing: dbSwing, Má dbSwing o Modelos de dbSwing.
- 3 Pulse sobre un componente de la paleta de componentes y haga clic sobre el diseñador de interfaces de usuario para colocar el componente en la aplicación.
- 4 Seleccione el componente en el árbol de componentes o en el diseñador de interfaces de usuario.

Dependiendo del tipo de componente y de la propiedad `layout` del `contentPane` que lo contiene, el diseñador hará que aparezcan controladores de tamaño en los bordes del componente seleccionado.

Algunos objetos (`JdbNavToolBar` y `JdbStatusLabel`) del componente se asocian automáticamente al conjunto de datos que tiene el foco. Para otros (como `JdbTable`), asigne valores a las propiedades `dataSet` o `columnName` del componente en el Inspector, para asociarlo con un `DataSet` instanciado.

En la siguiente lista se indican algunos de los componentes dbSwing disponibles en la pestaña dbSwing de la paleta de componentes:

- `TableScrollPane`
- `JdbTable`
- `JdbNavToolBar`
- `JdbStatusLabel`
- `JdbTextArea`
- `JdbComboBox`
- `JdbLabel`
- `JdbList`
- `JdbTextPane`
- `JdbTextField`

dbSwing ofrece importantes ventajas frente a Swing, como una mayor variedad de funciones y la posibilidad de enlazar con datos. Además, dbSwing es ligero, su aspecto visual puede simular el de muchas plataformas diferentes y cumple estrictamente los estándares de Swing. Si únicamente utiliza componentes dbSwing, tendrá la seguridad de que todos ellos son ligeros.

Si desea más información acerca del paquete dbSwing, consulte la documentación API de dbSwing.

Los módulos de datos y el modelador de datos

Los módulos de datos proporcionar un contenedor para los componentes de acceso a datos. Los módulos de datos simplifican el desarrollo de aplicaciones de base de datos mediante la división en módulos del código y la separación de la lógica de acceso a bases de datos y las reglas empresariales de las aplicaciones de la lógica de la interfaz de usuario. También se puede controlar la utilización de los módulos de datos haciendo llegar los archivos de clase (.class) únicamente a los desarrolladores.

El modelador de datos puede ayudarle a generar módulos de datos que encapsulan una conexión con una base de datos y las consultas que se van a ejecutar en ella.

Para obtener más información acerca de los módulos de datos y el Modelador de datos, consulte el capítulo [Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”](#).

Explorador de bases de datos

El Explorador de bases de datos (Herramientas | Explorador de bases de datos) es un visualizador de base de datos jerárquicas, que permite modificar los datos.

El Explorador de bases de datos muestra información sobre metadatos de la base de datos basada en JDBC, dentro de una ventana con dos paneles. El panel de la izquierda muestra un árbol que representa jerárquicamente un conjunto de bases de datos y sus tablas, vistas, procedimientos almacenados y metadatos asociados. El panel derecho muestra, en varias fichas, información descriptiva acerca de cada nodo del árbol. En algunos casos, los datos del panel derecho pueden editarse también.

Si desea más información acerca del Explorador de bases de datos, consulte el [Capítulo 15, “Tareas de administración de bases de datos”](#).

Monitor JDBC

El Monitor JDBC (Herramientas | Monitor JDBC) es una herramienta gráfica que se puede utilizar para supervisar el tráfico JDBC. EL Monitor JDBC realiza un seguimiento de cualquier controlador JDBC (es decir, cualquier subclase de `java.sql.Driver`) mientras lo utiliza JBuilder. El monitor de JDBC permite observar todos los datos que salen directamente del controlador JDBC.

Si desea más información e instrucciones de uso acerca del Monitor JDBC, consulte [“Seguimiento de conexiones de base de datos” en la página 15-11](#).

JDataStore y JBuilder

JDataStore es una polifacética solución de almacenamiento de datos, de altas prestaciones y bajo consumo de recursos, desarrollada enteramente para Java. JDataStore es:

- Una base de datos relacional embebida, con interfaces JDBC y DataExpress, que incorpora mecanismos de acceso multiusuario transaccional sin bloqueo y recuperación frente a fallos.
- Un almacén de objetos, que permite guardar objetos serializados, DataSet y otros flujos de archivo.
- Un componente JavaBean que puede manipularse con herramientas visuales de creación de Beans, como JBuilder.

Un Explorador de JDataStore visual basado íntegramente en Java, que permite administrar los datastores.

Si desea información más completa y actualizada sobre la utilización de los DataStore, consulte la *Guía del desarrollador de JDataStore*.

Diferencias entre el uso de los controladores JDataStore y JDBC

La configuración de una aplicación de base de datos para que acceda a un sistema de gestión de bases de datos relacional con controladores JDBC ofrece sustanciales ventajas frente a la configuración de la aplicación para que utilice JDataStore. En los siguientes apartados se destacan algunas de las ventajas que se derivan de este enfoque.

El uso del controlador JDBC es aconsejable para:

- Utilizar una API de JDBC estándar.
- Trabajar con datos SQL dinámicos: se puede utilizar un `QueryProvider` para efectuar una consulta en una base de datos SQL y trabajar con datos dinámicos, guardando los cambios cuando sea necesario.
- Aprovechar el acceso remoto con RemoteJDBC.

Nota JDataStore se puede utilizar con o sin controladores JDBC. De hecho, en la mayoría de los ejemplos y tutoriales a los que se hace referencia en este manual, JDataStore se utiliza con controladores JDBC. Si lo desea, puede consultar el tutorial “Edición fuera de línea con JDataStore” en la *Guía del desarrollador de JDataStore* para ver un ejemplo de cómo puede utilizarse

un componente `DataStore` para editar datos fuera de línea, en lugar de tener que realizar una conexión JDBC a un servidor JDataStore.

El uso de JDataStore es aconsejable para:

- Trabajar fuera de línea: los datos se pueden guardar y modificar dentro del sistema de archivo JDataStore, y las modificaciones se pueden almacenar cuando se restablece la conexión con la fuente de datos.
- Almacenar objetos y datos.
- Trabajar con grandes conjuntos de datos.

Ventajas adicionales de un JDataStore

Es recomendable utilizar un JDataStore en cualquiera de los siguientes supuestos:

- **Organización.**

Para organizar los componentes `StorageDataSet`, archivos y JavaBean/objetos serializados de una aplicación en un único almacenamiento totalmente Java, persistente, de altas prestaciones, portátil y compacto.

- **Réplica asíncrona de datos.**

Para los ordenadores móviles o no conectados a ninguna red, `StorageDataSet` incorpora un mecanismo de almacenamiento y reconciliación de los datos modificado extraídos de cualquier fuente de datos arbitraria (como JDBC, Application Server, SAP, BAAN, etc.).

- **Aplicaciones incrustadas.**

El consumo de recursos de JDataStore es muy reducido. Los componentes `StorageDataSet` también proporcionan una funcionalidad excelente para los componentes de la interfaz de usuario enlazados a datos.

- **Rendimiento.**

Para aumentar el rendimiento y ahorro de memoria en un `StorageDataSet` grande. Los componentes `StorageDataSet` que utilizan `MemoryStore` tienen un rendimiento ligeramente mejor que `DataStore` para un número pequeño de filas. `DataStore` almacena los datos de `StorageDataSet` y los indexa en un formato extremadamente compacto. A medida que aumenta el número de filas, los `StorageDataSet` que utilizan `DataStore` rinden mucho más y necesita menos memoria que los que utilizan `MemoryStore`.

Para obtener más información sobre los *DataStore*, consulte la *Guía del desarrollador de JDataStore*.

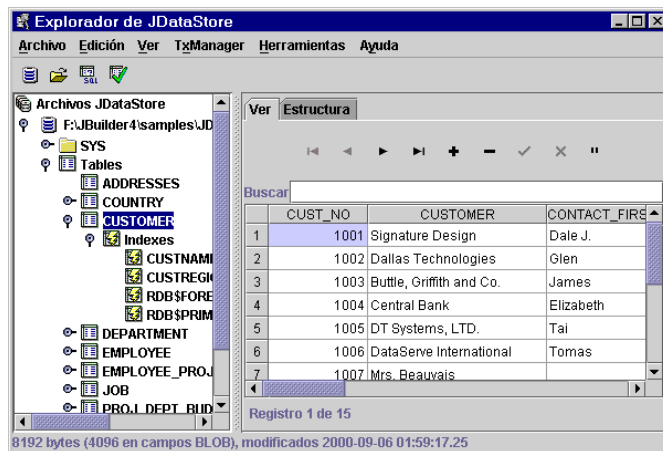
El Explorador de JDataStore

El Explorador de JDataStore permite:

- Examinar el contenido de un `DataStore`. El directorio de almacenamiento aparece en un control con forma de árbol, en el que se agrupa cada `DataSet` con sus respectivos índices. Cuando se selecciona un flujo de datos en este árbol, aparece su contenido (suponiendo que se trate de un tipo de archivo para el cual el explorador disponga de un visor, como los de texto, gif, o los `DataSet`).
- Realizar muchas operaciones de almacenamiento sin necesidad de escribir código. Es posible crear un nuevo `JDataStore`, importar en los `DataSet` archivos de texto delimitados, importar archivos en flujos de archivo, borrar índices, borrar `DataSet` u otros flujos de datos y verificar la integridad del `DataStore`.
- Administrar consultas que proporcionen datos para su almacenamiento en los `DataSet` del almacén, modificar los `DataSet` y guardar en las tablas del servidor las modificaciones realizadas.

Utilice el comando de menú Herramientas | Explorador de JDataStore para abrir el Explorador de JDataStore.

Figura 2.2 Explorador de JDataStore



Operaciones del explorador de JDataStore

Para crear un `JDataStore`:

- 1 Abra el Explorador de JDataStore seleccionando Herramientas | Explorador de JDataStore.
- 2 Seleccione Archivo | Nuevo o haga clic en el botón Nuevo JDataStore.

- 3 Introduzca un nombre para el nuevo almacén de datos y pulse Aceptar. Se abrirá el explorador, y se creará el nuevo almacén de datos.

Para importar en un DataSet un archivo de texto

- 1 Seleccione Herramientas | Importar | Texto en Tabla.
- 2 Escriba el nombre del archivo de texto y el nombre de almacén del conjunto de datos (DataSet) que desee crear.

El contenido del archivo de texto debe estar en un formato delimitado al cual JBuilder pueda exportar, y debe haber un archivo SCHEMA (.schema) con el mismo nombre en ese directorio, en el que se defina la estructura del conjuntos de datos de destino (para crear un archivo .schema, consulte [“Exportación de datos” en la página 3-4](#). El nombre que se asigna por defecto al almacén de datos es el del archivo de entrada, incluyendo la extensión. Puesto que esta operación genera un conjunto de datos, y no un flujo de archivos, quizás le interese omitir la extensión en el nombre del almacén.

- 3 Pulse Aceptar.

Para importar un archivo en un flujo de archivos:

- 1 Seleccione Herramientas | Importar | Archivo.
- 2 Especifique un nombre de archivo de entrada y un nombre de almacén y pulse Aceptar.

Para verificar el JDataStore abierto, seleccione Herramientas | Verificar JDataStore o pulse el botón Verificar JDataStore.

Se verificará todo el DataStore y los resultados de la verificación aparecerán en la ventana Histórico de verificación. Una vez haya cerrado la ventana Histórico, puede volver a abrirla seleccionando Ver | Histórico de verificación.

Para obtener más información acerca de la utilización del Explorador de DataStore, consulte la *Guía del desarrollador de JDataStore*.

InterBase y JBuilder

InterBase de Borland es una base de datos relacional de alto rendimiento, multiplataforma, compatible con los estándares SQL. InterBase incluye su propia versión de la base de datos de empleados, `employee.gdb`, para que pueda utilizar InterBase en lugar de JDataStore en los ejemplos y tutoriales. Para obtener más información sobre la configuración de Interbase e InterClient para utilizarlos en los tutoriales, consulte [“Conexión con una base de datos mediante los controladores JDBC de InterClient” en la página 4-14](#).

Si desea información adicional acerca de InterBase o para descargar la versión de prueba gratuita de InterBase, consulte el sitio web de Borland en <http://www.borland.com/interbase/index.html>.

Importación y exportación de datos desde un archivo de texto

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Para almacenar los datos que se importan de archivos de texto, en JBuilder se utiliza un componente `TableDataSet`. Cuando los datos se encuentran en el conjunto de datos, se pueden ver y modificar. Para guardar los cambios en el archivo de texto, expórtelos.

Para importar datos de un archivo de texto, utilice el componente `TextDataFile` para proporcionar la ubicación del archivo y los parámetros específicos de su estructura. Utilice un `StorageDataSet`, por ejemplo, un componente `TableDataSet`, para almacenar los datos localmente para poder verlos y editarlos. Los objetos `Column` se crean para que `TableDataSet` sepa el tipo de datos y el nombre del campo de cada columna de datos.

Las columnas de un `TableDataSet` se definen añadiendo columnas en la ventana de código fuente, en el diseñador de interfaces de usuario o cargando un archivo de texto con un archivo SCHEMA (.schema) válido. Este tema trata las dos primeras opciones. Importación de datos mediante un archivo SCHEMA existente se trata en el [Capítulo 16, "Tutorial: Importación y exportación de datos desde un archivo de texto"](#). El archivo de texto sólo tiene un archivo SCHEMA válido si JBuilder lo ha exportado previamente.

Estos son los temas tratados:

- ["Incorporación de columnas a un TableDataSet en el editor" en la página 3-2](#)
- ["Importación de datos con formato desde un archivo de texto" en la página 3-3](#)

- [“Extracción de datos de una fuente de datos JDBC”](#) en la página 3-3
- [“Exportación de datos”](#) en la página 3-4

Incorporación de columnas a un `TableDataSet` en el editor

Se pueden añadir columnas a un `TableDataSet` de dos maneras: De forma visual, en el diseñador de interfaces, o escribiendo código en el editor de la pestaña Fuente. La adición de columnas en el diseñador de interfaces se trata en el [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#). Si anteriormente se exportaron datos a un archivo de texto, JBuilder habrá creado un archivo SCHEMA que proporciona definiciones de columna cuando se abre el archivo de texto, por lo que no es necesario añadir columnas de forma manual.

Para añadir columnas con el editor es necesario definir nuevos objetos `Column` en la definición de clase de `Marco1.java`, como sigue:

- 1 Seleccione `Marco1.java` del panel de contenido y, a continuación, seleccione la pestaña Fuente. La definición de clase aparece en la ventana de código fuente. Añada las siguiente líneas de código:

```
Column column1 = new Column();  
Column column2 = new Column();
```

- 2 Busque el método `jbInit()` en el código fuente. Defina el nombre de la columna y el tipo de datos que se almacenarán en la columna de la forma siguiente:

```
column1.setColumnName("my_number");  
column1.setDataType(com.borland.dx.dataset.Variant.SHORT);
```

```
column2.setColumnName("my_string");  
column2.setDataType(com.borland.dx.dataset.Variant.STRING);
```

- 3 Añada las columnas nuevas a `TableDataSet` en la misma ventana de código fuente y el mismo método `jbInit()`, de la forma siguiente:

```
tableDataSet1.setColumns(new Column[] { column1,column2 } );
```

- 4 Compile la aplicación para asociar los nuevos objetos `Column` al conjunto de datos y, a continuación, añada los componentes visuales que desee.

Importación de datos con formato desde un archivo de texto

Los datos de alguna columna del archivo de texto quizá estén formateados para exportar datos, de modo que impidan su importación correcta. Este problema se puede resolver indicando un modelo de lectura de datos en `exportDisplayMask`. La propiedad `exportDisplayMask` se utiliza para importar datos cuando no hay ningún archivo SCHEMA asociado al archivo de texto. Si existe un archivo SCHEMA, tiene preferencia su configuración. Los modelos de sintaxis se definen en “Edit/display mask patterns” en la *DataExpress Component Library Reference*.

La fecha y el número de columnas tienen visualización y modelos de edición por defecto. Si no se establecen las propiedades, se utilizan los modelos de edición por defecto. Los modelos por defecto proceden del archivo `java.text.resources.LocaleElements` que coincide con los valores por defecto de la columna. Si no hay un local establecido para la columna, se utiliza el local del conjunto de datos. Si tampoco lo hay para el conjunto de datos, se utiliza el del sistema. La visualización por defecto de un número de coma flotante muestra tres cifras decimales. Si desea que haya más cifras decimales, deberá indicarlo mediante una máscara.

Extracción de datos de una fuente de datos JDBC

El código siguiente es un ejemplo de extracción de datos de una fuente JDBC, para su inserción en un `TextDataFile`. Una vez insertados los datos en el `TextDataFile`, puede utilizarse un `StorageDataSet`, por ejemplo un componente `TableDataSet`, para guardar los datos localmente con objeto de visualizarlos y editarlos. Para obtener más información sobre cómo realizar esto, consulte el [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#).

```
Database db = new Database();
db.setConnection(new
    com.borland.dx.sql.dataset.ConnectionDescriptor("jdbc:oracle:thin:@ " +
        datasource, username, password));
QueryDataSet qds = new QueryDataSet();
qds.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(db, "SELECT
    * FROM THETABLE", null, true, Load.ALL));
TextDataFile tdf = new TextDataFile();
tdf.setFileName("THEDATA.TXT");
tdf.save(qds);
```

Este código genera un archivo de datos y su correspondiente archivo SCHEMA.

Este método de acceso a los datos puede emplearse para crear una aplicación de copia de seguridad y restauración de tablas de base de datos que funcione desde la línea de comandos, por ejemplo. Para guardar esta información en una fuente de datos JDBC, consulte [“Guardar los cambios cargados de un TextDataFile en una fuente de datos JDBC” en la página 3-5](#).

Exportación de datos

La exportación de datos o *almacenamiento de datos en un archivo de texto*, guarda todos los datos de la vista actual en un archivo de texto, sobrescribiendo los datos existentes. Este tema trata los distintos modos de exportar datos. Se pueden exportar datos importados desde un archivo de texto al mismo archivo o a otro. Los datos pueden exportarse desde un `QueryDataSet` o un `ProcedureDataSet` a un archivo de texto. O se pueden resolver los datos de `TableDataSet` en una tabla SQL existente.

La exportación de datos a un archivo de texto se gestiona de forma distinta a la resolución de datos en una tabla SQL. Tanto `QueryDataSet` como `TableDataSet` son componentes de `StorageDataSet`. Cuando se suministran datos al conjunto de datos, `StorageDataSet` rastrea la información de estado (eliminada, insertada o actualizada) de todas las filas. Cuando los datos se *resuelven* en una fuente de datos como, por ejemplo, un servidor SQL, la información de estado de fila se utiliza para determinar qué filas de la tabla SQL añadir, borrar o modificar. Cuando una fila se ha resuelto correctamente, obtiene el nuevo estado de resolución `RowStatus.UPDATE_RESOLVED`, `RowStatus.DELETE_RESOLVED`, o `RowStatus.INSERT_RESOLVED`). Si el `StorageDataSet` se resuelve de nuevo, las filas resueltas anteriormente se omiten, a menos que existan cambios posteriores a la resolución. Al *exportar* datos a un archivo de texto, todos los datos de la vista actual se escriben en el archivo y no influyen en la información de estado de la fila.

Los datos exportados a un archivo de texto son sensibles a los criterios de clasificación y filtrado actuales. Si se especifica algún criterio de clasificación, los datos se guardan en el archivo de texto en dicho orden. Si el orden de fila es importante, elimine los criterios de clasificación antes de exportar los datos. Si se especifica algún criterio de filtro, se guardan únicamente los datos que lo cumplen. Esto es útil para guardar subconjuntos de datos en archivos diferentes, pero puede producir pérdidas de datos si se guarda inadvertidamente un archivo filtrado sobre uno existente.

Advertencia Elimine los criterios de filtro si quiere guardar todos los datos en el archivo original.

Exportación de datos de TableDataSet a un archivo de texto

La exportación de datos de un `QueryDataSet` a un archivo de texto es igual que la exportación de datos desde un componente `TableDataSet`, según se explica en el [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#). JBuilder crea un archivo `SCHEMA` que define cada columna, su nombre y tipo de datos para que pueda importarse en JBuilder más fácilmente.

Nota Las columnas BLOB no se exportan; se pasan por alto cuando se exportan los demás campos.

Almacenamiento de cambios desde un TableDataSet en una tabla SQL

Utilice un `QueryResolver` para almacenar los cambios en una tabla SQL. Si desea más información sobre la forma de utilizar `QueryResolver` para guardar cambios en una tabla SQL, consulte [“Personalización de la lógica del almacenador por defecto” en la página 8-17](#).

Antes de resolver los cambios en la tabla SQL, se debe establecer el nombre de la tabla y el de las columnas, como se muestra en el siguiente fragmento de código. La tabla SQL y el archivo `SCHEMA` deben existir. El archivo `SCHEMA` aplicable del `TableDataSet` debe coincidir con la configuración de la tabla SQL. Los tipos de datos variables de las columnas de `TableDataSet` deben corresponderse con los tipos JDBC de la tabla del servidor. Por defecto, todas las filas tienen un estado `INSERT`.

```
tableDataSet1.setTableName(string);
tableDataSet1.SetRowID(columnName);
```

Guardar los cambios cargados de un TextDataFile en una fuente de datos JDBC

Por defecto, los datos se cargan desde un `TextDataFile` con un estado de `RowStatus.Loaded`. Si se llama a un método `saveChanges()` de un `QueryDataSet` o de un `ProcedureDataSet` no se guardarán los cambios efectuados a un `TextDataFile`, ya que todavía no se considera que se han insertado las filas. Para que puedan guardarse los cambios, y que todas las filas cargadas del `TextDataFile` tengan asociado el estado `INSERTED`, asigne a la propiedad `TextDataFile.setLoadAsInserted` el valor `true`. A continuación, cuando llame al método `saveChanges()` de un `QueryDataSet` o `ProcedureDataSet`, los datos volverán a guardarse en la fuente de datos.

Si desea más información sobre la forma de utilizar `QueryResolver` para guardar cambios en una tabla SQL, consulte [“Personalización de la lógica del almacenador por defecto” en la página 8-17](#).

Conexión con bases de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Para poder utilizar los tutoriales de base de datos incluidos en este libro, deberá instalar el controlador JDBC de JDataStore. También puede utilizar el controlador JDBC de InterClient. Aquí se trata la configuración de JDataStore e InterClient para su uso en los tutoriales.

Sun ha colaborado con proveedores de bases de datos y herramientas de base de datos para crear una API independiente del SGBD (Sistema de gestión de bases de datos). De igual forma que ODBC (un producto Microsoft equivalente "grosso modo" a JDBC), JDBC está basado en X/Open SQL Call Level Interface (CLI). Éstas son algunas diferencias entre JDBC y ODBC:

- JDBC es una API escrita totalmente en Java que es realmente independiente de la plataforma. ODBC es una interfaz en lenguaje C que debe implementarse de forma nativa. La mayoría de las implementaciones sólo pueden ejecutarse en plataformas Microsoft.
- La mayoría de los controladores ODBC requieren la instalación de un complejo conjunto de módulos de código y configuración del registro en las estaciones de trabajo cliente. JDBC es una implementación totalmente en Java que puede ejecutarse directamente desde un servidor centralizado, local o remoto. El mantenimiento y la instalación de JDBC son mucho más sencillos que los de ODBC.

JDBC está respaldado por los más importantes proveedores de bases de datos, conectividad y herramientas, incluidos Oracle, Sybase, Informix, InterBase y DB2. Muchos fabricantes, entre ellos Borland, disponen de controladores JDBC. Los controladores ODBC ya existentes pueden utilizarse con el puente JDBC-ODBC suministrado por Sun. El uso del puente JDBC-ODBC no es una solución ideal, puesto que requiere la instalación de controladores ODBC y entradas del registro. Además, los controladores ODBC también se implementan de forma nativa, lo cual

compromete la compatibilidad entre plataformas y la seguridad de las applets.

Los componentes DataExpress JBCL de JBuilder se implementan con la Interfaz de Programación de Aplicaciones (API) de conexión de bases de datos Sun (JDBC). Para crear una aplicación de datos Java, el paquete Sun JDBC `sql` debe estar accesible. Si la conexión con el servidor de la base de datos se realiza mediante un controlador ODBC, también es necesario el software del puente JDBC-ODBC de Sun.

Si desea obtener más información sobre JDBC o el puente JDBC-ODBC puede consultar la página Web "JDBC Database Access API", en <http://java.sun.com/products/jdbc/>.

Conexión con bases de datos

Las aplicaciones JBuilder pueden conectarse con bases de datos SQL, remotas o locales, o con las creadas mediante otras aplicaciones Borland, tales como C++Builder o Delphi.

Para conectar con una base de datos SQL remota, es necesario utilizar uno de los controladores siguientes:

- Un controlador JDBC para el servidor. Algunas versiones de JBuilder incluyen controladores JDBC. Uno de estos controladores es InterClient. Consulte la página web de Borland en <http://www.borland.com/jbuilder/> para obtener información sobre los controladores JDBC que se incluyen en su versión de JBuilder o póngase en contacto con el departamento de servicio técnico de la empresa que le suministre el software de servidor para conseguir controladores JDBC.
- Un controlador de ODBC para el servidor, y emplearlo con el software del puente JDBC-ODBC.

Nota El controlador ODBC es una DLL que no se puede transportar. Resulta adecuado para el desarrollo local, pero no funciona con applets u otras soluciones de Java.

Para conectarse a bases de datos locales que no utilicen SQL como Paradox o Visual dBASE, utilice un controlador ODBC adecuado para el tipo de tabla y el nivel al que accede, en combinación con el software de puente JDBC-ODBC.

Nota Cuando ya no sea necesaria una conexión de base de datos, llame explícitamente al método `Database.closeConnection()` en su aplicación. Con ello se garantiza que la conexión JDBC no permanezca abierta más allá de lo necesario, y se permite que se libere la memoria utilizada por la instancia de esta conexión.

Incorporación de componentes Data Access a la aplicación

Los componentes `Database` son específicos de JDBC y gestionan las conexiones con JDBC. Para acceder a datos utilizando un componente `QueryDataSet` o `ProcedureDataSet`, deberá asignar a la propiedad `DataBase` del componente, un componente `Database` que haya sido instanciado. Una base de datos puede estar compartida por varios conjuntos de datos, que es lo que sucede a menudo.

En una aplicación de base de datos real, normalmente se colocaría el componente `Database` en un módulo de datos. Esto permite que todas las aplicaciones que acceden a la base de datos tengan una conexión común. Si desea más información sobre los módulos de datos, consulte el [Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”](#).

Para añadir el componente `Database` a la aplicación:

- 1 Cree un nuevo proyecto y archivos de aplicación mediante el Asistente para aplicaciones. (También puede usar estas instrucciones para añadir la conectividad de los datos a un proyecto y una aplicación ya existentes.) Para crear archivos de proyecto y aplicación:

- a Seleccione Archivo | Cerrar todo en el menú JBuilder para cerrar todas las aplicaciones.

Si no realiza este paso antes de realizar el siguiente, los nuevos archivos de la aplicación se añadirán al proyecto existente.

- b Elija Archivo | Nuevo y haga doble clic en el icono Aplicación para iniciar el asistente de Aplicaciones.

Acepte o modifique las opciones predeterminadas según sus preferencias.

- 2 Abra el diseñador de interfaces de usuario, resaltando el archivo Marco (por ejemplo, `Marco1.java`) en el panel de contenido, y seleccione la pestaña Diseño en la parte inferior del Visualizador de aplicaciones.



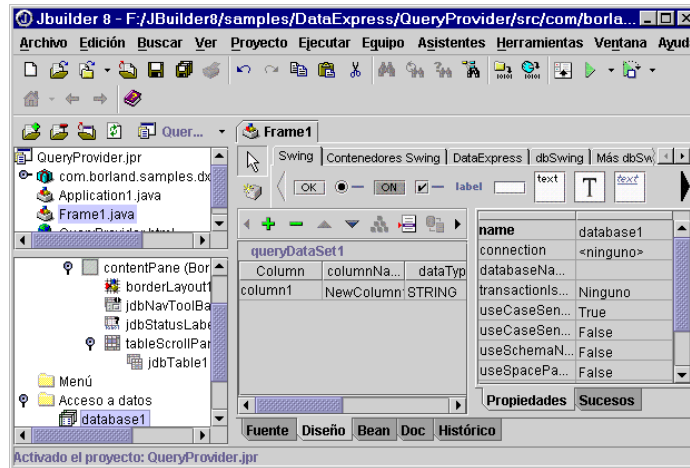
- 3 Abra la pestaña DataExpress de la Paleta de componentes y haga clic en el componente `Database`.
- 4 Haga clic en cualquier parte de la ventana del diseñador para añadir el componente `Database` a la aplicación.

De esta forma, añadirá la siguiente línea de código a la clase Marco:

```
Database database1 = new Database();
```

El componente `Database` aparece en el panel de contenido, con el siguiente aspecto:

Figura 4.1 Componente `Database` abierto en el panel de contenido



Asignación de valores a las propiedades de conexión de una base de datos

La propiedad `Database connection` contiene el controlador JDBC, la URL de conexión, el nombre de usuario y la contraseña. La URL de la conexión JDBC es el método de JDBC que permite especificar la ubicación de un proveedor de datos de JDBC (es decir, el servidor SQL). Contiene toda la información necesaria para efectuar una conexión satisfactoria, incluidos el nombre de usuario y la contraseña.

Se puede acceder al objeto `ConnectionDescriptor` escribiendo código o establecer las propiedades de la conexión mediante el Inspector. Para acceder desde el código al descriptor de conexión (`ConnectionDescriptor`), siga estas instrucciones:

- Si se asigna a `promptPassword` el valor `true`, también se debe llamar a `openConnection()` para la base de datos. `openConnection()` determina cuándo se muestra el cuadro de diálogo de solicitud de contraseña y cuándo se efectúa la conexión con la base de datos.
- Obtenga los datos de nombre de usuario y contraseña en cuanto se abra la aplicación. Para ello, llame a `openConnection()` al final del método `jbInit()` del marco principal.

Si la conexión no se abre expresamente, intenta abrirse cuando un componente o un conjunto de datos requiere datos.

A continuación, se describe la forma de asignar valores a las propiedades de conexión, al ejemplo JDataStore Employed, por medio del diseñador de interfaces de usuario.

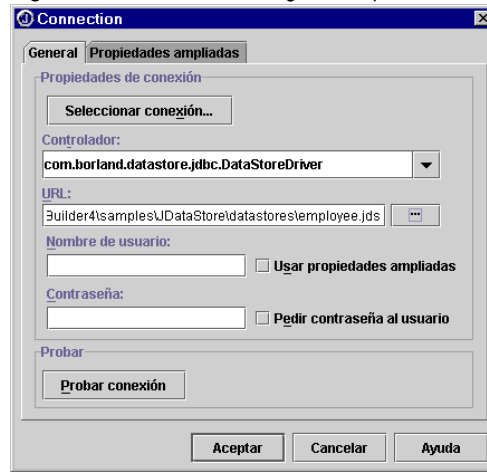
Nota Para utilizar la base de datos de ejemplo, se debe empezar por comprobar que el sistema está correctamente configurado para JDataStore. Si aún no lo ha hecho, consulte [“Configuración de JDataStore” en la página 4-7](#).

- 1 En el árbol de componentes, seleccione `database1`.
- 2 En el Inspector, seleccione el valor de la propiedad `connection` y haga clic en el botón de puntos suspensivos para abrir el editor de la propiedad `Connection`.
- 3 Asigne valores a las siguientes propiedades:

Propiedad	Descripción
Controlador	El nombre de clase del controlador JDBC que corresponde a la URL; para este ejemplo, seleccione <code>com.borland.datastore.jdbc.DataStoreDriver</code> en la lista.
URL	El buscador universal de recursos (dirección URL) de la base de datos, para este ejemplo. El valor predeterminado es <code>jdbc:borland:dslocal:directoryAndFile.jds</code> . Pulse el botón Examinar para seleccionar la base de datos local de JDataStore, ubicada en <code>/<jbuilder>/samples/JDataStore/datastores/employee.jds</code> . Utilice el botón Examinar para buscar este archivo y reducir la posibilidad de teclear un nombre erróneo. Cuando la URL esté seleccionada presentará el siguiente aspecto: Unix: <code>jdbc:borland:dslocal:/usr/local/<jbuilder>/samples/JDataStore/datastores/employee.jds</code> Windows: <code>jdbc:borland:dslocal:C:\jbuilder\samples\JDataStore\datastores\employee.jds</code>
Nombre de usuario	Introduzca el nombre de usuario autorizado para acceder a la base de datos del servidor. En los tutoriales de ejemplo funciona cualquier nombre de usuario.
Contraseña	La contraseña de usuario autorizado. No se necesita ninguna contraseña para los tutoriales.
Pedir contraseña al usuario	Indica si se debe pedir una contraseña al usuario al abrir una conexión de base de datos.

El cuadro de diálogo tendrá un aspecto parecido a éste:

Figura 4.2 Cuadro de diálogo Descriptor de conexiones



- 4 Haga clic en botón Probar conexión para comprobar que las propiedades de conexión se han establecido correctamente.

Los resultados del intento de conexión se muestran justo debajo del botón Probar conexión.

Si JDataStore se ha instalado con JBuilder, se pedirán el número de serie y la contraseña la primera vez que se utilice. Si aún no tiene disponible la información de JDataStore, puede introducirla más adelante. Para ello, elija Herramientas | Explorador de JDataStore y seleccione Administrador de licencias del menú Archivo.

Si no consigue conectarse con la base de datos de ejemplo, asegúrese de que ha elegido el uso de la base de datos de ejemplo JDataStore. Consulte [“Configuración de JDataStore” en la página 4-7](#) para obtener más información.

- 5 Pulse Aceptar para salir del cuadro de diálogo Conexión y guardar las propiedades de la conexión en el código fuente, cuando la conexión sea satisfactoria.

El código fuente, si ha seguido este ejemplo, será parecido a esto:

```
database1.setConnection(new
com.borland.dx.sql.dataset.ConnectionDescriptor(
"jdbc:borland:dslocal:
<jbuilder>/samples/JDataStore/datastores/employee.jds",
, , false, "com.borland.datastore.jdbc.DataStoreDriver"));
```

- 6 Seleccione un componente DBDisposeMonitor de la pestaña Más dbSwing y haga clic en el panel de contenido para añadirlo a la aplicación.

El componente DBDisposeMonitor cierra el JDataStore cuando se cierre la ventana.

- 7 Asigne a la propiedad `dataAwareComponentContainer` de `DBDisposeMonitor` el valor `this`.

Sugerencia Una vez establecida correctamente una conexión a la URL de una base de datos, utilice el Explorador de bases de datos para recorrer la información de metabases de datos JDBC y objetos de esquema de base de datos en el JDataStore, así como para ejecutar sentencias en SQL, y localizar y editar datos de tablas existentes.

Configuración de JDataStore

El Explorador de JDataStore permite ver y examinar el contenido del archivo JDataStore. Para iniciar el Explorador de JDataStore, seleccione Herramientas | Explorador de JDataStore. Para abrir el ejemplo de JDataStore, busque el archivo `<jbuilder>/samples/JDataStore/datastores/employee.jds`.

Si desea más información sobre la utilización del Explorador de DataStore, consulte la *Guía del desarrollador de JDataStore*.

Configuración de InterBase e InterClient

InterBase es un programa de fácil manejo para la gestión de bases de datos relacionales acordes con SQL. InterBase es independiente de los clientes y las herramientas, y acepta la mayoría de los entornos cliente y de creación de aplicaciones.

InterClient es un controlador JDBC escrito totalmente en Java para bases de datos de InterBase. InterClient contiene una biblioteca de clases Java que implementan la mayor parte de la API de JDBC y un conjunto de extensiones para la API de JDBC. Este paquete interactúa con JDBC Driver Manager y permite que las aplicaciones y applets Java del lado cliente interactúen con las bases de datos InterBase.

InterClient incluye un controlador en el lado servidor, llamado *InterServer*. Este software intermedio del lado servidor cumple la función de traductor entre los clientes basados en InterClient y el servidor de base de datos InterBase. Incluye las clases de desarrollo de aplicaciones de Java y un kit de desarrollo de servidores web.

Los desarrolladores pueden distribuir los clientes InterClient de dos maneras:

- Las *applets Java* son programas Java que pueden incluirse en una página HTML con la etiqueta `<APPLET>`, que se conectan al servidor a través de un servidor web y se visualizan y utilizan en un sistema cliente que usa un visualizador web que admite Java. Este método de distribución no requiere la instalación manual del paquete InterClient en el sistema

cliente. No obstante, sí precisa que el sistema cliente cuente con un visualizador que interprete Java.

- Las *aplicaciones Java* son programas Java independientes que se ejecutan en sistemas cliente. Este método de instalación requiere el paquete InterClient, así como tener instalado Java Runtime Environment (JRE) en el sistema cliente. JRE incluye el JDBC Driver Manager.

InterClient, como API escrita totalmente en Java para InterBase, admite el desarrollo cliente-servidor independiente de la plataforma para intranets empresariales e Internet. La ventaja de un controlador creado íntegramente en Java, frente a un controlador en código nativo, es que el primero permite publicar applets basados en InterClient sin tener que cargar manualmente los controladores JDBC de cada plataforma concreta en cada sistema cliente (los servidores web transfieren automáticamente las clases InterClient junto con las applets). Por lo tanto, se elimina la necesidad de gestionar bibliotecas de base de datos nativas locales, lo que simplifica la gestión y el mantenimiento de las aplicaciones de los clientes. Como parte de un applet Java, InterClient puede actualizarse de forma dinámica, lo que reduce aún más el coste de la distribución y el mantenimiento de las aplicaciones.

Uso de InterBase e InterClient con JBuilder

Para utilizar InterBase e InterClient con JBuilder, instale InterBase e InterClient siguiendo las instrucciones que aparecen en la documentación de InterBase, inicie el InterBase Server y, posteriormente, el InterServer de InterClient.

Si tiene problemas en la conexión, compruebe que tanto la base de datos de InterBase como InterServer se están ejecutando. InterServer y la base de datos se pueden ejecutar en el mismo equipo que la aplicación o en otro distinto. Por ello, hay varias configuraciones posibles. Es importante que la versión de InterClient sea compatible con la versión de la base de datos y JDK. Si desea más información sobre estos asuntos, consulte la documentación de InterBase e InterClient.

Si el servidor InterBase e InterServer se encuentran en una plataforma distinta de la de JBuilder, haga lo siguiente:

- Compruebe que InterBase e InterServer se están ejecutando en el servidor.
- Compruebe que InterClient se encuentra instalado en el cliente.
- Asegúrese de que la URL de `ConnectionDescriptor` en el cliente tiene la dirección IP correcta del servidor que ejecuta InterBase e InterServer.

Una vez instalado InterClient, añádalo a JBuilder con Herramientas | Configurar Enterprise; a continuación, añádalo a su lista de bibliotecas necesarias del proyecto en Proyecto | Propiedades. Para obtener más

información, consulte [“Adición de un controlador JDBC a JBuilder” en la página 4-10.](#)

Sugerencias para la utilización de las tablas de ejemplo de InterBase

Para obtener los mejores resultados, tenga en cuenta los siguientes consejos al utilizar las tablas de ejemplo:

- Realice una copia de seguridad de las bases de datos de ejemplo.

El programa de instalación instala bases de datos de ejemplo. Es recomendable realizar una copia de la base de datos de ejemplo `employee.jds` para poder restaurar con facilidad el archivo a su estado original después de experimentar con programaciones de la base de datos.

- Respete las restricciones de la base de datos.

Las bases de datos de ejemplo imponen numerosas restricciones a los valores de los datos, como es normal en una aplicación real. Estas restricciones afectan a todos los ejemplos en los que se añadan, inserten o actualicen datos de la tabla de empleados y en los que se intente guardar los cambios en la tabla del servidor.

- En los ejemplos de este manual se utiliza frecuentemente la tabla EMPLOYEE. Las siguientes restricciones se aplican a la tabla de empleados:
 - Todos los campos son necesarios (debe introducirse algún dato), excepto PHONE_EXT.
 - EMP_NO es un campo generado, por lo que no es necesario introducir ningún valor en cada nuevo registro. Además, ésta es la clave primaria, así que no debe modificarse.
 - Integridad referencial.
 - DEPT_NO debe existir en la tabla Department.
 - JOB_CODE, JOB_GRADE y JOB_COUNTRY deben existir en la tabla JOB.
 - SALARY debe ser mayor o igual que el valor del campo min_salary de la tabla job correspondiente a los campos asociados job_code, job_grade y job_country de la tabla job.
 - FULL_NAME es un valor generado por la consulta, por lo que no es necesario introducir ningún dato.
- La tabla CUSTOMER se utiliza también en los tutoriales sobre bases de datos. CUST_NO es un valor generado, por lo que no es necesario introducirlo para cada nuevo registro.

Cuando se utilizan las tablas de empleados, es más seguro modificar sólo los campos LAST_NAME, FIRST_NAME, PHONE_EXT de los registros existentes.

Para ver los metadatos de las tablas de ejemplo:

- 1 Seleccione Herramientas | Explorador de bases de datos.
El Explorador de bases de datos se utiliza en tareas de administración de bases de datos.
- 2 Para abrir una conexión con la base de datos, haga doble clic en su dirección URL.
- 3 Expanda el nodo Tablas para ver información acerca de cada una de las tablas de ejemplo.

Adición de un controlador JDBC a JBuilder

Tras instalar su controlador JDBC siguiendo las instrucciones del fabricante, siga los pasos que a continuación se indican, para configurarlo de cara a su utilización con JBuilder.

Nota Los controladores que aparecen en rojo en la Lista de controladores del cuadro de diálogo Propiedades de conexión no están instalados, por lo que no pueden ser seleccionados para su uso en JBuilder. Para poder configurarlos en JBuilder, debe instalarlos conforme a las instrucciones del fabricante.

Creación de los archivos .library y .config

Son tres los pasos para añadir un controlador de base de datos a JBuilder:

- Crear un archivo de biblioteca que contenga las clases de controladores, generalmente un archivo JAR y algún archivo adicional como los de documentación o código fuente.
- Derivar un archivo .config del archivo de biblioteca que JBuilder añade a su vía de acceso a clases al iniciarse.
- Añadir una nueva biblioteca al proyecto o, si desea que esté disponible para todos los nuevos proyectos, al proyecto por defecto.

Los dos primeros pasos pueden realizarse en un mismo cuadro de diálogo:

- 1 Abra JBuilder y seleccione Herramientas | Configurar Enterprise.
- 2 Abra la pestaña Controladores de bases de datos del cuadro de diálogo Configurar Enterprise.

La pestaña Controladores de base de datos muestra los archivos .config de todos los controladores de bases de datos actualmente definidos.

- 3 Haga clic en Añadir para añadir un controlador, pulse Nuevo para crear un archivo de biblioteca para el controlador. El archivo de biblioteca se utiliza para añadir un controlador a la lista de bibliotecas necesarias de los proyectos.

Nota También puede crear una biblioteca desde Herramientas | Configurar bibliotecas pero, como después tendría que utilizar Configurar Enterprise para obtener el archivo .config, es más sencillo hacerlo todo desde aquí.

- 4 Escriba un nombre y seleccione una ubicación para el nuevo archivo en el cuadro de diálogo Crear biblioteca.
- 5 Haga clic en Añadir y desplácese hasta la ubicación del controlador. Puede seleccionar el directorio que contiene el controlador y todos sus archivos de apoyo o puede, simplemente, seleccionar el archivo recopilatorio del controlador. Cualquiera de ambas opciones funcionará. JBuilder extrae la información necesaria.
- 6 Pulse Aceptar para cerrar el visualizador de archivos. Esto muestra la nueva biblioteca en la última posición de la lista de bibliotecas y la selecciona.
- 7 Pulse Aceptar. JBuilder crea un archivo de biblioteca en el directorio /lib de JBuilder con el nombre que se le dio (por ejemplo, `InterClient.library`). Le devuelve a la ficha Controladores de bases de datos que muestra el nombre del archivo .config correspondiente que se derivará del archivo de biblioteca (por ejemplo, `InterClient.config`).
- 8 Seleccione el nuevo archivo .config en la lista de controladores de bases de datos y haga clic en Aceptar. Esto sitúa el archivo .config en el directorio /lib/ext de JBuilder.
- 9 Cierre y reinicie JBuilder para que los cambios a los controladores de bases de datos tengan efecto y el nuevo controlador se ubique en la vía de acceso a clases de JBuilder.

Importante Si se modifica el archivo .library tras la derivación del archivo.config habrá de volver a generar este último utilizando Configurar Enterprise y reiniciar JBuilder.

Adición del controlador JDBC a proyectos

Los proyectos que se ejecutan desde dentro de JBuilder sólo utilizan su propia ruta de acceso a clases. Por lo tanto, para asegurarse de que el controlador JDBC está disponible para los nuevos proyectos que lo necesiten, defina la biblioteca y añádala a la lista por defecto de bibliotecas necesarias. Esto se hace desde JBuilder, de la siguiente forma:

- 1 Inicie JBuilder y cierre los proyectos abiertos.
- 2 Elija Proyecto | Propiedades de proyecto por defecto.
- 3 Seleccione la pestaña Bibliotecas necesarias de la ficha Vías de acceso y pulse Añadir.
- 4 Seleccione el nuevo controlador JDBC de la lista de bibliotecas y pulse Aceptar.
- 5 Haga clic en Aceptar para cerrar el cuadro de diálogo Propiedades del proyecto por defecto.

Nota También puede añadir el controlador JDBC a un proyecto existente. Simplemente abra el proyecto, seleccione Proyecto | Propiedades y utilice el procedimiento arriba descrito.

En este punto, JBuilder y el nuevo controlador JDBC están configurados para trabajar conjuntamente. El paso siguiente es crear o abrir un proyecto que utilice este controlador, añadirle un componente DataBase y configurar la propiedad `connection` para que pueda utilizar ese controlador para acceder a los datos. Para ver un ejemplo de cómo hacer esto, consulte [“Conexión con una base de datos mediante los controladores JDBC de InterClient” en la página 4-14](#).

El componente `Database` gestiona la conexión JDBC con un servidor SQL y es obligatorio en todas las aplicaciones de base de datos que acceden a datos de servidores. JDBC es la Interfaz de programación de aplicaciones de base de datos Sun, una biblioteca de componentes y clases desarrollada por Sun para poder acceder a fuentes de datos remotas. Los componentes se recopilan en el paquete `java.sql` y representan un marco de acceso a bases de datos SQL genérico y de bajo nivel.

El API de JDBC define clases Java que representan conexiones de base de datos, sentencias SQL, conjuntos resultado, metadatos de base de datos, etc. Permite a los programadores de Java emitir sentencias SQL y procesar el resultado. JDBC es la API principal para el acceso a bases de datos en Java. La API de JDBC se implementa mediante un gestor de controladores que admite la conexión de varios controladores con diferentes bases de datos. Para obtener más información sobre JDBC puede consultar la página web "Sun JDBC Database Access API", en <http://java.sun.com/products/jdbc/>.

JBuilder utiliza la API de JDBC para acceder a la información almacenada en bases de datos. Muchos de los componentes y clases de acceso a datos de JBuilder utilizan la API de JDBC. Por lo tanto, se debe instalar correctamente estas clases para poder utilizar los componentes de conectividad con base de datos de JBuilder. Además, se necesita un controlador de JDBC apropiado para conectar la aplicación de base de datos con un servidor remoto. Los controladores pueden agruparse en dos categorías principales: controladores implementados mediante métodos nativos que hacen de puente con bibliotecas de acceso a base de datos ya

existentes o controladores completamente basados en Java. Los controladores que no son 100% Java deben ejecutarse en el sistema cliente (local). Los controladores que sí lo son pueden cargarse desde el servidor o de forma local. Las ventajas de utilizar un controlador escrito completamente en Java son que puede descargarse como parte de un applet y que es independiente de la plataforma.

Algunas versiones de JBuilder incluyen controladores JDBC. Consulte la página Web de Borland en <http://www.borland.com/jbuilder/> para obtener información sobre los controladores JDBC que se incluyen en las versiones de JBuilder o póngase en contacto con el departamento de servicio técnico de la empresa que le suministre el software de servidor para conseguir controladores JDBC. Estos son algunos de los controladores que pueden venir incluidos en JBuilder:

- **DataStoreDriver**

DataStoreDriver es el controlador JDBC asociado a la base de datos JDataStore. Este controlador permite tanto acceso local como remoto. Ambos tipos de acceso requieren un nombre de usuario.

Nota

El nombre de usuario puede ser cualquier cadena. Si no se ha añadido ningún usuario a la tabla de usuarios del almacén, no hay necesidad de proporcionar uno. No obstante, en los almacenes transaccionales se requiere una cadena no nula como nombre de usuario.

Si desea más información sobre la conexión con bases de datos con el controlador DataStore, consulte el [Capítulo 17, “Tutorial: Creación de aplicaciones de base de datos distribuidas”](#).

- **InterClient**

InterClient es un controlador JDBC que se puede utilizar para conectarse con InterBase. InterClient se puede instalar por medio del programa de instalación InterClient. Una vez instalado, InterClient puede acceder a los datos InterBase de ejemplo, por medio de `ConnectionDescriptor`.

Si desea más información sobre la conexión a bases de datos con InterClient, consulte [“Conexión con una base de datos mediante los controladores JDBC de InterClient”](#) en la [página 4-14](#).

Las aplicaciones JBuilder pueden conectarse con bases de datos SQL, remotas o locales, o con las creadas mediante otras aplicaciones Borland, tales como C++Builder o Delphi. Para ello, averigüe con qué base de datos subyacente se conecta la aplicación y conéctese con ella por medio de su URL de conexión.

Conexión con una base de datos mediante los controladores JDBC de InterClient

En este apartado se explica cómo añadir un componente `Database`, es decir, un componente JDBC que gestiona una conexión JDBC, y cómo configurar las propiedades de este componente para poder acceder a una base de datos InterBase de ejemplo.

En una aplicación de base de datos real, normalmente se colocaría el componente `Database` en un módulo de datos. Esto permite que todas las aplicaciones que acceden a la base de datos tengan una conexión común. Si desea más información sobre los módulos de datos, consulte el [Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”](#).

Para añadir el componente `Database` a una aplicación y conectarse los archivos InterBase de ejemplo:

- 1 Siga los pasos de [“Configuración de InterBase e InterClient” en la página 4-7](#) y [“Adición de un controlador JDBC a JBuilder” en la página 4-10](#) para que el sistema esté configurado correctamente para acceder a los archivos InterBase de ejemplo.
- 2 Compruebe que InterServer se está ejecutando.
- 3 Cierre todos los proyectos, y cree una aplicación, o añada conectividad de datos a un proyecto y una aplicación.

Seleccione Archivo | Nuevo y haga doble clic en el icono Aplicación para crear un proyecto nuevo y archivos de aplicación. Acepte todas las opciones por defecto. JBuilder crea los archivos necesarios y los muestra en el panel de proyectos del Visualizador de aplicaciones. El archivo `Marco1.java` está abierto en el panel de contenido. `Marco1.java` contiene los componentes de la interfaz de esta aplicación.

- 4 Haga clic en la pestaña Diseño en `Marco1.java` en la parte inferior del panel de contenido.
- 5 Abra la pestaña Data Express de la Paleta de componentes y haga clic en el componente `Database`.
- 6 Haga clic en cualquier lugar del panel de contenido o en el diseñador de interfaces, para añadir el componente `Database` a la aplicación.
- 7 Configure la propiedad `connection` de `Database` a fin de especificar la URL, el nombre de usuario, la contraseña y los controladores JDBC.

La URL de la conexión JDBC es el método de JDBC que permite especificar la ubicación de un proveedor de datos de JDBC (es decir, el servidor SQL). En realidad puede contener toda la información necesaria para efectuar una conexión satisfactoria, incluidos el nombre de usuario y la contraseña.



Para asignar valores a la propiedad `connection`:

- a Compruebe que el objeto `Database` se encuentra seleccionado en el panel de contenido.

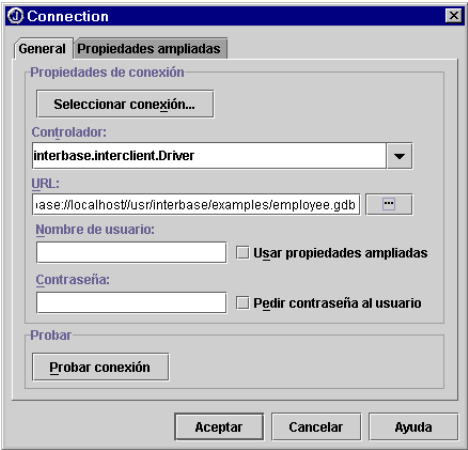
Haga doble clic en la propiedad `connection` en la ventana del Inspector, para abrir el editor de la propiedad `connection`. En este ejemplo, los datos se encuentran en un servidor InterBase local. Si estuviesen en un servidor remoto, habría que introducir la dirección IP del servidor en vez del parámetro “localhost”.

- b Asigne valores a las siguientes propiedades:

Propiedad	Valor
Controlador	<code>interbase.interclient.Driver</code>
URL	Busque el archivo de ejemplo de InterBase, <code>employee.gdb</code> , que se encuentra en el directorio <code>/examples</code> de InterBase. El contenido del campo URL tiene un aspecto parecido al siguiente: UNIX: <code>jdbc:interbase://localhost//usr/interbase/examples/employee.gdb</code> Windows: <code>jdbc:interbase://localhost/D:\InterBaseCorp\InterBase\examples\database\employee.gdb</code>
Nombre de usuario	<code>SYSDBA</code>
Contraseña	<code>masterkey</code>

El cuadro de diálogo tendrá un aspecto parecido a éste:

Figure 4.3 Cuadro de diálogo Conexión



- c Haga clic en botón `Probar conexión` para comprobar que las propiedades de conexión se han establecido correctamente.

El resultado del intento de conexión se muestra justo debajo del botón Probar conexión. En [“Mensajes de error de conexión habituales” en la página 4-23](#) puede encontrar soluciones a algunos problemas de conexión habituales.

- d Pulse Aceptar para salir del cuadro de diálogo y guardar las propiedades de la conexión en el código fuente cuando la conexión sea satisfactoria.

Sugerencia Una vez establecida correctamente una conexión a la URL de una base de datos, utilice el Explorador de bases de datos para recorrer la información de metabases de datos JDBC y objetos de esquema de base de datos, así como para ejecutar sentencias en SQL, o localizar y editar datos de tablas existentes.

Utilización de componentes Database en las aplicaciones

Después de que la aplicación incluya el componente `Database` se puede añadir otro componente `DataExpress` que recupere datos de la fuente de datos con la que se ha establecido la conexión. JBuilder utiliza consultas y procedimientos almacenados para obtener un conjunto de datos. Los componentes implementados para este fin son `QueryDataSet` y `ProcedureDataSet`. Estos componentes cooperan con el componente `Database` para obtener acceso a la base de datos del servidor SQL. Para obtener instrucciones sobre cómo utilizar estos componentes, consulte los siguientes apartados:

- [“Consultas en bases de datos” en la página 5-2](#)
- [“Utilización de consultas parametrizadas para obtener datos” en la página 5-13](#)
- [Capítulo 6, “Utilización de procedimientos almacenados”](#)

La mayoría de las aplicaciones de ejemplo y tutoriales utiliza una conexión `Database` con el archivo `JDataStore` de ejemplo `EMPLOYEE`, como se describe aquí. Algunas aplicaciones de ejemplo y tutoriales, sobre todo los que utilizan procedimientos almacenados para recuperar o guardar datos, utilizan una conexión con la base de datos de empleados `InterClient` a través del controlador `JDBC` de `InterClient`.

En la mayoría de las aplicaciones de base de datos se encapsulan los componentes `Database` y otros componentes `DataExpress` en un módulo de datos, en lugar de añadirlos directamente al marco de la aplicación. Para obtener más información acerca de la utilización del `DataModule` del paquete `DataExpress`, consulte [Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”](#).

Solicitud del nombre de usuario y la contraseña

Cuando se programan aplicaciones de base de datos es conveniente incluir un nombre de usuario y una contraseña en `ConnectionDescriptor` para que no sea necesario facilitarlos cada vez que se utiliza el Diseñador o se ejecuta la aplicación. Si `ConnectionDescriptor` se configura por medio del Diseñador, éste escribe el código automáticamente. Antes de distribuir una aplicación, es conveniente borrar del código el nombre de usuario y la contraseña y solicitarlos al usuario durante la ejecución, sobre todo si se distribuye el código fuente o si hay distintos usuarios con distintos derechos de acceso. Existen varias formas de pedir el nombre de usuario y la contraseña en el momento de la ejecución:

- Active la casilla de verificación Pedir contraseña al usuario de la propiedad `connection` de `Database`, en el editor, o escriba el código necesario para asignar a la propiedad `promptPassword` de `ConnectionDescriptor` el parámetro `true`.

En tiempo de ejecución, y cuando se muestran datos dinámicos en el diseñador, aparece un cuadro de diálogo de nombre de usuario y contraseña. Para que se presenten los datos es necesario introducir el nombre y la contraseña.

- Añada a la aplicación una instancia de `DBPasswordPromppter`, de `dbSwing`.

Esta opción proporciona un mayor control de la gestión del nombre de usuario y la contraseña. Es posible indicar la información requerida (el nombre de usuario, la contraseña o las dos cosas), cuántas veces puede intentar el usuario introducir la información y otras propiedades. El botón Aceptar se muestra atenuado hasta que se introduzca la información necesaria. El cuadro de diálogo se abre cuando se llama a su método `showDialog()`. Esto permite controlar el momento en que aparece. No olvide presentarlo en la aplicación antes de que otro componente visual intente abrir la base de datos y mostrar datos. El Diseñador no llama a `showDialog()`, por lo que es necesario indicar el nombre de usuario y la contraseña `ConnectionDescriptor` cuando se activa.

Agrupación de conexiones JDBC

Las aplicaciones que precisan un gran número de conexiones con bases de datos pueden beneficiarse de la agrupación de conexiones. Esta agrupación supone una mejora considerable del rendimiento, especialmente en aquellos casos en los que se han de abrir y cerrar un gran número de conexiones de bases de datos.

`JDataStore` incluye varios componentes que permiten trabajar con `DataSources` y conexiones de `JDBC 2.0`. Para utilizar estos componentes es

necesario J2EE. Si su versión de JBuilder no incluye el archivo J2EE.jar, deberá obtenerlo en la web de Sun y añadirlo a su proyecto como biblioteca necesaria. Consulte [“Incorporación de una biblioteca necesaria a un proyecto” en la página 10-6](#) para obtener instrucciones sobre cómo añadir una biblioteca necesaria.

El principal propósito de los grupos de conexiones es muy sencillo. En una aplicación en la que se abren y cierran muchas conexiones con bases de datos resulta eficiente mantener en un grupo los objetos `Connection` que queden libres para poder reutilizarlos posteriormente. De esta manera, desaparece la necesidad de abrir una nueva conexión física cada vez que se abre una conexión.

Los principales componentes de agrupación de conexiones y de `DataSource` proporcionados por un `JDataStore` son los siguientes:

- `JDBCDataSource` es una implementación de la interfaz `javax.sql.DataSource`. `JDBCDataSource` puede crear una conexión con un `JDataStore` o con cualquier otro controlador JDBC, según sus propiedades de conexión JDBC; sin embargo, no realiza agrupamiento de conexiones. Dado que es una implementación de `javax.sql.DataSource` se puede registrar en un servicio de denominación JNDI. Si desea obtener más información sobre los servicios de denominación JNDI, consulte la documentación del Kit de Desarrollo de Java (JDK) o la página web <http://www.javasoft.com>.
- `JDBCConnectionPool` es también una implementación de `javax.sql.DataSource` y, por lo tanto, también puede registrarse en un servicio de denominación JNDI. `JDBCConnectionPool` puede utilizarse para agrupar conexiones con cualquier controlador JDBC, ya que crea conexiones según las propiedades de conexión JDBC que tenga establecidas. `JDBCConnectionPool` cuenta con diferentes propiedades para la gestión de agrupación de conexiones como, por ejemplo, aquellas que especifican un número mínimo y máximo de conexiones.

Para utilizar `JdbcConnectionPool` es preciso asignar un valor a la propiedad `connectionFactory`. Esto permite a `JdbcConnectionPool` crear objetos `javax.sql.PooledConnection`. El valor de la propiedad `connectionFactory` debe hacer referencia a una implementación de `javax.sql.ConnectionPoolDataSource` (como `JdbcConnectionFactory`). La propiedad `connectionFactory` puede recibir igualmente un valor mediante la propiedad `dataSourceName`. Esta propiedad `dataSourceName` obtiene un `String`, que consultará en el servicio de denominación y así obtener la implementación del `javax.sql.ConnectionPoolDataSource`.

Para obtener una conexión del grupo, por lo general se llamará a `JdbcConnectionPool.getConnection()`. La conexión que devuelve este método no admite transacciones distribuidas, pero es compatible con cualquier controlador JDBC.

`JDBCConnectionPool` también permite las transacciones distribuidas (XA), sin embargo, esta característica sólo está disponible cuando `JDBCConnectionPool` se utiliza junto con el controlador JDBC `JDataStore`, y únicamente resulta útil utilizado a la par con un gestor de transacciones distribuidas, como Borland Enterprise Server. Para obtener más información sobre el soporte XA de `JDBCConnectionPool`, consulte el apartado “Agrupación de conexiones y soporte de transacciones distribuidas” en la *Guía del Desarrollador de JDataStore*.

- `JdbcConnectionFactory` es una implementación de `javax.sql.ConnectionPoolDataSource`. Se utiliza para crear objetos `javax.sql.PooledConnection` para una implementación de un agrupamiento de conexiones semejante a `JDBCConnectionPool`.

`JDBCConnectionPool` y `JDBCConnectionFactory` conjuntamente, pero también se pueden emplear de manera independiente. Esta última opción ofrece mayor flexibilidad. Por ejemplo, `JDBCConnectionFactory` puede utilizarse con un componente de agrupación de conexiones que utilice una estrategia diferente a la de `JDBCConnectionPool`. `JDBCConnectionFactory` puede funcionar con cualquier implementación de grupos de conexiones de JDBC 2.0 que permita que una implementación de `javax.sql.ConnectionPoolDataSource` (como `JDBCConnectionFactory`) sea quien le proporcione las conexiones `javax.sql.PooledConnections`.

Por otra parte, la eficiente estrategia de agrupación de `JDBCConnectionPool` puede utilizarse con otra implementación de factoría de conexiones. `JDBCConnectionPool` puede utilizarse con cualquier controlador JDBC que proporcione un componente de la `connection factory` que implemente `javax.sql.ConnectionPoolDataSource`.

Y ahora que ya se han visto las clases que forman parte de la agrupación de conexiones se analizará su funcionamiento más a fondo:

- La función del método `JdbcConnectionPool.getConnection()` es eliminar la necesidad de abrir una nueva conexión utilizando una que ya forme parte del grupo. Cuando se busca una conexión dentro del grupo, aparecerá una correspondencia si el nombre de usuario coincide con el utilizado al crear la conexión. La contraseña no se tiene en cuenta al buscar un usuario coincidente. La factoría solicitará la creación de una conexión solamente si no se encuentra ninguna correspondencia con el grupo.
- La agrupación de conexiones constituye una API relativamente sencilla, pero muy potente. La parte más difícil, como controlar las conexiones agrupadas y decidir si conviene utilizar una conexión o crearla, se hace enteramente de forma interna.
- Las aplicaciones que utilizan conexiones en grupo deben cerrarlas, siempre que no dejen de utilizarse. Así, pueden volver a utilizarse y mejora el rendimiento.

- La factoría que crea las conexiones para el grupo debe utilizar la misma configuración de propiedades en todas las conexiones, excepto el nombre de usuario y la contraseña. Por lo tanto, cada grupo de conexiones accede a una sola base de datos y todas las propiedades de conexión JDBC de todas las conexiones que componen el grupo tienen los mismos valores (aunque pueden tener nombres de usuario/contraseñas diferentes).

Optimización del rendimiento de JConnectionPool

El mecanismo de búsqueda para encontrar en el grupo una conexión con el mismo nombre de usuario consiste en una comparación rápida de las referencias a las cadenas de variables del nombre de usuario. Si es posible, pase la misma instancia de esa variable `String` en todas las peticiones de conexión. Una forma de hacerlo es utilizar siempre el mismo nombre para las conexiones en grupo, declarado como sigue:

```
public static final String POOL_USER = "CUSTOMER_POOL_USER";
```

Impresión del histórico de compresión

Tanto `JdbcConnectionPool` como `JdbcConnectionFactory` tienen propiedades `PrintWriter`. La mayor parte de las impresiones del histórico de compresión tienen la siguiente forma:

```
[<clase instancia código hash>]:<nombre de clase>.<nombre de método>(...)
```

Los valores hexadecimales que aparezcan entre (`[]`) en los archivos del histórico son valores `hashCode()` de un `Object`.

Ejemplo de agrupación

Lo que sigue es un ejemplo trivial de la utilización de conexiones en grupo. Este fragmento de código del módulo de datos muestra las líneas de código más importantes e imprescindibles en una aplicación que utilice conexiones agrupadas, sin que resulte necesario hacer demasiadas suposiciones sobre lo que dicha aplicación específica hará con esa tecnología. Si desea ver un ejemplo más serio de las conexiones en grupo, consulte el ejemplo `Web Bench` que se encuentra en `samples/JDataStore/WebBench`. Para obtener más información acerca de los módulos de datos, consulte el [Capítulo 10, “Utilización de módulos de datos para simplificar el acceso a los datos”](#).

```
import com.borland.dx.dataset.*;
import com.borland.dx.sql.dataset.*;
import com.borland.javax.sql.*;
import java.sql.*;

public class DataModule1 implements DataModule{
```



```

private static DataModule1 myDM;
private static final String POOL_USER = "POOL_USER";
private static JdbcConnectionFactory factory;
private static JdbcConnectionPool pool;

public DataModule1() {
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception{
    // Instancie una factoría de conexión
    factory = new JdbcConnectionFactory();

    // La siguiente línea asigna a la dirección URL un
    // archivo JdataStore local. La URL específica
    // dependerá de la ubicación
    // del archivo de JDataStore.
    factory.setUrl("jdbc:borland:dslocal:<vía de acceso><nombre de archivo>");
    factory.setUser(POOL_USER);
    factory.setPassword("");

    // Instancie el agrupamiento de conexiones
    pool = new JdbcConnectionPool();
    // Establezca la factoría de conexión como
    // factoría para este agrupamiento
    pool.setConnectionFactory(factory);

}

public Connection getConnection() {

    Connection con = null;

    try {
        con = pool.getConnection();
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return con;
}

public static DataModule1 getDataModule() {
    if (myDM == null) {
        myDM = new DataModule1();
    }
    return myDM;
}

```

```
        public static JdbcConnectionPool getPool() {  
            return pool;  
        }  
    }  
}
```

Es probable que el código para la lógica de la aplicación se escriba en un archivo de código fuente aparte. El siguiente fragmento de código muestra cómo solicitar conexiones del grupo y, posteriormente, cómo asegurarse de que las conexiones vuelven al grupo. Asimismo muestra cómo asegurarse de que se cierra el grupo cuando la aplicación finaliza.

```
public class doSomething {  
  
    static DataModule1 dm = null;  
  
    public doSomething() {  
    }  
  
    public static void main(String args[]) {  
  
        // Algunos de los métodos a los que se llama aquí podrían  
        // producir excepciones; por lo tanto, el manejo de excepciones  
        // es necesario.  
  
        try {  
            // Instancie el módulo de datos  
            dm = new DataModule1();  
            java.sql.Connection con = null;  
  
            // Esta aplicación recibe 100 conexiones  
            // y las devuelve al agrupamiento.  
            for (int i=0; i<100; i++){  
  
                try {  
                    // Obtenga una conexión  
                    con = dm.getPool().getConnection();  
                }  
  
                catch(Exception e) {  
                    e.printStackTrace();  
                }  
  
                finally {  
                    // Devuelva la conexión al agrupamiento  
                    con.close();  
                }  
            }  
        }  
        catch (Exception x) {  
            x.printStackTrace();  
        }  
    }  
}
```

```

        finally {
            try {
                // Cierre el agrupamiento antes de que
                // la aplicación salga.
                dm.getPool().shutdown();
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}

```

Resolución de problemas en conexiones de JDataStore e InterBase

JDBC puede generar mensajes de error al utilizarse para conectar con servidores SQL. Si desea ayuda con la resolución de problemas de conexión de JDataStore en los tutoriales:

- Consulte “Depuración de aplicaciones JDataStore”, en la *Guía del desarrollador de JDataStore*.
- Consulte las FAQ de JDataStore de Borlan en <http://community.borland.com/article/0,1410,19685,00.html>.
- Si desea información sobre los problemas de conexión a InterBase por medio de InterClient, consulte “[Mensajes de error de conexión habituales](#)” en la página 4-23.

Mensajes de error de conexión habituales

A continuación se presenta una serie de errores de conexión habituales y sus soluciones:

- No es posible encontrar el controlador InterClient. No se ha añadido InterClient al proyecto como biblioteca necesaria. Elija Proyecto | Propiedades y añada InterClient como biblioteca necesaria.
- No es posible cargar el controlador. No se ha añadido InterClient a CLASSPATH. Añada el archivo interclient.jar al script CLASSPATH de inicio de JBuilder o al CLASSPATH del entorno antes de ejecutar JBuilder.

Recuperación de datos de una fuente de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Este apartado se centra en el uso de la arquitectura DataExpress de JBuilder para la recuperación de datos de una fuente y el suministro de datos a una aplicación. Los componentes de los paquetes DataExpress encapsulan la conexión entre la aplicación y su origen de datos y el comportamiento necesario para el manejo de datos.

Para crear una aplicación de base de datos se recupera la información almacenada en el origen de datos y se crea una copia que la aplicación pueda gestionar dentro del ordenador en que se ejecuta. Los datos recuperados del proveedor se guardan en la memoria caché, dentro de un conjunto de datos. El sistema lleva el control de todas las modificaciones realizadas en la caché del `DataSet`, lo que permite a las implementaciones de almacenador (resolver) determinar lo que debe ser insertado, actualizado o borrado en la fuente de los datos. En JBuilder, se extrae un subconjunto de datos de la fuente a una subclase `StorageDataSet` de JBuilder. La subclase `StorageDataSet` que se utiliza depende de la forma en que se obtenga la información.

Si se utiliza el sistema de proveedores y almacenadores basta con dos interacciones entre la aplicación de base de datos y el origen de los datos: la conexión inicial, que recupera los datos, y la conexión final, que los almacena en el origen. La conexión entre el componente `DataSet` y el origen de datos se puede desconectar después de la transmisión de datos y sólo es necesario restablecerla mientras dura la transacción de almacenamiento.

Los componentes DataExpress proporcionan también la aceptación de la asociación directa de datos a componentes `dbSwing`. Basta con definir una propiedad del Inspector para asociar datos a componentes visuales.

En algunos ejemplos de este capítulo se utiliza un controlador `JDataStore` para acceder a los datos de un archivo `JDataStore`. En otros se emplea un controlador `JDBC` para acceder a los datos de tablas `InterBase`. Cada una de estas dos opciones tiene sus ventajas. Para elegir una de ellas se deben tener en cuenta las necesidades de la aplicación. Las dos opciones permiten:

- Transmitir directamente componentes visuales.
- Obtener un acceso completo a los datos, que incluye relaciones maestro-detalle, ordenación, filtrado y restricciones.
- Efectuar un seguimiento de la modificación de los datos recuperados para que se puedan almacenar correctamente en la fuente de datos.

Consultas en bases de datos

Un componente `QueryDataSet` es un `DataSet` específico de `JDBC` que gestiona un proveedor de datos de `JDBC`, como se define en la propiedad `query`. Mediante el uso de un componente `QueryDataSet` en `JBuilder` se pueden extraer datos de una fuente de datos a un componente `StorageDataSet`. Esto se denomina “suministrar”. Una vez suministrados los datos, pueden visualizarse y utilizarse localmente en componentes enlazados a datos. Para guardar los datos en la base de datos es preciso “resolver” las modificaciones. La arquitectura de `DataExpress` trata con más detalle en el [Capítulo 2, “Aplicaciones de base de datos JBuilder”](#).

Los componentes `QueryDataSet` permiten utilizar sentencias SQL para acceder o suministrar datos desde la base de datos. Puede añadir un componente `QueryDataSet` directamente a la aplicación o añadirlo a un módulo de datos para centralizar el acceso a los datos y controlar la lógica empresarial.

Para poder consultar una tabla SQL necesita los siguientes componentes, que se pueden suministrar escribiendo código o utilizando herramientas de diseño de `JBuilder`:

- `Database`

El componente `Database` encapsula una conexión de base de datos con el servidor SQL a través de `JDBC` y también proporciona algo de soporte para transacciones.

- `QueryDataSet`

El componente `QueryDataSet` proporciona la funcionalidad para ejecutar una sentencia de consulta (con o sin parámetros) en la tabla de una base de datos SQL y almacena el conjunto resultante de la consulta.

- `QueryDescriptor`

El objeto `QueryDescriptor` almacena las propiedades de la consulta, incluida la base de datos que se consulta, la cadena de consulta que se ejecuta y los parámetros opcionales de ésta.

`QueryDataSet` tiene funciones incorporadas para capturar datos de una fuente de datos JDBC. Sin embargo, las funciones incorporadas (en el formulario del almacenador por defecto) hacen algo más que capturar datos. También generan las consultas SQL INSERT, UPDATE y DELETE necesarias para guardar los cambios en el origen de datos después de la captura.

Las propiedades siguientes de `QueryDescriptor` influyen en la ejecución de la consulta. Estas propiedades pueden definirse de manera visual en el editor de propiedades de consultas. Si desea una explicación del editor de la propiedad `query` y sus herramientas y propiedades, consulte [“Asignación de propiedades en el cuadro de diálogo de consulta” en la página 5-4](#).

Propiedad	Efecto
base de datos	Especifica en qué objeto de conexión Database se ejecuta la consulta.
query	Una sentencia SQL (normalmente SELECT).
parámetros	Un <code>ReadWriteRow</code> opcional a partir del cual se rellenan parámetros, utilizado para las consultas parametrizadas.
executeOnOpen	Hace que <code>QueryDataSet</code> ejecute la consulta cuando se abre por primera vez. Esto resulta útil para mostrar datos reales durante el diseño. Es también posible en tiempo de ejecución.
loadOption	Un valor entero optativo que define cómo se cargan los datos en el conjunto de datos. Las opciones son: <ul style="list-style-type: none"> • Cargar todas las filas: carga todos los datos de una vez. • Cargar filas en paralelo: hace que la captura de filas de <code>DataSet</code> se realice en un hilo distinto. Esto permite acceder a los datos del <code>DataSet</code> y visualizarlos conforme el componente <code>QueryDataSet</code> va recibiendo las filas de la conexión a la base de datos. • Cargar cuando es necesario: carga las filas cuando se precisen más datos. • Cargar una fila cada vez: carga cuando es necesario y sustituye la fila anterior por la actual. Resulta útil en aplicaciones de procesos por lotes para volúmenes altos.

Para obtener datos, se puede utilizar un `QueryDataSet` de tres formas diferentes:

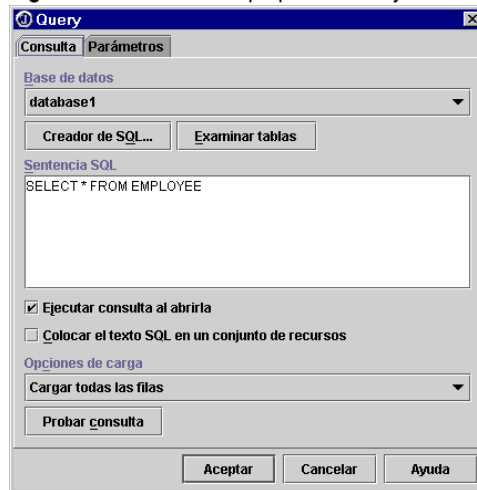
- Consultas no parametrizadas: se ejecuta la consulta y se capturan filas para incluirlas en el `QueryDataSet`.
- Consultas parametrizadas: en una consulta parametrizada, se utilizan las variables en la sentencia y se utilizan los parámetros actuales para establecer los de la consulta actual. Para obtener más información sobre consultas parametrizadas, consulte [“Utilización de consultas parametrizadas para obtener datos” en la página 5-13](#).

- Capturas dinámicas de todos los grupos de detalles. Los registros de conjunto de datos de detalle se obtienen por demanda y se almacenan en este conjunto. Para obtener más información, consulte [“Captura de detalles” en la página 9-8](#).

Asignación de propiedades en el cuadro de diálogo de consulta

El editor de la propiedad de consulta se abre cuando se pulsa el botón de puntos suspensivos del campo de valor de la propiedad `query` de `QueryDataSet`. Puede utilizar el editor para establecer visualmente las propiedades del `QueryDescriptor`, aunque tiene también otros usos. A continuación se muestra el editor de la propiedad `query` y se explican detalladamente las opciones del mismo.

Figura 5.1 Editor de la propiedad Query



Si desea obtener más información, consulte el tema `com.borland.dx.sql.dataset.QueryDescriptor` en la documentación *DataExpress Library Reference*.

Ficha Consulta

En la pestaña Consulta están disponibles las siguientes opciones:

- La lista desplegable **Base de datos** muestra los nombres de todos los objetos `Database` instanciados a los que puede asociarse este `QueryDataSet`. Para poder ejecutar la consulta satisfactoriamente, esta propiedad debe contar con un valor. Si desea ayuda sobre la creación de instancias de `Database`, consulte el [Capítulo 4, “Conexión con bases de datos”](#).

Cuando se selecciona un objeto `Database` se activan el creador de SQL y el botón Examinar tablas.

- Pulse el botón **Creador de SQL** para abrir el Creador de SQL. El Creador de SQL proporciona una representación visual de la base de datos y permite crear una Sentencia SQL mediante la selección de columnas, añadiendo una cláusula `Where`, una cláusula `Order By`, una cláusula `Group By`, así como ver y comprobar la Sentencia SQL generada. Cuando se pulsa Aceptar, la Sentencia SQL creada con el Creador de SQL se sitúa en el campo Sentencia SQL del cuadro de diálogo Consulta.
- Pulse el botón **Examinar tablas** para abrir el cuadro de diálogo Tablas y columnas disponibles. Dicho cuadro de diálogo presenta una lista de tablas de la `Database` especificada, así como las columnas de la tabla seleccionada. Los botones Pegar tabla y Pegar columna permiten crear rápidamente la sentencia de consulta, pegando el nombre de la tabla seleccionada (pulsando el botón Pegar tabla) o de la columna seleccionada (pulsando el botón Pegar columna) en la posición actual del cursor (punto de inserción) en la sentencia de consulta.

Este botón estará atenuado y no disponible mientras se muestre el valor “<ninguno>” en el campo `Database`. Para activar este botón, seleccione un objeto `Database` en el campo `Database`.

- La **Sentencia SQL** es una representación Java String de una sentencia SQL (normalmente una sentencia `SELECT`). Introduzca la sentencia de consulta que desee ejecutar en el objeto `Database` especificado en la lista desplegable `Database`. Pulse el botón Examinar tablas para pegar rápidamente los nombres de tabla y columna seleccionados en la sentencia de consulta. Esta propiedad es obligatoria; debe especificar una sentencia SQL válida. Si la sentencia SQL no devuelve un conjunto de resultados, se genera una excepción.

El ejemplo que contiene una sentencia SQL sencilla utilizado en este texto selecciona tres campos de la tabla `EMPLOYEE`:

```
SELECT emp_no, last_name, salary FROM employee
```

La siguiente sentencia SQL selecciona todos los campos de la misma tabla.

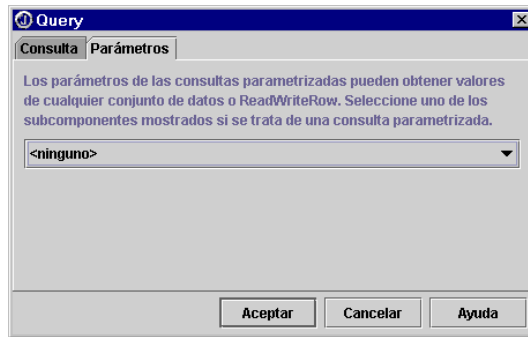
```
SELECT * FROM employee
```

- La opción **Ejecutar consulta** determina si la consulta se ejecutará automáticamente en cuanto se abra el `QueryDataSet`. Esta opción está activada por defecto, lo que permite mostrar los datos activos en el diseñador de interfaces de usuario cuando `QueryDataSet` está asociado a un componente enlazado a datos.
- Las **opciones de carga** son valores enteros opcionales que definen el método de carga de los datos en el conjunto de datos. Las opciones son:

- a Cargar todas las filas: carga todos los datos de una vez.
 - b Cargar filas en paralelo: hace que la captura de filas de `DataSet` se realice en un hilo distinto. Esto permite acceder a los datos del `DataSet` y visualizarlos conforme el componente `QueryDataSet` va recibiendo las filas de la conexión a la base de datos.
 - c Cargar cuando es necesario: carga las filas cuando se precisen más datos.
 - d Cargar una fila cada vez: carga cuando es necesario y sustituye la fila anterior por la actual. Resulta útil en aplicaciones de procesos por lotes para volúmenes altos.
- Si está activada la casilla **Colocar el texto SQL en un conjunto de recursos**, al abandonar el editor de la propiedad `query` aparecerá el cuadro de diálogo Crear archivo de recursos extraídos. Seleccione un tipo de archivo de recursos extraídos. Cuando pulse el botón Aceptar, el texto SQL se escribirá en un archivo de recursos, para permitirle continuar utilizando un código fuente que incluya sentencias SQL persistentes en algunas aplicaciones. Consulte [“Cómo colocar el texto SQL en un conjunto de recursos” en la página 5-7](#) si desea una descripción más detallada de esta función.
- Si está desactivada, la cadena SQL se escribe en el `QueryDescriptor` como una cadena incrustada en el código fuente.
- Haga clic en **Probar consulta** para comprobar la sentencia SQL y otras propiedades de este cuadro de diálogo con la `Database` especificada. Se mostrará el resultado (“Correcto” o “Fallo”) en el área gris que está justo debajo del botón Probar consulta. Si en este área se indica Correcto, se ejecutará correctamente la consulta. Si se indica Fallo, revise la información especificada en la `consulta`, buscando errores ortográficos y omisiones.

La ficha Parámetros

En la pestaña Parámetros, puede seleccionar un `ReadWriteRow` o `DataSet` opcional a partir del cual desee rellenar los parámetros en las consultas parametrizadas. Los valores se especifican a través de un `ReadWriteRow` instanciando Seleccione el objeto `ReadWriteRow` (o la subclase `ReadWriteRow`) que contenga los valores de los parámetros de consulta de la lista desplegable.

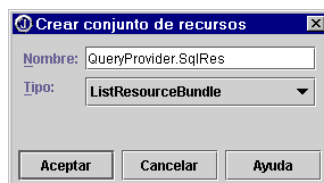
Figura 5.2 Ficha de Parámetros

Puede utilizar como parámetros del procedimiento o la consulta cualquier `ReadWriteRow`, como `ParameterRow`, `DataSet`, o `DataRow`. En una fila de parámetros (`ParameterRow`), las columnas pueden definirse fácilmente con los métodos `addColumnns` y `setColumns`. `DataSet` y `DataRow` sólo deben utilizarse si ya contienen las columnas en las que se encuentran los datos deseados. En [“Utilización de consultas parametrizadas para obtener datos” en la página 5-13](#) puede encontrar un ejemplo sobre esto.

Cómo colocar el texto SQL en un conjunto de recursos

Un `java.util.ResourceBundle` contiene objetos específicos de idioma. Si un programa necesita un recurso específico de idioma, puede cargarlo desde el archivo de recursos extraídos que sea apropiado para el idioma del usuario actual. De esta forma, puede escribir código para un programa que sea fundamentalmente independiente del idioma del usuario, lo que aísla la mayoría, e incluso toda, la información específica de idioma en los conjuntos de recursos.

El cuadro de diálogo Crear archivo de recursos extraídos se abre cuando se cierra el editor de consulta, si se ha definido una sentencia SQL en dicho editor de consulta y se ha marcado la opción Colocar el texto SQL en un conjunto de recursos. El cuadro de diálogo de recursos extraídos tiene el siguiente aspecto:

Figura 5.3 Cuadro de diálogo de recursos extraídos

Para utilizar un archivo de recursos extraídos en una aplicación:

- 1 Seleccione un tipo de archivo de recursos extraídos.

Para simplificar las cosas, JDK ofrece dos útiles subclases de `ResourceBundle`: `ListResourceBundle` y `PropertyResourceBundle`. La clase `ResourceBundle` es una clase abstracta. Para crear un archivo concreto de recursos extraídos, necesita derivarlo de `ResourceBundle` y proporcionar realizaciones concretas de algunas funciones que se recuperan del almacén en el que sitúe sus recursos, por ejemplo, una cadena. Puede almacenar recursos en este archivo haciendo clic con el botón derecho en una propiedad y especificando la clave. JBuilder escribe la cadenas en el archivo de recursos, con el formato correcto, que depende del tipo.

- Si selecciona `ListResourceBundle`, se genera un archivo Java y se añade al proyecto. Con `ListResourceBundle`, los mensajes (u otros recursos) se almacenan en una matriz 2-D de un archivo de recursos Java. `ListResourceBundle` también es una clase abstracta. Para crear un archivo de recursos extraídos real que pueda cargarse, dérivelo de `ListResourceBundle` e implemente `getContents()`, que muy probablemente señalará a una matriz bidimensional de pares de objetos clave. Para el ejemplo anterior, crearía una clase:

```
package myPackage;
public class myResource extends ListResourceBundle {
    Object[][] contents = {
        {"Hello_Message", "Howdy mate"}
    }
    public Object[][] getContents() {
        return contents;
    }
}
```

- Si seleccionara `PropertyResourceBundle`, se crearía un archivo de propiedades. `PropertyResourceBundle` es una clase concreta, es decir, no es necesario crear otra clase para poder utilizarla. Para los archivos de recursos extraídos de propiedades, el almacenamiento de recursos se lleva a cabo en archivos con la extensión `.properties`. Cuando se realiza un archivo de recursos extraídos de esta forma, sencillamente se suministra un archivo de propiedades con el nombre correcto y se almacena en el mismo lugar que los archivos de clase del paquete. En el ejemplo anterior, crearía un archivo `myResource.properties` y lo situaría en VÍA DE ACCESO DE CLASE o en el archivo `zip/jar`, junto con otras clases del paquete `myPackage`. Esta forma de archivo de recursos extraídos sólo puede contener cadenas, y se carga con mucha más lentitud que las implementaciones basadas en clase, como `ListResourceBundle`. No obstante, su uso está muy extendido, porque no implica trabajar con código fuente y no requiere volver a compilar. El contenido del archivo de propiedades es parecido a éste:

```
# comments
Hello_message=Howdy mate
```

2 Pulse Aceptar o Cancelar:

Cuando se hace clic en el botón Cancelar (o se anula la selección de Llevar texto SQL a archivo de recursos extraídos, en el cuadro de diálogo de la consulta), se escribe un `QueryDescriptor` como el siguiente en el archivo `Marco`. El texto SQL se escribe como una cadena incrustada en el código fuente.

```
queryDataSet1.setQuery(new
com.borland.dx.sql.dataset.QueryDescriptor(database1,
    "select * from employee", null, true, LOAD.ALL));
```

Cuando se pulsa el botón Aceptar, se crea un `queryDescriptor` como el siguiente:

```
queryDataSet1.setQuery(new
com.borland.dx.sql.dataset.QueryDescriptor(database1,
    sqlRes.getString("employee"), null, true, LOAD.ALL));
```

Siempre que guarde el texto SQL en el cuadro de diálogo `QueryDescriptor`, `JBuilder` creará automáticamente un archivo llamado `SqlRes.java`, que sitúa el texto de la cadena SQL dentro de `SqlRes.java` y crea una única etiqueta de cadena que inserta en el texto. Por ejemplo, en la sentencia de selección `SELECT * FROM employee`, introducida anteriormente, en el momento en que se pulse el botón Aceptar, se crea el archivo `SqlRes.java`, cuyo aspecto es similar a este:

```
public class SqlRes extends java.util.ListResourceBundle {
    static final Object[][] contents = {
        { "employee", "select * from employee" } };
    static final java.util.ResourceBundle res = getBundle("untitled3.SqlRes");
    public static final String getStringResource(String key) {
        return res.getString(key);
    }
    public Object[][] getContents() {
        return contents;
    }
}
```

Si la sentencia SQL se modifica, los cambios se guardan en `SqlRes.java`. No será necesario ningún cambio en el código dentro de `jbInit()`, ya que la cadena “etiqueta” no se puede modificar.

Para obtener más información acerca de archivos de recursos extraídos, consulte el `JavaDoc` de `java.util.ResourceBundle`, en la Ayuda de `JBuilder`, seleccionando `Ayuda | Referencia Java`. Después seleccione el paquete `java.util` y la clase `ResourceBundle`.

Consultas en bases de datos: Sugerencias

Este conjunto de temas de ayuda incluye sugerencias para:

- Mejorar el rendimiento de los datos.
- Abrir y cerrar conjuntos de datos con más eficacia.

- Comprobar que una consulta es actualizable.

Cómo mejorar el rendimiento del DataSet

En este apartado se ofrecen algunas recomendaciones para optimizar el rendimiento de los `QueryDataSets` y `QueryProviders`. Para conseguir mayor rapidez en las operaciones de extracción de datos, elimine el análisis de la consulta que el `QueryProvider` realiza por defecto cada vez que se ejecuta por primera vez una determinada consulta. En [“Metadatos persistentes de consulta” en la página 5-11](#) se explica la forma de hacerlo.

- Asigne a la propiedad `loadOption` de los componentes `Query/ProcedureDataSet` el valor `Load.ASYNCHRONOUS` (carga asíncrona) o `Load.AS_NEEDED` (cargar cuando se necesite). También puede asignar a esta propiedad el valor `Load.UNCACHED` (cargar sin guardar en caché) si va a guardar los datos una sola vez y por orden consecutivo.
- Si necesita manejar conjuntos de resultados de mayor tamaño, puede utilizar un `JDataStore` para mejorar el rendimiento. Con esta opción, los datos se guardan en disco en lugar de en memoria.
- Sentencias SQL en caché. Por defecto, si la función `java.sql.Connection.getMetaData().getMaxStatements()` devuelve un valor mayor que 10, `DataExpress` almacenará en memoria caché las sentencias preparadas tanto para las consultas como para los procedimientos almacenados. Llamando a la función `Database.setUseStatementCaching(true)` puede imponerse el almacenamiento de sentencias en memoria caché en `JBuilder`.

Las sentencias preparadas que se encuentran almacenadas en caché no se cierran hasta que no ocurre alguna de las siguientes circunstancias.

- Un cambio en alguna propiedad relacionada con el proveedor, por ejemplo la propiedad `query`.
- La aplicación del mecanismo de liberación de memoria (garbage collection) no utilizada sobre un componente `DataSet` (sentencia cerrada en un método `finalize()`).
- Una llamada a `QueryDataSet.closeStatement()`, `ProcedureDataSet.closeStatement()`, `QueryProvider.closeStatement()`, o `ProcedureProvider.closeStatement()`.

Para mejorar el rendimiento en las operaciones de inserción, borrado o actualización de datos:

- En las actualizaciones y borrados:
 - a Asigne a la propiedad `Resolver` el valor `QueryResolver`.
 - b Asigne a la propiedad `UpdateMode` (modo de actualización) de este `QueryResolver` el valor `UpdateMode.KEY_COLUMNS` (actualizar columnas clave).

Con ello se relaja el criterio optimista de concurrencia que se utiliza, pero se reduce el número de parámetros que intervienen en la operación de actualización o borrado.

- Asigne a la propiedad `useTransactions` de la base de datos el valor `false`. Esta propiedad tiene el valor `true` por defecto, si la base de datos acepta transacciones. Si el valor es `true`, cada sentencia de inserción, borrado o actualización se gestiona como una transacción independiente de envío automático. Cuando se asigna a `useTransactions` el valor `false`, todas las sentencias se procesan en una transacción.

Nota En este caso se debe llamar al método `commit()` del componente `Database` o `Connection` para efectuar la transacción (o llamar a `rollback()` para descartar todos los cambios).

- Puede obtener una mejora de velocidad adicional desactivando el indicador `resetPendingStatus` del método `Database.saveChanges()`. Con este indicador desactivado, el `DataExpress` no borrará el estado `RowStatus` de todas las filas insertadas, borradas o actualizadas. Sólo conviene hacerlo si no se va a llamar a `saveChanges()` con nuevas modificaciones en el `DataSet` sin antes llamar a `refresh`.

Metadatos persistentes de consulta

Por defecto, la primera vez que se ejecuta una consulta, se analiza para determinar si es actualizable. Este proceso consiste en analizar la cadena que define la consulta y llamar a varios métodos del controlador JDBC. Este análisis puede resultar bastante costoso. Sin embargo, es posible evitar que, durante la ejecución, se produzca el tiempo de procesamiento adicional, y realizar el análisis durante el diseño de un formulario o un modelo de datos.

Para ello:

- 1 Resalte el `QueryDataSet` en el diseñador, haga clic con el botón derecho sobre él y seleccione **Activar diseñador**.
- 2 En el diseñador de columnas, pulse el botón **Conservar todos los metadatos**.



Con ello se analizará la consulta, y se agregará al código un conjunto de valores de propiedades. Si desea más información sobre el botón **Mantener todos los metadatos**, consulte [“Utilización del diseñador de columnas para convertir metadatos en persistentes” en la página 7-4](#). Para definir las propiedades sin utilizar el diseñador:

- 1 Asigne a la propiedad `metaUpdate` del `StorageDataSet` el valor `NONE`.
- 2 Asigne a la propiedad `tableName` de `StorageDataSet` el nombre de tabla para consultas de tabla.

- 3 Defina la propiedad `rowID` de `Column` de forma que las columnas identifiquen una única fila.
- 4 Modifique la cadena de consulta para incluir columnas adecuadas para identificar a una fila (ver punto anterior), si es que no están ya incluidas. Estas columnas de deben haber ocultado con la propiedad `visible` o `hidden` de `Column`.
- 5 Asigne a las propiedades `precision`, `scale` y `searchable` de la columna los valores adecuados. Estas propiedades no son necesarios si la propiedad `metaDataUpdate` tiene un valor distinto de `NONE`.
- 6 Asigne valores a la propiedad `tableName` de `Column` para las consultas de varias tablas.
- 7 La propiedad `serverColumnName` de `Column` debe tener como valor el nombre de la columna de la tabla correspondiente si se utiliza un alias para una columna de la consulta.

Apertura y cierre de conjuntos de datos

`Database` y `DataSet` se abren implícitamente cuando se abren los componentes asociados a ellos. Si no se utiliza un componente visual, se debe abrir de forma explícita un `DataSet`. “Abrir” se propaga hacia arriba y “cerrar”, hacia abajo. Por ello, al abrir un `DataSet` se abre de forma implícita un `Database`. Un `Database` nunca está cerrado implícitamente.

Verificación de que una consulta es actualizable

Cuando `JBuilder` ejecuta una consulta, intenta verificar que ésta es actualizable y que puede resolverse en la base de datos. Si `JBuilder` determina que no es actualizable, intentará modificar la consulta para que lo sea, normalmente añadiendo columnas a la sentencia `SELECT`.

Si una consulta no es actualizable, y `JBuilder` no logra modificarla de manera que lo sea, los datos resultantes serán de sólo lectura.

Para hacer que un conjunto de datos sea actualizable, asigne a la propiedad `updateMetaData` el valor **NONE** y especifique el nombre de tabla y las columnas identificadoras de fila única del conjunto de datos. Estas últimas son un conjunto de columnas que puede identificar, sin duplicación, a una fila, como lo son las columnas de un índice primario o único. Si desea información sobre la forma de hacerlo, consulte [“Metadatos persistentes de consulta” en la página 5-11](#).

Se puede consultar una vista SQL, pero `JBuilder` no indica que los datos se han recibido de ella, como cuando proceden de una tabla SQL, por lo que es arriesgado que los conjuntos de datos no sean actualizables. Puede resolver este problema si escribe un almacenador personalizado.

Utilización de consultas parametrizadas para obtener datos

Una sentencia SQL parametrizada contiene variables, conocidas también como parámetros, cuyos valores pueden variar en tiempo de ejecución. Una consulta parametrizada utiliza estas variables para sustituir los valores de datos literales que aparecen en las sentencias SQL, como los que se utilizan en las cláusulas WHERE para comparaciones. Estas variables se denominan *parámetros*. Normalmente, los parámetros sustituyen a los valores transmitidos a la sentencia. Los valores de los parámetros se suministran antes de ejecutar la consulta. Al proporcionar diferentes conjuntos de valores y ejecutar la consulta para cada uno, puede hacer que una misma consulta devuelva conjuntos de datos diferentes.

La comprensión de la forma en que se suministran los datos a un `DataSet` es esencial para comprender en profundidad las consultas parametrizadas, por lo que es conveniente leer los temas [Capítulo 2, “Aplicaciones de base de datos JBuilder”](#) y [“Consultas en bases de datos” en la página 5-2](#) si todavía no lo ha hecho. Este tema se limita a las consultas parametrizadas.

Además de [“Parametrizar una consulta,”](#), se tratan los siguientes temas sobre consultas parametrizadas:

- [“Utilización de parámetros” en la página 5-19](#)
- [“Ejecución de la consulta parametrizada con nuevos parámetros” en la página 5-21](#)
- [“Consultas parametrizadas en relaciones maestro-detalle” en la página 5-22](#)

Parametrizar una consulta

El siguiente ejemplo muestra cómo suministrar datos a una aplicación empleando un componente `QueryDataSet`. Este ejemplo añade un `ParameterRow` con valores máximo y mínimo que pueden cambiarse en tiempo de ejecución. Cuando se cambian los valores de `ParameterRow`, la tabla actualiza automáticamente su presentación para que sólo se reflejen los registros que cumplan los criterios especificados con los parámetros.

Nota Antes de continuar con los siguientes pasos, se recomienda familiarizarse con las herramientas de diseño visual mediante el tutorial del [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#).

Puede encontrar una versión finalizada de la aplicación que se crea en los próximos pasos en el proyecto de ejemplo `ParameterizedQuery.jpx` situado en el directorio `/samples/DataExpress/ParameterizedQuery` de la instalación de JBuilder.

Creación de la aplicación

Para crear esta aplicación:

- 1 Seleccione Archivo | Cerrar todo.
- 2 Seleccione Archivo | Nuevo y haga doble clic en el icono Aplicación.
- 3 Acepte todos los valores por defecto para crear una nueva aplicación.
- 4 Active el diseñador de interfaces de usuario, seleccionando la pestaña Diseño.



- 5 Haga clic en el componente `Database` de la pestaña `DataExpress` de la paleta de componentes y, a continuación, haga clic en el panel de diseño para añadir el componente a la aplicación.

Abra el editor de la propiedad `Connection` del componente `Database`, pulsando el botón de puntos suspensivos (...) que se encuentra en el valor de la propiedad `connection` del Inspector.

- 6 Asigne valores a las propiedades de conexión para la tabla de ejemplo `EMPLOYEE` de `JDataStore`, de la forma siguiente:

Nombre de la propiedad	Valor
Controlador	<code>com.borland.datastore.jdbc.DataStoreDriver</code>
URL	Desplácese al archivo <code><jbuilder>/samples/JDataStore/datasources/employee.jds</code> , en el campo URL local.
Nombre de usuario	Introduzca su nombre
Contraseña	No es obligatoria

El cuadro de diálogo `Connection` contiene un botón `Probar conexión`. Púlselo para comprobar que las propiedades de conexión tienen los valores correctos. El resultado del intento de conexión se muestra junto al botón. Cuando la conexión sea satisfactoria, pulse `Aceptar`.

Si desea ver el código generado, haga clic en la pestaña `Fuente` y busque el código `ConnectionDescriptor`. Haga clic en la pestaña `Diseño` para continuar.

Para obtener más información sobre la forma de conectarse con una base de datos, consulte el [Capítulo 4, “Conexión con bases de datos”](#).

Cómo añadir filas de parámetros

A continuación se añade una `ParameterRow` con dos columnas: `low_no` y `high_no`. Después de asociar `ParameterRow` a `QueryDataSet`, se pueden utilizar componentes `JdbTextField` para cambiar el valor de `ParameterRow` de forma que sea posible actualizar la consulta con estos nuevos valores.

- 1 Añada un componente `ParameterRow` a la aplicación, desde la pestaña `DataExpress`.

- 2 En el árbol de componentes, pulse el icono de ampliación que se encuentra a la izquierda de `parameterRow1` para mostrar las columnas de `ParameterRow`.
- 3 Seleccione <nueva columna> y asigne los valores siguientes a sus propiedades, en el Inspector:

Nombre de la propiedad	Valor
<code>columnName</code>	<code>low_no</code>
<code>dataType</code>	<code>INT</code>
por defecto	15

El código que genera el diseñador para este paso se puede ver en el método `jbInit()`, en la pestaña Fuente. Haga clic en la pestaña Diseño para continuar.

- 4 Vuelva a seleccionar <nueva columna> para añadir la segunda columna a `ParameterRow` y defina las siguientes propiedades:

Nombre de la propiedad	Valor
<code>columnName</code>	<code>high_no</code>
<code>dataType</code>	<code>INT</code>
por defecto	50

Cómo añadir objetos `QueryDataSet`

- 1 Añada un componente `QueryDataSet` de la pestaña `DataExpress` a la aplicación.
- 2 Haga clic en el botón de puntos suspensivos (...) de la propiedad `query` para abrir el editor de la propiedad Query.
- 3 Asigne a la propiedad `query` de `queryDataSet1` los siguientes valores:

Nombre de la propiedad	Valor
<code>Database</code>	<code>database1</code>
<code>Sentencia SQL</code>	<code>select emp_no, first_name, last_name from employee where emp_no >= :low_no and emp_no <= :high_no</code>

- 4 Abra la pestaña Parámetros del editor de la propiedad Query.
- 5 Seleccione `parameterRow1` en el cuadro de lista desplegable para asociar el conjunto de datos a `ParameterRow`.
- 6 Abra la pestaña Consulta y pulse el botón Probar consulta para comprobar si la consulta se puede ejecutar. Cuando el espacio debajo

del botón indique **Correcto**, pulse **Aceptar** para cerrar el cuadro de diálogo.

Se añade el siguiente código para `queryDescriptor` al método `jbInit()`:

```
queryDataSet1.setQuery(new com.borland.dx.sql.dataset.  
    QueryDescriptor(database1,  
        "select emp_no, first_name, last_name from employee where emp_no  
        <= :low_no and emp_no >= :high_no",  
        parameterRow1, true, Load.ALL));
```

- 7 Añada el componente `DBDisposeMonitor` de la pestaña **Más dbSwing**. El componente `DBDisposeMonitor` cierra el `JDataStore` cuando se cierre la ventana.
- 8 Asigne a la propiedad `dataAwareComponentContainer` de `DBDisposeMonitor` el valor `this`.

Cómo añadir los componentes de la interfaz

En las siguientes instrucciones se presupone que el programador ha seguido el tutorial de bases de datos para principiantes y está familiarizado con la adición de componentes de la interfaz al diseñador.

Para añadir los componentes que permiten ver y manejar los datos de la aplicación:

- 1 Pulse sobre el componente `TableScrollPane` de la pestaña **dbSwing** de la paleta de componentes y colóquelo en el centro del panel del diseñador de interfaces de usuario.

Asegúrese de que la propiedad `constraints` tiene el valor `CENTER`.

- 2 Coloque un componente `JdbTable` de la pestaña **dbSwing** en el centro del componente `tableScrollPane` y asigne a su propiedad `dataSet` el valor `queryDataSet1`.

Puede observar que en la tabla del diseñador se muestran datos dinámicos.

- 3 Seleccione **Ejecutar** | **Ejecutar proyecto** para ejecutar la aplicación y examinar el conjunto de datos.
- 4 Cierre la aplicación que se está ejecutando.

Para añadir los componentes y crear la variable de consulta parametrizada en tiempo de ejecución:

- 1 Seleccione el componente `JPanel` en la pestaña **Contenedores Swing** y colóquelo en el árbol de componentes, directamente sobre el icono situado a la izquierda de `contentPane(BorderLayout)`.

De esta forma se garantiza que `JPanel (jPanel1)` se añade a la interfaz principal y no a `tableScrollPane1`, que ocupa actualmente todo el panel.

- 2 Asegúrese de que la propiedad `constraints` tiene el valor `NORTH`.
Si el tamaño de `tableScrollPane` se reduce de repente, mire si su propiedad `constraints` tiene aún el valor `CENTER`.
- 3 Seleccione `jPanel1` y establezca el valor de su propiedad `preferredSize` en `200,100`.
De este modo, dispondrá del tamaño suficiente como para contener el resto de los componentes de la interfaz de usuario.
- 4 Coloque un componente `JdbTextField` de la pestaña `dbSwing` a `jPanel1`.
Este componente contiene el valor mínimo.
- 5 Observe que `jdbTextField1` se coloca en el centro del panel, en la parte superior.
Esto se debe a que el diseño por defecto de los componentes `JPanel` es `FlowLayout`. Si prueba a arrastrar el componente a otro lugar, no se quedará en él, sino que volverá a su posición original.
Para controlar la colocación de los componentes de la interfaz en este panel, cambie la propiedad `layout` de `jPanel1` a `'null'` y arrastre `jdbTextField1` a la izquierda del panel.
- 6 Asigne a la propiedad `columns` de `jdbTextField1` el valor `10` para asignarle una anchura fija. Asigne a su propiedad `text` el valor `10` para que coincida con el parámetro mínimo por defecto introducido antes.
- 7 Añada un componente `JButton` de la pestaña `Swing` a `jPanel1`. Esta etiqueta identifica el campo `jdbTextField1` como mínimo.
- 8 Haga clic en `jLabel1`, en el diseñador de interfaces, y arrástrelo por encima de `jdbTextField1`.
- 9 Asigne a la propiedad `text` de `jLabel1` el valor `Minimum`. Arrastre el tirador de redimensionamiento central del borde derecho de la etiqueta y amplíela para que quepa en ella todo el texto.
- 10 Añada otro componente `JdbTextField` y otro `JLabel` a `jPanel1`, para el valor máximo. Arrastre estos dos componentes a la parte derecha del panel.
- 11 Asigne a la propiedad `columns` de `jdbTextField2` el valor `10`, y a su propiedad `text`, el valor `50`.
- 12 Defina la propiedad `text` de `jLabel2` en el valor `Maximum` y amplíe su anchura para que quepa todo el texto.
- 13 Alinee los cuatro componentes.
Mantenga pulsada la tecla `Ctrl` mientras pulsa `jLabel1` y `jdbTextField1`. Haga clic con el botón derecho del ratón y elija `Alinear a la izquierda` para que sus bordes izquierdos queden alineados. (Cuando se utiliza el

diseño null para el prototipo de una interfaz, el menú contextual cuenta con opciones de alineación.)

Alinee a la izquierda jLabel12 y jdbcTextField2. Alinee en la parte superior los dos campos de texto y las dos etiquetas.

- 14 Añada un componente JButton de la pestaña Swing a jPanel1. Coloque este botón en el centro, entre los dos campos de texto. Asigne a su propiedad text el valor Update.

Cuando se pulsa este botón se actualiza el resultado de la consulta parametrizada con los valores introducidos en los campos Minimum y Maximum.

- 15 Abra en el Inspector la pestaña Sucesos, seleccione el campo actionPerformed y haga doble clic en el campo de valor para crear un suceso actionPerformed() en el código fuente. Se abre el panel de código fuente, con el cursor situado entre las llaves de apertura y cierre, en el lugar adecuado para el nuevo suceso actionPerformed().

Añada el siguiente código para que el suceso tenga este aspecto:

```
void jButton1_actionPerformed(ActionEvent e) {
    try {
        // cambiar los valores de la fila de parámetros
        // y actualizar la presentación
        parameterRow1.setInt("low_no",
            Integer.parseInt(jdbcTextField1.getText()));
        parameterRow1.setInt("high_no",
            Integer.parseInt(jdbcTextField2.getText()));
        queryDataSet1.refresh();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

- 16 Guarde su trabajo y ejecute la aplicación. Debe tener un aspecto parecido al siguiente:

	EMP_NO	FIRST_NAME	LAST_NAME
1	11	K. J.	Weston
2	12	Terri	Lee
3	14	Stewart	Hall
4	15	Katherine	Young
5	20	Chris	Papadopoulos
6	24	Pete	Fisher
7	28	Ann	Bennet
8	29	Roger	De Souza
9	34	Janet	Baldwin

Para comprobar el ejemplo, introduzca un valor en el campo Valor mínimo y pulse Actualizar. La tabla sólo muestra los valores superiores al nuevo mínimo. Introduzca un valor en el campo Valor máximo y pulse el botón Actualizar. La tabla sólo muestra los valores inferiores al nuevo máximo.

Para guardar los cambios en la fuente de datos es necesario añadir un `QueryResolver`. Consulte [“Almacenamiento de cambios desde un QueryDataSet” en la página 8-3](#), si desea información sobre la forma de añadir botones con código de almacenamiento, o añada un componente `JdbNavToolBar` al panel de contenido y pulse su botón Guardar cambios como almacenador de consultas por defecto.

Consultas parametrizadas: Sugerencias

Este conjunto de temas de ayuda incluye sugerencias para:

- Averiguar la forma de utilizar parámetros con nombre y marcadores de parámetro.
- Ejecutar la consulta con nuevos parámetros.
- Utilizar una consulta parametrizada en una relación maestro-detalle.

Utilización de parámetros

Para asignar valores a los parámetros en una consulta parametrizada, primero debe crear un `ParameterRow` y añadirle columnas que cuenten con un nombre que serán los sustitutos de los valores que se pasen a la consulta.

Puede utilizar como parámetros del procedimiento o la consulta cualquier `ReadWriteRow`, como `ParameterRow`, `DataSet`, o `DataRow`. En una fila de parámetros (`ParameterRow`), las columnas pueden definirse fácilmente con los métodos `addColumnns` y `setColumns`. `DataSet` y `DataRow` sólo deben utilizarse si ya contienen las columnas en las que se encuentran los datos deseados.

Las clases `Row` se utilizan con mucha frecuencia en las API `DataExpress`. `ReadRow` y `ReadWriteRow` se utilizan de forma muy parecida a interfaces que indican la intención de su uso. Si utiliza una jerarquía de clases, la implementación se comparte y hay pocas ventajas de rendimiento sobre el uso de interfaces.

El siguiente texto ilustra la jerarquía de clases asociada a los métodos `DataSet`:

```
java.lang.Object
+----com.borland.dx.dataset.ReadRow
      +----com.borland.dx.dataset.ReadWriteRow
            +----com.borland.dx.dataset.DataSet
                  +----com.borland.dx.dataset.StorageDataSet
                        +----com.borland.dx.sql.dataset.QueryDataSet
```

- Los métodos `StorageDataSet` gestionan la estructura del conjunto de datos.
- Los métodos `DataSet` gestionan el desplazamiento.
- Los métodos `ReadWriteRow` permiten modificar valores de columnas (es decir, campos) en la fila actual.
- Los métodos `ReadWriteRow` permiten acceder en modo lectura a valores de columnas (es decir, campos) en la fila actual.
- `TableDataSet` y `QueryDataSet` heredan todos estos métodos.

Las clases `Row` habilitan el acceso a los valores de columna por ordinal y nombre de columna. Especificar las columnas por nombre es un método más consistente y legible que escribir un código. El acceso a las columnas por nombre no es tan rápido como por ordinal, pero sigue siendo bastante rápido si el `DataSet` contiene menos de veinte columnas, gracias a algunos algoritmos patentados de búsqueda de nombre/ordinal a alta velocidad. También es útil acostumbrarse a usar las mismas cadenas para todos los accesos a una columna, ya que esto ahorra memoria y es más fácil de introducir si hay muchas referencias a dicha columna.

`ParameterRow` se pasa en `QueryDescriptor`. El editor de la propiedad `query` permite seleccionar una fila de parámetros. La edición de `ParameterRow`, por ejemplo, para añadir una columna y cambiar sus propiedades, puede realizarse en el Inspector o mediante código.

Por ejemplo, se crea un objeto `ParameterRow` con dos campos: `low_no` y `high_no`. Es posible hacer referencias a `low_no` y `high_no` en la consulta parametrizada y compararlos con cualquier campo de la tabla. Consulte los siguientes ejemplos para ver varias maneras de utilizar estos valores.

En `JBuilder`, las consultas parametrizadas pueden ejecutarse con parámetros con nombre, con marcadores de parámetros o con una relación maestro-detalle. En los apartados siguientes se ofrece una explicación de cada uno de ellos.

- Con parámetros con nombre:

Si los marcadores de parámetro de la consulta se especifican con una coma, seguidos de un nombre alfanumérico, se buscará un nombre que coincida con el del parámetro. La columna en `ParameterRow` que tenga el mismo nombre que un marcador de parámetro se utilizará para establecer el valor del parámetro. Por ejemplo, en la siguiente sentencia SQL, los valores que se han de seleccionar se transmiten como nombres de parámetros.

```
SELECT * FROM employee where emp_no > :low_no and emp_no < :high_no
```

En esta sentencia SQL, `:low_no` y `:high_no` son marcadores de parámetro que sirven de sustitutos para los valores reales suministrados a la sentencia por la aplicación en tiempo de ejecución. El valor de este

campo puede proceder de un componente visual o estar generado mediante la escritura de código. En la fase de diseño se utiliza el valor por defecto de la columna. Cuando se asigna nombre a los parámetros, se les puede transmitir a la consulta en cualquier orden. JBuilder asocia los parámetros al conjunto de datos en el orden apropiado en tiempo de ejecución.

En [“Parametrizar una consulta” en la página 5-13](#), se añaden dos columnas a `ParameterRow` para los valores mínimo y máximo. El descriptor de consulta especifica que la consulta ha de devolver únicamente valores mayores que el mínimo y menores que el máximo.

- Con marcadores de parámetros JDBC de signo de interrogación (?):

Si se utilizan los marcadores de parámetro JDBC de signo de interrogación, las configuraciones del valor del parámetro se ordenan estrictamente de izquierda a derecha.

Por ejemplo, en la siguiente sentencia SQL, los valores que se han de seleccionar se transmiten como marcadores de parámetros JDBC de signo de interrogación (?):

```
SELECT * FROM employee WHERE emp_no > ?
```

En esta sentencia SQL, el valor “?” es un sustituto de un valor real que la aplicación suministra a la sentencia en tiempo de ejecución. El valor de este campo puede proceder de un componente visual o estar generado mediante la escritura de código. Cuando se utiliza un marcador de parámetro JDBC de signo de interrogación “?”, los valores se transmiten a la consulta estrictamente de izquierda a derecha. JBuilder asocia los parámetros a la fuente de los valores (`ReadWriteRow`) en este orden en tiempo de ejecución. Asociar parámetros significa asignar recursos de las sentencias y de sus parámetros, localmente y en el servidor, para mejorar el rendimiento de la ejecución de la consulta.

- Con una relación maestro-detalle:

Por definición, los conjuntos de datos maestro y detalle tienen, por lo menos, un campo en común. Este campo se utiliza como consulta parametrizada. Si desea más información sobre esta forma de proporcionar parámetros, consulte [“Consultas parametrizadas en relaciones maestro-detalle” en la página 5-22](#).

Ejecución de la consulta parametrizada con nuevos parámetros

Para ejecutar la consulta otra vez con nuevos parámetros, defina los nuevos valores en `ParameterRow` y llame a `QueryDataSet.refresh()` para ejecutar la consulta otra vez con los nuevos valores de los parámetros. Por ejemplo, para utilizar un componente de interfaz de usuario que asigne el valor de un parámetro, se puede utilizar una sentencia SQL como:

```
SELECT * FROM phonelist WHERE lastname LIKE :searchname
```

En este ejemplo, el valor del parámetro `:searchname` se puede suministrar desde un componente de interfaz de usuario. Para ello, el código tendría que:

- 1 Obtener el valor del componente cada vez que cambie.
- 2 Situarlo en el objeto `ParameterRow`
- 3 Suministrar este objeto a `QueryDataSet`
- 4 Llamar a `refresh()` en el `QueryDataSet`

Consulte el [“Parametrizar una consulta” en la página 5-13](#) para ver un ejemplo con archivos de muestra de JBuilder.

Si los valores que se desean asignar al parámetro `query` se encuentran en una columna de un conjunto de datos, se puede utilizar este conjunto de datos como el `ReadWriteRow` en el `QueryDescriptor`, desplazarse por el conjunto de datos y volver a ejecutar la consulta una vez para cada valor.

Consultas parametrizadas en relaciones maestro-detalle

En una relación maestro-detalle en la que `DelayedDetailFetch` tiene asignado el valor `true` (para capturar los detalles cuando se necesiten), puede especificar una sentencia SQL como:

```
SELECT * FROM employee WHERE country = :job_country
```

En este ejemplo, `:job_country` es el campo que utiliza este conjunto de datos detalle para vincularse a un conjunto de datos maestro. Se pueden especificar tantos parámetros y campos de vínculo maestro como sea necesario. En una relación maestro-detalle, el parámetro debe tener siempre asignado un nombre que coincida con el de la columna. Para obtener más información sobre las relaciones maestro-detalle y el parámetro `DelayedDetailFetch`, consulte el [Capítulo 9, “Establecimiento de una relación maestro-detalle”](#).

En un descriptor maestro-detalle, la asociación se realiza de forma implícita. Asociar de forma implícita significa que el programador no suministra realmente los valores de los datos, se recuperan de la fila maestra y se asocian implícitamente cuando se ejecuta la consulta de detalle. Asociar parámetros significa asignar recursos de las sentencias y de sus parámetros, localmente y en el servidor, para mejorar el rendimiento de la ejecución de la consulta.

Si los valores que se desean asignar al parámetro `query` se encuentran en una columna de un conjunto de datos (el conjunto de datos maestro), se puede utilizar este conjunto de datos como el `ReadWriteRow` del `QueryDescriptor`, desplazarse por el conjunto de datos y volver a ejecutar la consulta una vez por valor para que éstos se muestren en el conjunto de datos detalle.

Utilización de procedimientos almacenados

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Con un procedimiento almacenado, se encapsulan una o más sentencias SQL en una sola ubicación del servidor, que pueden ejecutarse como un lote. Los componentes `ProcedureDataSet` permiten acceder a los datos de la base de datos, o suministrarlos, con procedimientos almacenados, llamándolos con secuencias de escape de procedimientos de JDBC o mediante una sintaxis específica de servidor para llamadas a procedimientos. Para ejecutar un procedimiento almacenado con una SQL, cuando la salida es un conjunto de filas, se necesitan los siguientes componentes. Se puede suministrar esta información escribiendo código o mediante las herramientas de diseño de JBuilder.

- El componente `Database` encapsula una conexión de base de datos con el servidor SQL a través de JDBC y también proporciona algo de soporte para transacciones.
- El componente `ProcedureDataSet` proporciona la funcionalidad necesaria para ejecutar un procedimiento almacenado (con o sin parámetros) con una base de datos SQL y almacena el resultado de la ejecución de dicho procedimiento.
- El objeto `ProcedureDescriptor` guarda las propiedades del procedimiento almacenado, incluida la base de datos donde se efectúa la consulta, los procedimientos almacenados, las secuencias de escape o las llamadas a procedimientos que se ejecutan y cualquier parámetro opcional del procedimiento almacenado.

Cuando se suministran datos de fuentes de datos JDBC, `ProcedureDataSet` tiene funciones incorporadas que permiten capturar datos de un procedimiento almacenado que devuelve un cursor a un conjunto de

resultados. Las propiedades siguientes del objeto `ProcedureDescriptor` afectan a la ejecución de los procedimientos almacenados:

Propiedad	Propósito
<code>Database</code>	Especifica en qué objeto de conexión <code>Database</code> se ejecuta la consulta.
<code>Procedure</code>	Una representación Java String de una sentencia SQL o secuencia de escape de procedimiento almacenado que causa la ejecución de un procedimiento almacenado.
<code>Parameters</code>	Un <code>ReadWriteRow</code> opcional a partir del cual se rellenan los parámetros. Estos valores pueden adquirirse de cualquier <code>DataSet</code> o <code>ReadWriteRow</code> .
<code>executeOnOpen</code>	Hace que <code>ProcedureDataSet</code> ejecute el procedimiento cuando se abre por primera vez. Esto resulta útil para mostrar datos reales durante el diseño. Es también posible en tiempo de ejecución. El valor por defecto es <code>true</code> .
<code>loadOption</code>	Un valor entero optativo que define cómo se cargan los datos en el conjunto de datos. Las opciones son: <ol style="list-style-type: none"> 1 Cargar todas las filas: carga todos los datos de una vez. 2 Cargar filas en paralelo: hace que la captura de filas de <code>DataSet</code> se realice en un hilo distinto. Esto permite acceder a los datos del <code>DataSet</code> y visualizarlos conforme el componente <code>QueryDataSet</code> va recibiendo las filas de la conexión a la base de datos. 3 Cargar cuando es necesario: carga las filas cuando se precisen más datos. 4 Cargar una fila cada vez: carga cuando es necesario y sustituye la fila anterior por la actual. Resulta útil en aplicaciones de procesos por lotes para volúmenes altos.

Puede utilizar un `ProcedureDataSet` para ejecutar los procedimientos almacenados con y sin parámetros. Un procedimiento almacenado con parámetros puede obtener los valores de sus parámetros de cualquier `DataSet` o `ParameterRow`. [“Utilización de parámetros con procedimientos almacenados de Oracle PL/SQL”](#) en la página 6-7 proporciona un ejemplo.

Utilice el Explorador de bases de datos para examinar y editar objetos estructurales específicos de servidor de base de datos, incluidos campos, tablas, definiciones de procedimientos almacenados, disparadores e índices. Si desea más información acerca del Explorador de bases de datos, elija Herramientas | Explorador de bases de datos y consulte su ayuda en línea.

En los siguientes temas se tratan los componentes de procedimiento almacenado:

- [“Secuencias de escape, sentencias SQL y llamadas a procedimientos específicos de servidor”](#)
- [“Utilización de procedimientos almacenados de InterBase”](#)

- “Utilización de parámetros con procedimientos almacenados de Oracle PL/SQL”
- “Utilización de procedimientos almacenados de Sybase”

Asimismo, se suministra un tutorial, [Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados”](#), para ayudarle a comprender mejor los conceptos presentados en los siguientes apartados.

Procedimientos almacenados: sugerencias

Esta sección contiene sugerencias para ayudarle a:

- Comprender las opciones para el uso de procedimientos almacenados
- Crear los procedimientos utilizados en el tutorial

Secuencias de escape, sentencias SQL y llamadas a procedimientos específicos de servidor

Al introducir información en un campo de sentencia de escape, de procedimiento almacenado o de sentencia SQL, en el editor de la propiedad `procedure`, o en código, dispone de tres opciones para el tipo de sentencia. Son las siguientes:

- Seleccionar un procedimiento existente.

Para buscar un procedimiento existente en la base de datos, seleccione Examinar procedimientos en el editor de la propiedad `procedure`. Se mostrará una lista de los procedimientos disponibles en la base de datos con la que esté conectado. Si el servidor es InterBase y selecciona un procedimiento que no devuelve datos, recibirá un aviso a tal efecto. Si selecciona un procedimiento que devuelve datos, JBuilder intentará generar la sintaxis de escape correcta para la llamada a este procedimiento. No obstante, puede que resulte necesario modificar la sentencia generada automáticamente para que se siga correctamente la sintaxis de su servidor. Para otras bases de datos sólo se introduce el nombre del procedimiento del cuadro de diálogo Seleccionar procedimiento.

Si el procedimiento espera parámetros, deberán coincidir con los nombres de columna de los parámetros.

- Especifique una secuencia de escape de procedimiento de JDBC.

Para introducir una secuencia de escape de un procedimiento JDBC, utilice el siguiente formato:

```
{call PROCEDURENAME (?, ?, ?, ...)} para procedimientos
```

```
{?= call FUNCTIONNAME (?, ?, ?, ...)} para funciones
```

- Especifique la sintaxis específica del servidor para llamadas a procedimientos.

Si un servidor permite una sintaxis independiente para llamadas a procedimientos, puede especificar dicha sintaxis en lugar de un procedimiento almacenado existente o una secuencia de escape de procedimiento de JDBC. Por ejemplo, la sintaxis específica del servidor podría ser como ésta:

```
execute procedure PROCEDURENAME ?,?,?
```

En los dos últimos ejemplos, los marcadores de parámetro, o de signos de interrogación, pueden reemplazarse con parámetros con nombre de la forma :ParameterName. Si desea un ejemplo del uso de parámetros con nombre, consulte [“Utilización de parámetros con procedimientos almacenados de Oracle PL/SQL” en la página 6-7](#). Si desea un ejemplo del uso de procedimientos almacenados de InterBase, consulte [“Utilización de procedimientos almacenados de InterBase” en la página 6-7](#).

Creación manual de tablas y procedimientos para el tutorial

Los procedimientos almacenados constan de sentencias SQL. Es muy fácil escribir y compilar estas sentencias en JBuilder, creando un archivo Java, introduciendo las sentencias y después compilando el código. Si no tiene acceso al proyecto de ejemplo SimpleStoredProcedure o si desea saber cómo se crea una tabla e insertar, actualizar y eliminar procedimientos desde JBuilder, siga estos pasos:

- 1 En el menú, seleccione Archivo | Cerrar todo.
- 2 Seleccione Archivo | Nuevo proyecto.
Esto abre el Asistente para proyectos.
- 3 En el Asistente para proyectos, cambie el directorio de los archivos y el nombre de proyecto a SimpleStoredProcedure/ProcSetUp/ProcSetUp.jpx.
- 4 Seleccione Archivo | Nuevo y haga doble clic en Clase, en la ficha General de la galería de objetos.
- 5 Cambie el nombre de clase a ProcSetUp en el Asistente para clases y, a continuación, pulse Aceptar para crear el archivo ProcSetUp.java.
- 6 Edite el código en la ventana de código fuente o copie y pegue texto desde la ayuda en línea para que coincida con el texto que sigue:

```
package ProcSetUp;

import com.borland.dx.dataset.*;
import com.borland.dx.sql.dataset.*;
import java.sql.*;

public class CreateProcedures {
```

```

public static void main(String[] args) throws DataSetException {
    Database databasel = new Database();
    databasel.setConnection(new ConnectionDescriptor("jdbc:interbase://<IP
address or localhost>:<path to .gdb file>", "SYSDBA", "masterkey",
false, "interbase.interclient.Driver"));
    try { databasel.executeStatement("DROP PROCEDURE GET_COUNTRIES"); }
    catch (Exception ex) {};
    try { databasel.executeStatement("DROP PROCEDURE UPDATE_COUNTRY"); }
    catch (Exception ex) {};
    try { databasel.executeStatement("DROP PROCEDURE INSERT_COUNTRY"); }
    catch (Exception ex) {};
    try { databasel.executeStatement("DROP PROCEDURE DELETE_COUNTRY"); }
    catch (Exception ex) {};
    databasel.executeStatement(getCountriesProc);
    databasel.executeStatement(updateProc);
    databasel.executeStatement(deleteProc);
    databasel.executeStatement(insertProc);
    databasel.closeConnection();
}

static final String getCountriesProc =

"CREATE PROCEDURE GET_COUNTRIES RETURNS (           /r/n"+
"  COUNTRY VARCHAR(15),                           /r/n"+
"  CURRENCY VARCHAR(10) ) AS                       /r/n"+
"BEGIN                                             /r/n"+
"  FOR SELECT c.country, c.currency               /r/n"+
"  FROM country c                                /r/n"+
"  INTO :COUNTRY,:CURRENCY                       /r/n"+
"  DO                                             /r/n"+
"BEGIN                                           /r/n"+
"  SUSPEND;                                       /r/n"+
"  END                                           /r/n"+
"END;";

static final String updateProc =

"CREATE PROCEDURE UPDATE_COUNTRY(                 /r/n"+
"  OLD_COUNTRY VARCHAR(15),                       /r/n"+
"  NEW_COUNTRY VARCHAR(15),                       /r/n"+
"  NEW_CURRENCY VARCHAR(20) ) AS                  /r/n"+
"BEGIN                                           /r/n"+
"  UPDATE country                                /r/n"+
"  SET country = :NEW_COUNTRY                    /r/n"+
"  WHERE country = :OLD_COUNTRY;                 /r/n"+
"END;";

static final String insertProc =

```

```
"CREATE PROCEDURE INSERT_COUNTRY(           /r/n"+
"  NEW_COUNTRY VARCHAR(15),                /r/n"+
"  NEW_CURRENCY VARCHAR(20) ) AS           /r/n"+
"BEGIN                                     /r/n"+
"  INSERT INTO country(country,currency)   /r/n"+
"    VALUES (:NEW_COUNTRY,:NEW_CURRENCY); /r/n"+
"END;";
```

```
static final String deleteProc =
```

```
"CREATE PROCEDURE DELETE_COUNTRY(         /r/n"+
"  OLD_COUNTRY VARCHAR(15) ) AS           /r/n"+
"BEGIN                                     /r/n"+
"  DELETE FROM country                   /r/n"+
"    WHERE country = :OLD_COUNTRY;        /r/n"+
"END;";
}
```

- 7 Haga clic con el botón derecho del ratón en `ProcSetUp.java` en el panel de proyectos y pulse Ejecutar.

Este paso crea las tablas y procedimientos en el servidor.

- 8 En el menú, seleccione Archivo | Cerrar.

Este procedimiento es muy sencillo. En la documentación de la base de datos puede encontrar sugerencias para la creación de procedimientos almacenados más complejos.

Utilización de procedimientos específicos del fabricante

Esta sección contiene información para ayudarle a utilizar procedimientos almacenados con fabricantes de base de datos específicos. Se proporciona información para ayudarle a utilizar los siguientes tipos de procedimientos almacenados:

- Procedimientos almacenados de JDataStore y funciones definidas por el usuario.
- Procedimientos almacenados de InterBase.
- Procedimientos almacenados Oracle PL/SQL.
- Procedimientos almacenados de Sybase.

Utilización de los procedimientos almacenados de JdataStore y de las funciones definidas por el usuario

JDataStore 6 admite la utilización de procedimientos almacenados basados en Java y de funciones definidas por el usuario (UDF). Los

procedimientos almacenados y las UDF se deben añadir a la vía de acceso a clases (CLASSPATH) del proceso de servidor de JDataStore. Los procedimientos almacenados y las UDF para JDataStore se deben escribir en Java. Las UDF son funciones definidas por el usuario y diseñadas para su uso en subexpresiones de una sentencia SQL.

Para obtener más información, instrucciones de uso y ejemplos, consulte “Las UDF y los procedimientos almacenados” en la *Guía del programador de JDataStore*.

Utilización de procedimientos almacenados de InterBase

En InterBase, puede utilizar procedimientos SELECT para generar un DataSet. En la base de datos de ejemplo de InterBase, `employee.gdb`, el procedimiento almacenado `ORG_CHART` es uno de los procedimientos mencionados. Para llamar a este procedimiento desde JBuilder, especifique la siguiente sintaxis en el campo SQL StatCODEent o de escape del procedimiento almacenado en el campo del editor de la propiedad `procedure` o en el código:

```
select * from ORG_CHART
```

Si desea ver procedimientos almacenados de InterBase más complicados, utilice Explorador de bases de datos para examinar los procedimientos de este servidor. `ORG_CHART` es un ejemplo interesante, que devuelve un conjunto de resultados en el que se combinan datos de varias tablas. `ORG_CHART` está escrito en el lenguaje de procedimiento y disparador de InterBase, que incluye sentencias SQL de manipulación de datos además de estructuras de control y gestión de excepciones.

Los parámetros de salida de `ORG_CHART` se convierten en columnas del DataSet generado.

Para obtener más información sobre la forma de escribir procedimientos almacenados en InterBase, consulte la documentación de InterBase Server o examine el ejemplo de procedimiento almacenado escrito en InterBase que se encuentra en [“Creación manual de tablas y procedimientos para el tutorial” en la página 6-4](#).

Utilización de parámetros con procedimientos almacenados de Oracle PL/SQL

Por el momento, un `ProcedureDataSet` sólo puede rellenarse con procedimientos almacenados de Oracle PL/SQL si se utilizan controladores JDBC de Oracle tipo 2 o tipo 4. El procedimiento almacenado al que se llame debe ser una función con el tipo de retorno `CURSOR REF`.

Siga este esquema general a la hora de utilizar procedimientos almacenados de Oracle en JBuilder:

1 Defina la función mediante PL/SQL.

A continuación puede ver un ejemplo de una descripción de función definida en PL/SQL con un tipo de devolución de CURSOR REF. En este ejemplo se presupone la existencia de una tabla llamada MyTable1.

```
create or replace function MyFct1(INP VARCHAR2) RETURN rcMyTable1 as
  type rcMyTable1 is ref cursor return MyTable1%ROWTYPE;
  rc rcMyTable1;
begin
  open rc for select * from MyTable1;
  return rc;
end;
```

2 Configure un ParameterRow para pasarlo al ProcedureDescriptor.

Debe especificarse el parámetro de entrada INP en el ParameterRow, pero no el valor de retorno especial de un CURSOR REF. JBuilder utiliza la salida del valor de retorno para rellenar datos en ProcedureDataSet. A continuación se ofrece un ejemplo de cómo puede hacerse esto con ParameterRow:

```
ParameterRow row = new ParameterRow();

row.addColumn( "INP", Variant.STRING, ParameterType.IN);

row.setString("INP", "Input Value");

String proc = "{?=call MyFct1(?)}";
```

3 Seleccione el archivo Marco en el panel de proyecto y, a continuación, seleccione la pestaña Diseño.

4 Sitúe un ProcedureDataSet de la pestaña Data Express en el panel de contenido.

5 Haga doble clic en la propiedad procedure para abrir el cuadro de diálogo ProcedureDescriptor.

6 Seleccione database1 en la lista desplegable Database.

7 Escriba la siguiente sintaxis de escape en la sentencia SQL o de escape del procedimiento almacenado, o bien en el código:

```
{?=call MyFct1(?)}
```

8 Seleccione la pestaña Parámetros del cuadro de diálogo. Seleccione el ParameterRow al que acaba de asignar el valor row.

Para obtener información acerca del lenguaje Oracle PL/SQL, consulte la documentación del servidor Oracle.

Utilización de procedimientos almacenados de Sybase

Los procedimientos almacenados creados en servidores Sybase se crean en un modo de transacción “encadenado”. Para llamar a procedimientos almacenados de Sybase como parte de un `ProcedureResolver`, debe modificarlos para que se ejecuten en un modo de transacción no encadenado. Para ello, utilice el procedimiento almacenado de sistema de Sybase `sp_procmode` para cambiar el modo de transacción a “anymode” (cualquier modo) o “unchained” (no encadenado). Si desea más detalles, consulte la documentación de Sybase.

Aplicación de ejemplo con procedimientos almacenados específicos del servidor de base de datos

En el directorio `<jbuilder>/samples/DataExpress/ServerSpecificProcedures`, puede examinar una aplicación de ejemplo con código de procedimientos almacenados para bases de datos Sybase, InterBase y Oracle.

Creación de un proveedor de datos personalizado

JBuilder simplifica la escritura de proveedores personalizados cuando se accede a datos de una fuente de datos personalizada, como SAP, BAAN, IMS, OS/390, CICS, VSAM, DB2 y otras.

La recuperación y actualización de datos de una fuente de datos, como un servidor Oracle o Sybase, se limita a dos interfaces clave: proveedores y almacenadores. Los *Proveedores* rellenan un conjunto de datos desde una fuente de datos. Los *Almacenadores* guardan los cambios en la fuente de datos. Al separar claramente la recuperación y actualización de datos con dos interfaces, resulta sencillo crear nuevos componentes proveedores/almacenadores para nuevas fuentes de datos. JBuilder proporciona implementaciones para controladores JDBC estándar que proporcionan acceso a conocidas bases de datos como Oracle, Sybase, Informix, InterBase, DB2, MS SQL Server, Paradox, dBASE, FoxPro, Access y otras. Entre otras, destacan las siguientes:

- `OracleProcedureProvider`
- `ProcedureProvider`
- `ProcedureResolver`
- `QueryProvider`
- `QueryResolver`

Es posible crear implementaciones personalizadas de componentes proveedor/almacenador para EJB, Servidor de aplicaciones, SAP, BAAN, IMS, CICS, etc.

En el directorio `/samples/DataExpress/CustomProviderResolver` de la instalación de JBuilder se encuentra un proyecto de ejemplo con un proveedor y un almacenador personalizados. El archivo de ejemplo `TestFrame.java` es una aplicación con un marco que contiene una `JdbTable` y una `JdbNavToolBar`. Ambos componentes visuales están conectados a un componente `TableDataSet`, en el que los datos se suministran desde un proveedor (Provider) personalizado (definido en el archivo `ProviderBean.java`), y se guardan con un almacenador (Resolver) personalizado (definido en el archivo `ResolverBean.java`). Esta aplicación de ejemplo lee y guarda las modificaciones en el archivo de texto `data.txt`, que es un archivo de formato simple sin delimitaciones. La estructura del archivo `data.txt` se describe en el archivo de interfaz `DataLayout.java`.

En este capítulo se explica el funcionamiento de los proveedores de datos personalizados y se muestra cómo utilizarlos para un `TableDataSet` o cualquier `DataSet` derivado de un `TableDataSet`. El método principal que hay que implementar es `provideData(com.borland.dx.dataset.StorageDataSet dataSet, boolean toOpen)`. Este método accede a los metadatos de interés y carga los datos propiamente dichos en el conjunto de datos.

Obtención de metadatos

Los metadatos son información *sobre* los datos. Son ejemplos de metadatos el nombre de columna, el nombre de tabla, si la columna es parte de la única ID de fila o no, si permite la realización de búsquedas, su precisión, escala, etc. Esta información se obtiene normalmente de la fuente de datos. Los metadatos se almacenan entonces en propiedades de los componentes `Column` asociados a los componentes de cada una de las columnas del `StorageDataSet`, y en el propio `StorageDataSet`.

Cuando se obtienen datos de una fuente de datos y se almacenan en una de las subclases de `StorageDataSet`, normalmente no sólo se obtienen filas de datos de la fuente de datos, sino también metadatos. Por ejemplo, la primera vez que se le pide a `QueryDataSet` que realice una consulta, ejecuta por defecto dos consultas: una para el descubrimiento de metadatos y la segunda para capturar filas de datos que la aplicación visualice y gestione. Las consultas posteriores realizadas por esta instancia de `QueryDataSet` únicamente realizan capturas de datos de fila. Una vez detectados los metadatos, el componente `QueryDataSet` crea automáticamente objetos `columna` (`Column`) durante la fase de ejecución, a medida que se va necesitando. Se crea una `Columna` por cada columna de resultados de consulta que no se encuentra ya en un `QueryDataSet`. Después, cada `Column` obtiene algunas propiedades de los metadatos, por ejemplo, `columnName`, `tableName`, `rowId`, `searchable`, `precision`, `scale`, etc.

Cuando esté implementando el método abstracto `provideData()` a partir de la clase `Provider`, quizás necesite añadir al `DataSet` que esté utilizando las columnas de los datos proporcionados. Puede hacerlo llamando al

método `ProviderHelp.initData()` desde dentro de su implementación `provideData()`. El proveedor debe crear una matriz de `Columnas` para entregarla al método `ProviderHelp.initData()`. He aquí una lista de las propiedades de `Columna` que un `Proveedor` debería inicializar:

- `columnName`
- `dataType`

y, optativamente:

- `sqlType`
- `precision` (utilizado por el `DataSet`)
- `scale` (utilizado por el `DataSet`)
- `rowId`
- `searchable`
- `tableName`
- `schemaName`
- `serverColumnName`

Las propiedades optativas son útiles cuando se necesita guardar de nuevo en la fuente de los datos los cambios realizados. Las propiedades `precision` y `scale` son empleadas también por los componentes `DataSet` para aspectos de presentación de datos y restricciones.

Llamada a `initData`

Los argumentos del método

`ProviderHelp.initData(com.borland.dx.dataset.StorageDataSet dataSet, com.borland.dx.dataset.Columnn[], boolean, boolean, boolean)` se explican en el siguiente texto.

- `dataSet` es el `StorageDataSet` al cual estamos proporcionando datos.
- `metaDataColumns` es la matriz de `Columnas` creadas con las propiedades adecuadas, que no es necesario añadir ni fusionar con las `Columnas` ya existentes en el `DataSet`.
- `updateColumns` especifica si se desea fusionar las columnas con columnas persistentes ya existentes que tengan el mismo valor en la propiedad `columnName`.
- `keepExistingColumns` especifica si se desea conservar alguna columna no persistente.

Si el valor de `keepExistingColumns` es `true`, se conservarán también las columnas no persistentes. Varias propiedades de columnas de la matriz de columnas se fusionan con las de las columnas existentes en el `StorageDataSet` que tiene el mismo valor en la propiedad `name`. Si el número, el tipo y la posición de las columnas es diferente, este método puede cerrar el `StorageDataSet` asociado.

La propiedad `metaDataUpdate` del `StorageDataSet` se inspecciona al llamar al `ProviderHelp.initData`. Esta propiedad determina cuáles de las propiedades

de la `Columna` prevalecerán sobre las propiedades existentes en aquellas columnas persistentes que ya figuren en el `TableDataSet` antes de llamar al `ProviderHelp.initData`. Los valores permitidos para esta propiedad están definidos en la interfaz `MetaDataUpdate`.

Obtención de datos reales

Hay algunos métodos importantes del `DataSet` que no pueden utilizarse cuando se llama al método `Provider.provideData` para abrir un `DataSet`, mientras está teniendo lugar el proceso de apertura del `DataSet`, por ejemplo el método `StorageDataSet.insertRow()`.

Para poder cargar los datos, utilice el método `StorageDataSet.startLoading`. Este método devuelve una matriz de objetos `Variant` para todas las columnas de un `DataSet`. Primero hay que definir el valor en la matriz (el método `ProviderHelp.initData` devuelve los valores ordinales de las columnas) y a continuación ir cargando cada una de las filas llamando al método `StorageDataSet.loadRow()` para terminar llamando al método `StorageDataSet.endLoading()`.

Consejos para diseñar un proveedor de datos personalizado

Un proveedor bien diseñado es capaz de reconocer las propiedades `maxRows` y `maxDesignRows` del `StorageDataSet`. Los valores posibles de estas propiedades son:

Valor	Descripción
0	proporcionar sólo información de metadatos
-1	proporcionar todos los datos
n	proporcionar un máximo de n filas

Para determinar si se llamó al método `provideData()` en la fase de diseño, llame a `java.beans.Beans.isDesignTime()`.

Cómo funciona el método `provideData()` en un conjunto de datos maestro-detalle

Se llama al método `Provider.provideData()`:

- la primera vez que se abre el `StorageDataSet` (`toOpen` tiene el valor `true`)
- cuando se llama a `StorageDataSet.refresh()`
- cuando se necesita cargar un conjunto de datos de detalle en el cual la propiedad `fetchAsNeeded` tiene el valor `true`

Cuando se necesita cargar un conjunto de datos detalle en el cual la propiedad `fetchAsNeeded` vale `true`, el proveedor ignora la propiedad `provideData` durante la apertura de los datos, o simplemente proporciona

los metadatos. El proveedor usa también los valores de los campos `masterLink` para suministrar las filas correspondientes a un determinado conjunto de datos de detalle.

Utilización de columnas

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Una `columna` es una colección de información del mismo tipo, por ejemplo, una colección de números telefónicos, de puestos de trabajo, etc. Un conjunto de componentes `Column` se gestiona mediante un `StorageDataSet`.

Un objeto `Column` se puede crear de forma explícita en el código o generarse automáticamente al instanciar la subclase `StorageDataSet`; por ejemplo, por un `QueryDataSet` cuando se ejecuta una consulta. Cada `Column` contiene propiedades que describen o gestionan la columna de datos. Algunas propiedades de `Column` almacenan *metadatos* (definidos a continuación) que se obtienen normalmente de la fuente de datos. Otras propiedades `Column` se utilizan para controlar su aspecto y edición en componentes enlazados a datos.

Nota Frecuentemente se utilizan nombres de clases abstractas o de superclases para referirse a todas sus subclases. Por ejemplo, una referencia a un objeto `StorageDataSet` implica a cualquiera (o todas, según la utilización) de sus subclases `QueryDataSet`, `TableDataSet`, `ProcedureDataSet` y `DataSetView`.

Propiedades de columnas y metadatos

La mayoría de las propiedades de una `Columna` pueden modificarse sin necesidad de cerrar y volver a abrir el `DataSet`. Sin embargo, las propiedades siguientes no pueden modificarse a menos que se cierre el `DataSet`:

- `columnName`
- `dataType`
- `calcType`
- `pickList`
- `preferredOrdinal`

El diseñador de interfaces de usuario actualizará en tiempo real las propiedades visibles de la `Columna`, como `color`, `width`, o `caption`. Si desea más información sobre la obtención de metadatos, consulte [“Obtención de metadatos” en la página 6-10](#). La obtención de datos reales se trata con más detalle en [“Obtención de datos reales” en la página 6-12](#).

Propiedades Column distintas de metadatos

Las columnas poseen propiedades que no se obtienen de los metadatos y que puedan definirse (por ejemplo, `caption`, `editMask`, `displayMask`, colores `background` y `foreground` y `alignment`). Normalmente, se pretende que este tipo de propiedades controlen tanto el aspecto por defecto de este elemento de datos en los componentes enlazados a datos como la edición por parte del usuario. Las propiedades que ha de fijar el desarrollador suelen ser de este otro tipo.

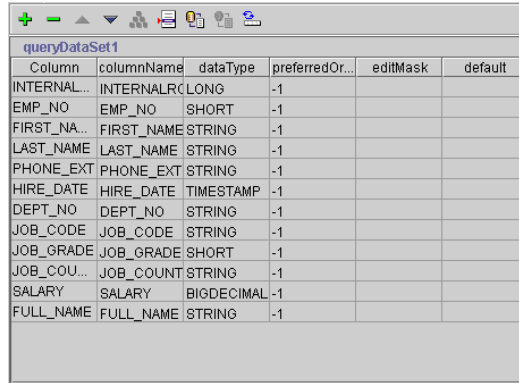
Visualización de información de columnas en el diseñador de columnas

Una forma de visualizar la información de las propiedades de la columna es utilizar el diseñador de columnas. En el diseñador de columnas aparece información sobre las propiedades seleccionadas, como el tipo de datos de la columna, dentro de una rejilla que puede recorrerse. Si se modifica o define una propiedad en el diseñador de columnas, la columna afectada se vuelve persistente. Las propiedades de la columna pueden modificarse desde el diseñador de columnas o desde el Inspector. Puede cambiar las propiedades que se muestran en el diseñador de columnas pulsando el botón **Propiedades**.

Para abrir el diseñador de columnas:

- 1 Abra cualquier proyecto que incluya un objeto `DataSet`.
En este ejemplo, abra `/samples/DataExpress/QueryProvider/QueryProvider.jpx` del directorio de instalación de JBuilder.
- 2 En el panel del proyecto, haga doble clic sobre el archivo `Marcol.java` y pulse sobre la ficha **Diseño** situada en la parte inferior del panel derecho del Visualizador de aplicaciones.
- 3 Pulse con el botón secundario del ratón sobre el objeto `queryDataSet1`, situado en el panel de contenido, y seleccione **Activar el diseñador** en el menú contextual.

Con ello aparecerá el diseñador de columnas mostrando el conjunto de datos en la ventana Diseño, que es parecido a éste cuando se aplica a la tabla de ejemplo EMPLOYEE:



Column	columnName	dataType	preferredOr...	editMask	default
INTERNAL...	INTERNALRC	LONG	-1		
EMP_NO	EMP_NO	SHORT	-1		
FIRST_NA...	FIRST_NAME	STRING	-1		
LAST_NAME	LAST_NAME	STRING	-1		
PHONE_EXT	PHONE_EXT	STRING	-1		
HIRE_DATE	HIRE_DATE	TIMESTAMP	-1		
DEPT_NO	DEPT_NO	STRING	-1		
JOB_CODE	JOB_CODE	STRING	-1		
JOB_GRADE	JOB_GRADE	SHORT	-1		
JOB_COU...	JOB_COUNT	STRING	-1		
SALARY	SALARY	BIGDECIMAL	-1		
FULL_NAME	FULL_NAME	STRING	-1		

Para asignar una propiedad a una columna, seleccione esa *Columna* e introduzca o seleccione un nuevo valor para la propiedad. El Inspector se actualiza para reflejar las propiedades (y los sucesos) de la columna seleccionada. Por ejemplo:



- 1 Pulse el botón **Propiedades** para mostrar el cuadro de diálogo **Propiedades mostradas**.
- 2 Seleccione la propiedad *min*, para que aparezca en el diseñador de columnas, y pulse **Aceptar**.
- 3 Acceda a la columna *min*, e introduzca la fecha de hoy en el campo *HIRE_DATE*.
- 4 Pulse *Intro* para cambiar el valor.

Para cerrar el diseñador de columnas, seleccione cualquier componente de interfaz de usuario en el panel de estructura, o haga clic con el botón derecho del ratón en un componente distinto, y seleccione **Activar el diseñador**. En otras palabras, la única manera de cerrar un diseñador es abriendo otro.

Consulte el tema [“Persistencia de los datos” en la página 12-26](#) si desea más información sobre la utilización del diseñador de columnas.

El botón Generar clase RowIterator

El Generador de RowIterator del Diseñador de columnas se puede utilizar para crear una nueva clase *RowIterator* o actualizar una clase *RowIterator* existente de un conjunto de datos. El Generador examina la propiedad *columnName* de todas las columnas del conjunto de datos y genera métodos *get* y *set* para todas las columnas.

Al pulsar el botón Generador de RowIterator, se abre un cuadro de diálogo que proporciona capacidades abreviadas de iteración (baja utilización de la memoria y vinculación rápida) para asegurar un acceso seguro de tipo estático a las columnas.

Las opciones del cuadro de diálogo Iterador de fila tienen los siguientes propósitos:

Tabla 7.1 Cuadro de diálogo Generador de RowIterator

Opción	Descripción
Ampliar RowIterator	Si está activada, la clase generada amplía el RowIterator. Esto muestra todos los métodos del RowIterator. Si esto es falso, se crea una nueva clase con un miembro RowIterator, el cual se delega para todas las operaciones. La ventaja de no ampliar RowIterator es que su clase de iterador puede controlar lo que se expone. La ventaja de ampliar RowIterator es que se necesita generar menos código, debido a que los métodos de vinculación y desplazamiento se heredan y no hay que delegarlos.
Eliminar subrayado; Poner en mayúscula la letra siguiente	Esto afecta a la forma en que se generan los nombres de métodos get y set a partir de la propiedad columnName de la columna. Si esta opción está activada, se elimina el subrayado y el carácter que sigue al subrayado se pone en mayúsculas.
Generar métodos de vinculación	Genera métodos de delegación para llamar a los métodos de asociación incrustados RowIterator.
Generar métodos de desplazamiento	Genera métodos de delegación para llamar a los métodos de desplazamiento incrustados RowIterator.

Si desea más información sobre los iteradores de fila, consulte la *DataExpress Component Library Reference*.

Utilización del diseñador de columnas para convertir metadatos en persistentes

El botón Hacer persistentes todos los metadatos del diseñador de columnas permite convertir en persistentes todos los metadatos que sean necesarios para abrir un QueryDataSet durante la ejecución.

Para ello, se realizan los siguientes cambios en el código fuente:

- Se modifica la consulta asociada al QueryDataSet para incluir las columnas identificadoras de fila.
- Se asigna el valor NONE a la propiedad metaDataUpdate del QueryDataSet.
- Se definen las propiedades tableName, schemaName, y resolveOrder del QueryDataSet, si es necesario.

- Se definen todas las columnas como persistentes, y se especifican otras propiedades diversas. En concreto, `precision`, `scale`, `rowId`, `searchable`, `tableName`, `schemaName`, `hidden`, `serverColumnName`, y `sqlType`.

JBuilder captura los metadatos automáticamente. Dado que algunos controladores JDBC tardan un tiempo considerable en responder a las solicitudes de metadatos, puede ser conveniente hacer persistentes los metadatos y dar instrucciones a DataExpress de no capturarlos. Delegando esta tarea en JBuilder en la fase de diseño, y generando el código necesario para la fase de ejecución, se obtiene una mejora de velocidad.

Consulte

- [“Metadatos persistentes de consulta” en la página 5-11.](#)

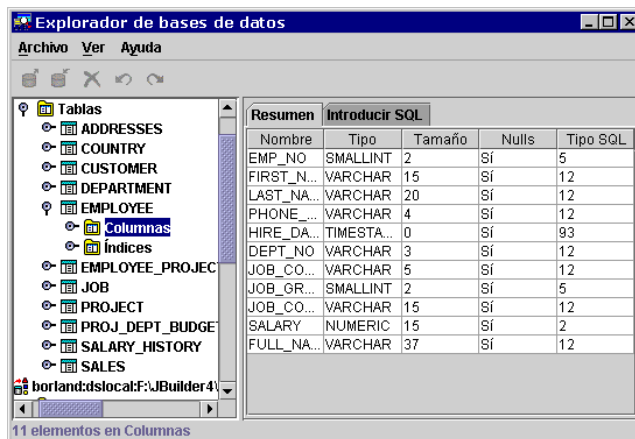
Cómo convertir metadatos en dinámicos con el Diseñador de columnas

- Advertencia** Si pulsa el botón Hacer dinámicos todos los metadatos, se ELIMINARÁ EL CÓDIGO del archivo fuente. Con ello desaparecerá todo el código de los valores de propiedades mencionados en el tema anterior, así como cualquier modificación de las propiedades relacionadas con metadatos nombradas anteriormente. Sin embargo, otras propiedades, como `editMask`, quedarán intactas.
- Nota** Para actualizar una consulta después de que una tabla pueda haber sido modificada en el servidor, primero hay que convertir los metadatos en dinámicos y después en persistentes, para utilizar los nuevos índices creados sobre la tabla de base de datos.

Ver información de columnas en el Explorador de base de datos

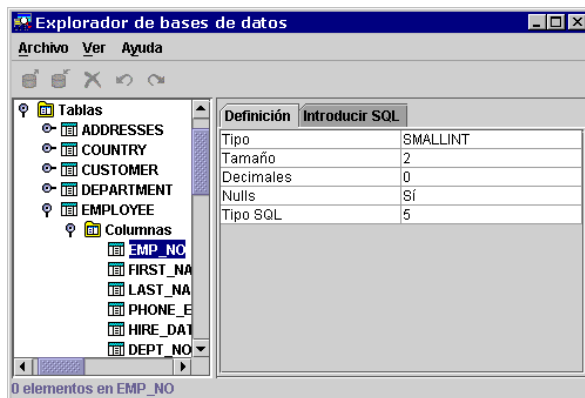
El Explorador de bases de datos es un visor de bases de datos jerárquicas, desarrollado íntegramente en Java, que permite también modificar los datos. Muestra información basada en JDBC sobre los metadatos de la base de datos, dentro de una ventana con dos paneles. El panel de la izquierda muestra un árbol que representa jerárquicamente un conjunto de bases de datos y sus tablas, vistas, procedimientos almacenados y metadatos asociados. El panel derecho muestra en varias fichas información descriptiva acerca de cada nodo del árbol.

Abra el Explorador de bases de datos seleccionando Explorador de bases de datos en el menú Herramientas de JBuilder.



Cuando abra la dirección de una URL de base de datos, podrá expandir el árbol para que aparezcan los objetos dependientes. Las columnas son objetos descendientes de una determinada tabla de base de datos. Como se muestra en la figura anterior, cuando se selecciona el objeto columna asociado a una tabla, en el panel derecho aparece la página de resumen, con una lista de las columnas, sus tipos de datos, tamaño y otras informaciones.

Si selecciona una columna del panel izquierdo, aparecerá únicamente la información asociada a este campo, como en la figura siguiente.



Si desea más información acerca de la utilización del Explorador de bases de datos, consulte su ayuda en pantalla

Optimización de una consulta

Este apartado contiene información sobre cómo utilizar columnas para mejorar el rendimiento de las consultas.

Configuración de las propiedades de columna

Se pueden definir propiedades `Column` mediante las herramientas de diseño visual de JBuilder o, manualmente, en el código. Cualquier columna que defina o modifique con las herramientas de diseño visuales será persistente.

Configuración de las propiedades de columna mediante las herramientas de diseño visual de JBuilder

El Inspector de componentes permite trabajar con las propiedades `Column`. Para definir propiedades `Column`:

- 1 Abra (o cree) un proyecto que contenga un `StorageDataSet` con el que desee trabajar. Si está creando un nuevo proyecto, consulte [“Consultas en bases de datos” en la página 5-2](#).
- 2 Abra el diseñador de interfaces de usuario haciendo doble clic en el objeto `Frame` del contenedor del panel de proyecto y haciendo clic en la pestaña Diseño del Visualizador de aplicaciones.
- 3 En el panel de contenido, seleccione el componente `StorageDataSet`.
- 4 Pulse el botón de ampliación contiguo a `StorageDataSet` para mostrar sus columnas.
- 5 Seleccione la `Column` con la que desea trabajar. El Inspector visualiza las propiedades y los sucesos de la columna. Defina las propiedades que desee.

Definición de propiedades en el código

Para definir manualmente en el código fuente las propiedades de una o más columnas de un `StorageDataSet`:

- 1 Proporcione datos al `StorageDataSet`.
Por ejemplo, ejecute una consulta con un componente `QueryDataSet`. En [“Consultas en bases de datos” en la página 5-2](#) puede encontrar un ejemplo.
- 2 Obtenga una matriz de referencias a los objetos `Column` ya existentes en el `StorageDataSet`, mediante una llamada al método `getColumn(java.lang.String)` de `ReadRow`.

- 3 Identifique con qué columna o columnas de la matriz desea trabajar leyendo sus propiedades, por ejemplo con la propiedad `getColumnName()` del componente `Column`.
- 4 Establezca las propiedades en las columnas adecuadas cuando sea necesario.

Nota Se debe asignar a la propiedad `persist` de la columna el valor **true** cuando se desee que los valores de las propiedades permanezcan vigentes la próxima vez que se suministren los datos. Esto se describe en el apartado siguiente.

Columnas persistentes

Una columna persistente es un objeto `Column` que ya forma parte de un `StorageDataSet` y a cuya propiedad `persist` se le ha asignado el valor **true** antes de suministrar los datos. Si la propiedad `persist` se define después de haber proporcionado los datos, deberá ejecutar otro comando `setQuery` con un nuevo `queryDescriptor` asociado a esa aplicación para identificar las columnas que son persistentes. Definiendo una `Column` como persistente, pueden mantenerse los valores de sus propiedades durante todo el transcurso de una operación de suministro de datos. Una columna persistente no hace que los datos de dicha columna de las filas de datos permanezcan inalterados durante todas las operaciones de suministro de datos.

Normalmente, un `StorageDataSet` crea automáticamente nuevos objetos `Column` por cada columna encontrada en los datos suministrados por la fuente de datos. Descarta todos los objetos `Column` explícitamente añadidos con anterioridad o creados automáticamente para un grupo de datos anterior. Esta acción de descartar los objetos `Column` anteriores provoca la pérdida de las configuraciones de propiedad en la antigua `Column` que quizá se deseen conservar.

Para evitarlo, marque un objeto `Column` como persistente asignando a su propiedad `persist` el valor **true**. Cuando una columna es persistente, no se descarta cuando suministran nuevos datos al `StorageDataSet`. En vez de ello, el objeto `Column` existente se utiliza de nuevo para controlar la misma columna en los datos recién suministrados. La concordancia de columnas se realiza por su nombre.

Cualquier columna que defina o modifique con las herramientas de diseño visuales será persistente. Las columnas persistentes se tratan con más detalle en [“Persistencia de los datos” en la página 12-26](#). Se pueden crear objetos `Column` explícitamente y adjuntarlos al `StorageDataSet`, mediante `addColumn()`, para añadir una sola columna o `setColumns()` para añadir varias columnas nuevas de una vez.

Cuando se utiliza `addColumn`, se debe establecer `Column` como persistente, antes de obtener datos de la fuente de datos, o se perderán las

configuraciones de las propiedades de la columna durante el suministro. La propiedad `persist` se define automáticamente con el método `setColumns`.

Nota El diseñador de interfaz llama al método `StorageDataSet.setColumns()` cuando se trabaja con columnas. Si desea cargar y modificar la aplicación en el diseñador de interfaz, utilice el método `setColumns` de forma que se puedan reconocer las columnas en fase de diseño. En fase de ejecución no hay ninguna diferencia entre `setColumns` y `addColumn`.

Combinación de metadatos dinámicos con columnas persistentes

Durante la fase de suministro, `StorageDataSet` obtiene primero metadatos de la fuente de datos, si es posible. Estos metadatos se utilizan para actualizar todas las columnas persistentes existentes que coincidan y para crear otras columnas que pueden ser necesarias. La propiedad `metaDataUpdate` de la clase `StorageDataSet` controla la medida de la actualización de los metadatos en las columnas persistentes.

Eliminación de columnas persistentes

Esta sección describe cómo deshacer una columna persistente para que una consulta modificada no devuelva más las columnas (no deseadas) del `StorageDataSet`.

Cuando se dispone de un `QueryDataSet` o de una `TableDataSet` con columnas persistentes, se declara que estas columnas han de existir en el `DataSet` resultante, aunque ya no existan en la fuente de datos correspondiente. Pero, ¿qué ocurre si no necesita ya estas columnas persistentes?

Cuando se altera la cadena de consulta de un `QueryDataSet`, las columnas persistentes no se pierden. En vez de ello, las nuevas columnas, obtenidas con la consulta en ejecución, se añaden a la lista de columnas. Cualquiera de estas nuevas columnas se puede hacer persistente estableciendo cualquiera de sus propiedades.

Nota Cuando se amplía un `StorageDataSet` haciendo clic en botón de ampliación del panel de estructura, la lista de columnas no cambia automáticamente al cambiar la cadena de consulta. Para actualizar la lista de columnas de acuerdo con los resultados de la consulta modificada, haga doble clic en `QueryDataSet` del panel de contenido. Esto ejecuta otra vez la consulta y añade todas las nuevas columnas encontradas en la consulta modificada.

Para eliminar una columna persistente que ya no necesite, selecciónela en el panel de contenido y pulse *Supr* o selecciónela en el diseñador de columnas y haga clic en el botón Borrar de la barra de herramientas. Esto provoca las acciones siguientes:

- Se marca la columna como no persistente.

- Se elimina todo código que establezca propiedades en esta columna.
- Se elimina toda lógica de manejadores de sucesos situada en esta columna.

Para verificar que una columna persistente eliminada ya no forma parte del `QueryDataSet`, haga doble clic en el conjunto de datos en el panel de contenido. Esto ejecuta de nuevo la consulta y visualiza todas las columnas en el `QueryDataSet` resultante.

Utilización de columnas persistentes para añadir columnas vacías a un DataSet

Ocasionalmente se puede desear añadir una o más columnas adicionales a un `StorageDataSet`, columnas no suministradas desde la fuente de datos y con las que no se necesita resolución en la fuente de datos. Por ejemplo, podría añadir columnas adicionales en las siguientes circunstancias y de esta forma:

- Adición de una columna adicional para objetivos internos. Si desea ocultar la columna en los componentes enlazados a datos, asigne a la propiedad `visible` de `Column` el valor `false`.
- Construir un nuevo `DataSet` manualmente, añadiendo las columnas deseadas antes de realizar el cómputo de los datos almacenados en sus filas.
- Construir un nuevo `DataSet` para almacenar los datos de una fuente de datos personalizada que no admitan los suministradores de JBuilder y, por lo tanto, no suministre metadatos automáticamente.

En estos casos, se puede añadir de forma explícita una columna al `DataSet`, antes o después de suministrar los datos. El `columnName` debe ser único y no se puede duplicar un nombre ya existente en los datos suministrados. Adicionalmente, si se suministran datos después de añadir la columna, cerciórese de marcar `Column` como persistente para que no deseche cuando se suministren nuevos datos.

Para añadir manualmente una columna en el código fuente, siga las instrucciones que figuran en [“Columnas persistentes” en la página 7-8](#).

Para añadir una columna manualmente mediante las herramientas de diseño visual de JBuilder, realice los pasos siguientes:

- 1 Siga los tres primeros pasos de [“Configuración de las propiedades de columna mediante las herramientas de diseño visual de JBuilder” en la página 7-7](#) para colocar los metadatos en las columnas enumeradas en el panel de contenido.

Si desea añadir columnas a un `DataSet` vacío, puede omitir estos pasos.

- 2 Seleccione `<nueva columna>`.

Esta opción aparece al final de la lista de columnas.

- 3 En el Inspector, establezca `columnName`, asegurándose de que es diferente de los nombres de columna existentes.
- 4 Establezca todas las propiedades necesarias de la nueva columna.

JBuilder crea el código del nuevo objeto `Column` persistente y lo adjunta al `DataSet`. El nuevo `Column` existe incluso antes de que se suministren los datos. Dado que su nombre difiere de cualquier otro nombre de columna, este `Column` no se rellena con datos durante la fase de suministro; todas las filas de este `Column` tienen valores `null` (nulos).

Control del orden de columnas en un DataSet

Cuando se suministran datos a un `StorageDataSet`, éste realiza las siguientes acciones:

- Borra todas las columnas no persistentes, trasladando las columnas persistentes a la izquierda
- Fusiona las columnas procedentes de los datos suministrados con las columnas persistentes. Si una columna persistente tiene el mismo nombre y tipo de datos que una columna suministrada, se considerará que es la misma.
- Coloca las columnas suministradas en el conjunto de datos en el orden especificado en la consulta o procedimiento.
- Añade las columnas restantes (las que sólo están definidas en la aplicación) en el orden en que estén definidas en el método `setColumns()` del `DataSet`.
- Intenta mover hasta el lugar deseado todas las columnas cuya propiedad `preferredOrdinal` tenga asignado ese valor. (Si las dos columnas tienen el mismo valor en la propiedad `preferredOrdinal`, no será posible hacerlo.)

Eso significa que:

- Las columnas que están definidas en la aplicación, y que no son suministradas por la columna o el procedimiento, aparecerán después de las columnas que sí hayan sido suministradas.
- Si se definen las propiedades de algunas columnas (tanto si son suministradas como si están definidas) en una aplicación, pero no en otras, su orden no cambiará.
- Se puede cambiar la posición de cualquier columna modificando su propiedad `preferredOrdinal`. Las columnas cuya propiedad `preferredOrdinal` no está definida mantienen sus posiciones relativas.

Almacenamiento de cambios en la fuente de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Después de recuperar los datos de una fuente de datos y modificarlos en `StorageDataSet`, es recomendable volver a guardarlos en la fuente de datos. Todos los cambios realizados en un `DataSet` pueden incorporarse a la fuente de datos, por ejemplo, un servidor SQL. Este proceso se denomina *almacenamiento*. Hay una compleja tecnología para conciliación de datos que trata los posibles conflictos de edición.

Entre el momento en que se recupera el subconjunto local de datos de una fuente de datos, y el momento en que se intentan guardar las actualizaciones en la fuente de datos, pueden originarse diversas situaciones que deben ser gestionadas por la lógica de almacenamiento. Por ejemplo, puede ocurrir que al intentar guardar los cambios, otro usuario haya actualizado la misma información en el servidor. ¿Debe el almacenador guardar la nueva información a pesar de todo? ¿Debe visualizar la información actualizada del servidor y compararla con sus actualizaciones? ¿Debe desechar los cambios? Según sea la aplicación, variarán las necesidades de reglas de almacenamiento.

La lógica de almacenamiento de actualizaciones puede ser bastante compleja. Cuando se guardan los cambios pueden ocurrir errores como transgresiones de la restricciones de integridad del servidor y los conflictos de almacenamiento. Puede aparecer un conflicto en el almacenamiento; por ejemplo, al borrar una fila que ya estaba borrada o al actualizar una fila que había actualizado otro usuario. JBuilder proporciona gestión por defecto de estos errores situando el `DataSet` en la fila afectada (si no se ha borrado) y visualizando el error encontrado en un cuadro de diálogo de mensaje.

Al almacenar cambios en la fuente de datos, estos cambios se procesan normalmente por grupos denominados *transacciones*. El dispositivo de DataExpress utiliza una única transacción para guardar por defecto en la fuente de datos todas las inserciones, actualizaciones y eliminaciones realizadas en el `DataSet`. Para ofrecer mayor control, JBuilder permite modificar el proceso de transacción por defecto.

DataExpress proporciona también un almacenador genérico formado por clases e interfaces básicas. Este dispositivo puede ampliarse para proporcionar un comportamiento almacenador personalizado cuando sea necesario mayor control sobre la fase de almacenamiento. Este dispositivo genérico también permite crear almacenadores para fuentes de datos no JDBC que normalmente no admiten procesos de transacción.

En los apartados siguientes se describen ambas opciones de almacenamiento de datos:

- [“Almacenamiento de cambios desde un QueryDataSet” en la página 8-3](#) versa sobre la gestión de almacenamiento básico proporcionada por DataExpress y su proceso de transacción por defecto.

Cuando se ha establecido una relación maestro-detalle entre dos o más conjuntos de datos, es preciso utilizar procedimientos especiales de almacenamiento. Para obtener más información, consulte [“Almacenamiento de los cambios en una relación maestro-detalle” en la página 9-11](#).

- [“Almacenamiento de cambios en las fuentes de datos con procedimientos almacenados” en la página 8-5](#) explica el almacenamiento en la fuente de datos de los cambios efectuados en un `ProcedureDataSet`.
- [“Almacenamiento de datos de varias tablas” en la página 8-12](#) proporciona los parámetros necesarios para almacenar cambios cuando una consulta incluye a más de una tabla.
- [“Utilización de conjuntos de datos con RMI \(conjuntos de datos transportables\)” en la página 8-15](#) proporciona un método para hacer fluir los datos de un `DataSet` mediante la creación de un objeto Java (`DataSetData`) que contenga los datos de un `DataSet`.
- [“Personalización de la lógica del almacenador por defecto” en la página 8-17](#) describe la forma de definir las reglas de almacenamiento personalizadas mediante un componente `QueryResolver` y sucesos de almacenamiento.
- [“Exportación de datos” en la página 3-4](#) describe cómo exportar datos a un archivo de texto.

Almacenamiento de cambios desde un QueryDataSet

Se pueden utilizar distintas implementaciones de *almacenador* para guardar los cambios en la fuente de datos. `QueryDataSets` utiliza un objeto `QueryResolver` para guardar los cambios, por defecto. El almacenador por defecto se puede redefinir mediante la propiedad `StorageDataSet.resolver`. Cuando se suministran datos al conjunto de datos, `StorageDataSet` rastrea la información de estado (eliminada, insertada o actualizada) de todas las filas. Cuando los datos se *resuelven* en una fuente de datos como, por ejemplo, un servidor SQL, la información de estado de fila se utiliza para determinar qué filas de la tabla SQL añadir, borrar o modificar. Cuando una fila se ha resuelto correctamente, obtiene el nuevo estado de resuelta (`RowStatus.UPDATE_RESOLVED`, `RowStatus.DELETE_RESOLVED` o `RowStatus.INSERT_RESOLVED`). Si el `StorageDataSet` se resuelve de nuevo, las filas resueltas anteriormente se omiten, a menos que existan cambios posteriores a la resolución.

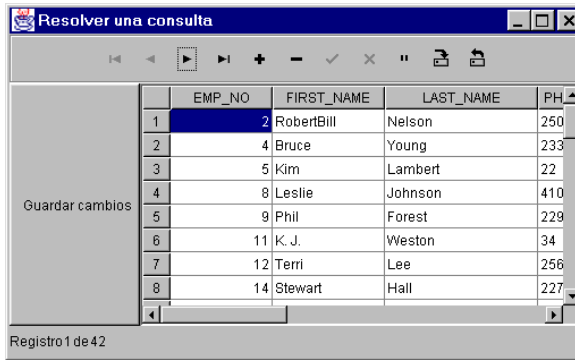
Este tema explora la funcionalidad de los almacenadores básicos proporcionados por JBCL. Amplía los conceptos explicados en [“Consultas en bases de datos” en la página 5-2](#) a la fase de resolución en la que se guardan los cambios en la fuente de datos.

Para poder avanzar en este ejemplo, comience por los archivos de ejemplo ya finalizados que se encuentran en el directorio `/samples/DataExpress/QueryProvider` o cree la aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

[“Consultas en bases de datos” en la página 5-2](#) explica la fase de suministro, cuando se obtienen los datos de la fuente de datos. En este material se enseña a instanciar un `QueryDataSet` y los componentes de la interfaz de usuario asociados, y muestra los datos recuperados del archivo `employee` de ejemplo de `JDataStore`. El botón Guardar de la barra de herramientas `JdbNavToolBar` se puede utilizar para guardar los cambios efectuados en los datos en el archivo `employee`. En el siguiente apartado se añade un botón que también ejecuta un código de almacenamiento básico. Cuando se pulsa el botón personalizado o el botón Guardar de la barra de herramientas, los cambios realizados en los datos de `QueryDataSet` se guardan en el archivo de datos de empleado mediante el `QueryResolver` por defecto del `QueryDataSet`.

Cómo añadir un botón para guardar los cambios de un QueryDataSet

El código fuente de la aplicación completa se encuentra en el directorio `/samples/DataExpress/QueryResolver` de la instalación de `JBUILDER`. La aplicación en ejecución presentará este aspecto:

Figura 8.1 Interfaz de usuario para guardar cambios desde un QueryDataSet

Para crear esta aplicación:

- 1 Cree una sencilla aplicación de base de datos, según se describe en [“Suministro de datos de los ejemplos” en la página 11-2](#).

Si ya ha creado la aplicación, simplemente ábrala. Si no ha seguido el tutorial, puede encontrar los archivos de proyecto completos en el directorio `/samples/DataExpress/QueryProvider` de la instalación de JBuilder.

Nota Puede ahorrar tiempo haciendo copias de seguridad de estos archivos antes de modificarlos, ya que otros ejemplos de este manual utilizan como punto de partida la aplicación simple de base de datos que se crea en [“Suministro de datos de los ejemplos” en la página 11-2](#).

- 2 Seleccione el archivo Marco en el panel de contenido.
- 3 Añada un componente `JButton` de la pestaña Swing de la paleta de componentes. Asigne a la propiedad `text` del botón el valor `Guardar cambios`. (Consulte, al principio de este ejemplo, la aplicación completa para obtener información sobre la colocación general de los controles en la interfaz de usuario.)
- 4 Compruebe que el componente `JButton` sigue seleccionado y abra la pestaña Sucesos del inspector. Seleccione el método `actionPerformed()` y haga doble clic en él. Con ello se cambia el foco del Visualizador de aplicaciones del diseñador de interfaces de usuario al panel de código fuente y se visualiza la función vacía del método `actionPerformed()`.

Añada el siguiente código al método `actionPerformed()`:

```
try {
    databasel.saveChanges(queryDataSet1);
    System.out.println("Save changes succeeded");
}
catch (Exception ex) {
    // mostrar la excepción en JdbStatusLabel si la
    // aplicación la contiene,
```



```
// de lo contrario, mostrar un cuadro de diálogo de error
DBExceptionHandler.handleException(ex); }
```

Si utiliza nombres diferentes para las instancias de los objetos, por ejemplo, `database1`, reemplácelos según proceda.

- 5 Ejecute la aplicación seleccionando Ejecutar | Ejecutar proyecto. La aplicación se compila y visualiza en ventanas independientes. Los datos se visualizan en una tabla, con un botón Guardar cambios, la barra de herramientas y una etiqueta de estado que informa de la posición de la fila actual y del número de filas.

Si se produce algún error, aparece un panel de error que indica la línea donde se produjo. Ya que el código del botón personalizado es la fuente de errores más probable, le sugerimos que compruebe que el código anterior se introdujo correctamente. Realice las correcciones necesarias para ejecutar la aplicación en ésta y en otras áreas.

Cuando se ejecuta la aplicación, se observa el siguiente comportamiento:

- Utilice el teclado, el ratón o la barra de herramientas para desplazarse por los datos visualizados en la tabla. La etiqueta de estado se actualiza al ir navegando.
- Se puede redimensionar la ventana para visualizar más campos o desplazarse con la barra de desplazamiento horizontal.



Realice cambios en los datos visualizados en la tabla insertando, borrando y actualizándolos. Puede guardar los cambios en el servidor pulsando el botón Guardar cambios creado o el botón Guardar de la `JdbNavToolBar`.

Nota

Debido a las restricciones de datos en la tabla de empleados, la operación de guardar puede fracasar dependiendo de los datos que se cambien. Dado que otras modificaciones pueden devolver errores, es recomendable realizar cambios sólo en los valores del nombre (`FIRST_NAME`) y el apellido (`LAST_NAME`) de las filas existentes hasta que se familiarice con las restricciones de esta tabla.

Almacenamiento de cambios en las fuentes de datos con procedimientos almacenados

Se pueden utilizar distintas implementaciones de *almacenador* para guardar los cambios en la fuente de datos. `QueryDataSets` utiliza un objeto `QueryResolver` para guardar los cambios, por defecto. El almacenador por defecto se puede redefinir mediante la propiedad `StorageDataSet.resolver`.

Este tema trata sobre la funcionalidad básica del almacenador proporcionada por el paquete `DataExpress` para los componentes `ProcedureDataSet`. Amplía los conceptos que se exploran en [Capítulo 6, “Utilización de procedimientos almacenados”](#) explorando los distintos métodos para guardar los cambios efectuados en la fuente de datos.

En el [Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados”](#) se explica la forma de ejecutar un procedimiento almacenado para recuperar datos. El tutorial crea una tabla en el servidor y después crea procedimientos de inserción, actualización y eliminación que proporcionan información sobre la forma de almacenar los cambios en el origen. Este tutorial utiliza el IDE de JBuilder para crear una instancia del componente `ProcedureDataSet` y los componentes de la interfaz asociados, y muestra en una tabla los datos que devuelve `JDataStore`. El botón Guardar de la barra de herramientas `JdbNavToolBar` se puede utilizar para guardar los cambios efectuados en los datos en el archivo `employee`, después de definir ciertas propiedades.

En este apartado se amplía el tutorial sobre recuperación por medio de la adición de capacidad de almacenamiento básica. Es posible hacerlo de dos maneras con un componente `ProcedureDataSet`. Los siguientes apartados tratan ambas opciones con más detalle.

- Un botón que activa un código básico de almacenamiento o una `JdbNavToolBar` cuyo botón Guardar también lleve a cabo una función de almacenamiento básico. Consulte [“Guardar cambios por medio de QueryResolver” en la página 8-6](#).
- Un `ProcedureResolver` que necesite codificación especial del procedimiento almacenado para la base de datos en la que deberán almacenarse éstos. Dispone de un ejemplo de este sistema en el [“Guardar cambios con un ProcedureResolver” en la página 8-9](#).

Guardar cambios por medio de QueryResolver

Si la propiedad `resolver` de un `ProcedureDataSet` no se encuentra definida, el almacenador por defecto es un objeto `QueryResolver` que genera consultas INSERT, UPDATE y DELETE para guardar los cambios. `QueryResolver` requiere que las propiedades `tableName` y `rowID` se encuentren definidas. En este ejemplo se explica la forma de hacerlo.

La aplicación terminada de este ejemplo está disponible como proyecto completo en el subdirectorio `/samples/DataExpress/SimpleStoredProcedure` del directorio de instalación de JBuilder. Puede encontrar otras aplicaciones de muestra que hacen referencia a procedimientos almacenados en varios servidores en el directorio `/samples/DataExpress/ServerSpecificProcedures/`.

Para completar la aplicación y guardar los cambios en la tabla `COUNTRY`:

- 1 Seleccione Archivo | Cerrar si tiene proyectos abiertos. Seleccione Archivo | Abrir. Abra el archivo de proyecto creado para el tutorial del [Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados.”](#) Se le añadirá la capacidad de almacenamiento al proyecto existente.

Con lo hecho hasta ahora, ya es posible ejecutar la aplicación y visualizar y desplazarse por los datos. No obstante, para poder introducir, actualizar o eliminar registros, es necesario suministrar más información a `QueryResolver` como se muestra a continuación. Por defecto se llama al `QueryResolver`, a menos que se haya definido un `ProcedureResolver` (consulte [“Guardar cambios con un ProcedureResolver” en la página 8-9](#)). A continuación, realice el siguiente procedimiento:

- 2 Seleccione `Marco1.java` en el panel del proyecto. Active el diseñador de interfaces de usuario, seleccionando la pestaña Diseño.
- 3 Seleccione `procedureDataSet1` en el árbol de componentes.
- 4 En el Inspector, asigne a la propiedad `tableName` de `procedureDataSet1` el valor `“COUNTRY”`.
- 5 Verifique que la propiedad `resolvable` del `procedureDataSet1` tiene el valor `True`.
- 6 En el panel de proyecto, haga clic en el icono de ampliación que se encuentra a la izquierda de `procedureDataSet1` para exponer las columnas del conjunto de datos.
- 7 Seleccione la columna clave denominada `COUNTRY`.
- 8 Asigne a la propiedad `rowID` de la columna `COUNTRY` el valor `True`.
- 9 Elija Ejecutar | Ejecutar proyecto para ejecutar la aplicación.

La aplicación se compila y visualiza en ventanas independientes. Los datos se visualizan en una tabla, con una barra de herramientas y una etiqueta de estado que informa de la posición de la fila actual y del número de filas. Ahora se pueden introducir, actualizar o eliminar registros y guardar los cambios en la base de datos.

Cuando se ejecuta la aplicación, se observa el siguiente comportamiento:

- Utilice el teclado, el ratón o la barra de herramientas para desplazarse por los datos visualizados en la tabla. La etiqueta de estado se actualiza al ir navegando.
- Se puede redimensionar la ventana para visualizar más campos o desplazarse con la barra de desplazamiento horizontal.

En el ejemplo anterior, se podría añadir un `JButton` codificado de manera que se encargara de guardar los cambios en lugar de `JdbNavToolBar`. Para obtener más información acerca de cómo hacer esto, consulte [“Cómo añadir un botón para guardar los cambios de un QueryDataSet” en la página 8-3](#). Con el botón control seleccionado en el árbol de componentes, seleccione la pestaña Sucesos del Inspector, el método `actionPerformed()`, haga doble clic en su campo de valor e introduzca este código en la ventana de código fuente:

```

try {
    database1.saveChanges(procedureDataSet1);
    System.out.println("Save changes succeeded");
}
catch (Exception ex) {
    // mostrar la excepción en JdbStatusLabel si
    // la aplicación la contiene;
    // de lo contrario, mostrar un cuadro de diálogo de error
    DBExceptionHandler.handleException(ex); }

```

Si utiliza nombres diferentes para las instancias de los objetos, por ejemplo, `database1`, reemplácelos según proceda.

Codificación de procedimientos almacenados para gestionar el almacenamiento

Para utilizar un `ProcedureResolver`, necesitará implementar tres procedimientos almacenados en la base de datos y especificarlos como propiedades del `ProcedureResolver`. Estos tres procedimientos son los siguientes:

- `insertProcedure`, al que se llama para cada fila que se ha de insertar en el `DataSet`. Los parámetros disponibles para llamar a `insertProcedure` son:
 - La fila introducida, tal como aparece en el `DataSet`.
 - El `ParameterRow` optativo indicado en el `ProcedureDescriptor`.

El procedimiento almacenado deberá diseñarse para introducir un registro en la tabla apropiada para los datos de esa fila. `ParameterRow` puede utilizarse para obtener resúmenes o para introducir parámetros opcionales.

- `updateProcedure`, al que se llama por cada fila que cambia en el `DataSet`. Los parámetros disponibles para llamar a un `updateProcedure` son:
 - La fila modificada, tal como aparece en el `DataSet`.
 - La fila original, tal como era cuando se suministraron los datos al `DataSet`.
 - El `ParameterRow` optativo indicado en el `ProcedureDescriptor`.

El procedimiento almacenado deberá diseñarse para actualizar un registro en la tabla apropiada en función de los datos originales y los modificados. Dado que la fila original y la modificada tienen los mismos nombres de columna, se ha ampliado la sintaxis de denominación de parámetro con un método para identificar la fila de datos designada. El parámetro denominado “:ORIGINAL.CUST_ID” indica el `CUST_ID` de la fila de datos original, y “:CURRENT.CUST_ID” indica el `CUST_ID` de la fila de datos modificada. De igual modo, un parámetro “:parameter.CUST_ID” indicará el campo `CUST_ID` en una `ParameterRow`.

- `deleteProcedure`, al que se llama por cada fila eliminada del `DataSet`. Los parámetros disponibles para llamar a `deleteProcedure` son:
 - La fila original, tal como era cuando se suministraron los datos al `DataSet`.
 - El `ParameterRow` optativo indicado en el `ProcedureDescriptor`.

El procedimiento almacenado deberá diseñarse para eliminar un registro en la tabla apropiada, en función de los datos originales de esa fila.

Un ejemplo con código que utiliza este método para almacenar datos en una base de datos se muestra en [“Guardar cambios con un ProcedureResolver” en la página 8-9](#). Para el caso particular de InterBase, consulte [“Ejemplo: Utilización de procedimientos almacenados de InterBase con parámetros de devolución” en la página 8-11](#).

Guardar cambios con un ProcedureResolver

Este ejemplo muestra cómo guardar cambios en la base de datos mediante el diseñador de interfaces de JBuilder, un componente `ProcedureDataSet` y un `ProcedureResolver`. Puede encontrar aplicaciones de muestra que hacen referencia a procedimientos almacenados en varios servidores en el directorio `/samples/DataExpress/ServerSpecificProcedures`.

Para terminar la aplicación y guardar los cambios en la tabla COUNTRY con procedimientos personalizados de inserción, actualización y borrado, abra el archivo de proyecto creado en el [Capítulo 18, “Tutorial: Recuperación de datos mediante procedimientos almacenados.”](#)

El proyecto actual contiene un componente `JdbNavToolBar`. Además de permitir el desplazamiento por la tabla, la barra de herramientas proporciona un botón Guardar cambios. En este momento, el botón utiliza un `QueryResolver`. Pero una vez que se haya proporcionado un almacenador personalizado a través del `ProcedureResolver`, el botón Guardar cambios llamará en su lugar a los procedimientos especificados de inserción, actualización y eliminación.

Con lo hecho hasta ahora, ya es posible ejecutar la aplicación y visualizar, y desplazarse por los datos. No obstante, para poder insertar, actualizar o eliminar registros, es necesario suministrar la siguiente información sobre cómo gestionar estos procesos. Con el proyecto abierto:

- 1 Seleccione el archivo Marco en el panel de contenido y abra la pestaña Diseño para activar el diseñador de interfaces.
- 2 Coloque un componente `ProcedureResolver` de la pestaña DataExpress de la paleta de componentes en el panel de contenido. Haga clic en el panel de contenido para añadir el componente a la aplicación.

- 3 Configure la propiedad `database` del `ProcedureResolver` como la base de datos instanciada, `database1` en el Inspector.
- 4 Asigne a la propiedad `deleteProcedure` el valor `DELETE_COUNTRY` de la siguiente manera:
 - a Seleccione `procedureResolver1` en el árbol de componentes y haga clic sobre la propiedad `deleteProcedure` en el Inspector.
 - b Haga doble clic sobre el campo del valor de la propiedad `deleteProcedure` para abrir el cuadro de diálogo `deleteProcedure`.
 - c Asigne a la propiedad `Database` el valor `database1`.
 - d Haga clic en Examinar procedimientos y a continuación haga doble clic en el procedimiento llamado `DELETE_COUNTRY`.

La siguiente sentencia se escribe en el campo Secuencia de escape de procedimiento almacenado o SQL:

```
execute procedure DELETE_COUNTRY :OLD_COUNTRY
```

- e Modifique la sentencia y escriba:

```
execute procedure DELETE_COUNTRY :COUNTRY
```

Consulte el texto del procedimiento en “[Creación manual de tablas y procedimientos para el tutorial](#)” en la página 6-4 o mediante el Explorador de bases de datos (Herramientas | Explorador de bases de datos).

Nota No haga clic sobre Probar procedimiento, porque este procedimiento no devuelve ningún resultado.

- 5 Asigne a la propiedad `insertProcedure` el valor `INSERT_COUNTRY`:
 - a Seleccione la propiedad `insertProcedure` de `ProcedureResolver` y haga doble clic en ella para abrir el cuadro de diálogo `insertProcedure`.
 - b Asigne al campo `Database` el valor `database1`.
 - c Pulse Examinar procedimientos y haga doble clic en el procedimiento llamado `INSERT_COUNTRY`.
 - d Edite el código generado para que quede de la siguiente forma:

```
execute procedure INSERT_COUNTRY :COUNTRY, :CURRENCY
```

Nota No haga clic sobre Probar procedimiento, porque este procedimiento no devuelve ningún resultado.

- 6 Asigne a la propiedad `updateProcedure` el valor `UPDATE_COUNTRY` de la siguiente manera:
 - a Seleccione la propiedad `updateProcedure` de `ProcedureResolver` y haga doble clic en ella para abrir el cuadro de diálogo `updateProcedure`.
 - b Asigne a la propiedad `Database` el valor `database1`.

- c Pulse Examinar Procedimientos y haga doble clic en el procedimiento llamado UPDATE_COUNTRY.
- d Edite el código generado para que quede de la siguiente forma:

```
execute procedure UPDATE_COUNTRY :ORIGINAL.COUNTRY, :CURRENT.COUNTRY,
:CURRENT.CURRENCY
```

Nota No haga clic sobre Probar procedimiento, porque este procedimiento no devuelve ningún resultado.

- 7 Seleccione procedureDataSet1 en el panel de proyecto. Asigne a la propiedad resolver el valor procedureResolver1
- 8 Seleccione procedureDataSet1. Modifique su propiedad metaDataUpdate a None.
- 9 Elija Ejecutar | Ejecutar proyecto para ejecutar la aplicación.

Cuando se ejecuta la aplicación, se pueden visualizar, editar, introducir y eliminar datos en la tabla. Guarde los cambios realizados con el botón Guardar cambios de la barra de herramientas. En este ejemplo no es posible eliminar valores existentes en la columna COUNTRY porque se ha establecido la integridad referencial. Para comprobar el procedimiento DELETE deberá añadir un valor en la columna COUNTRY y después eliminarlo.

Ejemplo: Utilización de procedimientos almacenados de InterBase con parámetros de devolución

Un procedimiento almacenado de InterBase que devuelve valores recibe llamadas distintas de cada controlador. A continuación se enumera la sintaxis de diferentes controladores para la siguiente función:

```
CREATE PROCEDURE fct (x SMALLINT)
RETURNS (y SMALLINT)
AS
BEGIN
    y=2*x;
END
```

Llamada al procedimiento fct desde distintos controladores:

- Visigenic e InterClient versión 1.3 y anteriores

```
execute procedure fct ?
```

Si se llama al procedimiento a través de un controlador JDBC directo, la salida se captura en un conjunto de resultados de una sola fila. JBuilder permite esta sintaxis para gestionar los valores de salida:

```
execute procedure fct ? returning_values ?
```

JBuilder captura el conjunto del resultado y utiliza el valor para configurar el segundo parámetro.

- InterClient versión 1.4 y posterior:

```
{call fct(?,?)}
```

donde los marcadores de parámetro deberán colocarse al final de los parámetros de entrada.

Almacenamiento de datos de varias tablas

Se puede especificar una consulta sobre varias tablas en un `QueryDataSet` y `JBuilder` puede almacenar los cambios a dicho `DataSet`. `SQLResolver` puede almacenar consultas SQL que tengan referencias a más de una tabla. El proceso de autodetección (discovery) de metadatos identificará a qué tabla pertenece cada columna, y propondrá un orden de almacenamiento de modificaciones entre las distintas tablas. Las propiedades que define el mecanismo de autodetección de metadatos son las siguientes:

- `Column` - `tableColumnName`
- `Column` - `schemaName`
- `Column` - `serverColumnName`
- `StorageDataSet` - `tableName`
- `StorageDataSet` - `resolveOrder`

La propiedad `tableName` del `StorageDataSet` no se define. El nombre de la tabla (`tableName`) se identifica columna por columna.

La propiedad `resolveOrder` es una matriz de cadena que especifica el orden de almacenamiento en las resoluciones multitabla. Las consultas INSERT y UPDATE utilizan el orden de esta matriz, las consultas DELETE utilizan el orden inverso. Si se elimina una tabla de la lista, sus columnas no se almacenarán.

Consideraciones sobre el tipo de vinculación entre tablas durante la consulta

Una consulta SQL multitabla habitualmente define un vínculo entre tablas en la cláusula WHERE de la consulta. En función de la naturaleza del vínculo y de la estructura de las tablas, este vínculo puede ser de cuatro tipos distintos (dadas una tabla principal T1 y una tabla vinculada T2):

- **1:1**

Hay exactamente un registro en T2 que se corresponde con un registro en T1 y viceversa. Una base de datos relacional puede tener este diseño para determinadas tablas, por motivos de claridad o para limitar el número de columnas por tabla.

- **1:M**

Pueden existir varios registros en T2 que se corresponden con un registro en T1, pero sólo un registro en T1 se corresponde con un registro en T2. Ejemplo: cada cliente puede tener varios pedidos.

- **M:1**

Hay exactamente un registro en T2 que se corresponde con un registro en T1, pero varios registros en T1 pueden corresponderse con un registro en T2. Ejemplo: cada pedido puede tener un identificador de producto, que se asocia con un nombre de producto en la tabla de productos. Es un ejemplo de búsqueda expresada directamente en SQL.

- **M:M**

El caso más general.

JBuilder utiliza un criterio simplificado para resolver varias tablas vinculadas: JBuilder sólo resuelve vinculaciones de tipo 1:1. No obstante, dado que resulta difícil detectar qué tipo de vinculación describe una consulta SQL determinada, JBuilder supone que toda consulta multitabla es del tipo 1:1. Si las tablas vinculadas no son del tipo 1:1, el almacenamiento de otros tipos se gestiona de la siguiente manera:

- **1:M**

Por lo general, resulta innecesario duplicar los campos maestros en cada registro-detalle durante la consulta. En su lugar, se crea un conjunto de datos de detalle independiente, que permite un almacenamiento adecuado de los cambios.

- **M:1**

Este tipo se gestionará habitualmente mediante el dispositivo de búsqueda. No obstante, si la búsqueda es sólo para visualizar (no se pueden editar estos campos), será posible gestionarlo como consulta multitabla. En, al menos, una columna se debe marcar la propiedad `rowId` de la tabla con la búsqueda como no resoluble.

- **M:M**

Esta relación entre tablas es poco frecuente y a veces se da como resultado de un error de especificación.

Referencias a tablas y columnas (alias) en una cadena de consulta

Una cadena de consulta puede incluir referencias a tablas y referencias a columnas o alias.

- Los alias de tabla no se utilizan habitualmente en consultas de una sola tabla, pero se emplean con frecuencia en consultas de varias tablas para

simplificar la cadena de consulta o para diferenciar tablas con un mismo nombre pero que pertenecen a diferentes usuarios.

```
SELECT A.a1, A.a2, B.a3 FROM Table_Called_A AS A, Table_Called_B AS B
```

- Las referencias a columnas se utilizan habitualmente para dar un nombre a una columna calculada, pero también pueden utilizarse para diferenciar columnas homónimas procedentes de tablas diferentes.

```
SELECT T1.NO AS NUMBER, T2.NO AS NR FROM T1, T2
```

- En JBuilder, si hay un alias de columna presente en la cadena de la consulta, pasa a ser el `columnName` de la `Column`. El nombre presente en la tabla original se asigna a la propiedad `tableColumnName`. El `QueryResolver` utiliza `tableColumnName` cuando genera las consultas de almacenamiento.
- Si hay un alias de tabla en la cadena de la consulta, se utiliza para identificar el `tableName` de una `Column`. El alias en sí no se visualiza a través de la API de JBuilder.

Control de la configuración de las propiedades de Column

Las propiedades `tableName`, `schemaName` y `serverColumnName` son configuradas por el `QueryProvider` para un `QueryDataSet` a menos que la propiedad `metaDataUpdate` no incluya `metaDataUpdate.TABLENAME`.

¿Qué pasa cuando una tabla no se puede actualizar?

Si no hay `rowId` en una tabla determinada de una consulta, no se guardarán las actualizaciones sobre esta tabla, al realizar la llamada `saveChanges()`.

Nota La posibilidad de actualización depende de otros aspectos, que se describen con más detalle en [“Consultas en bases de datos” en la página 5-2](#).

¿Cómo especificar que nunca debe actualizarse una tabla?

En las consultas de varias tablas, una de las tablas puede ser actualizable y la otra no. La propiedad `resolveOrder` de `StorageDataSet` es una matriz de cadena que especifica el orden de almacenamiento en un almacenamiento multitabla. Las consultas INSERT y UPDATE utilizan el orden de esta matriz, las consultas DELETE utilizan el orden inverso. Si se elimina una tabla de la lista, sus columnas no se almacenarán.

Si sólo utiliza una tabla, asigne a la propiedad `metaDataUpdate` el valor NONE y no defina ninguna de las propiedades de almacenamiento (`rowID`, `tableName`, etc.).

Utilización de conjuntos de datos con RMI (conjuntos de datos transportables)

Los conjuntos de datos transportables permiten la creación de un objeto Java (`DataSetData`) que contenga todos los datos de un *DataSet*. De igual manera, el objeto `DataSetData` puede utilizarse para proporcionar un *DataSet* con datos e información de las columnas.

El objeto `DataSetData` implementa la interfaz `java.io.Serializable`, y puede serializarse posteriormente mediante el objeto `writeObject` de `java.io.ObjectOutputStream` y leerse mediante el objeto `readObject` de `java.io.ObjectInputStream`. Este método convierte los datos en una matriz de bytes que se pasa mediante sockets u otro medio de transporte. Otra posibilidad es pasar el objeto a través de Java RMI, que efectuará la serialización directamente.

Además de guardar un conjunto completo de datos en el *DataSet*, se pueden guardar sólo los cambios en el conjunto de datos. Esta funcionalidad puede implementar un servidor de nivel intermedio que se comunique con un SGBD y un cliente con bajo consumo de recursos y capacidad para editar un *DataSet*.

Ejemplo: Utilización de conjuntos de datos transportables

Un ejemplo que ilustra la utilización de un *DataSet* transportable se da en un sistema de 3 niveles con una aplicación servidor Java que responde a solicitudes del cliente sobre datos de determinadas fuentes de datos. El servidor puede utilizar componentes `QueryDataSet` o `ProcedureDataSet` de `JBuilder` para proporcionar los datos al equipo servidor. Los datos se pueden extraer mediante `DataSetData.extractDataSet` y enviarse por una línea de comunicación al cliente. En el ordenador cliente, los datos pueden cargarse en un `TableDataSet` y editarse con controles `DataSet` de `JBuilder` o mediante llamadas a la API de Java de *DataSet*. La aplicación servidor puede extraer todos los datos de su *DataSet* con el objeto de estar preparada para servir a otras aplicaciones cliente.

Cuando el usuario de la aplicación cliente desea guardar los cambios, los datos pueden extraerse con `DataSetData.extractDataSetChanges` y enviarse al servidor. Antes de que el servidor cargue estos cambios, deberá obtener los tipos físicos de las columnas en el SGBD utilizando los metadatos del *DataSet*. A continuación, el *DataSet* recibe los cambios y se aplican los habituales almacenadores de `JBuilder` para almacenar los datos en el SGBD.

En caso de ocurrir errores de almacenamiento, podrían no ser detectados por las acciones de la interfaz de usuario si el almacenamiento se está desarrollando en un equipo servidor remoto. El almacenador puede

gestionar los errores mediante la creación de un `DataSet` de errores. Cada mensaje de error deberá etiquetarse con el valor `INTERNALROW` de la fila en la que ocurrió el error. `DataSetData` puede transportar estos errores a la aplicación cliente. Si el `DataSet` todavía está activo, la aplicación cliente puede vincular los errores con el `DataSet` y mostrar el texto del error para cada fila.

Utilización de métodos `DataSet` transportables

Los métodos estáticos `extractDataSet` y `extractDataSetChanges` rellenan los `DataSetData` con miembros de datos privados no transitorios, que especifican:

1 Información de metadatos que consta de:

- `columnCount`
- `rowCount`
- `columnNames`
- `dataTypes`
- `rowId`, `hidden`, `internalRow` (propiedades de columna)

Las propiedades se almacenan actualmente en los 3 bits más significativos de cada tipo de datos. Todo tipo de datos es un byte. La `columnCount` se almacena implícitamente como la longitud de la matriz `columnNames`.

2 Bits de estado para cada fila. Se almacena un `short` para cada fila.

3 Bits nulos (null) para cada elemento de datos. Se almacenan 2 bits para cada elemento de datos. Los valores que se pueden utilizar son:

- 0) Datos normales
- 1) Null asignado
- 2) Null sin asignar
- 3) Null sin cambios

El último valor se utiliza sólo para `extractDataSetChanges`. Los valores que permanecen sin cambios en la versión `UPDATED` se almacenan como null, ahorrando espacio para los binarios extensos, etc.

4 Los mismos datos, organizados en una matriz de datos por columna. Si la columna de datos es de tipo `Variant.INTEGER`, se utiliza una matriz `int` para los valores de esa columna.

5 En los `extractDataSetChanges`, se añade una columna especial, `INTERNALROW`, a la sección de datos. Esta columna de datos contiene valores largos que designan el `internalRow` del `DataSet` del cual se extrajeron los datos. Esta columna de datos debe utilizarse para informar sobre errores, llegado el caso de que los cambios no pudieran aplicarse en el SGBD de destino.

El método `loadDataSet` cargará los datos en un `DataSet`. Se añadirán todas las columnas que hagan falta en el `DataSet`. Observe que las propiedades y tipos físicos como `sqlType`, `precision` y `scale` no se incluyen en el objeto `DataSetData`. Estas propiedades deben buscarse directamente en el SGBD. No obstante, estas propiedades no son necesarias para las tareas de edición. La columna especial `INTERNALROW` se visualiza como las demás columnas del conjunto de datos.

Personalización de la lógica del almacenador por defecto

JBuilder simplifica la escritura de almacenadores personalizados para sus datos cuando se accede a datos de una fuente de datos personalizada, como EJB, servidores de aplicaciones, SAP, BAAN, IMS, OS/390, CICS, VSAM, DB2 y otras.

La recuperación y actualización de datos de una fuente de datos, como un servidor Oracle o Sybase, se limita a dos interfaces clave: proveedores y almacenadores. Los *Proveedores* transmiten datos de una fuente de datos a un objeto `StorageDataSet`. Los *Almacenadores* guardan los cambios en la fuente de datos. Al separar claramente la recuperación y actualización de datos con dos interfaces, resulta sencillo crear nuevos componentes proveedores/almacenadores para nuevas fuentes de datos. JBuilder proporciona implementaciones para controladores JDBC estándar que proporcionan acceso a conocidas bases de datos como Oracle, Sybase, Informix, InterBase, DB2, MS SQL Server, Paradox, dBASE, FoxPro, Access y otras. Entre otras, destacan las siguientes:

- `OracleProcedureProvider`
- `ProcedureProvider`
- `ProcedureResolver`
- `QueryProvider`
- `QueryResolver`

En el directorio `/samples/DataExpress/CustomProviderResolver` de la instalación de JBuilder se encuentra un proyecto de ejemplo con un proveedor y un almacenador personalizados. El archivo de ejemplo `TestApp.java` es una aplicación con un marco que contiene una `JdbTable` y una `JdbNavToolBar`. Ambos componentes visuales están conectados a un componente `TableDataSet`, en el que los datos se suministran desde un proveedor (Provider) personalizado (definido en el archivo `ProviderBean.java`), y se guardan con un almacenador (Resolver) (definido en el archivo `ResolverBean.java`). Esta aplicación de ejemplo lee y guarda las modificaciones en el archivo de texto `data.txt`, que es un archivo de formato simple sin delimitaciones. La estructura del archivo `data.txt` se describe en el archivo de interfaz `DataLayout.java`.

Puede encontrar un ejemplo que describe la forma de escribir un `ProcedureResolver` personalizado en [“Guardar cambios con un `ProcedureResolver`” en la página 8-9](#).

Cómo funciona el almacenador por defecto

Si no se instancia de forma específica un componente `QueryResolver` al almacenar los cambios en los datos en la fuente de datos, la lógica del almacenador incorporado crea un componente por defecto `QueryResolver`. Este tema analiza la utilización de `QueryResolver` para personalizar el proceso de almacenamiento.

El `QueryResolver` es un componente de la JBCL que implementa la interfaz `SQLResolver`. Es esta interfaz `SQLResolver` la que utiliza `ResolutionManager` durante el proceso de almacenamiento de cambios en la bases de datos. Como su propio nombre indica, la clase `ResolutionManager` gestiona la fase de almacenamiento.

Todos los `StorageDataSet` tienen una propiedad `resolver`. Si no se ha establecido esta propiedad cuando se llama al método `Database.saveChanges()`, éste crea una interfaz `QueryResolver` por defecto e intenta guardar los cambios del `DataSet`.

Incorporación de un componente `QueryResolver`

Para añadir un componente `QueryResolver` a una aplicación mediante las herramientas de diseño visual de JBuilder, realice los pasos siguientes:

- 1 Abra el proyecto existente al que desee añadir lógica de almacenamiento personalizada.

El proyecto debe incluir un objeto `Database` y otro objeto `QueryDataSet`. Si necesita instrucciones, consulte [“Consultas en bases de datos” en la página 5-2](#).

- 2 Haga doble clic sobre el archivo `Marco`, en el panel de contenido, y seleccione la pestaña Diseño para abrir el diseñador de interfaces de usuario.
- 3 Haga clic en el componente `QueryResolver` (aquí mostrado) de la pestaña `DataExpress` de la paleta de componentes.
- 4 Haga clic en el árbol de componentes o el diseñador de la interfaz de usuario, en cualquier parte de ellos, para añadirlo a la aplicación.

El diseñador de interfaces genera código fuente que crea un objeto `QueryResolver` por defecto.

- 5 Conecte el `QueryResolver` al `DataSet` correspondiente.

Para ello, utilice el Inspector para asignar valores a la propiedad `resolver` de `StorageDataSet`, por ejemplo `queryDataSet1`, el objeto `QueryResolver`, adecuado, que por defecto es `queryResolver1`.

Se puede conectar el mismo `QueryResolver` a más de un `DataSet` siempre que los distintos objetos `DataSet` compartan la misma gestión de sucesos. Si cada `DataSet` necesita una gestión de sucesos personalizada, se debe crear un `QueryResolver` independiente para cada `StorageDataSet`.

Intercepción de sucesos del almacenador

El proceso de almacenamiento se controla interceptando los sucesos `Resolver`. Cuando se selecciona un objeto `QueryResolver` en el panel de contenido, la pestaña Sucesos del Inspector muestra sus sucesos. Los sucesos que se pueden controlar (definidos en la interfaz `ResolverListener`) se pueden agrupar en las tres categorías siguientes:

- Notificación de la acción que se va a ejecutar: Todos los errores se considerarán como excepciones normales y no como sucesos de error:
 - `deletingRow()`
 - `insertingRow()`
 - `updatingRow()`
- Notificación de la ejecución de una acción:
 - `deletedRow()`
 - `insertedRow()`
 - `updatedRow()`
- Errores condicionales que se han producido: Éstos son errores internos, no errores del servidor:
 - `deleteError()`
 - `insertError()`
 - `updateError()`

Cuando el gestor de almacenamiento se prepara para ejecutar una acción de eliminación, inserción o actualización, se genera la correspondiente notificación de suceso del primer conjunto de sucesos (`deletingRow`, `insertingRow` y `updatingRow`). Entre otros parámetros pasados con la notificación a estos sucesos, se pasa un objeto `ResolverResponse`. Es responsabilidad del manejador de sucesos (también denominado `monitor de sucesos`) determinar si la acción es o no apropiada y devolver una de las repuestas (`ResolverResponse`) siguientes:

- `resolve()` indica al gestor de almacenamiento que continúe almacenando esta fila.
- `skip()` indica al gestor de almacenamiento que omita esta fila y continúe con el resto.

- `abort()` indica al gestor de almacenamiento que detenga el almacenamiento.

Si la respuesta del suceso es `resolve()` (la respuesta por defecto), se generará un suceso del segundo conjunto (`deletedRow`, `insertedRow` o `updatedRow`) según corresponda. No se espera respuesta de estos sucesos. Sólo existen para comunicar a la aplicación qué acción se ha ejecutado.

Si la respuesta del suceso es `skip()`, no se resuelve la fila actual y el proceso de almacenamiento continúa en la siguiente fila de datos.

Si el suceso provoca que se interrumpa el proceso del almacenamiento, se llamará al método `inserting`, el cual a su vez llama al método `response.abort()`. No se genera ningún suceso de error, porque los sucesos de error están programados para responder a errores internos. Sin embargo, para cancelar el proceso de almacenamiento de modificaciones se lanza una excepción genérica de tipo `ResolutionException`.

Si durante el proceso de almacenamiento se produce un error, por ejemplo, el servidor no permite eliminar una fila, se genera el suceso de error apropiado (`deleteError`, `insertError` o `updateError`). Estos sucesos se pasan a los siguientes elementos:

- El `DataSet` original que participó en el almacenamiento.
- Un `DataSet` temporal que ha sido filtrado sólo para visualizar las filas afectadas.
- La `Exception` que se ha producido.
- Un objeto `ErrorResponse`.

Las acciones siguientes son responsabilidad del manejador de sucesos:

- Examinar la `Exception`
- Determinar cómo continuar.
- Comunicar esta decisión al gestor de almacenamiento. Esta decisión se comunica mediante una de las siguientes respuestas `ErrorResponse`:
 - `abort()` indica al gestor de almacenamiento que suspenda todas las resoluciones.
 - `retry()` indica al gestor de almacenamiento que intente de nuevo la última operación.
 - `ignore()` indica al gestor de almacenamiento que continúe y haga caso omiso del error.

Si el manejador de sucesos lanza una `DataSetException`, se la considera como un `ResolverResponse.abort()`. Además, se dispara el suceso de error anteriormente descrito, pasando por la `Exception` del usuario.

Utilización de sucesos de almacenador

Puede encontrar un ejemplo de sucesos de almacenador en `ResolverEvents.jpx` y los archivos asociados, en el subdirectorio `/samples/DataExpress/ResolverEvents` del directorio de instalación de JBuilder. En la aplicación `ResolverEvents`:

- Se asocia una tabla a la tabla `Customer` de la base de datos de ejemplo `JDataStore`.
- El botón Guardar cambios crea un objeto `QueryResolver` personalizado que toma el control del proceso de almacenamiento.

En la aplicación en ejecución, se observa el comportamiento siguiente:

- La eliminación de filas no está permitida. Se impide incondicionalmente todo intento de eliminar una fila de datos. Sirve para demostrar la utilización del suceso `deletingRow`.
- Sólo se permite insertar filas cuando el cliente es de los Estados Unidos. El proceso se suspende cuando el cliente no es de los Estados Unidos. Sirve para demostrar la utilización del suceso `insertingRow` y de una `ResolverResponse` de `abort()`.
- Las actualizaciones de filas se realizan añadiendo los valores antiguos y nuevos del nombre del cliente a `ListControl`. Esto demuestra cómo acceder a la información nueva y a la anterior durante el proceso de almacenamiento.

Cómo crear un almacenador de datos personalizado

En este apartado se muestra cómo crear almacenadores de datos personalizados y cómo utilizarlos como almacenadores para un `TableDataSet` y cualquier `DataSet` derivado de `TableDataSet`. El principal método que debe implementarse es `resolveData()`. Este método recoge las modificaciones realizadas en un `StorageDataSet` y vuelve a almacenar en la fuente de los datos los cambios efectuados.

Para volver a almacenar en la fuente los cambios realizados:

- 1 Asegúrese de que esté bloqueado el `StorageDataSet`, para impedir que puedan producirse cambios en el proveedor durante el proceso de almacenamiento de modificaciones. Para ello deberá llamar a los siguientes métodos:

- `ProviderHelp.startResolution(dataSet, true);`
- `ProviderHelp.endResolution(dataSet);`

Importante

Introduzca las siguientes líneas entre las llamadas a estos dos métodos.

- 2 Localice los cambios en los datos creando un `DataSetView` para cada una de las filas insertadas, borradas o actualizadas. Puede hacerlo utilizando las siguientes llamadas a métodos:

- `StorageDataSet.getInsertedRows(DataSetView);`
- `StorageDataSet.getDeletedRows(DataSetView);`
- `StorageDataSet.getUpdatedRows(DataSetView);`

Es importante tener en cuenta que:

- Las filas insertadas pueden contener filas borradas (que no sería necesario almacenar).
 - Las filas borradas pueden contener filas insertadas (que no deberían almacenarse).
 - Las filas actualizadas pueden contener filas borradas e insertadas (que no deberían tratarse como actualizaciones).
- 3 Cierre todas los componentes `DataSetView` una vez almacenados los datos, pues, de lo contrario, se producirá una excepción durante el proceso de almacenamiento. Si no están cerrados los componentes `DataSetView`, el `StorageDataSet` mantendrá referencias a ellas y, por tanto, nunca liberarán su memoria.

Tratamiento de errores del almacenador

Hay muchas formas de corregir los errores, pero hay que indicar al `DataSet` que modifique el estado de las filas que hayan cambiado. Para ello:

- 1 Modifique cada una de las filas marcada con el estado `RowStatus.PENDING_RESOLVED`.

El código que se emplea para marcar la fila actual de esta manera es el siguiente:

```
DataSet.markPendingStatus(true);
```

Llame a este método por cada una de las filas insertadas, borradas o actualizadas que se estén almacenando.

- 2 Para poner a cero el bit `RowStatus.PENDING_RESOLVED`, llame a uno o varios de los siguientes métodos:

El método que deberá utilizar dependerá del mecanismo que emplee para resolver los errores:

- `markPendingStatus(false);`

El método `markPendingStatus` pone a cero el bit de la fila actual.

- `resetPendingStatus(boolean resolved);`

Este método `resetPendingStatus` pone a cero los bits de todas las filas del `DataSet`.

- `resetPendingStatus(long internalRow, boolean resolved);`

Este método `resetPendingStatus` pone a cero el bit de la fila correspondiente al identificador `internalRow` especificado.

- 3 Ponga a cero el parámetro `resolved` utilizando uno de los métodos `resetPendingStatus`, asignando el valor **true** para las filas cuyos cambios se hayan reflejado realmente en la fuente de datos.

Cuando se pone a cero el bit `PENDING_RESOLVED`, las filas conservan el estado de las modificaciones registradas. Las filas deberán reiniciarse y almacenarse, para que:

- El estado de las filas insertadas (`INSERTED`) y actualizadas (`UPDATED`) pase a ser `LOADED` (cargadas).
- Las filas eliminadas (`DELETED`) desaparezcan del `DataSet`.

Las modificaciones de filas que no hayan sido efectuadas pondrán a cero el bit `PENDING_RESOLVED`, pero los cambios seguirán registrándose en el `DataSet`.

Algunos almacenadores optarán por abandonar todas las modificaciones si se detecta algún error. De hecho, ese es el comportamiento por defecto del `QueryDataSet`. Otros almacenadores pueden optar por confirmar ciertas modificaciones y conservar las modificaciones fallidas, para poder mostrar mensajes de error.

Almacenar relaciones maestro-detalle

El almacenador de modificaciones en relaciones maestro-detalle plantea ciertos problemas que es preciso tener en cuenta. Si la fuente de los datos tiene reglas de integridad referencial, quizás sea necesario almacenar las modificaciones en los componentes `DataSet` en un determinado orden.

Cuando se utiliza JDBC, JBuilder proporciona la clase `SQLResolutionManager`. Esta clase se encarga de garantizar que el conjunto de datos maestro almacene sus filas insertadas antes de permitir que el conjunto de datos de detalle almacene su fila insertada, a la vez que garantiza que los conjuntos de datos de detalle almacenen sus filas borradas antes de que se almacenen las filas borradas del conjunto de datos maestro. Si desea más información sobre la resolución de relaciones maestro-detalle, consulte [“Almacenamiento de los cambios en una relación maestro-detalle” en la página 9-11](#).

Establecimiento de una relación maestro-detalle

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Las bases de datos bien diseñadas incluyen tablas múltiples. El objetivo de una tabla es almacenar toda la información necesaria de forma eficaz y accesible. Por tanto, es recomendable dividir una base de datos en tablas capaces de identificar de manera independiente entidades (personas, lugares y cosas) y actividades (sucesos, transacciones, etc.) importantes para la aplicación. Para definir tablas es necesario comprender las relaciones entre ellas. La creación de varias tablas pequeñas vinculadas entre sí reduce la cantidad de datos redundantes, con lo que disminuye la posibilidad de cometer errores, a la vez que se facilita la actualización de la información.

En JBuilder, con un `MasterLinkDescriptor` es posible unir o vincular dos o más conjuntos de datos, siempre que compartan al menos un campo. Una relación maestro-detalle suele ser una relación entre conjuntos de datos de uno a muchos. Por ejemplo, supongamos que se dispone de un conjunto de datos de clientes y otro de pedidos de los mismos clientes y el número del cliente es un campo común a ambos. Se puede crear una relación maestro-detalle que permita desplazarse por el conjunto de datos de clientes y visualizar, en el conjunto de datos detalle, sólo los registros de los pedidos realizados por el cliente del registro actual.

Un conjunto de datos maestro se puede vincular a varios conjuntos de datos, mediante el mismo campo o mediante campos diferentes. También se puede crear una relación maestro-detalle que desencadene una relación de uno a muchos y de muchos a muchos. Aunque las relaciones de muchos a uno o uno a uno se pueden gestionar en el contexto maestro-detalle, es mejor gestionarlas mediante la utilización de campos de consulta para ver todos los datos como parte de un conjunto de datos. Si desea información sobre el almacenamiento de los cambios efectuados en

los datos de varios conjuntos de datos, consulte [“Almacenamiento de datos de varias tablas” en la página 8-12](#).

No es necesario que los conjuntos de datos maestro y detalle sean del mismo tipo. Por ejemplo, se puede utilizar un `QueryDataSet` como conjunto de datos maestro y un `TableDataSet` como el conjunto de datos detalle.

Tanto `QueryDataSet` como `TableDataSet` y `DataSetView` pueden utilizarse como conjuntos de datos maestro o detalle.

Estos son los temas tratados:

- [Definición de una relación maestro-detalle.](#)
- [Captura de detalles.](#)
- [Edición de datos en conjuntos de datos maestro-detalle.](#)
- [Pasos en la creación de una relación maestro-detalle.](#)
- [Almacenamiento de los cambios en una relación maestro-detalle.](#)

Definición de una relación maestro-detalle

Para definir una relación maestro-detalle es necesario vincular columnas con el mismo tipo de datos. Por ejemplo, si los datos del conjunto de datos maestro son del tipo INT, los datos del conjunto de datos detalle también deben ser INT. Si los datos del conjunto de datos detalle fueran del tipo LONG, no aparecería ninguna coincidencia. Los nombres de las columnas deben ser diferentes. Se pueden vincular columnas que no tengan índices en el servidor.

No hay limitaciones en la forma de ordenar la información del conjunto de datos maestro. La vinculación entre conjuntos de datos maestros y detalle se realiza de la misma manera que el mantenimiento de vistas ordenadas, con un índice mantenido. Esto supone que un conjunto de datos detalle siempre se ordena con las columnas detalle de vínculo como las columnas de orden más a la izquierda. Los criterios de clasificación adicionales deben ser compatibles con las columnas detalle de vínculo. Para que se produzca esta compatibilidad, el descriptor de clasificación no puede incluir ninguna columna detalle de vínculo o, si lo hace, debe especificarse en el mismo orden tanto en la columna detalle de vínculo como en el descriptor de clasificación. Deben especificarse todas las columnas detalle de vínculo incluidas en el descriptor de clasificación.

Los datos se pueden filtrar tanto en el conjunto de datos maestro como en el conjunto de datos detalle o en ambos. Una relación maestro-detalle por sí sola es similar a un filtro aplicado al conjunto de datos detalle, sin embargo, los filtros pueden utilizarse sobre uno de los conjuntos, además de la relación maestro-detalle.

Para crear una relación maestro-detalle, se puede utilizar una sentencia SQL JOIN en vez de un `MasterLinkDescriptor`. Una sentencia SQL JOIN es un operador relacional que genera una única tabla a partir de dos,

mediante la comparación de valores de columna determinados (columnas de enlace) en cada uno de los conjuntos de datos. El resultado es un conjunto de datos único con las filas formadas por la concatenación de las filas de los dos conjuntos de datos siempre que los valores de las columnas de enlace coincidan. Para actualizar consultas JOIN con JBuilder, consulte [“Almacenamiento de datos de varias tablas” en la página 8-12](#).

Creación de una aplicación con una relación maestro-detalle

Este ejemplo muestra cómo crear una relación maestro-detalle mediante los archivos de ejemplo de JBuilder. La estructura básica de la aplicación implica la construcción de dos consultas, una que seleccione todos los países de la tabla COUNTRY del archivo employee de ejemplo y otra que seleccione todos los empleados. Este tutorial está disponible como proyecto completo en el subdirectorio `/samples/DataExpress/MasterDetail` del directorio de instalación de JBuilder.

El conjunto de datos COUNTRY es el maestro, con la columna COUNTRY como campo de vínculo con EMPLOYEE, el conjunto de datos detalle. Ambos conjuntos de datos están asociados a `JdbTables` y cuando se desplaza por la tabla COUNTRY, la tabla EMPLOYEE visualiza todos los empleados que viven en el país del registro actual.

Para crear esta aplicación:

- 1 Cierre todos los proyectos abiertos (Archivo | Cerrar).
- 2 Seleccione Archivo | Nuevo y haga doble clic en el icono Aplicación para crear una nueva aplicación.
Acepte los valores por defecto.
- 3 Seleccione la pestaña Diseño en el panel de contenido.
- 4 Seleccione un componente `DataBase` de la pestaña DataExpress de la paleta de componentes y haga clic en el árbol de componentes o en el diseñador de interfaces de usuario para añadir el componente a la aplicación.
- 5 Abra la propiedad `connection` del componente `Database` en el Inspector y asigne valores a las propiedades como sigue, si el sistema está configurado para el uso de la muestra de `JDataStore` de la forma descrita en [“Configuración de JDataStore” en la página 4-7](#).

Nombre de la propiedad	Valor
Controlador	<code>com.borland.datastore.jdbc.DataStoreDriver</code>
URL	Busque en su sistema el archivo <code><jbuilder>/samples/JDataStore/datastores/employee.jds</code>

Nombre de la propiedad	Valor
Nombre de usuario	Introduzca su nombre
Contraseña	No es obligatoria

El cuadro de diálogo `Connection` contiene un botón Probar conexión. Púlselo para comprobar que las propiedades de conexión tienen los valores correctos. El resultado del intento de conexión se muestra junto al botón. Cuando la conexión sea satisfactoria, pulse Aceptar.

Nota La biblioteca de servidor de `JDataStore` se añade al proyecto cuando se establece una conexión con una base de datos `JDataStore`.

El código que genera el diseñador para este paso se puede ver en la sección `ConnectionDescriptor` de la pestaña Fuente. Haga clic en la pestaña Diseño para continuar.

- Para añadir un componente `QueryDataSet` a la aplicación, selecciónelo en la pestaña Data Express y haga clic en el árbol de componentes.

Este componente configura la consulta para el conjunto de datos maestro. Desde el Inspector, seleccione la propiedad `query` del componente `QueryDataSet` y asigne valores a la propiedad de la siguiente forma:

Nombre de la propiedad	Valor
Database	<code>database1</code>
Sentencia SQL	<code>SELECT * FROM COUNTRY</code>

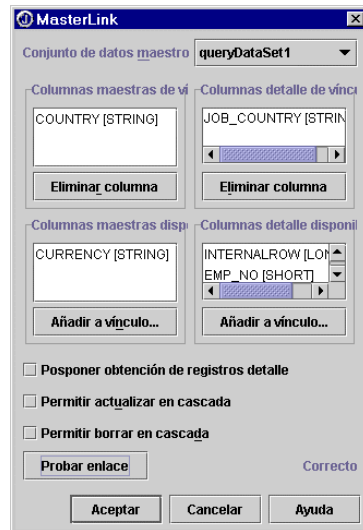
- Pulse Probar consulta para comprobar si la consulta se puede ejecutar; a continuación, cuando el área de estado indique `Correcto`, pulse Aceptar para cerrar el cuadro de diálogo.
- Añada otro componente `QueryDataSet` a la aplicación, seleccione su propiedad `query` en el Inspector, pulse el botón de puntos suspensivos (...) para abrir el cuadro de diálogo Consulta y configure las siguientes propiedades:

Nombre de la propiedad	Valor
Database	<code>database1</code>
Sentencia SQL	<code>SELECT * FROM EMPLOYEE</code>

Con ello se configura la consulta para el conjunto de datos detalle.

- 9 Pulse Probar consulta para comprobar si la consulta se puede ejecutar; a continuación, cuando el área de estado indique **Correcto**, pulse **Aceptar** para cerrar el cuadro de diálogo.
- 10 Seleccione la propiedad `masterLink` para el conjunto de datos de detalle (`queryDataSet2`) en el Inspector, pulse el botón de puntos suspensivos (...) para abrir el cuadro de diálogo MasterLink y configure las propiedades del siguiente modo:
 - a La propiedad `Master Dataset` proporciona un menú desplegable de conjuntos de datos disponibles. Seleccione el conjunto de datos que contenga los registros maestros del conjunto de datos detalle actual, en este caso `queryDataSet1`.
 - b Los campos de vínculo describen los campos que se utilizan para determinar la coincidencia de datos entre los componentes de los conjuntos de datos maestro y detalle. Para seleccionar una columna del conjunto de datos maestro para vincularla con una columna del conjunto de datos detalle, seleccione su nombre, en este caso `COUNTRY` (un campo de cadena), en la lista Columnas maestras disponibles y, a continuación, seleccione el botón **Añadir a vínculos maestros**. Esta columna se visualiza en el cuadro Columnas maestras de vínculo.
 - c Para seleccionar la columna del conjunto de datos detalle para vincularla con una columna del conjunto de datos maestro, seleccione su nombre, en este caso `JOB_COUNTRY` (un campo de cadena), en la lista Columnas detalle disponibles y, a continuación, seleccione el botón **Añadir a vínculos detalle**. Esta columna se visualiza en el cuadro Columnas detalle de vínculo.
 - d La opción **Posponer obtención de registros de datos detalle** determina si los registros de los conjuntos de datos detalle han de capturarse todos a la vez o sólo para un maestro concreto cuando sea necesario (accede al registro maestro). Desactive esta casilla lo que asignará a `fetchAsNeeded` el valor `false`. Para obtener más información sobre las capturas, consulte [“Captura de detalles” en la página 9-8](#).
 - e Haga clic en el botón **Probar enlace**.

El cuadro de diálogo debería presentar el siguiente aspecto cuando la prueba resulte correcta.



f Haga clic en Aceptar para cerrar el cuadro de diálogo MasterLink.

- 11 Añada a la aplicación un componente `DBDisposeMonitor` desde la pestaña Más dbSwing.

El componente `DBDisposeMonitor` cierra el `JDataStore` cuando se cierre la ventana.

- 12 Asigne a la propiedad `dataAwareComponentContainer` de `DBDisposeMonitor` el valor `this`.

Para crear una interfaz de usuario para esta aplicación:

- 1 En el árbol de componentes, seleccione `contentPane(BorderLayout)` y asigne a su propiedad `layout` el valor `null`.
- 2 Agregue un componente `JdbNavToolBar` desde la ficha dbSwing y colóquelo en el área de la parte superior del panel del Diseñador de interfaces de usuario.

`JdbNavToolBar` se autovincula automáticamente al `DataSet` que tenga el foco.

- 3 Añada un componente `JdbStatusLabel` y colóquelo en la zona por debajo del panel, en el diseñador de interfaces.

`JdbStatusLabel` se autovincula automáticamente al `DataSet` que tenga el foco.

- 4 Para añadir un componente `TableScrollPane` a la aplicación justo debajo de `jdbNavToolBar1`, selecciónelo en la pestaña dbSwing, haga clic y

arrastre el contorno del panel a la parte superior derecha del panel de diseño.

El desplazamiento no está disponible por defecto en los componentes Swing o extensiones dbSwing; para obtenerlo, se añade un componente Swing o dbSwing de desplazamiento a JScrollPane o a TableScrollPane. TableScrollPane proporciona capacidades especiales a JdbTable sobre JScrollPane. Para obtener más información consulte la documentación de dbSwing.

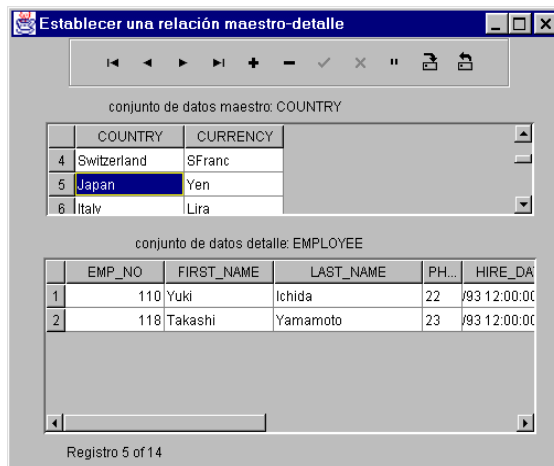
- 5 Coloque el componente JdbTable en el centro de tableScrollPane1 del diseñador de interfaces de usuario y asigne a su propiedad dataSet el valor queryDataSet1.
- 6 Añada otro componente TableScrollPane a la parte inferior del panel de la ventana de diseño.

Se convierte en el objeto tableScrollPane2.

- 7 Coloque un componente JdbTable en tableScrollPane2 y asigne a su propiedad dataSet el valor queryDataSet2.
- 8 Compile y ejecute la aplicación seleccionando Ejecutar | Ejecutar proyecto.

Ahora se puede desplazar por los registros (COUNTRY) maestro y observar cómo cambian los registros (EMPLOYEE) detalle de manera que reflejan sólo los empleados del país actual.

La aplicación en ejecución presentará este aspecto:



Captura de detalles

En una relación maestro-detalle, los valores de los campos maestros determinan los registros detalle que se visualizan. Los registros del conjunto de datos detalle pueden capturarse de una vez o cuando un maestro determinado lo necesite (al visitar el registro maestro).

Tenga cuidado cuando utilice las opciones `cascadeUpdates` y `cascadeDelete` en las relaciones maestro-detalle. Al utilizarlas, puede que se actualice o borre una fila de un conjunto de datos detalle, pero no las demás. Por ejemplo, un manejador del suceso `deleting()` del `editListener` podría permitir que se borren determinadas filas del detalle, pero no otras. En el caso de las actualizaciones en cascada, si algunas de las filas de un conjunto de detalle pueden actualizarse y otras no, podría ocurrir que al final los datos detalle quedasen huérfanos. Para obtener más información sobre las opciones `cascadeUpdates` y `cascadeDelete`, consulte el tema `MasterLinkDescriptor` en la *DataExpress Library Reference*.

Captura de todos los detalles de una vez

Si el parámetro `fetchAsNeeded` tiene el valor `false` (o la opción Posponer obtención de registros detalle está desactivada en el cuadro de diálogo `masterLinkDescriptor`), todos los datos detalle se capturan de una vez. Utilice esta configuración cuando el conjunto de datos detalle sea pequeño. Con ella se puede ver una instantánea, la vista más coherente de los datos. Cuando se llama al método `refresh()`, se actualizan a la vez todos los conjuntos de detalle.

Por ejemplo, al principio el conjunto de datos se alimenta con todos los datos del conjunto de datos detalle. Cuando la opción `fetchAsNeeded` se configura como `false`, se puede instanciar un componente `DataSetView` mediante el cual ver el conjunto de datos detalle y que aparecen todos sus registros, teniendo en cuenta que son filtrados por la vista de la información del vínculo proporcionado desde el conjunto de datos maestro.

Obtención de registros detalle a petición

Si el parámetro `fetchAsNeeded` tiene el valor `true` (o la opción Posponer obtención de registros detalle está seleccionada en el cuadro de diálogo `masterLinkDescriptor`), todos los registros detalle se capturan y almacenan en el conjunto de datos detalle. Este tipo de relación maestro-detalle es, en realidad, una consulta parametrizada donde los valores de los campos maestros determinan qué registros detalle se visualizan. La utilización de esta opción es especialmente recomendable para mejorar el rendimiento cuando la tabla de la base de datos remota sea muy extensa (no residen en

memoria todos los datos, sino que se cargan cuando es necesario). También debe utilizarse esta opción cuando no se esté interesado en la mayoría de los datos detalle. Los datos vistos están más actualizados pero no son tan coherentes como una instantánea de los datos con el parámetro `fetchAsNeeded` como `false`. Se captura un conjunto de registros detalle de golpe, se sitúa en memoria; a continuación, se captura otro conjunto de registros detalle y se sitúa en memoria. Mientras se efectúa este proceso, el primer conjunto de registros detalle puede haber cambiado en la tabla de la base de datos remota; sin embargo, estos cambios no se advertirán hasta que se actualicen los detalles. Cuando se llama al método `refresh()` sólo se actualizan los conjuntos detalle actuales.

Por ejemplo, al principio el conjunto de datos detalle está vacío. Cuando se accede al registro maestro, por ejemplo Hernández, se capturan todos los registros detalle de Hernández. Cuando se accede a otro registro maestro, p. ej. García, se capturan todos los registros detalle de García y se añaden al conjunto de datos detalle. Si se instancia un componente `DataSetView` para ver el conjunto de datos detalle, aparecen todos los registros de Hernández y de García, pero no aparecen registros de otros nombres.

Cuando la propiedad `fetchAsNeeded` tiene el valor `true`, debe existir una cláusula `WHERE` que defina la relación de las columnas de detalle en el `QueryDataSet` actual con un parámetro que representa el valor de una columna en el conjunto de datos maestro. Si la consulta parametrizada tiene marcadores de parámetro con nombre, el nombre debe coincidir con uno existente en el conjunto de datos. Si se utilizan marcadores de parámetros “?” de JDBC, las columnas de vínculo de detalle se asocian a los marcadores de parámetros de izquierda a derecha, como se define en la propiedad `masterLink`. Cuando el maestro se desplaza a una fila por primera vez, la asociación de los valores de parámetro se realiza de forma implícita. La consulta se vuelve a ejecutar para capturar cada grupo de detalle nuevo. Si no existe cláusula `WHERE`, JBuilder lanza una `DataSetException.NO_WHERE_CLAUSE`. Cuando la captura se gestiona de este modo, los grupos de detalle se capturan en transacciones independientes, si no existen transacciones explícitas activas. Si desea más información sobre las relaciones maestro-detalle dentro de consultas parametrizadas, consulte [“Consultas parametrizadas en relaciones maestro-detalle” en la página 5-22](#).

Cuando el conjunto de datos maestro tiene asociados dos o más conjuntos de datos detalle, y la propiedad `fetchAsNeeded` de todos ellos es `true`, los detalles recuerdan qué grupos de detalle han intentado capturar mediante una consulta o un procedimiento almacenado parametrizado por las columnas maestras de vínculo activas. Esta memoria puede vaciarse llamando al método `StorageDataSet.empty()`. No hay memoria para las propiedades `masterLink` que asignen a `fetchAsNeeded` el valor `true`.

Si el conjunto de datos detalle es un `TableDataSet` se hace caso omiso al parámetro `fetchAsNeeded` y todos los datos se capturan de una vez.

Edición de datos en conjuntos de datos maestro-detalle

Los valores de una columna maestra de vínculo (la columna vinculada al conjunto de datos detalle) no se pueden eliminar ni cambiar si el registro maestro tiene registros detalle asociados.

Por defecto, las columnas detalle de vínculo no aparecen en un componente de interfaz de usuario `JdbTable`, ya que estas columnas duplican los valores de las columnas de vínculo maestro, que aparecen. Cuando se inserta una nueva fila en un conjunto de datos detalle, `JBuilder` inserta los valores coincidentes de los campos que no se muestran.

Pasos en la creación de una relación maestro-detalle

Para crear un vínculo maestro-detalle entre dos componentes de un conjunto de datos, uno debe representar al conjunto de datos maestro y el otro al conjunto de datos detalle:

- 1 Cree o abra una aplicación con al menos dos componentes de conjunto de datos, uno de los cuales representa el conjunto de datos maestro y el otro, el conjunto de datos de detalle.

Puede utilizar la aplicación maestro-detalle de ejemplo que se incluye en el proyecto `MasterDetail.jpx` ubicado en el directorio `<jbuilder>/samples/DataExpress/MasterDetail`.

- 2 Seleccione el archivo Marco en el panel de contenido. Active el diseñador de interfaces de usuario, seleccionando la pestaña Diseño.
- 3 Seleccione el conjunto de datos detalle, en el árbol de componentes, y su propiedad `masterLink` en la ficha Propiedades del Inspector. Especifique en el editor de propiedades personalizadas `masterLink` las siguientes propiedades del conjunto de datos detalle:
 - La propiedad `masterDataSet` proporciona un menú de selección de conjuntos de datos disponibles. Seleccione el conjunto de datos que contenga los registros maestro del conjunto de datos detalle actual.
 - Las columnas de vínculo describen qué columnas deben utilizarse para determinar la coincidencia de datos entre los componentes de los conjuntos de datos maestro y detalle. Para seleccionar la columna del conjunto de datos maestro que se va a vincular con el conjunto de datos detalle, haga doble clic en el nombre de la columna de la lista Columnas maestras disponibles. Esta columna se visualiza ahora en la propiedad Columnas maestras de vínculo.

- Para seleccionar la columna del conjunto de datos detalle que se va a vincular con el conjunto de datos maestro, haga doble clic en el nombre de la columna de la lista `Columnas detalle disponibles`. Se muestra el tipo de datos de todas las columnas. Si selecciona una columna detalle cuyo tipo no coincide con el de la columna maestra, no ocurre nada, ya que las columnas de vínculo deben ser del mismo tipo. Si se selecciona adecuadamente, la columna aparece en la propiedad `Columnas de vínculo de detalle`.
 - Para vincular los dos conjuntos de datos en más de una columna, repita los dos pasos anteriores hasta vincular todas las columnas.
 - Para no capturar los registros detalle hasta que no sean necesarios, active la casilla `Posponer obtención de registros detalle`. Para obtener más información sobre esta opción, consulte [“Captura de detalles” en la página 9-8](#).
 - Haga clic en `Probar enlace` para comprobar que la conexión entre los conjuntos de datos es correcta. En el área de estado se indicará `En curso`, `Correcto` o `Fallo`.
 - Para completar la especificación, pulse `Aceptar`.
- 4 Incorpore dos controles visuales (como `JdbTables`) para poder ver y modificar datos. Asigne valores a la propiedad `dataSet` de uno como el conjunto de datos maestro y a la propiedad `dataSet` del otro como el conjunto de datos detalle.
- 5 Compile y ejecute la aplicación.

El conjunto de datos maestro visualiza todos los datos. El conjunto de datos detalle muestra los registros cuyos valores coincidan con los de las columnas vinculadas de la fila actual del conjunto de datos maestro, pero no muestra las columnas vinculadas.

Almacenamiento de los cambios en una relación maestro-detalle

En JBuilder, los datos se recuperan de un servidor o de un archivo de texto al conjunto de datos. Una vez “suministrados” los datos al conjunto de datos, se pueden editar y se puede trabajar desde el programa o en componentes enlazados a datos con una copia local de los datos. Para guardar los datos en la base de datos o en el archivo de texto, se deben “almacenar” los cambios en la base de datos. Las diferentes opciones de almacenamiento de los cambios en la base de datos se tratan en el [Capítulo 8, “Almacenamiento de cambios en la fuente de datos”](#), y las opciones de exportación de datos a un archivo de texto se tratan en [“Exportación de datos” en la página 3-4](#).

En una relación maestro-detalle, se suministran al menos dos conjuntos de datos (cualquier combinación de tablas de bases de datos y/o archivos de

datos de texto) a otros dos conjuntos de datos como mínimo. En general, existen tres formas de almacenar los cambios de una relación maestro-detalle:

- Coloque un `JButton` en la aplicación y escriba el código de almacenador del botón que confirma los datos de cada conjunto de datos. Un ejemplo de esto se puede encontrar en el tema [“Almacenamiento de cambios desde un `QueryDataSet`” en la página 8-3](#).

Si ambos conjuntos de datos son `QueryDataSets`, se pueden guardar los cambios en las tablas maestra y detalle mediante el método `saveChanges(DataSet [])` de la `Database` en vez del método `saveChanges()` para cada conjunto de datos. Una llamada al método `Database.saveChanges(DataSet [])` mantiene sincronizados los conjuntos de datos y confirma el envío de todos los datos en una transacción. Si se utilizan llamadas distintas al método `DataSet.saveChanges()`, los conjuntos de datos no se mantienen sincronizados, y el envío de datos se confirma en transacciones independientes. Consulte [“Resolución de conjuntos de datos maestro-detalle en fuentes de datos JDBC” en la página 9-12](#) para obtener más información.

- Coloque un `QueryResolver` en la aplicación para personalizar el almacenamiento. Consulte [“Personalización de la lógica del almacenador por defecto” en la página 8-17](#) para obtener más información.
- Coloque un componente `JdbNavToolBar` en la aplicación y pulse Guardar. Puede utilizar un solo `JdbNavToolBar` para los dos conjuntos de datos. El componente `JdbNavToolBar` se autovincula automáticamente al conjunto de datos que tenga el foco.

Consulte

- [Capítulo 8, “Almacenamiento de cambios en la fuente de datos”](#)

Resolución de conjuntos de datos maestro-detalle en fuentes de datos JDBC

Dado que una relación maestro-detalle incluye por definición al menos dos conjuntos de datos, la manera más simple de resolver los datos en la fuente de datos es utilizar el método `saveChanges(DataSet [])` del componente `Database` (suponiendo que se utilicen componentes `QueryDataSet`).

Ejecutar el método `saveChanges(DataSet [])` de `Database`, guarda en la fuente de datos JDBC, en una única transacción y por defecto, todas las inserciones, eliminaciones y actualizaciones realizadas en el conjunto de datos. Cuando se ha utilizado la propiedad `masterLink` para establecer

relaciones maestro-detalle entre dos conjuntos de datos, los cambios en los conjuntos de datos relacionados se guardan en el siguiente orden:

- 1 Borrados
- 2 Actualizaciones
- 3 Inserciones

En las eliminaciones y actualizaciones se procesa primero el conjunto de datos detalle. En las inserciones, se procesan antes los conjuntos de datos maestros.

Si una aplicación está utilizando una `JdbNavToolBar` para disponer de las funciones guardar y actualizar, deberá asignar el valor `false` a la propiedad `fetchAsNeeded`, para evitar que las modificaciones no guardadas se pierdan. Ello se debe a que cuando la propiedad `fetchAsNeeded` tiene asignado el valor `true`, cada conjunto de detalle se captura individualmente, y también se actualiza individualmente. Si en su lugar se utiliza el método `Database.saveChanges(DataSet[])`, todas las modificaciones se enviarán a todos los conjuntos de datos vinculados, en el orden correcto y en la misma transacción.

Utilización de módulos de datos para simplificar el acceso a los datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Un módulo de datos es un contenedor especializado para componentes de acceso a datos. Los módulos de datos simplifican las tareas de desarrollo relacionadas con el acceso a datos. Los módulos de datos ofrecen un contenedor centralizado para todos los componentes de acceso a datos. Esto permite modularizar el código y separar las reglas comerciales y la lógica de acceso a las bases de datos de la lógica de la interfaz de usuario de la aplicación. También se puede controlar la utilización de los módulos de datos haciendo llegar los archivos de clase (.class) únicamente a los desarrolladores.

Una vez definidos los componentes `DataSet` y sus correspondientes componentes `Column` en un módulo de datos, todas las aplicaciones que utilicen el módulo tienen acceso coherente a los conjuntos de datos y a las columnas sin necesidad de volver a crearlos en cada aplicación cuando se desea utilizarlos. No es necesario que los módulos de datos residan en el mismo directorio o paquete que el proyecto. Se pueden almacenar en una ubicación donde los programadores y las aplicaciones puedan compartirlos.

`DataModule` es una interfaz que declara el comportamiento básico de un módulo de datos. Para utilizar esta interfaz escribiendo código, implementéla en la clase del módulo de datos y añada componentes de datos.

Cuando se crea un módulo de datos y se añade cualquier componente que vaya a aparecer automáticamente en la sección Acceso a datos del panel de contenido (`Database`, `DataSet`, `DataStore`), se genera un método de

obtención `getter()`. Eso significa que todos estos componentes estarán disponibles en una lista de opciones para el proyecto que hace referencia al módulo de datos. Por ejemplo, eso significa que es posible:

- Añadir un componente `Database` a un módulo de datos.
- Compilar el módulo de datos.
- Añadir un componente `QueryDataSet` a la aplicación que contiene el módulo de datos, o al propio módulo de datos.
- Seleccionar “`DataModule1.database1`” (o algo similar) en el cuadro de opciones Base de datos del cuadro de diálogo de la propiedad `query` (consulta).

En este capítulo se explican dos formas de crear un módulo de datos:

- Por medio de las herramientas de diseño de JBuilder.
- Por medio del modelador de datos.

Creación de un módulo de datos con las herramientas de diseño

En los siguientes apartados se describe cómo se han de crear módulos de datos mediante las herramientas de diseño visual como el Asistente para módulos de datos o el diseñador de interfaces de usuario.

Creación del módulo de datos con el asistente

Para crear un módulo de datos:

- 1 Cree un proyecto.
- 2 Seleccione Archivo | Nuevo y haga doble clic sobre el icono de Módulo de datos.
- 3 Especifique un nombre para el paquete y la clase del módulo de datos.
JBuilder definirá automáticamente el nombre del archivo Java y la vía de acceso en función del nombre que se indique. Para crear el módulo de datos mediante el diseñador de JBuilder, desactive la casilla de selección Llamar al Modelador de datos.
- 4 Haga clic en Aceptar para cerrar el cuadro de diálogo.
La clase módulo de datos se ha creado e incorporado al proyecto.
- 5 Para abrir el panel de contenido haga doble clic en el archivo de módulo de datos en el panel de proyectos.
- 6 Presente el código fuente.

Observe que el asistente ha generado un código para la clase del módulo de datos ligeramente distinto al generado por otros asistentes. Al método `getDataModule()` se le asigna el valor **public static**. El fin de este método es que varios marcos compartan sólo una instancia de este módulo de datos. El código generado por este método es:

```
public static DataModule1 getDataModule() {
    if (myDM == null) {
        myDM = new DataModule1();
    }
    return myDM;
}
```

El código generado por este método es:

- Declara este método como **static**. Esto significa que es posible llamarlo sin instanciar un objeto de la clase `DataModule`.
- Devuelve una instancia de la clase `DataModule`.
- Comprueba si existe una instancia de un `DataModule`.
- Crea y devuelve un `DataModule` si no existe uno.
- Devuelve un objeto `DataModule` si se ha instanciado alguno.

La clase del módulo de datos contiene ahora todos los métodos necesarios y un método vacío para `jbInit()` al que se añaden los componentes de datos y la lógica comercial personalizada.

Adición de componentes de datos al módulo de datos

Para añadir componentes al módulo de datos mediante el diseñador de interfaces de usuario:

- 1 Para abrir el panel de contenido haga doble clic en el archivo de módulo de datos en el panel de proyectos.
- 2 Seleccione la pestaña Diseño del panel de contenido para activar el diseñador de interfaces de usuario.
- 3 Añada los componentes de datos a la clase del módulo de datos.

Por ejemplo:

- a Seleccione un componente `Database` de la pestaña `DataExpress` de la paleta de componentes.
- b Haga clic en el árbol de componentes o en el diseñador de interfaces para añadir el componente `Database` al módulo de datos.
- c Asigne valores a la propiedad `connection` mediante el `connectionDescriptor` de la base de datos. En el [Capítulo 4, “Conexión con bases de datos”](#) se trata la configuración de la propiedad `connection` en el Inspector.

Los componentes de datos se añaden a un módulo de datos de la misma forma que se incorporan a un archivo Marco. Para obtener más información sobre la adición de componentes de datos, consulte el [Capítulo 5, “Recuperación de datos de una fuente de datos”](#)

Nota JBuilder crea automáticamente el código de un método público que lee los componentes DataSet colocados en el módulo de datos. Gracias a ello, los componentes DataSet aparecen como propiedades (de sólo lectura) del módulo de datos. Esto también permite que, en el Inspector, los DataSet sean visibles para la propiedad dataSet de los componentes enlazados a datos cuando estos últimos se emplean junto con los módulos de datos en el mismo contenedor.

Después de completar este apartado, el archivo de módulo de datos tendrá un aspecto parecido al siguiente:

```
paquete de módulo de datos:ejemplo;

import com.borland.dx.dataset.*;
import com.borland.dx.sql.dataset.*;

public class DataModule1 implements DataModule{
    private static DataModule1 myDM;
    Database databasel = new Database();
    public DataModule1() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    private void jbInit() throws Exception{
        databasel.setConnection(new
        com.borland.dx.sql.dataset.ConnectionDescriptor("
        jdbc:borland:dslocal:/usr/local/<jbuilder>/samples/JDataStore/
        datastores/employee.jds", "your name", "", false,
        "com.borland.datastore.jdbc.DataStoreDriver", props));
    }
    public static DataModule1 getDataModule() {
        if (myDM == null)
            myDM = new DataModule1();
        return myDM;
    }
    public com.borland.dx.sql.dataset.Database getDatabasel() {
        return databasel;
    }
}
```

Cómo añadir la lógica empresarial al módulo de datos

Una vez añadidos los componentes de datos al módulo y establecidas las propiedades correspondientes, se puede añadir al módulo la lógica comercial personalizada. Por ejemplo, puede otorgar el derecho a eliminar registros a algunos usuarios y no a otros. Para cerciorarse de que estas reglas se cumplen, habrá de añadir código a varios sucesos de los componentes `DataSet` en el módulo de datos.

Nota Ni los valores de las propiedades ni el código de lógica comercial añadido a los componentes del modelo de datos pueden redefinirse en la aplicación que utiliza el modelo. Si existe un comportamiento que no desea imponer en todas las aplicaciones que utilizan este modelo de datos, considere la posibilidad de crear modelos adecuados a grupos de aplicaciones o de usuarios.

Para añadir código a los sucesos de los componentes, realice los pasos siguientes:

- 1 Para abrir el panel de contenido, haga doble clic en el archivo de módulo de datos en el panel de proyectos.
- 2 Seleccione la pestaña Diseño del panel de contenido para activar el diseñador de interfaces de usuario.
- 3 Seleccione el componente al que desee añadir lógica empresarial y abra la pestaña Sucesos del Inspector.
- 4 Haga doble clic en el suceso donde desea que resida la lógica empresarial.

JBuilder crea un método vacío en el archivo de código Java para que se pueda añadir el código de lógica empresarial personalizada.

Utilización de un módulo de datos

Antes de utilizar un módulo de datos en una aplicación, deberá guardarlo y compilarlo. En el módulo de datos que esté utilizando:

- 1 Elija Archivo | Guardar todo.
Escriba el nombre del proyecto, del paquete y del módulo de datos.
- 2 Compile la clase de módulo de datos seleccionando Ejecutar | Ejecutar Make del proyecto.
Esta orden crea los archivos de la clase de módulo de datos en el directorio indicado en Proyecto | Propiedades de proyecto, Vía de salida.
- 3 Seleccione Archivo | Cerrar.

Para hacer referencia al módulo de datos en la aplicación, primero deberá añadirlo al proyecto como biblioteca necesaria.

Incorporación de una biblioteca necesaria a un proyecto

Estas instrucciones generales sobre cómo añadir bibliotecas necesarias utilizan un módulo de datos como ejemplo, sin embargo, los mismos pasos sirven para añadir otras bibliotecas. Una biblioteca puede ser un archivo de clase, como un módulo de datos, o un archivo comprimido de tipo `.jar`.

Seleccione Proyecto | Propiedades de proyecto. Seleccione Bibliotecas necesarias en la pestaña Vías de acceso y añada la clase o archivo correspondiente a la nueva biblioteca. En el caso específico de un módulo de datos, se tratará del archivo de clase del módulo de datos que se acaba de compilar.

Para ello:

- 1 Pulse Añadir.
- 2 Haga clic en Nuevo.
- 3 Introduzca el nombre de la biblioteca (como Módulo de datos Employee).
- 4 Seleccione la ubicación del archivo `<nombre de biblioteca>.library`.

Puede elegir entre las carpetas JBuilder, Project y una personal. Si está ejecutando JBuilder desde una red, y quiere que su biblioteca esté disponible para todo el mundo, deberá elegir la carpeta JBuilder. De esta forma el archivo `<nombre de biblioteca>.library` estará en el directorio `/lib` de su instalación de JBuilder. Si es usted el único desarrollador que necesita acceder a su biblioteca, puede elegir una de las otras opciones, a fin de que el archivo `.library` se guarde en su ordenador.

- 5 Pulse Añadir.
- 6 Desplácese a la carpeta que contiene la vía de acceso al archivo de clase o recopilatorio al que desea añadirlo.

JBuilder determina automáticamente las vías de acceso a los archivos de clase, los archivos fuente y a la documentación que se encuentra en esta carpeta.

- 7 Pulse Aceptar.
- 8 Pulse Aceptar.
- 9 Pulse Aceptar.

En este momento la nueva biblioteca debe estar en la lista de bibliotecas necesarias.

Hacer referencia a un módulo de datos en una aplicación

Una vez añadido el módulo de datos como biblioteca, para hacer referencia a él en una aplicación deberá proceder como sigue:

- 1 Cierre todos los proyectos abiertos (Archivo | Cerrar proyectos).
- 2 Seleccione Archivo | Nuevo y haga doble clic sobre Aplicación en la ficha General de la galería de objetos.
Se creará tanto una aplicación como un proyecto.
- 3 Introduzca la información de paquete y clase.
- 4 Seleccione el archivo Marco de la aplicación en el panel de contenido.
- 5 Asegúrese de que DataExpress es una de las bibliotecas necesarias. Si DataExpress no aparece en la lista de Bibliotecas en las Propiedades del proyecto,
 - a Pulse Añadir.
 - b Seleccione DataExpress.
 - c Haga clic en Aceptar hasta que se cierre el cuadro de diálogo Propiedades de proyecto.
- 6 Importe el paquete al que pertenece la clase del módulo de datos (si fuera externa al paquete) eligiendo Asistentes | Usar módulo de datos.
- 7 Haga clic en el botón de puntos suspensivos (...) para abrir el cuadro de diálogo Seleccione un módulo de datos.

Aparece un árbol con todas las clases y paquetes conocidos. Desplácese al directorio de los archivos de clase generados cuando se guardó y compiló el módulo de datos (debe aparecer bajo un nodo del árbol, con el mismo nombre que el paquete, si el módulo de datos forma parte de un paquete). Seleccione la clase del módulo de datos. Si la clase de módulo de datos no aparece aquí, compruebe que el proyecto se ha compilado sin errores y que se ha añadido correctamente a las bibliotecas necesarias para el proyecto.

- 8 Pulse Aceptar.

Si al llegar a este punto aparece un mensaje de error, compruebe las bibliotecas en las propiedades del proyecto, así como la ubicación del archivo de clase del módulo de datos.

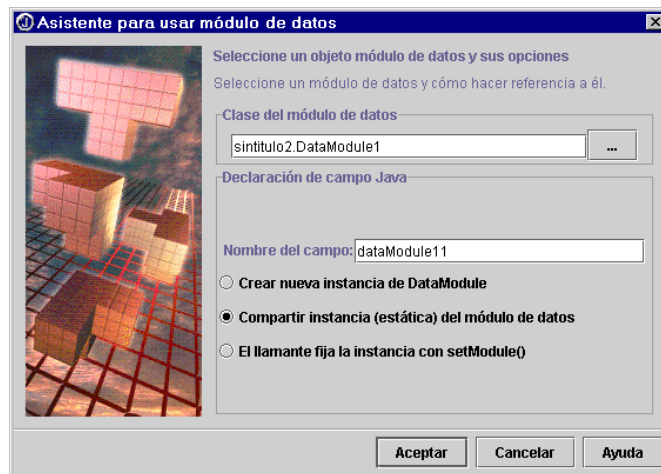
Si hace clic en la pestaña Diseño para abrir el diseñador de interfaces de usuario, la instancia del módulo de datos aparece en el panel de estructura. Hacer clic en la entrada del módulo de datos no presenta los componentes `DataSet` ni sus componentes `Column`. Esto es intencionado y pretende evitar que la lógica empresarial contenida en el módulo de datos puede modificarse desde fuera de éste.

Al diseñar la aplicación, observará que la propiedad `dataSet` de los componentes de interfaz de usuario incluye todos los componentes `DataSetView` y `StorageDataSet` incluidos en el módulo de datos. Es posible acceder a ellos aunque no aparezcan individualmente en el panel de contenido.

Si tiene un módulo de datos complejo y/o una lógica empresarial que no desee que otro programador o usuario modifique, encapsularlos en un componente reutilizable es la manera idónea de facilitar el acceso a los datos y, al mismo tiempo, hacer cumplir y controlar la lógica empresarial.

Cuadro de diálogo Usar módulo de datos

Cuando seleccione Asistentes | Usar módulo de datos, aparecerá el siguiente cuadro de diálogo:



Seleccione un módulo de datos pulsando el botón de puntos suspensivos del campo Clase del módulo de datos. Aparece un árbol con todas las clases y paquetes conocidos. Si no encuentra la clase `DataModule` en la lista, utilice Archivo | Propiedades de proyecto para añadir el paquete o el recopilatorio a las bibliotecas. Diríjase a la ubicación de los archivos de clase generados tras haber guardado y compilado el módulo de datos. Seleccione la clase del módulo de datos.

En el cuadro Declaración de campo Java, el nombre de campo por defecto es el del módulo de datos, seguido de un "1". Éste es el nombre que se utiliza para la variable miembro que se genera en el código. Se hace referencia al módulo de datos por el nombre que se le ha dado en el árbol de componentes. Elija un nombre que describa los datos del módulo de datos, como por ejemplo `EmployeeDataModule`.

En la sección de declaración de campos de Java, puede elegir entre las siguientes formas de utilizar el módulo de datos en la aplicación:

- Crear nueva instancia del módulo de datos
Si sólo hay una subclase `Frame` en la aplicación, seleccione esta opción.
- Compartir instancia (estática) del módulo de datos
Si tiene intención de hacer referencia al módulo de datos en varios marcos de la aplicación y desea compartir una única instancia de la clase `DataModule` personalizada, seleccione esta opción.
- El llamante fija la instancia con `setModule()`
Seleccione esta opción si tiene varios módulos de datos distintos, por ejemplo, uno que obtiene los datos en el equipo local y otro que los obtiene en el equipo remoto.

Pulse Aceptar para añadir el módulo de datos al paquete e incorporar el código apropiado dentro del archivo fuente actual, para crear una instancia del módulo de datos.

Según las opciones del cuadro de diálogo mostrado anteriormente, se añade el siguiente código al método `jbInit()` del archivo Marco. Asegúrese de que la opción Compartir instancia (estática) del módulo de datos está seleccionada:

```
dataModule12 = com.borland.samples.dx.datamodule.DataModule1.getDataModule();
```

Si se ha seleccionado Crear nueva instancia del `DataModule`, se añade el siguiente código al método `jbInit()` del archivo Marco:

```
dataModule12 = new com.borland.samples.dx.datamodule.DataModule1();
```

Si se ha seleccionado El llamante fije la instancia con `SetModule()`, en la clase que se está editando se añade un método `setModule()`.

Creación de módulos de datos con el modelador de datos

El IDE de JBuilder ofrece las herramientas necesarias para generar rápidamente aplicaciones de consulta de bases de datos. El modelador de datos puede crear módulos de datos que encapsulan una conexión con una base de datos y las consultas que se van a ejecutar en ella. El Asistente para aplicaciones de módulo de datos puede utilizar este módulo de datos para crear una aplicación de base de datos cliente-servidor.

Creación de consultas con el modelador de datos

JBuilder simplifica extraordinariamente la labor de visualización y actualización de la información de una base de datos. El Modelador de datos de JBuilder permite crear consultas SQL visualmente y guardarlas en módulos de datos Java de JBuilder.

Para iniciar un nuevo proyecto:

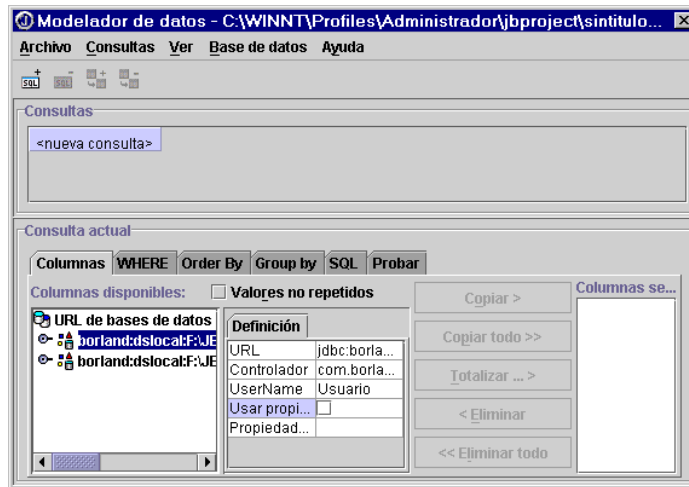
- 1 Elija Archivo | Nuevo proyecto para iniciar el asistente para proyectos.
- 2 Elija la ubicación y el nombre del proyecto.
- 3 Pulse el botón Finalizar.

Para obtener información más detallada acerca de la creación de proyectos, consulte *Creación de aplicaciones con JBuilder*.

Para abrir el Modelador de datos:

- 1 Seleccione Archivo | Nuevo.
- 2 Haga doble clic en el icono Módulo de datos.
- 3 Escriba el nombre de la clase y del paquete del módulo de datos que está creando y active la opción Llamar al modelador de datos.
- 4 Pulse Aceptar. Se abre el Modelador de datos.

Figura 10.1 Modelador de datos



Para abrir en el modelador de datos un módulo de datos Java existente:

- 1 Haga clic con el botón derecho del ratón en el módulo del panel de proyecto.
- 2 Seleccione Abrir con el modelador de datos.

Apertura de URL

Antes de empezar a crear una consulta SQL se debe abrir una URL de conexión. Esto se puede hacer de varias formas:

- Haga doble clic en la URL que accede a los datos.
- Pulse el botón de ampliación.
- Seleccione la URL y elija Base de datos | Abrir URL de conexión.

Si la base de datos a la que desea acceder no aparece en el campo URL de bases de datos del modelador de datos, se puede agregar.

- 1 Para ello, escoja Base de datos | Añadir URL de conexión, con lo cual aparecerá el cuadro de diálogo URL.
- 2 Seleccione un controlador ya instalado de los que aparecen en la lista desplegable, o bien escriba el que desee utilizar.

En el caso de los ejemplos se puede elegir
`com.borland.datastore.jdbc.DataStoreDriver.`

- 3 Introduzca la URL de los datos a los que desee acceder, o bien selecciónela con el botón Examinar.

Para estos ejemplos, puede seleccionar la base de datos `employee.jds` que se encuentra en el directorio de ejemplos de la instalación de JBuilder, /
`samples/JDataStore/datastores/employee.jds`. Se puede utilizar el botón Examinar para buscar este archivo y reducir la posibilidad de teclear un nombre erróneo.

Inicio de una consulta

La generación de una consultar comienza realizando las siguientes acciones:

- 1 Seleccione las columnas que desee añadir a la consulta de una tabla o seleccione una función de totalización que actúe sobre una determinada columna.
 - a Para ver las tablas, haga doble clic en el nodo Tablas o pulse el botón de ampliación Tablas.
 - b Elija en la lista la tabla en la que desea efectuar la consulta y haga doble clic en ella. Haga doble clic en el nodo Columnas para presentar todas las columnas de la tabla seleccionada.

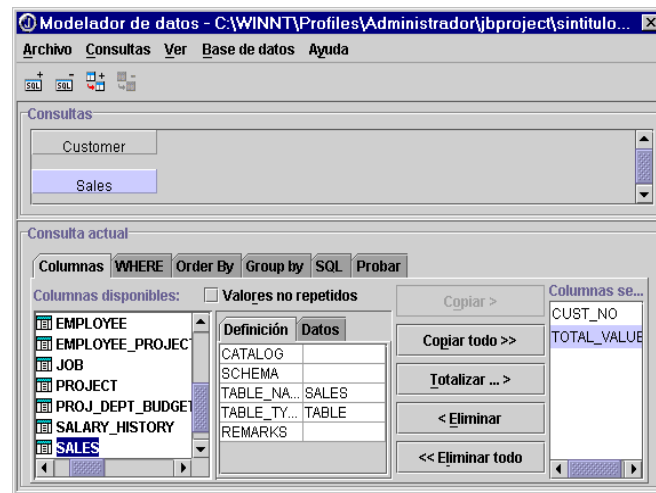
2 Añada una o más columnas a la sentencia SELECT de una consulta:

La sentencia SELECT, de recuperación de datos, devuelve un número variable de filas de un número fijo de columnas. El modelador de datos ayuda en la creación de la sentencia SELECT. La cláusula SELECT indica la lista de columnas que se van a recuperar.

- a Seleccione una columna que desee añadir desde la tabla a la que quiera acceder.
- b Pulse el botón Copiar.

El nombre de la columna seleccionada aparecerá en el cuadro Columnas seleccionadas y el de la tabla, en el panel Consultas de la parte superior. Seleccione todas las columnas que necesite de esa tabla. Para seleccionarlas todas, pulse el botón Copiar todo.

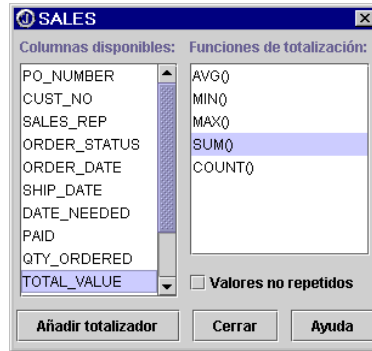
Figura 10.2 Selección de columnas



3 Añada una función de totalización a la consulta.

Las funciones de totalización proporcionan un valor de resumen basado en un conjunto de valores. Las funciones de totalización son SUM, AVG, MIN, MAX y COUNT.

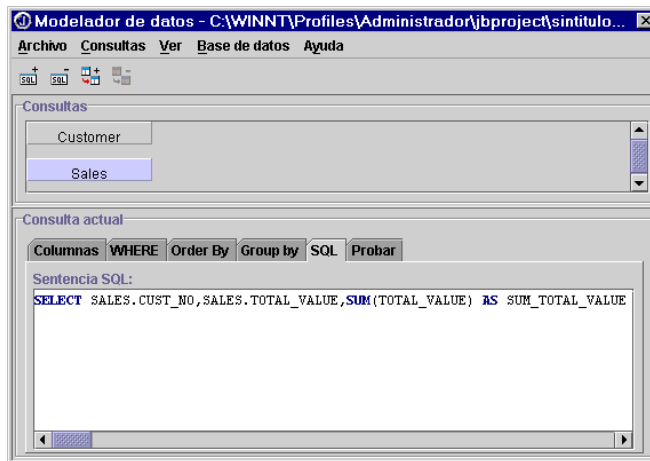
- a Pulse el botón Totalizar para abrir el cuadro de diálogo.

Figura 10.3 Cuadro de diálogo Totalizar

- b** En la lista de Columnas disponibles, haga clic en la columna cuyos valores desee totalizar.
- c** En la columna Funciones de totalización, seleccione la función que quiera aplicar a esa columna.
- d** Si prefiere que la función se aplique sólo a valores no repetidos de la columna seleccionada, marque la casilla Valores no repetidos.
- e** Seleccione Añadir totalización para añadir la función a la consulta.

A medida que seleccione columnas y añada funciones, se irá generando una sentencia SQL SELECT. Cuando se efectúan totalizaciones de datos se debe incluir una cláusula GROUP BY. Si desea información sobre las cláusulas GROUP BY, consulte [“Adición de la cláusula Group by” en la página 10-14](#).

Para ver la sentencia SQL, abra la pestaña SQL.



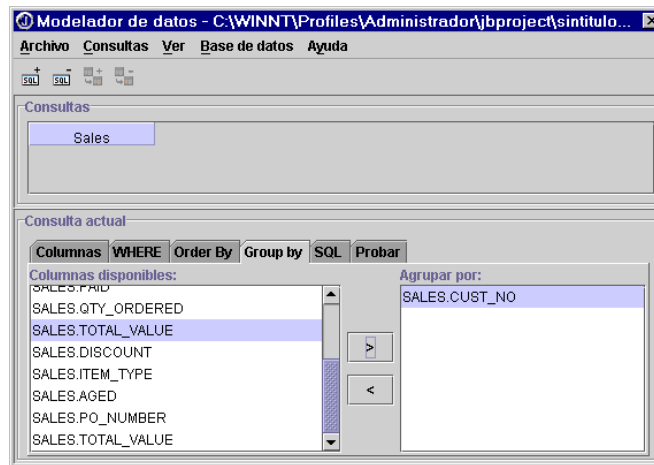
Adición de la cláusula Group by

La cláusula GROUP BY agrupa los datos devueltos por una sentencia de selección y se utiliza a menudo en combinación con las funciones de totalización. Cuando se utiliza con funciones de totalización se sigue el siguiente proceso:

- En primer lugar, los datos están restringidos por una cláusula WHERE, si la hay.
- Los datos se agrupan a partir del campo indicado en la cláusula GROUP BY.
- Las funciones de totalización se aplican a los grupos y se presenta una fila de resumen por cada grupo.

Para añadir una cláusula Group by a la consulta, pulse la pestaña Group By para presentar su ficha.

Figura 10.4 Ficha Group By



En el cuadro Columnas disponibles aparece una lista de las columnas de la consulta que está seleccionada en el panel Consultas del modelador de datos. El cuadro GROUP BY contiene los nombres de las columnas por las cuales se agrupará la consulta. Por defecto, hasta que no se seleccione alguna columna, la consulta no estará agrupada por ninguna en concreto.

Para añadir a la consulta una cláusula GROUP BY,

- 1 Seleccione la columna por la que desee agruparla.
- 2 Pulse el botón de adición (>) para pasar el nombre de la columna al cuadro GROUP BY.

Se añade, así, una cláusula Group by a la sentencia SQL SELECT. Para mostrarlo, haga clic en la pestaña SQL.

Selección de filas con valores de columna unívocos

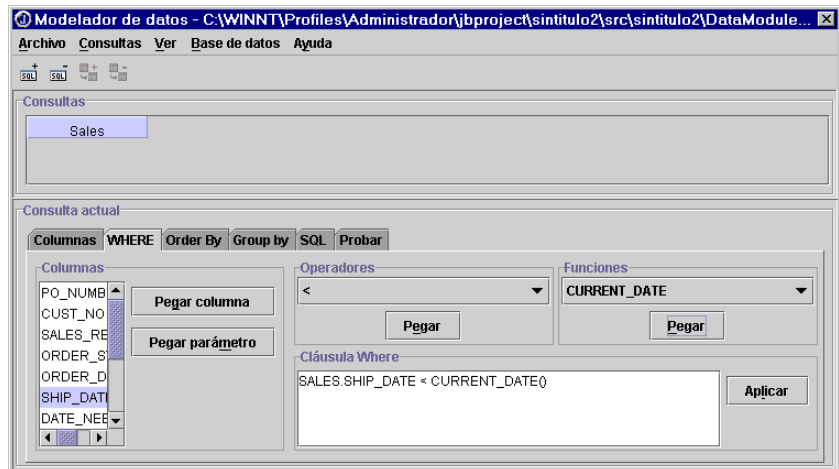
En ocasiones, interesa ver únicamente las filas que contienen valores de columna no repetidos. Si se añade la palabra clave **DISTINCT** a la sentencia **SELECT**, sólo aparecerán filas con valores no repetidos. **DISTINCT** afecta a todas las columnas de la sentencia **SELECT**.

Para añadir la palabra clave **DISTINCT**, seleccione la opción **Valores no repetidos** en la ficha **Columnas**.

Inclusión de una cláusula WHERE

Cuando se añade una cláusula **WHERE** a una sentencia de selección, indica la condición de búsqueda que se debe cumplir para que las filas se incluyan en la tabla de resultado. Para añadir una cláusula **Where** a la consulta **SQL**, haga clic en la pestaña **Where**.

Figura 10.5 Ficha Where



En la lista **Columnas** de la izquierda aparecen las columnas de las tablas de la consulta que está seleccionada en el panel **Consultas** del modelador de datos.

Genere la consulta en el cuadro **Cláusula WHERE**, utilizando las **Columnas**, **Operadores** y **Funciones** de la siguiente forma:

- 1 Si desea incluir en el cuadro **Cláusula WHERE** el nombre de una columna, selecciónela en la lista y pulse el botón **Pegar columna**.
- 2 Para pasar una columna como parámetro, como en una consulta parametrizada, selecciónela en la lista **Columnas** y pulse el botón **Pegar parámetro**.
- 3 Seleccione un operador de la lista desplegable de operadores y pulse el botón **Pegar**.

Cada cláusula WHERE requiere al menos un operador.

- 4 Si la consulta requiere una función, seleccione la necesaria de la lista desplegable Funciones y pulse el botón Pegar.

A medida que vaya pegando selecciones, se irá generando una cláusula WHERE. También se puede completar la consulta editando directamente el texto del cuadro Cláusula WHERE. Por ejemplo, supongamos que estamos generando una cláusula WHERE como la siguiente:

```
WHERE PAIS = 'FRANCIA'
```

Para ello, tendremos que seleccionar y pegar la columna PAIS y el operador =. Para finalizar la consulta, introducimos directamente el valor del dato, en este caso, 'FRANCIA'.

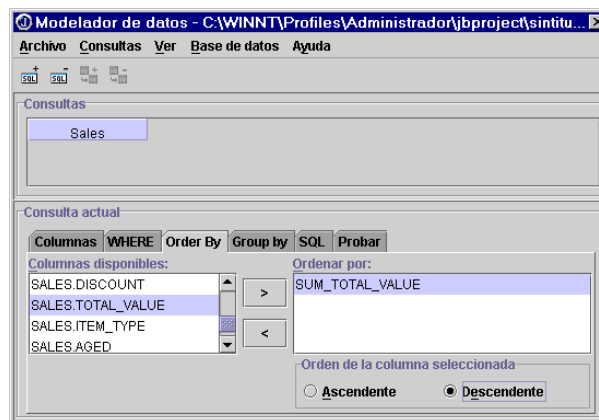
Cuando haya terminado de especificar la cláusula WHERE, pulse el botón Aplicar. La cláusula WHERE se añade a la sentencia SQL SELECT. Para mostrarlo, haga clic en la pestaña SQL.

Inclusión de una cláusula ORDER BY

Se utiliza una cláusula ORDER BY para ordenar o reordenar los datos de la tabla de resultado. Para especificar cómo deberán clasificarse las filas de una tabla:

- 1 En el panel Consultas, seleccione la que desee ordenar.
- 2 Pulse la pestaña Order by del panel Consulta actual.

Figura 10.6 Ficha Order by



- 3 En el cuadro Columnas disponibles, seleccione la columna por la que prefiera ordenar los resultados de la consulta y pulse el botón con el símbolo (>) para pasarla al cuadro ORDER BY.
- 4 Elija la opción Orden de la columna seleccionada que desee utilizar.

La opción Ascendente ordena la columna empezando por el valor más pequeño y terminando por el más grande, mientras que Descendente la

ordena al revés. Por ejemplo, si la columna que se desea ordenar es alfabética, Ascendente la clasificará por orden alfabético y Descendente lo hará en orden inverso.

Si desea ordenar la consulta por varias columnas, puede pasarlas al cuadro ORDER BY. Seleccione primero la columna que determinará el orden de clasificación principal, después la del secundario, y así sucesivamente. Por ejemplo, si la consulta incluye una columna de País y otra de Clientes, y quiere ver todos los clientes de un país, tendrá que pasar al cuadro ORDER BY primero la columna País, y luego la de Clientes.

Edición directa de la consulta

Cuando se utiliza el modelador de datos para crear una consulta, se puede examinar y editar la sentencia SQL SELECT en cualquier momento.

Para ver la sentencia SELECT, haga clic en la pestaña SQL. Para editarla, introduzca directamente los cambios en la sentencia SELECT.

Comprobación de consultas

Es posible comprobar los resultados de la consulta en el modelador de datos. La consulta creada en este tema no se ejecuta; estas explicaciones están encaminadas a facilitar la comprensión del proceso y no a activar las propiedades de la consulta que se va a ejecutar.

Para ver los resultados de la consulta que está generando,

- 1 Pulse sobre la pestaña Comprobar.
- 2 Pulse el botón Ejecutar consulta.

Si la consulta es parametrizada, se abre el cuadro de diálogo Especificar parámetros con el fin de que pueda introducir sus respectivos valores. Seleccione Aceptar para ejecutar la consulta y ver los resultados. Los valores introducidos no se guardan en el módulo de datos.

Generación de consultas múltiples

Si desea generar varias consultas, seleccione Consultas | Añadir, y el modelador de datos quedará preparado para generar una nueva consulta. A medida que seleccione columnas de una o varias tablas, los nombres de éstas irán apareciendo en el campo <nueva consulta>.

Especificación de una relación maestro-detalle

Para definir una relación maestro detalle entre dos consultas:

- 1 Abra el cuadro de diálogo Enlazar consultas. Puede hacerlo de dos maneras:
 - a Seleccionando Consultas | Enlazar.

- b** En el panel Consultas, haga clic y arrastre el puntero desde la consulta que desee definir como maestro y soltándola sobre la de detalle.

Figura 10.7 Cuadro de diálogo Enlazar consultas

Consulta maestra: **Customer** Consulta de detalle: **Sales**

Especificar la relación entre la consulta maestra y la consulta detalle:

Customer	Sales
CUST_NO	CUST_NO

Relación: Muchos-a-muchos

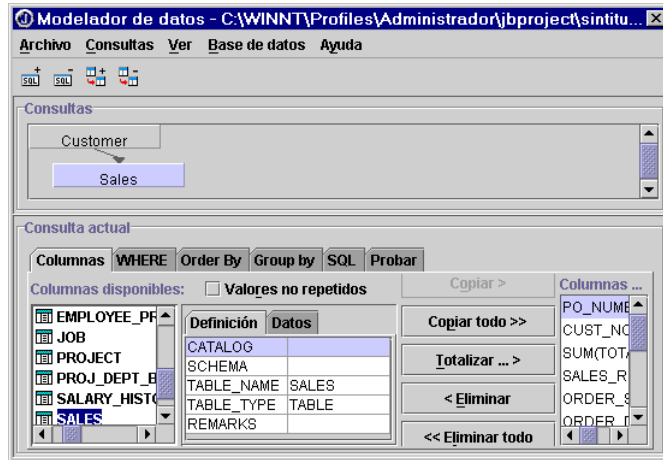
☒ Retrasar obtención de registros detalle hasta que se necesiten
☐ Permitir actualizar en cascada
☐ Permitir borrar en cascada

Aceptar Cancelar Ayuda

- 2** En la lista Consulta maestra, seleccione la que desee utilizar para el maestro.
- 3** En la lista Consulta de detalle, seleccione la que desee usar para el detalle.

Aparecerá una serie de campos predeterminados para las consultas maestras y de detalle. Si no son los que desea utilizar, haga las modificaciones oportunas.
- 4** Puede utilizar la tabla para especificar visualmente las columnas que enlazarán las consultas maestras y de detalle:
 - a** Haciendo clic en la primera fila de la tabla situada justo debajo de la columna de la consulta maestra, aparece una lista desplegable de todas las columnas especificadas de la tabla maestra. Seleccione la columna por la cual desee agrupar los datos de detalle.
 - b** Haciendo clic en la fila de la tabla situada justo debajo de la columna de la consulta de detalle, aparece una lista desplegable de todas las columnas con el mismo tamaño y tipo de datos que la columna maestra que está seleccionada. Seleccionando la columna apropiada, quedarán enlazadas las tablas de maestro y de detalle.
 - c** Pulse Aceptar.

Al cerrar el cuadro de diálogo Enlazar consultas, aparecerá una flecha entre las dos consultas indicando las relaciones entre ellas.

Figura 10.8 Flecha que muestra la relación entre columnas

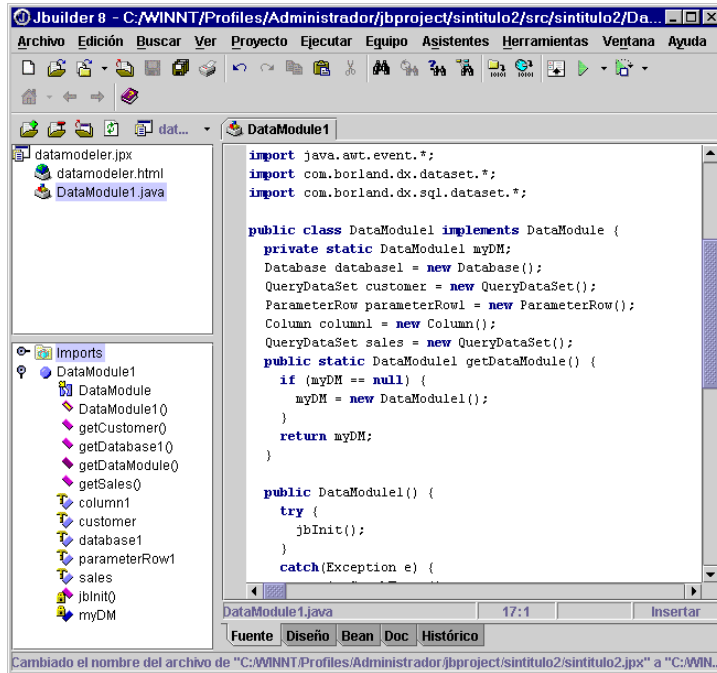
Si desea más información sobre las relaciones maestro-detalle, consulte el [Capítulo 9, “Establecimiento de una relación maestro-detalle”](#)

Almacenamiento de consultas

Para guardar los módulos de datos que se han generado:

- 1 Seleccione Archivo | Guardar en el Modelador de datos y especifique un nombre con una extensión .java.
- 2 Salga del modelador de datos.
El archivo resultante aparecerá en el proyecto.
- 3 Compile el módulo de datos.

Haga doble clic en el archivo, en el panel de proyectos, para abrirlo en el panel de contenido y presentar el código generado por el modelador de datos.

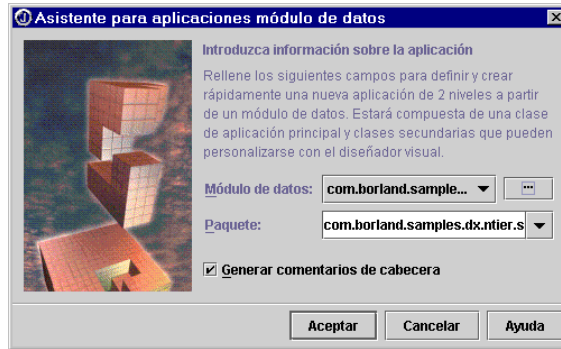
Figura 10.9 El editor muestra el código generado por el modelador de datos

Generación de aplicaciones de base de datos

A partir del módulo de datos compilado, JBuilder puede generar aplicaciones cliente-servidor con el Asistente para crear aplicación a partir de un módulo de datos.

Para abrir el Asistente para aplicaciones módulo de datos, pulse el botón correspondiente de la galería de objetos:

- 1 Seleccione Archivo | Nuevo y, a continuación, la pestaña General.
- 2 Haga doble clic en el icono Aplicación de módulo de datos.

Figura 10.10 Asistente para aplicaciones módulo de datos

- 3 En el cuadro de diálogo que se muestra, seleccione el archivo de modulo de datos a partir del que quiere generar la aplicación.

Puede seleccionarse cualquiera de los módulos de datos disponibles, o uno creado con el modelador de datos.

- 4 Pulse Aceptar.

Este asistente crea automáticamente una aplicación de base de datos. El asistente genera varios archivos Java y un archivo HTML.

- Los archivos que constituyen el cliente se encuentran en un paquete client2tier:
 - Uno o varios `UIBeans.java`: Cada bean implementa una interfaz de usuario en columna para un conjunto de datos.
 - `ClientAboutBoxDialog.java`: Implementa el cuadro de diálogo cliente Ayuda Acerca de.
 - `ClientFrame.java`: Marco de la aplicación cliente que es el contenedor de la interfaz de usuario cliente por defecto. Implementa la barra de menús de la aplicación.
 - `ClientResources.java`: Contiene cadenas de la aplicación cliente que hay que traducir.
- `<módulodatos>TwoTierApp.java`: La aplicación.
- `<módulodatos>AppGenFileList.html`: Lista de los archivos generados, con una descripción breve de cada uno.

Utilización de módulos de datos generados en el código

Una vez creado el módulo de datos con el modelador de datos, se puede utilizar en la aplicación que se ha escrito. Siga estos pasos:

- 1 Ejecute el Asistente para usar módulos de datos.

Añade en el código fuente del marco de la aplicación un método `setModule()` que identifica el módulo de datos. El método `setModule()` que crea el asistente llama al método `jbInit()` del marco. Asimismo, el asistente borra la llamada que el constructor del marco realiza a `jbInit()`.

- 2 Desde el código fuente del archivo de aplicación, llame al método `setModule()` del marco, pasando una clase del módulo de datos.

Por ejemplo, supongamos que se ha utilizado el modelador de datos para crear un módulo de datos denominado `CountryDataModelModule`. para acceder, desde una aplicación, a la lógica contenida en este módulo de datos, debe añadirse a la clase marco de aquella un método `setModule()`.

Para añadir este método `setModule()` y eliminar el método `jbInit()` del constructor del marco:

- 1 Añada el módulo de datos a la lista de bibliotecas requeridas (en el cuadro de diálogo Proyecto | Propiedades de proyecto).
- 2 Seleccione Asistentes | Usar módulo de datos, con el código fuente del marco visible en el editor.
- 3 Indique el módulo de datos que desea emplear con el asistente.
- 4 Active la opción El llamante fija la instancia con `setModule()`.
- 5 Pulse Aceptar.

Éste es el código resultante de estas acciones:

```
package com.borland.samples.dx.myapplication;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//importa el paquete en el que se encuentra el módulo de datos
import com.borland.samples.dx.datamodule.*;

public class Marco1 extends JFrame {
    BorderLayout borderLayout1 = new BorderLayout();
    CountryDataModelModule countryDataModelModule1;

    //Construir el marco sin llamar a jbInit()
    public Marco1() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    }

    //Inicialización de componente
    private void jbInit() throws Exception{
        this.getContentPane().setLayout(borderLayout1);
        this.setSize(new Dimension(400, 300));
        this.setTitle("Título del marco");
    }

    //Sobreescribo por lo que se puede salir de System Close
```



```
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}

// El asistente para usar módulos de datos ha añadido este código
public void setModule(CountryDataModelModule countryDataModelModule1) {
    this.countryDataModelModule1 = countryDataModelModule1;
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
}
```

Observe que ahora se llama al método `jbInit()` del marco después de definir el módulo y no desde el constructor del marco.

Acto seguido, ha de llamarse al nuevo método `setModule()` desde el código fuente principal de la aplicación. Desde el constructor de la aplicación, llame a `setModule()`, pasándole la clase del módulo de datos. Así queda el código de la aplicación:

```
package com.borland.samples.dx.myapplication;

import javax.swing.UIManager;

public class Application1 {
    boolean packFrame = false;

    //Crear la aplicación
    public Application1() {
        Marcol frame = new Marcol();

        // Esta es la línea de código que se ha añadido
        frame.setModule(new untitled3.CountryDataModelModule());

        //Validar marcos que tienen tamaños preestablecidos
        //Empaquetar marcos con información de tamaño preferente útil, p. ej. del
        diseño
        if (packFrame)
            frame.pack();
        else
            frame.validate();
        frame.setVisible(true);
    }

    //método Main
    public static void main(String[] args) {
```

```
        try {  
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
        }  
        catch(Exception e) {  
        }  
        new Application1();  
    }  
}
```

Filtrado, clasificación y localización de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Una vez finalizada la fase de suministro de la aplicación, y con los datos en el componente `DataSet` de un paquete `DataExpress` adecuado, se puede empezar a trabajar con la funcionalidad central de la aplicación y su interfaz de usuario. En este capítulo se muestran algunas de las operaciones más frecuentes en las aplicaciones de base de datos, como el filtrado, la clasificación y la localización de información.

Una característica de diseño del paquete `DataExpress` es que la manipulación de los datos es independiente de su obtención. Sea cual fuere el tipo de componentes `DataSet` utilizado en la obtención de los datos, éstos se manipulan y conectan a controles exactamente de la misma forma. Casi todos los ejemplos de este capítulo utilizan el componente `QueryDataSet`, componente que se puede sustituir, sin tener que cambiar el código del cuerpo principal de la aplicación, por `TableDataSet` o por cualquier subclase de `StorageDataSet`.

Todos los ejemplos que se tratan en este capítulo han sido creados con el Visualizador de aplicaciones y las herramientas de diseño de JBuilder. Siempre que sea posible, se debe generar el código Java fuente con estas herramientas. Donde sea necesario, se mostrará el código que hay que modificar, para que la aplicación ejecute una tarea determinada.

La información que se presenta en este capítulo da por supuesto que ya conoce el funcionamiento del entorno de JBuilder. No se ofrecen pasos detallados acerca de cómo se debe utilizar la interfaz de usuario. Si todavía no está muy familiarizado con JBuilder, diríjase a [Capítulo 16, “Tutorial: Importación y exportación de datos desde un archivo de texto”](#), “El Visualizador de aplicaciones” en *Introducción a JBuilder* o “Introducción al diseñador” en *Diseño de aplicaciones con JBuilder*.

Todo el material que se incluye en este capítulo requiere acceso SQL a datos almacenados en bases de datos locales. Si desea instrucciones sobre la forma de configurar JBuilder para utilizar el controlador JDataStore de ejemplo, consulte [“Adición de un controlador JDBC a JBuilder” en la página 4-10](#).

Se recomienda utilizar los ejemplos como guía al añadir estas funciones a la aplicación. Para muchos de los conceptos que se tratan en este capítulo, se proporcionan proyectos finalizados y archivos fuente en Java con comentarios en el archivo fuente en donde se ha considerado necesario. Todos los archivos a los que hacen referencia estos ejemplos se encuentran en el directorio de ejemplos de JBuilder.

Para crear una aplicación de base de datos, es necesario, en primer lugar, conectarse con una base de datos y suministrar datos a un `DataSet`. [“Suministro de datos de los ejemplos” en la página 11-2](#) configura una consulta que se puede utilizar en algunos de los ejemplos de este capítulo. La siguiente lista de opciones adicionales de funcionalidad de base de datos (filtros, clasificación, localización de datos) pueden combinarse entre sí, por ejemplo, puede optarse por ocultar temporalmente todos los empleados cuyos apellidos empiecen por las letras de la “M” a la “Z”.

- El filtrado temporal oculta determinadas filas de un `DataSet`.
- La clasificación cambia el orden de un `DataSet` con o sin filtros.
- La localización sitúa el cursor en el `DataSet` con o sin filtros.

Suministro de datos de los ejemplos

En este apartado se explica la forma de configurar una aplicación de base de datos sencilla, que se puede utilizar con los ejemplos de este capítulo. La consulta que se va a utilizar en estos ejemplos es:

```
SELECT * FROM EMPLOYEE
```

Esta sentencia SQL selecciona todas las columnas de una tabla denominada `EMPLOYEE`, que se incluye en el `JDataStore` de ejemplo.

Para configurar una aplicación para su uso con los ejemplos:

- 1 Cierre todos los proyectos abiertos (elija Archivo | Cerrar proyecto).
- 2 Seleccione Archivo | Nuevo proyecto.
- 3 Escriba en el Asistente para proyectos el nombre y la vía de acceso del proyecto. Pulse el botón Finalizar.
- 4 Seleccione Archivo | Nuevo en el menú y haga doble clic en el icono Aplicación de la ficha General de la galería de objetos.
- 5 Indique en el Asistente para aplicaciones el nombre del proyecto y la clase. Pulse el botón Finalizar.

- 6 Active el diseñador de interfaces de usuario, seleccionando la pestaña Diseño.
- 7 Haga clic en el componente `Database` de la pestaña `DataExpress` de la paleta de componentes y, a continuación, haga clic en el árbol de componentes o en el diseñador de interfaces de usuario para añadir el componente a la aplicación.
- 8 Abra el editor de la propiedad `connection` del componente `Database`, seleccionándolo y haciendo doble clic en el botón puntos suspensivos de la propiedad `connection` en el Inspector. Asigne valores a las propiedades de conexión para la tabla del empleado de ejemplo de `JDataStore`, de la forma siguiente: La URL de conexión apunta al directorio de instalación. Si se ha instalado `JBuilder` en un directorio distinto, es necesario indicarlo aquí.

Nombre de la propiedad	Valor
Controlador	<code>com.borland.datastore.jdbc.DataStoreDriver</code>
URL	Busque el archivo <code><jbuilder>/samples/JDataStore/datastores/employee.jds</code>
Nombre de usuario	Introduzca su nombre
Contraseña	No es obligatoria

El cuadro de diálogo `Connection` contiene un botón `Probar conexión`. Púlselo para comprobar que las propiedades de conexión tienen los valores correctos. Los resultados del intento de conexión se muestran en el área de estado. Cuando la conexión sea satisfactoria, pulse `Aceptar`. Si la conexión no se efectúa adecuadamente, asegúrese de que ha seguido todos los pasos del [Capítulo 4, “Conexión con bases de datos”](#).

- 9 Añada un componente `QueryDataSet` al diseñador, haciendo clic en el componente `QueryDataSet` de la pestaña `DataExpress` y después haga clic en el árbol de componentes.

Seleccione en el Inspector la propiedad `query` del componente `QueryDataSet`, pulse el botón de puntos suspensivos para que se abra el cuadro de diálogo `QueryDescriptor` y asigne valores a las siguientes propiedades:

Nombre de la propiedad	Valor
Database	<code>database1</code>
Sentencia SQL	<code>SELECT * FROM EMPLOYEE</code>

Haga clic sobre Probar consulta para cerciorarse de que se ejecuta correctamente. Cuando en el área de estado se indique Correcto, cierre el cuadro de diálogo pulsando Aceptar.

- 10 Añada el componente `DBDisposeMonitor` de la pestaña Más dbSwing. El componente `DBDisposeMonitor` cierra el `JDataStore` cuando se cierre la ventana.
- 11 Asigne a la propiedad `dataAwareComponentContainer` de `DBDisposeMonitor` el valor `this`.

Para ver los datos en la aplicación, añada los siguientes componentes a la interfaz y asócielos al conjunto de datos como se explica a continuación:

- 1 Seleccione `ContentPane(BorderLayout)` y asigne a su propiedad `layout` el valor `null`.
- 2 Coloque un `JdbNavToolBar` en la zona de la parte superior del panel, en el diseñador de interfaces. `jdbNavToolBar1` se asocia automáticamente al objeto `DataSet` que tiene el foco, por lo que no es necesario definir su propiedad `dataSet`.
- 3 Coloque un `JdbStatusLabel` en la zona de la parte inferior del panel, en el diseñador de interfaces. `jdbStatusLabel1` se asocia automáticamente al objeto `DataSet` que tiene el foco, por lo que no es necesario asignar valores a su propiedad `dataSet`.
- 4 Añada un componente `TableScrollPane` de la pestaña dbSwing al centro del panel, en el diseñador de interfaces.
- 5 Coloque un componente `JdbTable` en el centro de `tableScrollPane1` y asigne a su propiedad `dataSet` el valor `queryDataSet1`.

Puede observar que, en este momento, el diseñador muestra datos dinámicos.

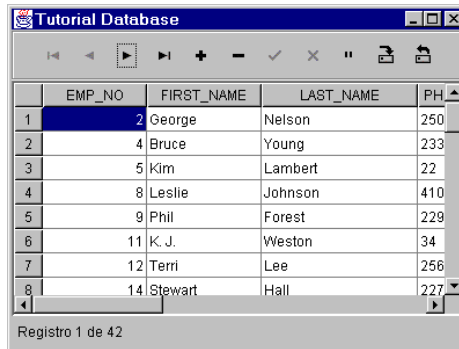
- 6 Seleccione Ejecutar | Ejecutar proyecto para ejecutar la aplicación y examinar el conjunto de datos.

El conjunto de datos `EMPLOYEE` contiene 42 registros y 11 campos. En la etiqueta de estado de esta aplicación se ve el número de registros visibles. Cuando la aplicación se ejecuta por primera vez, en la etiqueta de estado se puede leer Registro 1 de 42. Algunos ejemplos eliminan filas de la vista. La etiqueta de estado muestra el número de filas recuperadas en el conjunto de datos de cada aplicación.

Puede encontrar más información sobre la recuperación de datos para la aplicación en el [Capítulo 5, “Recuperación de datos de una fuente de datos”](#).

La aplicación en ejecución debe tener un aspecto similar a éste:

Figura 11.1 Ejecución de la aplicación de base de datos



	EMP_NO	FIRST_NAME	LAST_NAME	PH
1	2	George	Nelson	250
2	4	Bruce	Young	233
3	5	Kim	Lambert	22
4	8	Leslie	Johnson	410
5	9	Phil	Forest	229
6	11	K. J.	Weston	34
7	12	Terri	Lee	256
8	14	Stewart	Hall	227

Registro 1 de 42

Filtrado de datos

Los filtros ocultan temporalmente las filas de un conjunto de datos, lo que permite seleccionar, ver y trabajar con un subconjunto de filas del conjunto de datos. Por ejemplo, se pueden presentar todos los pedidos de un cliente, todos los clientes de un país determinado o todos los pedidos no entregados en un plazo de dos días. En lugar de ejecutar una consulta nueva cada vez que se cambien los criterios, se puede utilizar un filtro que muestre una vista nueva.

En JBuilder, el usuario proporciona el código de filtro que el conjunto de datos llama mediante un suceso por cada fila de datos para determinar si la fila se incluye o no en la vista actual. El método debe examinar la fila transmitida cada vez que se le llama y, a continuación, indicar si se la debe incluir en la vista o no. Esto se indica llamando a los métodos `add()` o `ignore()` de un objeto `RowFilterResponse` pasado.

El usuario captura el código para el suceso `filterRow` de un conjunto de datos mediante la ficha Sucesos del Inspector. Cuando se abre el conjunto de datos, o al dejarlo abierto de manera implícita al ejecutar un marco con un control asociado al conjunto de datos, el filtro se implementa. En este ejemplo se utilizan componentes de interfaz que permiten al usuario solicitar un filtro nuevo sobre la marcha.

Un filtro de un conjunto de datos es un mecanismo que limita las filas visibles. El conjunto de datos subyacente no se ha modificado, sólo ha cambiado la vista actual de los datos, que es transitoria. Una aplicación puede cambiar los registros de la vista actual “sobre la marcha”, en respuesta a una solicitud del usuario (como se muestra en el ejemplo siguiente) o de acuerdo con la lógica de la aplicación (por ejemplo, mediante la visualización de todas las filas que se van a borrar antes de guardar los cambios, para confirmar o cancelar la operación). Cuando se

utiliza una vista filtrada de los datos y se envía una edición ajena a las especificaciones de filtro, la fila desaparece de la vista, aunque permanece en el conjunto de datos.

Se pueden utilizar varias vistas del mismo conjunto de datos a la vez utilizando un `DataSetView`. Para obtener más información sobre como trabajar con múltiples vistas del mismo conjunto de datos, consulte [“Presentación de una vista alternativa de los datos” en la página 12-24](#).

En ocasiones, se confunde el filtrado con la clasificación y la búsqueda.

- El filtrado temporal oculta determinadas filas de un `DataSet`.
- La clasificación cambia el orden de un `DataSet` con o sin filtros. Para obtener más información sobre la clasificación de datos, consulte [“Clasificación de datos” en la página 11-9](#).
- La localización sitúa el cursor en el `DataSet` con o sin filtros. Para obtener más información sobre la localización de datos, consulte [“Localización de datos” en la página 11-15](#).

Añadir y eliminar filtros

Este apartado muestra la forma de utilizar un `RowFilterListener` de conjunto de datos para ver únicamente filas que cumplan los criterios de filtro. En este ejemplo, se crea un `JdbTextField` que permite especificar la columna que se ha de filtrar. A continuación, se crea otro `JdbTextField` que permite especificar el valor que debe haber en la columna para que el registro se incluya en la vista. Se añade un `JButton` para permitir que el usuario determine cuándo aplicar los criterios de filtro y para mostrar sólo las filas cuyas columnas especificadas contengan exactamente los valores especificados.

En este tutorial, utilizará un componente `QueryDataSet` conectado con un componente `Database` para capturar datos, pero los filtros pueden aplicarse a cualquier componente `DataSet`.

El ejemplo terminado está disponible como proyecto completo en el subdirectorio `/samples/DataExpress/FilterRows` del directorio de instalación de JBuilder.

Para crear esta aplicación:

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#). Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos.
- 2 Haga clic en la pestaña Diseño.
- 3 Añada dos componentes `JdbTextField` de la pestaña `dbSwing` y un componente `JButton` de la pestaña `Swing`. Los componentes `JdbTextField`

permiten introducir un campo y un valor para filtrar. El componente `JButton` ejecuta el mecanismo de filtrado.

- 4 Defina el nombre de la columna que se va a filtrar y su formateador. Para ello, seleccione la pestaña Fuente y añada esta sentencia **import** a las demás sentencias **import**:

```
import com.borland.dx.text.VariantFormatter;
```

- 5 Añada estas definiciones de variable a las existentes en la definición de clase:

```
Variant v = new Variant();
String columnName = "Last_Name";
String columnValue = "Young";
VariantFormatter formatter;
```

- 6 Especifique el mecanismo de filtro. Las filas incluidas en una vista se limitan añadiendo un `rowFilterListener` y utilizándolo para definir qué filas se deben mostrar. La acción por defecto en un `rowFilterListener` es excluir la fila. El código debe llamar al método `add()` de `RowFilterResponse` por cada fila que deba incluirse en la vista. En este ejemplo, comprobará si los campos `columnName` y `columnValue` están en blanco. Si alguno está en blanco, todas las filas se añaden a la vista actual.

Para crear `RowFilterListener` como adaptador de sucesos mediante las herramientas de diseño visual:

- a Seleccione la pestaña Diseño.
- b Seleccione `queryDataSet1` en el árbol de componentes.
- c Seleccione la pestaña Sucesos en el Inspector.
- d Seleccione el suceso `filterRow`.
- e Haga doble clic en el cuadro de valor de `filterRow`.

Se genera automáticamente un `RowFilterListener` como clase interna, que llama a un nuevo método de su clase, denominado `queryDataSet1_filterRow`.

- f Añada el código de filtro al suceso `queryDataSet1_filterRow`. Se puede copiar el código de la ayuda en pantalla seleccionando el código y pulsando **Ctrl+C** o mediante Edición | Copiar del Visualizador de ayuda.

```
void queryDataSet1_filterRow(ReadRow row, RowFilterResponse
response) {
    try {
        if (formatter == null || columnName == null ||
            columnValue == null || columnName.length() == 0 ||
            columnValue.length() == 0)
            // los campos definidos por el usuario están en blanco; se añaden
            las filas
```

```

        response.add();
    } else {
        row.getVariant(columnName, v);
        // obtiene el valor de fila de la columna
        // le da formato de cadena
        String s = formatter.format(v);
        // true significa que esta fila se muestra
        if (columnValue.equals(s))
            response.add();
        else response.ignore();
    }
}
catch(Exception e) {
    System.err.println("Filter example failed");
}
}

```

7 Redefina el suceso `actionPerformed` de `JButton` para que vuelva a disparar el filtrado real de los datos. Para ello:

- a** Seleccione la pestaña Diseño.
- b** Seleccione `JButton` en el árbol de componentes.
- c** Haga clic en la pestaña Sucesos en el Inspector.
- d** Seleccione el suceso `actionPerformed` y haga doble clic en el cuadro de valor de este suceso.

La pestaña Fuente muestra el stub del método `jButton_actionPerformed`. El siguiente código utiliza la clase adaptador para realizar el filtrado propiamente dicho de los datos desconectando y volviendo a conectar el adaptador de sucesos `rowFilterListener` generado en el paso anterior.

e Añada este código al stub generado.

```

void jButton1_actionPerformed(ActionEvent e) {

    try {

        // Obtener nuevos valores para las variables que usa el filtro
        // y hacer que los datos se vuelvan a filtrar.

        columnName = jdbcTextField1.getText();
        columnValue = jdbcTextField2.getText();
        Column column = queryDataSet1.getColumn(columnName);
        formatter = column.getFormatter();

        // Volver a calcular los filtros

        queryDataSet1.refilter();

        // Ahora, la tabla vuelve a trazar sólo las columnas que cumplen
        // estos criterios

    }
}

```

```

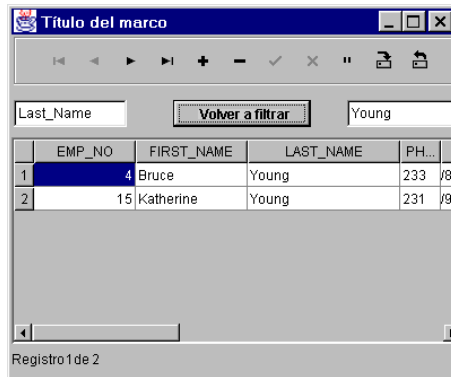
    catch (Exception ex) {
        System.err.println("Filter example failed");
    }
}

```

8 Compilar y ejecutar la aplicación.

La aplicación en ejecución presentará este aspecto:

Figura 11.2 Filtros de ejecución de aplicación



Para probar esta aplicación:

- 1 Escriba el nombre de la columna que desea filtrar (por ejemplo, Last_Name) en el primer `JdbTextField`.
- 2 Introduzca el valor a partir del cual desea filtrar en el segundo `JdbTextField` (por ejemplo, Nelson).
- 3 Haga clic en el `JButton`.

Nota Si se deja en blanco el nombre de columna o el valor, se cancelan los filtros y se pueden ver todos los valores.

Clasificación de datos

Cuando se clasifica un conjunto de datos, se define un índice que permite mostrar los datos ordenados sin tener que reordenar las filas en la tabla del servidor.

Los conjuntos de datos pueden clasificarse por una columna o por varias. Cuando se define una clasificación por varias columnas, el conjunto de datos se clasifica de la siguiente manera:

- Primero por la columna principal.
- La columna secundaria definida en la ordenación resuelve la ordenación cuando se repiten las columnas definidas en la ordenación principal.

- Las restantes columnas de clasificación resuelven también la ordenación cuando aparecen repeticiones en la columna inmediatamente anterior.
- Si sigue habiendo elementos idénticos después de haber clasificado por la última columna definida, se mostrarán en el mismo orden en que aparezcan en la tabla del servidor.

Puede clasificar los datos de cualquier subclase `DataSet`, incluidos los componentes `QueryDataSet`, `ProcedureDataSet`, `TableDataSet` y `DataSetView`. Cuando se clasifican datos en `JBuilder`, observe que:

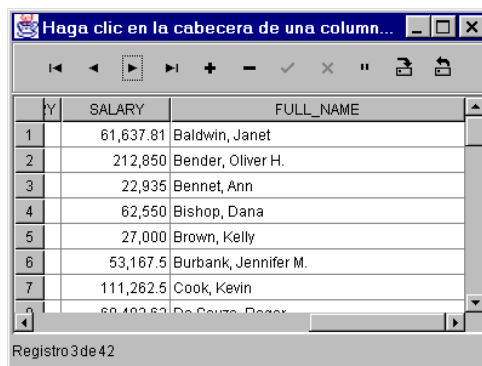
- La distinción entre mayúsculas y minúsculas se aplica sólo cuando se clasifican datos de tipo cadena (`String`).
- La distinción entre mayúsculas y minúsculas se aplica a todas las columnas de cadenas en una clasificación de varias columnas.
- El sentido de clasificación (ascendente o descendente) se fija columna por columna.
- Los valores nulos (`null`) aparecen al principio si la clasificación es descendente, y al final si es ascendente.

La clasificación y la indexación de datos son operaciones muy relacionadas. Consulte [“Clasificación e indexación” en la página 11-12](#) para obtener más información sobre los índices.

Clasificación de datos en una `JdbTable`

Si la aplicación incluye una `JdbTable` que esté asociada a un `DataSet`, se puede clasificar en una única columna de la rejilla haciendo clic en la cabecera de columna en la aplicación en ejecución. Cuando se hace clic se alterna el orden de ascendente a descendente.

Figura 11.3 Haga clic en la cabecera de una columna para ordenar durante la ejecución



Este método sólo permite clasificar por una columna. Al hacer clic en el encabezado de otra columna, el orden de clasificación será sustituido por el de la nueva columna seleccionada.

Clasificación de datos mediante las herramientas de diseño visual de JBuilder

Si se necesita que la aplicación clasifique en un orden específico, las herramientas de diseño visual de JBuilder permiten establecer estas propiedades rápidamente. La propiedad `sort` de `DataSet` proporciona un método sencillo para:

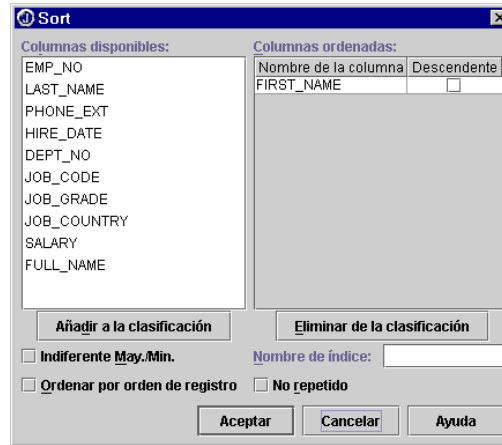
- Ver las columnas que controlan actualmente el orden de clasificación.
- Seleccionar las columnas clasificables en el `DataSet`.
- Añadir columnas seleccionadas a la especificación de clasificación y eliminarlas de ella.
- Establecer la sensibilidad a mayúsculas y minúsculas de la clasificación.
- Fijar el orden de clasificación ascendente o descendente columna por columna.
- Establecer restricciones de clasificación que sólo permitan añadir o actualizar en un `DataSet` columnas que tengan valores clave no repetidos.
- Crear un índice reutilizable para una tabla.

En este ejemplo, se clasifica un conjunto de datos en orden ascendente por apellidos. Para asignar valores a las propiedades de clasificación mediante las herramientas de diseño visual de JBuilder, realice los pasos siguientes:

- 1 Abra o cree el proyecto de [“Suministro de datos de los ejemplos” en la página 11-2](#).
- 2 Haga clic en la pestaña Diseño. Seleccione el `QueryDataSet` en el panel de estructura.
- 3 En el Inspector, seleccione y haga doble clic en el área situada junto a la propiedad `sort`. De esta forma se presenta el editor de la propiedad `sort`.
- 4 Especifique los valores de las opciones que influyan en el orden de clasificación de los datos. En este caso, seleccione el campo `LAST_NAME` en la lista Columnas disponibles y haga clic en Añadir a la clasificación.
- 5 Si se equivoca al seleccionar la columna, haga clic en el botón Eliminar de la clasificación, y repita el paso anterior.

El cuadro de diálogo tendrá un aspecto parecido a éste:

Figura 11.4 Cuadro de diálogo Sort



- 6 Pulse el botón Aceptar.
Los valores de propiedad que se especifican en este cuadro de diálogo se almacenan en el objeto `SortDescriptor`.
- 7 Seleccione Ejecutar | Ejecutar proyecto para compilar y ejecutar la aplicación. Tendrá un aspecto parecido a éste:

Figura 11.5 Aplicación ordenada durante la ejecución

Ordenar los datos por Last_Name

	EMP_NO	FIRST_NAME	LAST_NAME	PH
1	34	Janet	Baldwin	2
2	105	Oliver H.	Bender	255
3	28	Ann	Bennet	5
4	83	Dana	Bishop	290
5	109	Kelly	Brown	202
6	71	Jennifer M.	Burbank	289
7	107	Kevin	Cook	894
8	29	Deena	De Souza	200

Registro 16 de 42

Clasificación e indexación

Existen dos opciones en el cuadro de diálogo Sort que requieren una explicación más detallada: Único y Nombre de índice. La clasificación y la indexación son funciones muy parecidas. A continuación, se describen con más detalle los índices con nombre y los índices sin repeticiones.

Sin repeticiones

Active la opción No repetido para crear un índice único que posibilite una restricción en los datos del `StorageDataSet`. Gracias a esta restricción, sólo

las filas con valores no repetidos para las columnas que se hayan definido como `sortKeys` en el `SortDescriptor` se pueden añadir o actualizar en un `DataSet`.

- La opción Sin repeticiones es una restricción que afecta al conjunto de datos (`DataSet`), y no sólo al índice. Si se define un índice sin repeticiones para una columna, se está afirmando que no existen en el conjunto de datos dos filas que tengan el mismo valor en esa columna. Si en el conjunto de datos hay dos o más filas con el mismo valor en la columna sin repeticiones *cuando se crea el índice por primera vez*, las filas repetidas se pasan a un conjunto de datos de “duplicados”.
- La primera vez que se aplica la propiedad `sort` sin repeticiones (Único), las filas que incumplen esa restricción se copian a un `DataSet`. Se puede acceder a ese `DataSet` llamando al método `StorageDataSet.getDuplicates()`. El `DataSet` de repeticiones puede borrarse llamando al método `StorageDataSet.deleteDuplicates()`.
- En cada momento puede estar definida varias veces la propiedad `sort` sin repeticiones en un mismo `StorageDataSet`. Si existe un `DataSet` de repeticiones procedente de una configuración anterior de la propiedad `sort` sin repeticiones, no se podrán configurar otros valores para la propiedad `sort` sin repeticiones hasta que no se borren los anteriores. Esto se ha diseñado así para evitar que puedan borrarse filas importantes si se configura mal la propiedad `sort` sin repeticiones.
- Si un índice sin repeticiones se clasifica por varias columnas diferentes, la restricción se aplicará a todas las columnas a la vez: dos filas pueden tener el mismo valor en una misma columna de clasificación, pero no puede haber ninguna fila que tenga el mismo valor que otra en todas las columnas de clasificación.
- La opción Único resulta útil para realizar consultas en tablas de un servidor que tenga un índice sin repeticiones. Antes de que el usuario empiece a editar el conjunto de datos, puede definirse un índice sin repeticiones sobre las columnas que están indexadas en el servidor, con la certeza de que no habrá repeticiones. Con ello se garantiza que el usuario no pueda crear filas que se rechazarían como repeticiones al guardar los cambios en el servidor.

Nombre de índice

Introduzca un nombre en este campo para crear un índice con nombres. Este será el nombre definido por el usuario que se asociará a la especificación de clasificación (índice) que se define en el cuadro de diálogo.

- El índice con nombre implementa los órdenes de clasificación (es decir, los índices), e impone la restricción de ausencia de repeticiones, incluso si se deja de ver los datos en ese orden. Decir que el índice *mantiene* los órdenes significa que cada índice se actualiza para reflejar las

inserciones, borrados y modificaciones en sus columnas de clasificación. Por ejemplo, si se define una clasificación sin repeticiones en la columna NumCliente del conjunto de datos de Clientes, y se quiere visualizarlos ordenados por códigos postales, para lo cual habrá que definir una clasificación específica, no se podrá introducir ningún cliente nuevo que tenga un valor NumCliente repetido.

- Los índices con nombre sirven para volver a utilizar una clasificación definida con anterioridad. El índice se mantiene actualizado para que pueda reutilizarse. Si se asigna a la propiedad `sort` de un conjunto de datos un nuevo descriptor de clasificación (`sortDescriptor`) con los mismos parámetros que la clasificación existente, se utilizará esta última.
- Para ver un conjunto de datos en el orden definido por un índice, asigne valores a su propiedad `sort` por medio del constructor `sortDescriptor`, que extrae el nombre del índice.

Clasificación de datos en el código

El código para instanciar un `SortDescriptor` se puede introducir manualmente o se pueden utilizar las herramientas de diseño de JBuilder para generarlo. El código generado automáticamente por las herramientas de diseño de JBuilder presenta un aspecto similar al siguiente:

```
queryDataSet1.setSort(new com.borland.dx.dataset.SortDescriptor("",
    new String[] {"LAST_NAME", "FIRST_NAME", "EMP_NO"}, new boolean[]
    {false, false, false, }, true, false, null));
```

En este segmento de código, el `sortDescriptor` se instancia utilizando como columnas de clasificación primero los campos LAST_NAME y FIRST_NAME y después el campo de número de empleado (EMP_NO), que sirve para resolver el orden en caso de que dos empleados tengan el mismo nombre. La clasificación es ascendente y no distingue entre mayúsculas y minúsculas.

Para volver a visualizar los datos sin clasificar, cierre el conjunto de datos y asigne al método `setSort` el valor `null`, como se muestra a continuación: Los datos se mostrarán en el orden en el que se añadieron a la tabla.

```
queryDataSet1.setSort(null);
```


Localización de datos

Una necesidad básica de toda aplicación de datos es encontrar los datos especificados. En este tema se tratan los dos tipos siguientes de localización:

- Una localización interactiva mediante `JdbNavField`, en la que el usuario puede introducir valores que desee localizar mientras se ejecuta la aplicación.
- Una localización donde los valores de búsqueda son definidos por el programa.

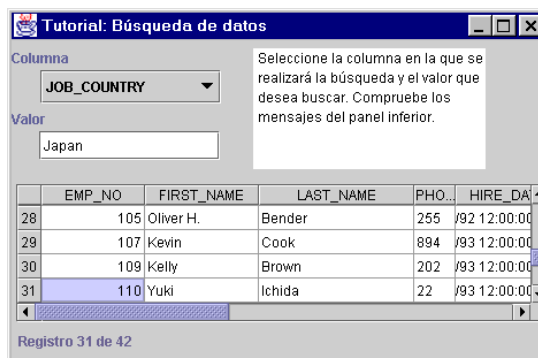
Búsqueda de datos con JdbNavField

La biblioteca `dbSwing` incluye un `JdbNavField` que proporciona funcionalidad de localización en un control de interfaz de usuario. El `JdbNavField` incluye una función de búsqueda incremental para las columnas de tipo `String`. Su propiedad `columnName` especifica en qué columna se lleva a cabo la localización. Si no se define, la localización se efectúa en la última columna visitada del componente `JdbTable`.

Si se incluye un componente `JdbStatusLabel` en la aplicación, aparece `JdbNavField` y se muestran mensajes en la etiqueta de estado.

El directorio `/samples/DataExpress/LocatingData` de la instalación de `JBuilder` incluye un ejemplo finalizado de una aplicación que utiliza el `JdbNavField` bajo el nombre de proyecto `LocatingData.jpx`. Este ejemplo muestra cómo asignar valores a una columna determinada para la operación de búsqueda, así como la utilización del componente `JdbComboBox` que permite al usuario seleccionar la columna en la que se quiere buscar el valor. La aplicación completa tiene el siguiente aspecto:

Figura 11.6 Aplicación de ejemplo con `JdbNavField`



Para crear esta aplicación:

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos. Compruebe la captura de pantalla de la aplicación en ejecución que aparece anteriormente para ver la situación aproximada de los componentes.

- 2 Añada a la interfaz de usuario un `JdbNavField` de la pestaña Más dbSwing de la paleta de componentes, y asigne a la propiedad `dataSet` el valor `queryDataSet1`.
- 3 Añada un componente `JdbComboBox` desde la pestaña Más dbSwing de la paleta de componentes al panel de diseño.
- 4 Asigne a la propiedad `items` del componente `jdbComboBox1` los valores de nombre de columna `EMP_NO`, `FIRST_NAME` y `LAST_NAME`.
- 5 Seleccione la pestaña Sucesos en el Inspector. Seleccione el suceso `itemStateChanged()` del componente `jdbComboBox1` y haga doble clic en su campo de valor. Se añade a la fuente un stub para el suceso `itemStateChanged()`, y el cursor se coloca en el lugar donde se debe insertar el siguiente código, que permite al usuario indicar en qué columna se deben buscar los datos.

```
void jdbComboBox1_itemStateChanged(ItemEvent e) {
    jdbNavField1.setColumnName(jdbComboBox1.getSelectedItem().toString());
    jdbNavField1.requestFocus();
}
```

Este código examina el cambio efectuado en el componente `JdbComboBox`. Si determina que se ha seleccionado un valor de columna distinto, la propiedad `columnName` de `JdbNavField` adopta como valor la columna mencionada en `JdbComboBox`. Esto indica a `JdbNavField` que efectúe localizaciones en la Columna especificada. El foco cambia entonces a `JdbNavField` por lo que se puede introducir el valor que se ha de buscar.

- 6 Seleccione el componente `JdbTextArea` en la pestaña dbSwing.

Coloque el componente `JdbTextArea` junto al componente `JdbComboBox` en el diseñador de interfaces de usuario. En el Inspector, asigne valores a su propiedad `text` de forma que el usuario sepa que debe seleccionar la columna en la que se deben buscar los datos. Por ejemplo, se puede escribir Seleccione la columna en que desea efectuar la búsqueda y escriba el valor buscado. En la barra de estado se facilitan instrucciones para la búsqueda.

Como alternativa, si sólo desea realizar la localización en una Columna determinada, asigne a la propiedad `columnName` del componente

`JdbNavField` el valor de la columna de `DataSet` en la que desee buscar datos (por ejemplo, `LAST_NAME`).

- 7 Añada un componente `JdbLabel` de la pestaña `dbSwing`. Colóquelo junto a `jdbNavField1`. Asigne a su propiedad `text` el valor: Valor buscado.

Nota Para obtener instrucciones adicionales, vea la instantánea de la aplicación ejecutada anteriormente en esta sección.

- 8 Ejecute la aplicación.

La etiqueta de estado se actualiza para adaptarse al estado actual de la aplicación. Por ejemplo:

- Seleccione en `JdbComboBox` el nombre de la columna en la que desea efectuar la búsqueda. En el área de estado se muestra el mensaje Introduzca un valor y pulse *Intro* para comenzar la búsqueda.
- Empiece a escribir el valor que busca en el `JdbNavField`. Si realiza la localización en una columna `String`, observe que, al ir escribiendo, el `JdbNavField` realiza una búsqueda incremental a partir de cada pulsación de tecla. En todos los otros tipos de datos, pulse *Intro* para ejecutar la localización. Si no se encuentra un valor en la tabla, el área de estado pasa a No se encontró ningún valor de columna coincidente.
- Pulse las teclas *Flecha arriba* o *Flecha abajo* para ejecutar una “localización antes de” o “localización después de”, respectivamente. Si aparece una coincidencia, en el área de estado aparece el mensaje Se encontró un valor de columna coincidente. Pulse arriba/abajo para encontrar otros.

Localización de datos mediante la escritura de código

Esta sección explora las bases para localizar datos escribiendo código, así como las condiciones que afectan a la operación de localización.

Cuando localice datos mediante la escritura de código:

- 1 Instancie una `DataRow` basada en el `DataSet` donde desee buscar. Si no desea buscar en todas las columnas del `DataSet`, cree una fila de datos (`DataRow`) “con ámbito” es decir, que sólo contenga las columnas para las que quiera especificar los valores de localización). (Consulte [“Localización de datos mediante una DataRow” en la página 11-18.](#))
- 2 Asigne los valores que hay que localizar en las columnas apropiadas de `DataRow`.
- 3 Llame al método `locate(ReadRow, int)`, especificando las opciones de localización que desee en forma del parámetro `int`. Compruebe el valor devuelto para determinar si la localización tuvo éxito o no.

- 4 Para volver a buscar, llame de nuevo a `locate()` y especifique una opción de localización diferente, como, por ejemplo, `Locate.NEXT` (localizar el siguiente) o `Locate.LAST` (localizar el último). Si desea más información sobre las opciones de `Locate`, consulte el Resumen de campo de la clase `Locate`.

Las funciones centrales de localización utilizan el método `locate(ReadRow, int)`. El primer parámetro, `ReadRow`, es de un tipo de clase abstracta. Normalmente se utiliza su subclase (instanciable), `DataRow`. El segundo parámetro representa la opción de localización y se define en el Resumen de campo de la clase `Locate`. Los campos de la clase `Locate` representan las opciones que permiten controlar dónde empieza la búsqueda y cómo se realiza, por ejemplo si diferencia entre mayúsculas y minúsculas. (Si desea más información sobre las opciones de búsqueda, consulte [“Opciones de localización” en la página 11-19](#)). Si se encuentra una fila coincidente, la posición actual se desplaza a esa fila. Todos los componentes enlazados a datos que se conectan con el mismo `DataSet` localizado se desplazan juntos hasta la fila localizada.

El método `locate()` busca en la vista actual del `DataSet`. Esto significa que las filas excluidas de la visualización por un `RowFilterListener` no se incluyen en la búsqueda.

La vista del `DataSet` puede estar clasificada o no; en el primer caso, el método `locate()` busca las filas coincidentes de acuerdo con la secuencia de clasificación.

Para localizar un valor `null` en una columna dada de un `DataSet`, incluya la columna en el parámetro `DataRow` del método `locate()` pero no le asigne un valor.

Sugerencia Si el método `locate()` no encuentra una coincidencia cuando usted cree que existe, compruebe si hay valores `null` en algunas columnas; recuerde que están incluidas en la búsqueda todas las columnas de la `DataRow`. Para evitar esto, utilice una `DataRow` de ámbito que contenga sólo las columnas deseadas.

Localización de datos mediante una DataRow

Una `DataRow` es similar a un `DataSet` en cuanto a que contiene varios componentes `Column`. Sin embargo, almacena sólo una fila de datos. Los valores qué localizar se especifican en la `DataRow`.

Si se crea la `DataRow` basándose en el mismo `DataSet` localizado, la `DataRow` contiene los mismos nombres de columna y tipos de datos y orden de columnas que el `DataSet` en el que está basada. Por defecto, todas las columnas de la `DataRow` están incluidas en la operación de localización; para excluir columnas, cree una `DataRow` *con ámbito* que sólo contenga columnas especificadas del `DataSet`. Se crea una `DataRow` con ámbito utilizando uno de los siguientes constructores `DataRow`:

- `DataRow(DataSet, String)`
- `DataRow(DataSet, String[])`

Tanto la `DataRow` como el `DataSet` son subclases de `ReadWriteRow`. Ambos heredan los mismos métodos de manipulación de contenidos, por ejemplo, `getInt(String)` y `setInt(String, int)`. Por lo tanto, puede trabajar con objetos `DataRow` utilizando muchos de los mismos métodos que con `DataSet`.

Opciones de localización

La operación de localización se controla mediante opciones. Algunas constantes se encuentran definidas en la clase `com.borland.dx.dataset.Locate`. Puede combinar opciones de localización mediante el operador OR bit a bit; muchas de las combinaciones más útiles ya están definidas como constantes. Cuatro de las opciones de localización (`FIRST`, `NEXT`, `LAST`, y `PRIOR`) determinan la forma en que se realizará la búsqueda en las columnas del `DataSet`. Las opciones `CASE_INSENSITIVE` y `PARTIAL` definen qué se considera un valor de coincidencia. La constante `FAST` influye en la preparación de la operación de localización.

Debe especificar desde dónde comienza la localización y en qué dirección se desplaza a través de las filas del `DataSet`. Elija una de las opciones siguientes:

- `FIRST` comienza en la primera fila, independientemente de la posición actual, y se desplaza hacia abajo.
- `LAST` comienza en la última fila y se desplaza hacia arriba.
- `NEXT` comienza en la posición actual y se desplaza hacia abajo.
- `PRIOR` comienza en la posición actual y se desplaza hacia arriba.

Si una de estas constantes no se especifica para una operación de localización, se ejecuta una `DataSetException` de `NEED_LOCATE_START_OPTION`.

Para buscar todas las filas coincidentes en un `DataSet`, llame una vez al método `locate()` con la opción de localización `FIRST`. Si se encuentra una coincidencia, vuelva a ejecutar la localización con la opción `NEXT_FAST`, llamando al método repetidamente con esta opción de localización hasta que devuelva `false`. La opción de localización `FAST` especifica que los valores de localización no han cambiado, de forma que no es necesario leerlos en `DataRow` otra vez. Para buscar todas las filas coincidentes que empiezan en la parte inferior de la vista, use las opciones `LAST` y `PRIOR_FAST`.

La opción `CASE_INSENSITIVE` especifica que los valores de la cadena se consideran coincidentes aunque difieran en las mayúsculas. Especificar si una operación de localización es o no `CASE_INSENSITIVE` es opcional y sólo tiene significado en las columnas `String`; se hace caso omiso para los demás tipos de datos. Si esta opción se utiliza en una localización multicolumna, la distinción entre mayúsculas y minúsculas se aplica a todas las columnas `String` implicadas en la búsqueda.

La opción `PARTIAL` especifica que un valor de fila se considera coincidente con el correspondiente valor de localización si comienza con el primer carácter del valor de localización. Por ejemplo, puede utilizar como valor de localización la letra “M” para encontrar todos los apellidos que empiecen por esa letra. Al igual que `CASE_INSENSITIVE`, `PARTIAL` es optativo, y sólo tiene sentido en las búsquedas en columnas de cadenas (`Strings`).

Las localizaciones multicolumna que utilizan `PARTIAL` se diferencian de otras localizaciones multicolumna en que el orden de las columnas es importante. El constructor de una `DataRow` multicolumna de ámbito toma una matriz de nombres de columnas. Estos nombres no tienen que aparecer en la lista en el mismo orden en que aparecen en el `DataSet`. Como la opción `PARTIAL` se aplica sólo a la última columna especificada, es importante controlar qué columna es la última en la matriz.

En una operación de localización multicolumna mediante la opción `PARTIAL`, una fila de `DataSet` debe coincidir con los valores de todas las columnas de `DataRow` excepto la última. Si la última columna comienza por el valor de localización, el método es correcto. Si no, el método falla. Si la última columna de `DataRow` no es una columna `String`, el método `locate()` ejecuta una `DataSetException` de `PARTIAL_SEARCH_FOR_STRING`.

Localización para gestionar cualquier tipo de datos

Los datos almacenados en los componentes `DataExpress` se almacenan en objetos `Variant`. Para visualizar los datos se utiliza una representación `String` de la variante. Para escribir el código que lleva a cabo una localización generalizada que gestione las columnas de cualquier tipo de datos, utilice uno de los métodos `setVariant()` y uno de los `getVariant()`.

Por ejemplo, puede decidir escribir una rutina de localización generalizada que acepte un valor y busque en el `DataSet` la fila que lo contiene. Se puede emplear el mismo bloque de código para trabajar con cualquier tipo de datos ya que los datos siguen siendo de tipo variante. Para mostrar los datos utilice la clase de formateador apropiada, o cree una personalizada.

Orden de columnas en DataRow y en DataSet

Mientras que una `Column` del `DataSet` sólo puede aparecer una vez en la `DataRow`, el orden de las columnas puede ser diferente en una `DataRow` con ámbito que en un `DataSet`. En algunas operaciones de localización, el orden de las columnas puede ser importante. Por ejemplo, puede afectar a las localizaciones multicolumna al utilizar la opción `PARTIAL`. Para obtener más información, consulte el párrafo sobre localizaciones multicolumna con la opción `PARTIAL`, [“Opciones de localización” en la página 11-19](#).

Cómo añadir funcionalidad a las aplicaciones de base de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Una vez finalizada la fase de suministro de la aplicación, y con los datos en el componente `DataSet` de un paquete `DataExpress` adecuado, se puede empezar a trabajar con la funcionalidad central de la aplicación y su interfaz de usuario. El capítulo anterior, [Capítulo 11, “Filtrado, clasificación y localización de datos”](#), introducía la clasificación, filtrado y localización de datos en un conjunto de datos. Este capítulo demuestra otras tareas habituales en las aplicaciones de base de datos.

Una característica de diseño del paquete `DataExpress` es que la manipulación de los datos es independiente de su obtención. Sea cual sea el tipo de componentes `DataSet` utilizado en la obtención de los datos, éstos se manipulan y conectan a controles exactamente de la misma forma. Casi todos los ejemplos de este capítulo utilizan el componente `QueryDataSet`, componente que se puede sustituir, sin tener que cambiar el código del cuerpo principal de la aplicación, por `TableDataSet` o por cualquier subclase de `StorageDataSet`.

Todos estos ejemplos han sido creados con el Visualizador de aplicaciones y las herramientas de diseño de JBuilder. Siempre que sea posible, se debe generar el código Java fuente con estas herramientas. Donde sea necesario, se mostrará el código que hay que modificar, su posición y la forma en que la aplicación ejecuta una tarea determinada.

Los ejemplos y tutoriales a los que se hace referencia en este capítulo incluyen acceso SQL a datos almacenados en un `JDataStore` local. En el directorio de ejemplos de `DataExpress` de JBuilder (`/samples/DataExpress`) puede encontrar proyectos finalizados y archivos fuente de Java con comentarios en el archivo fuente si procede. Se recomienda utilizar los ejemplos como guía al añadir estas funciones a la aplicación.

Para crear una aplicación de base de datos, es necesario, en primer lugar, conectarse con una base de datos y suministrar datos a un `DataSet`. Estos pasos para conectarse con una base de datos se describen en [Capítulo 17, “Tutorial: Creación de aplicaciones de base de datos distribuidas”](#). El tutorial configura una consulta que se puede utilizar para cada una de las siguientes tareas de base de datos:

- [“Creación de consultas” en la página 12-2](#) incluye información sobre la creación de consultas mediante una lista de selección.
- [“Utilización de columnas calculadas” en la página 12-8](#).
- [“Totalización de datos con campos calculados” en la página 12-11](#).
- [“Incorporación de una plantilla de edición o visualización para formatear datos” en la página 12-17](#).
- [“Presentación de una vista alternativa de los datos” en la página 12-24](#).
- [“Persistencia de los datos” en la página 12-26](#).
- [“Utilización de tipos de datos variantes” en la página 12-28](#).

Creación de consultas

Una `Column` puede derivar sus valores de:

- Datos de una columna de base de datos.
- Como resultado de la importación de un archivo de texto.
- Como resultado de un cálculo, que puede incluir columnas calculadas, datos agregados, datos que se consultan en otros conjuntos de datos o datos elegidos mediante una lista de selección.

En este apartado se trata el suministro de valores a una columna por medio de listas de selección y la creación de consultas que muestran valores de otras columnas.

Introducción de datos con listas de selección

Se puede utilizar una lista de selección para buscar un valor en una columna de otro conjunto de datos para su introducción. Por ejemplo, este tipo de búsqueda puede mostrar una relación de opciones en una lista desplegable. La utilización de listas de selección puede reducir considerablemente el tiempo necesario para la introducción de datos.

En el siguiente apartado se describe el proceso de creación (mediante las herramientas de diseño de JBuilder) de una lista de selección que puede utilizarse para asignar el valor de una columna de la lista de valores disponible a otro conjunto de datos. En estas instrucciones se recogen los

pasos necesarios para consultar un valor en una lista de selección cuyo fin sea la introducción de datos, en este caso, para seleccionar un país para un cliente o un empleado. En este ejemplo, la propiedad `pickList` de una columna permite definir qué columna de qué conjunto de datos se utiliza para suministrar los valores de la lista de selección. Las opciones de introducción de datos se mostrarán en un componente visual, por ejemplo, en una tabla, cuando se ejecute la aplicación.

Cuando se está ejecutando la aplicación se puede insertar una fila en la tabla, y cuando se introduce un valor para el campo `JOB_COUNTRY` se puede seleccionar en la lista de selección desplegable. El país seleccionado se inserta automáticamente en el campo `JOB_COUNTRY` del conjunto de datos `EMPLOYEE`.

Cómo añadir un campo de lista de selección

En los pasos siguientes se enseña a crear una lista de selección que se puede utilizar para definir el valor de la columna `JOB_COUNTRY` a partir de la lista de países disponibles en la tabla `COUNTRY`. Cuando el usuario selecciona un país en la lista de selección, su selección se escribe automáticamente en el campo actual de la tabla. El proyecto de ejemplo, `Picklist.jpx`, ubicado en el subdirectorio `/samples/DataExpress/Picklist` de la instalación de JBuilder, es una aplicación completa que utiliza la lista de selección que se describe en los pasos siguientes.

- 1 Cree una sencilla aplicación de base de datos, según se describe en [“Suministro de datos de los ejemplos” en la página 11-2](#).

- 2 Añada un componente `QueryDataSet` a la aplicación.

Con ello se formula la consulta para alimentar la lista de opciones.

- 3 Seleccione la propiedad `query` de `queryDataSet2` en el Inspector, pulse el botón de puntos suspensivos (...), para abrir el editor de la propiedad `Query`, y asigne a la propiedad `query` los siguientes valores:

Opción	Valor
Database	<code>databasel</code>
Sentencia SQL	<code>select COUNTRY from COUNTRY</code>

- 4 Haga clic en Probar consulta y, una vez finalizada, pulse Aceptar para cerrar el cuadro de diálogo.
- 5 Amplíe el componente `queryDataSet1` en el árbol de componentes, para poder ver todas las columnas, y seleccione `JOB_COUNTRY`.

- 6 Seleccione la propiedad `pickList` en el Inspector, pulse el botón de puntos suspensivos (...), para abrir el editor de la propiedad `PickList`, y asigne a la propiedad `pickList` los siguientes valores:

Nombre de la propiedad	Valor
Lista de selección/Conjunto de datos de consulta	<code>queryDataSet2</code>
<code>queryDataSet2</code>	<code>COUNTRY</code>
Tipo de datos	<code>STRING</code>
Mostrar en lista de selección	<code>activada</code>
<code>queryDataSet1</code>	<code>JOB_COUNTRY</code>

Pulse Aceptar.

- 7 Pulse sobre la ficha Fuente y escriba el siguiente código después de la llamada a `jbInit()`:

```
queryDataSet2.open();
```

De este modo se abre `queryDataSet2`, asociado a la tabla `EMPLOYEE_PROJECT`. Normalmente, los componentes visuales asociados a datos, como `JdbTable`, abren automáticamente el conjunto de datos pero, en este caso, no hay ningún componente visual asociado al conjunto de datos, por lo que se debe abrir expresamente.

- 8 Ejecute la aplicación seleccionando Ejecutar | Ejecutar proyecto.

Cuando se está ejecutando la aplicación, se puede insertar una fila en la tabla, y cuando se introduce un valor para el campo `JOB_COUNTRY`, se puede seleccionar en la lista de selección desplegable. El país seleccionado se inserta automáticamente en el campo `JOB_COUNTRY` del conjunto de datos `EMPLOYEE`.

Eliminación de un campo de lista de selección

Para eliminar una lista de selección:

- 1 Seleccione en el árbol de componentes la columna que contiene la lista de selección.
- 2 Abra el cuadro de diálogo `pickListDescriptor` haciendo clic en la propiedad `pickList` del Inspector.
- 3 Asigne al campo Lista de selección/Conjunto de datos de consulta el valor `<none>` (ninguno).

Creación de consultas mediante columnas calculadas

En este apartado se trata el uso de campos de consulta para mostrar valores de columnas de otro conjunto de datos.

Este tipo de consulta recupera los valores de una tabla concreta según criterios especificados por el usuario y los muestra como parte de la tabla actual. Para poder crear una columna calculada, se necesita crear un nuevo objeto `Column` en `StorageDataSet`, definir su `calcType` de manera apropiada y codificar el manejador del suceso `calcFields`. Los valores de la consulta solo son visibles cuando la aplicación se está ejecutando. Las columnas de consulta se pueden definir y ver en `JBuilder`, pero las columnas de consulta definidas en `JBuilder` no se resuelven en la fuente de datos, ni se proporcionan desde ella, aunque pueden exportarse a un archivo de texto.

Un ejemplo de una consulta sobre un campo de una tabla diferente para su visualización es consultar el número de una pieza para visualizar una descripción de la pieza o consultar el código postal de una ciudad y un estado concretos.

El método `lookup()` utiliza los criterios de búsqueda especificados para buscar la primera fila coincidente. Al localizar la fila, se devuelven los datos de esa fila, pero el cursor no se desplaza a ella. El método `locate()` es parecido a `lookup()`, pero el cursor se desplaza, de hecho, a la primera fila coincidente. Para obtener más información acerca del método `locate()`, consulte [“Localización de datos” en la página 11-15](#).

El método `lookup()` puede utilizar un `DataRow` con ámbito definido (un `DataRow` con menos columnas que el `DataSet`) para contener los valores que se buscan y las opciones definidas en la clase `Locate` para controlar la búsqueda. Esta `DataRow` con ámbito contiene solo las columnas en que se consulta y los datos coincidentes, si los hubiera. Generalmente se consultan valores de otra tabla, por lo que es necesario instanciar una conexión con ella en la aplicación.

Este ejemplo muestra la utilización de una columna calculada para buscar y recuperar un nombre de empleado (en `EMPLOYEE`) mediante el número de empleado de `EMPLOYEE_PROJECT`. Este tipo de campo de consulta sirve únicamente para visualizaciones. Los datos que esta columna contiene en tiempo de ejecución no se conservan porque ya existen en algún otro lugar de la base de datos. La estructura física de la tabla y de los datos subyacentes al conjunto de datos no cambia. Las columnas calculadas son, por defecto, de sólo lectura. Este proyecto se puede ver como aplicación completa ejecutando el proyecto de ejemplo `Lookup.jpj`, ubicado en el subdirectorio `/samples/DataExpress/Lookup` del directorio de instalación de `JBuilder`.

Para obtener más información sobre la utilización del suceso `calcFields` en la definición de una columna calculada, consulte [“Utilización de columnas calculadas” en la página 12-8](#).

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos.

- 2 Añada un componente `QueryDataSet` a la aplicación.

Esto suministra los datos para llenar la tabla base en la que posteriormente se añadirán las columnas para realizar consultas en otras tablas. Asigne a la propiedad `query` de `queryDataSet2` los siguientes valores:

Para esta opción	Seleccione
Database	<code>database1</code>
Sentencia SQL	<code>select * from EMPLOYEE_PROJECT</code>

- 3 Haga clic en Probar consulta y, una vez finalizada, pulse Aceptar para cerrar el cuadro de diálogo.
- 4 Seleccione el elemento `JdbTable` en el panel de contenido y cambie su propiedad `dataSet` a `queryDataSet2`.

Esto permite ver los datos en el diseñador y en la aplicación en ejecución.

- 5 Pulse el icono de expansión que se encuentra a la izquierda de `queryDataSet2` en el árbol de componentes para poder ver todas las columnas, seleccione `<nueva columna>`, y asigne las siguientes propiedades del Inspector a la nueva columna:

Nombre de la propiedad	Valor
<code>calcType</code>	<code>CALC</code>
<code>title</code>	<code>EMPLOYEE_NAME</code>
<code>columnName</code>	<code>EMPLOYEE_NAME</code>
<code>dataType</code>	<code>STRING</code>

La nueva columna se muestra en la lista de columnas y en el control de tabla. Puede editar manualmente el método `setColumns()` para cambiar la posición de ésta o de otra columna. En la columna de consulta de la tabla del diseñador no se visualiza ningún dato. Las consultas sólo son visibles cuando se ejecuta la aplicación. El tipo de dato `STRING` se utiliza aquí por ser el tipo de dato de la columna `LAST_NAME` que se

especifica más tarde como la columna de consulta. Por defecto, las columnas calculadas son de sólo lectura.

- 6 Seleccione la ficha Eventos del Inspector (se supone que la nueva columna está aún seleccionada en el panel de contenido) y haga doble clic sobre el evento `calcFields`.

El cursor se encuentra en el lugar adecuado del panel de código fuente.

- 7 Escriba el siguiente código, que lleva a cabo la consulta y coloca el valor buscado en la columna que se acaba de definir.

```
void queryDataSet2_calcFields(ReadRow changedRow, DataRow
    calcRow, boolean isPosted) throws DataSetException{
    // Definir un DataRow que contenga el número de empleado que se busca
    // en queryDataSet1, y otro que contenga la fila del empleado
    // datos encontrados.
    DataRow lookupRow = new DataRow(queryDataSet1, "EMP_NO");
    DataRow resultRow = new DataRow(queryDataSet1);

    // El número EMP_NO de la fila actual de queryDataSet2 es
    // el criterio de búsqueda.
    // Se busca la primera coincidencia, ya que EMP_NO puede estar repetido.
    // Si se encuentran resultados, concatenar los campos de nombre
    // de los datos del empleado y poner el resultado en dataRow;
    // de lo contrario, la columna se queda en blanco.

    lookupRow.setShort("EMP_NO", changedRow.getShort("EMP_NO"));
    if (queryDataSet1.lookup(lookupRow, resultRow,
        Locate.FIRST))
    calcRow.setString("EMPLOYEE_NAME",
        resultRow.getString("FIRST_NAME") +
        " " + resultRow.getString("LAST_NAME"));
    }
}
```

- 8 Pulse sobre la ficha Fuente y escriba el siguiente código después de la llamada a `jbInit()`.

```
queryDataSet1.open();
```

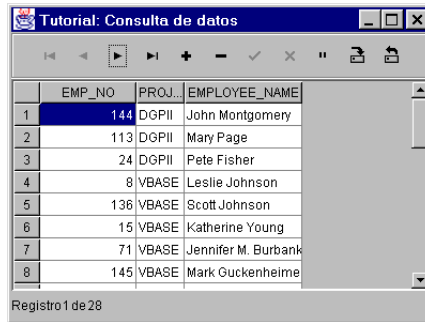
De este modo se abre `queryDataSet1`, asociado a la tabla `EMPLOYEE`.

Normalmente, los componentes visuales asociados a datos, como `JdbTable`, abren automáticamente el conjunto de datos, pero en este caso no hay ningún componente visual asociado al conjunto de datos, por lo que se debe abrir expresamente.

- 9 Elija Ejecutar | Ejecutar proyecto para ejecutar la aplicación.

La aplicación en ejecución tendrá este aspecto:

Figura 12.1 Aplicación de consultas



	EMP_NO	PROJ...	EMPLOYEE_NAME
1	144	DGP11	John Montgomery
2	113	DGP11	Mary Page
3	24	DGP11	Pete Fisher
4	8	VBASE	Leslie Johnson
5	136	VBASE	Scott Johnson
6	15	VBASE	Katherine Young
7	71	VBASE	Jennifer M. Burbank
8	145	VBASE	Mark Guckenheime

Registro 1 de 28

Cuando la aplicación está en ejecución, los valores de la columna de consulta calculada se ajustan automáticamente para cambiar en todas las columnas, en este caso en la columna EMP_NO, citada por la expresión de cálculo. Si el campo EMP_NO está modificado, la consulta visualiza el valor asociado con el actual al enviar el valor.

Utilización de columnas calculadas

Normalmente, una `Column` de un `StorageDataSet` obtiene sus valores de los datos de una columna de la base de datos o de la importación de un archivo de texto. Una columna también puede obtener sus valores de una expresión calculada. JBuilder admite dos tipos de columnas calculadas: calculadas y totalizadas.

Para crear una columna calculada, es necesario crear un nuevo objeto `Column` persistente en `StorageDataSet` y suministrar la expresión al manejador del suceso `calcFields` del objeto `StorageDataSet`. En JBuilder se pueden definir y ver las columnas calculadas. Los valores calculados sólo son visibles en la aplicación en ejecución. Las columnas calculadas definidas en JBuilder no se resuelven en la fuente de datos, ni se proporcionan desde ella, aunque pueden escribirse en un archivo de texto. Si desea más información sobre la manera de definir en el diseñador una columna calculada, consulte [“Creación de una columna calculada en el diseñador” en la página 12-9](#). Para obtener más información sobre las columnas, consulte el [Capítulo 7, “Utilización de columnas”](#).

La fórmula de una columna calculada suele utilizar expresiones que implican a otras columnas del conjunto de datos en la generación del valor de cada fila del conjunto de datos. Por ejemplo, un conjunto de datos puede no tener columnas calculadas para `QUANTITY` y `UNIT_PRICE` y sí para `EXTENDED_PRICE`. `EXTENDED_PRICE` se calcula multiplicando los valores de `QUANTITY` y `UNIT_PRICE`.

Las columnas totalizadas y calculadas se pueden utilizar para agrupar y resumir datos, por ejemplo, para resumir el total de ventas por trimestre. Los cálculos de totalización se pueden especificar totalmente mediante las propiedades y se puede incluir cualquier número de columnas en el grupo. Se admiten cuatro tipos de totalización (suma, cuenta, min y max) así como un mecanismo de creación de métodos de totalización personalizados. Para obtener más información, consulte [“Totalización de datos con campos calculados” en la página 12-11](#).

Las columnas calculadas también son útiles en el mantenimiento de consultas en otras tablas. Por ejemplo, el número de pieza se puede utilizar para recuperar la descripción de la pieza y visualizarla en la línea correspondiente de la factura. Para obtener más información sobre la utilización como campo de consulta de un campo calculado, consulte [“Creación de consultas” en la página 12-2](#).

Los valores de todas las columnas calculadas de una fila se calculan en la misma llamada de suceso.

Estos son los temas tratados:

- [“Creación de una columna calculada en el diseñador” en la página 12-9](#).
- [“Totalización de datos con campos calculados” en la página 12-11](#).
- El `aggDescriptor` en [“Asignación de valores a las propiedades de AggDescriptor” en la página 12-15](#).
- [“Creación de un manejador personalizado de sucesos de totalización” en la página 12-16](#).

Creación de una columna calculada en el diseñador

Este ejemplo se basa en el ejemplo de [“Suministro de datos de los ejemplos” en la página 11-2](#). La tabla de la base de datos en la que se realiza la consulta es EMPLOYEE. La premisa del ejemplo es que la empresa da a todos los empleados un aumento del 10%. Cree una nueva columna llamada NEW_SALARY y una expresión que multiplique el dato SALARY por 1,10 y coloque el valor resultante en la columna NEW_SALARY. El proyecto completo se encuentra disponible en el directorio `/samples/DataExpress/CalculatedColumn` de la instalación de JBuilder, con el nombre `CalculatedColumn.jpr`.

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos.

- En el árbol de componentes, pulse sobre el icono de ampliación situado junto a `queryDataSet1` para mostrar todas las columnas, seleccione `<nueva columna>` y configure las siguientes propiedades en el Inspector:

Nombre de la propiedad	Valor
<code>calcType</code>	<code>CALC</code>
<code>title</code>	<code>NEW_SALARY</code>
<code>columnName</code>	<code>NEW_SALARY</code>
<code>dataType</code>	<code>BIGDECIMAL</code>
<code>currency</code>	<code>true</code>

Si se añade más de una columna, se puede editar manualmente el método `setColumns()` para cambiar la posición de la columna nueva o de cualquier otra columna persistente. En la columna calculada de la tabla del diseñador no se visualiza ningún dato. Los cálculos sólo son visibles cuando se ejecuta la aplicación. El tipo de dato `BIGDECIMAL` se utiliza aquí por ser el tipo de dato de la columna `SALARY` que se utiliza en la expresión de cálculo. Las columnas calculadas siempre son de sólo lectura.

- Selecione el objeto `queryDataSet1`, vaya a la pestaña Sucesos del Inspector, seleccione el manejador de sucesos `calcFields` y haga doble clic en su valor.

De esta manera, se crea en la ventana de código fuente el método vacío para el método del suceso.

- Para calcular el aumento de salario, modifique el método del suceso como sigue:

```
void queryDataSet1_calcFields(ReadRow changedRow, DataRow
calcRow, boolean isPosted) throws DataSetException{
    //calcular el nuevo sueldo
    calcRow.setBigDecimal("NEW_SALARY",
        changedRow.getBigDecimal("SALARY").multiply(new
        BigDecimal(1.1)));
}
```

`calcFields` llama a este método siempre que se guarda el valor de un campo o se envía una fila. Este suceso pasa una entrada que son los valores actuales de la fila (`changedRow`), una fila de salida para poner los cambios que desee efectuar en la fila (`calcRow`) y un booleano (`isPosted`) que indica si la fila se incluye en el `DataSet` o no. Quizá no se desee volver a calcular los campos de las filas que todavía no hayan sido enviadas.

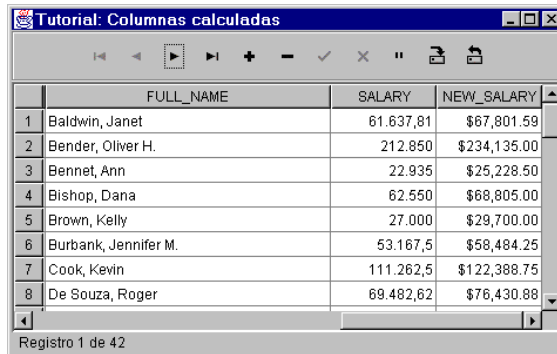
- 5 Para utilizar el tipo de datos `BIGDECIMAL`, es necesario importar la clase `java.math.BigDecimal`. Agregue esta sentencia a las sentencias `import` existentes.

```
import java.math.BigDecimal;
```

- 6 Ejecute la aplicación para ver la expresión de cálculo resultante.

Cuando la aplicación se está ejecutando, los valores de la columna calculada se ajustan automáticamente para cambiar en todas las columnas citadas por la expresión de cálculo. Las columnas `NEW_SALARY` muestran el valor de $(SALARY * 1.10)$. La aplicación en ejecución presentará este aspecto:

Figura 12.2 Columnas calculadas



	FULL_NAME	SALARY	NEW_SALARY
1	Baldwin, Janet	61.637,81	\$67,801.59
2	Bender, Oliver H.	212.850	\$234,135.00
3	Bennet, Ann	22.935	\$25,228.50
4	Bishop, Dana	62.550	\$68,805.00
5	Brown, Kelly	27.000	\$29,700.00
6	Burbank, Jennifer M.	53.167,5	\$58,484.25
7	Cook, Kevin	111.262,5	\$122,388.75
8	De Souza, Roger	69.482,62	\$76,430.88

Totalización de datos con campos calculados

Se puede utilizar la característica de totalización de una columna calculada para resumir de distintas formas los datos. Las columnas con un `calcType` de `aggregated` tiene capacidad para:

- Agrupar y resumir datos para determinar las asociaciones.
- Calcular una suma.
- Contar el número de ocurrencias del valor de un campo.
- Definir un totalizador personalizado que puede utilizarse para definir un método de totalizador propio.

`AggDescriptor` se utiliza para especificar las columnas que se van a agrupar, la columna que se va a totalizar y la operación de totalización que se va a ejecutar. El descriptor `aggDescriptor` se describe con más detalle en los siguientes apartados. La operación de totalización es una instancia de una de las siguientes clases:

- CountAggOperator
 - SumAggOperator
 - MaxAggOperator
 - MinAggOperator
- Una clase de totalización personalizada definida por el programador.

Crear columnas totalizadas calculadas es más sencillo que crear columnas calculadas, ya que no es necesario ningún método de suceso (a menos que se cree un componente de totalización personalizado). Se puede calcular la totalización del conjunto de datos entero o agrupar una o más columnas del conjunto de datos y calcular un valor totalizado para cada grupo. La columna totalizada calculada se define en el conjunto de datos resumido por lo que todas las filas de un grupo tendrán el mismo valor en la columna calculada (el valor totalizado del grupo). La columna está oculta por defecto. Se puede elegir mostrar la columna o mostrar su valor en otro control, esto último es lo que hace este apartado.

Ejemplo: Totalización de datos con campos calculados

En este ejemplo, realizará una consulta en la tabla SALES y creará un componente `JdbTextField` para presentar la suma del campo `TOTAL_VALUE` para el campo `CUST_NO`. Para ello, primero debe crear una columna llamada `GROUP_TOTAL`. A continuación, asigne a la propiedad `calcType` de la columna el valor `aggregated` y cree una expresión que resuma el campo `TOTAL_VALUE` de la tabla `SALES` con un número personalizado y coloque el valor resultante en la columna `GROUP_TOTAL`. El proyecto completo está disponible en el directorio `/samples/DataExpress/Aggregating` de la instalación de JBuilder.

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos.

- 2 Pulse sobre `queryDataSet1` en el árbol de componentes.

Con ello se formula la consulta para alimentar el conjunto de datos con los valores que se han de añadir.

- 3 Abra la propiedad `query` de `queryDataSet1` y modifique la sentencia SQL para darle la siguiente forma:

```
SELECT CUST_NO, PO_NUMBER, SHIP_DATE, TOTAL_VALUE from SALES
```

- 4 Pulse el botón Probar consulta para comprobar la consulta y garantizar su validez; una vez finalizada, pulse Aceptar.

- 5 En el árbol de componentes, pulse sobre el icono de ampliación situado junto a `queryDataSet1`, seleccione **<nueva columna>** y configure las siguientes propiedades en el Inspector:

Nombre de la propiedad	Valor
<code>title</code>	<code>GROUP_TOTAL</code>
<code>columnName</code>	<code>GROUP_TOTAL</code>
<code>currency</code>	<code>True</code>
<code>dataType</code>	<code>BIGDECIMAL</code>
<code>calcType</code>	<code>aggregated</code>
<code>visible</code>	<code>Sí</code>

Se instancia una nueva columna y se añade el código siguiente al método `jbInit()`. Para ver el código, abra la pestaña Fuente. Seleccione la pestaña Diseño para continuar.

```
column1.setCurrency(true);
column1.setCalcType(com.borland.dx.dataset.CalcType.AGGREGATE);
column1.setCaption("GROUP_TOTAL");
column1.setColumnName("GROUP_TOTAL");
column1.setDataType(com.borland.dx.dataset.Variant.BIGDECIMAL);
```

- 6 Añada un `JdbTextField`, desde la ficha `dbSwing` de la paleta de componentes, en el Diseñador de interfaces de usuario, asigne a su propiedad `dataSet` el valor `queryDataSet1` y a la propiedad `columnName` el valor `GROUP_TOTAL`.

Este control muestra los datos totalizados. Es recomendable añadir un `JdbTextArea` para describir qué visualiza el campo de texto.

No se muestran datos en el `JdbTextField` del diseñador. Los cálculos sólo son visibles cuando se ejecuta la aplicación. El tipo de dato `BIGDECIMAL` se utiliza aquí por ser el tipo de dato de la columna `TOTAL_VALUE` que se utiliza en la expresión de cálculo. Las columnas totalizadas siempre son de sólo lectura.

- 7 Seleccione una a una las siguientes columnas y asigne a su propiedad visible el valor `yes`.

- `PO_NUMBER`
- `CUST_NO`
- `SHIP_DATE`

De este modo, se garantiza que las columnas que se muestran en la tabla son persistentes. Las columnas persistentes se encierran entre corchetes en el panel de estructura.

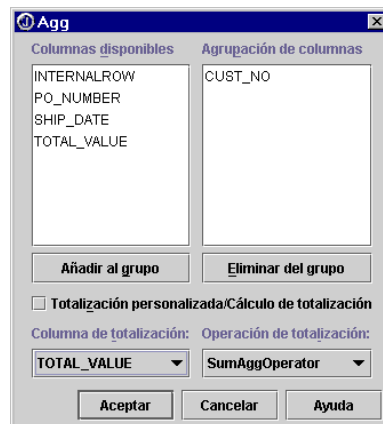
- 8 Seleccione la columna GROUP_TOTAL en el panel de contenido; a continuación, para definir la totalización de esta columna, haga doble clic en la propiedad `agg`, de modo que se abra su editor de propiedades.

En el editor de la propiedad `Agg`:

- a Elija CUST_NO en la lista Columnas disponibles. Pulse Añadir al grupo para seleccionar este campo como el campo que será utilizado para definir el grupo.
- b Seleccione TOTAL_VALUE en la lista Columna de totalización, para seleccionarla como la columna que debe contener los datos que se van a agregar.
- c Seleccione `SumAggOperator` en la lista Operación de totalización, para seleccionarla como la operación que debe realizarse.

Basándose en las selecciones anteriores, ahora tendrá la suma de todas las ventas a un cliente dado.

- 9 Puse Aceptar cuando el editor de la propiedad `agg` presente el siguiente aspecto:

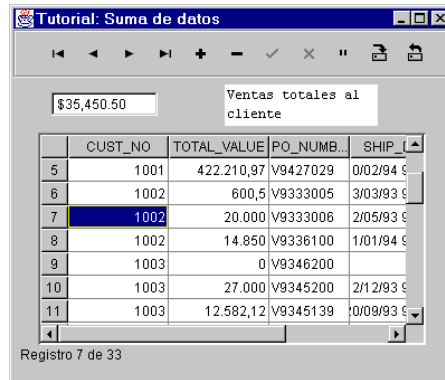


Los pasos anteriores generan el siguiente código fuente en el método `jbInit()`:

```
column1.setAgg(new com.borland.dx.dataset.AggDescriptor(new
    String[] {"CUST_NO"}, "TOTAL_VALUE", new
    com.borland.dx. dataset.SumAggOperator()));
```

- 10 Ejecute la aplicación seleccionando Ejecutar | Ejecutar Proyecto para ver los resultados de la totalización.

La aplicación en ejecución presentará este aspecto:



Cuando la aplicación está en ejecución, los valores del campo totalizado se ajustan automáticamente para cambiar en el campo TOTAL_VALUE. Asimismo, el valor presentado en `JdbTextField` muestra el total de CUST_NO para la fila que está seleccionada.

Asignación de valores a las propiedades de `AggDescriptor`

El editor de la propiedad `agg` ofrece una interfaz sencilla para crear y modificar objetos `AggDescriptor`. El constructor de un objeto `AggDescriptor` requiere la siguiente información:

- Agrupación de columnas: una matriz de columnas (en cualquier orden) que indican el nombre de las columnas utilizadas para definir un subconjunto de filas del `DataSet` en el que debe tener lugar la totalización.
- Columna de totalización: una cadena que representa el nombre de la columna cuyos valores se totalizan.
- Operación de totalización: nombre de un objeto de tipo `AggOperator` que lleva a cabo la operación de totalización.

El editor de la propiedad `agg` extrae posibles nombres de columna para utilizarlos como columnas de agrupación, y los presenta como una lista de columnas disponibles. Sólo están permitidos nombres de columnas no calculadas y no totalizadas en la lista de agrupación de columnas.

Si el `DataSet`, para cuya `Column` se define la propiedad `agg`, tiene un descriptor `MasterLink` (es decir, si el `DataSet` es un `DataSet` de detalle), los nombres de columnas vinculantes por defecto se añaden a la lista de agrupación de columnas cuando se define un nuevo `AggDescriptor`.

Puede utilizar los botones situados debajo de la lista de agrupación de columnas y columnas disponibles para desplazar el nombre de la columna resaltada en la lista que está sobre el botón a la lista opuesta. Asimismo, si

hace doble clic en un nombre de columna de una lista, lo desplazará a la lista opuesta. Los elementos de las dos listas son de sólo lectura. Dado que la ordenación de nombres de columna no tiene significado dentro de un grupo, los nombres de columna siempre se añaden al final de su lista de destino. Están permitidos los grupos vacíos (null).

El control de la opción Columna de totalización contiene una lista de todos los nombres de columnas no totalizadas en el `DataSet` actual, obtenida también de CMT. Aunque el conjunto actual de componentes `AggOperator` suministrado con el paquete `DataExpress` no admite tipos de columnas totalizadas no numéricas, no se han restringido las columnas de la lista a tipos numéricos, dada la posibilidad de que un `AggOperator` personalizado por un usuario pudiera admitir tipos de cadena y de fecha.

El control de la opción Operación de totalización muestra la lista de los objetos `AggOperator` incorporados en el paquete `DataExpress`, así como todos los `AggOperator` definidos por el usuario dentro del mismo contexto de clase que la `Column` del `AggDescriptor`.

Los usuarios que deseen efectuar cálculos con valores totalizados (por ejemplo, la suma de los elementos de línea pedidos, multiplicados por una constante), deben activar la casilla Totalización personalizada/Cálculo de totalización. De esta forma, se desactivan los controles de opción Columna de totalización y Operación de totalización, y se sustituyen sus valores con “null” en el constructor `AggDescriptor`, con lo que se convierte en un tipo calculado de valores totalizados. Si la casilla Totalización personalizada/Cálculo de totalización no está activada, los controles de opción Columna de totalización y Operación de totalización se habilitan.

Creación de un manejador personalizado de sucesos de totalización

Puede utilizar métodos de totalización distintos a los proporcionados en `JBUILDER`, creando un manejador de sucesos de totalización personalizado. Una forma para crear un manejador de sucesos de totalización personalizado consiste en codificar los sucesos `calcAggAdd` y `calcAggDelete` en el diseñador de interfaz. Ambos son sucesos `StorageDataSet` a los que se llama después de notificar a `AggOperator` una operación de actualización.

Una utilización típica de estos sucesos es el cálculo de totales de columnas en una tabla de elementos de línea (como SALES). Los totales se pueden calcular mediante el `SumAggOperator` incorporado. Pueden añadirse más columnas totalizadas cuando a la propiedad `aggOperator` de `AggDescriptor` se le ha asignado el valor `null`. Estas columnas adicionales se pueden utilizar, por ejemplo, para aplicar impuestos, descontar un porcentaje del subtotal, calcular costes de envío y, a continuación, calcular el total final.

Otra forma de crear una clase de totalización personalizada es implementar un componente de operador de totalización personalizado

ampliado a partir de `AggOperator` para implementar métodos abstractos. La ventaja de implementar un componente es poder reutilizarlo en otros componentes `DataSet`. Es recomendable crear clases de totalización para calcular promedios, desviaciones estándar o variaciones.

Incorporación de una plantilla de edición o visualización para formatear datos

Todos los datos almacenados internamente, como números y fechas, se introducen y visualizan como cadenas de texto. *Dar formato* es la conversión de la representación interna en su cadena equivalente. *Analizar sintácticamente* es la conversión de la representación de la cadena en su representación interna. Ambas conversiones están definidas por reglas especificadas por modelos basados en cadenas.

Toda operación de formato y análisis sintáctico de datos en el paquete `dataset` está controlada por la clase `VariantFormatter`, que se define únicamente para cada `Column` de un `DataSet`. Para facilitar la utilización de esta clase, existen propiedades de cadenas correspondientes que, al ser definidas, construyen una `VariantFormatter` para la columna mediante la sintaxis de "modelo" básica definida en las clases `java.text.Format` de JDK.

Existen cuatro tipos distintos de modelos según el tipo de datos del elemento controlado.

- 1 Modelos numéricos
- 2 Modelos de fecha y hora
- 3 Modelos de cadena
- 4 Modelos booleanos

Consulte “String-based patterns (masks)” en la *DataExpress Component Library Reference* para obtener más información sobre los modelos.

Entre las propiedades de nivel de `Column` que utilizan estos modelos basados en cadenas se incluyen las siguientes:

- La propiedad `displayMask`, que define el modelo utilizado para el formato básico de los datos y la introducción de éstos.
- La propiedad `editMask`, que define el modelo utilizado para una introducción de datos más avanzada mediante pulsaciones de teclas (llamado también análisis sintáctico).
- La propiedad `exportDisplayMask`, que define el modelo utilizado para importar y exportar datos a archivos de texto.

Las implementaciones por defecto de `VariantFormatter` para cada `Column` son implementaciones sencillas escritas para ofrecer velocidad. Las columnas que utilicen caracteres de puntuación, como las fechas, utilizan un modelo por defecto derivado del local de la columna. Para redefinir el

formato por defecto, por ejemplo, puntos como separación de millares o comas decimales, especifique el modelo de cadena de la propiedad que desee establecer (`displayMask`, `editMask` o `exportDisplayMask`).

La definición de `displayMask`, `editMask` o `exportDisplayMask` como una cadena vacía o `null` (nula) tiene un significado especial, puesto que selecciona su modelo del `Locale` (idioma) por defecto. Este es el comportamiento por defecto de JBuilder en columnas del tipo `Date`, `Time`, `Timestamp`, `Float`, `Double`, y `BigDecimal`. De esta forma, JBuilder garantiza que una aplicación que utiliza los valores por defecto selecciona automáticamente el formato de visualización apropiado cuando se ejecuta en un local diferente.

Nota Cuando se escriben aplicaciones internacionales que utilizan idiomas distintos de `en_US` (inglés de Estados Unidos), se deben utilizar en los modelos los separadores del estilo de Estados Unidos (por ejemplo, la coma en los millares y el punto en los decimales). Esto permite escribir una aplicación que utilice el mismo conjunto de modelos independientemente de su local de destino. Al utilizar un idioma distinto de `en_US`, JDK traduce estos caracteres a sus equivalentes localizados y visualiza la configuración adecuada. Si desea un ejemplo sobre el uso de modelos en una aplicación internacional, consulte el archivo `IntlDemo.jpx`, que se encuentra en el directorio `/samples/dbSwing/MultiLingual` de la instalación JBuilder.

Para redefinir los formatos por defecto de los valores numéricos y de fecha almacenados en los archivos de idioma, asigne valores a las propiedades `displayMask`, `editMask` o `exportDisplayMask` (según corresponda) en el componente `Column` del `DataSet`.

Las posibilidades de formato proporcionadas por los modelos de cadenas del paquete `DataExpress` son normalmente suficientes para la mayoría de las necesidades de formato. Ante necesidades de formato más específicas, el mecanismo de formato incluye interfaces y clases de propósito general que pueden extenderse para crear clases de formato personalizadas.

Máscaras de visualización

Las máscaras de visualización son modelos de cadena utilizados para dar formato a los datos visualizados en la `Column`, por ejemplo, en una `JdbTable`. Las máscaras de visualización pueden añadir espacios o caracteres especiales dentro del elemento de datos para su visualización.

Las máscaras de visualización se utilizan también para analizar sintácticamente la cadena introducida por el usuario, convirtiéndola en el tipo de datos correctos de la `Column`. Si la sintaxis de los datos introducidos no cumple la especificación de la máscara de visualización, no se podrá abandonar el campo hasta que la introducción de datos sea correcta.

Sugerencia Las cadenas introducidas por el usuario que no se pueden analizar sintácticamente con modelo especificado generan mensajes de validación. Estos mensajes aparecen en el control `JdbStatusLabel` cuando `JdbStatusLabel` y el control de interfaz que visualiza los datos para su edición (por ejemplo, una `JdbTable`) están establecidos en el mismo `DataSet`.

Máscaras de edición

Antes de comenzar la edición, las máscaras de visualización gestionan todas las acciones de aplicación de formato y análisis sintáctico. Las máscaras de edición son modelos de cadena opcionales utilizados para controlar la edición de datos de la `Column` y para analizar sintácticamente los datos pulsación de tecla a pulsación.

Los literales incluidos en la visualización del modelo pueden guardarse opcionalmente con los datos si la `Column` tiene especificada una máscara de edición. Las posiciones de los modelos donde se introducen los caracteres se visualizan por defecto como subrayados (`_`). Cuando se escriben datos en la `Column` con una máscara de edición, los datos introducidos se contrastan, con cada pulsación de tecla, con los caracteres que los modelos permiten en esa posición de la máscara.

Los caracteres no permitidos en una ubicación determinada del modelo no son aceptados y el cursor se desplaza a la siguiente posición sólo cuando se satisface el criterio de dicha ubicación en el modelo.

Utilización de máscaras en la importación y exportación de datos

Cuando se importan datos en un componente `DataExpress`, `JBuilder` busca un archivo `.SCHEMA` (`.schema`) que tenga el mismo nombre que el archivo de datos. Si encuentra uno, tiene prioridad la configuración del archivo `SCHEMA`. Si no lo encuentra, busca en la propiedad `exportDisplayMask` de la columna. Utilice el `exportDisplayMask` para dar formato a los datos que importe.

A menudo, los archivos de datos contienen caracteres con formatos de moneda que no pueden leerse directamente en una columna numérica. Se puede utilizar un modelo de `exportDisplayMask` para leer los valores sin el formato de moneda. Una vez en `JBuilder`, deben establecerse las máscaras de visualización y/o edición para restablecer el formato de moneda (o cualquier otro formato).

Cuando se exportan datos, `JBuilder` utiliza `exportDisplayMask` para dar formato a los datos para la exportación. Al mismo tiempo, crea un archivo `SCHEMA` para que los datos puedan importarse fácilmente en un componente `DataExpress`.

Modelos dependientes del tipo de datos

Las secciones siguientes describen y proporcionan ejemplos de modelos de cadenas para distintos tipos de datos.

Modelos para datos numéricos

Los modelos para datos numéricos se componen de dos partes: la primera especifica el modelo para números positivos (números mayores que 0) y la segunda para números negativos. Las dos están separadas por punto y coma (;). Los símbolos con los que se crean las máscaras numéricas se describen en el apartado “Numeric data patterns” de *DataExpress Component Library Reference*.

Los componentes `Column` numéricos siempre tienen máscaras de presentación y edición. Si no se establecen estas propiedades de forma explícita, se obtienen modelos por defecto mediante el siguiente orden de búsqueda:

- 1 Desde el idioma del componente `Column`.
- 2 Si no está definido un idioma para `Column`, desde el idioma del objeto `DataSet`.
- 3 Si no está definido un idioma para `DataSet`, desde el idioma por defecto del sistema. Los datos numéricos se visualizan por defecto con tres decimales.

Las columnas numéricas permiten cualquier número de dígitos a la izquierda de los decimales; sin embargo, las máscaras limitan este número al especificado en ellas. Para garantizar que se pueden introducir en una `Column` todos los valores válidos, se deben especificar en el modelo dígitos suficientes a la izquierda de la coma decimal.

Además, toda máscara numérica tiene un carácter adicional, situado a la izquierda del elemento de los datos, que almacena el signo del número.

El código que establece la máscara de visualización del primer modelo de la tabla es el siguiente:

```
column1.setDisplayMask(new String("###%"));
```

En la tabla siguiente se explican las especificaciones de modelo de los datos numéricos.

Especificación del modelo	Valor de los datos	Valor formateado	Significado
###%	85	85%	Todos los dígitos son opcionales, los ceros a la izquierda no se visualizan, los valores se dividen por 100 y se muestran como porcentajes.
##,##0.0#^ cc;-##,##0.0#^ cc	500.0 -500.5 004453.3211 -00453.3245	500.0 cc -500,5 cc 4.453,32 cc -453,32 cc	El "0" indica un dígito obligatorio, los ceros no se suprimen. Los números negativos están precedidos del signo menos (-). Junto al valor se visualiza el literal "cc". El cursor se sitúa en el vértice del acento circunflejo (^) y los dígitos se desplazan hacia la izquierda conforme se escriben.
\$\$,###.##;(\$\$,###.##)	4321.1 -123.456	\$4,321.1 (\$123.46)	Todos los dígitos son opcionales, incluye separador de millares, separador decimal y el símbolo de moneda. Los valores negativos aparecen entre paréntesis. Si escribe un signo menos (-) o un paréntesis a la izquierda (), JBuilder encierra el valor entre paréntesis.

Modelos para datos de fecha y hora

Las columnas que contienen datos de fecha, hora y horarios siempre tienen máscaras de visualización y edición. Si no se establecen estas propiedades de forma explícita, se obtienen modelos por defecto mediante el siguiente orden de búsqueda:

- 1 Desde el idioma del componente `Column`.
- 2 Si no está definido un idioma para `Column`, desde el idioma del objeto `DataSet`.
- 3 Si no está definido un idioma para `DataSet`, desde el idioma por defecto del sistema.

Los símbolos con los que se crean las máscaras de fecha, hora, y marca temporal se describen en el apartado "Date, time, and timestamp patterns" de la *DataExpress Component Library Reference*.

Por ejemplo, el código que establece la máscara de edición del primer modelo de la tabla es el siguiente:

```
column1.setDisplayMask(new String("MMM dd, yyyyG"));
```

En la tabla siguiente se explican las especificaciones de modelo de los datos de fecha y hora.

Especificación del modelo	Valor de los datos	Valor formateado	Significado
MM-dd-yyyyG.	Enero 14, 1900 Febrero 2, 1492	Ene 14, 1900AC Feb 02, 1492AD	Devuelve la abreviatura del mes, espacio (literal), dos dígitos para el día, cuatro para el año y el designador de la era.
MM/d/aa H:m	Julio 4, 1776 3:30am Marzo 2, 1776 3:30am	07/4/76 3:30 03/2/92 23:59	Devuelve el número del mes, uno o dos dígitos para el día (según proceda), dos para el año, y la hora y los minutos utilizando el reloj de 24 horas.

Modelos para datos de cadena

Los modelos para dar formato y editar datos de texto son específicos de las clases de DataExpress. Consisten en hasta cuatro partes, separadas por punto y coma, de las cuales sólo la primera es obligatoria. Estas partes son las siguientes:

- 1 El modelo de cadena.
- 2 Si los literales deben almacenarse con los datos o no. Un valor de 1, comportamiento por defecto, indica el almacenamiento de los literales con los datos. Un valor de 0 elimina los literales.
- 3 El carácter que se debe utilizar como indicador vacío. Este carácter indica los espacios a introducir en los datos. Si se omite esta parte, se utiliza el carácter subrayado.
- 4 El carácter que se debe utilizar para reemplazar las posiciones vacías en la salida. Si se omite esta parte de la máscara, las posiciones vacías se eliminan.

Los símbolos de máscaras para utilizar datos de texto se describe en “Text patterns” de la *DataExpress Component Library Reference*.

Por ejemplo, el código que establece las máscaras de visualización y edición del primer modelo de la tabla es el siguiente:

```
column1.setDisplayMask(new String("00000{-9999}"));  
column1.setEditMask(new String("00000{-9999}"));
```

En la tabla siguiente se explican las especificaciones del modelo.

Especificación del modelo	Valor de los datos	Valor formateado	Significado
00000{-9999}	950677394 00043 1540001	95067-7394 00043 00154-0001	Visualiza ceros a la izquierda para los 5 dígitos de la izquierda (obligatorio), también puede incluir una raya literal y 4 dígitos. Utilice este modelo para códigos postales de Estados Unidos.
L0L 0L0	H2A2R9 M1M3W4	H2A 2R9 M1M 3W4	La L especifica cualquier letra A-Z, su introducción es obligatoria. El 0 (cero) especifica cualquier dígito de 0 a 9, su introducción es obligatoria y no permite los signos más (+) y menos (-). Utilice este modelo para códigos postales de Canadá.
{(999)} 000-0000^!;0	4084311000	(408) 431-1000	Un modelo para un número telefónico con código de área opcional entre paréntesis. El (^) sitúa el cursor a la derecha del campo y los datos se desplazan hacia la izquierda conforme se introducen. Para garantizar que los datos se almacenan correctamente de derecha a izquierda, utilice el símbolo ! (Los valores numéricos realizan esto de forma automática.) El cero (0) indica que los literales no se almacenan con los datos.

Modelos para datos booleanos

El componente `BooleanFormat` utiliza un modelo basado en cadena que es útil para utilizar datos que pueden aceptar dos valores, almacenados como `true` o `false`. A los datos que se encuentren dentro de esta categoría se les da formato mediante los valores de cadena que se especifiquen. Este formateador también tiene la capacidad de dar formato a valores `null` o sin asignar.

Por ejemplo, se puede almacenar información sobre el género en una columna de tipo `boolean` pero hacer que JBuilder dé formato al campo para visualizar y aceptar los valores de entrada de “Varón” y “Mujer”, como muestra el código siguiente:

```
column1.setEditMask("Varón;Mujer;");
column1.displayMask("Varón;Mujer;");
```

La tabla siguiente muestra los modelos booleanos válidos y sus efectos de formato:

Especificación del modelo	Formato para los valores verdaderos (true)	Formato para los valores falsos (false)	Formato para los valores nulos (null)
varón;mujer	varón	mujer	(cadena vacía)
T,F,T	T	F	T
Sí,No,Ns/Nc	Sí	No	Ns/Nc
fumador;;	fumador	(cadena vacía)	(cadena vacía)
fumador;no fumador;	fumador	no fumador	(cadena vacía)

Presentación de una vista alternativa de los datos

Se pueden clasificar y filtrar los datos de cualquier `StorageDataSet`; sin embargo, hay situaciones en las que es necesario presentar los datos del `StorageDataSet` mediante la utilización simultánea de distintos tipos de clasificación y condiciones de filtro. El componente `DataSetView` ofrece esta posibilidad.

El componente `DataSetView` permite también un nivel adicional de indirección que proporciona mayor flexibilidad al cambiar los vínculos de los componentes de la interfaz. Si fuera necesario volver a vincular los componentes de la interfaz y hubiera varios, vincúlelos a un `DataSetView` en vez de vincularlos directamente al `StorageDataSet`. Cuando sea necesario volver a vincular, cambie el componente `DataSetView` por el `StorageDataSet` apropiado, con lo que se realiza un único cambio que afecta también a todos los componentes de la interfaz conectados al `DataSetView`.

Para crear un objeto `DataSetView` y definir su propiedad `storageDataSet` como el objeto `StorageDataSet` que contiene los datos que desea ver:

- 1 Cree una aplicación siguiendo las instrucciones de [“Suministro de datos de los ejemplos” en la página 11-2](#).

Este paso le permite conectarse con una base de datos, leer datos de una tabla y visualizar y modificar dichos datos en un componente enlazado a datos.

- 2 Añada un componente `DataSetView` desde la pestaña `DataExpress` al árbol de componentes o al diseñador de interfaces de usuario.
- 3 Asigne a la propiedad `storageDataSet` del componente `DataSetView` el valor `queryDataSet1`.

`DataSetView` se desplaza con independencia de su `StorageDataSet` asociado.

- 4 Añada otros `TableScrollPane` y `JdbTable` al Diseñador de interfaces y, para permitir que los controles se desplacen juntos, asigne a la propiedad `dataSet` de la `JdbTable` el valor `dataSetView1`.
- 5 Compile y ejecute la aplicación.

`DataSetView` muestra los datos de `QueryDataSet`, pero no duplica su almacenamiento. Presenta los datos originales, sin filtrar ni ordenar, en el `QueryDataSet`.

Se puede establecer un filtro y un criterio de clasificación en el componente `DataSetView` que difiera de los del `StorageDataSet` original. La vinculación de un `DataSetView` a un `StorageDataSet` y la definición de nuevos criterios de ordenación o filtrado no tiene efecto sobre los criterios de ordenación y filtrado de `StorageDataSet`.

Para definir criterios de filtrado u ordenación en un `DataSetView`:

- 1 Haga doble clic sobre el archivo Marco en el panel del proyecto y, a continuación, seleccione la pestaña Diseño.
- 2 Seleccione el componente `DataSetView`.
- 3 En la ficha Propiedades del Inspector:
 - a Seleccione la propiedad `sort` para cambiar el orden en que se muestran los registros en el `DataSetView`.
Consulte [“Clasificación de datos” en la página 11-9](#) si desea obtener más información sobre `sortDescriptor`.
 - b Seleccione la propiedad `masterLink` para definir un superconjunto de datos para esta vista.
Consulte el [Capítulo 9, “Establecimiento de una relación maestro-detalle”](#), si desea más información sobre `masterLinkDescriptor`.
- 4 En la página Eventos del Inspector, seleccione el método `filterRow` para ocultar filas temporalmente en la vista `DataSetView`. Si desea más información sobre el filtrado, consulte [“Filtrado de datos” en la página 11-5](#).

Por defecto, es posible editar, eliminar e insertar datos en el `DataSetView`. Cuando realice estas operaciones en el `DataSetView`, también las estará realizando en el `StorageDataSet` con el que esté asociado el `DataSetView`.

- Asigne a la propiedad `enableDelete` el valor `false` para inhabilitar la capacidad del usuario para eliminar datos del `StorageDataSet`.
- Asigne a la propiedad `enableInsert` el valor `false` para inhabilitar la capacidad del usuario para insertar datos en el `StorageDataSet`.
- Asigne a la propiedad `enableUpdate` el valor `false` para inhabilitar la capacidad del usuario para actualizar datos en el `StorageDataSet`.

Persistencia de los datos

Entre el momento en que se desarrolla una aplicación y el momento en que el usuario la ejecuta, pueden producirse muchos cambios en los datos en su fuente. Normalmente se actualizan los datos de la fuente de datos. Pero, lo que es más importante, pueden ocurrir cambios estructurales, que causarían un fallo grave en la aplicación. Si se dan estas condiciones, se puede:

- Permitir que la aplicación falle si se produce un suceso de este tipo. Por ejemplo, el nombre de una columna de una tabla de consulta se cambia en el servidor de base de datos, pero esto no se descubre hasta que se intenta editar en la aplicación la columna de consulta.
- Detener la aplicación y presentar un mensaje de error. Dependiendo de dónde se encuentre la fuente de los datos no disponibles, este planteamiento reduce la posibilidad de que se efectúen actualizaciones parciales en los datos.

Por defecto, las columnas que se muestran en un componente enlazado a datos se determinan en tiempo de ejecución basándose en las columnas que aparecen en el `DataSet`. Si la estructura de los datos de la fuente se ha actualizado y es incompatible con su aplicación, se genera un error de tiempo de ejecución cuando se detecta la situación.

JBuilder admite la persistencia de datos como una alternativa para el tratamiento de las situaciones de este tipo. Utilice esta función si la correcta ejecución de su aplicación depende de que estén disponibles determinadas columnas de datos. Esto asegura que la columna estará disponible y que los datos se mostrarán en el orden especificado. Si la columna fuente de la `Column` persistente cambia o se elimina, se genera una `Excepción` en lugar de un error de tiempo de ejecución.

Conversión de las columnas en persistentes

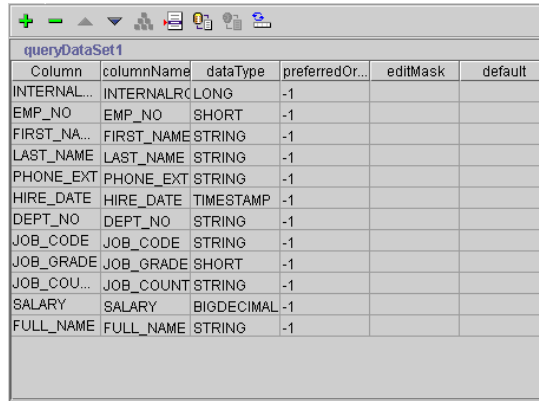
Puede convertir una columna en persistente mediante cualquier propiedad del nivel `Column` (por ejemplo, una máscara de edición). Cuando una columna se ha convertido en persistente, su nombre se encierra entre corchetes ([]).

Para configurar una propiedad a nivel de `Columna`:

- 1 Abra cualquier proyecto que contenga un objeto `DataSet`; elija, por ejemplo, cualquier archivo de proyecto (.jpx) del directorio `/samples/DataExpress/` de la instalación de JBuilder.
- 2 Haga doble clic en el archivo Marco (por ejemplo, `BasicAppFrame.java`) para llevarlo al panel de contenido y pulse la pestaña Diseño.

- Haga doble clic en el objeto `DataSet`. Se mostrará el diseñador de columnas del conjunto de datos, que es parecido a éste cuando se aplica a la tabla de ejemplo de empleados:

Figura 12.3 Diseñador de columnas



Column	columnName	dataType	preferredOr...	editMask	default
INTERNAL...	INTERNALRC	LONG	-1		
EMP_NO	EMP_NO	SHORT	-1		
FIRST_NA...	FIRST_NAME	STRING	-1		
LAST_NAME	LAST_NAME	STRING	-1		
PHONE_EXT	PHONE_EXT	STRING	-1		
HIRE_DATE	HIRE_DATE	TIMESTAMP	-1		
DEPT_NO	DEPT_NO	STRING	-1		
JOB_CODE	JOB_CODE	STRING	-1		
JOB_GRADE	JOB_GRADE	SHORT	-1		
JOB_COU...	JOB_COUNT	STRING	-1		
SALARY	SALARY	BIGDECIMAL	-1		
FULL_NAME	FULL_NAME	STRING	-1		

- Seleccione la `Columna` a cuya propiedad desee asignar valores. El Inspector se actualiza para reflejar las propiedades (y los sucesos) de la columna seleccionada.
- Defina cualquier propiedad, introduciendo un valor en su cuadro de valor del Inspector. Si no desea cambiar ninguna propiedad de la columna, puede fijar un valor y luego restablecer el predeterminado.

Como demostración, defina un valor mínimo para una `columna` que contenga datos numéricos, introduciendo un valor numérico en la propiedad `min`. JBuilder encierra automáticamente el nombre de la columna entre corchetes (`[]`).

En el diseñador de columnas, las columnas de este conjunto de datos se muestran en una tabla del diseñador de interfaces de usuario. Se proporciona una barra de herramientas para añadir, eliminar, desplazarse y reestructurar el conjunto de datos:

- El botón Insertar columna en el conjunto de datos inserta una columna nueva en el ordinal preferido de la columna resaltada en la tabla.
- El botón Eliminar quita la columna del conjunto de datos.
- Los botones Adelantar y Retrasar hacia abajo modifican el ordinal preferido de las columnas y cambian el orden de presentación en los controles enlazados a datos, como, por ejemplo, un control de tabla.
- El botón Seleccione las propiedades que desea mostrar permite elegir las propiedades que se desea visualizar en el diseñador.
- El botón Reestructurar sólo está disponible si la propiedad `store` del conjunto de datos se ha definido como una propiedad `DataStore`. Si

desea más información sobre objetos `DataStore`, consulte [“JDataStore y JBuilder” en la página 2-14](#) o la *Guía del desarrollador de JDataStore*.

El botón Reestructurar compila el componente `this` y ejecuta una MV aparte para realizar una reestructuración del objeto `DataStore` asociado al conjunto de datos. Mientras se ejecuta la reestructuración, se muestra un cuadro de diálogo que presenta su estado y permite cancelarla.

- El botón Conservar todos los metadatos hará eso con todos los metadatos necesarios para abrir un `QueryDataSet` en fase de ejecución. Consulte [“Utilización del diseñador de columnas para convertir metadatos en persistentes” en la página 7-4](#).
- El botón Convertir todos los metadatos en dinámicos permite actualizar las consultas después de que cambie la tabla del servidor. Para poder utilizar nuevos índices creados en la tabla de base de datos, primero debe convertir los metadatos en dinámicos y luego hacer que sean persistentes. El botón Convertir todos los metadatos en dinámicos ELIMINARÁ EL CÓDIGO del archivo fuente. Consulte [“Cómo convertir metadatos en dinámicos con el Diseñador de columnas” en la página 7-5](#).
- Cuando se pulsa el botón Generar clase `RowIterator` se abre un cuadro de diálogo que permite las iteraciones ligeras (que utilizan poca memoria y aceleran la vinculación) que garantiza el acceso seguro a las columnas de tipo estático. Consulte [“El botón Generar clase `RowIterator`” en la página 7-3](#) para obtener más información.

Para cerrar el diseñador de columnas, haga doble clic en cualquier componente de la interfaz de usuario, como `contentPane`, en el panel de estructura, o clic en otro componente, y seleccione Activar diseñador. La única manera de cerrar un diseñador es abriendo otro.

Utilización de tipos de datos variantes

Las columnas pueden contener muchos tipos de datos. En este tema se explica el almacenamiento de objetos Java en una columna. Las columnas se explican de forma más completa en el [Capítulo 7, “Utilización de columnas”](#).

Almacenamiento de objetos Java

`DataSet` y `DataStore` pueden almacenar objetos Java en las columnas de un `DataSet`. Los campos de una tabla SQL, considerados por JDBC como de tipo `java.sql.Types.OTHER`, se mapean en columnas con tipo de datos `Variant.OBJECT`, aunque también puede fijarse el tipo de datos de una columna como `Object` y asignar (set) y obtener (get) valores a través de la API normal del conjunto de datos.

Si se utiliza un `DataStore`, los objetos deben ser serializables. Si no lo son, se genera una excepción cuando el `DataStore` intenta guardar el objeto. Además, la clase debe existir en `CLASSPATH` cuando se intente leer un objeto. En caso contrario, la operación no tendrá éxito.

Tenga en cuenta la siguiente información para dar formato y editar una columna que contiene un objeto Java:

- Formato y edición por defecto.

En el diseñador de interfaces de usuario, se asigna un formateador por defecto a las columnas `Object`. Cuando se edita el objeto, sencillamente es un objeto de tipo `java.lang.String`, independientemente de cuál fuera su tipo original.

- Formato y edición personalizados.

Es conveniente definir la propiedad `formatter` de la columna para redefinir las funciones predeterminadas o, por lo menos, impedir que se puedan efectuar cambios. Puede utilizar un formateador personalizado para definir el formato correcto y analizar los objetos que se mantienen en la columna.

Se utiliza un formateador de columna para todos los registros del conjunto de datos. Esto implica que no es posible mezclar tipos de objetos en una columna determinada. Esta restricción sólo se aplica a la edición personalizada.

Otros controles y sucesos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Aquí se facilita más información sobre el uso de los controles y sucesos. En “[Suministro de datos de los ejemplos](#)” en la [página 11-2](#) se configura una consulta que se puede utilizar como punto de partida para cualquiera de los asuntos que se tratan en este capítulo.

Entre los temas de este capítulo se incluyen:

- “[Sincronización de componentes visuales](#)” en la [página 13-1](#).
- “[Acceso a la información de modelo y datos de un componente de la interfaz de usuario](#)” en la [página 13-2](#).
- “[Visualización de la información de estado](#)” en la [página 13-3](#).
- “[Gestión de errores y excepciones](#)” en la [página 13-5](#).

Sincronización de componentes visuales

Se pueden asociar a un mismo `DataSet` varios componentes enlazados a datos. En tales casos, los componentes se desplazan juntos y, cuando se cambia la posición de fila de un componente, la posición de fila de todos los componentes cambia para compartir el mismo cursor. Esta sincronización de componentes que comparten un mismo `DataSet` facilita enormemente el desarrollo de esta parte de la interfaz de usuario de la aplicación.

El `DataSet` gestiona un “pseudoregistro”, un área de memoria donde se almacenan temporalmente las últimas filas insertadas o los cambios realizados en la fila actual. Los componentes que comparten como fuente de datos el mismo `DataSet`, comparten el mismo “pseudoregistro”. Esto permite que las actualizaciones sean visibles tan pronto como las entradas en los campos finalicen.

La sincronización de varios componentes visuales se realiza asignando a cada una de sus propiedades `dataSet` el mismo conjunto de datos. Cuando los componentes están vinculados al mismo conjunto de datos, se desplazan juntos y permanecen sincronizados automáticamente en la misma fila de datos. Se les denomina *cursores compartidos*.

Por ejemplo: si en su programa utiliza un `JdbNavToolBar` y un `JdbTable`, y los conecta con el mismo `QueryDataSet`, cuando haga clic en el botón “Último” de `JdbNavToolBar` también se mostrará automáticamente el último registro de `QueryDataSet` en el `JdbTable`. Si estos controles se establecen para distintos componentes `dataSet`, no se sitúan automáticamente en la misma fila de datos. Varios de los componentes `dbSwing`, incluidos `JdbNavToolBar` y `JdbStatusLabel`, se autovinculan automáticamente al `DataSet` que tenga el foco.

El método `goToRow(borland.jbcl.dataset.ReadRow)` proporciona una manera de sincronizar dos componentes `DataSet` en la misma fila (a la que `dataSet` está conectado) aunque estén activos diferentes criterios de filtro o de clasificación.

Acceso a la información de modelo y datos de un componente de la interfaz de usuario

Si asigna valores a la propiedad `dataSet` de un componente, evite el acceso a la información de modelo y datos de `DataSet` mediante programa a través del componente, hasta después de crear el par del componente; básicamente, esto quiere decir hasta que el componente se muestra en la interfaz de usuario de la aplicación.

Las operaciones que no devuelven resultados, o cuyos resultados son incorrectos o incoherentes, cuando se ejecutan antes de que el componente se muestre en la interfaz de la aplicación, incluyen cualquier operación que tenga acceso al modelo del componente. Esto puede incluir:

- operaciones `<componente>.get()` o `<componente>.set()`
- `<componente>.insertRow()`
- y demás.

Para asegurar la ejecución satisfactoria de estas operaciones, verifique la notificación del suceso `open()` (abrir) generada por `DataSet`. Cuando se produzca la notificación del suceso, podrá estar seguro de que el componente y su arquitectura de modelo se inicializan correctamente.

Visualización de la información de estado

Muchas aplicaciones de datos proporcionan información sobre el estado de los datos, además de visualizarlos. Por ejemplo, una zona determinada de una ventana contiene a menudo información sobre la posición actual de la fila, los mensajes de error, etc. `dbSwing` incluye un componente `JdbStatusLabel` que proporciona un mecanismo para este tipo de información de estado. Tiene una propiedad `text` que permite asignar una cadena de texto que se visualiza en la `JdbStatusLabel`. Esta cadena sobrescribe los contenidos de la `JdbStatusLabel` y se sobrescribe ella misma cuando se escribe la siguiente cadena en la `JdbStatusLabel`.

El componente `JdbStatusLabel` se conecta automáticamente con el conjunto de datos que tiene el foco. El componente `JdbStatusLabel` no presenta los datos de `DataSet`, sino la siguiente información de estado generada por `DataSet`:

- Posición actual en la fila.
- Número de fila.
- Errores de validación.
- Notificaciones de actualización de datos.
- Mensajes de localización.

Creación de aplicaciones con componentes `JdbStatusLabel`

Este apartado describe el uso de las herramientas de diseño de JBuilder para añadir un componente `dbSwing JdbStatusLabel` a una aplicación.

Para añadir `JdbStatusLabel` a la interfaz de usuario de la aplicación existente, realice lo siguiente:

- 1 Abra los archivos del proyecto de la aplicación a la que desee añadir un `JdbStatusLabel`.

Esta aplicación debe incluir un componente `JdbTable`, un componente `Database` y un componente `QueryDataSet`. Si no cuenta con una aplicación, utilice los archivos creados para [“Suministro de datos de los ejemplos” en la página 11-2](#). Asegúrese de que el diseño de `contentPane` se ha definido en `null`.

- 2 Haga doble clic en el archivo Marco del panel de proyectos del Visualizador de aplicaciones para abrirlo en el panel de contenido y pulse la pestaña Diseño de la parte inferior del Visualizador de aplicaciones.
- 3 Haga clic en la pestaña `dbSwing` de la paleta de componentes y seleccione el botón del componente `JdbStatusLabel`.

- 4 Coloque `JdbStatusLabel` bajo el componente `JdbTable`. El componente `jdbStatusLabel1` se conecta automáticamente al objeto `DataSet` que tenga el foco.

Normalmente, los componentes `JdbStatusLabel` se utilizan en combinación con otros componentes de la interfaz, normalmente con componentes `JdbTable` que muestran los datos del conjunto de datos. Con ello, se configuran ambos componentes de manera que se muevan para rastrear juntos el mismo `DataSet` y se suelen conocer como cursores compartidos.

Cuando se añade `JdbStatusLabel`, observará que el control `JdbStatusLabel` presenta información indicando que el cursor está en la Fila 1 de x (en donde x es el número de registros del `DataSet`).

- 5 Haga doble clic en `QueryDataSet`.
Aparecerá el Diseñador de columnas.
- 6 Seleccione las columnas `LAST_NAME` y `FIRST_NAME` y, en el Inspector, asigne a la propiedad `required` el valor `true` para las dos.
- 7 Asigne a la propiedad `min` de la columna `SALARY` el valor 25000.
- 8 Ejecute la aplicación.

Ejecución de la aplicación `JdbStatusLabel`

Cuando se ejecuta la aplicación, se observa que, al desplazarse por el conjunto de datos, el indicador de fila se actualiza para reflejar la posición de la fila actual. De forma similar, cuando se añaden o borran filas de datos, el número de filas también se actualiza simultáneamente.

Para comprobar la visualización de la información de validación, realice los pasos siguientes:

- 1 Inserte una nueva fila de datos. Intente enviar esta fila sin introducir ningún valor en las columnas `FIRST_NAME` y `LAST_NAME`. Se visualizará un mensaje en la `JdbStatusLabel` para indicar que no se puede enviar la fila, debido a que los valores de los campos no son válidos o han desaparecido.
- 2 Introduzca valores para las columnas `FIRST_NAME` y `LAST_NAME`. Introduzca un número en la columna `SALARY` (1000) que no llegue al valor mínimo. Cuando se sale de la fila, `JdbStatusLabel` visualiza el mismo mensaje de que no se puede enviar la fila debido a que los valores de los campos no son válidos o han desaparecido.

Asignando el texto de `JdbStatusLabel` en puntos relevantes del programa, se sobrescribe el mensaje que en ese momento esté mostrando `JdbStatusLabel` con el texto especificado. Este mensaje de texto se sobrescribe, a su vez, cuando se establece el siguiente texto o cuando se

genera el siguiente mensaje de estado de `DataSet`. El mensaje de estado es el resultado del desplazamiento por los datos de la tabla, los errores de validación al editar los datos, etc.

Gestión de errores y excepciones

Con el uso mediante programación de clases `DataExpress`, la mayor parte de la gestión de errores se realiza a través de extensiones `DataExpress` de la clase `java.lang.Exception`. Todas las clases de excepción `DataSet` son del tipo `DataSetException` y sus subclases.

La clase `DataSetException` puede tener encadenados otros tipos de excepciones (por ejemplo, `java.io.IOException` y `java.sql.SQLException`). En estos casos el método `DataSetException` tiene un mensaje apropiado que describe el error desde la perspectiva de una API de nivel superior. Puede utilizar el método `getExceptionChain()` de `DataSetException` para obtener las excepciones encadenadas. Las excepciones encadenadas (una única lista vinculada) no son `DataSetException` situadas en una API de nivel inferior.

El paquete `dataset` tiene incorporada la gestión `DataSetException` para componentes enlazados a datos `dbSwing`. Los controles como tales no conocen lo que es una `DataSetException`. Simplemente, trabajan con las actualizaciones de datos y las operaciones de acceso, dejando la gestión de los errores a la `DataSetException` incorporada.

En componentes enlazados a datos `dbSwing`, la gestión de errores `DataSetException` por defecto funciona de la siguiente forma:

- Si un control realiza una operación que origina una `DataSetException`, se presenta un cuadro de diálogo `Excepción` con el mensaje del error. Este cuadro de diálogo tiene un botón `Detalles` que visualiza el rastreo de la pila.
- Si `DataSetException` tiene excepciones encadenadas, éstas pueden verse en el cuadro de diálogo `Excepción` mediante los botones `Previa` y `Siguiente`.
- Si la excepción que se lanza es `ValidationException` (una subclase de `DataSetException`), el cuadro de diálogo `Excepción` sólo se abre si no hay monitores `StatusEvent` en el `DataSet` (por ejemplo, el control `JdbStatusLabel`). Una restricción errónea genera una `ValidationException`, por ejemplo, un valor máximo o mínimo fuera de rango, o datos que no sigan el formato de la máscara, un intento de actualizar una columna de sólo lectura, etc. Si un control `JdbStatusLabel` está asociado a un `DataSet`, se convierte automáticamente en un monitor `StatusEvent`. Esto permite al usuario ver los mensajes de las restricciones erróneas en la etiqueta de estado.

Redefinición de la gestión de los controles por defecto de `DataSetException`

Se puede redefinir parte de la gestión de errores por defecto registrando un monitor `StatusEvent` con `DataSet`. Esto impide que se visualicen los mensajes `ValidationException` en el cuadro de diálogo Excepciones.

La gestión por defecto de `DataSetException` para los controles puede desactivarse también en el nivel `DataSet`, si se asigna a su propiedad `displayErrors` el valor `false`. Como se trata de una propiedad del nivel `DataSet`, es necesario establecerla en cada `DataSet` de la aplicación, para desactivar realmente la gestión de errores por defecto de todos los objetos `DataSet` de dicha aplicación.

Para controlar completamente la gestión de `DataSetException` para todos los controles `dbSwing` y objetos `DataSet`, cree su propia clase manejador y conéctela con el monitor `ExceptionEvent` de la clase `DataSetException`.

La mayoría de los sucesos del paquete `dataset` ejecutan una `DataSetException`. Esto es muy útil cuando los manejadores de sucesos utilizan las API de `dataset` (que normalmente lanzan `DataSetException`). Esto evita tener que codificar la lógica `try/catch` de cada manejador de sucesos que se escriba. Actualmente, las herramientas de diseño de `JBuilder` no insertan la cláusula “throws `DataSetException`” en el código fuente java que generan; si lo desea, puede añadirla manualmente.

Creación de una aplicación de base de datos distribuida con DataSetData

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

El proyecto de ejemplo `DataSetData.jpx`, que se encuentra en el subdirectorio `/samples/DataExpress/StreamableDataSets` del directorio de instalación de JBuilder, contiene una aplicación de base de datos distribuida completa que utiliza la RMI (Llamada a métodos remotos) de Java y `DataSetData`. Incluye una aplicación de servidor que obtiene sus datos de la tabla de ejemplo `employee` de `JDataStore` y los envía a través de RMI en forma de `DataSetData`. Un `DataSetData` se utiliza para pasar datos como argumentos al Método RMI, o como flujo de entrada a una servlet Java.

Una aplicación cliente se comunica con el servidor mediante un *Provider* (proveedor) personalizado y un *Resolver* (almacenador) personalizado. La aplicación cliente muestra los datos en una tabla. La modificación efectuada en el cliente puede guardarse mediante un botón Guardar de `JdbNavToolBar`.

Si desea más información sobre la creación de proveedores personalizados, consulte [“Creación de un proveedor de datos personalizado” en la página 6-9](#). Si desea más información sobre la creación o personalización de almacenadores, consulte [“Personalización de la lógica del almacenador por defecto” en la página 8-17](#).

Si desea información actualizada sobre esta aplicación de ejemplo, consulte el archivo `DataSetData.html` del directorio `/samples/DataExpress/StreamableDataSets/`.

Ejemplo de aplicación de base de datos distribuida (con RMI Java y DataSetData)

En el proyecto de ejemplo, que se encuentra en `/samples/DataExpress/StreamableDataSets/DataSetData.jpx`, se demuestra la utilización de la clase `DataSetData` de `DataExpress` en la creación de aplicaciones distribuidas de base de datos. Además de utilizar objetos `DataSetData` para transmitir datos de base de datos entre un servidor RMI y un cliente, en este ejemplo se explica la utilización de un `DataSet` (conjunto de datos), un `Provider` (proveedor) y un `Resolver` (almacenador) personalizados. La aplicación de ejemplo contiene los siguientes archivos:

- Archivos de interfaz

`EmployeeApi.java` es una interfaz que define los métodos que queremos tener en el extremo remoto.

- Archivos de servidor

`DataServerApp.java` es servidor RMI que se deriva de `UnicastRemoteObject`.

- Archivos de proveedor

`ClientProvider.java` es una implementación de un proveedor (`Provider`). El método `provideData` es una implementación de un método de `com.borland.dx.dataset.Provider`. Se ha de buscar el servicio “`DataServerApp`” en el host que se especifica en la propiedad `hostName`, para después hacer que el método remoto llame y cargue nuestro `DataSet` con los contenidos.

- Archivos almacenadores (resolvers).

`ClientResolver.java` es una implementación de un almacenador (`Resolver`). El método `resolveData` es una implementación de `com.borland.dx.dataset.Resolver`. En primer lugar, se ha de buscar el servicio “`DataServerApp`” en el host que se especifica en la propiedad `hostName`. y, a continuación, extraemos los cambios a una instancia de `DataSetData`. Por último, realizamos la llamada al método remoto, atendemos los errores que puedan haberse producido en el almacenador, y cambiamos los bits de estado de todas las filas modificadas que han de ser resueltas.

- Archivos de cliente.

`ClientApp.java` es una aplicación cliente RMI. Para obtener más información, consulte `ClientFrame.java`.

- Otros archivos.

`Res.java` es un archivo de recursos para internacionalizar la aplicación.

`ClientFrame.java` es el marco de `ClientApp`. El `DataSet` mostrado en la rejilla es un `TableDataSet` con un proveedor y un almacenador personalizados. Si desea más información, consulte `ClientProvider.java` y `ClientResolver.java`.

`DataSetServerFrame.java` es el marco mostrado por `DataSetServerApp`.

Configuración del ejemplo de aplicación

Para ejecutar el ejemplo de aplicación:

- 1 Abra esta aplicación en JBuilder. Para ello, seleccione Archivo | Abrir y desplácese hasta `/samples/DataExpress/StreamableDataSets/DataSetData.jpx`.
- 2 Seleccione Proyecto | Propiedades de proyecto para poder ver las propiedades del proyecto. Defina las siguientes opciones:
 - Abra la pestaña Ejecutar.
 - Compruebe que la propiedad `"java.rmi.server.codebase"`, pasada a la máquina virtual del servidor RMI por medio de un argumento de línea de comandos, señala al lugar adecuado de las clases del servidor RMI (`"file:/usr/local/<jbuilder>/samples/DataExpress/StreamableDataSets/classes/"` por defecto).
 - Compruebe que la propiedad `"java.security.policy"` señala al archivo `SampleRMI.policy`, incluido en este proyecto (`"file:/usr/local/<jbuilder>/samples/DataExpress/StreamableDataSets/SampleRMI.policy"` por defecto).
 - Cierre el cuadro de diálogo Propiedades de proyecto.
- 3 Inicie el registro de RMI seleccionando Herramientas | Registro RMI desde JBuilder. El registro se puede activar y desactivar desde el menú Herramientas.
- 4 Seleccione el archivo `DataSetServerApp` en el panel de proyecto. Haga clic con el botón derecho del ratón y seleccione Ejecutar para iniciar el servidor RMI.
- 5 Seleccione el archivo `ClientApp` en el panel de proyecto. Haga clic con el botón derecho del ratón y elija Ejecutar para iniciar el cliente RMI.
- 6 Modifique los datos de la tabla `ClientApp` y pulse el botón Guardar cambios (para volver a guardar los cambios en el servidor) o Actualizar (para volver a cargar los datos desde el servidor). Cada vez que se guardan o se actualizan datos, aumenta el contador de solicitudes de nivel medio.

¿Qué ocurre?

Estos pasos permiten al `DataServerApp` registrarse automáticamente como servidor de RMI. `DataServerApp` responde a dos solicitudes de clientes RMI: `provideEmployeeData` y `resolveEmployeeChanges`, definidas en la interfaz remota `RMI EmployeeApi.java`.

El archivo `ClientApp` es un marco con un objeto `JdbTable` y una barra de herramientas `JdbNavToolBar` para la presentación de datos en conjuntos de datos de `DataExpress`. A través del proveedor personalizado, `ClientProvider.java`, se suministran datos al `DataSet` y, a través de un almacenador personalizado, `ClientResolver.java`, se guardan los datos en el origen. `ClientProvider.java` rellena sus datos de tabla llamando al método remoto `provideEmployeeData()` de `DataServerApp` vía RMI. A continuación, `DataServerApp` consulta datos de una tabla de un servidor de base de datos JDBC y los pasa a un conjunto de datos. Después, extrae los datos del conjunto de datos en un objeto `DataSetData` y los vuelve a enviar a `ClientProvider` por medio de RMI. `ClientProvider` carga los datos del objeto `DataSetData` en el conjunto de datos de `ClientApp`, y los datos aparecen en la tabla.

Cuando llega el momento de almacenar en la base de datos los cambios efectuados en la tabla, `ClientResolver.java`, el “almacenador personalizado” del conjunto de datos `ClientApp` extrae al objeto `DataSetData` los cambios que es necesario enviar al servidor de base de datos. `ClientResolver` llama entonces al método remoto `resolveEmployeeChanges()` de `DataServerApp` mediante RMI, y le pasa como parámetro el objeto `DataSetData` que contiene las actualizaciones necesarias.

`DataServerApp` utiliza `DataExpress` para almacenar los cambios en el servidor de base de datos. Si se produce un error (debido, por ejemplo, a una transgresión de las reglas empresariales o una restricción de datos), las filas de los paquetes `DataServerApp` que no se han podido guardar en la base de datos en un objeto `DataSetData` vuelven a `ClientResolver`. `ClientResolver` extrae las filas que no se pueden almacenar en el objeto `DataSetData` a la tabla de `ClientApp`, lo que permite que las filas problemáticas se conecten y vuelvan a almacenarse en el servidor.

`DataServerApp` es el nivel intermedio de la aplicación. Puede aplicar sus propias reglas empresariales y restricciones entre el servidor y el cliente de base de datos. Por supuesto, puede proporcionar una cantidad indeterminada de métodos adicionales a los que se puede acceder de forma remota para la implementación de lógica empresarial y tareas relacionadas con las aplicaciones.

Paso de metadatos mediante `DataSetData`

Los metadatos que se pasan en un objeto `DataSetData` son muy limitados. Solamente se pasan las siguientes propiedades de `Columna`:

- `columnName`

- `dataType`
- `precision`
- `scale`
- `hidden`
- `rowId`

Las propiedades de columna restantes que un servidor necesite pasar a una aplicación cliente, deberán entregarse como una matriz de Columnas, a través de RMI. El objeto `Column` en sí es serializable, por lo que una aplicación cliente podría diseñarse para obtener estas propiedades de columna antes de necesitar los datos. Las columnas deberán añadirse como persistentes antes de cargar el `DataSetData`.

Distribución de aplicaciones en tres niveles

Para distribuir la aplicación en varios niveles:

- 1 Seleccione `DataServerApp.java` en el panel de proyecto. Modifique la URL de conexión con base de datos del constructor de forma que señale a una conexión con una base de datos remota a la que se tenga acceso. La base de datos constituye el tercer y último nivel.
- 2 Para recompilar y actualizar el archivo `DataServerApp.class`, seleccione Proyecto | Ejecutar Make del proyecto.
- 3 Distribuya `DataServerApp.class` a un equipo remoto al que esté conectado. `DataServerApp` se ejecuta en el segundo nivel (el intermedio).
- 4 Inicie el registro de RMI en el equipo de nivel intermedio.
- 5 Inicie `DataServerApp` en el nivel intermedio.

Nota A partir de JDK 1.2 es necesario autorizar al servidor RMI derechos de seguridad especiales para que espere y acepte solicitudes de RMI de clientes de red. Normalmente, estos derechos se establecen en un archivo de seguridad Java definido mediante una propiedad especial, `java.security.policy`, que se pasa por medio de un argumento de línea de comandos a la máquina virtual del servidor. Esto es similar a la propiedad `java.rmi.server.codebase`, que también debe pasarse a la MV del servidor. Este proyecto incluye el archivo `SampleRMI.policy` como archivo de muestra de la política de seguridad de RMI.

Cuando inicie `DataServerApp` en el nivel intermedio, no olvide que las propiedades `java.security.policy` y `java.rmi.server.codebase` están definidas en los lugares adecuados del equipo de nivel intermedio.

- 6 Haga doble clic en `ClientFrame.java`, en el panel de proyectos de JBuilder, para llevarlo al panel de contenido. Seleccione la pestaña Diseño para abrir el diseñador. En el árbol de componentes, seleccione `clientProvider1` y asigne a la propiedad `hostName` el nombre del servidor del equipo de nivel intermedio.

- 7 Seleccione `clientResolver1` y asigne a la propiedad `hostName` el nombre del servidor del equipo de nivel intermedio.
- 8 Seleccione Proyecto | Ejecutar Make del proyecto para volver a generar `ClientApp`.

Inicie `ClientApp` en el cliente (primer nivel). Para ello, haga clic con el botón derecho del ratón en el archivo `ClientApp.java` del panel de proyecto y seleccione Ejecutar.

Para más información

- Consulte la documentación de RMI en el sitio web de Sun en <http://java.sun.com/j2se/1.4/docs/guide/rmi/>.
- Encontrará más información sobre la escritura de *proveedores* y *almacenadores* personalizados en la aplicación de conjunto de datos de ejemplo `/samples/DataExpress/CustomProviderResolver/CustomProviderResolver.jpx`.

Tareas de administración de bases de datos

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

En este capítulo se explica cómo realizar las tareas normales de administración de bases de datos. Se explican los siguientes temas:

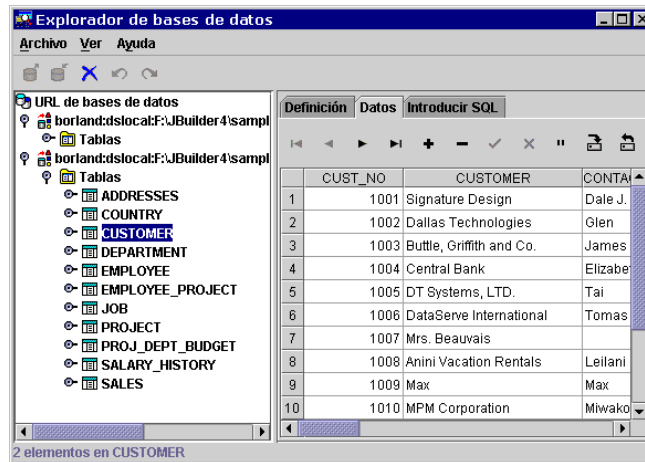
- “Explorar tablas de bases de datos y metadatos con el Explorador de bases de datos” en la página 15-1.
- “Uso del Explorador de bases de datos para las tareas de administración” en la página 15-8.
- “Seguimiento de conexiones de base de datos” en la página 15-11.

Explorar tablas de bases de datos y metadatos con el Explorador de bases de datos

El Explorador de bases de datos es un visualizador de base de datos jerárquicas, que permite modificar los datos. Muestra información basada en JDBC sobre los metadatos de la base de datos, dentro de una ventana con dos paneles. El panel de la izquierda muestra un árbol que representa jerárquicamente un conjunto de bases de datos y sus tablas, vistas, procedimientos almacenados y metadatos asociados. El panel derecho muestra en varias fichas información descriptiva acerca de cada nodo del árbol. En algunos casos, los datos del panel derecho pueden editarse también.

Abra el Explorador de bases de datos seleccionando Explorador de bases de datos en el menú Herramientas de JBuilder.

Figura 15.1 Explorador de bases de datos



A través de una conexión persistente con una base de datos, el Explorador de bases de datos permite:

- Inspeccionar los objetos esquema de la base de datos, lo que incluye las tablas, los datos de las tablas, las columnas (campos), índices, claves primarias, claves externas, definiciones de procedimientos almacenados y parámetros de dichos procedimientos.
- Ver, crear y modificar direcciones URL de la base de datos.
- Introducir y ejecutar sentencias SQL con las que realizar consultas sobre una base de datos.
- Crear, ver y editar datos de tablas ya existentes.

Inspección de objetos esquema de una base de datos

La ventana del Explorador de bases de datos contiene un menú, una barra de herramientas, una barra de estado y dos paneles de información sobre la base de datos.

- El panel de la izquierda muestra un árbol jerárquico de objetos que contiene direcciones URL de base de datos, tablas (y sus columnas, claves primarias y externas), vistas, tablas del sistema y procedimientos almacenados (con sus parámetros).

Un icono ampliado junto a un objeto del panel izquierdo indica que ese objeto contiene otros objetos dependientes de él. Para ver dichos objetos, haga clic sobre el icono ampliado. Cuando un objeto está expandido, mostrando sus objetos dependientes, el icono ampliado se convierte en un icono contraído. Para ocultar los objetos dependientes, haga clic sobre el glifo contraído.

- El panel de la derecha contiene varias fichas con pestañas que muestran el contenido de los objetos que aparecen resaltados en el panel izquierdo. Las fichas con pestañas del panel izquierdo varían en función del tipo de objeto que esté resaltado en el panel izquierdo. Por ejemplo, cuando lo que está resaltado en el panel izquierdo es un alias de base de datos, en el panel derecho aparece una ficha de definición que contiene la dirección URL de la base de datos, el controlador, el nombre de usuario y otros parámetros o propiedades. Si el nombre de un parámetro aparece en negrita, significa que ese parámetro no se puede modificar. Todos los demás parámetros que aparecen en el panel derecho pueden editarse desde ahí. En el panel derecho pueden aparecer las siguientes fichas, con sus respectivas pestañas:
 - Definición
 - Introducir SQL
 - Resumen
 - Datos

Si desea más información, ejecute el Explorador de bases de datos, seleccionando Herramientas | Explorador de bases de datos, y consulte su ayuda en pantalla *Explorador de bases de datos*.

Configuración de controladores para acceder a bases de datos locales y remotas

El Explorador de bases de datos permiten recorrer las bases de datos que aparecen en el apartado Histórico de conexiones a las URL del archivo `<home>/.jdatastore/databasepilot.properties`. Cuando se establece una conexión con una base de datos utilizando el editor de la propiedad `connection` de un componente `Database`, se añaden entradas a esta lista.

El Explorador de bases de datos puede utilizarse para ver, crear y modificar direcciones URL de base de datos. En el proceso siguiente se supone que la dirección URL está cerrada, y se muestra una lista de las tareas que hay que realizar, describiendo brevemente los pasos necesarios para ejecutarla.

- Ver URL
 - a Seleccione en el panel izquierdo la URL que desee ver. En el panel derecho aparece la ficha Definición.
 - b Pulse el icono ampliado situado junto a una dirección URL de una base de datos (o haga doble clic sobre ella) en el panel izquierdo para ver su contenido.
- Crear una URL
 - a Seleccione una URL o base de datos en el panel de la izquierda.
 - b Haga clic con el botón derecho para abrir el menú de contexto.

- c Elija Nuevo (o seleccione Archivo | Nuevo en el menú).
- d Seleccione un controlador en la lista desplegable o introduzca la información de ese controlador. Para poder utilizarlos, los controladores deben estar instalados y los archivos de los controladores deben figurar en la sentencia CLASSPATH en el script de configuración de JBuilder
- e Acceda hasta la dirección URL deseada o bien escríbala directamente.
- f En la ficha Definiciones del panel derecho, especifique UserName y las propiedades deseadas.
- g Haga clic en el botón Aplicar de la barra de herramientas para actualizar los parámetros de la conexión.
- Modificar una URL
 - a Seleccione la URL que desee ver en el panel izquierdo. En el panel derecho aparece la ficha Definiciones.
 - b Introduzca las modificaciones que desee en los valores de la ficha Definiciones.
 - c Haga clic en el botón Aplicar de la barra de herramientas para actualizar los parámetros de la conexión.
- Borrar una URL
 - a Seleccione la tabla que desee ver en el panel izquierdo.
 - b Seleccione Archivo | Borrar en el menú, para eliminar la URL.

Nota Si está creando una nueva dirección URL ODBC y está ejecutando Windows NT, para poder conectarse a esa base de datos deberá definir su fuente de datos ODBC a través del Panel de control de Windows.

Ejecución de sentencias SQL

La ficha Introducir SQL abre una ventana en la que se pueden introducir sentencias SQL o se puede especificar y ejecutar un archivo .SQL. La parte principal de la ventana es un cuadro de edición en el cual se introducen las sentencias SQL. A la derecha de este cuadro hay tres botones, el botón Ejecutar, el botón Siguiente y el botón Anterior. Cuando se ejecuta una sentencia SELECT SQL, los resultados de la consulta aparecen en una tabla editable, situada debajo del cuadro de edición. Quizás necesite redimensionar la ventana para ver todos sus componentes. La ficha tendrá este aspecto:

Figura 15.2 Ficha Introducir SQL del Explorador de bases de datos



Para consultar una base de datos mediante una sentencia SQL:

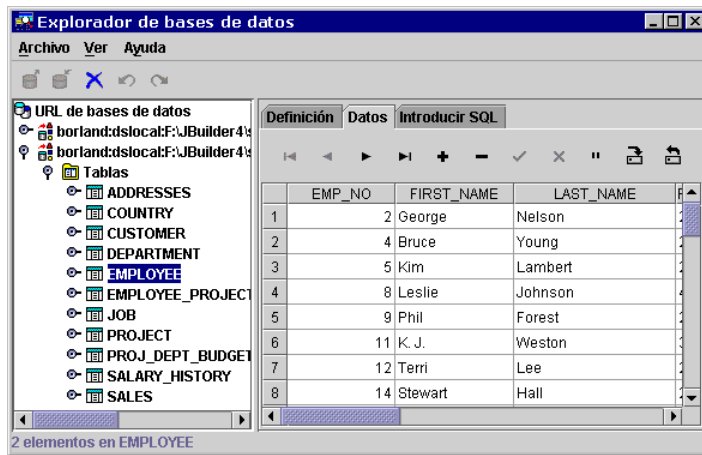
- 1 Abra una base de datos seleccionando su URL en el panel izquierdo e introduciendo el nombre de usuario y la contraseña si es necesario.
- 2 Seleccione la base de datos o uno de sus nodos dependientes, en el panel izquierdo.
- 3 Haga clic en la pestaña Introducir SQL del panel derecho, con lo cual aparecerá un cuadro de edición en el que podrá escribir o seleccionar una sentencia SQL.
- 4 Introduzca (o pegue) una sentencia SQL en el cuadro de edición o haga clic sobre el botón Cargar SQL e introduzca el nombre de un archivo SQL. Si introduce sentencias que no incluyan la cláusula SELECT, se ejecutarán pero no se devolverá ningún conjunto de resultados.
- 5 Pulse el botón Ejecutar para ejecutar la consulta.

Es posible copiar sentencias SQL de un archivo de texto, de una ventana de ayuda o de otras aplicaciones, y pegarlas en el cuadro de edición. En algunos servidores SQL es necesario que el nombre de la tabla se introduzca entre comillas.

Nota Si la sintaxis de la sentencia SQL introducida es incorrecta, se mostrará un mensaje de error. El campo Introducir SQL puede modificarse como se desee para corregir errores sintácticos.

Utilización del Explorador para ver y editar los datos de la tabla

Seleccione la ficha Datos para mostrar los datos de una tabla, vista o sinónimo que haya seleccionado. Es posible introducir registros en una tabla que aparezca en la ficha Datos, así como modificar sus registros, siempre y cuando la tabla permita acceso de escritura y esté seleccionada la casilla Solicitar consultas reales de la ficha Consulta del menú Ver | Opciones. En la ficha Datos aparece una tabla con datos procedentes de la tabla seleccionada. Encima de la tabla aparece una barra de herramientas que permite recorrer los datos y modificarlos. Éste es el aspecto de la ficha de datos:



Puede utilizar el Explorador de bases de datos para ver, editar, insertar y eliminar datos de tablas. En la lista siguiente se indican las operaciones necesarias y se describen brevemente los pasos que deben realizarse para ejecutar cada una de ellas.

- Ver los datos de una tabla.
 - a Seleccione la tabla que desee ver en el panel izquierdo.
 - b Haga clic en la pestaña Datos del panel derecho para ver una tabla de todos los datos de la tabla.
 - c Utilice los botones de la barra de herramientas situados encima de la tabla para recorrer los registros de uno en uno.
- Editar un registro.
 - a Asegúrese de que esté seleccionada la casilla Solicitar consultas reales en el menú Ver | Opciones.
 - b Edite los campos de los registros existentes en la tabla.

- c** Para enviar el registro insertado al conjunto de datos local, seleccione un registro de la tabla o pulse el botón Guardar de la barra de herramientas.
 - d** Para cancelar una modificación antes de pasar a otro registro, pulse el botón Cancelar de la barra de herramientas o la tecla *ESC*.
 - e** Para guardar los cambios en la base de datos, haga clic en el botón Guardar cambios.
- Insertar un nuevo registro.
 - a** Sitúe el cursor en la fila posterior a donde desea insertar la nueva fila.
 - b** Pulse el botón Insertar de la barra de herramientas. Aparecerá una fila en blanco.
 - c** Introduzca los datos de cada columna. Desplácese entre las columnas con el ratón o pulsando la tecla Tab para ir al siguiente campo.
 - d** Para registrar en la base de datos los datos insertados, seleccione otro registro de la tabla o bien pulse el botón Guardar de la barra de herramientas.
 - e** Para cancelar una inserción antes de pasar a otro registro, pulse el botón Cancelar de la barra de herramientas o la tecla *Esc*.
 - f** Para guardar un cambio en la base de datos, haga clic en el botón Guardar cambios.
- Borrar un registro.
 - a** Coloque el cursor sobre la fila que desee borrar.
 - b** Pulse el botón Borrar de la barra de herramientas.

Las modificaciones sólo surtirán efecto una vez aplicadas. Para aplicar las modificaciones y convertir los cambios en permanentes:

- 1** Haga clic en el botón Registrar de la barra de herramientas. Esto registra los cambios solamente en el conjunto de datos local (no en la base de datos).
- 2** Pulse el botón Guardar cambios para guardar las modificaciones en la base de datos.

Uso del Explorador de bases de datos para las tareas de administración

En esta sección se ofrece una introducción de la creación, alimentación y eliminación de tablas mediante SQL. Estas tareas normalmente están reservadas para un administrador de bases de datos, pero pueden realizarse con gran facilidad en JBuilder.

Creación de la fuente de datos SQL

JBuilder es un entorno de desarrollo de aplicaciones en el que se pueden crear aplicaciones que accedan a bases de datos, pero no incluye opciones de menú de características que creen tablas de servidor SQL.

Normalmente, es una opción reservada a un ABD (Administrador de bases de datos). Sin embargo, se pueden crear tablas fácilmente mediante SQL y el Explorador de bases de datos.

Este tema no intenta ser un tutorial de lenguaje SQL, sino mostrar cómo se pueden utilizar sentencias SQL en JBuilder. Para obtener más información sobre la sintaxis SQL, consulte cualquier manual sobre el tema. Una referencia a la que se recurre con frecuencia es *A Guide to the SQL Standard* de C.J. Date.

Nota En muchos sistemas, el administrador (ABD) limita los derechos de creación de tablas únicamente a usuarios autorizados. Si no consigue crear una tabla, póngase en contacto con el administrador y cerciórese de que tiene derecho a realizar la operación.

Para crear una tabla simple, primero deberá configurar una dirección URL con la que conectarse a la base de datos. Si no está familiarizado con este procedimiento, siga estas instrucciones:

- 1 Seleccione Herramientas | Explorador de bases de datos.
- 2 Desde el Explorador de bases de datos, seleccione Archivo | Nuevo o haga clic en el botón derecho sobre una dirección URL existente y seleccione Nuevo en el menú de contexto. Aparecerá el cuadro de diálogo Nueva URL.
- 3 Seleccione un controlador en la lista desplegable o introduzca la información de ese controlador. Consulte Controladores JDBC de bases de datos en la ayuda en pantalla del Explorador de bases de datos para ver una descripción de los distintos tipos de controladores existentes.
- 4 Acceda hasta la dirección URL deseada o escríbala directamente. Cuando en el campo Controlador esté seleccionado un controlador de bases de datos reconocido por JBuilder, el botón puntos suspensivos estará activado.

- 5 Pulse Aceptar para cerrar el cuadro de diálogo.
- 6 En la ficha Definiciones del panel derecho, especifique el nombre de usuario (UserName) y todas las propiedades que desee.
- 7 Haga clic en el botón Aplicar de la barra de herramientas para actualizar los parámetros de la conexión.

Una vez establecida una conexión, puede especificar una sentencia SQL para ejecutarla sobre la base de datos. Existen dos formas de hacer esto. La primera es a través del cuadro de diálogo Crear tabla. Para crear la tabla MiTabla utilizando el cuadro de diálogo Crear tabla:

- 1 Seleccione Archivo | Crear tabla en el Explorador de bases de datos.
- 2 Escriba MiTabla en el campo Nombre de la tabla.
- 3 Pulse el botón Añadir.
- 4 Escriba Apellido en la columna Nombre de columna.
- 5 Seleccione VARCHAR como valor en la columna Tipos de datos.
- 6 Escriba 20 en la columna Decimales.
- 7 Haga clic en el botón Fila siguiente. Se crea un fila.
- 8 Escriba Nombre en la columna Nombre de columna.
- 9 Seleccione VARCHAR como valor en la columna Tipos de datos.
- 10 Escriba 20 en la columna Decimales.
- 11 Haga clic en el botón Fila siguiente. Se crea un fila.
- 12 Escriba salario en la columna Nombre de columna.
- 13 Seleccione NUMERIC como valor de la columna Tipo de datos.
- 14 Escriba 10 en la columna Decimales.
- 15 Escriba 2 en la columna Escala.
- 16 Haga clic en el botón Ejecutar.
- 17 Observe que se ha creado una sentencia SQL en el área de texto SQL.
- 18 Pulse Aceptar. La tabla se crea en la fuente de datos actual.

La segunda manera de crear una tabla es especificar una sentencia SQL `CREATE TABLE` en la pestaña Introducir SQL. Por ejemplo, para crear MiTabla2 en la fuente de datos a la que esté conectado:

- 1 Haga clic sobre la pestaña Introducir SQL del Explorador de bases de datos.
- 2 Introduzca lo siguiente en el área de texto.

```
create table mytable2 (
  lastName char(20),
```

```
firstName char(20),  
salary numeric(10,2) )
```

3 Haga clic en el botón Ejecutar.

Ya ha finalizado la creación de una tabla vacía que se puede utilizar en una consulta. Verifique, mediante el Explorador de bases de datos, que la tabla se creó correctamente. Esto es lo que aparecerá:

- Una lista de tablas en la fuente de datos, incluida la recién creada (MITABLA).
- Una lista de columnas de la tabla seleccionada. Seleccione MITABLA y la lista de columnas visualiza FIRSTNAME, LASTNAME y SALARY.

Alimentación de una tabla SQL con datos mediante JBuilder

Una vez creada una tabla vacía, puede rellenarla fácilmente con datos por medio del Explorador de bases de datos (en este ejemplo) o creando una aplicación con las herramientas de diseño visual de JBuilder. Seleccione la ficha Datos para mostrar los datos de una tabla, vista o sinónimo que haya seleccionado. Es posible modificar o introducir registros en una tabla de la ficha Datos del Explorador de bases de datos, siempre y cuando la tabla permita acceso de escritura y esté activada la casilla Solicitar consultas reales en el cuadro de diálogo Ver | Opciones. En la ficha Datos aparece una tabla con datos procedentes de la tabla seleccionada.

- 1 Siga las instrucciones de [“Creación de la fuente de datos SQL” en la página 15-8](#).
- 2 Seleccione la tabla que acaba de crear en la ventana izquierda y, a continuación, la pestaña Datos de la ventana derecha. En el panel derecho aparecerá una tabla con datos procedentes de la tabla seleccionada. Encima de la tabla aparece una barra de herramientas que permite recorrer los datos y modificarlos.
- 3 Puede utilizar el Explorador de bases de datos para ver, editar, insertar y eliminar datos de tablas. Si necesita más información sobre estas tareas, consulte [“Utilización del Explorador para ver y editar los datos de la tabla” en la página 15-6](#).

Eliminación de tablas en JBuilder

Una vez creada una o más tablas de comprobación, es necesario aprender a limpiarlas y eliminarlas. Siga las instrucciones de [“Creación de la fuente de datos SQL” en la página 15-8](#), pero sustituya la sentencia SQL por la siguiente:

```
drop table mitabla
```

Puede comprobar si la operación se ha realizado correctamente examinando si la tabla sigue apareciendo en la ventana izquierda del Explorador de bases de datos.

Seguimiento de conexiones de base de datos

JBuilder proporciona una clase de seguimiento de JDBC que puede realizar un seguimiento del tráfico de JDBC. JBuilder ofrece una interfaz de usuario, que se abre mediante Herramientas | Monitor JDBC, para trabajar con esta clase durante el diseño. Si desea más información sobre el uso de esta clase en la ejecución, consulte [“Utilización del Monitor JDBC en una aplicación en ejecución” en la página 15-12.](#)

EL Monitor JDBC realiza un seguimiento de cualquier controlador JDBC (es decir, cualquier subclase de `java.sql.Driver`) mientras lo utiliza JBuilder. El monitor de JDBC permite observar todos los datos que salen directamente del controlador JDBC.

Acerca del Monitor JDBC

Para iniciar el monitor JDBC, seleccione Herramientas | Monitor JDBC. Aparecerá la ventana de este módulo:

Figura 15.3 Monitor JDBC



En el Monitor JDBC, se pueden llevar a cabo las siguientes acciones:

- Pulse el botón Cerrar de la ventana del monitor JDBC, para que desaparezca su ventana.
- Para seleccionar texto en el área de registro, resáltelo con el ratón o con el teclado.
- Para guardar el texto seleccionado (o todo el texto, si no hay nada seleccionado) en un archivo, pulse el botón Guardar en archivo.

- Para borrar el texto seleccionado (o todo el texto, si no hay nada seleccionado), pulse el botón Borrar histórico.
- Haga clic en la casilla Activar la salida del histórico para habilitar o deshabilitar el envío de datos al archivo histórico.
- Pulse el botón Tamaño del histórico para definir la cantidad máxima de información de registro que se debe conservar (8 K por defecto).
- Con el cursor sobre el área de texto, pulse *F1* o el botón Ayuda para abrir la ayuda del Monitor JDBC. La ayuda sólo aparece en el modo de diseño.

Utilización del Monitor JDBC en una aplicación en ejecución

Para llevar un seguimiento de las conexiones de bases de datos en tiempo de ejecución, se debe incluir en la aplicación un `MonitorButton` o un `MonitorPanel`. `MonitorButton` es un `JavaBean` que permite ejecutar el monitor JDBC en cualquier aplicación que se esté ejecutando. Para ello, la instancia del monitor JDBC en uso debe ser abierta por la aplicación. Una instancia del monitor JDBC abierta desde el IDE sólo hace seguimiento de las actividades de la base de datos durante la fase de diseño. Si pulsa el botón Monitor, aparecerá un cuadro de diálogo con el Monitor JDBC.

Puede utilizar `MonitorPanel` para poner el monitor directamente en una ficha. Tiene las mismas propiedades que `MonitorButton`.

Incorporación de `MonitorButton` a la paleta

Puede poner `MonitorButton` en la paleta de componentes siguiendo estos pasos:

- 1 Seleccione Herramientas | Configurar paleta.
- 2 Seleccione DataExpress en el campo Hojas de la pestaña Hojas.
- 3 Abra la pestaña Añadir componentes.
- 4 Seleccione JBCL en el campo Seleccionar biblioteca.
- 5 Escoja la ficha de paleta en la que desee colocar el `MonitorPanel`.
- 6 Seleccione No Filtrar.
- 7 Pulse Añadir desde la biblioteca
- 8 Visualice `com.borland.jbcl.sql.monitor.MonitorButton`.
- 9 Pulse Aceptar para cerrar el cuadro de diálogo Buscar clase.
- 10 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de la paleta.

Utilización de la clase `MonitorButton` en el código

Cuando se añade `MonitorButton` a la paleta, es posible soltarlo en una aplicación. También puede añadir una instancia de `MonitorButton` en código, como sigue:

```
MonitorButton monitorButton1 = new  
    com.borland.jbcl.sql.monitor.MonitorButton();  
this.add(monitorButton1);
```

Propiedades `MonitorButton`

Las siguientes propiedades de componente están disponibles en `MonitorButton` para controlar el estado por defecto del monitor:

Propiedad	Efecto
<code>outputEnabled</code>	Activa o desactiva el seguimiento del controlador.
<code>maxLogSize</code>	Tamaño máximo del registro de seguimiento. El valor por defecto es 8K.

Tutorial: Importación y exportación de datos desde un archivo de texto

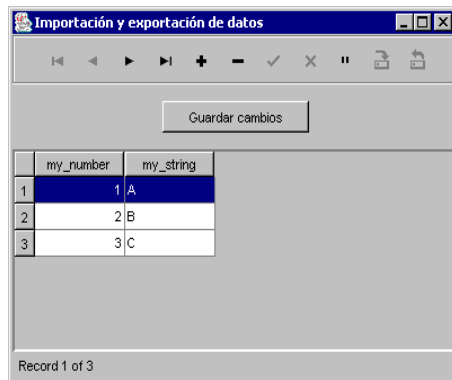
El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

En este tutorial se muestra cómo proporcionar datos a una aplicación por medio de un componente `TableDataSet` y un archivo de datos de texto. En este tutorial, el archivo de datos de texto se crea manualmente, pero este tipo de archivo también se puede exportar desde la mayoría de las bases de datos para ordenadores personales. En este tutorial se realizarán las siguientes tareas:

- Crear un proyecto con JBuilder.
- Crear un archivo de datos de texto sencillo.
- Generar una aplicación.
- Añadir componentes `DataExpress` para consultar y almacenar datos del archivo de texto.
- Añadir componentes `dbSwing` para crear una interfaz de usuario.
- Añadir un componente `Swing JButton` para exportar datos.
- Compilar y ejecutar la aplicación.
- Utilizar modelos para exportar campos numéricos, de fecha y hora y de texto.

Cuando se finalice el tutorial, la aplicación debe tener un aspecto similar a este:

Figura 16.1 Aplicación de importación y exportación de base de datos



La aplicación finalizada se encuentra en el archivo de proyecto de ejemplo `TextFileImportExport.jpx`, situado en el directorio `<jbuilder>/samples/DataExpress/TextFileImportExport/`. Los usuarios con acceso de sólo lectura a los ejemplos de JBuilder deben copiar el directorio `samples` en un directorio con permiso de lectura y escritura.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-8](#).

Paso 1: Creación de un proyecto

Para desarrollar una aplicación de base de datos en JBuilder, es necesario primero crear un proyecto. Para ello:

- 1 Elija Archivo | Nuevo proyecto para iniciar el Asistente para proyectos.
- 2 Escriba `TextFileImportExport` en el campo Nombre.
- 3 Asegúrese de que la opción Generar archivo de notas del proyecto se encuentra seleccionada.
- 4 Pulse Finalizar para cerrar el Asistente para proyectos y crearlo. No se necesita hacer ningún cambio en los valores por defecto en los Pasos 2 y 3 del asistente.

El archivo de proyecto `TextFileImportExport.jpx` y el archivo HTML del proyecto se muestran en el panel del proyecto.

Paso 2: Crear el archivo de texto

Ahora, creará un archivo de datos de texto para importar datos a la aplicación de base de datos. Cree un archivo de texto según se describe en los siguientes pasos:

- 1 Cree un nuevo archivo de texto con el nombre `ImportTest.txt` en el directorio que contiene el archivo de proyecto, `TextFileImportExport.jpx`, creado en el paso anterior.

Seleccione Archivo | Archivo nuevo para abrir el cuadro de diálogo Crear archivo. En el campo Nombre, escriba `ImportTest`. Seleccione `txt` en la lista desplegable Tipo. Asegúrese de que en el campo Directorio se encuentra el directorio que contiene el archivo `TextFileImportExport.jpx`.

- 2 Introduzca las dos filas siguientes y dos columnas de datos (una de valores enteros y otra de valores de tipo cadena) en el nuevo archivo de texto.

Pulse *Intro* al final de cada file. Introduzca las comillas además de los datos.

```
1, "A"  
2, "B"  
3, "C"
```

- 3 Guarde y cierre el archivo.
- 4 Añada el archivo al proyecto (Proyecto | Añadir archivos/paquetes).
Desplácese hasta el archivo en la pestaña Explorador del cuadro de diálogo Añadir archivos o paquetes al proyecto, selecciónelo y pulse Aceptar para añadirlo al proyecto.

Paso 3: Generación de una aplicación

El Asistente para aplicaciones crea archivos de código fuente Java que se añaden al proyecto que se acaba de crear.

Para generar los archivos fuente de una aplicación con el Asistente para aplicaciones, siga estas instrucciones:

- 1 Abra la galería de objetos seleccionando Archivo | Nuevo.
- 2 Seleccione la pestaña General y haga doble clic en el icono Aplicación para que se abra el Asistente para aplicaciones.
- 3 Acepte los valores por defecto del Paso 1 del Asistente para aplicaciones y pulse Finalizar.

Los nuevos archivos de código fuente Java se añaden al proyecto y se muestran como nodos en el panel del proyecto. El código fuente del archivo `Marcol.java` aparece abierto en el panel de contenido.

- 4 Seleccione Archivo | Guardar todo para guardar los archivos de código fuente y el archivo de proyecto.

Paso 4: Incorporación de componentes DataExpress a la aplicación

El diseñador de interfaces de usuario se utiliza para añadir componentes DataExpress a `Marcol.java`.

- 1 Abra la pestaña Diseño para `Marcol.java` en el panel de contenido.
- 2 Seleccione un componente `TextDataFile` de la pestaña DataExpress de la paleta de componentes y haga clic en el árbol de componentes o en el diseñador de interfaces de usuario para añadir el componente a la aplicación.



El nuevo componente `TextDataFile`, `textDataFile1`, aparece como un nodo en el árbol de componentes del panel de estructura.

- 3 Seleccione las siguientes propiedades en el Inspector y asígneles los valores indicados:

Para configurar la propiedad `fileName`, seleccione el campo situado a la derecha del nombre de la propiedad y pulse el botón de puntos suspensivos (...) para abrir el cuadro de diálogo `FileName`. Pulse el botón de puntos suspensivos (...) del cuadro de diálogo `FileName`, localice el archivo `ImportTest.txt`, selecciónelo y pulse Abrir. Haga clic en Aceptar para cerrar el cuadro de diálogo `FileName`.

Nombre de la propiedad	Valor
<code>delimiter</code>	"(comillas dobles)
<code>separator</code>	,(coma)
<code>fileName</code>	<path_to_text_data_file> (vía de acceso a <code>ImportTest.txt</code> , incluido el nombre de archivo)

Un delimitador de un archivo de texto es un carácter que se utiliza para definir el principio y el final de un campo de cadena. Por defecto, el *delimitador* del tipo de datos de cadena son comillas dobles. Para este tutorial no se necesitan cambios.

Un separador de un archivo de texto es un carácter que se utiliza para distinguir los valores de una columna de los de otra. Por defecto, el carácter *separator* es un tabulador (`/t`). En este ejemplo, el separador es

una coma (,). Al utilizar otros archivos de texto, modifique consecuentemente estas propiedades.

Indique la vía de acceso completa y el nombre de archivo en el campo `fileName`.



- 4 Seleccione un componente `TableDataSet` de la ficha Data Express de la paleta de componentes y haga clic en el árbol de componentes o diseñador de interfaces para añadir el componente a la aplicación.

- 5 Seleccione su propiedad `dataFile` y asígnele el valor `textDataFile1`.

- 6 Añada columnas al componente `TableDataSet`.

Este tutorial describe la incorporación de columnas al conjunto de datos mediante el diseñador de interfaces de usuario. Si desea información sobre la forma de añadir columnas por medio del editor, consulte [“Incorporación de columnas a un TableDataSet en el editor” en la página 3-2](#). Si ya ha realizado este tutorial anteriormente y exportó datos a un archivo de texto, JBuilder habrá creado un archivo SCHEMA (.schema) que proporciona definiciones de columna cuando se abre el archivo de texto, por lo que no necesitará añadir columnas de forma manual.

- a Haga clic en el botón de ampliación, a la izquierda del componente `TableDataSet`, para mostrar las columnas.

En este caso, no existen columnas.

- b Seleccione `<nueva columna>` y asigne las siguientes propiedades del Inspector a la primera columna:

Nombre de la propiedad	Valor
<code>dataType</code>	<code>SHORT</code>
<code>title</code>	<code>my_number</code>
<code>columnName</code>	<code>my_number</code>

- c Seleccione `<nueva columna>` y asigne las siguientes propiedades del Inspector a la segunda columna:

Nombre de la propiedad	Valor
<code>dataType</code>	<code>STRING</code>
<code>title</code>	<code>my_string</code>
<code>columnName</code>	<code>my_string</code>

- 7 Seleccione Archivo | Guardar todo para guardar los archivos de código fuente y el archivo de proyecto.

Ahora ya tiene instalados los componentes básicos necesarios para recuperar y almacenar datos del archivo de texto. A continuación, se creará una interfaz de usuario para mostrar y modificar los datos.

Paso 5: Adición de componentes dbSwing para crear una interfaz de usuario

Ahora se puede crear una interfaz de usuario para la aplicación de bases de datos. El modo más rápido de hacerlo es utilizar los componentes dbSwing del diseñador de interfaces de usuario.

Nota Normalmente, el primer paso en la definición de una interfaz de usuario consiste en decidir el diseño adecuado para la aplicación (cuál es la disposición visual de los componentes y qué gestor de diseño Java se va a utilizar para controlar su colocación.) Sin embargo, aprender a utilizar los gestores de diseño Java es una tarea complicada. Para que este tutorial pueda centrarse en la creación de aplicaciones de bases de datos, se utilizará el diseño por defecto (`BorderLayout`), y se controlará la colocación de los componentes por medio de su propiedad `constraints`.

Para aprender más sobre la utilización de diseños, consulte “El diseñador” y “Gestores de diseño” en *Diseño de aplicaciones con JBuilder*.

En los siguientes pasos se añaden a la aplicación los siguientes componentes de la interfaz desde la pestaña dbSwing de la paleta de componentes:

- `JdbTable` se utiliza para mostrar datos bidimensionales en un formato similar al de una hoja de cálculo.
- `JdbNavToolBar` es un conjunto de botones que ayudan a recorrer los datos que se muestran en `JdbTable`, que permite moverse rápidamente en el conjunto de datos mientras la aplicación se está ejecutando.
- `JdbStatusLabel` muestra información sobre el registro o la operación actual y los mensajes de error.

Estos componentes se añaden a `contentPane` (`BorderLayout`), que es un objeto `JPanel`, y el contenedor principal de la interfaz en el que se van a disponer los componentes visuales.

- 1 Seleccione la pestaña dbSwing en la paleta de componentes del diseñador de interfaces de usuario.
- 2 Seleccione el componente `JdbNavToolBar` y haga clic en el área próxima al borde central superior del panel del diseñador de interfaces de usuario.



Se añade al panel una instancia de `JdbNavToolBar`, denominada `jdbNavToolBar1`, y se muestra en el árbol de componentes. Por defecto, el componente `JdbNavToolBar` detecta automáticamente otros componentes

enlazados a datos del mismo contenedor raíz (como JFrame) y se desplaza por el `DataSet` del componente que posee actualmente el foco. Por lo tanto, no es necesario definir la propiedad `dataSet` de `jdbNavToolBar1` en el Inspector.

`jdbNavToolBar1` es ahora el componente seleccionado, y se debe extender a lo largo del borde superior del panel. No se preocupe si se encuentra en un sitio distinto del esperado. El gestor de diseño controla la colocación, que calcula según el lugar en el que se haya hecho clic. Si se ha acercado demasiado a la izquierda, a la derecha o al centro del panel, es posible que haya interpretado que se deseaba colocar el componente en un lugar distinto. Esto se puede resolver en el paso siguiente.

3 Examine la propiedad `constraints` de `jdbNavToolBar1` en el Inspector.

Debe tener el valor `NORTH`. Si no es así, haga clic en el valor para abrir la lista desplegable y elija `North`.



4 Seleccione el componente `JdbStatusLabel` y haga clic en el área próxima al borde central superior del panel del diseñador de interfaces de usuario.

Una instancia de `JdbStatusLabel`, llamada `jdbStatusLabel1`, se añade al panel y se muestra en el árbol de componentes. `jdbStatusLabel1` debe tener un valor `SOUTH` de la propiedad `constraints`. Si no es así, cámbielo en el inspector. `jdbStatusLabel1` se asocia automáticamente al conjunto de datos que tiene el foco.



5 Seleccione un componente `TableScrollPane` en la pestaña `dbSwing` de la paleta de componentes y haga clic en el centro del panel del diseñador de interfaces de usuario para añadir el componente a la aplicación.

El componente `TableScrollPane`, `tableScrollPane1`, aparece como un nodo en el árbol de componentes del panel de estructura.



6 Seleccione un componente `JdbTable` en la pestaña `dbSwing` de la paleta de componentes y haga clic en el árbol de componentes o diseñador de interfaces de usuario para añadir el componente a la aplicación.

El componente `JdbTable`, `jdbTable1`, aparece como un nodo bajo `tableScrollPane1` en el árbol de componentes del panel de estructura.

7 Asigne a la propiedad `dataSet` de `jdbTable1` el valor `tableDataSet1`.

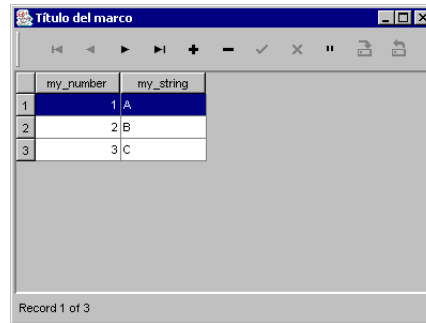
Cuando se asigna a la propiedad `dataSet` de `jdbTable1` el valor `tableDataSet1`, los datos del archivo de texto aparecen en el diseñador de interfaces de usuario:

Si no se especifica un archivo de datos o no se definen correctamente las columnas, aparece un cuadro de diálogo de error. Si no se instancia un componente visual para ver los datos, se debe abrir explícitamente el archivo desde el código fuente para acceder a los datos.

- 8 Seleccione Ejecutar | Ejecutar proyecto para compilar y ejecutar la aplicación.

La aplicación en ejecución presentará este aspecto:

Figura 16.2 Aplicación Importar/Exportar en ejecución



- 9 Cierre la aplicación que se está ejecutando.

Al ejecutar esta aplicación, los datos del archivo de texto se cargan en un `TableDataSet` y se visualizan en el componente visual de la tabla al que están asociados. Ahora los datos del conjunto de datos se pueden ver, editar, añadir y borrar. Un componente `TableDataSet` se puede utilizar como tabla maestra o de detalle en una relación maestro-detalle. Para guardar los cambios en el archivo de texto, hay que exportar los datos al archivo. Si desea más información sobre la exportación, consulte [“Exportación de datos” en la página 3-4](#).

Paso 6: Adición de un componente Swing JButton

El componente Swing JButton se utiliza para exportar datos desde la aplicación en ejecución. La exportación de datos o *almacenamiento de datos en un archivo de texto*, guarda todos los datos de la vista actual en un archivo de texto, sobrescribiendo los datos existentes. En este tutorial, aprenderá a almacenar datos de un componente `TableDataSet` en el archivo de texto utilizado originalmente para importar los datos. Al *exportar* datos a un archivo de texto, todos los datos de la vista actual se escriben en el archivo y no influyen en la información de estado de la fila.

- 1 Seleccione la pestaña Diseño del panel de contenido.
- 2 Seleccione el elemento `contentPane` (`BorderLayout`) en el panel de contenido y cambie su propiedad `layout` a `null` en el inspector.
- 3 En el árbol de componentes, seleccione `tableScrollPane1`; a continuación, en el diseñador de interfaces de usuario, utilice el tirador superior para modificar el tamaño del componente de modo que se pueda incluir un botón.

Si desea ayuda sobre la colocación general de los componentes, consulte la imagen de pantalla de la aplicación ejecutada, más adelante en este tutorial.

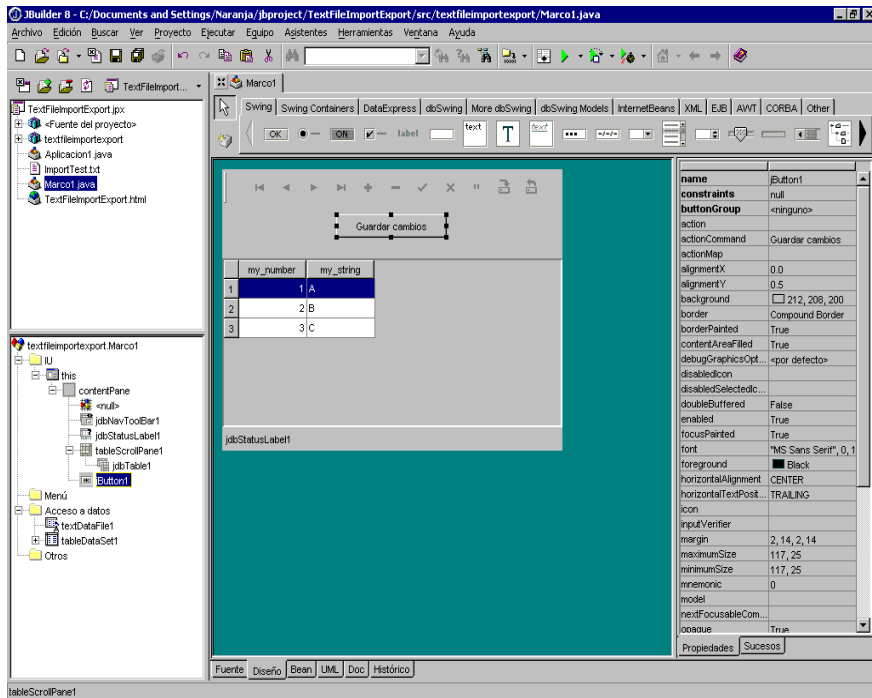
- 4 Añada un componente JButton desde la ficha Swing del diseñador de interfaces de usuario y, a continuación, en la pestaña Propiedades del Inspector, asigne a la propiedad text del componente JButton el valor
- 5 Abra la pestaña Sucesos del Inspector, seleccione el método actionPerformed() y haga doble clic en él.

Con ello, se cambia el foco del Visualizador de aplicaciones del diseñador de interfaces de usuario al panel de código fuente y se visualiza el stub del método actionPerformed() en el código fuente.

Añada el siguiente código al método actionPerformed():

```
try {
    tableDataSet1.getDataFile().save(tableDataSet1);
    System.out.println("Cambios guardados");
}
catch (Exception ex) {
    System.out.println("Cambios NO guardados");
    System.err.println("Excepción: " + ex);
}
```

Figura 16.3 Aplicación de importación y exportación de base de datos con JButton



- 6 Seleccione Archivo | Guardar todo para guardar los archivos de código fuente y el archivo de proyecto.

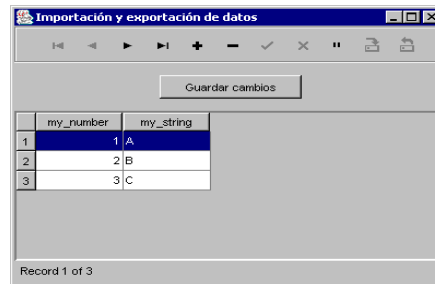
Paso 7: Compilación y ejecución de la aplicación

Cuando compile y ejecute la aplicación, ésta incluirá un botón Guardar cambios para exportar datos. Cuando se exportan datos de TableDataSet a un archivo de texto, JBuilder crea un archivo SCHEMA (.schema) que define las columnas por nombre y tipo de datos. La siguiente vez que se importen datos en JBuilder, no será necesario definir las columnas ya que esta información se especifica en el archivo SCHEMA.

- 1 Ejecute la aplicación seleccionando Ejecutar | Ejecutar proyecto.

Cuando ejecute la aplicación, ésta aparecerá en su propia ventana. Los datos se visualizan en una tabla, con el botón Guardar cambios.

Figura 16.4 Aplicación de exportación de datos a un archivo de texto en ejecución



- 2 Con la aplicación en ejecución, seleccione el campo de cadena en el primer registro de la ventana Marco y cambie el valor del campo de A a Apple.
- 3 Guarde los cambios en el archivo de texto pulsando el botón Guardar cambios.
- 4 Abra ImportTest.txt en el panel de contenido y observe que ahora contiene los siguientes datos:

```
1, "Apple"  
2, "B"  
3, "C"
```

- 5 Cierre el archivo de texto.

JBuilder crea automáticamente un archivo SCHEMA para definir el contenido del archivo de texto.

- 6 Consulte el archivo SCHEMA en el editor de texto. Observe que este archivo contiene información sobre el nombre de los campos que se han exportado y sobre el tipo de datos que se exportaron en dicho campo. Tiene el siguiente aspecto:

```
[  
    FILETYPE = VARYING  
    FILEFORMAT = Encoded  
    ENCODING = Cp1252  
    DELIMITER = "  
    SEPARATOR = ,  
    FIELD0 = my_number,Variant.SHORT,-1,-1,  
    FIELD1 = my_string,Variant.STRING,-1,-1,
```

7 Cierre el archivo SCHEMA.

Puede continuar editando, insertando, eliminando y guardando datos hasta que cierre la aplicación, pero debe hacer clic en el botón Guardar cambios para escribir los cambios en el archivo de texto. Cuando se guardan los cambios, se sobrescribe el archivo de texto existente con los datos de la vista actual.

Paso 8: Uso de modelos en la exportación de campos numéricos de fecha, hora y texto

Por defecto, JBuilder espera la entrada de datos y exporta los campos de fecha, hora y moneda conforme a la propiedad `locale` de la columna. Se puede utilizar la propiedad `exportDisplayMask` para leer o guardar los campos de fecha, hora y números en un modelo diferente. Estos pasos siguientes enseñan a crear una `exportDisplayMask` para una nueva columna del tipo DATE (Fecha).

- 1 Seleccione `Marco1.java` del panel de contenido y, a continuación, seleccione la pestaña Diseño. Amplíe el componente `tableDataSet1` en el árbol de componentes haciendo clic en el icono de expansión que tiene a la izquierda. Seleccione `<nueva columna>` y modifique como sigue las propiedades de la columna, en el Inspector:

- a `dataType` el valor `DATE`
- a `caption` y `columnName` el valor `my_date`

- 2 Ejecute la aplicación. En la ventana de la aplicación en ejecución, introduzca una fecha en la columna `my_date` de la primera fila. Por defecto, debe introducir dicha fecha con el formato `dd/MM/aa`, como `16/11/95`. Pulse el botón Guardar cambios para guardar los cambios en el archivo de texto.

- 3 Abra el archivo en un editor de texto. Contendrá ahora los siguientes datos:

```
1, "Apple", 1995-11-16  
2, "B"  
3, "C"
```

- 4 Cierre el archivo de texto.

- 5 Consulte el archivo SCHEMA en el editor de texto. Observe que se ha añadido el nuevo campo de fecha a la lista de campos. Tiene el siguiente aspecto:

```
[ ]
FILETYPE = VARYING
FILEFORMAT = Encoded
ENCODING = Cp1252
DELIMITER = "
SEPARATOR = ,
FIELD0 = my_number,Variant.SHORT,-1,-1,
FIELD1 = my_string,Variant.STRING,-1,-1,
FIELD2 = my_date,Variant.DATE,-1,-1,
```

- 6 Cierre el archivo SCHEMA.

En el siguiente paso se muestra lo que ocurre cuando se cambia el patrón de fecha, se modifican los datos y se guardan los cambios.

- 1 Cierre la aplicación en ejecución y los archivos de texto y vuelva al diseñador de JBuilder. Seleccione la columna `my_date` e introduzca el modelo siguiente en la propiedad `exportDisplayMask` del Inspector: `MM-dd-aaaa`. La sintaxis de los modelos se define en “String-based patterns (masks)” en *DataExpress Component Library Reference*. Este tipo de modelo guarda el campo de la fecha de la siguiente forma: 11-16-1995.
- 2 La aplicación daría un error si tratara de ejecutarla, porque el formato del campo fecha del archivo de texto no coincide con el que la aplicación intenta abrir. Edite de forma manual el archivo de texto y elimine el valor “11/16/95” de la primera fila.

En vez de realizar el paso anterior, se puede introducir manualmente el código que establezca una propiedad `exportDisplayMask` para la importación de datos y otra para la exportación.

- 3 Ejecute la aplicación e introduzca una fecha del tipo 16/11/1995, en la columna `my_date` de la primera fila y pulse el botón Guardar cambios para guardar los cambios en el archivo de texto.
- 4 Abra el archivo en un editor de texto. Contendrá ahora los siguientes datos:

```
1, "Apple",11-16-1995
2, "B"
3, "C"
```

- 5 Cierre el archivo de texto.
- 6 Consulte el archivo SCHEMA en el editor de texto. Observe que el formato del campo de fecha se visualiza como parte de la definición de campo. Cuando se utiliza el formato por defecto, este valor está vacío, como en la definición `FIELD0`. Tiene el siguiente aspecto:

```
[  
    FILETYPE = VARYING  
    FILEFORMAT = Encoded  
    ENCODING = Cp1252  
    DELIMITER = "  
    SEPARATOR = ,  
    FIELD0 = my_number,Variant.SHORT,-1,-1,  
    FIELD1 = my_string,Variant.STRING,-1,-1,  
    FIELD2 = my_date,Variant.DATE,-1,-1,MM-dd-yyyy
```

7 Cierre el archivo SCHEMA.

La siguiente vez que se importe el archivo de texto, los datos se importarán de la información existente en el archivo SCHEMA. Para ver los datos en la tabla con un modelo diferente, asigne un valor a la propiedad `displayMask`. Para modificar los datos en la tabla con un modelo diferente, asigne un valor a la propiedad `editMask`. Estas propiedades influyen únicamente en la vista y en la edición de los datos, no en la forma de guardarlos. Por ejemplo, para introducir datos en un campo de moneda sin que sea necesario cada vez introducir el símbolo de moneda, utilice un `displayMask` que utilice el símbolo de moneda y una `editMask` que no lo contenga. Los datos se pueden guardar en el archivo de texto con o sin el símbolo de moneda al configurar el `exportDisplayMask`.

Tutorial: Creación de aplicaciones de base de datos distribuidas

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

En este tutorial se describe cómo desarrollar una aplicación de base de datos de ejemplo mediante los componentes DataExpress y las herramientas de diseño de JBuilder. Donde ha resultado necesario, el código generado por las herramientas de diseño se ha modificado para conseguir un comportamiento personalizado.

Esta aplicación demuestra la siguiente funcionalidad:

- Se conecta con la base de datos JDataStore de ejemplo, `employee.jds`, utilizando los componentes `Database` y `QueryDataSet`. (Consulte [Capítulo 4, “Conexión con bases de datos”](#) y [“Consultas en bases de datos”](#) en la página 5-2).
- Contiene una `JdbTable` que visualiza los datos y muestra las características siguientes:
 - Columnas persistentes, que son columnas en las que la información de la estructura, procedente normalmente del servidor, se especifica en su lugar como una propiedad de la columna. Con ello se consigue un mejor rendimiento, así como la persistencia de las propiedades a nivel de columna. (Consulte [“Columnas persistentes”](#) en la [página 7-8](#) para obtener más información sobre esta función.) En el diseñador, haga doble clic sobre el conjunto de datos para abrir el diseñador de columnas, en el que podrá ver más datos sobre cada una de las columnas.
 - Formato de los datos visualizados en el `JdbTable` utilizando máscaras de visualización (la columna `HIRE_DATE`). (Consulte

[“Incorporación de una plantilla de edición o visualización para formatear datos” en la página 12-17](#)).

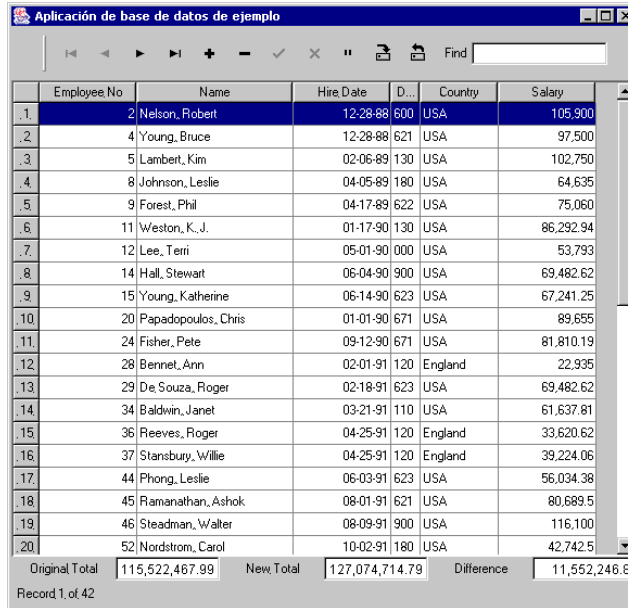
- Edición de datos controlada mediante máscaras (la columna `HIRE_DATE`). (Consulte [“Incorporación de una plantilla de edición o visualización para formatear datos” en la página 12-17](#)).
- Campos calculados y acumulados cuyos valores proceden de la evaluación de una expresión (las columnas `NEW_SALARY`, `ORG_TOTAL`, `NEW_TOTAL`, `DIFF_SALARY`, Y `DIFF_TOTAL`). (Consulte [“Utilización de columnas calculadas” en la página 12-8](#)).
- Incluye un control `JdbStatusLabel` que visualiza la información de desplazamiento, mensajes de validación de datos, etc. Los mensajes se escriben en el control `JdbStatusLabel`, cuando procede, o según las instrucciones del programa. (Consulte [“Visualización de la información de estado” en la página 13-3](#)).
- Muestra un componente `JdbNavToolBar` que facilita el desplazamiento por los datos que se muestran en la tabla.
- Permite localizar los datos de forma interactiva, por medio de un objeto `JdbNavField` incrustado en `JdbNavToolBar`. Para obtener más información sobre la localización de datos, consulte [“Localización de datos” en la página 11-15](#).
- Utiliza un objeto `DBDisposeMonitor` para cerrar automáticamente la conexión de la base de datos cuando se cierra el marco.
- Resuelve los cambios realizados a los datos en `QueryDataSet` mediante el comportamiento de resolución por defecto. (Consulte [“Cómo funciona el almacenador por defecto” en la página 8-18](#)). El botón Guardar de `JdbNavToolBar` es el que guarda la información. Los mensajes relativos al proceso de resolución se visualizan en el control `JdbStatusLabel`.

En este tutorial se realizarán las siguientes tareas:

- Crear un proyecto con JBuilder.
- Generar una aplicación.
- Añadir componentes DataExpress para acceder a datos de la base de datos.
- Diseñar las columnas de la aplicación.
- Añadir componentes dbSwing para crear una interfaz de usuario.
- Totalizar datos con campos calculados.

Cuando se finalice el tutorial, la aplicación debe tener un aspecto similar a este:

Figura 17.1 Aplicación básica de base de datos



Employee No.	Name	Hire Date	D...	Country	Salary
1	Nelson, Robert	12-28-88	600	USA	105,900
2	Young, Bruce	12-28-88	621	USA	97,500
3	Lambert, Kim	02-06-89	130	USA	102,750
4	Johnson, Leslie	04-05-89	180	USA	64,635
5	Forest, Phil	04-17-89	622	USA	75,060
6	Weston, K. J.	01-17-90	130	USA	86,292.94
7	Lee, Terri	05-01-90	000	USA	53,793
8	Hall, Stewart	06-04-90	900	USA	69,482.62
9	Young, Katherine	06-14-90	623	USA	67,241.25
10	Papadopoulos, Chris	01-01-90	671	USA	89,655
11	Fisher, Pete	09-12-90	671	USA	81,810.19
12	Bennet, Ann	02-01-91	120	England	22,935
13	De Souza, Roger	02-18-91	623	USA	69,482.62
14	Baldwin, Janet	03-21-91	110	USA	61,637.81
15	Reeves, Roger	04-25-91	120	England	33,620.62
16	Stansbury, Willie	04-25-91	120	England	39,224.06
17	Phong, Leslie	06-03-91	623	USA	56,034.38
18	Ramanathan, Ashok	08-01-91	621	USA	80,689.5
19	Steadman, Walter	08-09-91	900	USA	116,100
20	Nordstrom, Carol	10-02-91	180	USA	42,742.5

Original Total	115,522,467.99	New Total	127,074,714.79	Difference	11,552,246.8
----------------	----------------	-----------	----------------	------------	--------------

Record 1 of 42

La aplicación finalizada se encuentra en el archivo de proyecto de ejemplo BasicApp.jpx,, situado en el directorio </jbuilder/samples/DataExpress/BasicApp/. Pueden existir pequeñas diferencias entre la aplicación creada en este tutorial y la aplicación de ejemplo. Los usuarios con acceso de sólo lectura a los ejemplos de JBuilder deben copiar el directorio samples en un directorio con permiso de lectura y escritura.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-8](#).

Paso 1: Creación de un proyecto

Para desarrollar una aplicación de base de datos en JBuilder, es necesario primero crear un proyecto. Para ello:

- 1 Seleccione Archivo | Nuevo proyecto con el fin de abrir el Asistente para proyectos.

- 2 Escriba `BasicApp` en el campo Nombre.
- 3 Pulse Finalizar para cerrar el Asistente para proyectos y crearlo. No se necesita hacer ningún cambio en los valores por defecto en los Pasos 2 y 3 del asistente.

El archivo de proyecto `BasicApp.jpx` se muestra en el panel del proyecto.

Paso 2: Generación de una aplicación

El Asistente para aplicaciones crea archivos de código fuente `.java` que se añaden al proyecto que se acaba de crear.

Para generar los archivos fuente de una aplicación con el Asistente para aplicaciones, siga estas instrucciones:

- 1 Abra la galería de objetos seleccionando Archivo | Nuevo.
- 2 Seleccione la pestaña General y haga doble clic en el icono Aplicación para que se abra el Asistente para aplicaciones.
- 3 En el Paso 1 de Asistente para aplicaciones, acepte el nombre del paquete por defecto, `basicapp`, escriba `BasicApp` en el campo Clase y pulse siguiente.

Nota El nombre de paquete utilizado en este tutorial, `basicapp`, difiere del nombre de paquete utilizado en la aplicación de ejemplo, `com.borland.samples.dx.basicapp`, pero las aplicaciones son las mismas.

- 4 En el Paso 2 del Asistente para aplicaciones, escriba `BasicAppFrame` en el campo Clase, escriba `Sample Database Application` en el campo Título y pulse Finalizar.

Los nuevos archivos de código fuente Java se añaden al proyecto y se muestran como nodos en el panel del proyecto. El código fuente para `BasicAppFrame.java` aparece abierto en el panel de contenido.

- 5 Seleccione Archivo | Guardar todo para guardar los archivos de código fuente y el archivo de proyecto.

Paso 3: Incorporación de componentes DataExpress a la aplicación

El diseñador de interfaces de usuario se utiliza para añadir componentes DataExpress a `BasicAppFrame.java`. Se añadirán los siguientes componentes DataExpress a la aplicación:

- Database
- QueryDataSet
- DBDisposeMonitor

Estos componentes proporcionan la estructura de base de datos subyacente para la aplicación.

- 1 Seleccione la pestaña Diseño de `BasicAppFrame.java` en el panel de contenido para activar el diseñador de interfaces de usuario.

La paleta de componentes aparece en la parte superior del diseñador de interfaces de usuario.



- 2 Haga clic en el componente `Database` de la pestaña DataExpress de la paleta de componentes y, a continuación, haga clic en el árbol de componentes o en el diseñador de interfaces de usuario para añadir el componente a la aplicación.

El nuevo componente `Database`, `database1`, aparece bajo el nodo Acceso a datos del panel de estructura; además, se añaden las siguientes líneas de código a la clase Marco:

```
Database database1 = new Database();
```

- 3 Seleccione el componente `database1` en el panel de estructura y la propiedad `connection` en el Inspector y pulse el botón de puntos suspensivos (...) para abrir el cuadro de diálogo Conexión.
- 4 Configure las propiedades de conexión a la tabla de empleados de ejemplo de `JDataStore` utilizando los valores de campo de la siguiente tabla.

La dirección URL de conexión apunta al archivo `employee.jds` situado en un subdirectorio de la instalación de JBuilder, `<jbuilder>`.

Nombre de la propiedad	Valor
Controlador	<code>com.borland.datastore.jdbc.DataStoreDriver</code>
URL	Busque el archivo <code><jbuilder>/samples/JDataStore/datastores/employee.jds</code>
Nombre de usuario	Introduzca su nombre
Contraseña	No es obligatoria

El cuadro de diálogo Connection contiene un botón Probar conexión. Púselo para comprobar que las propiedades de conexión tienen los valores correctos. Los resultados del intento de conexión se muestran en el área de estado. Cuando la conexión sea satisfactoria, pulse Aceptar. Si la conexión no se efectúa adecuadamente, asegúrese de que ha seguido todos los pasos del [Capítulo 4, "Conexión con bases de datos"](#).

- 5 Añada un componente `QueryDataSet` al diseñador, haciendo clic en el componente `QueryDataSet` de la pestaña DataExpress y, después, haga clic en el árbol de componentes.

Seleccione en el Inspector la propiedad `query` del componente `QueryDataSet`, pulse el botón de puntos suspensivos para que se abra el cuadro de diálogo `QueryDescriptor` y asigne valores a las siguientes propiedades:

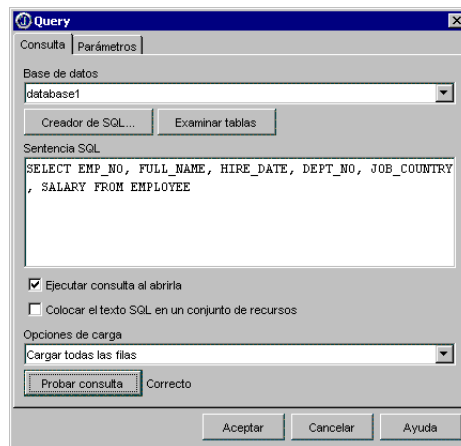
Nombre de la propiedad	Valor
Database	database1
Sentencia SQL	SELECT EMP_NO, FULL_NAME, HIRE_DATE, DEPT_NO, JOB_COUNTRY, SALARY FROM EMPLOYEE

Al abrir el componente `QueryDataSet`, la sentencia SQL se ejecutará automáticamente contra el componente `Database` especificado.

- Haga clic sobre Probar consulta para cerciorarse de que se ejecuta correctamente.

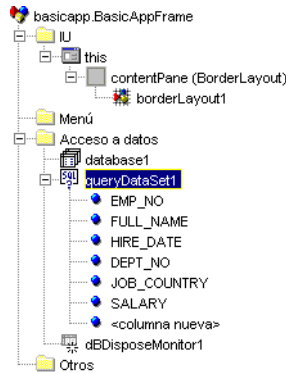
Si la consulta se realiza correctamente, el cuadro de diálogo Consulta presenta el siguiente aspecto.

Figura 17.2 Cuadro de diálogo de la consulta



Si el cuadro de diálogo Consulta indica Fallo, revise la información especificada en la consulta, buscando errores ortográficos y omisiones.

- Pulse Aceptar para cerrar el cuadro de diálogo Consulta.
- Añada el componente `DBDisposeMonitor` de la pestaña Más dbSwing. El componente `DBDisposeMonitor` cierra el `JDataStore` cuando se cierre la ventana.
- Asigne a la propiedad `dataAwareComponentContainer` de `DBDisposeMonitor` el valor `this`.
- Amplíe el nodo `queryDataSet1` en el panel de estructura.

Figura 17.3 Nodo queryDataSet1 expandido

Las columnas seleccionadas del ejemplo de base de datos Employee de JDataStore, `employee.jds`, se muestran en el nodo `queryDataSet1`.

Ahora ya tiene los componentes básicos instalados para recuperar y almacenar datos de la base de datos Employee. A continuación, se creará una interfaz de usuario para mostrar y modificar los datos.

Paso 4: Diseño de las columnas de la aplicación

Antes de añadir una interfaz de usuario a la aplicación, deberemos:

- Añadir nuevas columnas y modificar columnas existentes.
- Especificar un cálculo para las columnas calculadas.

Añadir columnas y modificar propiedades de columnas

- 1 Expanda el nodo `queryDataSet1` del panel de estructura y haga doble clic en `<nueva columna>` para añadir una columna.

Al hacer esto, el diseñador de columnas aparece en el panel de contenido y se cargan las propiedades de la nueva columna en el Inspector.

- 2 Cambie la propiedad `columnName` en el Inspector del valor `NewColumn1` al valor `NEW_SALARY`.



3 Pulse el botón Insertar columna para añadir cuatro columnas adicionales con los siguientes valores de la propiedad `columnName`:

- `DIFF_SALARY`
- `ORIG_TOTAL`
- `NEW_TOTAL`
- `DIFF_TOTAL`

4 Configure las propiedades de la columnas según se describe en la siguiente tabla:

Column	Nombre de la propiedad	Valor
HIRE_DATE	title	Hire Date
HIRE_DATE	displayMask	MM-dd-aa
HIRE_DATE	editMask	MM-dd-aaaa
NEW_SALARY	title	NEW_SALARY
NEW_SALARY	calcType	calculated
NEW_SALARY	dataType	BIGDECIMAL
NEW_SALARY	visible	FALSE
EMP_NO	title	Employee No
FULL_NAME	title	Name
FULL_NAME	width	16
DEPT_NO	title	Dept.
JOB_COUNTRY	title	Country
JOB_COUNTRY	width	15
SALARY	title	Salary
ORIG_TOTAL	calcType	aggregated
ORIG_TOTAL	title	ORIG_TOTAL
ORIG_TOTAL	dataType	BIGDECIMAL
NEW_TOTAL	calcType	aggregated
NEW_TOTAL	title	New Total
NEW_TOTAL	dataType	BIGDECIMAL
DIFF_SALARY	calcType	calculated
DIFF_SALARY	title	DIFF_SALARY
DIFF_SALARY	dataType	BIGDECIMAL
DIFF_TOTAL	calcType	aggregated
DIFF_TOTAL	title	Diff. Total
DIFF_TOTAL	dataType	BIGDECIMAL

Si se modifican las propiedades de una columna, ésta se convierte en persistente. Cuando una columna se ha convertido en persistente, su nombre aparece entre corchetes ([]) en el panel de estructura.

Cuando termine de modificar las columnas, el diseñador de columnas debería presentar un aspecto similar al siguiente:

Figura 17.4 Columnas queryDataSet1 en el diseñador de columnas

Column	columnNa...	data Type	preferredOr...	editMask	def
column3	EMP_NO	SHORT	-1		
column4	FULL_NAME	STRING	-1		
column1	HIRE_DATE	TIMESTAMP	-1	MM-dd-yyyy	
column5	DEPT_NO	STRING	-1		
column6	JOB_COUNT	STRING	-1		
column7	SALARY	BIGDECIMAL	-1		
column2	NEW_SALAR	BIGDECIMAL	-1		
column8	ORIG_TOTAL	BIGDECIMAL	-1		
column9	NEW_TOTAL	BIGDECIMAL	-1		
column10	DIFF_SALAR	BIGDECIMAL	-1		
column11	DIFF_TOTAL	BIGDECIMAL	-1		

Especificación de cálculos para las columnas calculadas

Las columnas NEW_SALARY y DIFF_SALARY son columnas calculadas. En este tutorial, vamos a conceder a cada empleado un aumento del 10%. El cálculo suma el dato SALARY existente al producto de SALARY por 0,10. El valor resultante se coloca en la columna NEW_SALARY. La columna DIFF_SALARY se calcula restando el valor SALARY existente al valor NEW_SALARY.

Para añadir el cálculo:

- 1 Seleccione el nodo `queryDataSet1` en el panel de estructura, abra la pestaña Sucesos del Inspector y haga doble clic sobre el controlador del suceso `calcFields`.

Esto crea el stub para el método del suceso en `BasicAppFrame.java` y muestra el código fuente para el nuevo método en el panel de contenido.

- 2 Agregue la siguiente sentencia a las sentencias de importación existentes en `BasicAppFrame.java` para importar la clase `java.math.BigDecimal` necesaria para el tipo de datos `BIGDECIMAL` especificado para las columnas calculadas.

```
import java.math.BigDecimal;
```

- 3 Modifique el método del suceso de modo que calcule el valor para NEW_SALARY y DIFF_SALARY del siguiente modo:

```
void queryDataSet1_calcFields(ReadRow changedRow, DataRow calcRow,
    boolean isPosted)
    throws DataSetException {
    BigDecimal bDin = changedRow.getBigDecimal("Salary");
    BigDecimal bDout = bDin.add(new BigDecimal(bDin.doubleValue()*10.0/100));
    calcRow.setBigDecimal("NEW_SALARY", bDout);
```

```
        calcRow.setBigDecimal("DIFF_SALARY", bDout.subtract(bDin));  
    }
```

`calcFields` llama a este método siempre que se guarda el valor de un campo o se envía una fila. Este suceso pasa una entrada que son los valores actuales de la fila (`changedRow`), una fila de salida para poner los cambios que desee efectuar en la fila (`calcRow`) y un booleano (`isPosted`) que indica si la fila se incluye en el `DataSet` o no. Quizá no se desee volver a calcular los campos de las filas que todavía no hayan sido enviadas.

Algunas de las columnas que añadimos en este paso son columnas de totalización. Trataremos más adelante en el tutorial este tipo de columnas. Ahora, añadiremos una interfaz de usuario a la aplicación de modo que podamos ver su aspecto.

Paso 5: Adición de componentes dbSwing para crear una interfaz de usuario

Ahora se puede crear una interfaz de usuario para la aplicación de base de datos. El modo más rápido de hacerlo es utilizar los componentes dbSwing del diseñador de interfaces de usuario.

Nota Normalmente, el primer paso en la definición de una interfaz de usuario consiste en decidir el diseño adecuado para la aplicación (cuál es la disposición visual de los componentes y qué gestor de diseño Java se va a utilizar para controlar su colocación). Sin embargo, aprender a utilizar los gestores de diseño Java es una tarea complicada. Para que este tutorial pueda centrarse en la creación de aplicaciones de base de datos, se utilizará el diseño por defecto (`BorderLayout`) y se controlará la colocación de los componentes por medio de su propiedad `constraints`.

Para aprender más sobre la utilización de diseños, consulte “Introducción al diseñador” y “Gestores de diseño” en *Diseño de aplicaciones con JBuilder*.

En los siguientes pasos se añaden a la aplicación los siguientes componentes de la interfaz desde la pestaña dbSwing de la paleta de componentes:

- `JdbTable` se utiliza para mostrar datos bidimensionales en un formato similar al de una hoja de cálculo.
- `JdbNavToolBar` es un conjunto de botones que ayudan a recorrer los datos que se muestran en `JdbTable`, que permite moverse rápidamente en el conjunto de datos mientras la aplicación se está ejecutando.
- `JdbStatusLabel` muestra información sobre el registro o la operación actual y los mensajes de error.

Estos componentes se añaden a `contentPane` (`BorderLayout`), que es un objeto `JPanel`, y el contenedor principal de la interfaz en el que se van a disponer los componentes visuales. Se utilizarán componentes `JPanel` adicionales para separar los componentes de navegación del componente `JdbStatusLabel`.

Para añadir el componente `JdbTable`:

- 1 Seleccione la pestaña Diseño de `BasicAppFrame.java` en el panel de contenido para activar el diseñador de interfaces de usuario.
- 2 Seleccione la pestaña `dbSwing` en la paleta de componentes del diseñador de interfaces de usuario.
- 3 Para añadir el componente a la aplicación, seleccione un componente `TableScrollPane` de la pestaña `dbSwing` de la paleta de componentes y pulse sobre `contentPane` del árbol de componentes o en el centro de la superficie de diseño del diseñador de interfaces de usuario.

El componente `TableScrollPane`, `tableScrollPane1`, aparece como un nodo en el árbol de componentes del panel de estructura.



- 4 Para añadir el componente a la aplicación, seleccione un componente `JdbTable` de la pestaña `dbSwing` de la paleta de componentes y pulse sobre `tableScrollPane1` del árbol de componentes o en el centro de la superficie de diseño del diseñador de interfaces de usuario.

El componente `JdbTable`, `jdbTable1`, aparece como un nodo bajo `tableScrollPane1` en el árbol de componentes del panel de estructura.

- 5 Asigne a la propiedad `dataSet` de `jdbTable1` el valor `queryDataSet1`.

Cuando se asigna a la propiedad `dataSet` de `jdbTable1` el valor `queryDataSet1`, los datos del archivo de texto aparecen en el diseñador de interfaces de usuario:

Figura 17.5 Componente `JdbTable` en el diseñador de interfaces de usuario

	Employee No	Name	Hire Date	De...	J
1	2	Nelson, Robert	12-28-88	600	U
2	4	Young, Bruce	12-28-88	621	U
3	5	Lambert, Kim	02-06-89	130	U
4	8	Johnson, Leslie	04-05-89	180	U
5	9	Forest, Phil	04-17-89	622	U
6	11	Weston, K. J.	01-17-90	130	U
7	12	Lee, Terri	05-01-90	000	U
8	14	Hall, Stewart	06-04-90	900	U
9	15	Young, Katherine	06-14-90	623	U
10	20	Papadopoulos, Chris	01-01-90	671	U
11	24	Fisher, Pete	09-12-90	671	U
12	28	Bennet, Ann	02-01-91	120	E
13	29	De Souza, Roger	02-18-91	623	U

A continuación, añadiremos algunos componentes de navegación, incluido un componente `JdbNavToolBar`. Los componentes `JPanel`

ayudarán a separar los diferentes tipos de elementos de la interfaz de usuario.

Para añadir los elementos de navegación:



- 1 Abra la pestaña Contenedores Swing de la paleta de componentes del diseñador de interfaces de usuario, seleccione el componente `JPanel` y pulse sobre el nodo `contentPane` del panel de estructura.



- 2 Asigne a la propiedad `layout` de `jPanel1` el valor `FlowLayout`.

- 3 Abra la pestaña Más dbSwing de la paleta de componentes, seleccione el componente `JdbNavField` y pulse sobre el nodo `jPanel1` del panel de estructura.

El `JdbNavField` incluye una función de búsqueda incremental para las columnas de tipo `String`. Su propiedad `columnName` especifica en qué columna se lleva a cabo la localización. Si no se define, la localización se efectúa en la última columna visitada del componente `JdbTable`.

- 4 Asigne a la propiedad `preferredSize` de `jdbNavField1` el valor `125, 21`.



- 5 Abra la pestaña Swing de la paleta de componentes, seleccione el componente `JLabel` y pulse sobre el nodo `jPanel1` del panel de estructura.

- 6 Asigne a la propiedad `text` de `jLabel1` el valor `Find`.



- 7 Abra la pestaña dbSwing de la paleta de componentes, seleccione el componente `JdbNavToolBar` y pulse sobre el nodo `jPanel1` del panel de estructura.

Se añade al panel una instancia de `JdbNavToolBar`, denominada `jdbNavToolBar1`, y se muestra en el árbol de componentes. Por defecto, el componente `JdbNavToolBar` detecta automáticamente otros componentes enlazados a datos del mismo contenedor raíz, y se desliza por el `DataSet` del componente que posee actualmente el foco. Por lo tanto, no es necesario definir la propiedad `dataSet` de `jdbNavToolBar1` en el Inspector.

Nota

Quizás sea necesario redimensionar el espacio de trabajo del diseñador para poder ver todos los componentes de interfaz de usuario.

Ahora, ya estamos listos para añadir el componente `JdbStatusLabel`.

Para añadir el componente `JdbStatusLabel`:

- 1 Añada otro componente `JPanel` al nodo `contentPane` del panel de estructura.
- 2 En el Inspector, asigne a la propiedad `constraints` de `jPanel2` el valor `South`.



- 3 Abra la pestaña dbSwing de la paleta de componentes, seleccione el componente `JdbStatusLabel` y haga clic en el área próxima al borde central inferior del panel del diseñador de interfaces de usuario.

Una instancia de `JdbStatusLabel`, llamada `jdbStatusLabel1`, se añade al panel y se muestra en el árbol de componentes. `jdbStatusLabel1` se asocia automáticamente al objeto `DataSet` que tiene el foco.

- 4 Seleccione Ejecutar | Ejecutar proyecto para compilar y ejecutar la aplicación.

La aplicación en ejecución presentará este aspecto:

Figura 17.6 Aplicación básica de base de datos con barra de navegación y etiqueta de estado

	Employee No	Name	Hire Date	Dept.	Country	Salary
1	2	Nelson, Robert	12-28-88	600	USA	105,900
2	4	Young, Bruce	12-28-88	621	USA	97,500
3	5	Lambert, Kim	02-06-89	130	USA	102,750
4	8	Johnson, Leslie	04-05-89	180	USA	64,635
5	9	Forest, Phil	04-17-89	622	USA	75,060
6	11	Weston, K. J.	01-17-90	130	USA	86,292.94
7	12	Lee, Terri	05-01-90	000	USA	53,793
8	14	Hall, Stewart	06-04-90	900	USA	69,482.62
9	15	Young, Katherine	06-14-90	623	USA	67,241.25
10	20	Papadopoulos, Chris	01-01-90	671	USA	89,655
11	24	Fisher, Pete	09-12-90	671	USA	81,810.19
12	28	Bennet, Ann	02-01-91	120	England	22,935
13	29	De Souza, Roger	02-18-91	623	USA	69,482.62
14	34	Baldwin, Janet	03-21-91	110	USA	61,637.81
15	36	Reeves, Roger	04-25-91	120	England	33,620.62
16	37	Stansbury, Willie	04-25-91	120	England	39,224.06
17	44	Phong, Leslie	06-03-91	623	USA	56,034.38
18	45	Ramanathan, Ashok	08-01-91	621	USA	80,689.5
19	46	Steadman, Walter	08-09-91	900	USA	116,100

Record 1 of 42

Utilice la barra de navegación y el campo de navegación para desplazarse por los registros. Observe cómo la barra de estado se actualiza a medida que se va desplazando.

- 5 Cierre la aplicación en ejecución y guarde todos los cambios (Archivo | Guardar todo).

Para completar la aplicación, se añaden algunos componentes `JdbTextField` a la interfaz de usuario para mostrar los datos de las columnas de totalización.

Paso 6: Totalización de datos con campos calculados

Ahora, añadiremos componentes `JdbTextField` para que muestren los datos de las columnas de totalización, `ORIG_TOTAL`, `NEW_TOTAL` y `DIFF_TOTAL`. Estos componentes residirán en un componente `JPanel` independiente dentro del componente `JPanel` que contiene el `JdbStatusLabel`.

Para añadir los componentes `JdbTextField` para los datos de las columnas de totalización:

- 1 Abra la pestaña Contenedores Swing de la paleta de componentes del diseñador de interfaces de usuario, seleccione el componente `JPanel` y pulse sobre el nodo `jPanel2` del panel de estructura.

Esto añade un nuevo componente `JPanel`, `jPanel3`, dentro de `jPanel2`.

- 2 Asigne a la propiedad `layout` de `jPanel3` el valor `GridLayout` y a la propiedad `layout` de `jPanel2` el valor `BorderLayout`.



- 3 Abra la pestaña `dbSwing` de la paleta de componentes, seleccione el componente `JdbTextField` y pulse sobre el nodo `jPanel3` del panel de estructura.

- 4 Asigne a la propiedad `dataSet` de `jdbTextField1` el valor `queryDataSet1`, y a la propiedad `columnName`, el valor `ORIG_TOTAL`.

- 5 Abra la pestaña `Swing` de la paleta de componentes, seleccione el componente `JLabel` y pulse sobre el nodo `jPanel3` del panel de estructura.

Si es necesario, vuelva a colocar el componente `JLabel` (`jLabel2`) en el diseñador de interfaces de usuario para situarlo a la izquierda del componente `jdbTextField1`.

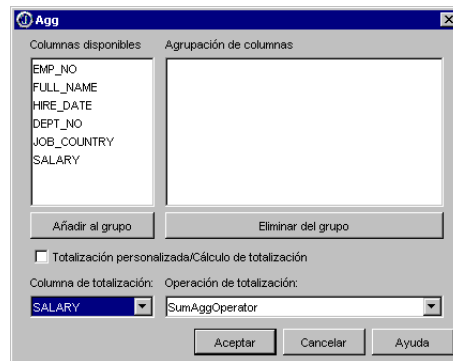
- 6 Asigne a la propiedad `horizontalAlignment` de `jLabel2` el valor `LEADING` y a la propiedad `text` el valor `Original Total`.
- 7 Añada dos componentes más, `JdbTextField` y `JLabel`, y configure sus propiedades según se describe en la tabla siguiente:

Componente	Nombre de la propiedad	Valor
<code>jdbTextField2</code>	<code>dataSet</code>	<code>queryDataSet1</code>
<code>jdbTextField2</code>	<code>columnName</code>	<code>NEW_TOTAL</code>
<code>jLabel3</code>	<code>horizontalAlignment</code>	<code>CENTER</code>
<code>jLabel3</code>	<code>text</code>	<code>New Total</code>
<code>jdbTextField3</code>	<code>dataSet</code>	<code>queryDataSet1</code>
<code>jdbTextField3</code>	<code>columnName</code>	<code>DIFF_TOTAL</code>
<code>jLabel4</code>	<code>horizontalAlignment</code>	<code>CENTER</code>
<code>jLabel4</code>	<code>text</code>	<code>Difference</code>

Si es necesario, ajuste la posición de los componentes en el diseñador de interfaces de usuario.

- 8 Expanda el nodo `queryDataSet2` del panel de estructura y seleccione la columna `ORIG_TOTAL`.
- 9 En el Inspector, seleccione la propiedad `agg`. A continuación, pulse el botón de puntos suspensivos (...) para abrir el cuadro de diálogo Totalizar.

Figura 17.7 Cuadro de diálogo Agg



- 10 Seleccione `SALARY` en la lista desplegable *Columna de totalización*, elija `SumAggOperator` en la lista desplegable *Operación de totalización* y pulse *Aceptar*.
- 11 Seleccione la columna `NEW_TOTAL` en el panel de estructura y abra el cuadro de diálogo *Totalizar*.
- 12 Seleccione `NEW_SALARY` en la lista desplegable *Columna de totalización*, elija `SumAggOperator` en la lista desplegable *Operación de totalización* y pulse *Aceptar*.
- 13 Seleccione la columna `DIFF_TOTAL` en el panel de estructura y abra el cuadro de diálogo *Totalizar*.
- 14 Seleccione `DIFF_SALARY` en la lista desplegable *Columna de totalización*, elija `SumAggOperator` en la lista desplegable *Operación de totalización* y pulse *Aceptar*.
- 15 Seleccione *Ejecutar* | *Ejecutar proyecto* para compilar y ejecutar la aplicación.

La aplicación debería mostrar los datos totalizados en los nuevos componentes `JdbTextField`.

Tutorial: Recuperación de datos mediante procedimientos almacenados

El desarrollo de aplicaciones de base de datos es una función de JBuilder Enterprise

Este tutorial enseña a suministrar datos a una aplicación utilizando el diseñador de interfaces de usuario de JBuilder y un componente `ProcedureDataSet`. El ejemplo muestra también la manera de adjuntar el conjunto de datos resultante a un `JdbTable` y un `JdbNavToolBar` para ver y editar los datos.

En este tutorial se realizarán las siguientes tareas:

- Creación de tablas y procedimientos para el tutorial.
- Incorporación de los componentes `DataSet`.
- Cómo añadir componentes visuales.

La aplicación finalizada se puede ver abriendo el archivo de proyecto de ejemplo, `SimpleStoredProcedure.jpx`, en `<jbuilder>/samples/DataExpress/SimpleStoredProcedure/`. En el directorio `<jbuilder>/samples/DataExpress/ServerSpecificProcedures` existen ejemplos adicionales que muestran cómo utilizar procedimientos almacenados en diversos servidores.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-8](#).

Paso 1: Creación de tablas y procedimientos para el tutorial

En estos pasos se ejecuta un procedimiento almacenado que crea una tabla e inserta, actualiza y borra procedimientos del servidor InterBase (no olvide seguir las instrucciones de instalación de [“Configuración de InterBase e InterClient” en la página 4-7](#)). Este procedimiento está escrito en lenguaje InterBase.

- 1 El servidor InterBase Server e InterServer deben estar ejecutándose en el mismo sistema, si no se ha desconectado.
- 2 Seleccione Archivo | Cerrar todo en el menú para cerrar todos los proyectos.
- 3 Elija Archivo | Abrir y abra el proyecto `ProcedureSetUp.jpx`, que se encuentra en el subdirectorio `/jbuilder/samples/DataExpress/SimpleStoredProcedure/ProcedureSetup` del directorio de instalación de JBuilder.

Si el proyecto no está disponible, o si desea inspeccionar el archivo `CreateProcedures.java`, consulte el apartado [“Creación manual de tablas y procedimientos para el tutorial” en la página 6-4](#).

- 4 Seleccione Proyecto | Propiedades de proyecto.
- 5 Seleccione la ficha Bibliotecas necesarias y elija InterClient.
Esta opción está disponible si se ha establecido la configuración según lo indicado en [“Adición de un controlador JDBC a JBuilder” en la página 4-10](#).
- 6 Haga doble clic en `CreateProcedures.java` en el panel del proyecto y modifique la vía de acceso al archivo `employee.gdb` de InterBase de manera que apunte al archivo instalado en su ordenador. (Utilice barras normales (/) en la vía de acceso.)
- 7 Guarde el archivo, haga clic con el botón derecho en `CreateProcedures.java` en el panel de proyecto, y seleccione Ejecutar.
Este paso crea las tablas y procedimientos en el servidor.
- 8 Elija Herramientas | Explorador de bases de datos para comprobar que se han creado las tablas y procedimientos.
- 9 En el menú, seleccione Archivo | Cerrar proyecto.

Paso 2: Incorporación de los componentes DataSet

Para crear esta aplicación y rellenar un conjunto de datos a partir del procedimiento almacenado:

- 1 Seleccione Archivo | Nuevo y haga doble clic en el icono Aplicación.

Acepte todas las opciones predeterminadas o modifique los nombres de vía de acceso y proyecto para que resulten más descriptivos.

- 2 Abra el cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto) y abra la pestaña Bibliotecas necesarias.

- 3 Añada InterClient.

Esta opción está disponible si se ha establecido la configuración según lo indicado en [“Adición de un controlador JDBC a JBuilder” en la página 4-10](#).

- 4 Cierre el cuadro de diálogo.

- 5 Active el diseñador de interfaces de usuario, seleccionando la pestaña Diseño.

- 6 Seleccione el componente `Database` en la pestaña DataExpress de la paleta de componentes y haga clic sobre cualquier zona del árbol de componentes.

- 7 Abra el editor de la propiedad `connection` del componente `Database`, seleccionando el botón puntos suspensivos de la propiedad `connection` en el Inspector.

- 8 Asigne a las propiedades del componente `connection` las tablas de ejemplo de InterBase, definiéndolas como se indica en la siguiente tabla.

Se presupone que se ha completado [“Configuración de InterBase e InterClient” en la página 4-7](#).

Nombre de la propiedad	Valor
Controlador	<code>interbase.interclient.Driver</code>
URL	<code>jdbc:interbase://<dirección IP o host local>/<vía de acceso al archivo .gdb></code>
Nombre de usuario	<code>SYSDBA</code>
Contraseña	<code>masterkey</code>

El cuadro de diálogo `Connection` contiene un botón **Probar conexión**. Púlselo para comprobar que las propiedades de conexión están correctamente establecidas. El resultado del intento de conexión se muestra junto al botón. Cuando el texto indique **Correcto**, pulse **Aceptar** para cerrar el cuadro de diálogo.

El código que genera el diseñador para este paso se puede ver en `ConnectionDescriptor`, en la ficha Fuente. Seleccione la pestaña Diseño para continuar.

- 9 Coloque, en el panel de contenido, un componente `ProcedureDataSet` de la pestaña DataExpress de la paleta de componentes y configure la

propiedad `procedure` del componente `ProcedureDataSet` de la siguiente forma:

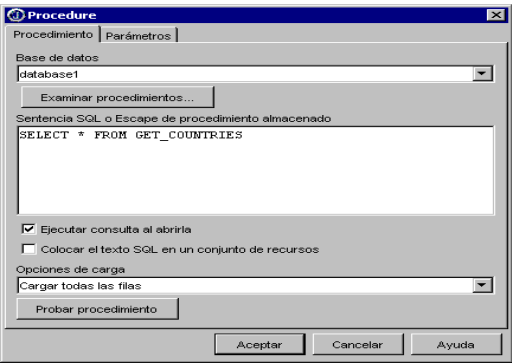
Nombre de la propiedad	Valor
Database	database1
Sentencia SQL o Escape de procedimiento almacenado	SELECT * FROM GET_COUNTRIES

Se crearon varios procedimientos al ejecutar `Createprocedures.java`. El procedimiento `GET_COUNTRIES` es el único que devuelve un conjunto de resultados. La sentencia `SELECT` se utiliza para llamar a un procedimiento en el lenguaje `InterBase`.

Sugerencia

Puede utilizar el botón `Examinar procedimientos` en proyectos futuros para averiguar qué procedimientos almacenados están disponibles. Consulte [“Secuencias de escape, sentencias SQL y llamadas a procedimientos específicos de servidor”](#) en la [página 6-3](#) para obtener más información.

Haga clic en `Probar procedimiento` para cerciorarse de que el procedimiento se ejecuta correctamente. Cuando el espacio gris debajo del botón indique `Correcto`, como se muestra más abajo, pulse `Aceptar` para cerrar el cuadro de diálogo.



El código que se genera para este paso se puede ver en `setProcedure`, en la [ficha Fuente](#).

10 Haga clic en la pestaña `Diseño` para continuar.

Paso 3: Cómo añadir componentes visuales

Este paso muestra cómo crear una interfaz de usuario mediante componentes `dbSwing`.

- 1 Seleccione `contentPane(BorderLayout)` en el árbol de componentes.

- Haga clic en el componente `JdbNavToolBar` de la pestaña `dbSwing` de la paleta de componentes y colóquelo por encima del panel, en el diseñador de interfaces.

`JdbNavToolBar1` se autovincula automáticamente al `DataSet` que tenga el foco.

- Asigne a la propiedad `constraints` de `jdbNavToolBar1` el valor `NORTH`.
- Coloque un componente `JdbStatusLabel` en el área de la parte inferior del panel del diseñador de interfaces de usuario y asigne a su propiedad `constraints` el valor `SOUTH`.

`jdbStatusLabel1` se autovincula automáticamente al `DataSet` que tenga el foco.

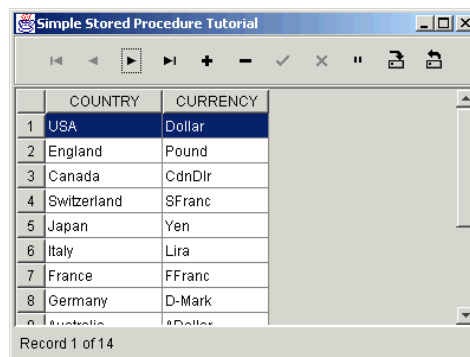
- Coloque un componente `TableScrollPane` de la pestaña `dbSwing` en el centro del panel, en el diseñador de interfaces.

Asegúrese de que la propiedad `constraints` tiene el valor `CENTER`.

- Seleccione `tableScrollPane1` y coloque un `JdbTable` en el centro.
- Asigne a la propiedad `dataSet` de `jdbTable1` el valor `procedureDataSet1`.
- Seleccione **Ejecutar** | **Ejecutar proyecto** para ejecutar la aplicación y examinar el conjunto de datos.

La aplicación en ejecución presentará este aspecto:

Figura 18.1 Recuperación de datos con la aplicación de procedimientos almacenados en ejecución



Por defecto, el botón **Guardar cambios** de la barra de herramientas guarda los cambios con un `QueryResolver`. Para personalizar las funciones almacenadoras en la aplicación para poder editar, insertar y eliminar datos en la aplicación ejecutada, consulte el [Capítulo 8, “Almacenamiento de cambios en la fuente de datos.”](#)

Índice

Símbolos

? como marcador de parámetros JDBC 2-9

A

abrir

- conjuntos de datos 5-21

acceder

- a datos 4-1, 5-1

- desde componentes de interfaz de usuario 13-2

- desde fuentes de datos personalizadas 9-10

- desde un módulo de datos 10-5

- fuentes de datos JDBC 4-1

- a la información de modelo 13-2

actualizar

- datos desde fuentes de datos 5-13

- fuentes de datos 15-6

- tablas S 8-3

administración de bases de datos 15-1, 15-8

administrador de controladores 4-1

agg, editor

- de la propiedad 5-14

agrupar

- datos 4-7

- recursos 10-17

almacenadores

- personalizados 8-8, 8-21, 14-1

- por defecto 8-18

almacenar

- datos 5-13, 8-21

- gestión de errores 8-19

- personalización de la lógica de

- almacenamiento 8-8

- por defecto 8-18

- procedimientos almacenados 8-5

- QueryDataSet 5-4

- relación maestro-detalle 12-11

- varias tablas 8-12

- ejemplo 9-3

- multitabla 8-12

- orden de almacenamiento 8-14

- ProcedureResolver 12-24

almacenar datos

- personalizar sucesos 8-18

añadir

- columnas

- a archivos de texto importados 3-2

- a consultas parametrizadas 5-21

- con fines internos 7-11

componentes

- a módulos de datos 10-5

- parámetros a las consultas 5-19

analizar sintácticamente

- cadenas 5-4

- datos 12-17

API de JDBC 4-1, 5-1

aplicaciones

- base de datos (2 niveles) 10-9

- cliente

- desarrollar con InterClient 2-16

- de base de datos 2-2

- crear 9-2

- distribuidas 14-1

- generar 1-1, 10-20

- introducción 1-1, 2-1

- de dos niveles

- generar 10-20

- de ejemplo

- DataSetData 14-2

- de ejemplo ResolverEvents 8-19, 8-21

- de varios niveles

- distribuir 14-5

- generar 10-20

archivos 3-1, 8-19

- ASCII

- Consulte también* archivos de texto

- de recursos extraídos 10-17

- SCHEMA 3-3

- y exportDisplayMasks 9-8

archivos .config

- crear para controladores 4-11

archivos de texto 3-1

- a fuentes JDBC 2-11

- a tablas SQL 3-3

- con formato 3-3

- importar 3-3

- exportar 3-1, 3-5

- importar 2-14, 3-1

arquitectura

- DataExpress 5-1

ASCII 3-1

asistente para nuevos módulos de datos 15-11

asociar valores de parámetro 5-13

B

base de datos (campo)

- en QueryDescriptor 8-19

bases de datos 1-1, 4-3

- acceder 4-10

- conectarse 4-3, 4-17
 - conexión mediante JDBC 4-9
 - de archivos de sólo texto 2-8
 - desarrollo 1-1
 - en aplicaciones distribuidas 14-1
 - explorar 15-1
 - índices 15-1
 - iu 11-1
 - locales
 - acceder 4-10
 - presentar la información 7-5
 - propiedad de conexión 4-4
 - propiedades (ejemplos) 12-1
 - realizar consultas 10-8
 - relacionales 9-1
 - remotas
 - acceder 4-10
 - conectarse 4-2
 - bibliotecas
 - añadir al proyecto 10-2
 - crear 4-11
 - de componentes de DataExpress
 - descripción 2-1
 - requerido 10-2
 - Borland
 - asistencia
 - a desarrolladores 1-10
 - técnica 1-10
 - contacto 1-10
 - e-mail 1-12
 - informar sobre errores 1-12
 - recursos en línea 1-11
 - World Wide Web 1-11
 - botón
 - Aplicar 10-15
 - Pegar
 - columna 10-15
 - parámetro 10-15
 - buscar datos 8-12
 - de fuentes de datos JDBC 3-3, 9-10
 - optimizar 7-5
 - registros detalle 5-22, 9-2, 9-8
- ## C
-
- cadenas
 - analizar sintácticamente 5-4
 - calcular
 - coste de mercancía 4-8
 - descuentos 4-8
 - impuestos de venta 4-8
 - totales 4-8
 - campos
 - bases de datos
 - explorar 15-1
 - comunes 9-1
 - de datos
 - exportar 11-5
 - de hora
 - exportar 11-5
 - de texto
 - exportar 11-5
 - numéricos
 - exportar 11-5
 - requeridos 9-10
 - vínculos sobre datos comunes 11-12
 - caracteres especiales 12-18
 - cargar datos 13-2
 - cerrar
 - conjuntos de datos 5-21
 - clases
 - DataSetException 8-6
 - LocateOptions 11-17
 - MasterLinkDescriptor 15-8
 - ResolutionManager 5-13
 - RowIterator 7-3
 - SqlRes 10-17
 - Variant
 - encontrar datos 11-20
 - VariantFormatter 12-17
 - clasificar datos 11-9
 - con herramientas de diseño 11-11
 - crear relaciones maestro-detalle 11-12
 - en tablas 11-10
 - orden de clasificación 15-4
 - por programa 11-14
 - cláusula
 - Group By 10-14
 - Order By
 - añadir 4-17
 - Where 10-15
 - codificación de sucesos
 - de módulos de datos 10-21
 - colocar el texto SQL en un conjunto de recursos
 - en QueryDescriptor 10-17
 - columnas 7-1, 7-8
 - añadir a StorageDataSet 7-10
 - calculadas 12-1, 12-8, 12-11
 - creación de listas de selección con 12-2
 - ejemplo 12-9
 - generación de búsquedas con 12-2, 12-5
 - tipos admitidos 12-8
 - totalizar datos 12-11, 12-12, 12-15, 17-1
 - tutorial 17-1
 - cambiar propiedades 7-2
 - clasificar 11-1
 - control del orden 7-11
 - de consulta 12-2

- crear 12-2
- ejemplo 12-5
- definición de propiedades persistentes para 7-7
- establecer propiedades 7-1, 7-2
- explorar 15-1
- filtrar datos 11-1
- localizar datos 11-1
- persistentes 9-10
 - añadir 7-10
 - borrar 7-9
 - control de la actualización de metadatos con 7-9
 - descripción general 7-8
- utilizar 7-1
- valores de consulta. *Consulte* consultas
- vínculos sobre datos comunes 11-12
- visualización de información 7-2, 7-5
- comandos Ejecutar consulta 10-10
- componentes
 - Column 2-9, 7-1, 7-8
 - asignar valores a propiedades 9-10, 12-22
 - guardar objetos de Java en 17-1
 - manipular 9-10
 - persistentes 9-10, 12-22
 - propiedad form 17-1
 - propiedad local 11-5
 - visualizar 12-22
 - Database 4-1, 4-3
 - descripción general 2-6, 4-1
 - ejemplo 4-3
 - utilizar 4-3
 - DataExpress 1-12
 - DataSetView 9-1
 - DataRow 2-9, 5-6, 11-18, 13-5
 - DataSet 2-1, 2-6
 - almacenar objetos Java 17-1
 - clasificar datos 11-1
 - con RMI 8-15
 - filtrar datos 11-1
 - guardar cambios 15-6
 - localizar datos 11-1
 - transportables 8-15
 - DataSetData 5-2, 8-15, 14-1, 14-2, 14-4
 - métodos
 - extractDataSet 8-16
 - extractDataSetChanges 8-16
 - DataSetView 2-8, 7-9, 11-9
 - propiedades 9-1
 - utilizar 12-1
 - DataStore 2-7
 - DataStoreDriver 2-7
 - dbSwing
 - crear interfaces de usuario para bases de datos 2-11
 - utilizar 13-1
- de bases de datos XML 4-2
- enlazados a datos 13-1
 - mostrar columnas en 12-22
 - presentación de datos por defecto 7-2
- JBCL
 - enlazados a datos 13-1
- JdbNavField 11-15, 11-19
- JdbNavToolBar 8-3
- JdbStatusLabel 5-19
- JdbTable 11-10
- JFC 13-1
- JFC enlazado a datos 13-1
- ParameterRow 2-8, 2-9
- ProcedureDataSet 2-2, 2-7, 6-7, 6-9, 11-9
 - almacenar datos 11-12
 - almacenar un ejemplo 8-5
 - guardar cambios 12-24
- ProcedureResolver
 - codificación 9-3
 - guardar cambios 12-24
 - utilizar 8-5, 11-12
- QueryDataSet 2-2, 3-5, 5-2, 5-4, 5-10, 5-15, 8-3, 11-9, 12-5, 16-1
 - almacenar cambios 4-9
 - valor de la propiedad de consulta 10-8
- QueryDescriptor 6-1, 10-17
 - configurar las propiedades visualmente 10-8
 - ficha Consulta 8-19
- QueryProvider
 - para consultas en varias tablas 6-10
- QueryResolver 6-4, 8-3, 8-8, 8-18, 15-6
 - con procedimientos almacenados 11-12
 - guardar cambios 11-12
 - interceptar sucesos 8-18
 - personalizar 5-13, 8-18
- sincronizar 12-26
- SQLResolver 5-13, 6-4, 8-18, 15-6
 - con ProcedureResolver 7-10
- StorageDataSet 1-12, 2-6, 3-1, 7-11
 - guardar cambios 15-6
- TableDataSet 2-8, 3-3, 3-5, 5-10, 11-5
- TextDataFile 2-8, 2-11, 3-5
- conexiones 4-1, 4-17
 - agrupación de conexiones JDBC 4-1
 - con bases de datos
 - tutorial 17-1
 - de base de datos
 - agrupar 4-1
 - seguimiento 15-12
 - descripción general 4-1
- JDBC 17-1
 - descripción general 17-1
 - gestionar 2-6

- manipulación del tráfico 15-12
- seguimiento 15-12
- tutorial 5-12
- problemas y soluciones 4-7
- SQL 17-1
- tutorial 5-12
- conjuntos de datos
 - abrir 13-2
 - apertura explícita 5-21
 - asociar valores de parámetro 5-12
 - cerrar 5-21
 - de sólo lectura 8-15
 - devolver como sólo lectura 8-15
 - mejorar el rendimiento 8-3
 - transportables 8-15
 - utilizar 5-2
 - vincular 9-1
- consultas 5-2
 - cláusula Group By 10-14
 - cláusula Where 10-15
 - componentes necesarios 10-17
 - comprobar 10-10
 - con una cláusula WHERE 10-17
 - crear 7-5
 - crear con el modelador de datos 5-13
 - crear parametrizadas 5-2
 - descripción general 8-12
 - editar directamente 10-19
 - ejecutar 6-1
 - en varias tablas 10-17
 - generar 10-8
 - guardar en módulo de datos 10-17
 - guardar en módulos de datos
 - Java 10-17
 - maestro-detalle 5-12
 - múltiples en el modelador de datos 7-7
 - optimizar 10-16
 - parametrizadas 10-10, 10-15
 - propiedades de columna de varias tablas 6-10
 - SQL 5-2
 - Explorador de bases de datos 6-1
 - ver resultados 10-10
 - verificar la capacidad de actualización 5-13
- consultas parametrizadas 5-19, 10-10
 - añadir columnas 5-21
 - asociar valores 5-13
 - ejemplo 5-22
 - para registros maestro-detalle 5-21
 - proporcionar nuevos valores 5-7
- consultas SQL
 - añadir parámetros 5-19
 - cláusula Group By 10-14
 - cláusula Where 10-15
 - componentes necesarios 10-17
 - comprobar 10-10
 - con recursos 10-17
 - descripción general 8-12
 - editar directamente 10-19
 - guardar en módulo de datos 10-17
 - guardar en módulos de datos
 - Java 10-17
 - maestro-detalle 5-12
 - múltiples en el modelador de datos 7-7
 - optimizar 10-16
 - orden de
 - clasificación 4-17
 - ver resultados 10-10
 - verificar la capacidad de actualización 5-13
- contraseña
 - solicitar 12-19
- control de los datos introducidos por el usuario 4-7
- controladores
 - añadir al proyecto 1-8
 - añadir JDBC 4-11
 - configuración de bases de datos 4-10
 - de base de datos
 - añadir a JBuilder 4-23
 - añadir al proyecto 1-8
 - configurar 4-10
 - totalmente en Java 15-3
 - InterServer 15-3
 - JDBC 4-2
 - añadir a JBuilder 4-11, 4-23
 - añadir al proyecto 1-8
 - configurar 15-3
 - controladores JDBC de JDataStore 2-14
 - especificado en la base de datos 4-4
 - InterClient 2-11
 - ventajas 12-4
- controles 13-1
- convenciones de la documentación 1-10, 12-18
- conversión de cadenas con máscaras 5-4
- coste de mercancía
 - calcular 4-8
- Creador de SQL 10-8
- crear
 - consultas 10-17
 - relaciones maestro-detalle 8-23, 9-11
 - tablas SQL 17-1
- cuadro de diálogo
 - del Asistente para usar módulos de datos
 - explicación 2-1
 - enlazar consultas 5-12
- cursores compartidos 12-26

D

DataExpress

- aplicaciones 2-2
- arquitectura 1-1, 2-1, 5-1
- componentes 2-1, 8-3
 - acceso a datos con 2-10
 - para componentes EJB 8-1

DataModule (interfaz)

- descripción general 4-20

DataRow

- encontrar datos 13-5
- orden de columna en locate 5-6

DataSetData 14-1

datos 5-1

- acceder 9-2
- almacenar 4-9, 12-2, 15-6
 - datos
 - comportamiento por defecto 8-18
- almacenar en caché 12-2
- buscar 11-1, 11-15, 11-18, 11-19, 13-5
- cambiar 15-2
- cargar 13-2
- clasificar 11-1
- de cadena
 - modelos 12-21
- de fecha
 - modelos 12-21
- de hora
 - modelos 12-20
- de tablas
 - editar 15-2
 - visualizar 15-2
- editar 12-26
- explorar 15-2
- exportar 3-5
- filtrar 11-1
- insertar 15-2
- manipular 11-1
- numéricos 3-3
 - importar 3-3
 - modelos 13-1
- persistencia 12-2
- persistentes 12-22
- recuperar 2-10, 5-1, 9-2, 9-10
- relaciones
 - 1 a 10-17
 - 1 a 1 10-17
- requeridos 9-10
- resolver
 - personalizar 8-21
- suministrar 5-1, 13-2
- vista alternativa 7-9
- visualizar 12-26

Datos (ficha)

- Explorador de bases de datos 12-26
- datos (importación y exportación)
 - tutorial 7-2
- DBA 15-8
 - tareas 15-1
- descuentos
 - calcular 4-8
- desplazamiento
 - conjuntos de datos múltiples 12-26
 - sincronizar componentes 12-26
- diseñador de columnas 7-2, 9-10, 12-22
 - activar 7-2
 - botones
 - Generar clase RowIterator 7-3
 - opción de metadatos 7-4, 10-9
- distribuir
 - aplicaciones de varios niveles 14-5
- documentación
 - convenciones 1-10
 - convenciones de plataformas 12-18

E

editar

- datos
 - control de los datos introducidos por el usuario 12-18
 - maestro-detalle 11-6
- tablas SQL 12-16

editar/mostrar máscara 12-1

editMask (propiedad) 12-18

editor

- de la propiedad sort 11-11
- de propiedades de consultas
 - tutorial 17-1

ejemplos

- almacenar las modificaciones en los datos 4-9
- añadir información de estado 12-12
- bases de datos 7-9
- columnas calculadas 12-9
- consultas parametrizadas 5-22
- creación de consultas 12-2
- creación de listas de selección 6-4
- crear procedimientos almacenados 12-24
- datos totalizados con columnas calculadas 8-8
- de bases de datos 5-13
 - almacenar cambios 4-9
 - almacenar ProcedureDataSet 8-5
 - columnas calculadas 12-9
 - configurar JDataStore 2-14
 - consultas parametrizadas 12-5
 - creación de consultas 12-2
 - creación de listas de selección 6-4

- crear procedimientos almacenados 12-24
- filtrar datos 13-5
- relación maestro-detalle 12-12
- ResolverEvents 8-21
- StreamableDataSets 11-2
- totalizaciones calculadas 8-17
- vistas alternativas 7-9
- de FilterRows 6-9
- filtrar datos 13-5
- MasterDetail 12-12
- procedimientos almacenados
 - codificación 9-3
- relación maestro-detalle 12-12
- ResolverEvents 8-21
- TestFrame.java 13-3
- totalizaciones calculadas 8-17
- eliminar
 - columnas persistentes 11-15
 - tablas 12-28
- encontrar datos 11-18, 11-19
 - interactivamente 11-15
 - opciones de localización 11-17
 - orden de las columnas 5-6
 - por programa 13-5
 - variants 11-20
- errores
 - gestión de excepciones 15-10
 - Consulte también* excepciones
- etiquetas de estado
 - añadir a aplicaciones 4-7
- excepciones
 - gestionar 8-6
- Explorador
 - de bases de datos 2-16
 - configurar 4-10
 - Datos (ficha) 12-26, 15-2
 - Introducir SQL (ficha) 6-1
 - utilizar 15-1
 - ventana 7-5
 - visualizar información de las columnas 2-16
 - de JDataStore 3-4
- exportar datos 3-5
 - a archivos de texto 3-1
 - desde un QueryDataSet 16-1
 - utilización de modelos 11-5
- extraer datos 5-1

F

- fechas
 - importar 3-3
- fichas
 - Group By
 - modelador de datos 10-14

- Order by 4-17
- Where
 - modelador de datos 10-15
- filtrar datos 5-1
 - ejemplo 13-5
- flujo de datos 8-15
- formatear datos 12-17
 - máscaras para 7-9
- fuentes
 - Convenciones empleadas en la documentación de JBuilder 1-10
 - de datos 5-13
 - acceder 5-1
 - conectarse 4-17
 - JDBC 1-12, 2-11, 3-3, 9-10
 - de archivos de texto 2-11
 - de datos JDBC 5-1
 - acceder 4-1

G

- generar (aplicaciones de bases de datos) 1-1
- gestionar
 - errores 8-6
 - excepciones 8-6
- grupos
 - de datos 4-7
 - de noticias
 - Borland 1-11
 - public 12-16
- guardar
 - cambios 4-9, 5-13, 8-5
 - a QueryDataSet 5-4
 - relación maestro-detalle 12-11
- datos
 - ejemplo 9-3
 - ProcedureResolver 12-24
 - utilizar QueryResolver 11-12
 - varias tablas 8-12

H

- Hacer persistentes todos los metadatos (opción) 10-9

I

- importar datos 2-14
 - de archivos de texto 3-1
- impuestos de venta
 - calcular 4-8
- índices
 - base de datos 15-1
 - sin repeticiones 4-10

- sin repeticiones e índices con nombre, diferencias 4-10
- información
 - de estado 4-7
 - introducida por el usuario
 - analizar sintácticamente 12-18
 - controlar 4-7
- instalar
 - servidor de JDataStore 6-1
- InterBase
 - configurar para JBuilder 2-10
 - ejemplo de procedimientos almacenados 8-11
 - parámetros de devolución de procedimientos almacenados 12-19
 - sugerencias 12-19
- InterBase e InterClient
 - utilizar con JBuilder 4-14
- InterClient
 - configurar en JBuilder 4-23
 - configurar para JBuilder 2-10
 - controladores JDBC 2-11
 - errores de conexión 4-7
 - instalar 15-3
- interfaces
 - DataModule 2-1, 4-20, 10-5
 - explicación 10-5
 - ResolverListener 8-18
 - RowFilterListener 6-9
- INTERNALROW 8-15, 12-20
- Internet
 - desarrollar aplicaciones cliente/servidor 15-3
- InternetBeans Express 4-7
- Intranet
 - desarrollar aplicaciones cliente/servidor 15-3
- introducción de datos
 - modelos 12-1
- Introducir SQL (ficha)
 - Explorador de bases de datos 6-1

J

- Java
 - controladores de base de datos 2-16
 - objetos con conjuntos de datos 8-15
 - RMI con bases de datos 14-1
- JConnectionPool
 - optimizar rendimiento 2-5
- JDataStore
 - comprobar 2-14
 - controladores JDBC 2-14
 - crear 2-14
 - instalación del servidor local 2-14
 - operaciones 2-14
 - package 8-3

- utilizar 12-2
 - ventajas 12-4
- JDBC 1-1, 4-9
 - agrupación de conexiones 4-1
- JdbNavToolBar
 - guardar
 - datos 8-3
- join de tablas 9-1

L

- lista de
 - consultas 12-1
 - selección 12-1, 12-2
 - ejemplo 12-19
 - eliminar 12-17
- llamadas a procedimientos
 - específicas de servidor 6-7
- Local InterBase Server (servidor InterBase local) 12-19
- locale
 - propiedad 11-5
- localización en varias columnas
 - orden de las columnas 5-6
- lógica comercial 10-21

M

- manejadores de sucesos
 - totalización personalizada 17-1
- manipulación del tráfico JDBC 15-12
- marcadores de parámetros 2-9
- máscaras
 - de edición 12-1
 - añadir 12-20
 - propiedades
 - editMask 12-18
 - de exportación 12-17, 12-20
 - de fecha 5-10
 - ejemplos 10-6
 - de hora 5-10
 - ejemplos 8-14
 - de importación 12-17, 12-20
 - de visualización 7-9, 12-1
 - añadir 12-20
 - para editar 12-18
 - para formatos de datos 7-9
 - para importar/exportar 12-17
- MasterLinkDescriptor (tabla)
 - aspectos generales de su utilización 15-8
- mejoras para extracción de datos 7-5
- MetaDataUpdate
 - propiedad
 - con varias tablas 6-10

- metadatos 7-1
 - actualización en columnas persistentes 5-11
 - buscar 7-1
 - definir como dinámicos 7-4
 - explorar 15-1
 - obtener 6-12
 - persistencia 8-16, 10-9
 - visualizar 7-5

- métodos
 - extractDataSet 5-2, 8-16
 - extractDataSetChanges 8-16
 - insertRow() 13-2
 - locate 13-5
 - provideData 13-2
 - saveChanges() 15-10
 - StorageDataSet
 - insertRow() 13-2
 - startLoading() 13-2

- miembros de datos
 - no transitorios 8-16
 - privados 8-16
 - private 12-20

- modelador de datos
 - aplicaciones cliente/servidor 12-23
 - aplicaciones de dos niveles 12-23
 - crear consultas 7-5

- modelos 12-1, 12-17
 - acceder a la información sobre 10-1
 - booleanos 5-10
 - datos booleanos 10-3
 - datos de cadena 12-21
 - datos de fecha 12-21
 - datos de hora 12-20
 - datos numéricos 13-1
 - de cadena 5-10
 - ejemplos 8-14
 - de datos booleanos 10-3
 - ejemplos 8-14
 - ejemplos 10-6
 - numéricos 5-10
 - ejemplos 10-6
 - para la introducción de datos 4-7

- módulos de datos 10-5
 - añadir
 - a bibliotecas 10-2
 - componentes 10-5
 - lógica empresarial 10-21
 - archivos de clase 15-12
 - asistentes 2-1, 15-11
 - compilar 15-12
 - crear 10-9, 15-11
 - guardar 15-12
 - hacer referencia 2-1, 15-12
 - Java

- guardar consultas 10-17
 - utilizar 2-1, 15-12
 - generado 15-11
- monitor JDBC 15-12
 - en aplicaciones 15-12
 - iniciar 15-13
 - utilizar 15-13

- MonitorButton
 - añadir a la paleta 7-2
 - propiedades 12-15
 - utilizar 11-21

- mostrar
 - caracteres especiales 12-18
 - datos en componentes enlazados a datos 2-5
 - información de estado 4-7

N

- nombre de usuario
 - solicitar 12-19

O

- objetos
 - almacenar 17-1
 - con DataSets 8-15
 - de AggDescriptor 5-14
 - específicos de idioma 10-17
 - Java 17-1

- opciones
 - cascadeDeletes 9-8
 - cascadeUpdates 9-8
 - de carga (campo)
 - en QueryDescriptor 8-19
 - de orden de clasificación seleccionado 4-17
- optimizar la extracción de datos 7-5

- orden de
 - almacenamiento
 - especificar 8-14
 - clasificación 15-4
 - sin repeticiones 4-10
 - clasificación ascendente 4-17
 - clasificación descendente 4-17
 - las columnas
 - encontrar datos 5-6

P

- paquetes
 - DataSet 8-5
 - relacionados con bases de datos 8-5
- ParameterRow 6-3
- parámetros
 - de devolución 12-19
 - especificar 6-1

- return 12-19
- Parámetros (pestaña)
 - QueryDescriptor 6-1
- PARTIAL, opción
 - localización en varias columnas 5-6
- persistencia
 - datos 12-2
- plataformas
 - convenciones 12-18
- Posponer obtención de registros detalle 9-8
- procedimientos
 - almacenados
 - almacenar 8-5
 - crear 6-7
 - de Sybase
 - ejemplo 8-9
 - descripción general 6-9
 - ejemplo 6-7
 - ejemplos 8-9, 8-11, 18-1
 - InterBase 12-19
 - Oracle PL/SQL
 - ejemplo 18-1
 - parámetros de devolución 12-19
 - ProcedureResolver 12-24
 - tutorial 6-9
 - resolver 6-4
- ProcedureResolver
 - propiedades
 - deleteProcedure 7-10
 - insertProcedure 7-10
 - updateProcedure 7-10
- proceso de almacenamiento de cambios
 - controlar 8-18
- propiedades
 - de columna
 - para consultas en varias tablas 6-10
 - presentación de datos 2-5
 - de conexión
 - base de datos 4-4
 - deleteProcedure 9-3
 - enableDelete 9-1
 - enableInsert 9-1
 - enableUpdate 9-1
 - exportDisplayMask 9-8
 - ejemplo 11-5
 - fetchAsNeeded 9-8
 - formatter
 - utilizar 17-1
 - insertProcedure 9-3
 - masterDataSet 8-23
 - masterLink 15-8
 - query 6-1
 - conceptos básicos 8-19
 - editor 10-8

- resolveOrder 8-12, 8-14
- rowID
 - utilizar 15-10
- schemaName 8-12
- storageDataSet 9-1
- tableColumnName 8-12
- tableName 8-12
- updateProcedure 9-3
- proveedores
 - creación personalizada 9-10
 - de datos 5-1
 - personalizados 9-10, 14-1
- ProviderHelp
 - método initData 13-2
- proyectos
 - añadir controladores de bases de datos 1-8

R

- reconciliar datos 15-6
- recuperar datos 2-10, 4-1, 5-1, 9-2, 9-10
 - a través de procedimientos almacenados 8-14
 - desde un módulo de datos 10-5
- registros detalle
 - capturar 9-2, 9-8
- relación maestro-detalle 5-12
 - consultas 5-21
 - crear 8-23, 9-11
 - definición 11-12
 - resolver 12-11
 - personalizada 6-3
- relaciones
 - muchos a muchos 10-17
 - muchos a uno 10-17
 - uno a muchos 9-1, 10-17
- remota
 - servidores 4-1
- ResolverResponse 8-18
- restricciones
 - activar 15-4
 - de datos
 - activar 15-4
- resúmenes de datos 4-7, 8-8, 18-1
- RMI
 - con bases de datos 14-1
 - flujo de datos 8-15

S

- secuencias de escape 6-7
 - de procedimiento de JDBC 14-3
- seguimiento
 - conexiones 15-12
 - controladores JDBC 15-12

- Sentencia SQL (campo)
 - en QueryDescriptor 8-19
- sentencias
 - SQL 6-1, 6-9, 14-3
 - definición 10-8
- serialización de objetos 8-15
- servidores
 - de base de datos
 - comunicación con 2-1
 - de JDataStore
 - instalar 6-1
 - SQL
 - conectarse 4-1
- setResolver 8-8
- SimpleStoredProcedure
 - ejemplo 6-7, 6-9, 11-12
- sincronizar
 - componentes 12-26
- sinónimo
 - visualización de datos en 12-26
- solicitar
 - nombre de usuario y contraseña 12-19
- SQL, bases de datos
 - conectarse 4-17
- SQLResolver
 - utilización con varias tablas 8-12
- StatusEvent listener 8-6
- StreamableDataSets
 - ejemplo
 - ejecutar 11-2
- sucesos
 - añadir lógica empresarial 10-21
 - de almacenamiento 5-13
- suministrar
 - datos
 - con consultas parametrizadas 5-2
 - de fuentes de datos JDBC 9-10
 - para los ejemplos de bases de datos 9-2
 - definición 8-14

T

- tablas
 - borrar 12-28
 - cargar 12-16
 - crear 17-1
 - editar datos 12-26
 - enlazadas 10-17
 - tipos 10-17
 - explorar 15-1
 - guardar cambios 8-3
 - maestras 15-8
 - editar 11-6
 - no actualizable 15-10

- realizar consultas 9-2
- SQL
 - actualizar 15-6
 - borrar 12-28
 - cargar 12-16
 - crear 17-1
 - de archivos de texto 3-3
 - guardar cambios 8-3
 - guardar datos en un archivo de texto 3-3
 - visualización de datos 12-26
- tablas de
 - datos
 - visualización de columnas de vínculo con el
 - detalle 15-10
 - detalle 15-8
 - editar 11-6
- tipos de datos
 - Variant.OBJECT
 - en columnas 17-1
 - variantes 17-1
- totales
 - calcular 4-8
- totalizar datos 12-1
 - crear columnas totalizadas 12-11, 12-15
 - ejemplo 8-17, 10-11
 - personalizar métodos de totalización 17-1
- transacciones 4-9
 - proceso por defecto 15-6
- tutoriales
 - aplicación básica de base de datos 17-1
 - conexiones JDBC 10-11
 - datos totalizados con columnas calculadas 18-1
 - de bases de datos
 - conexiones JDBC 10-11
 - totalizaciones calculadas 10-11
 - importación y exportación de datos desde un
 - archivo de texto 7-2
 - totalizaciones calculadas 10-11

U

- Usenet, grupos de noticias 12-16
- utilizar 7-9

V

- ValidationException 8-6
- velocidad
 - mejorar el conjunto de datos 8-3
- vincular conjuntos de datos 9-1
- vistas
 - de datos 2-8
 - visualización de datos en 12-26