SERVIDORES DE BASES DE DATOS

Lo que veremos en este capítulo es sólo la "punta del iceberg" de lo que se puede hacer con un motor de base de datos y el modelo de objetos ADO. Nos concentraremos en el uso de SQL Server como motor porque, así como Visual Basic es el lenguaje más popular de la actualidad, SQL Server se perfila como el más popular entre los motores "grandes" de bases de datos. Y no tiene nada que envidiarles a sus competidores.

Capítulo 5

Visual Basic y SQL Server

En su versión 7, SQL Server ha crecido en dos sentidos: por un lado, incorporó funcionalidad, performance y confiabilidad para cubrir las necesidades de grandes empresas con sistemas igualmente grandes; por el otro, se adaptó al entorno de escritorio a través de una versión que –manteniendo intacta casi toda su funcionalidad– es capaz de correr en Windows 95/98. Esto lo convierte en el compañero ideal de Visual Basic, el lenguaje que se aplica en entornos igualmente variados.

En los próximos párrafos veremos cómo se manejan las herramientas que SQL Server pone a disposición del programador, y cómo se interactúa con él a través de Visual Basic y ADO.

EN EL CD

VERSIÓN TRIAL DE SQL SERVER 7

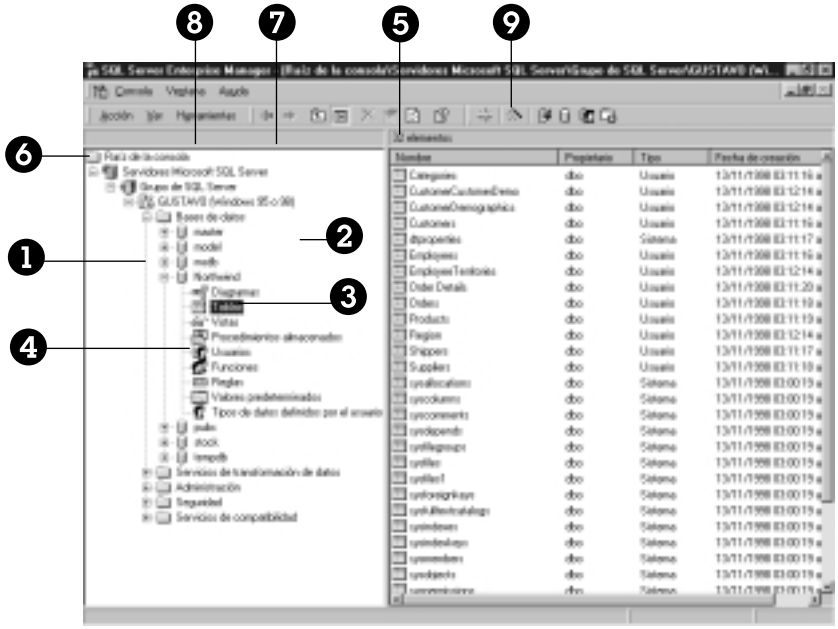
La versión trial de SQL Server 7 le permitirá –por un tiempo limitado– poner a prueba todos los ejemplos que se realizarán en este capítulo con dicho motor de bases de datos.

Herramientas de SQL Server

SQL Server 7 brinda un conjunto de aplicaciones que permiten administrar e interactuar con uno o más servidores y con las bases de datos que residen en ellos. La más importante de ellas es el **Administrador corporativo**, una "consola" con interfase gráfica y amigable que permite llevar a cabo cualquier tarea relativa al manejo de las bases de datos. Otras herramientas que veremos a continuación son el **Analizador de consultas**, una aplicación utilizada para enviar *scripts* (secuencias de comandos) SQL al servidor para su ejecución y análisis, y el **Administrador de servicios**, un pequeño programa que permite poner en marcha el "motor" de SQL y otros servicios relacionados. Con estos elementos, cualquier programador cuenta con más de lo necesario para operar cómodamente con un servidor de bases de datos SQL Server.

El Administrador corporativo presenta una interfase de usuario dividida en dos "*frames*": a la izquierda muestra un árbol que contiene todos los elementos que puede manejar el Administrador, y a la derecha contiene el detalle del objeto seleccionado en el **frame** de la izquierda.

GUÍA VISUAL Nº 1



❶ Árbol de consola. Contiene todos los objetos administrables (desde grupos de servidores y servidores registrados hasta tablas y vistas) organizados de forma jerárquica.

❷ Servidores registrados. Desde el administrador se puede manejar más de un único servidor, y se debe registrar cada uno.

❸ Bases de datos. Figuran todas las que se encuentran en un servidor determinado.

❹ Objetos que componen las bases de datos. Se incluyen diagramas, tablas, vistas, procedimientos, usuarios, funciones, reglas, valores predeterminados y tipos de datos personalizados.

❺ Barra de descripción. Brinda información adicional sobre el conjunto de objetos del frame de detalle.

❻ Menúes estándar. Abarcan las opciones dependientes de Acción y Ver.

❼ Botones estándar. Incluyen flechas para avanzar y retroceder (estilo Internet Explorer), para subir un nivel, para mostrar u ocultar el árbol de la consola, para abrir la ventana de propiedades, para actualizar el contenido de la ventana y para solicitar ayuda.

❽ Menú de complementos. Es el que depende de la opción Herramientas.

❾ Botones de complementos. Incluye botones para crear un nuevo objeto (del tipo seleccionado en el frame de la derecha), para ejecutar un asistente, para registrar un nuevo servidor, para crear una nueva base de datos, para establecer un nuevo inicio de sesión y para agendar una tarea o trabajo (en inglés, job).

Cuando se desea manejar, mediante el Administrador corporativo, un servidor SQL Server 7 instalado en una máquina remota, lo primero que hay que hacer es registrar este servidor en el árbol de la consola (en cambio, cuando el servidor corre en forma local, **aparece registrado automáticamente**). Esta tarea es sumamente sencilla. En primer lugar, en el menú **Acción**, se selecciona la opción **Nuevo registro de servidor SQL Server...**. Con esto se iniciará el Asistente para registro de servidor SQL Server (**Figura 1**).

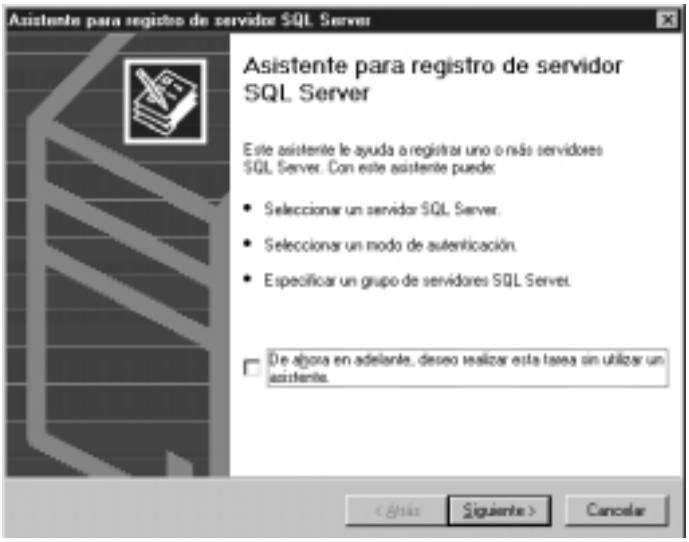


***Figura 1**. El Asistente para registro de servidor SQL Server da la posibilidad de anularlo para que en el futuro se pueda realizar esta tarea sin su ayuda.*

Cliqueando en el botón **Siguiente** de la primera pantalla del asistente aparece otra pantalla en la cual se debe ingresar el nombre del servidor que se desea registrar. Éste debe haber sido previamente definido mediante la **Herramienta de red de cliente** de SQL Server 7, especificando el protocolo de red a utilizar y demás parámetros de comunicación (que el administrador de la red sabrá indicar). Suponiendo que vamos a registrar un servidor denominado "ADMINSUELDOS", en el cuadro de texto que encabeza la lista titulada **Servidores disponibles** deberíamos ingresar este nombre –si es que no figura en la lista– y luego cliquear en **Agregar >** para que pase a la lista de **Servidores agregados** (**Figura 2**). Si hubiera más de un servidor disponible, se podrían registrar varios de una sola vez, pasándolos a la lista de la derecha antes de cliquear en **Siguiente**.

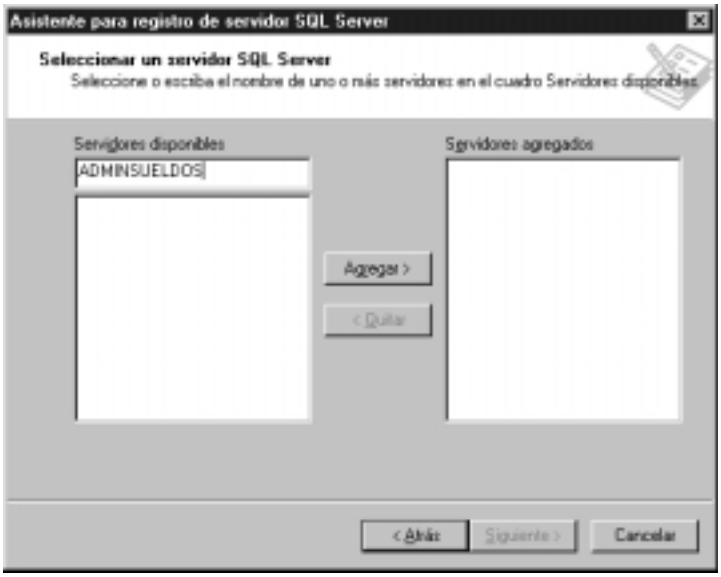


***Figura 2**. En un entorno de red donde se hayan definido múltiples servidores de bases de datos, la lista de la izquierda aparecerá poblada por todos sus nombres; basta con elegir uno o más y pasarlos a la lista de la derecha para efectuar su registro.*

Al proceder al siguiente paso, el asistente solicita el **modo de autenticación** que se utilizará cada vez que se deba conectar al servidor. La elección en este caso depende de las políticas de seguridad que se hayan establecido en la red. Pero generalmente (incluso en el caso de que el servidor corra localmente) se debe seleccionar la segunda opción, que indica que se utilice **Autenticación SQL Server**, o sea, que el nombre de usuario y password que se empleen para conectarse a la base se validarán contra la lista de usuarios habilitados por SQL Server (la otra opción determina que se validen contra los usuarios habilitados por Windows NT).

Hecho esto, se puede cliquear en **`Siguiente`** y pasar al próximo paso, en el cual se debe determinar si la conexión se hará automáticamente con un nombre de usuario y password específicos o si éstos se solicitarán cada vez que se haga la conexión. En caso de que se elija la primera opción, se puede optar por seleccionar el usuario SA –por *system administrator*, se supone–, sin password, que se define por defecto en las instalaciones nuevas de SQL Server (**Figura 3**). Lógicamente, si la base de datos requiere un mínimo de seguridad para evitar el acceso de usuarios no autorizados, la primera medida a tomar

consiste en crear un nuevo usuario con los mismos derechos que SA, pero con otro nombre y otra password, e inmediatamente eliminar el usuario SA.

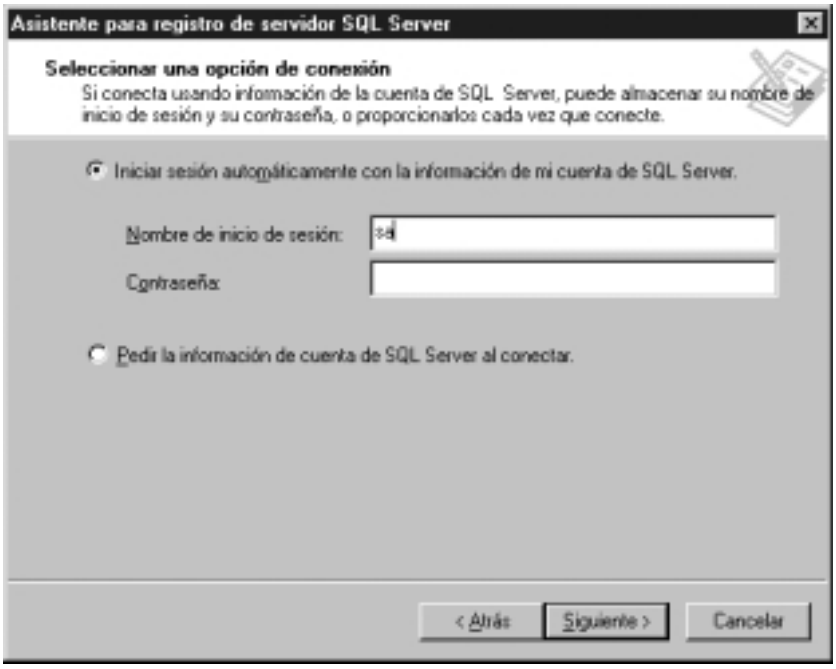


***Figura 3**. Para bases de datos que no requieren un control estricto del acceso, conviene seleccionar la opción de inicio automático, y elegir SA como nombre de usuario (sin password) para "loguearse" siempre a la base con los máximos privilegios.*

En el paso siguiente hay que seleccionar un grupo de servidores bajo el cual se colocará el nuevo registro de servidor. Esta agrupación es útil en empresas que cuentan con gran cantidad de servidores reunidos por sectores (por ejemplo, Producción, Ventas, Administración, etc.). Salvo estos casos, se puede registrar el nuevo servidor bajo el grupo que se crea por defecto; si no aparece ninguno, se puede crear un grupo dándole el nombre que uno desee (**Figura 4**).

***Figura 4**. La utilización de conjuntos de servidores cumple la única función de facilitar la administración en redes con gran cantidad de servidores agrupados por sectores.*

En la última pantalla del asistente se muestra la lista de servidores a agregar y el botón **`Finalizar`**. Al cliquear en él, el asistente intentará registrar el o los servidores seleccionados. Puede que se produzca un error durante el proceso de registración. En la mayoría de los casos, el origen de la falla será un problema en la especificación de las direcciones de red y los protocolos a utilizar, y demás parámetros de comunicación. Para solucionarlo, hay que recurrir a la Herramienta de red de cliente y examinar la configuración de la misma con ayuda del administrador de la red.

Una vez registrado el servidor, se puede cliquear en el "+" a la izquierda del ícono que lo representa en el árbol de la consola para desplegar todos sus elementos. Luego, haciendo clic en el botón "+" a la izquierda de la carpeta **Bases de datos** aparecen todas las bases definidas dentro de este servidor.

El **Administrador de servicios** es una pequeña aplicación que permanece activa en la Bandeja de íconos de la barra de Tareas de Windows de la computadora donde está instalado SQL Server. Haciendo doble clic en este ícono aparece la ventana del Administrador de ser-

vicios. En ella es posible seleccionar cualquiera de los servicios de SQL Server para ver su estado o para iniciarlo, detenerlo o ponerlo en pausa (**Figura 5**).

***Figura 5**. Mediante el ícono del Administrador de servicios en la barra de Tareas se activa la ventana del mismo. Una vez abierta, se puede utilizar para ver el estado de los servicios de SQL Server, así como iniciarlos, detenerlos o ponerlos en pausa.*

Normalmente, los servicios disponibles son el servidor de bases de datos en sí mismo (MSSQLServer), el coordinador de transacciones distribuidas (MSDTC) y el agente de SQL Server (SQLServerAgent). Basta con iniciar el servicio MSSQLServer para que estén disponibles todas las bases del servidor.

El **Analizador de consultas** es una interfase desde la cual podrá ejecutar directamente cualquier instrucción o secuencia de instrucciones SQL contra una base de datos existente en cualquier servidor disponible. Al entrar en el Analizador, se abre una pantalla previa que brinda la posibilidad de establecer una conexión con un servidor (**Figura 6**).

***Figura 6**. La ventana preliminar del Analizador de consultas permite elegir el servidor con el cual conectarse y la forma de autenticación a utilizar.*

En ella se establece el nombre del servidor con el que se desea conectar y la información de autenticación. En el caso de que el servidor esté instalado localmente, en el campo del nombre habrá que ingresar **"(local)"**; luego, si no se eliminó el usuario por defecto de la lista de usuarios de SQL Server, se puede colocar "**sa**" como nombre de inicio de sesión y dejar vacío el campo de contraseña.

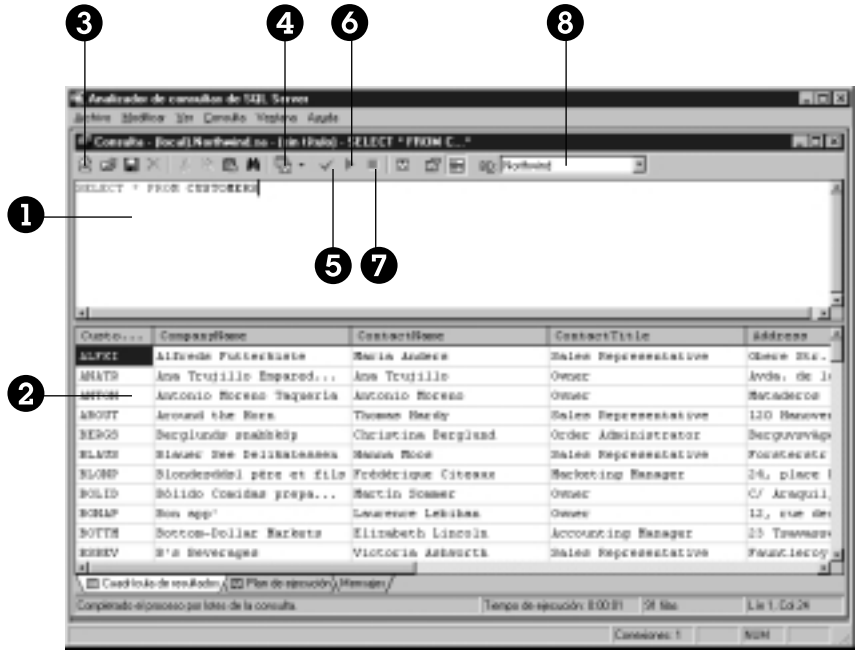
HAY QUE SABERLO

ASEGÚRESE DE QUE ESTÉ CORRIENDO

Si el servicio **MSSQLServer** no está "levantado" en la máquina correspondiente, el Analizador de consultas no se podrá conectar. Por lo tanto, antes de intentar conectarse, asegúrese –mediante el Administrador de servicios– de que el SQL Server esté funcionando, ya sea en la máquina local o en una remota.
También es posible iniciar el servicio directamente desde el Administrador corporativo.

Una vez conectado, el Analizador de consultas despliega su ventana principal. En la siguiente guía visual se detallan los principales elementos de esta ventana.

GUÍA VISUAL Nº 2



❶ Panel de edición. En él se ingresan las secuencias de instrucciones SQL. Para ejecutar sólo una parte de las instrucciones que figuran en este panel, basta con seleccionar esa porción antes de dar la orden de ejecución.

❷ Paneles de resultados. Aquí aparecen los resultados de la consulta luego de la ejecución de la misma, así como los mensajes que pudieran haber surgido en el proceso. Según las opciones seleccionadas, los resultados pueden aparecer en forma de texto plano o en una cuadrícula, y puede mostrarse o no el plan de ejecución armado por SQL Server para resolver la consulta.

❸ Botones **Nuevo**, **Abrir**, **Guardar** y **Borrar**. Con ellos se puede abrir una ventana con una nueva consulta o una consulta existente, guardar el código SQL o los resultados de la consulta actual, y borrar la ventana de edición de la consulta actual.

❹ Botón de modo de ejecución y de opciones de presentación. Permite determinar si los resultados se van a mostrar en forma de texto o en forma de grilla, y si se va a mostrar o no el plan de ejecución.

❺ Botón **Analizar**. Analiza el código SQL de la consulta para ver si tiene errores. Equivale a presionar **Ctrl + F5**.

❻ Botón **Ejecutar**. Inicia la ejecución de la consulta. Equivale a presionar **F5**.

❼ Botón **Detener**. Cuando una consulta está en proceso, este botón se activa, dando la posibilidad de parar la ejecución. Equivale a presionar **Alt + Enter**.

❽ Lista de bases de datos disponibles. Permite elegir la base de datos sobre la que se ejecutará la consulta.

Para poner a prueba el Analizador de consultas, seleccione la base de datos **Northwind** (si no fue borrada, ya que es la base de datos de ejemplo que se crea al instalar el servidor) y ejecute la siguiente instrucción:

```
SELECT * FROM CUSTOMERS
```

MÁS DATOS

A FALTA DE SQL SERVER...

Si no cuenta con SQL Server pero tiene el Office 2000 Premium, igual podrá aprovechar el motor de bases de datos de SQL Server, denominado MSDE (Microsoft Data Engine). Se instala a partir del primer CD de Office 2000 Premium, al ejecutar el programa **SetupSQL.exe** de la carpeta **SQL\X86\SETUP**.

Al emplear esta alternativa, no contará con el Administrador corporativo ni con el Analizador de consultas, pero podrá usar Access 2000 en su lugar. MSDE carece, además, de algunas de las funcionalidades de replicación de datos que tiene SQL Server.

Para trabajar con una base de datos de SQL Server o MSDE desde Access, se debe elegir la opción **Nueva...** del menú **Archivo** en Access. Luego, en la ventana de opciones para crear una nueva base de datos, seleccionar **Proyecto (base de datos existente)** (**Figura 7**).

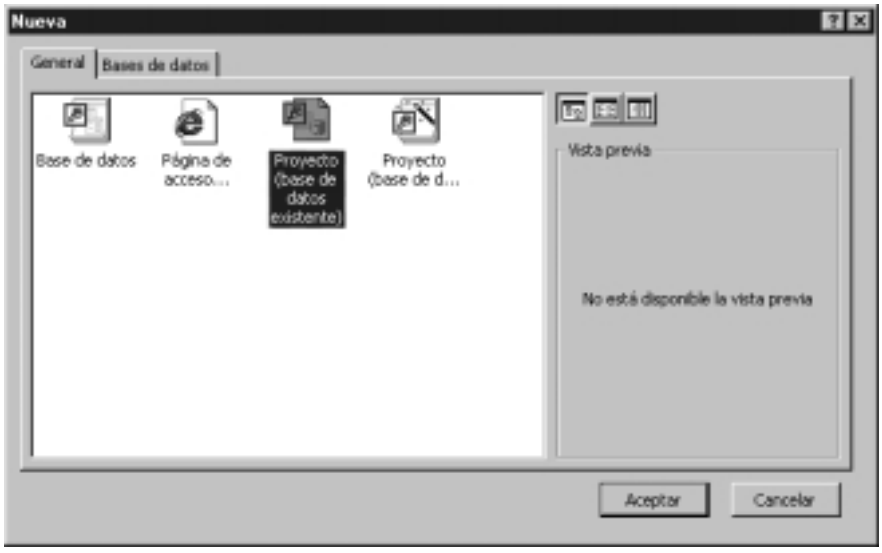


***Figura 7**. La opción de crear un nuevo proyecto de Access con una base de datos existente permite trabajar con una base en un servidor SQL Server o MSDE.*

Luego se le debe asignar al proyecto un nombre cualquiera, que se almacenará como un archivo con extensión **adp**. Hecho esto, aparece la ventana **`Propiedades de Data Link`** (**Figura 8**). En ella se establece el nombre del servidor y la información de autenticación –igual que lo visto anteriormente para el Analizador de consultas de SQL Server–, y se elige la base de datos con la que se va a trabajar.

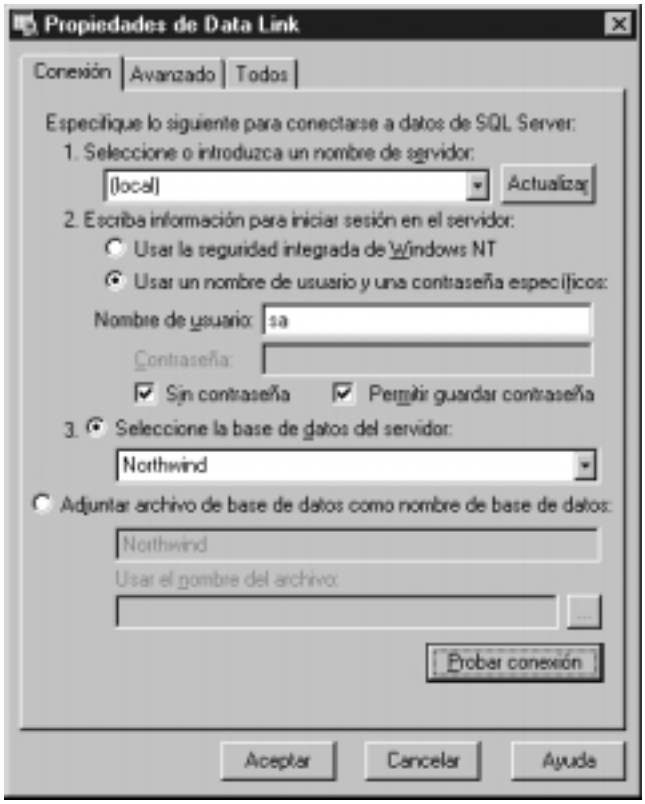


***Figura 8**. Los datos solicitados para conectarse con un servidor SQL Server o MSDE siempre son los mismos, sin importar desde dónde intente conectarse: nombre del servidor, información de autenticación y base de datos a utilizar.*

Si tiene MSDE instalado localmente, puede colocar **"(local)"** como nombre de servidor, "**sa**" como nombre de usuario (sin password) y "**Northwind**" como la base de datos a utilizar. Recuerde que, para poder establecer la conexión, el servicio debe estar activo (puede activarlo con el Administrador de servicios, que viene incluido con MSDE al igual que con SQL Server).

Luego de establecidos todos estos datos, podrá comenzar a utilizar la base de datos seleccionada directamente desde Access. Encontrará que tiene varias de las funcionalidades básicas tanto del Administrador corporativo como del Analizador de consultas de SQL Server, lo cual se combina con las opciones que brinda normalmente Access para toda MDB. Por ejemplo, puede originar y modificar tablas, vistas y diagramas de bases de datos; crear, ejecutar y modificar *stored procedures*; realizar informes, formularios y páginas de acceso a datos, etc. (**Figura 9**).

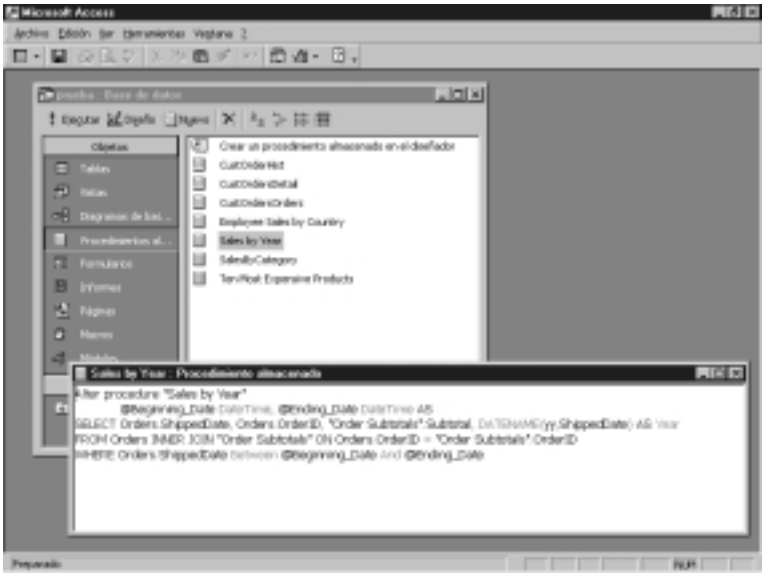


Figura 9. *Access combina la funcionalidad normal para sus bases "nativas" (MDB) –como creación de formularios, módulos y páginas de acceso a datos– con funciones propias del Administrador corporativo y Analizador de consultas de SQL Server.*

Utilizar la vista de datos

Visual Basic 6 dispone de una herramienta, denominada **vista Datos**, que permite manipular bases de datos para crear o modificar diagramas, tablas, vistas y *stored procedures*. La herramienta se activa seleccionando la opción **`Ventana de la vista Datos`** del menú **`Ver`**, o bien cliqueando en el ícono correspondiente de la barra de Herramientas estándar (**Figura 10**).

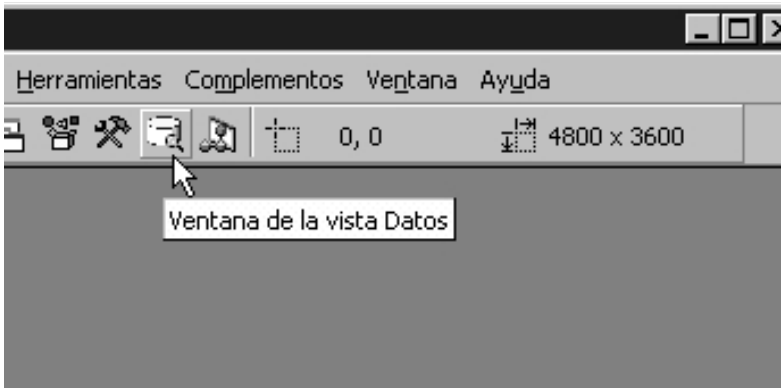


***Figura 10**. La ventana de la vista Datos brinda acceso a herramientas de desarrollo que actúan directamente sobre la base de datos.*

Una vez abierta la ventana de la vista Datos, se deben crear **vínculos de datos** para las bases que se vayan a utilizar. Las alternativas en materia de posibles fuentes de datos abarcan a toda aquella fuente que cuente con un **proveedor OLE DB** asociado. Esto incluye a las bases de datos de Access (ya sea Access 2000 o anteriores), SQL Server (o MSDE), Oracle, servicios OLAP y muchos más, entre los que figuran todas las alternativas que ofrece ODBC.

Por ejemplo, supongamos que necesitamos crear un vínculo de datos para la base de ejemplo **Northwind** en un servidor SQL Server o MSDE. Para ello, primero cliqueamos en el ícono correspondiente en la parte superior de la ventana de la vista Datos (**Figura 11**).

***Figura 11**. En la ventana de la vista Datos se pueden crear tantos vínculos de datos como se desee, incluso relacionados con distintas bases de datos, para tenerlos siempre "a mano" durante el desarrollo de un proyecto.*

Aparecerá una ventana para configurar el vínculo de datos, titulada **`Propiedades de Data Link`**, que es bastante similar a la que muestra Access para crear una conexión con una base de MSDE o SQL Server. Esta ventana cuenta con dos secciones principales: **`Proveedor`** y **`Conexión`** (las otras dos son irrelevantes en esta instancia).

En la sección **`Proveedor`** se debe seleccionar el proveedor OLE DB que se utilizará para la conexión. Para este caso particular, seleccionamos **`Microsoft OLE DB Provider for SQL Server`** (**Figura 12**).

***Figura 12.** La ventana **`Propiedades de Data Link`** aparecerá siempre que se desee establecer una conexión a bases de datos a través de OLE DB.*

Luego pasamos a la segunda sección, donde se ingresan los consabidos parámetros de conexión: nombre del servidor (ingresar **"(local)"** si está instalado en la máquina local), datos de autenticación y base de datos a utilizar. Una vez ingresados estos parámetros y probada la conexión, se puede cliquear en **`Aceptar`** para crear el vínculo de datos. El paso siguiente consiste en darle un nombre a la nueva conexión, para lo cual se puede utilizar el mismo nombre de la base de datos a emplear (**Figura 13**).

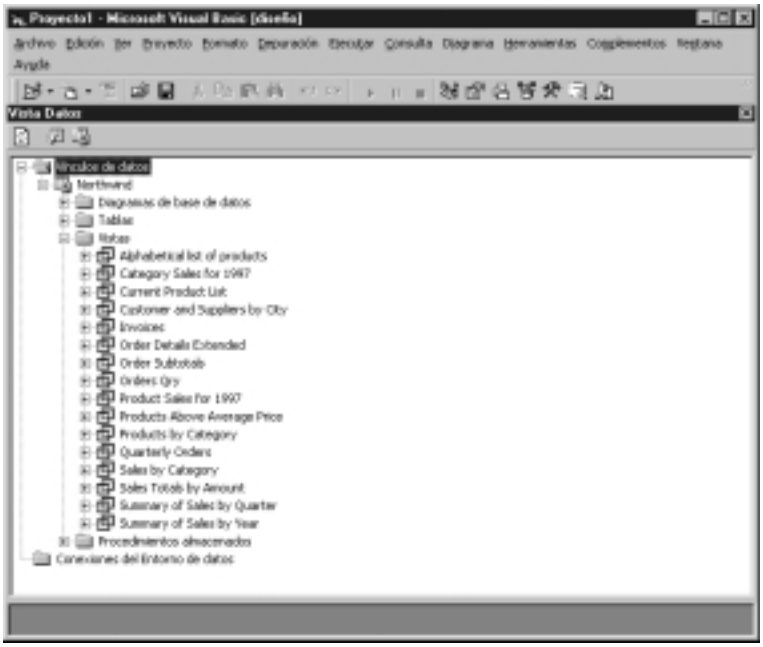


***Figura 13**. Una vez establecida la conexión con la base de datos, todos sus elementos aparecen en el entorno de Visual Basic de la misma manera en que se los ve en la herramienta propia de la base de datos en cuestión.*

A partir de entonces se pueden utilizar los signos "+" a la izquierda de cada nodo del árbol para expandir sus elementos. Por ejemplo, las vistas. Al cliquear con el botón derecho en los elementos desplegados aparecen distintos conjuntos de acciones posibles, que dependen de la naturaleza del elemento. Para las vistas, por ejemplo, se ofrece la opción de abrirla (para ver su conjunto de resultados), modificarla, eliminarla o examinar sus propiedades. También se brinda la opción de agregar un nuevo elemento del grupo elegido; ya sea una tabla, una vista, un *stored procedure* o un diagrama.

MÁS DATOS

COMO SI ESTO FUERA POCO...

La vista Datos brinda, en algunos casos, una funcionalidad que ni siquiera se encuentra en el Administrador corporativo de SQL Server. El editor de *stored procedures* es un ejemplo. La ventana del editor es bastante más práctica que la que ofrece el Administrador corporativo (**Figura 14**), y da la posibilidad adicional de depurar el código con ejecución paso a paso, seguimiento de variables, etc.

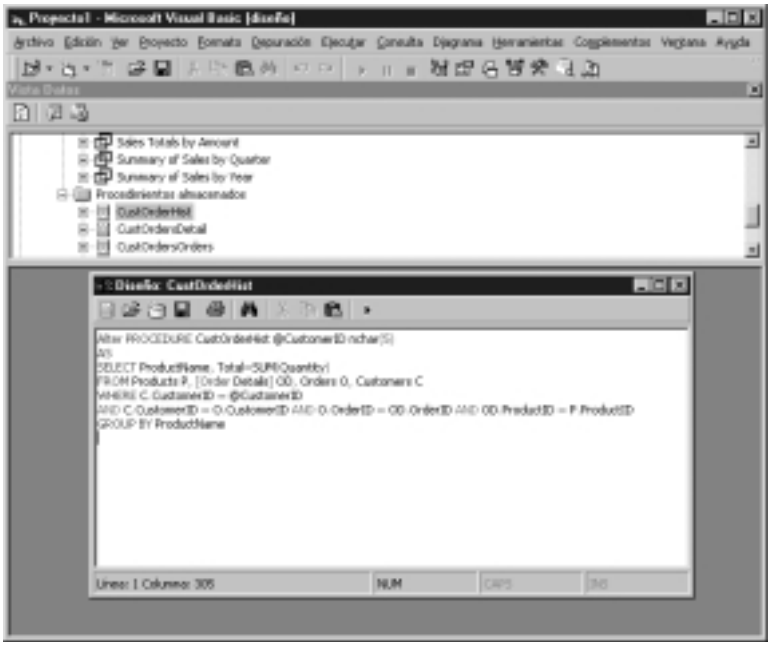


***Figura 14**. La ventana de edición de* stored procedures *que ofrece la vista Datos de Visual Basic 6 resulta bastante más práctica que la que incluye el Administrador corporativo.*

Creación de una base de datos en el servidor

Hasta ahora hemos visto ejemplos que utilizan la base de datos **Northwind**, que ya viene creada tanto en SQL Server como en MSDE. Nos falta ver cómo se hace para **crear** una base de datos en un servidor.

La manera más sencilla es utilizar el Administrador corporativo. Con esta herramienta, basta con seleccionar el nodo **`Bases de datos`** de cualquier servidor registrado en el árbol de la consola y luego ejecutar la opción **`Nueva base de datos`** del menú **`Acción`**. Con esto, aparecerá una ventana de propiedades de la base de datos a crear. En ella, simplemente se indica el nombre de la nueva base, el tamaño (en MB) que ocupará en disco, las restricciones de crecimiento y las características del archivo a utilizar para registro de transacciones (**Figura 15**).

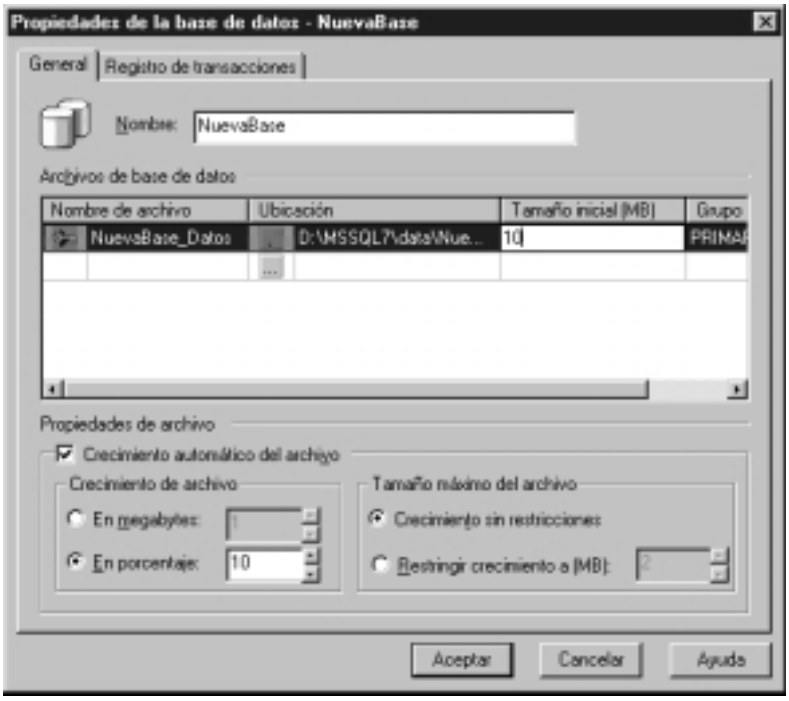


***Figura 15**. El Administrador corporativo de SQL Server sólo requiere unos pocos parámetros para crear una nueva base de datos.*

Si no se cuenta con el Administrador corporativo (por ejemplo, en caso de que se utilice MSDE en vez de SQL Server), se puede utilizar Access 2000 para crear la nueva base en el servidor. Con Access 2000 habrá que originar un **`proyecto para una base de datos nueva`**. Esta opción es una de las que se muestran al elegir la opción **`Archivo/Nueva...`** del menú principal del programa.

Luego de darle un nombre al proyecto, se inicia el **Asistente para creación de bases de datos de Microsoft SQL Server** (**Figura 16**). En unos pocos y sencillos pasos, este asistente lo guiará en la tarea de crear una nueva base en el servidor.

***Figura 16**. El Asistente para creación de bases de datos de Microsoft SQL Server recorre todos los pasos necesarios para cumplir la tarea de crear una nueva base de datos en un servidor SQL Server o MSDE.*

Otra opción consiste en usar el **servicio de transformación de datos**. Con esta herramienta podrá crear una nueva base en el servidor con la misma estructura y los mismos datos de una base preexistente, que puede provenir de otro servidor SQL Server o de un servidor de otra marca (DB/2, Informix, Oracle o Sybase), de una MDB de Access, de una base de datos Paradox, de un conjunto de archivos DBF, de un archivo de Excel o de archivos de texto plano, entre otras opciones. Esta utilidad se incluye tanto con SQL Server como con MSDE. En el menú de Windows, normalmente figura como **`Importar y exportar datos`** (**Figura 17**).

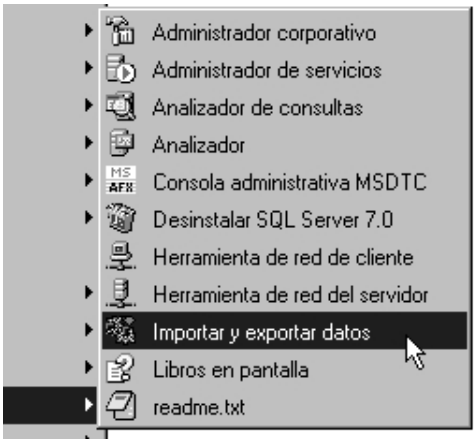


Figura 17. *El servicio de transformación de datos figura entre las opciones del menú de SQL Server como* ***Importar y exportar datos****.*

Para hacer un ejemplo –que, de paso, nos servirá para posteriores ejercicios de este capítulo–, vamos a migrar la base de Access **Videoclub.mdb** (utilizada en los ejercicios del capítulo anterior) a una base de SQL Server o MSDE.

Migración de una base de Access a SQL Server — PASO A PASO

1. Abra el Asistente para transformación de datos, seleccionando la opción **Importar y exportar datos** del menú de Windows (en el grupo de programas **Microsoft SQL Server 7**). Se desplegará la primera ventana del Asistente (**Figura 18**), que explica cuál es su función y permite cliquear en **Siguiente** para pasar al próximo paso.

Figura 18.

❷ En la segunda ventana del Asistente se debe definir cuál es el origen de los datos para la transformación. En la lista desplegable titulada **`Origen`** seleccionamos la opción **`Microsoft Access`**. Al hacer esto, el *frame* inferior de la ventana cambia para mostrar el conjunto de datos requeridos para acceder a la base de Access; o sea: **`Nombre de archivo`**, **`Nombre de usuario`** y **`Contraseña`** (**Figura 19**). Dado que vamos a utilizar la base del ejercicio anterior, en el campo **`Nombre de archivo`** ingresamos **`C:\EjerVB\Videoclub\Videoclub.mdb`**, y dejamos vacíos los campos **`Nombre de usuario`** y **`Contraseña`**. Hecho esto, estamos en condiciones de cliquear en **`Siguiente`**.

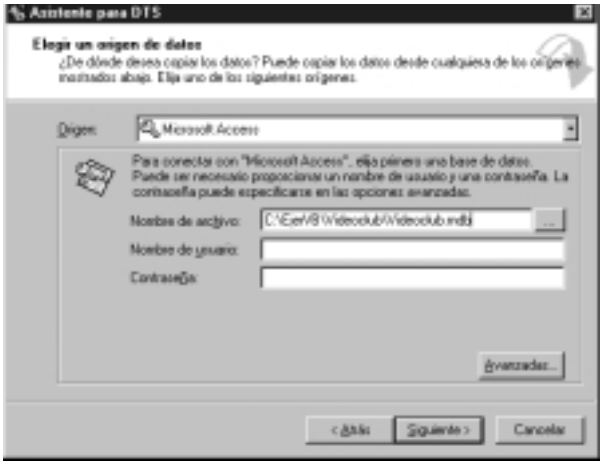


Figura 19.

❸ En el tercer paso se nos solicita el destino de los datos. Dado que la opción por defecto –**Microsoft OLE DB Provider for SQL Server**– es la que usaremos, vamos a ingresar directamente la información solicitada en la parte inferior de la ventana; los consabidos nombre de servidor y datos de autenticación. La diferencia con las ocasiones anteriores en las que establecimos una conexión con el servidor consiste en que esta vez le pediremos que nos conecte con una base nueva, no con una preexistente. Para hacer esto, en la lista desplegable titulada **Base de datos** seleccionamos la opción **<nuevo>** (**Figura 20**).

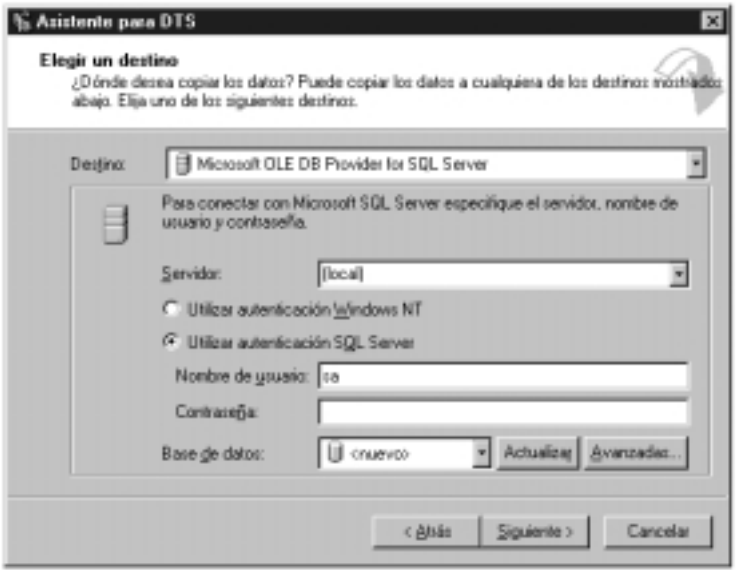


Figura 20.

❹ Inmediatamente se abrirá una ventana titulada **Crear base de datos** (**Figura 21**). En ella –al igual que en las opciones vistas antes para crear bases en el servidor– se solicita el nombre para la nueva base y los tamaños iniciales de los archivos de datos y de registro. Si bien estos tamaños son flexibles (es decir, SQL Server expandirá los archivos a medida que le vayan quedando chicos), es bueno hacer una estimación del tamaño requerido en función de la suma de los tamaños de cada tabla, calculando el tamaño de registro y multiplicándolo por el número de registros en la tabla. A eso hay que agregarle un espacio prudencial para almacenar índices.

Para el ejemplo que nos ocupa, ingrese **Videoclub** como nombre para la nueva base, y asígnele un espacio de **5 MB** para el archivo de datos y **1 MB** para el archivo de registro. Es habitual asignar al archivo de registro (que contiene las transacciones en curso hasta el momento en que se les hace un *commit*) un 20 por ciento del tamaño establecido para los datos.

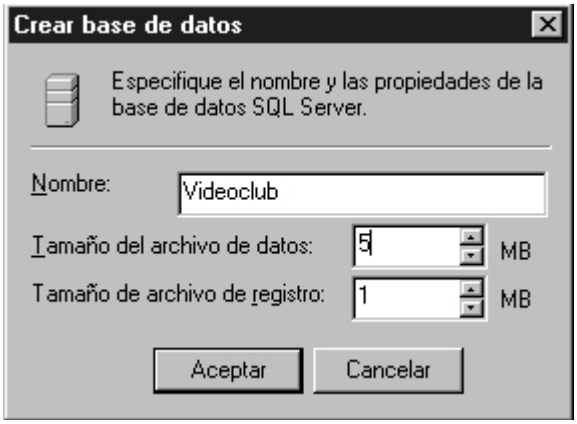


Figura 21.

❺ El paso siguiente consiste, simplemente, en decidir si se van a copiar las tablas íntegras de la base de datos de origen o si se utilizarán instrucciones SQL para definir los conjuntos de datos a migrar (**Figura 22**). En el primer caso, seleccione **`Copiar las tablas de la base de datos de origen`**.

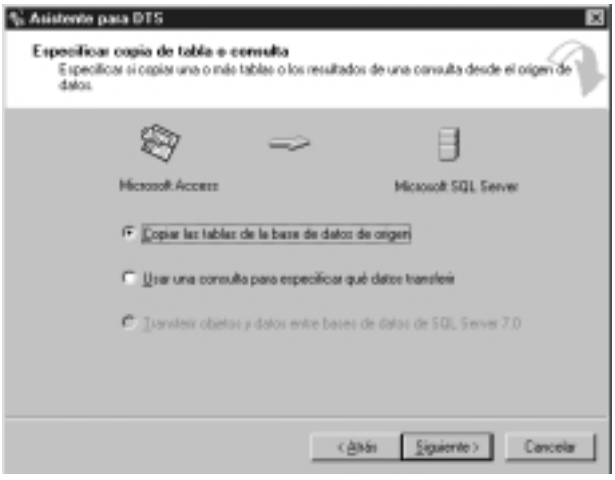


Figura 22.

6 En este punto, el Asistente muestra las tablas existentes en la base de origen, y brinda la opción de seleccionar aquellas que se desee migrar. Selecciónelas todas. Verá que al hacerlo, se completan las otras dos columnas de la grilla: **Tabla de destino** y **Transformar**. En la primera, el Asistente coloca nombres de tabla que coinciden con los de la base de origen. También muestra un ícono para cada tabla, que, si tiene una estrella en la esquina superior izquierda, indica que la tabla se creará durante la migración (**Figura 23**).

Figura 23.

En la columna **Transformar** aparecen botones con puntos suspensivos para cada tabla seleccionada. Si presiona cualquiera de ellos, tendrá la oportunidad de modificar la forma en que el asistente migrará los datos de la tabla correspondiente. Por ejemplo, puede deseleccionar ciertas columnas de ciertas tablas para que no se incluyan en la migración, o hacer que se aplique una conversión a un tipo de datos diferente para alguna columna. Pero en este ejemplo dejaremos las opciones de transformación tal como las infiere el Asistente.

7 El asistente presenta, en esta anteúltima pantalla, una serie de opciones referidas a qué hacer con la operación de migración. Se muestran dos *frames*, titulados "Cuando" y "Guardar". En el primero se ofrece la opción de ejecutar la migración inmediatamente o programarla en un paquete DTS (*data transformation service*, servicio

de transformación de datos) para una ejecución posterior. En el segundo *frame* se da la opción de crear el paquete DTS, independientemente de que la ejecución sea inmediata o diferida (**Figura 24**). En este caso, optaremos por ejecutar la operación inmediatamente.

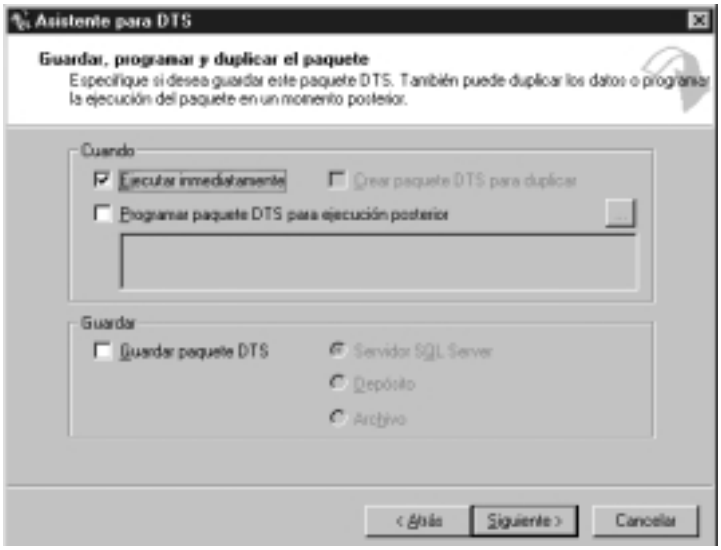


Figura 24.

8 Por último, el Asistente mostrará un resumen de las opciones seleccionadas. Para iniciar la importación de la base de Access en SQL Server, bastará con cliquear en **`Finalizar`**. Inmediatamente surgirá una ventana que muestra los pasos de la migración y el estado de progreso de cada uno (**Figura 25**).

Figura 25.

Elementos para programar desde la base

Con el fin de centralizar la lógica de las aplicaciones y los datos que éstas emplean, los servidores de bases de datos brindan utilidades de programación para llevar a cabo procesos de fondo, que no necesitan interacción con el usuario. Siempre que sea posible, es conveniente resolver las necesidades de una aplicación implementando procesos que son ejecutados directamente por el servidor de bases de datos, puesto que de este modo se efectúan mucho más rápidamente. Además, al estar centralizados en el servidor, resulta más fácil modificarlos, ya que se hace una sola vez, sin necesidad de crear nuevas versiones de la aplicación e instalarlas a cada usuario.

Para programar directamente sobre la base de datos, hay que dominar los siguientes conceptos: ***stored procedures*** (procedimientos almacenados), ***triggers*** (desencadenadores) y ***rules*** (reglas).

Stored procedures

Los ***stored procedures*** (procedimientos almacenados) son secuencias de instrucciones SQL que se almacenan como un objeto más de los que conforman la base de datos, y se ejecutan como una unidad. Se los puede invocar desde un intérprete de comandos SQL (como el Analizador de consultas de SQL Server 7), desde cualquier aplicación con acceso a la base de datos o desde otro procedimiento almacenado.

Las bases de datos **precompilan** los procedimientos almacenados al momento de crearlos o modificarlos, de modo tal de no tener que interpretar sus instrucciones cada vez que se los ejecuta. De este modo, siempre es más rápido invocar un procedimiento almacenado que ejecutar la misma secuencia de instrucciones del procedimiento desde un intérprete de comandos o desde un programa en Visual Basic.

Al igual que los procedimientos de cualquier lenguaje de programación, los procedimientos almacenados de las bases de datos admiten parámetros de entrada y de salida. Pero, además, en algunas bases de datos –como SQL Server–, un procedimiento almacenado también puede devolver el resultado de una consulta, en forma de datos organizados en filas y columnas, como si se la ejecutara directamente. Este resultado puede ser obtenido como un recordset de ADO si se invoca el procedimiento desde Visual Basic, o bien visualizado en la ventana de resultados del intérprete de comandos.

En este sentido, un procedimiento almacenado estaría actuando de

manera similar a una vista, puesto que ambos devuelven conjuntos de resultados. Sin embargo, el procedimiento almacenado puede utilizar sus parámetros de entrada para definir la consulta a ejecutar, cosa que una vista no puede hacer, ya que –en general– las vistas no admiten parámetros (Access rompe esta regla en cierto modo, porque las consultas de Access sí admiten parámetros).

MÁS DATOS

PROCEDIMIENTOS ALMACENADOS EN ORACLE

En PL-SQL –la implementación de SQL de Oracle– los procedimientos almacenados no devuelven un conjunto de resultados. Únicamente interactúan con su entorno mediante parámetros de entrada y salida. La única forma de obtener conjuntos de resultados en forma de recordsets (filas y columnas) es a través de las vistas. Pero para evitar la "rigidez" de las vistas, que no aceptan parámetros que modifiquen su comportamiento, Oracle admite la definición de **funciones**.

Al igual que en cualquier lenguaje de programación, las funciones son análogas a los procedimientos, excepto que devuelven un único valor obtenido de acuerdo con sus parámetros de entrada. Y, a diferencia de los procedimientos almacenados, las funciones pueden invocarse desde una instrucción **SELECT**, e incluso desde una vista.

Esto les da una gran flexibilidad a las vistas, y compensa el hecho de que los procedimientos almacenados no devuelvan conjuntos de resultados. Lógicamente, esto también obliga a observar la programación que se realizará sobre la base con una filosofía diferente de la de SQL Server u otras bases de datos.

Por ejemplo, imaginemos un sistema de cuentas corrientes en el que se debe definir un listado de saldos. En Oracle, se podría establecer una función **SaldoDeCuenta** que recibiera, como parámetro de entrada, un código de cuenta, y devolviera el saldo actual calculándolo mediante operaciones con los datos de varias tablas. Luego se podría definir una vista con la siguiente instrucción SQL (suponiendo una tabla **Cuentas** con el código de la cuenta y el nombre de la misma):

```
CREATE VIEW ListadoDeSaldos
AS
SELECT CodigoCuenta, Nombre, SaldoDeCuenta(CodigoCuenta)
FROM CUENTAS
```

Luego, si se desea acotar el listado –por ejemplo– a los códigos de cuenta mayores o iguales que 100 y menores o iguales que 200, se puede utilizar la vista con la siguiente instrucción:

```
SELECT *
FROM ListadoDeSaldos
WHERE CodigoCuenta > 100 AND CodigoCuenta < 200
```

En SQL Server, en cambio, habría que definir un procedimiento almacenado que reciba como parámetro el rango de códigos que se desea listar. El procedimiento –que también podría llamarse **ListadoDeSaldos**– se encargaría de recorrer la tabla **Cuentas** para el rango solicitado, calculando el saldo para cada una y armando el listado en una tabla temporaria para devolverlo como un conjunto de resultados. La invocación al mismo desde una instrucción SQL, para el mismo rango de cuentas del ejemplo anterior, sería la siguiente:

```
ListadoDeSaldos 100, 200
```

Para ilustrar el uso de procedimientos almacenados, vamos a crear uno en la base de datos **Videoclub**, que arrojará un detalle de los alquileres registrados en la base, con el nombre del cliente que efectuó cada alquiler y la película rentada.

Creando un *stored procedure* PASO A PASO

❶ Entre en Visual Basic 6, sin abrir ningún proyecto. Active la ventana de la vista Datos. Agregue una conexión a la base de datos **Videoclub** en el servidor SQL Server donde la haya importado (**Figura 26**).

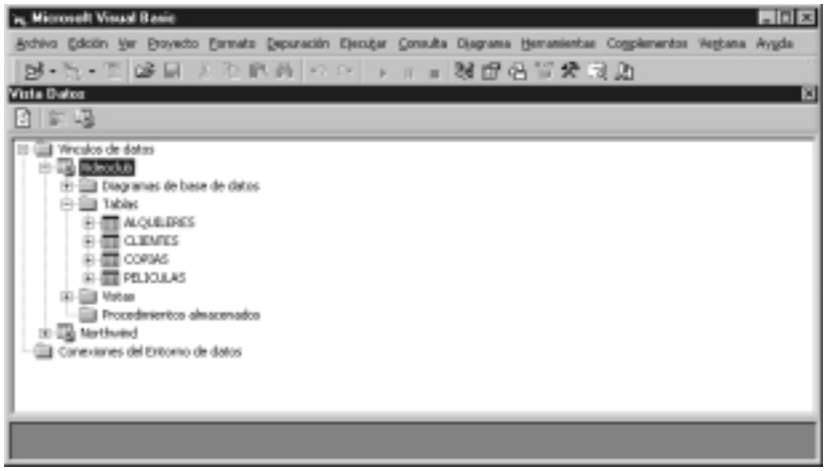


Figura 26.

❷ Cliquee con el botón derecho del mouse en la carpeta **Procedimientos almacenados** y, en el menú contextual, seleccione la op-

ción **`Nuevo procedimiento almacenado`**. Aparecerá una ventana de edición de procedimientos, con la estructura básica para crear uno nuevo (**Figura 27**).

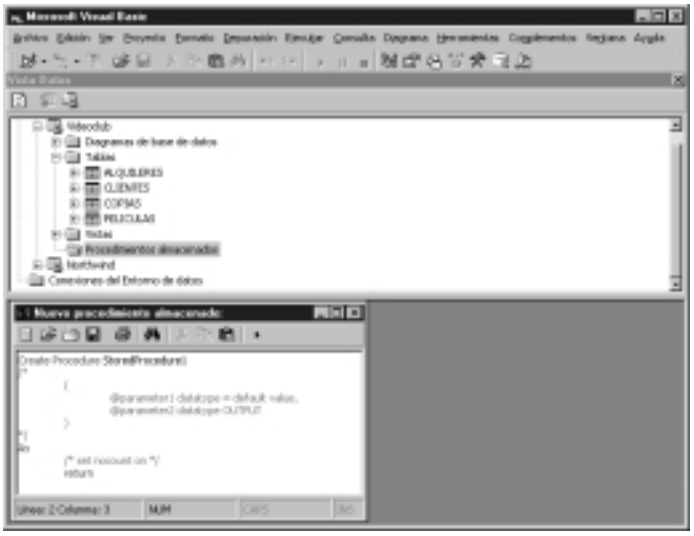


Figura 27.

❸ Modifique el código del procedimiento, reemplazando el nombre del mismo ("StoredProcedure1", a la derecha de la instrucción **`Create Procedure`**) por **`ListadoAlquileres`**. Luego elimine los símbolos que delimitan comentarios ("/*" y "*/"). El resultado hasta aquí debería quedar como muestra la **Figura 28**.

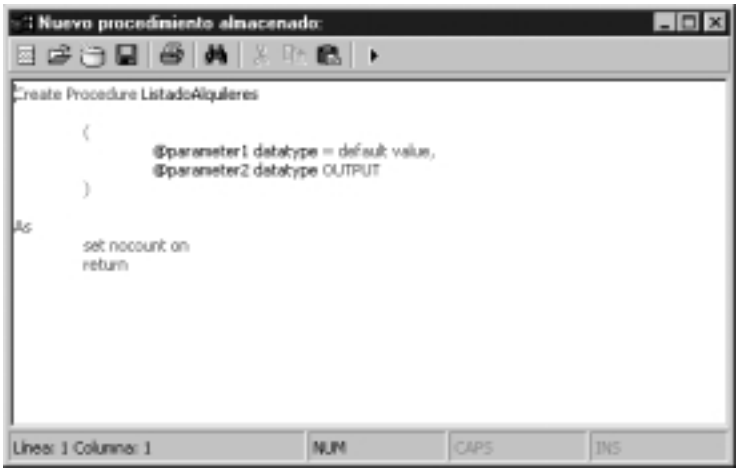


Figura 28.

❹ Luego reemplace los parámetros que aparecen por defecto –**@parameter1** y **@parameter2**– por los parámetros **@CodPeliculaDesde**, **@CodPeliculaHasta**, **@CodClienteDesde** y **@CodClienteHasta**, todos ellos de tipo **INT** (equivalente al tipo **Entero largo** de VB). El encabezado del procedimiento debería, entonces, quedar así:

```
Create Procedure ListadoAlquileres

    (
            @CodPeliculaDesde int = 0,
            @CodPeliculaHasta int = 0,
            @CodClienteDesde int = 0,
            @CodClienteHasta int = 0
    )

As
```

Algunas observaciones con respecto a la sintaxis:

- Todo lo que figura entre el nombre del procedimiento a crear (en este caso, **ListadoAlquileres**) y la palabra clave **As** se toma como parámetros del procedimiento.
- La arroba ("@") se utiliza para definir variables o parámetros, distinguiéndolos de este modo de los campos de la base de datos, que se referencian simplemente por su nombre.
- La declaración de cada parámetro debe estar seguida de su tipo de datos (en este caso, **int**) y, opcionalmente, un valor por defecto a continuación del signo igual ("="); en este caso, el valor por defecto es cero para todos los parámetros.
- Para definir un parámetro de salida, se le debe agregar la palabra clave OUTPUT luego de la indicación del tipo de datos. Por ejemplo: **@Resultado CHAR(10) OUTPUT**.

❺ Ahora, entre las instrucciones **set nocount on** y **return**, escriba la siguiente consulta SQL:

```
SELECT      A.CodPelicula,
            P.Titulo,
            A.CodCopia, A.CodCliente, C.Nombre
```

```
    FROM ALQUILERES A INNER JOIN
                PELICULAS P ON A.CodPelicula = P.CodPelicula
                INNER JOIN CLIENTES C ON A.CodCliente = C.CodCliente
    WHERE       A.CodPelicula >= @CodPeliculaDesde
                AND A.CodPelicula <= @CodPeliculaHasta
                AND A.CodCliente >= @CodClienteDesde
                AND A.CodCliente <= @CodClienteHasta
```

Observe que, de no ser por los parámetros que limitan el listado (colocados dentro de la cláusula **WHERE**), la consulta podría perfectamente ejecutarse desde un intérprete de comandos de SQL para obtener el listado deseado.

6 Guarde el nuevo procedimiento en la base presionando **Ctrl+S** o cliqueando el botón **Guardar en la base de datos** de la barra de herramientas del editor (**Figura 29**).

Figura 29.

Para poner a prueba el procedimiento almacenado, puede crear un formulario con un ADO data control (ADODC) y un control DataGrid, similar al construido en el **Capítulo 2** para mostrar la forma de acceder a un servidor de bases de datos. En la propiedad **RecordSource** del ADODC debe seleccionar el tipo de comando **8 – adCmdUnknown**, mientras que en el recuadro titulado "Texto del comando (SQL)" deberá ingresar la llamada al procedimiento junto con los parámetros actuales **ListadoAlquileres(1, 5, 1, 3)** (**Figura 30**).

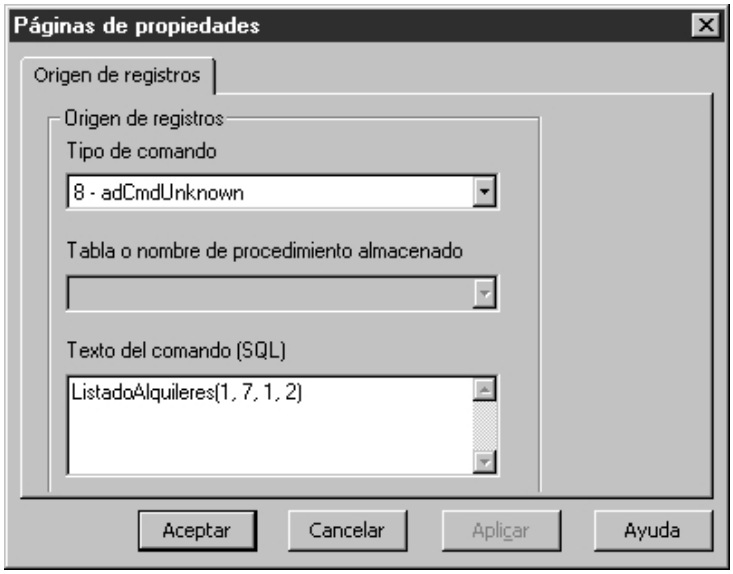


***Figura 30**. En la página de propiedades del ADODC se define el tipo de comando a utilizar para el origen de registros y, debajo, el texto del comando en cuestión.*

Observe que, en virtud de los valores que estamos pasando como parámetros al procedimiento, el listado de alquileres quedará restringido entre la película 1 y la 5, y entre el cliente 1 y el 3. Para hacer que el programa sea un poco más funcional, habría que permitirle al usuario ingresar estos valores, armando el contenido de la propiedad **`RecordSource`** dinámicamente, en tiempo de ejecución, en función de los parámetros ingresados (queda como ejercicio para el lector).

Triggers

Un ***trigger*** (desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente al ocurrir un evento al que está asociado. Normalmente, los *triggers* se asocian a las acciones de insertar (**`INSERT`**), eliminar (**`DELETE`**) o actualizar (**`UPDATE`**) una tabla en particular. Dentro del código del *trigger* se pueden analizar los datos a insertar, eliminar o actualizar, y realizar tareas de validación o actualización de datos relacionados.

Una posible aplicación de *triggers* es la grabación de datos de auditoría sobre una tabla determinada. Se asocian sendos *triggers* para la inserción, eliminación o actualización de filas, y en cada caso se agrega una fila en una tabla de auditoría "gemela" a la que se está actualizan-

do, registrando los datos insertados o eliminados, o bien los datos tal como estaban antes de hacer **UPDATE**, completándolos con la fecha y la hora en que se hizo la operación.

Lo bueno de los *triggers* es que –a diferencia de las rutinas de validación incluidas en una aplicación– se ejecutan **siempre**. Sin importar si los datos provienen de una aplicación interactiva o de un proceso automático por lotes, la base de datos garantiza que el *trigger* se ejecutará cuando corresponda.

OJO CON LOS GATILLOS

Al igual que sus análogos de la vida real (*trigger* en inglés significa **gatillo**), hay que tomar muchas precauciones para utilizarlos, ya que pueden traer consecuencias no deseadas.

Por ejemplo, un *trigger* de **UPDATE** en una tabla puede "disparar" una acción sobre un conjunto de registros muy grande, que haga demorar la acción de **UPDATE** más de lo debido. En una aplicación interactiva, esto podría hacer pensar al usuario que la aplicación se "colgó". O aun peor: el tiempo que demora la acción desencadenada por el *trigger* podría ser mayor que el *timeout* (tiempo máximo de espera) de la aplicación, lo que haría fracasar redondamente la operación de **UPDATE**.

Por otra parte, la acción desencadenada por un *trigger* podría provocar otros *triggers* en otras tablas, haciendo muy fácil que la situación se salga de control. En el peor de los casos, se podría llegar a crear un círculo vicioso, en el que los *triggers* se desencadenaran mutuamente ad-infinitum (si bien los motores bases de datos, en general, detectan esta posibilidad, lo que impide la creación de tales *triggers*).

Para ilustrar el uso de *triggers*, vamos a crear uno de inserción y otro de eliminación en la tabla **Alquileres** de la base **Videoclub**, para actualizar el campo **Alquilada** de la tabla **Copias**. De este modo, nos aseguraremos de que cada vez que se inserte una fila (o sea, cuando se alquile una copia) o se elimine una (el cliente devuelva la copia en cuestión), el campo **Alquilada** se modifique para mostrar la realidad. Aquí se ve claramente cómo se puede utilizar la base de datos para asegurar el cumplimiento de las **reglas de negocio** de una aplicación, sin que haga falta que la aplicación en sí se ocupe de llevar a cabo esta tarea.

Creando *triggers* PASO A PASO

❶ Entre en Visual Basic 6, sin abrir ningún proyecto. Active la ventana de la vista Datos. Despliegue los elementos de la conexión **Videoclub** (creada en el ejercicio anterior), cliqueando en el signo "+" a la izquierda de la misma. A su vez, despliegue los elementos de la carpeta **Tablas**. Luego, haciendo clic con el botón derecho en la tabla **Alquileres**, seleccione la opción **`Nuevo desencadenante...`** del menú contextual (**Figura 31**).

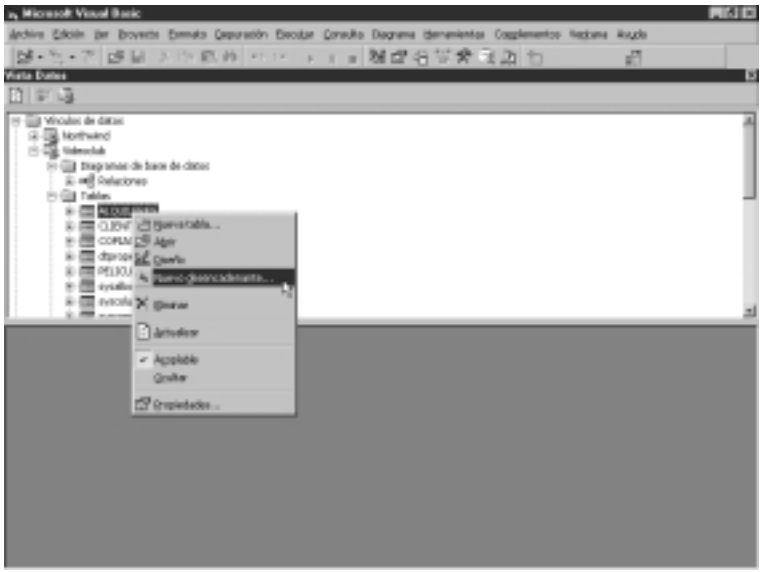


Figura 31.

❷ Aparecerá una ventana de edición similar a la utilizada para armar procedimientos almacenados, con la estructura básica del *trigger* y algunas instrucciones sugeridas, puestas como comentarios. Comience por cambiarle el nombre; asígnele uno que dé una idea de la acción que realiza (si bien el nombre no se utiliza para nada en particular). En vez de **`ALQUILERES_Trigger1`**, colóquele **`ALQUILERES_UpdateCopias`** (**Figura 32**).

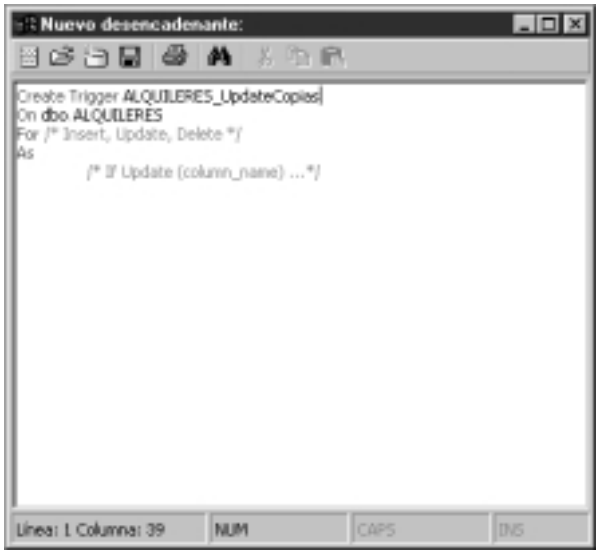


Figura 32.

❸ Luego deberá indicar ante qué eventos se accionará el *trigger*. En este caso, se trata de un *trigger* de inserción o borrado, por lo cual la cláusula **FOR** (después de la línea que dice **On dbo.ALQUILERES**) deberá quedar así:

```
For Insert, Delete
```

❹ Después deberá ingresar el código del *trigger* propiamente dicho. Primero examinaremos si se insertó una fila en la tabla y averiguaremos la cantidad de filas de la tabla "conceptual" **Inserted**. En el código de un *trigger* se pueden utilizar dos tablas virtuales –**Inserted** y **Deleted**–, que tienen una estructura idéntica a la de la tabla en la que se define el *trigger*, y almacenan una fila con los datos a insertar o a actualizar (en el caso de un *trigger* para **INSERT** o para **UPDATE**), o con los datos eliminados (en el caso de un *trigger* para **DELETE**), respectivamente.
A continuación de la palabra clave **As**, ingrese el siguiente código:

```
IF (SELECT COUNT(*) FROM Inserted) > 0
     UPDATE COPIAS
     SET Alquilada = 1
     WHERE CodPelicula = (SELECT CodPelicula FROM Inserted) AND
             CodCopia = (SELECT CodCopia FROM Inserted)
```

La instrucción **IF** en el código de la página 28 pregunta si la cantidad de filas de la tabla **Inserted** es mayor que cero. En caso afirmativo, hace un **UPDATE** de la tabla **Copias**, colocando el valor 1 en el campo **Alquilada** en aquella fila donde el código de película y el código de copia coincidan con los de la tabla **Inserted**.

❺ Luego ingresamos las instrucciones análogas para el caso en que se elimine una fila de la tabla **Alquileres**:

```
IF (SELECT COUNT(*) FROM Deleted) > 0
     UPDATE COPIAS
     SET Alquilada = 0
     WHERE CodPelicula = (SELECT CodPelicula FROM Deleted) AND
             CodCopia = (SELECT CodCopia FROM Deleted)
```

❻ Guarde el *trigger* en la base de datos presionando **Ctrl + S** o cliqueando en el botón **Guardar en la base de datos** de la barra de Herramientas del editor. Una vez guardado, el *trigger* debería verse como muestra la **Figura 33**.

Figura 33.

Para poner a prueba el *trigger*, deberá insertar y luego eliminar una fila válida (es decir, con códigos existentes de película, copia y cliente) en la tabla **Alquileres**, examinando la fila correspondiente de la tabla **Copias** luego de insertar y luego de eliminar. Si todo anda bien, después de hacer la inserción, el campo **Alquilada** de la fila correspon-

diente en la tabla **Copias** debería tomar valor 1, y volver a 0 cuando se elimine la fila de **Alquileres**.

Para hacer estas pruebas, puede vincular las tablas a una base de Access y luego realizar la inserción y la eliminación a mano, o bien utilizar instrucciones **`INSERT`** y **`DELETE`** desde el Analizador de consultas de SQL Server. Una tercera opción es esperar a llegar más adelante en este capítulo, cuando modifiquemos el programa en VB que registraba alquileres de películas, creado en el **Capítulo 3**.

Reglas y tipos definidos por el usuario

Una tercera forma de trasladar la complejidad de una aplicación al motor de base de datos consiste en la utilización de **reglas** que se asocian a determinados campos cuando se desea que sus valores cumplan ciertas condiciones. El uso de reglas se potencia con los tipos de datos definidos por el usuario, ya que éstos permiten reutilizarlas cuando se aplican a muchos campos distintos.

Una manera de definir las reglas de forma interactiva consiste en utilizar el Administrador corporativo de SQL Server. Si no se cuenta con esta herramienta, también es posible ejecutar las instrucciones SQL necesarias para determinar las reglas desde el Analizador de consultas, o desde cualquier intérprete de comandos SQL (ver más adelante). Otra alternativa es crear restricciones **`CHECK`** en las propiedades de la tabla en cuestión, lo cual se puede hacer desde la ventana de la vista Datos en VB, entrando en la opción de diseño de la tabla.

Para ejemplificar el uso de reglas, vamos a agregar el campo **Categoria** a la tabla **Películas** de la base **Videoclub**. Este campo va a ser un CHAR(1), que sólo admitirá los siguientes valores:

'C': comedia
'D': drama
'A': aventuras
'F': ciencia ficción
'O': documental

La responsabilidad de que sólo estos valores puedan ser asignados al campo correrá por cuenta de la regla asociada a este campo.

Para definir la regla **`CategoriaValida`** puede utilizar el Administrador corporativo. Despliegue los elementos de la base **Videoclub**, cliquee con el botón derecho en el ítem **`Reglas`** y seleccione la opción **`Nueva regla`** en el menú contextual (**Figura 34**).

***Figura 34**. La creación de una nueva regla se lleva a cabo abriendo el menú contextual que depende del nodo* **`Reglas`** *de la base de datos.*

Aparecerá una ventana titulada **`Propiedades de la regla`**. En la parte superior debe ingresar el nombre de la regla, y en el recuadro inferior, el código de la misma. En el primero escriba **`CategoriaValida`**, y en el recuadro de código ingrese la siguiente instrucción SQL:

```
@valor IN ('C', 'D', 'A', 'F', 'O')
```

En esta instrucción SQL se utiliza la variable **`@valor`**, pero igualmente se podría haber empleado cualquier otra. La condición que se debe cumplir es que el contenido del campo sea uno de los elementos del conjunto de la derecha.

En la parte inferior de la ventana **`Propiedades de la regla`** se observan dos botones: **`Enlazar UDT...`** y **`Enlazar columnas...`** (**Figura 35**). Con el segundo es posible enlazar la regla a ciertas columnas, para que todo dato que se ingrese en ellas sea evaluado según la regla. El otro botón es aún más práctico, ya que permite enlazar la regla a un **tipo de datos definido por el usuario** (UDT, *user defined type*). Claro que, para hacer esto, primero hay que determinar el tipo de datos con el que se va a enlazar.

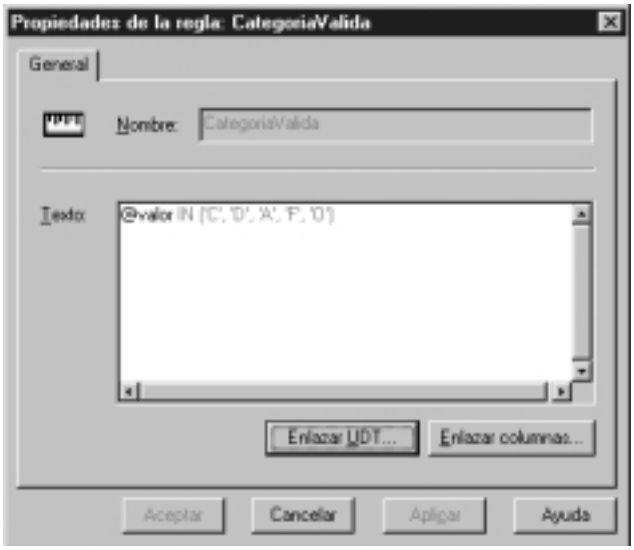


Figura 35. *El texto de toda regla debe ser una condición lógica expresada en lenguaje SQL.*

Para definir el tipo de datos **`Categoría`** hay que cliquear con el botón derecho en el ítem **`Tipos de datos definidos por el usuario`** de la base **Videoclub**. Luego, en el menú contextual, seleccionar **`Nuevo tipo de datos definido por el usuario...`**. Con esto, surgirá una ventana como la de la **Figura 36**.

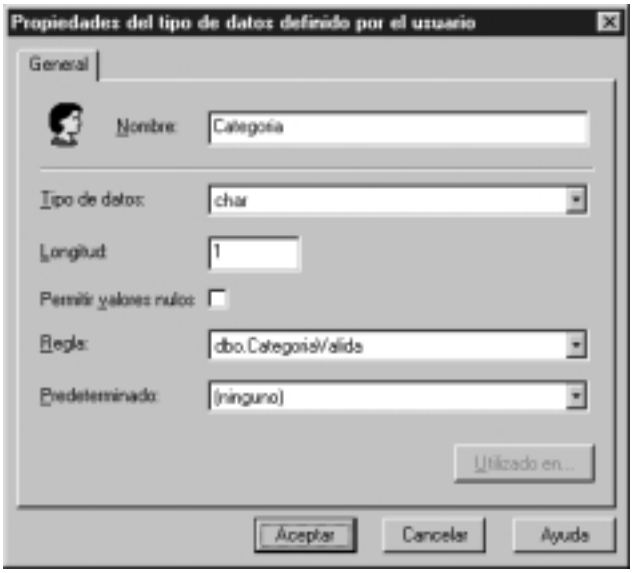


Figura 36. *Los tipos de datos definidos por el usuario relacionan un tipo de datos estándar (como puede ser CHAR(1)) con una regla.*

En ella, debe consignar el nombre del nuevo tipo de datos (**Categoria**), el tipo de datos estándar (**char**), la longitud máxima (**1**), si permite valores nulos (**no**), si tiene una regla asociada (**CategoriaValida**) y si lleva algún valor predeterminado (en este caso, **ninguno**).

Sólo falta crear un nuevo campo en la tabla **Películas** y asignarle, en vez de un tipo de datos estándar, el tipo de datos **Categoria**. Esto hará que el nuevo campo "herede" todas las características del tipo de datos en cuestión, incluyendo sus reglas asociadas. Si se diera el caso de que en otra tabla fuera necesario incorporar un campo con las mismas características, bastaría con asignarle el tipo de datos **Categoria**. Y si el día de mañana cambia alguna de sus características –por ejemplo, si en vez de ser un único carácter fueran dos, o si se incorporara la categoría 'X' para películas de acción–, bastará con modificar el tipo de datos y/o la regla para que el cambio se extienda a todos los campos asociados.

Si no cuenta con el Administrador corporativo, puede correr las siguientes secuencias de comandos SQL para crear la regla **CategoriaValida** y el tipo de datos **Categoria**:

```
create rule CategoriaValida as @valor IN ('C', 'D', 'A', 'F', 'O')
GO

setuser N'dbo'
GO

EXEC sp_addtype N'Categoria', N'char (1)', N'not null'
GO

setuser
GO

setuser N'dbo'
GO

EXEC sp_bindrule N'dbo.CategoriaValida', N'Categoria'
GO

setuser
GO
```

Desatando el poder de ADO

En los siguientes párrafos encararemos la tarea de migrar y ampliar la funcionalidad del programa de **Videoclub** creado en el **Capítulo 3**, y lo adaptaremos para que funcione con SQL Server. Mediante el uso del modelo ADO, aprovecharemos las ventajas del motor de SQL Server, tales como el uso de *triggers*. Veremos también cómo el hecho de haber utilizado instrucciones SQL simplifica el trabajo de migración.

Finalmente, empleando la herramienta **`Entorno de datos`**, crearemos un informe que toma los datos arrojados por el procedimiento almacenado **`ListadoAlquileres`**.

Migración de DAO a ADO

El hecho de haber construido el programa de **Videoclub** con una cierta visión de futuro nos facilitará la tarea de migrarlo a SQL Server y ADO. En términos generales, el trabajo consistirá en reemplazar el uso de objetos **`Database`** de DAO por objetos **`Connection`** de ADO. Luego, para operar con tablas y consultas, se utilizan –al igual que en DAO– objetos Recordset, aunque, como veremos a continuación, hay algunos pequeños detalles que difieren en el uso de estos objetos en DAO y en ADO. Además, hay también un leve cambio que hacer en virtud de la diferencia entre la implementación de SQL en el motor Jet (el que opera con las bases de Access) y en el motor de SQL Server.

Migrar a un programa basado en DAO al modelo ADO PASO A PASO

1. Cargue el proyecto **Videoclub.vbp** en VB y guárdelo con otro nombre, o con el mismo en una carpeta diferente. También deberá almacenar el formulario con otro nombre o en otra carpeta.

2. Entre en la ventana **`Referencias`** del proyecto. Elimine la referencia a DAO y seleccione (marcando con un tilde) la referencia a Microsoft ActiveX Data Objets 2.0 Library (o una versión posterior, como la 2.1, si aparece en la lista).

❸ Active la ventana de edición de código y sitúese en el procedimiento **Guardar**, como muestra la **Figura 37**.

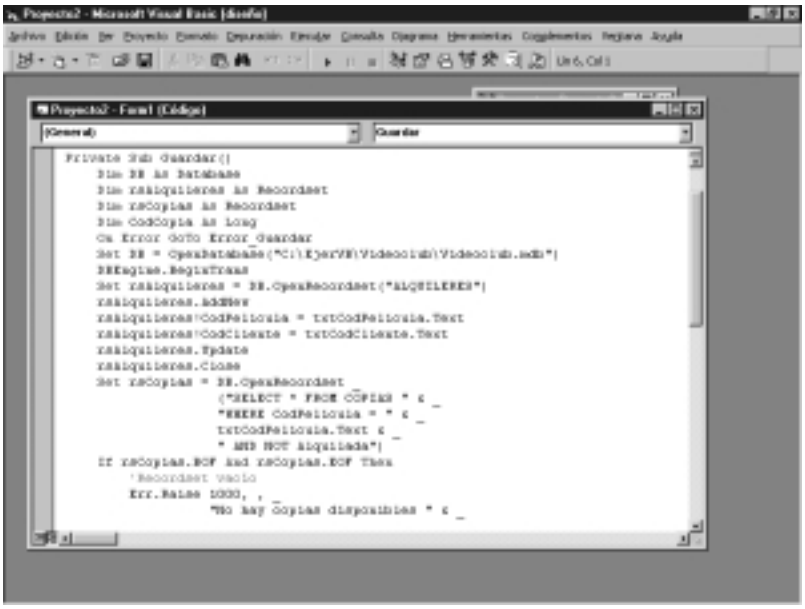


Figura 37.

❹ Modifique las líneas en las que se declaran las variables objeto.

Antes:

```
Dim DB As Database
Dim rsAlquileres As Recordset
Dim rsCopias As Recordset
```

Después:

```
Dim DB As New Connection
Dim rsAlquileres As New Recordset
Dim rsCopias As New Recordset
```

¿Por qué el cambio en los objetos utilizados? En primer lugar, el objeto **Connection** se usa en lugar del objeto **Database** de DAO porque, en ADO, todas las bases de datos (sean locales o remotas) se acceden mediante una conexión; por eso es que no tendría sentido un objeto **Database** en ADO. Luego, los objetos

Recordset se crean directamente en la declaración (mediante la palabra clave **New**), puesto que no hace falta especificar el origen del recordset en el momento de crear el mismo. En DAO, en cambio, los recordsets se crean en el preciso momento de especificar de dónde provendrán sus datos.

❺ Reemplace la línea en la que se abre la base de datos por una donde se establece la conexión.

Antes:

```
Set DB = OpenDatabase("C:\EjerVB\Videoclub\Videoclub.mdb")
```

Después:

```
DB.Open "Provider=SQLOLEDB;" & _
        "User ID=sa;" & _
        "Pwd=;" & _
        "Initial Catalog=Videoclub;" & _
        "Data Source=(local)"
```

Esta línea abre la conexión con la base de datos utilizando una **cadena de conexión para OLE DB**. En ella, se establece el nombre del proveedor OLE DB (o *Provider*; en este caso, SQLOLEDB); el nombre de usuario (*User ID*) y la contraseña (*Pwd*); la base de datos a utilizar (*Initial Catalog*), y la dirección o nombre del servidor (*Data Source*) con el que se desea conectar (en este caso, "(local)").

❻ Luego, el método **BeginTrans**, que en el caso de DAO pertenecía al objeto global **DBEngine**, en este caso depende de la conexión a la base de datos (el objeto que denominamos **DB**). Por lo tanto, hay que hacer la siguiente modificación en la línea donde se inicia la transacción.

Antes:

```
DBEngine.BeginTrans
```

Después:

```
DB.BeginTrans
```

❼ En el siguiente paso se debe abrir el recordset para acceder a la tabla **Alquileres**. En ADO, como el objeto **rsAlquileres** ya había sido creado en la declaración de variables, basta con utilizar su método **Open** –con los parámetros que se ven a continuación– para abrirlo.

Antes:

```
Set rsAlquileres = DB.OpenRecordset("ALQUILERES")
```

Después:

```
rsAlquileres.Open "ALQUILERES", _
                  DB, _
                  adOpenKeyset, _
                  adLockOptimistic, _
                  adCmdTable
```

El método **Open** utiliza los siguientes parámetros:

- **Origen de los datos** (*Source*; en este caso, la tabla **Alquileres**), que es una cadena de caracteres con el nombre de una tabla o una vista, el código de una consulta SQL o una llamada a un procedimiento almacenado.
- **Conexión activa** (*ActiveConnection*; en este caso, el objeto **DB**), que debe ser un objeto de tipo **Connection** previamente abierto.
- **Tipo de cursor** (*CursorType*; en este caso, la constante **adOpenKeyset**), que determina el tipo de acceso a realizar con el recordset. Puede utilizarse cualquiera de las constantes *CursorTypeEnum* de ADO, según se desee abrir el recordset como sólo lectura, como recordset dinámico, etc.
- **Tipo de bloqueo** (*LockType*; en este caso, la constante **adLockOptimistic**), que determina si se desea aplicar bloqueo optimista, optimista para actualización por lotes, pesimista o de sólo lectura. Se puede utilizar cualquiera de las constantes *LockTypeEnum* de ADO.
- **Opciones** (*Options*; en este caso, la constante **adCmdTable**). Con este parámetro se pueden especificar ciertas opciones, como el tipo de

origen de datos a utilizar. Dado que estamos refiriendo directamente a la tabla **Alquileres**, utilizamos aquí la constante **adCmdTable** para indicar que el tipo de origen es una tabla.

❽ Las instrucciones que siguen deben permanecer sin cambios hasta llegar a la apertura del recordset **rsCopias**. En este caso, se vuelve a utilizar el método **Open**, sólo que, en vez de hacer referencia a una tabla, aludiremos a una consulta SQL.

Antes:

```
Set rsCopias = DB.OpenRecordset _
                (“SELECT * FROM COPIAS “ & _
                “WHERE CodPelicula = “ & _
                txtCodPelicula.Text & _
                “ AND NOT Alquilada”)
```

Después:

```
rsCopias.Open “SELECT * FROM COPIAS “ & _
               “WHERE CodPelicula = “ & _
               txtCodPelicula.Text & _
               “ AND Alquilada = 0”, _
               DB, _
               adOpenKeyset, _
               adLockOptimistic, _
               adCmdText
```

Obsérvese que el código SQL para definir esta consulta es *casi* igual para la base de Access y para la de SQL Server. La única diferencia es que, al no reconocer SQL Server el tipo de datos booleano, en vez de poner como condición **AND NOT Alquilada**, se debe escribir **AND Alquilada = 0** (una sutileza).

Los parámetros utilizados en el método **Open** son similares a los empleados más arriba para abrir la tabla **Alquileres**, con la diferencia de que, en el último parámetro (**Opciones**), en vez de especificar **adCmdTable** se indica **adCmdText**, porque no se está abriendo directamente una tabla, sino una consulta SQL.

❾ Las instrucciones que siguen a la apertura del recordset **rsCopias** también deben permanecer sin cambios, excepto el método **Edit**, que no es aceptado por los recordsets de ADO y, por lo tanto, se debe eliminar la línea de código donde aparece.
Luego volvemos a utilizar el recordset **rsAlquileres**, pero esta vez abriéndolo como una consulta SQL (de forma análoga a como hicimos con el recordset **rsCopias**).

Antes:

```
Set rsAlquileres = DB.OpenRecordset _
                   ("SELECT * FROM ALQUILERES " & _
                   "WHERE CodPelicula = " & _
                   txtCodPelicula.Text & _
                   " AND CodCliente = " & _
                   txtCodCliente.Text)
```

Después:

```
rsAlquileres.Open "SELECT * FROM ALQUILERES " & _
                  "WHERE CodPelicula = " & _
                  txtCodPelicula.Text & _
                  " AND CodCliente = " & _
                  txtCodCliente.Text, _
                  DB, _
                  adOpenKeyset, _
                  adLockOptimistic, _
                  adCmdText
```

❿ A partir de aquí, sólo resta eliminar la línea donde se invoca al método **Edit** del recordset **rsAlquileres** y cambiar el objeto **DBEngine** por el objeto **DB** en la invocación de los métodos **CommitTrans** y **RollbackTrans**. Una vez concluido, el procedimiento completo debería quedar así:

```
Private Sub Guardar()
    Dim DB As New Connection
    Dim rsAlquileres As New Recordset
    Dim rsCopias As New Recordset
    Dim CodCopia As Long
```

```
On Error GoTo Error_Guardar
DB.Open "Provider=SQLOLEDB;" & _
        "User ID=sa;" & _
        "Pwd=;" & _
        "Initial Catalog=Videoclub;" & _
        "Data Source=(local)"
DB.BeginTrans
rsAlquileres.Open "ALQUILERES", _
                  DB, _
                  adOpenKeyset, _
                  adLockOptimistic, _
                  adCmdTable
rsAlquileres.AddNew
rsAlquileres!CodPelicula = txtCodPelicula.Text
rsAlquileres!CodCliente = txtCodCliente.Text
rsAlquileres.Update
rsAlquileres.Close
rsCopias.Open "SELECT * FROM COPIAS " & _
              "WHERE CodPelicula = " & _
              txtCodPelicula.Text & _
              " AND Alquilada = 0", _
              DB, _
              adOpenKeyset, _
              adLockOptimistic, _
              adCmdText
If rsCopias.BOF And rsCopias.EOF Then
    'Recordset vacío
    Err.Raise 1000, , _
              "No hay copias disponibles " & _
              "de la película seleccionada"
Else
    rsCopias.MoveFirst
    CodCopia = rsCopias!CodCopia
    rsCopias!Alquilada = True
    rsCopias.Update
End If
rsAlquileres.Open "SELECT * FROM ALQUILERES " & _
                  "WHERE CodPelicula = " & _
                  txtCodPelicula.Text & _
                  " AND CodCliente = " & _
                  txtCodCliente.Text, _
```

```
                            DB, _
                            adOpenKeyset, _
                            adLockOptimistic, _
                            adCmdText
        rsAlquileres!CodCopia = CodCopia
        rsAlquileres.Update

    Error_Guardar:
        If Err.Number <> 0 Then
            DB.RollbackTrans
            MsgBox "Error: " & Err.Number & _
                   " - " & Err.Description
        Else
            DB.CommitTrans
            MsgBox "Alquiler registrado " & _
                   "exitosamente"
            txtCodPelicula.Text = ""
            txtCodCliente.Text = ""
            txtCodPelicula.SetFocus
        End If
    End Sub
```

Puede utilizar este programa adaptado al modelo ADO para poner a prueba los *triggers* creados para la tabla **Alquileres**.

El entorno de datos

Los entornos de datos (objetos *data environment*) son elementos útiles para reducir al máximo la necesidad de escribir código cuando se utiliza acceso a datos mediante ADO. La herramienta empleada para construir entornos de datos es el **diseñador de entornos de datos**. Esta herramienta está implementada como un diseñador ActiveX. Normalmente, se accede a ella mediante la opción **`Proyecto-/Más diseñadores ActiveX.../Data Environment`** del menú de Visual Basic 6. Si esta opción no aparece, se debe seleccionar el ítem **`Data Environment Instance 1.0`** en la ventana **`Referencias`**.

Al seleccionar la opción mencionada del menú de Visual Basic 6, se crea un nuevo entorno de datos (**Figura 38**), en el cual es posible colocar objetos de acceso a datos y manipular sus propiedades, de la misma forma en que se colocan controles sobre un formulario.

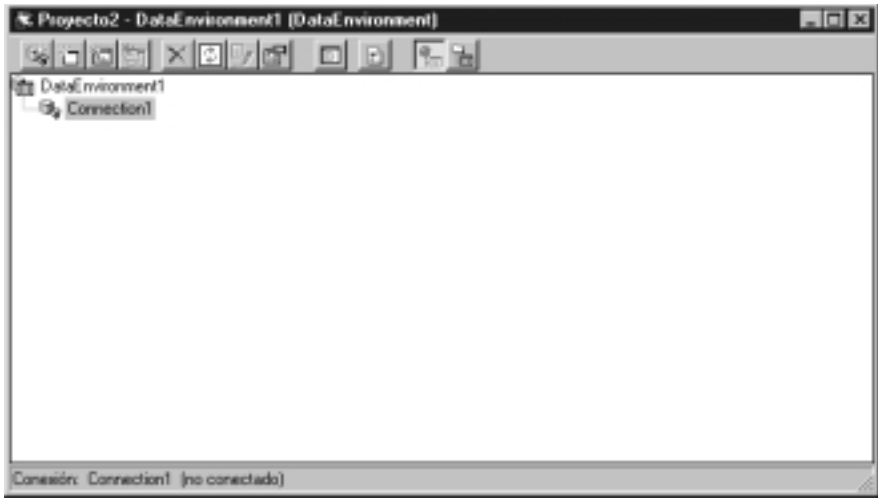


Figura 38. *El entorno de datos reúne elementos de acceso a datos (tablas, vistas, procedimientos almacenados, comandos, etc.) y permite definir sus propiedades, para facilitar su uso a la hora de escribir código.*

El diseño del entorno de datos se simplifica enormemente con la ayuda de la ventana de la vista de datos, ya que permite crear objetos directamente arrastrándolos desde esta última, con lo cual muchas de las propiedades de estos objetos ya tendrán asignados los valores correctos.

Para ampliar la funcionalidad de nuestro proyecto **Videoclub** vamos a agregarle un entorno de datos que luego utilizaremos para crear un listado de alquileres a partir del *stored procedure* generado anteriormente en este capítulo.

Crear un entorno de datos — PASO A PASO

1. En Visual Basic, cargue el proyecto **Videoclub** modificado en el ejercicio anterior para utilizar ADO. Agréguele un entorno de datos, mediante la opción **`Proyecto/Más diseñadores ActiveX.../Data Environment`**.

2. Aparecerá la ventana del editor de entornos de datos. En ella, seleccione el elemento titulado **`DataEnvironment1`** y presione **`F4`** para activar la ventana de propiedades de dicho objeto. Cámbiele el valor de la propiedad **`(Nombre)`** por **`EntornoVideoclub`**.

❸ Luego, también en la ventana del editor de entornos de datos, seleccione el objeto **Connection1** y elimínelo, presionando la tecla **Supr** o cliqueando con el botón derecho y seleccionando la opción **Eliminar** en el menú contextual.

❹ Abra la ventana de la vista Datos. En ella debería figurar una conexión denominada **Videoclub** (si no aparece, créela tal como se explicó anteriormente en este capítulo).

❺ Despliegue los elementos de la conexión **Videoclub** (cliqueando en el "+" a la izquierda de este elemento) y luego despliegue también los elementos de la carpeta **Procedimientos almacenados**. Entre estos últimos debería aparecer el procedimiento **ListadoAlquileres**, creado anteriormente en este capítulo. Cliquee en él, arrástrelo hasta la ventana del editor de entornos de datos y déjelo caer sobre el elemento **EntornoVideoclub**.

❻ Con esta acción se creará un objeto conexión –denominado **Connection1**– debajo de **EntornoVideoclub**, y un objeto comando –denominado **ListadoAlquileres**– debajo de **Connection1**. Si cliquea en el "+" a la izquierda de **ListadoAlquileres**, podrá ver los campos que contiene este comando (**Figura 39**).

Figura 39.

❼ Seleccione el objeto **Connection1** y abra su ventana de propiedades (presionando **F4**). Cámbiele el valor de la propiedad (**Nombre**) por **ConexionVideoclub**.

Habiendo definido el entorno de datos con sus correspondientes objetos conexión y comando, bastará con arrastrar y soltar elementos sobre un formulario para crear un informe a partir del procedimiento **`ListadoAlquileres`**.

Pero antes, una observación: como sugiere el ejercicio anterior, el entorno de datos puede utilizarse como un repositorio que reúna todos los objetos de acceso a datos a utilizar a lo largo de un proyecto. Estos objetos pueden emplearse luego para construir fácilmente informes, ventanas de carga de datos y ABMs. También se los puede usar para referenciarlos desde el código de un módulo o de un módulo de clase, lo cual evita el trabajo de crear objetos ADO y fijar sus propiedades cada vez que se deba programar un acceso a datos.

Esto último sugiere que –a diferencia de los controles Data– los entornos de datos pueden utilizarse también en la capa de acceso a datos de un proyecto, y no sólo en la capa de interfase (más sobre esto en el **Capítulo 6**).

Para redondear nuestro humilde programa de **Videoclub**, vamos a diseñar un informe que mostrará el detalle de los alquileres registrados.

Para ello, el primer paso consiste en seleccionar la opción **`Agregar Data Report`** del menú **`Proyecto`**. Con esto, aparecerá una ventana de diseño de un nuevo informe (**Figura 40**).

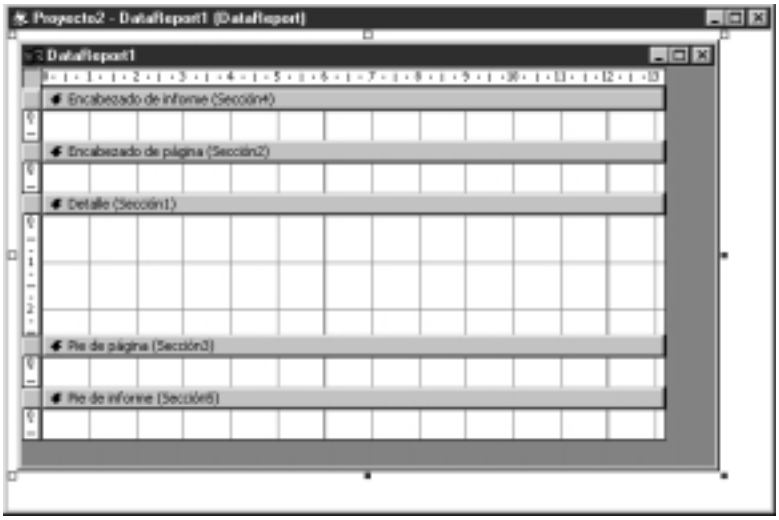


***Figura 40**. El Data Report es como un formulario, dividido en las secciones propias de un informe (encabezado, título, detalles, etc.), que también permite colocar controles simples, tales como rótulos, imágenes y cuadros de texto.*

Esta ventana es análoga a un formulario común y corriente de Visual Basic, puesto que admite que se coloquen controles sobre ella y se establezcan propiedades para los mismos; aunque los posibles controles se obtienen de un conjunto de herramientas especiales, diferentes de las que se utilizan para los formularios.

Para establecer el origen de los datos de este reporte, hay que definir el valor de sus propiedades **DataSource** y **DataMember**. Al abrir la ventana de propiedades del informe (presionando **F4**) veremos que el valor de **DataSource** se puede elegir de una lista de opciones; y lo más interesante es que esta lista nos muestra los entornos de datos incluidos en el proyecto (en este caso, uno solo: **EntornoVideoclub**).

Habiendo definido el valor la propiedad **DataSource**, sucede que para la propiedad **DataMember** también se nos ofrece una serie de opciones para elegir. Y resulta que las opciones aquí ofrecidas son nada más ni nada menos que los objetos **Command** definidos en el entorno de datos elegido como **DataSource** (en este caso, uno solo: **ListadoAlquileres**).

Resumiendo: **los Data Reports son lo suficientemente inteligentes como para aprovechar el trabajo hecho durante la definición de los entornos de datos de un proyecto**.

Habiendo, entonces, definido la propiedad **DataSource** como **EntornoVideoclub** y la propiedad **DataMember** como **ListadoAlquileres**, podemos proceder a diseñar "visualmente" el informe. Pero primero, ya que tiene abierta la ventana de propiedades, aproveche para cambiar las propiedades **(Name)** y **Caption** del reporte, asignándoles –respectivamente– "ListadoAlquileres" y "Listado de alquileres" (**Figura 41**).

***Figura 41**. Al fijar un par de propiedades (con valores definidos en el entorno de datos) el informe ya cuenta con una conexión a la base de datos.*

El trabajo de diseño visual del informe consiste, simplemente, en arrastrar los campos que expone el comando **`ListadoAlquileres`** (en la ventana del entorno de datos que lo contiene) hasta un lugar apropiado del informe. Comencemos arrastrando el campo **CodPelicula** hasta la sección **Detalle** (**Sección 1**) del informe. Observará que esto crea dos elementos dentro del informe: el campo en sí y un rótulo que lo identifica (**Figura 42**). Lamentablemente, el Data Report no parece ser tan inteligente como para colocar el rótulo donde debería estar; esto es, en la sección **Encabezado de página** (**Sección 2**). Por tal razón, es necesario llevarlo a mano (si algún lector conoce la forma de automatizar esto, sus comentarios serán agradecidos).

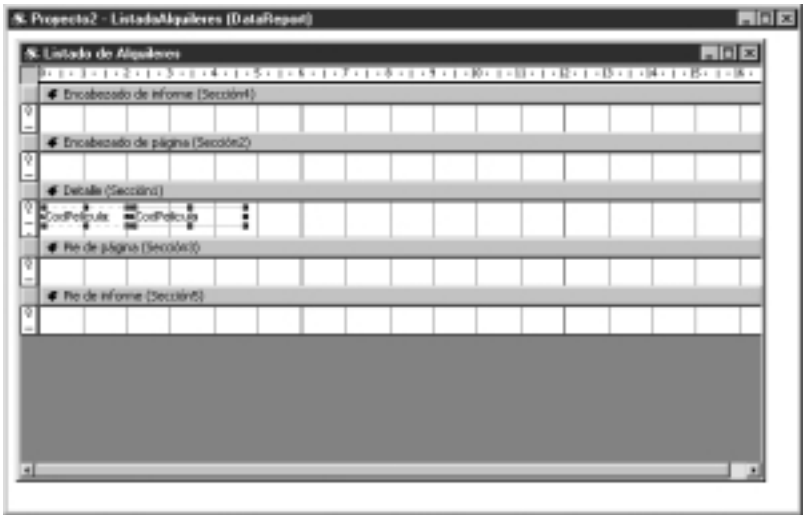


***Figura 42**. Al arrastrar un campo del entorno de datos al Data Report, tanto el campo en sí como el rótulo que lo identifica se colocan en la sección de detalles.*

Luego deberá repetir el mismo proceso para el resto de los campos a colocar en el informe. Puede mover y redimensionar los campos y rótulos, así como cambiar la propiedad **`Caption`** de estos últimos, para una mejor visualización de los datos. Finalmente, el diseño del informe debería quedar como muestra la **Figura 43**.

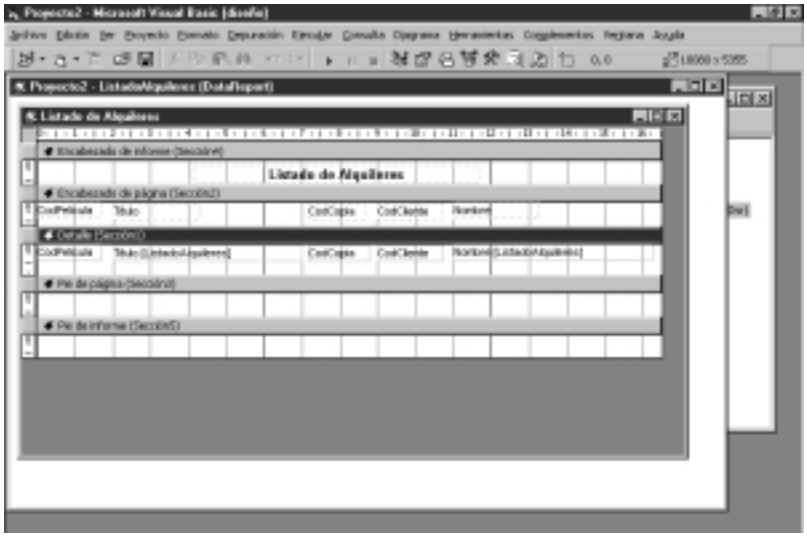


***Figura 43**. Con un poco de trabajo, el diseño del informe adquiere un aspecto bastante "profesional".*

Ahora, sólo falta crear una llamada al nuevo informe desde alguna parte del programa. Una solución bastante simple es colocar un botón titulado **Listado** (y denominado **cmdListado**) en el único formulario del proyecto (**Figura 44**).

***Figura 44**. El botón **Listado** brinda un acceso rápido al informe **ListadoAlquileres**.*

Si el comando a utilizar para emitir el listado fuera una vista de la base de datos, o incluso un procedimiento almacenado sin parámetros, en el código del evento **Click** del botón **cmdListado** sólo haría falta invocar al método **Show** del informe **ListadoAlquileres** para que éste apareciera; igual que si estuviéramos abriendo un formulario. Pero como tuvimos el buen tino de definir el procedimiento con cuatro parámetros –**CodPeliculaDesde**, **CodPeliculaHasta**, **CodClienteDesde** y **CodClienteHasta**– para hacerlo más flexible, el trabajo para invocar el informe va a ser un poco más tedioso.

En primer lugar, en el código del evento **Click** del botón **cmdListado** hay que buscar la forma de pedir al usuario que ingrese los cuatro parámetros y cargar cada uno de ellos en uno de los elementos de la colección **Parameters** del objeto comando **ListadoAlquileres**. La forma más simple es la siguiente:

```
Private Sub cmdListado_Click()
    With EntornoVideoclub.Commands("ListadoAlquileres")
        .Parameters("@CodPeliculaDesde").Value = _
         InputBox("Código de película inicial")
        .Parameters("@CodPeliculaHasta").Value = _
         InputBox("Código de película final")
        .Parameters("@CodClienteDesde").Value = _
         InputBox("Código de cliente inicial")
        .Parameters("@CodClienteHasta").Value = _
         InputBox("Código de cliente final")
    End With
```

Con cada una de las instrucciones encerradas en el bloque **With ... End With** estamos pidiendo un valor (con la función **InputBox**) y cargándolo en el ítem correspondiente de la colección **Parameters** del comando **ListadoAlquileres**.

Luego ejecutamos el método **Show** del informe **ListadoAlquileres**, de la misma forma en que mostraríamos un formulario:

```
    ListadoAlquileres.Show vbModal
```

Cuando se ejecute esta instrucción, se abrirá el informe, mostrando el resultado de ejecutar el procedimiento **ListadoAlquileres** con los parámetros especificados.

Hasta aquí hemos cubierto los pasos esenciales para la generación del informe (ya puede ejecutar el programa y probar el funcionamiento del botón **cmdListado**, si lo desea). Sin embargo, hay un problema: si se cliquea varias veces seguidas el botón **cmdListado** con distintos parámetros, el informe mostrará extrañamente siempre los mismos datos que al principio.

La razón de esto es que, la primera vez que se abre el informe, la ejecución del procedimiento **ListadoAlquileres** genera automáticamente un recordset llamado **rsListadoAlquileres** en el entorno de datos. Y este recordset es persistente, con lo cual las sucesivas veces que se abra el informe los resultados provendrán del recordset que se creó originalmente. Por eso es que los valores no cambian, aun enviando distintos parámetros.

Para evitar este problema, hay que cerrar el recordset **rsListadoAlquileres** luego de emitir el informe, agregando la siguiente instrucción:

```
    EntornoVideoclub.rsListadoAlquileres.Close
```

De este modo, la próxima vez que se pida el informe, el recordset se volverá a crear, y ejecutará el procedimiento con los parámetros que correspondan.

El código completo del evento **Click** del botón **cmdListado** debería quedar así:

```
Private Sub cmdListado_Click()
    With EntornoVideoclub.Commands("ListadoAlquileres")
        .Parameters("@CodPeliculaDesde").Value = _
         InputBox("Código de película inicial")
        .Parameters("@CodPeliculaHasta").Value = _
         InputBox("Código de película final")
        .Parameters("@CodClienteDesde").Value = _
         InputBox("Código de cliente inicial")
        .Parameters("@CodClienteHasta").Value = _
         InputBox("Código de cliente final")
    End With
    ListadoAlquileres.Show vbModal
    EntornoVideoclub.rsListadoAlquileres.Close
End Sub
```

Ejercicios para lectores valientes

- Si realizó el ejercicio sugerido en el Capítulo 3 acerca de agregar una función para registrar la devolución de películas, adapte esa función para que figure también en el proyecto de este capítulo, funcionando contra SQL Server.
- Complete el entorno de datos agregando comandos para acceder directamente a las tablas de **Productos** y **Clientes**, y use estos comandos para crear ABMs de ambas tablas.