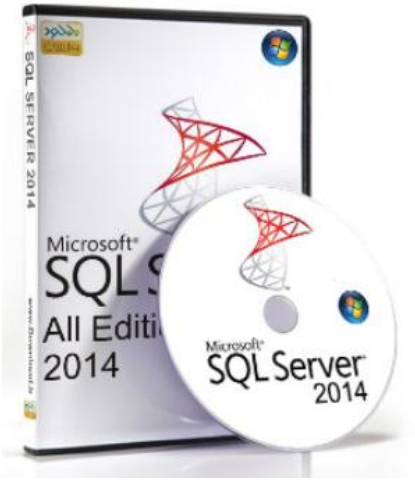
- 1



Microsoft®
SQL Server®

Introducción

SQL Server 2014 es un elemento fundamental de la Plataforma de Datos de Microsoft, capaz de gestionar cualquier tipo de datos, en cualquier sitio y en cualquier momento. Le permite almacenar datos de documentos estructurados, semiestructurados o no estructurados como son las imágenes, música y archivos directamente dentro de la base de datos. SQL Server 2014 le ayuda a obtener más rendimiento de los datos, poniendo a su disposición una amplia gama de servicios integrados como son consultas, búsquedas, sincronizaciones, informes y análisis. Sus datos pueden almacenarse y recuperarse desde sus servidores más potentes del Data Center hasta los desktops y dispositivos móviles, permitiéndole tener un mayor control sobre la información sin importar dónde se almacena físicamente.

2014
Microsoft®
SQL Server®

SQL Server 2014 permite crear aplicaciones críticas mediante una tecnología de seguridad in-memory de alto rendimiento en OLTP, almacenes de datos, Business Intelligence y análisis. Según Gartner, SQL Server 2014 es la solución líder para estas cargas de datos y utiliza un conjunto de herramientas comunes para implementar y administrar bases de datos tanto en la nube como en el entorno local.

SQL Server 2014 le permite utilizar sus datos en aplicaciones a medida desarrolladas con Microsoft® .NET y Visual Studio y también desde su propia Arquitectura Orientada a Servicio (SOA) y los procesos empresariales empleando Microsoft® BizTalk® Server.

Además, las personas que gestionan la información pueden acceder directamente a los datos con las herramientas que utilizan habitualmente como Microsoft® Office 2013. SQL Server 2014 le ofrece una plataforma de datos, fiable, productiva e inteligente para cubrir todas sus necesidades.

SQL Server permite ejecutar aplicaciones de misión crítica, reduciendo costos de administración de infraestructura de datos y brindando introspectiva e información a todos los usuarios.

Confiable: Permite a las organizaciones ejecutar sus aplicaciones más críticas con niveles de seguridad, confiabilidad y escalabilidad muy altos.

Productivo: Permite reducir el tiempo y los costos requeridos para desarrollar y administrar sus infraestructuras de datos.

Inteligente: Ofrece una plataforma integral que brinda introspectiva e información donde sus usuarios lo desean.

Nuevas funcionalidades de SQL Server 2014

Protección de la Información

¿Cómo podría saber si ha copiado la base de datos del servidor de producción de un cliente y lo han instalado en otra base de datos o si están accediendo a la información? Con SQL 2014, puede proteger la información con una clave de protección (Encriptación).

¿Cómo podría saber que datos están siendo leídos y modificados, a qué hora y por quién? SQL 2014 da la opción de Auditoría de Datos.

Continuidad del Negocio

Si sus clientes necesitan estar siempre en línea con sus sistemas sin caídas, SQL 2014 ofrece mejoras en una técnica llamada “Mirroring”, el cual es una copia o espejo de la base de datos.

Si el disco se daña, donde reside los datos, SQL 2014 recupera la información de una copia reciente de los datos dañados al otro equipo espejo de manera transparente.

Ahorro en espacio en disco, mediante la técnica de comprensión, ahorrando costos en compra de discos si es que el volumen de la información de Base de Datos empieza a crecer en forma rápida.

Datos Geoespaciales

Poder manejar información geográfica, la que hoy en día es de alta importancia en las organizaciones, con todo el tema de la globalización

Acceder a la Información desde cualquier lugar en cualquier momento

Con SQL 2014 podre crear rápidamente aplicaciones conectadas a la base de datos con la funcionalidad de funcionar en forma desconectada y después sincronizarlos con la base de datos central sin perder la línea de negocio y manteniendo los datos validados

Reportes

Poder acceder a reportes directamente desde Word, mejoras en los tipos de gráficos en los reportes, haciéndolos más entendibles y poder editar los reportes de Microsoft Office, sin saber dónde fue diseñado el reporte.

Microsoft SQL Server 2014 es una plataforma de bases de datos y análisis de datos para aplicaciones OLTP (OnLine Transaction Processing – Procesamiento de transacciones en línea), para diseño, creación y administración de data warehouses, y para aplicaciones de comercio electrónico (e-commerce).

Servicios de SQL Server 2014

En función de los componentes que decida instalar, el programa de instalación de SQL Server instalará los servicios siguientes:

Servicios de bases de datos de SQL Server: servicio del Database Engine (Motor de base de datos) relacional de SQL Server.

Agente SQL Server: ejecuta trabajos, supervisa SQL Server, activa alertas y habilita la automatización de algunas tareas administrativas.

Nota:

Para que SQL Server y el Agente SQL Server se ejecuten como servicios de Windows, SQL Server y el Agente SQL Server deben estar asignados a una cuenta de usuario de Windows

Analysis Services: proporciona funciones de procesamiento analítico en línea (OLAP) y minería de datos para aplicaciones de Business Intelligence.

Reporting Services: administra, ejecuta, crea, programa y envía informes.

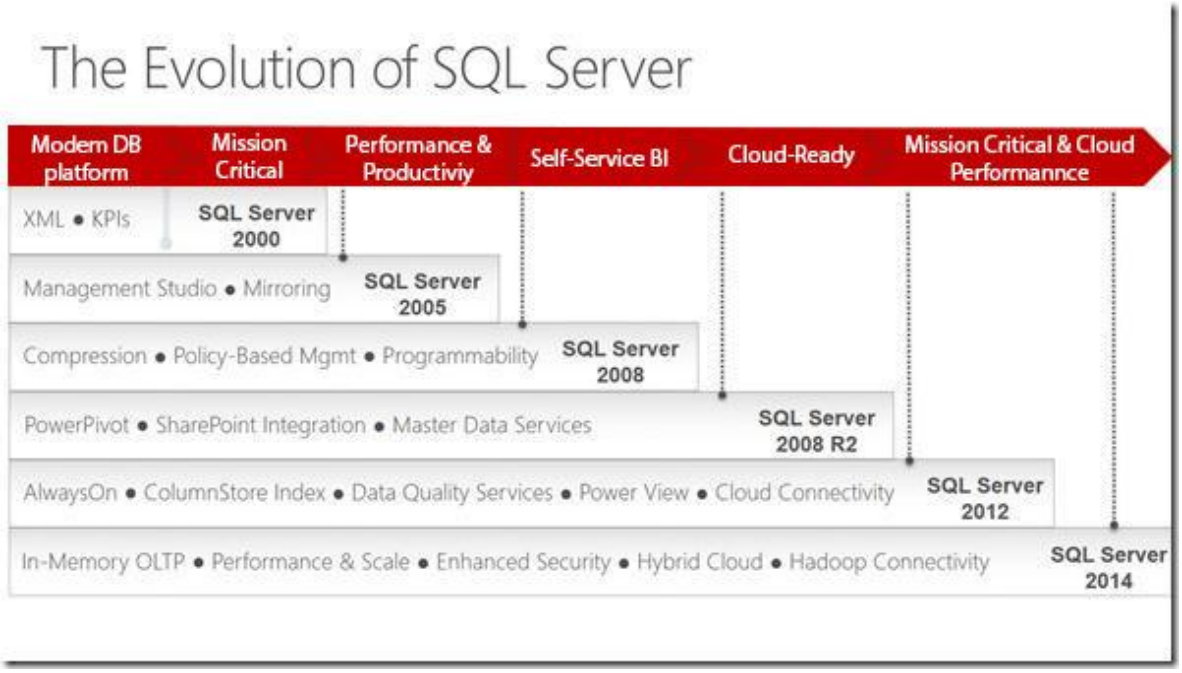
Integration Services: proporciona compatibilidad de administración para el almacenamiento y la ejecución de paquetes de Integration Services.

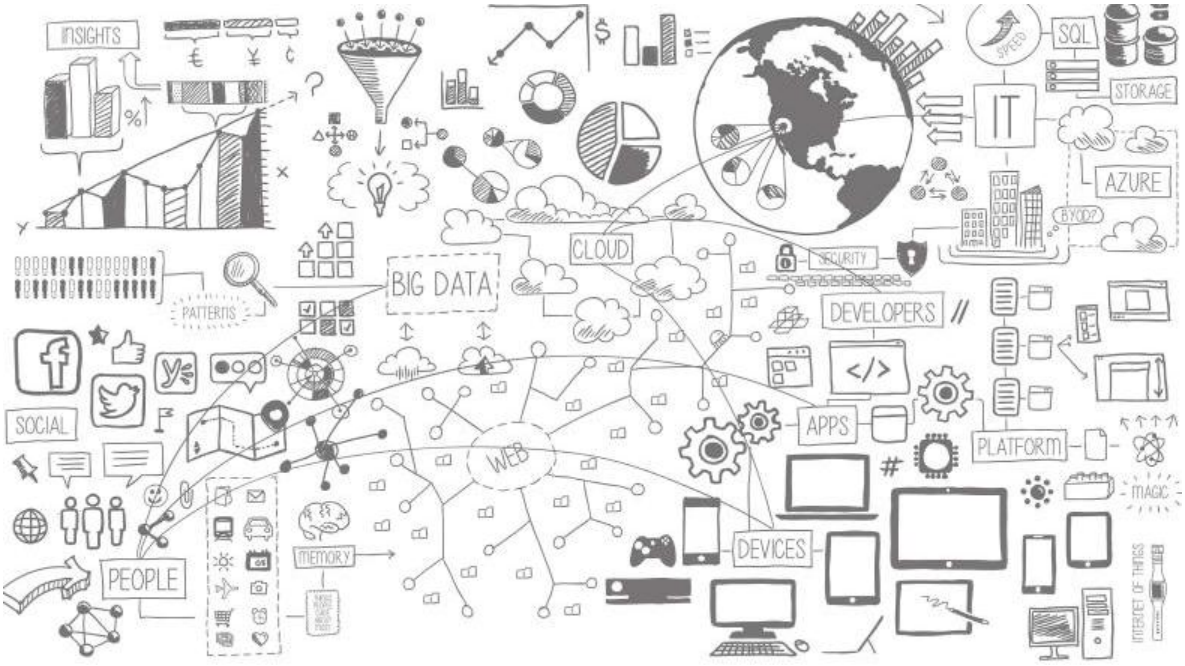
Explorador de SQL Server: servicio de resolución de nombres que proporciona información de conexión de SQL Server a los equipos cliente.

Búsqueda de texto completo: crea rápidamente índices de texto completo del contenido y de las propiedades de los datos estructurados y semiestructurados para permitir el filtrado de documentos y la separación de palabras en SQL Server.

Servicio auxiliar de Active Directory de SQL Server: publica y administra los servicios de SQL Server en Active Directory.

Objeto de escritura de SQL: permite que las aplicaciones de copias de seguridad y restauración funcionen en el marco del Servicio de instantáneas de volumen (VSS).


The Evolution of SQL Server
Modern DB platform
Mission Critical
Performance & Productiviy
Self-Service BI
Cloud-Ready
Mission Critical & Cloud Performannce
XML • KPIs
SQL Server 2000
Management Studio • Mirroring
SQL Server 2005
Compression • Policy-Based Mgmt • Programmability
SQL Server 2008
PowerPivot • SharePoint Integration • Master Data Services
SQL Server 2008 R2
AlwaysOn • ColumnStore Index • Data Quality Services • Power View • Cloud Connectivity
SQL Server 2012
In-Memory OLTP • Performance & Scale • Enhanced Security • Hybrid Cloud • Hadoop Connectivity
SQL Server 2014


INSIGHTS
BIG DATA
CLOUD
SQL
STORAGE
IT
AZURE
SECURITY
DEVELOPERS
PATTERNS
SOCIAL
APPS
WEB
PLATFORM
MAGIC
MEMORY
DEVICES
PEOPLE

Instalación de SQL Server 2014

Microsoft SQL Server 2014 proporciona soporte nativo para las versiones de 32 bits de Microsoft Windows, así como soporte completo para las ediciones de 64 bits de SQL Server que se ejecutan sobre hardware de 64 bits.

El SQL Server 2014 Installation Wizard (Asistente para Instalación de SQL Server 2014) está basado en Windows Installer y proporciona una simple presentación de árbol para la instalación de los componentes de SQL Server: SQL Server 2014 Database Engine, Analysis Services, Reporting Services, Notification Services, Data Transformation Services, Replication, y Management Tools.

El instalador incluye un verificador de pre-instalación que identifica las configuraciones no soportadas de las computadoras y guía al usuario para solucionar los problemas que pudieran presentarse.

Requerimientos de hardware (32 bits)

Hardware	Requerimiento mínimo
Procesador	Intel Pentium ó compatible de 500MHz. Se recomienda 1GHz ó más.
RAM	512MB. Se recomienda 1GB ó más.

Espacio de disco	Componentes de base de datos de SQL Server: 95 a 300 MB, 250 MB típico. Analysis Services: 50 MB mínimo, 130 MB típico. Reporting Services: 50 MB mínimo para Report Server, 30 MB para Report Designer. La instalación del producto completo, incluyendo las bases de datos y código de ejemplo requiere aproximadamente 1 GB.
Monitor	VGA ó superior. Se requiere 1,024x768 ó superior para las herramientas gráficas de SQL Server.

Requerimientos del sistema operativo

Los sistemas operativos soportados incluyen:

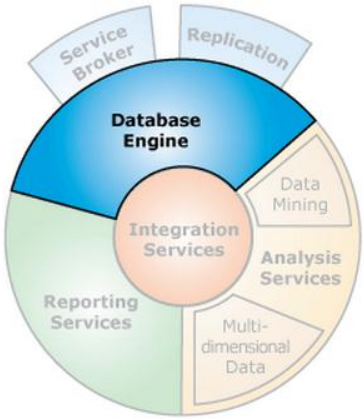
- Windows Server 2008 Standard / Enterprise / Datacenter Edition
- Windows 2003 Embedded / Web Edition
- Windows Small Business Server 2003 Standard / Premium Edition
- Windows 2008 Advanced / Datacenter Server con SP4 ó posterior

El soporte para Web Service nativo (SOAP/HTTP) solo está disponible para instancias de SQL Server 2014 corriendo bajo Windows Server 2003.

Requerimientos de Internet

Componente	Requerimiento
Software de Internet	Se requiere Microsoft Internet Explorer 6.0 SP1 ó posterior para todas las instalaciones de SQL Server 2014, así como para Microsoft Management Console (MMC) y HTML Help.
Internet Information Services (IIS)	Para escribir aplicaciones XML se debe configurar IIS. Se requiere IIS 5.0 ó superior para Reporting Services.

SQL Server Database Engine



Database Engine es el servicio base para almacenamiento, procesamiento y manejo de la seguridad de los datos. Proporciona acceso controlado y procesamiento rápido de transacciones para satisfacer los requerimientos de las aplicaciones empresariales que demandan gran consumo de datos, y soporte para alta disponibilidad de datos.

SQL Server Management Studio

SQL Server Management Studio es un conjunto de herramientas administrativas que permite administrar los componentes que pertenecen a SQL Server. Este entorno integrado permite a los usuarios realizar varias tareas, como realizar copias de seguridad de los datos, editar consultas y automatizar funciones comunes en una misma interfaz.

Las herramientas que SQL Server Management Studio contiene ahora son:

El Editor de código es un editor de secuencias completo que sirve para escribir y modificar secuencias de comandos. El Editor de código reemplaza al Analizador de consultas incluido en versiones anteriores de SQL Server. SQL Server Management Studio ofrece cuatro versiones del Editor de código: el Editor de consultas de SQL, el Editor de consultas MDX, el Editor de consultas XML y el Editor de consultas de Microsoft SQL Server Compact Edition.

El Explorador de objetos sirve para buscar, modificar e incluir objetos que pertenecen a instancias de SQL Server en secuencias de comandos o ejecutarlos.

El Explorador de plantillas sirve para buscar plantillas y crear secuencias de comandos para ellas.

El Explorador de soluciones sirve para organizar y almacenar secuencias de comandos relacionadas como partes de un proyecto.

La Ventana Propiedades sirve para mostrar las propiedades actuales de los objetos seleccionados.

SQL Server Management Studio también ofrece nueva funcionalidad, que incluye:

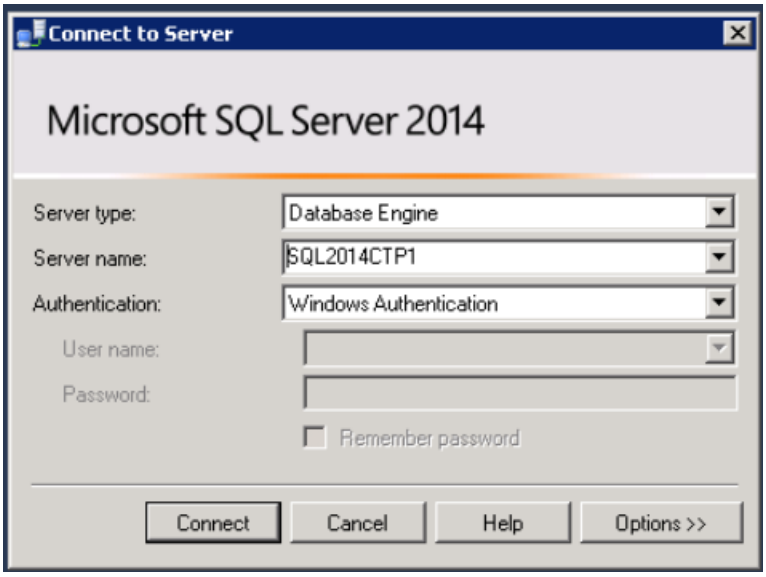
Acceso sin conexión. Puede escribir y modificar secuencias de comandos sin conectarse a una instancia de SQL Server.

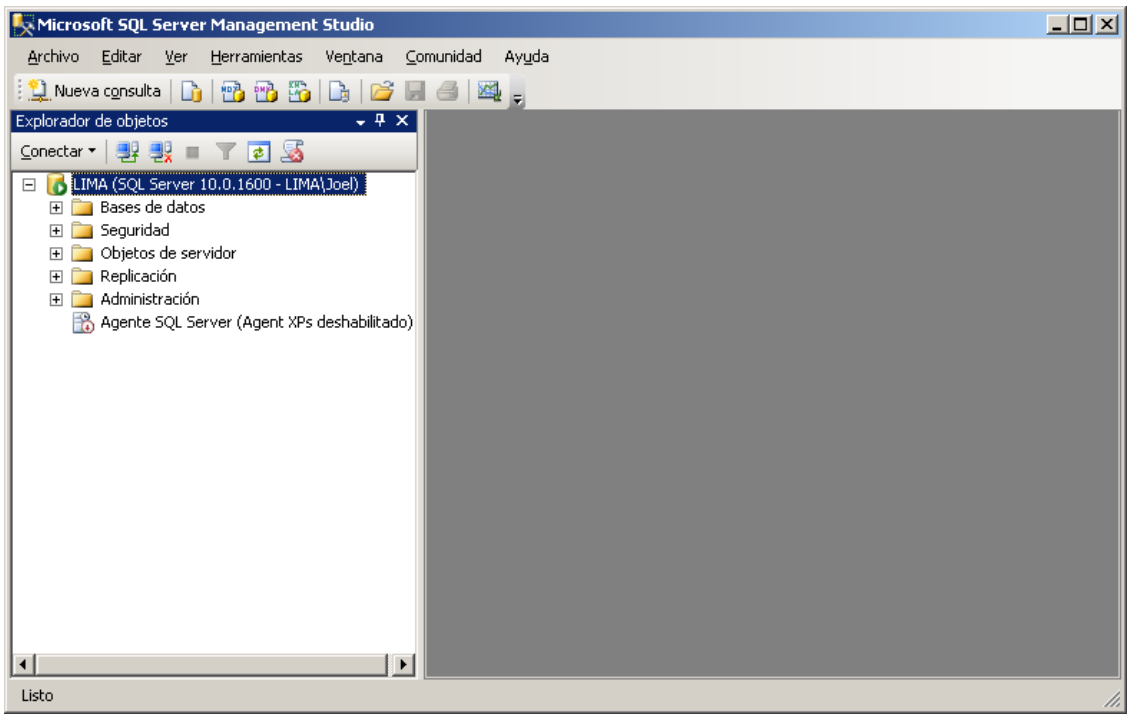
Creación de secuencias de comandos desde cualquier cuadro de diálogo. Puede crear una secuencia de comandos desde cualquier cuadro de diálogo para que pueda leer, modificar, almacenar y reutilizar las secuencias de comandos después de crearlas.

Cuadros de diálogo no modales. Al obtener acceso a un cuadro de diálogo de interfaz de usuario, puede examinar otros recursos en SQL Server Management Studio sin cerrar el cuadro de diálogo

Iniciar Server Management Studio

1. Para cargar la herramienta ejecute la secuencia botón **Inicio**, **Todos los programas**, **Microsoft SQL Server 2014**, **SQL Server Management Studio**.
2. Se abre la ventana **Conectar con el servidor**. En ella, verifique que en Tipo de servicio está seleccionado SQL Server.
3. En Nombre del servidor, seleccione el servidor SQL con el que se desea conectar.
4. Indique el tipo de autenticación con el que se identifica ante el servidor, y haga clic en **Connect**.
5. Se abre el **SQL Server Management Studio** con la conexión al servidor indicado.


Connect to Server
Microsoft SQL Server 2014
Server type:
Database Engine
Server name:
SQL2014CTP1
Authentication:
Windows Authentication
User name:
Password:
Remember password
Connect
Cancel
Help
Options >>


Microsoft SQL Server Management Studio
Archivo Editar Ver Herramientas Ventana Comunidad Ayuda
Nueva consulta
Explorador de objetos
Conectar
LIMA (SQL Server 10.0.1600 - LIMA\Joel)
Bases de datos
Seguridad
Objetos de servidor
Replicación
Administración
Agente SQL Server (Agent XPs deshabilitado)
Listo

El proceso de autenticación

Es un proceso que se utiliza para comprobar que una entidad u objeto es quien dice ser, en el caso de SQL Server 2014 es el proceso que permite verificar las credenciales de un usuario que desea acceder a un servidor SQL.

.Tipos de autenticación en SQL Server

En SQL Server tenemos dos tipos de autenticación:

- **Autenticación integrada a Windows:** cuando con la misma cuenta con que accedemos a Windows podemos acceder a SQL Server. Para que esto sea posible, se requiere que la cuenta de usuario Windows (por ejemplo, la cuenta CEPS\PC-402-01 con la que ingreso a la red Windows) esté registrada también como cuenta de inicio de sesión (login name) de SQL Server.
- **Autenticación SQL:** cuando después de haber accedido a Windows con la cuenta de usuario de Windows (por ejemplo, la cuenta CEPS\PC-402-01), accedemos a SQL Server utilizando una cuenta de inicio de sesión ó login name de SQL Server (por ejemplo, sa).

Como se ve, cualquiera que sea el tipo de autenticación, si no contamos con un login name, no podemos acceder a SQL Server. Este login name puede ser una cuenta Windows registrada como login name, ó un login name estándar de SQL.

Modos de autenticación

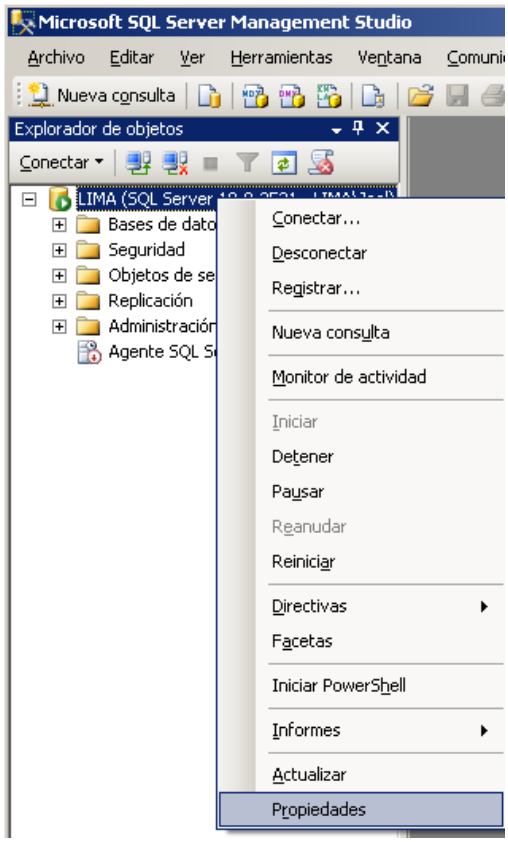
Durante la instalación, debe seleccionar un modo de autenticación para Database Engine (Motor de base de datos). Hay dos modos posibles: modo de autenticación de Windows y modo mixto. El modo de autenticación de Windows habilita la autenticación de Windows y deshabilita la autenticación de SQL Server. El modo mixto habilita tanto la autenticación de Windows como la de SQL Server. La autenticación de Windows está disponible siempre y no se puede deshabilitar

Un servidor SQL puede ser configurado para aceptar cualquiera de los siguientes dos modos de autenticación:

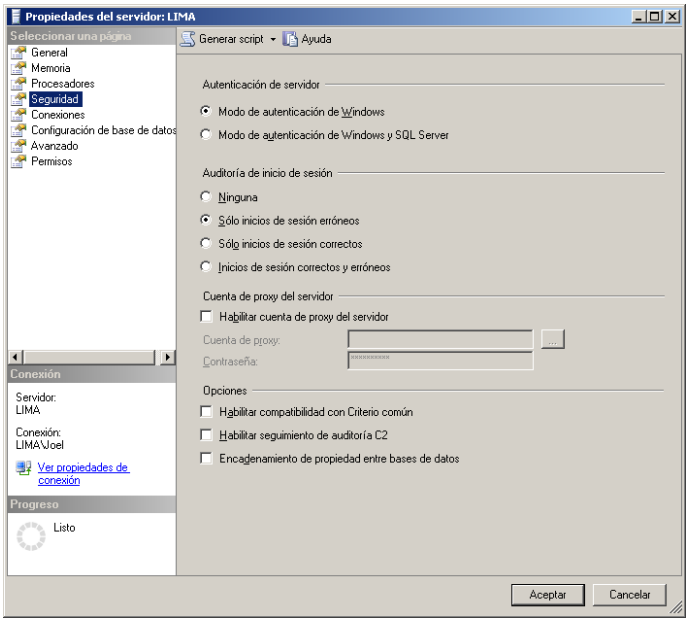
- **Solo Windows:** solo se puede acceder utilizando la autenticación integrada a Windows.
- **Modo mixto:** se puede acceder con ambos tipos de autenticación: la autenticación integrada a Windows, y la autenticación SQL.

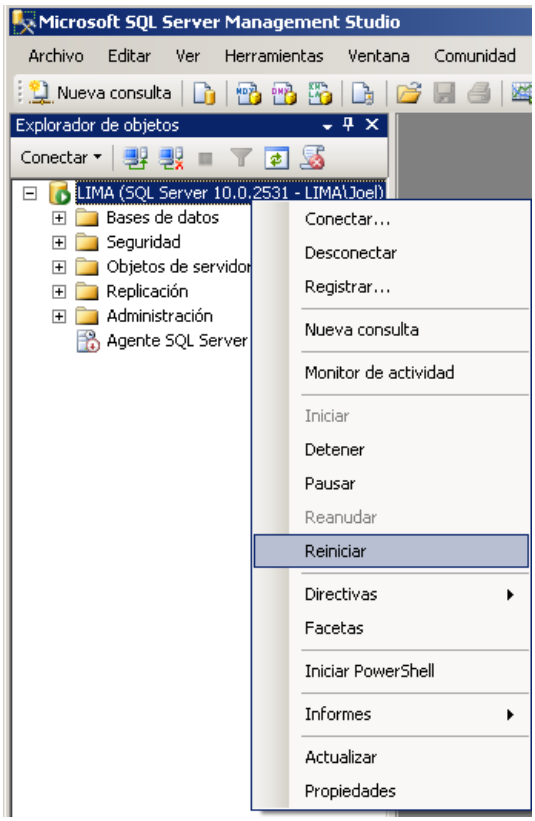
Verificación del modo de autenticación de un servidor SQL

1. En la ventana Explorador de objetos haga un clic secundario sobre el nodo del servidor a verificar, y luego seleccione Propiedades
2. Para verificar el modo de autenticación, seleccione la página Seguridad...



Para cambiar el modo de autenticación seleccione el modo y luego tiene que reiniciar el servicio.





Las bases de datos de sistema

Cuando se instala SQL Server, se crean cuatro bases de datos de sistema, dos bases de datos de usuario, y dos bases de datos del Report Server.
Las bases de datos de sistema contienen las tablas de sistema, las que a su vez contienen metadatos; es decir, los datos que permiten operar y administrar el sistema. Las bases del sistema son: master, model, tempdb y msdb.

Base de datos del sistema master

La base de datos master registra toda la información del sistema de SQL Server. Registra todas las cuentas de inicio de sesión y todas las opciones de configuración del sistema. La base de datos master registra la existencia del resto de bases de datos, incluida la ubicación de los archivos de base de datos. La base de datos master registra la información de inicialización de SQL Server, y siempre mantiene disponible una copia de seguridad reciente de master.

Base de datos del sistema model

La base de datos model se utiliza como plantilla para todas las bases de datos creadas en un sistema. Cuando se emite una instrucción CREATE DATABASE, la primera parte de la base de datos se crea copiando el contenido de la base de datos model, el resto de la nueva base de datos se llena con páginas vacías. Como tempdb se crea de nuevo cada vez que se inicia SQL Server, la base de datos model siempre tiene que existir en un sistema SQL Server.

Base de datos del sistema tempdb

La base de datos tempdb almacena todas las tablas y todos los procedimientos almacenados temporales. También satisface otras necesidades de almacenamiento temporal, como las tablas de trabajo generadas por SQL Server. La base de datos tempdb es un recurso global; las tablas y los procedimientos almacenados temporales de todos los usuarios conectados al sistema se almacenan en ella; tempdb se vuelve a crear cada vez que se inicia SQL Server, de forma que el sistema se inicia con una copia limpia de la base de datos. Como las tablas y los procedimientos almacenados temporales se eliminan automáticamente al desconectar, y

cuando se cierra el sistema no hay conexiones activas, en tempdb nunca hay nada que se tenga que guardar de una sesión de SQL Server a otra.

Base de datos del sistema msdb

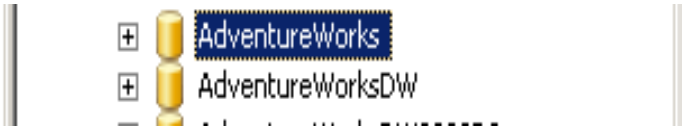
La base de datos msdb la utiliza el Agente SQL Server para programar alertas y trabajos, y para registrar operadores.

Base de datos del sistema

Base de datos de ejemplo

Las creadas durante la instalación del servidor sirven como bases de datos de ejemplo.

- **AdventureWorks** (usuario): Base de datos de Adventure Works Cycles, una multinacional que fabrica y vende bicicletas a nivel mundial.
- **AdventureWorksDW** (usuario): Base de datos de análisis de Adventure Works Cycles.



Utilizar SQL Server Manangement Studio para emitir consultas

Transact SQL

Transact SQL es el lenguaje utilizado por SQL Server 2014 para escribir comandos que luego serán enviados al servidor para su ejecución.

Transact SQL es la implementación de la empresa Microsoft del lenguaje estándar de consultas estructurado (SQL).

Transact SQL no es CASE-SENSITIVE, es decir, no diferencia mayúsculas de minúsculas como otros lenguajes de programación como C o Java.

Comentarios

Un comentario es una aclaración que el programador incluye en el código. Son soportados 2 estilos de comentarios, el de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales como son:

-- Para un comentario de línea simple

/* ... */ Para un comentario de varias líneas

Scripts y lotes.

Un script de Transact SQL es un conjunto de sentencias de Transact SQL en formato de texto plano que se ejecutan en un servidor de SQL Server. Un script está compuesto por uno o varios lotes.

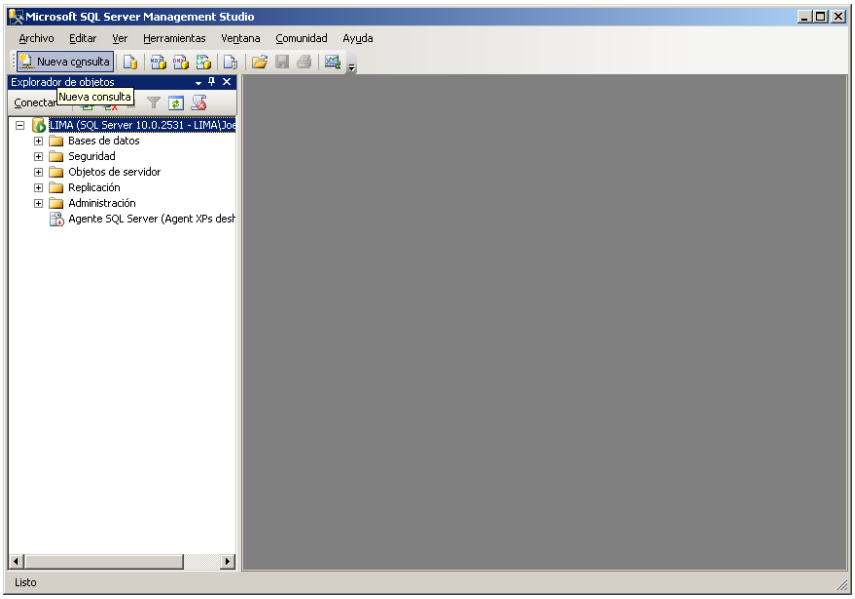
Un lote delimita el alcance de las variables y sentencias del script. Dentro de un mismo script se diferencian los diferentes lotes a través de la instrucción GO.

En ocasiones es necesario separar las sentencias en varios lotes, porque Transact SQL no permite la ejecución de ciertos comandos en el mismo lote, si bien normalmente también se utilizan los lotes para realizar separaciones lógicas dentro del script.

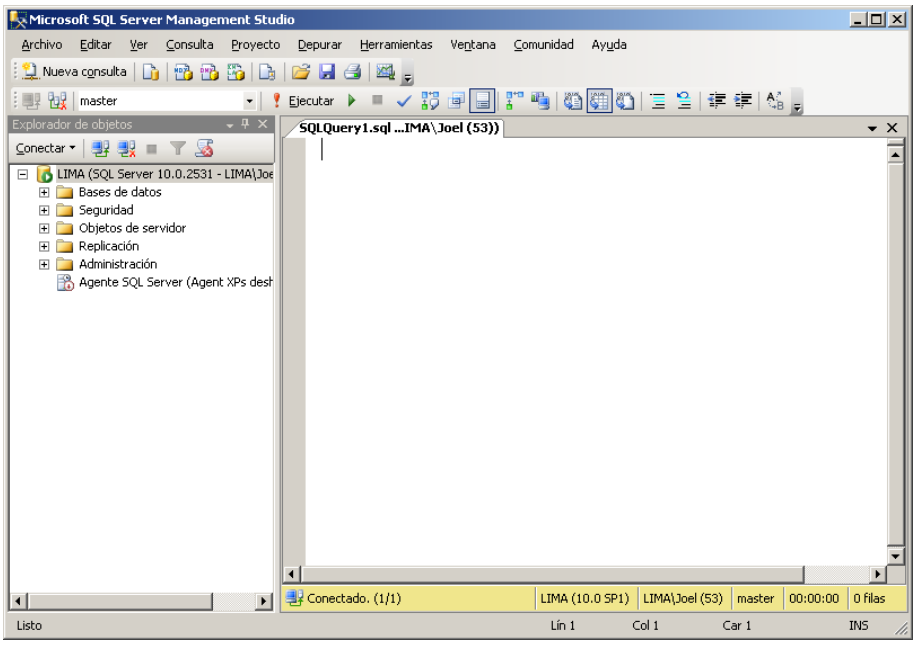
Consultas utilizando Transact SQL

En primer lugar activar Microsoft SQL Server Manangement Studio y conectar al servidor.

A continuación dar un clic en la pestaña Nueva Consulta



Se activar la ventana de edición de código SQL Query1.sql

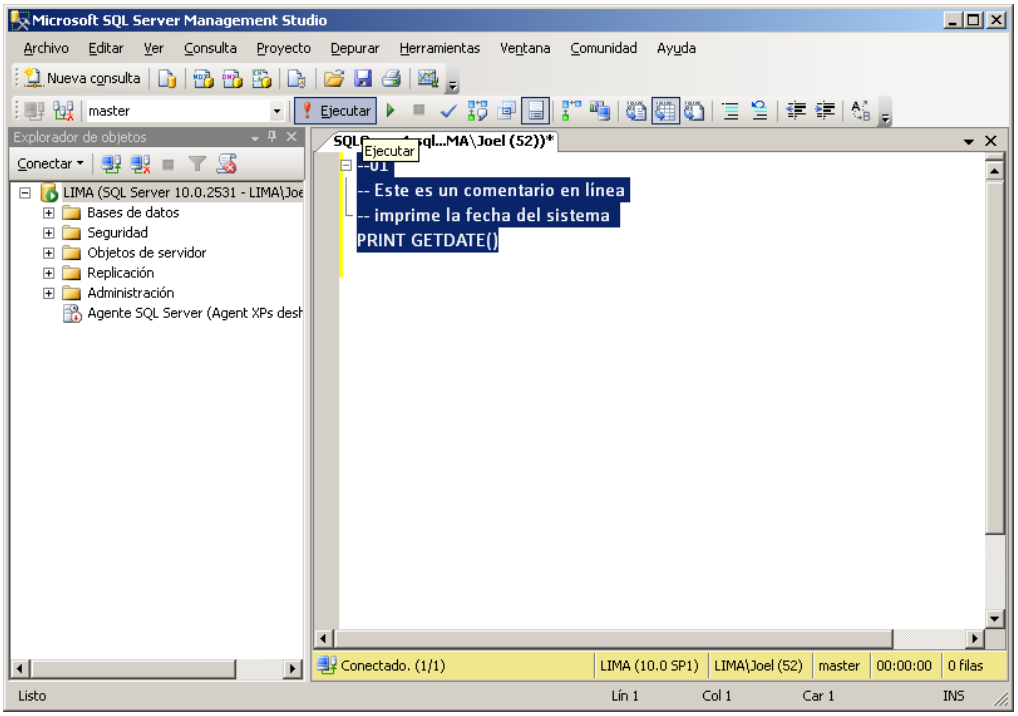


En la ventana de edición activa se debe empezar a escribir los comandos Transact SQL.

Para empezar digite lo siguiente:

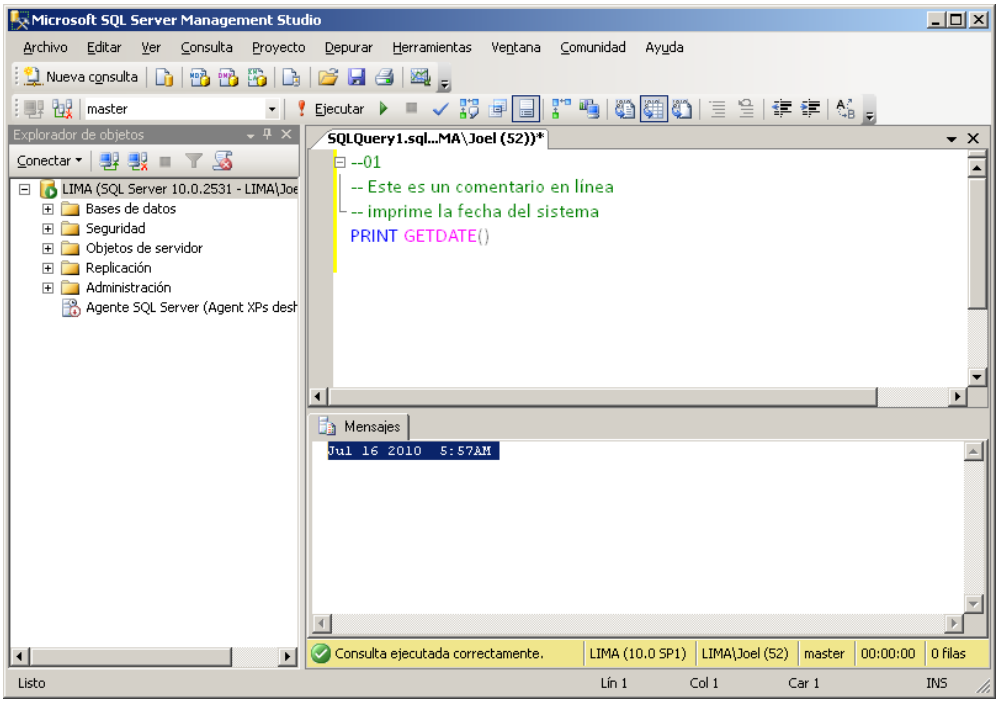
Ejercicio 01

```
-- Este es un comentario en línea

-- imprime la fecha del sistema

PRINT GETDATE()

--observe el panel de resultados
```



Luego seleccione el bloque de código que desea ejecutar y dar un clic en el botón Ejecutar o pulsar la tecla **F5**.

En la siguiente figura observe el panel Mensajes, para el ejemplo anterior se muestra la fecha del sistema. El comando PRINT se utiliza en este ejemplo para mostrar el resultado de la invocación a la función GETDATE()



Para continuar escriba lo siguiente:

Ejercicio 02

```
PRINT  'Estamos utilizando Transact-SQL'

PRINT  'para escribir un conjunto de comandos SQL'
```

Ejercicio 03

```
-- se declara la variable @Fecha de tipo fecha

DECLARE @Fecha Datetime

-- se asigna a la variable  @Fecha la fecha del sistema

SET @Fecha = GETDATE()

-- se imprime la fecha

PRINT @Fecha
```

Ejercicio 04

--04 Ejecutar una consulta a la tabla Products de la base de datos AdventureWorks

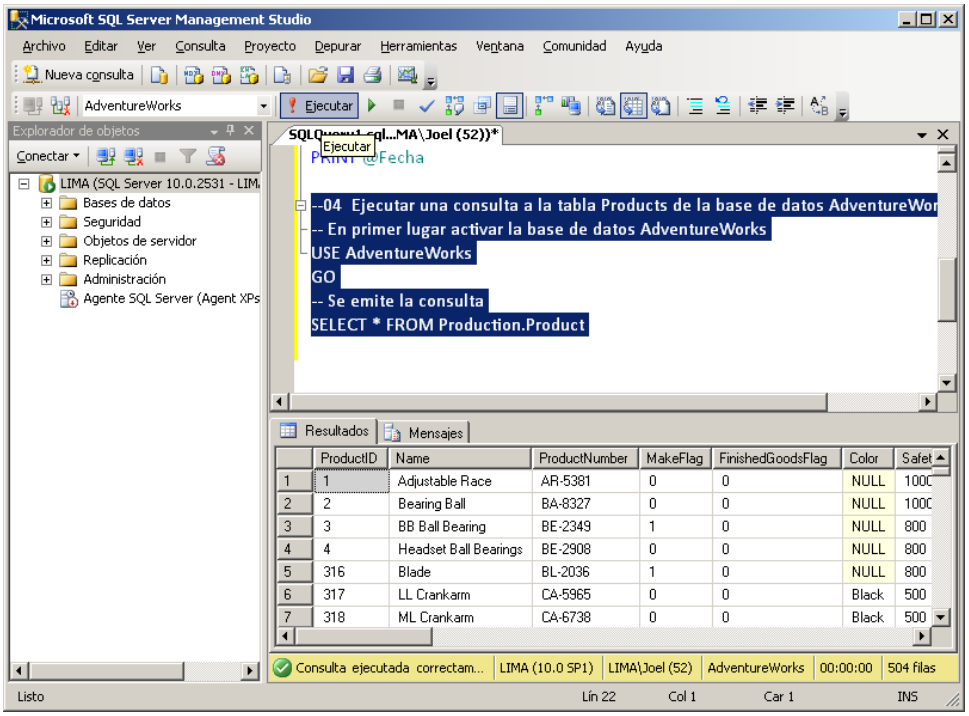
-- En primer lugar activar la base de datos AdventureWorks

USE AdventureWorks

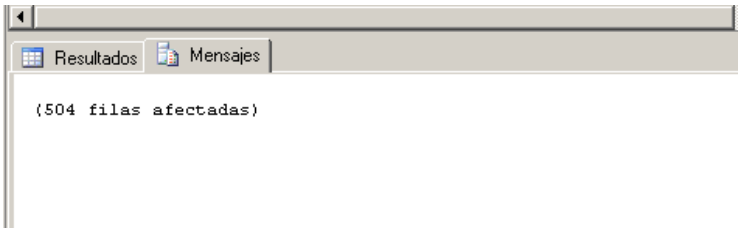
GO

-- Se emite la consulta

SELECT * FROM Production.Product



Seleccione cada ejercicio y ejecute el bloque o lote, en el gráfico anterior se muestra en la pestaña Resultados el contenido de la tabla Product de la base de datos AdventureWorks. En la pestaña Mensajes se muestra el número de filas afectadas.



El Script creado se puede grabar y luego recuperar el contenido del archivo para una posterior utilización.

Las tablas del sistema

Las tablas de sistema almacenan metadatos acerca del sistema y de los objetos de la base de datos.

El Catálogo de la Base de Datos

Cada una de las bases de datos, ya sea de sistema o de usuario, contiene una colección de tablas de sistema que almacenan datos que describen a una base de datos específica. Esta colección de tablas se conoce como el Catálogo de la Base de Datos, y pertenecen al esquema sys de la base de datos.

A partir de esta versión de SQL Server, las tablas de sistema no pueden ser consultadas directamente. Para consultar el Catálogo de una base de datos podemos utilizar las Vistas del Catálogo (Catalog Views) ó las Vistas de Esquema de Información (Information Schema Views).

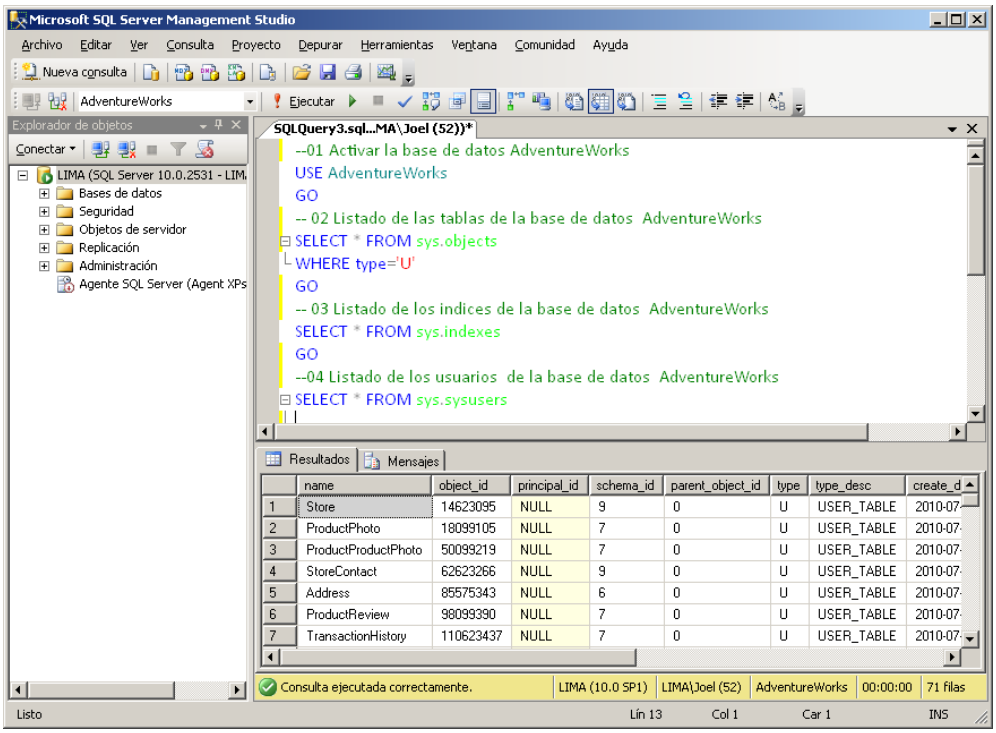
El siguiente cuadro muestra algunas de las vistas del catálogo que forman el Catálogo de la Base de Datos.

Vista del Catálogo	Base de datos	Descripción
sys.sysobjects	Todas	Tiene una fila por cada objeto almacenado en la base de datos.
sys.sysusers	Todas	Tiene una fila por cada usuario que tiene acceso a la base de datos.
sys.sysindexes	Todas	Tiene una fila por cada índice creado para las tablas de la bases de datos.

A continuación se presentan ejemplos del uso de las vistas del catálogo de base de datos:

Ejercicio 05

```
-- Activar la base de datos AdventureWorks
USE AdventureWorks
GO
-- Listado de las tablas de la base de datos  AdventureWorks
SELECT * FROM sys.objects
WHERE type='U'
-- Listado de los índices de la base de datos  AdventureWorks
SELECT * FROM sys.indexes
GO
-- Listado de los usuarios  de la base de datos  AdventureWorks
SELECT * FROM sys.sysusers
GO
```



El Catálogo del Sistema

Es una colección de tablas de sistema que almacenan datos acerca del sistema y todas las bases de datos. Esta colección de tablas solo está presente en la base de datos de sistema master.

El siguiente cuadro muestra algunas de las vistas del catálogo que forman el Catálogo del Sistema.

Tabla de sistema	Base de datos	Descripción
sys.databases	master	Tiene una fila por cada base de datos en el servidor.
sys.syslogins	master	Tiene una fila por cada login name que puede conectarse al servidor.
sys.sysmessages	master	Tiene una fila por cada mensaje de error del sistema o advertencia que el servidor puede retornar.

Consultas a los Catálogos

Cuando se desea obtener información de un catálogo podemos hacerlo mediante una consulta a una vista del catálogo, utilizando un procedimiento almacenado del sistema, o mediante vistas de esquemas de información.

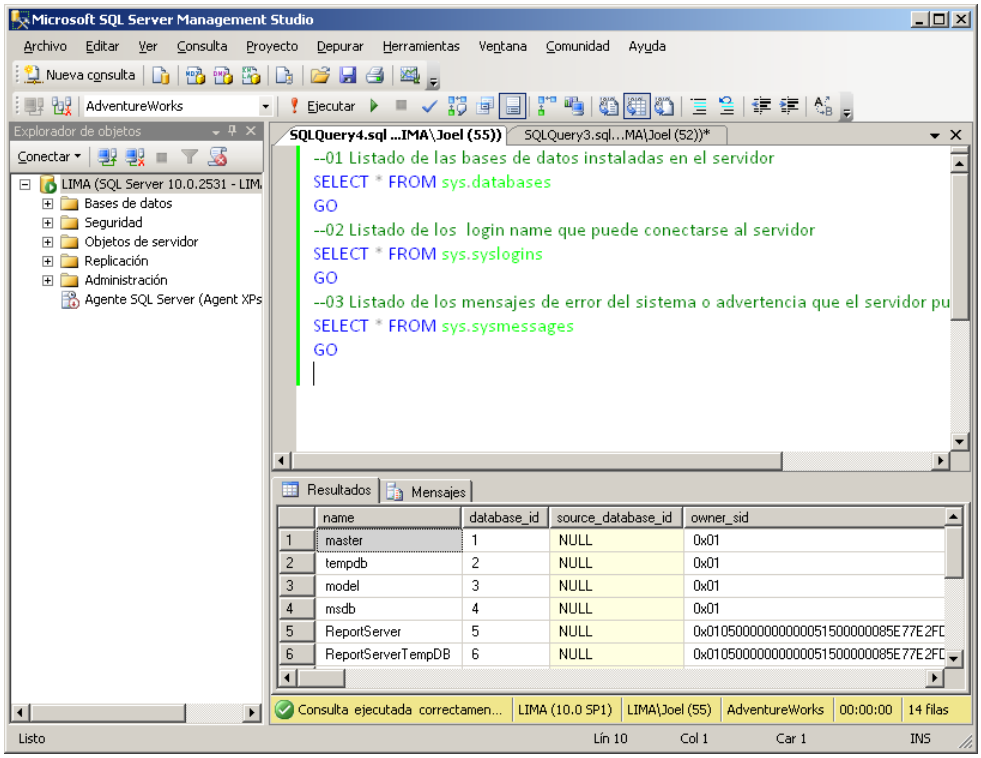
Ejemplos:

Ejercicio 06

```
--01 Listado de las bases de datos instaladas en el servidor
SELECT * FROM  sys.databases
GO
```

```
--02 Listado de los  login name que puede conectarse al servidor
SELECT * FROM sys.syslogins
GO
--03 Listado de los mensajes de error del sistema o advertencia
--  que el servidor puede retornar.
SELECT * FROM sys.sysmessages
GO
```

En la siguiente figura se muestra un listado de las bases de datos instaladas en el servidor.



Con vista de esquema de información

Una vista es una consulta pre construida que se almacena en la base de datos, y que ser ejecutada por un programa. SQL Server cuenta con un conjunto de vistas denominadas de esquema de información que permiten averiguar sobre el esquema de una base de datos.

El identificador de estas vistas tiene la forma INFORMATION_SCHEMA. objetos
Ejemplos:

Ejercicio 07

--01 Muestra un listado de las tablas en la base de datos AdventureWorks

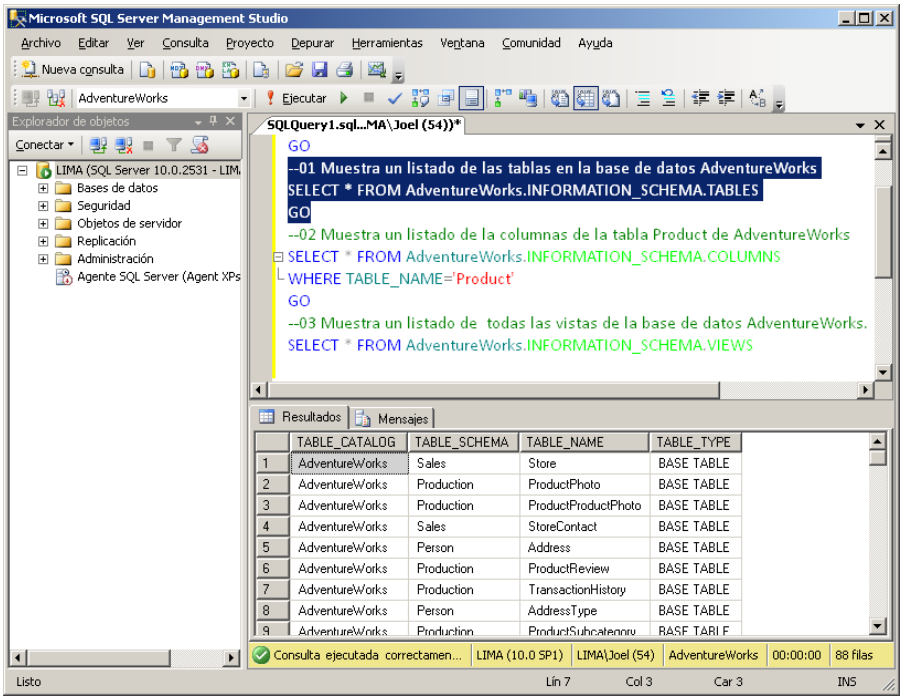
```
SELECT * FROM AdventureWorks.INFORMATION_SCHEMA.TABLES
GO
```

--02 Muestra un listado de la columnas de la tabla Product de AdventureWorks

```
SELECT * FROM AdventureWorks.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME='Product'
```

--03 Muestra un listado de todas las vistas de la base de datos AdventureWorks.

```
SELECT * FROM AdventureWorks.INFORMATION_SCHEMA.VIEWS
```



Utilizando funciones de metadatos

SQL Server define varias categorías de funciones que retornan información acerca de la base datos activa y objetos de la base de datos. Estas funciones retornan solo un valor y son conocidas como funciones escalares

Ninguna de las funciones de metadatos es determinista. Esto significa que no siempre devuelven los mismos resultados al ser llamadas, aunque el conjunto de valores de entrada sea el mismo.

DB_ID (): Retorna el identificador de la base de datos activa.

DB_NAME() : Retorna el nombre de la base de datos para el ID especificado.

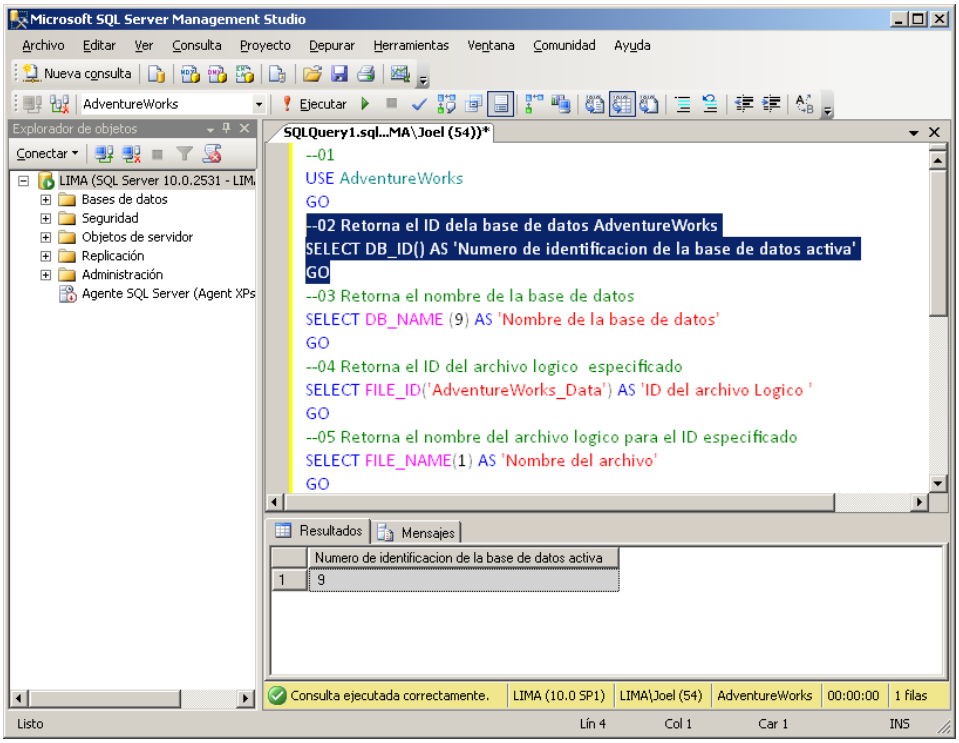
FILE_ID(): Retorna el identificador del archivo lógico especificado.

FILE_NAME(): Retorna el nombre del archivo lógico para el ID especificado.

Ejercicio 08

```
--01
USE AdventureWorks
GO
--02 Retorna el ID dela base de datos AdventureWorks
SELECT DB_ID() AS 'Número de identificación de la base de datos activa'
GO
--03 Retorna el nombre de la base de datos
SELECT DB_NAME (9) AS 'Nombre de la base de datos'
GO
--04 Retorna el ID del archivo lógico especificado
SELECT FILE_ID('AdventureWorks_Data') AS 'ID del archivo Logico '
GO
--05 Retorna el nombre del archivo lógico para el ID especificado
SELECT FILE_NAME(1) AS 'Nombre del archivo'
```

GO



Utilizando Procedimientos almacenados del sistema

Se pueden realizar muchas actividades administrativas e informativas de Microsoft SQL Server a través de los procedimientos almacenados del sistema.

A continuación se explicará brevemente un procedimiento almacenado del sistema (ya que como observará más adelante éste se usará con frecuencia)

El procedimiento almacenado del sistema sp_helpdb

Este procedimiento almacenado del sistema presenta información acerca de una base de datos especificada o de todas las bases de datos.

Sintaxis

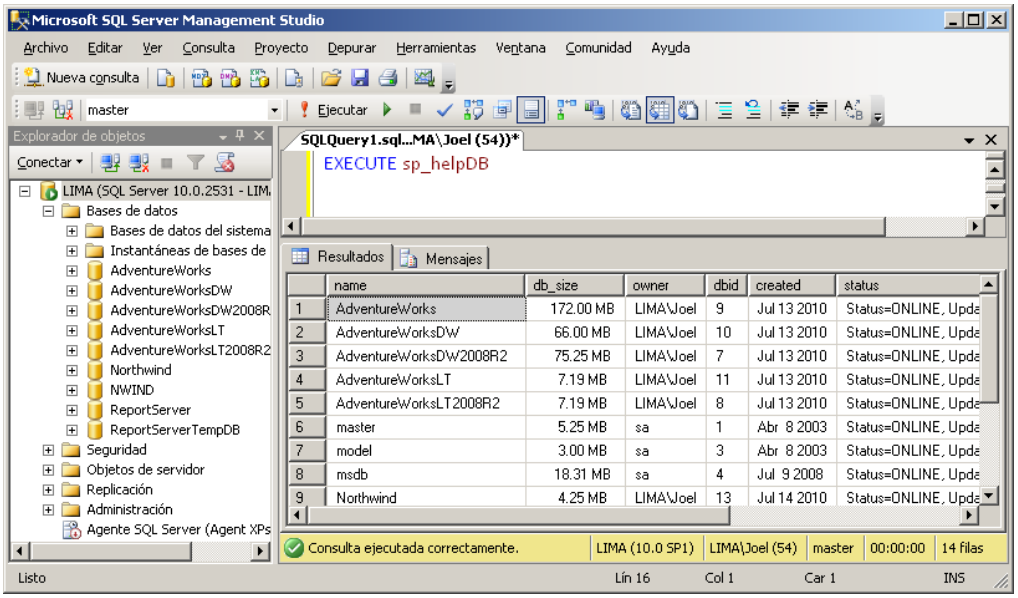
sp_helpdb [[@dbname=] 'name']

Argumentos: name

Es el nombre de la base de datos cuya información se va a presentar. Si no se especifica nombre, sp_helpdb devuelve información acerca de todas las bases de datos master.dbo.sysdatabases.

Ejercicio 09

EXECUTE sp_helpDB



Al ejecutarse muestra la información de todas las bases de datos instaladas en el servidor

Ejercicio 10

EXECUTE sp_helpDB AdventureWorks

Al ejecutar se presentará la información de la base de datos AdventureWorks.

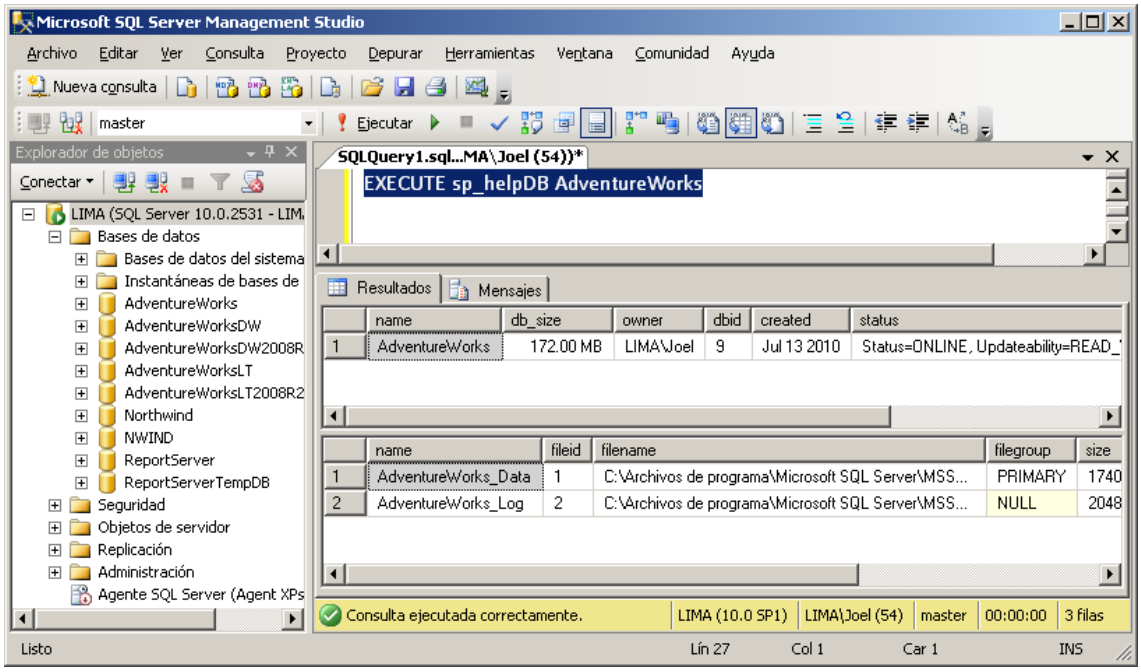
Ejecutar un procedimiento almacenado

Para ejecutar un procedimiento almacenado, utilice la instrucción EXECUTE de Transact-SQL. Puede ejecutar un procedimiento almacenado sin necesidad de utilizar la palabra clave EXECUTE, si el procedimiento almacenado es la primera instrucción del lote.

Ejercicio 11

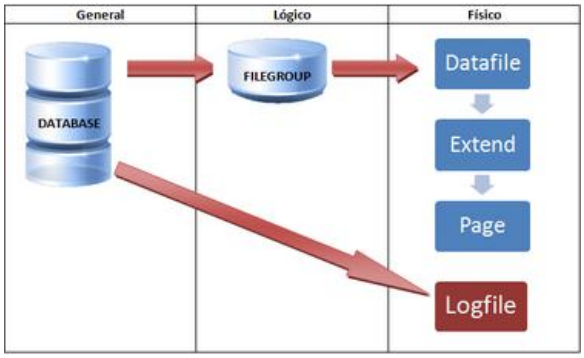
EXECUTE sp_helpDB AdventureWorks

Ejecute la sentencia anterior pulsando la tecla F5 y el resultado en modo cuadrícula. Los resultados se muestran en la siguiente gráfico:



2

Creación de base de datos



En la actualidad se reconoce que la información y la calidad que esta posee, además del tratamiento automatizado no sólo es necesario para el eficiente funcionamiento de toda organización sino que se ha convertido en uno de los principales elementos de competitividad. Por este motivo el almacenamiento de los datos y su pronta disponibilidad para los usuarios se hace indispensable para el manejo y funcionamiento de la organización.

La arquitectura Cliente / Servidor presenta dos componentes: programas que proporcionan una interfaz al usuario del sistema que permitan tener acceso a los datos y un programa gestor de base de datos que faciliten la administración y mantenimiento de los datos en el servidor. Así por ejemplo si está interesado en crear una aplicación de facturación deberá crear una estructura de base de datos que permita administrador las diferentes transacciones que se realicen en la empresa, además debe tener disponibles una interfaz que permita a los usuarios manipular los datos que necesite.

Base de datos

Una base de datos es una colección de objetos que contienen y manejan datos. Los objetos que se pueden almacenar en una base de datos son:

- Tablas
- Vistas
- Procedimientos Almacenados
- Opciones predeterminadas
- Reglas
- Disparadores
- Indicies
- Tipos de datos
- Restricciones
- Diagramas

Estos objetos se definen para poder realizar distintas operaciones con los datos que se almacenan en una base de datos, deben estar relacionados con un determinado tema o un proceso determinado como por ejemplo un sistema de pedidos.
Los componentes de datos de SQL Server están compuestos de un conjunto de tablas en la que se almacena un conjunto específico de datos que están estructurados. Una tabla está conformada por un conjunto de filas (registros o tuplas) y columnas (Atributos).

Archivos de base de datos.

Como mínimo, todas las bases de datos de SQL Server 2014 tienen dos archivos del sistema operativo: un archivo de datos y un archivo de registro. Los archivos de datos contienen datos y otros objetos, como tablas, índices, procedimientos almacenados y vistas. Los archivos de registro contienen la información necesaria para recuperar todas las transacciones de la base de datos. Los archivos de datos se pueden agrupar en grupos de archivos para su asignación y administración.

Tipos de archivos de la base de datos

Las bases de datos de SQL Server 2014 tienen tres tipos de archivos: principal, secundario y de registro de transacciones.

Principal

El archivo de datos principal incluye la información de inicio de la base de datos y apunta a los demás archivos de la misma. Los datos y objetos del usuario se pueden almacenar en este archivo o en archivos de datos secundarios. Cada base de datos tiene un archivo de datos

principal. La extensión recomendada para los nombres de archivos de datos principales es .**mdf**.

Secundario

Los archivos de datos secundarios son opcionales, están definidos por el usuario y almacenan los datos del usuario. Se pueden utilizar para distribuir datos en varios discos

Colocando cada archivo en una unidad de disco distinta. Además, si una base de datos supera el tamaño máximo establecido para un archivo de Windows, puede utilizar los archivos de datos secundarios para permitir el crecimiento de la base de datos. La extensión recomendada para los nombres de archivos de datos secundarios es .**ndf**.

Registro de transacciones

Los archivos del registro de transacciones contienen la información de registro que se utiliza para recuperar la base de datos. Cada base de datos debe tener al menos un archivo de registro. La extensión recomendada para los nombres de archivos de registro es **.ldf**.

Por ejemplo, puede crearse una base de datos sencilla denominada Ventas con un archivo principal que contenga todos los datos y objetos y un archivo de registro con la información del registro de transacciones. Por otra parte, puede crearse una base de datos más compleja, Pedidos, compuesta por un archivo principal y cinco archivos secundarios. Los datos y objetos de la base de datos se reparten entre los seis archivos, y cuatro archivos de registro adicionales contienen la información del registro de transacciones.

De forma predeterminada, los datos y los registros de transacciones se colocan en la misma unidad y ruta de acceso para administrar los sistemas de un solo disco, pero puede que esto no resulte óptimo para los entornos de producción. Se recomienda colocar los archivos de datos y de registro en distintos discos.

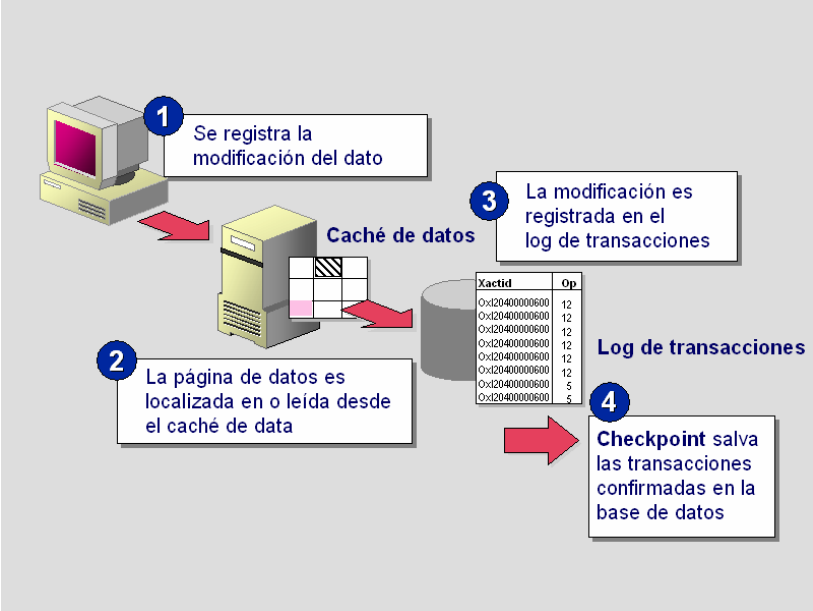
Concepto de Transacción

Una **transacción** es un conjunto de modificaciones de datos (operaciones) que debe ser procesado como una unidad.

Una transacción asegura que las operaciones se llevarán a cabo completas, o en caso contrario, la transacción será anulada para garantizar la consistencia de los datos.

El registro de transacciones de SQL Server

El archivo de registro de transacciones almacena todas las transacciones llevadas a cabo con las sentencias INSERT, UPDATE y DELETE. El proceso se lleva a cabo de la siguiente manera:



- Una aplicación envía una modificación de datos al servidor SQL.
- Las páginas de datos afectadas por la modificación son localizadas en la memoria caché, o leídas desde el disco a la memoria caché, si no se encuentran en ella.
- Cada sentencia de modificación de datos es registrada en el transaction log antes que el cambio se lleve a cabo en la base de datos.
- El punto de control (checkpoint) escribe las transacciones confirmadas (commited) en la base de datos.
- Si el sistema falla, el proceso de recuperación automático usa el registro de transacciones para recuperar todas las transacciones confirmadas y anula todas las transacciones incompletas.

Creación de una base de datos

Para crear una base de datos, debe determinar su nombre, su propietario, su tamaño, los archivos y los grupos de archivos que se utilizarán para almacenarla.

Cuando se crea una base de datos SQL Server crea automáticamente un registro de Transacciones para ella. Esto se registra en un archivo especial donde se guarda toda actividad relacionada con dicha base de datos, siempre que un usuario modifique o elimine información de la base de datos el registro de transacciones tomara nota de la acción. Por defecto, estos registros tienen el mismo nombre que la base de datos, pero utilizan la extensión. .ldf, los archivos de transacciones proporcionan cierto nivel de seguridad a los datos. En caso de falla, es posible aplicar las transacciones que se encuentran en este archivo en otra copia de la base de datos.

La instrucción CREATE DATA BASE (T-SQL)

La instrucción CREATE DATA BASE crea una base de datos y los archivos que se utilizan almacenar la base de datos.

Sintaxis:

```
CREATE DATABASE nombre_basedatos
ON [ PRIMARY ] (
        NAME = nombre_lógico_data ,
        FILENAME = 'ubicación_y_nombre_archivo_data' ,
        SIZE = tamaño [ KB|MB|GB|TB ] ,
        MAXSIZE = tamaño_máximo [ KB|MB|GB|TB|UNLIMITED ] ,
        FILEGROWTH = incremento_crecimiento [ KB|MB|% ] )

LOG ON (
        NAME = nombre_lógico_log ,

        FILENAME = 'ubicación_y_nombre_archivo_log' ,
```

SIZE = tamaño [KB|MB|GB|TB] ,

MAXSIZE = tamaño_máximo [KB|MB|GB|TB|UNLIMITED] ,

FILEGROWTH = incremento_crecimiento [KB|MB|%])

- La cláusula LOG ON define las propiedades del archivo de registro de transacciones.
- *nombre_lógico_data*, *nombre_lógico_log* es el nombre a utilizar cuando en una sentencia SQL se tiene que hacer referencia al archivo de datos o al archivo de log respectivamente.
- *ubicación_y_nombre_archivo* es una cadena que incluye la ruta y el nombre del archivo. La ruta debe especificar una carpeta existente en el servidor en el que está instalado SQL.
- *tamaño* especifica el tamaño del archivo.
- *tamaño_máximo* es el máximo tamaño que puede alcanzar el archivo si se requiriera de espacio adicional.
- *incremento_crecimiento* es la cantidad de espacio que se añade al archivo cada vez que se necesita espacio adicional. Se puede especificar como una magnitud constante en KB ó MB, ó como una tasa de crecimiento (%)

Observación

Todas las bases de datos tienen al menos un grupo de archivos principal. Todas las tablas del sistema se encuentran en el grupo de archivos principal. Una base de datos puede tener también grupos de archivos definidos por el usuario. Si se crea un objeto con una clausula ON grupo Archivos que especifica un grupo de archivos definidos por el usuario, todas las páginas del objeto se asignan desde el grupo de archivos especificado. Las páginas de todos los objetos de usuario creados sin una clausula ON grupo Archivos, o con una clausula O DEFAULT, se asignan desde el grupo de archivos predeterminado. Cuando se crea una

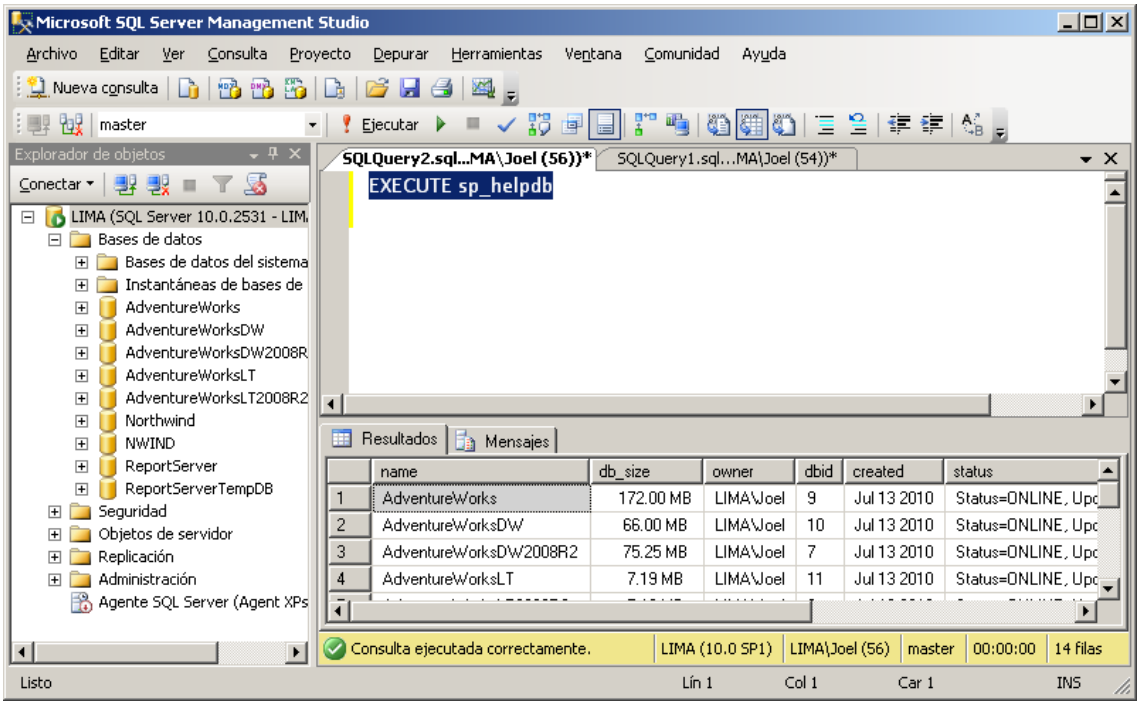
Base de datos por primera vez, el grupo de archivos principal es el grupo de archivos predeterminado.

Procedimientos almacenados del sistema

SQL Server presenta un conjunto de Procedimientos Almacenados que permiten mostrar información de las bases de datos creadas en el servidor.

El procedimiento almacenado sp_helpdb

Para mostrar un informe de una base de dato o de todas las bases de datos de un servidor con SQL Server, ejecute sp_helpbd y se presentará una pantalla como la siguiente.



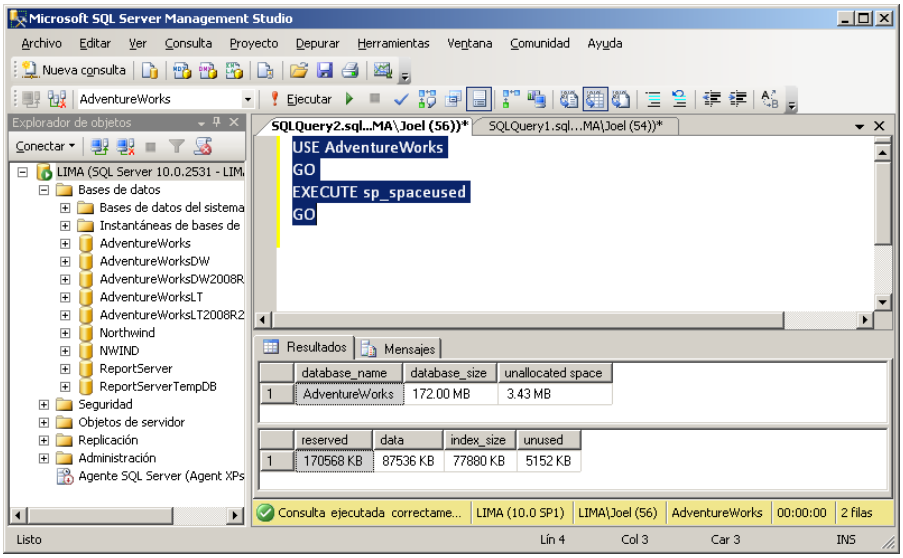
El procedimiento almacenado sp_spaceused

Para obtener un informe acerca del espacio utilizando en una base de datos, emplee el procedimiento almacenado del sistema sp_spaceused.

Escriba el siguiente Script y luego ejecútelo.

Ejercicio 12

```
USE AdventureWorks
GO
EXECUTE sp_spaceused
GO
```



El procedimiento almacenado sp_helpfile

Utilice el procedimiento almacenado del sistema sp_helplife para obtener el informe de los archivos de la base de datos.

Ejercicio 13

```
USE AdventureWorks
GO

EXECUTE sp_helpfile
GO
```

Antes de crear una base de datos de usuario en el servidor SQL Server tenga en cuenta lo siguiente:

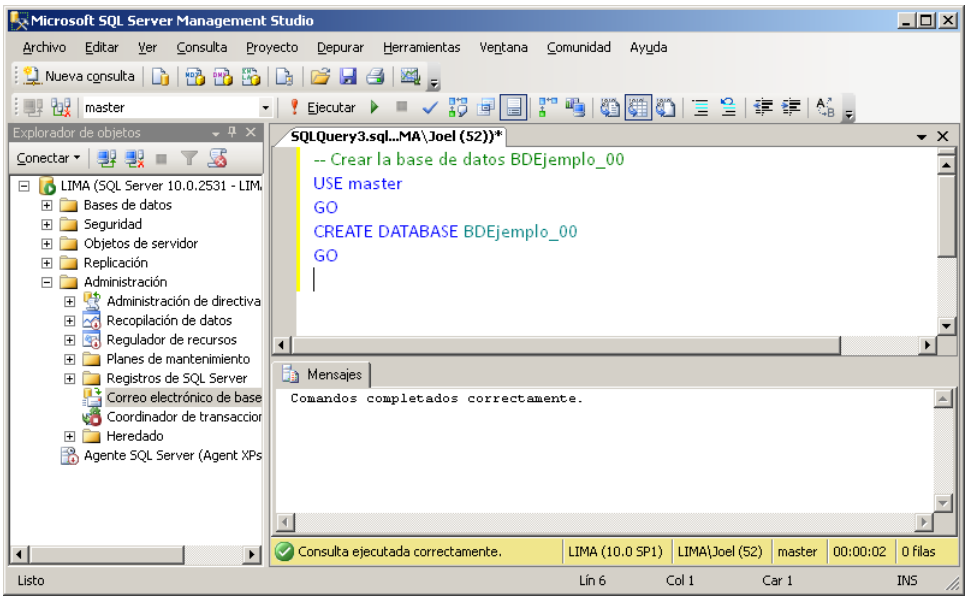
- Sólo pueden crear bases de datos las cuentas de inicio de sesión que pertenecen a los roles fijos de servidor sysadmin y dbcreator, aunque el permiso se puede conceder a otras cuentas.

- La cuenta de inicio de sesión que crea la base de datos se convierte en el usuario dueño (dbo) de la base de datos.
- Defina el nombre y tamaño de la base de datos, los archivos (y sus propiedades) en los que residirá la base de datos.

Ejemplos de creación de base de datos utilizando T_SQL

Ejercicio 14: Escriba el siguiente Script que permite crear la base de datos BDEjemplo_00

```
-- Crear la base de datos BDEjemplo_00
USE master
GO
CREATE DATABASE BDEjemplo_00
GO
```



Ejercicio 15: Crear la base de datos BdEjemplo_01 que especifique los archivos de registro de datos y de transacciones (código T-SQL)

Nota: Para la ejecución del ejercicio se debe crear una carpeta DATA en el disco C

C:\DATA; en esta carpeta se crearán los archivos de datos y de registro de transacciones.

```
USE master
GO
CREATE DATABASE BdEjemplo_01

ON
```

```
(NAME = BdEjemplo_01_dat,
FILENAME = 'C:\DATA\Bd_01dat.mdf',
SIZE = 5,
MAXSIZE = 20,
FILEGROWTH = 5 )

LOG ON
(NAME = BdEjemplo_01_log,
FILENAME = 'C:\DATA\Bd_01log.ldf',
SIZE = 2MB,
MAXSIZE = 15MB,

FILEGROWTH = 2 MB)
GO
```

El ejemplo anterior crea una base de datos llamada BdEjemplo_01. Debido a que no se utiliza la palabra clave PRIMARY, el primer archivo (BdEjemplo_01_dat) se convierte, de forma predeterminada, en el archivo principal. Como no se especifican MB ni KB en el parámetro SIZE del archivo BdEjemplo_01_dat, de forma predeterminada, dicho parámetro indica MB y el tamaño se asigna en megabytes. El tamaño del archivo BdEjemplo_01_log se asigna en megabytes porque se ha indicado explícitamente el sufijo MB en el parámetro SIZE.

Ejercicio 16: Crear la base de datos BdEjemplo_02 mediante la especificación de múltiples archivos de registro de datos y de transacciones.

```
USE master
GO
CREATE DATABASE BdEjemplo_02
ON
PRIMARY ( NAME = Logi1,
        FILENAME = 'C:\DATA\logidat1.mdf',
        SIZE = 5MB,
        MAXSIZE = 10,

        FILEGROWTH = 1),
( NAME = Arch2,
        FILENAME = 'C:\DATA\logidat2.ndf',
        SIZE = 5MB,
        MAXSIZE = 10,
        FILEGROWTH = 1),
( NAME = Arch3,
 FILENAME ='C:\DATA\logidat3.ndf',
```

```
SIZE = 5MB,
MAXSIZE = 10,
        FILEGROWTH = 1)

LOG ON
( NAME = Logilog1,
        FILENAME = 'C:\DATA\logilog1.ldf',
        SIZE = 5MB,
        MAXSIZE = 10,
        FILEGROWTH = 1),
( NAME = Archlog2,
        FILENAME = 'C:\DATA\logilog2.ldf',
SIZE = 5MB,
        MAXSIZE = 10,
        FILEGROWTH = 1)
GO
```

Creación de grupos de archivos

Cada base de datos tiene un grupo de archivos principal. Este grupo de archivos contiene el archivo de datos principal y cualquier otro archivo secundario que no se encuentre en otro grupo de archivos. Se pueden crear grupos de archivos definidos por el usuario para agrupar archivos con fines administrativos y de asignación y ubicación de datos.

Por ejemplo, pueden crearse tres archivos, Datos1.ndf, Datos2.ndf y Datos3.ndf, en tres unidades de disco respectivamente para asignarlos posteriormente al grupo de archivos grArchivos1. A continuación, se puede crear una tabla específicamente para el grupo de archivos grArchivos1. Las consultas de datos de la tabla se distribuirán por los tres discos, con lo que mejorará el rendimiento. Puede obtenerse la misma mejora del rendimiento con un solo archivo creado en un conjunto de bandas RAID (matriz redundante de discos independientes). No obstante, los archivos y grupos de archivos permiten agregar fácilmente nuevos archivos a discos nuevos.

Todos los archivos de datos se almacenan en los grupos de archivos que se indican en la tabla siguiente.

Grupo de archivos	Descripción
Principal	Grupo de archivos que contiene el archivo principal. Todas las tablas del sistema se asignan al grupo de archivos principal.
Definido por el usuario	Cualquier grupo de archivos creado específicamente por el usuario al crear la base de datos o al modificarla.

Grupo de archivos predeterminado

Cuando se crean objetos en la base de datos sin especificar a qué grupo de archivos pertenecen, se asignan al grupo de archivos predeterminado. Siempre existe un grupo de archivos designado como predeterminado. Los archivos del grupo de archivos predeterminado deben ser lo suficientemente grandes como para dar cabida a todos los objetos nuevos no asignados a otros grupos de archivos.

Crear grupos de archivos

Se pueden crear grupos de archivos cuando se crea la base de datos, o bien posteriormente cuando se agregan archivos a la misma. Sin embargo, después de agregar archivos a la base de datos, no es posible moverlos a otro grupo de archivos.

En cada base de datos pueden crearse hasta 32.767 grupos de archivos. Los grupos de archivos sólo pueden contener archivos de datos. Los archivos del registro de transacciones no pueden formar parte de un grupo de archivos.
Para agregar un grupo de archivos al crear una base de datos utilizar:

- CREATE DATABASE (Transact-SQL)

Para agregar un grupo de archivos a una base de datos utilizar:

- ALTER DATABASE (Transact-SQL)

Reglas para el diseño de archivos y grupos de archivos

Las siguientes reglas afectan a los archivos y grupos de archivos:

- Un archivo o un grupo de archivos no puede ser utilizado por más de una base de datos.
- Un archivo puede pertenecer únicamente a un grupo de archivos.
- Los archivos del registro de transacciones nunca pueden formar parte de un grupo de archivos.

Ejercicio 17

En el ejemplo siguiente se crea la base de datos **BDEjemplo03**, que tiene los siguientes grupos de archivos:

El grupo de archivos principal, con los archivos **Apri1_dat** y **Apri2_dat.**

Un grupo de archivos denominado **VentasGrupo1**, que contiene el archivo **AGrp1Fi1_dat**

```
USE master
GO
CREATE DATABASE BDEjemplo03
ON PRIMARY
( NAME = APri1_dat,
   FILENAME = 'C:\DATA\APri1dat.mdf',

  SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 15% ),
( NAME = APri2_dat,
   FILENAME = 'C:\DATA\APri2dat.ndf',
   SIZE = 10,
   MAXSIZE = 50,

 FILEGROWTH = 15% ),
FILEGROUP VentasGrupo1
```

```
( NAME = AGrp1Fi1_dat,
   FILENAME = 'C:\DATA\AGrp1Fi1dat.ndf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = Arch_log,
   FILENAME = 'C:\DATA\Archlog.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

Cambiar el tamaño de la base de datos

Si desea cambiar el tamaño de la base de datos puede usar la instrucción ALTER DATABASE o el Administrador corporativo de SQL Server. Para reducir una base de datos debe usar los comandos DBCC SHRINKDATABASE o DBCC SHRINKFILE

La instrucción ALTER DATABASE

La instrucción ALTER DATABASE permite realizar cambios a una base de datos. Permite agregar o quitar archivos y grupos de archivos de una base de datos. También se puede usar para modificar los atributos de los archivos o grupos de archivos, por ejemplo para cambiar el nombre o el tamaño de un archivo.

Ejercicio 18: El siguiente ejemplo modifica la base de datos BdEjemplo_01 para agregarle un archivo de datos de 5 MB.

```
ALTER DATABASE BdEjemplo_01
ADD FILE
(
 NAME = Test1dat2,
 FILENAME = 'C:\DATA\t1dat2.ndf',
 SIZE = 5MB,
 MAXSIZE = 10MB,
```

```
 FILEGROWTH = 5MB
)
GO
```

Ejemplo: Agregar un grupo de archivos con nombre Bd_Ejemplo04FG1 a la base de datos BdEjemplo_04. A continuación agregue dos archivos de 5 MB al grupo de archivos; finalmente al grupo bd_Ejemplo04FG1 sea el grupo de archivos predeterminado.

Ejercicio 19

```
USE master
GO
ALTER DATABASE BdEjemplo_01
ADD FILEGROUP Bd_Ejemplo04FG1
GO


ALTER DATABASE BdEjemplo_01
ADD FILE
( NAME = test1dat3,
  FILENAME = 'C:\DATA\t1dat3.ndf',
  SIZE = 5MB,
  MAXSIZE = 10MB,
  FILEGROWTH = 5MB),
( NAME = test1dat4,
  FILENAME = 'C:\DATA\t1dat4.ndf',
  SIZE = 5MB,
  MAXSIZE = 10MB,
  FILEGROWTH = 5MB)
TO FILEGROUP Bd_Ejemplo04FG1
GO
ALTER DATABASE BdEjemplo_01
```

```
MODIFY FILEGROUP Bd_Ejemplo04FG1 DEFAULT
GO
```

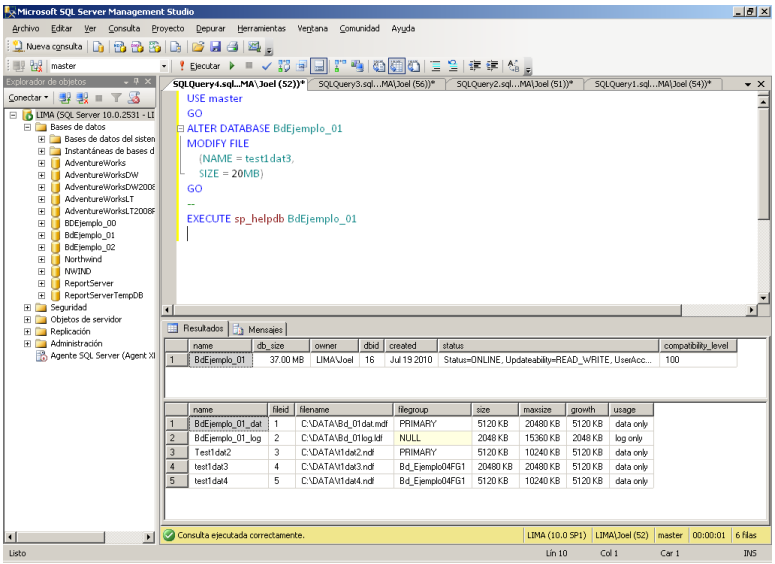
Expansión de la base de datos

Puede expandir la base de datos añadiéndole archivos adicionales, los archivos de datos crecerán automáticamente hasta que se acabe el espacio en el disco. Recuerde que los datos que están guardados a través de varios archivos en una base de datos dividirán automáticamente la información de esos archivos.

Ejercicio 20: El ejemplo siguiente aumenta el tamaño al archivo test1dat3 agregado a la base de datos BdEjemplo_01.Observe que se utiliza el nombre lógico del archivo.

```
USE master
GO
ALTER DATABASE BdEjemplo_01
MODIFY FILE
   (NAME = test1dat3,
   SIZE = 20MB)
GO
```

Ahora puede ejecutar el procedimiento almacenado de sistema sp_heldb para verificar que se haya agrandado satisfactoriamente la base de datos:



Reducción de la base de datos (T-SQL)

DBCC SHRINKDATABASE.

Puede reducir una base de datos completa usando el comando DBCC SHRINKDATABASE, esta es la sintaxis.

```
DBCC SHRINKDATABASE
( database_name | database_id | 0
    [ , target_percent ]
    [ , { NOTRUNCATE | TRUNCATEONLY } ]
)
[ WITH NO_INFOMSGS ]
```

DBCC SHRINKFILE

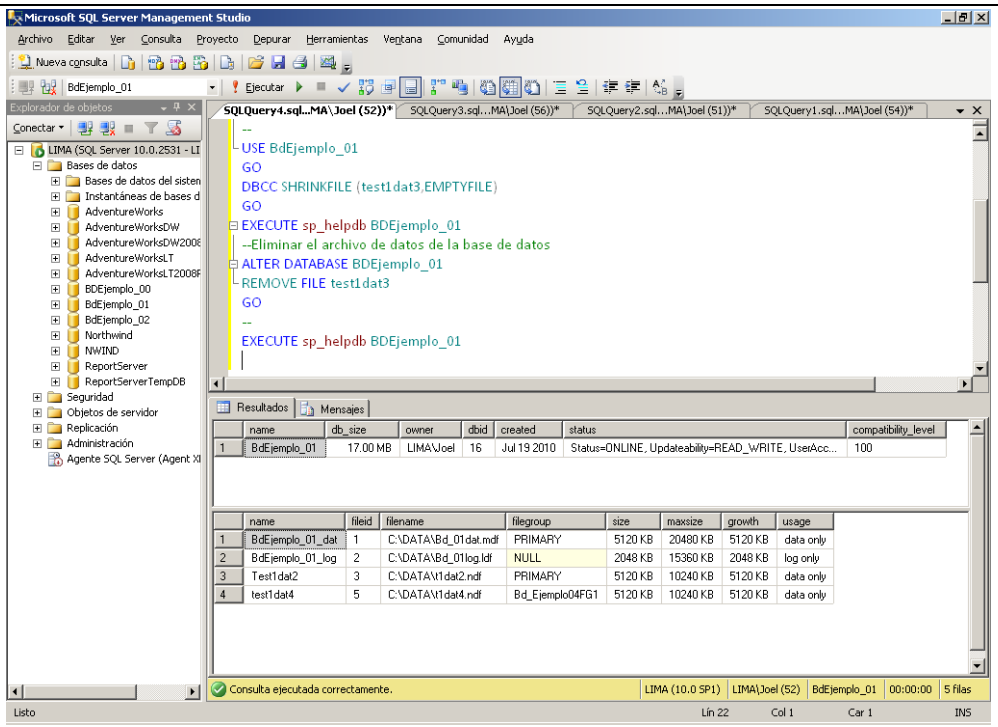
Reduce el tamaño del archivo de datos o de registro para la base de datos actual, o vacía un archivo moviendo los datos del archivo especificado a otros archivos del mismo grupo de archivos, permitiendo quitar el archivo de la base de datos. Puede reducir un archivo a un tamaño menor que el tamaño especificado cuando se creó. Así se restablece el tamaño mínimo de archivo al valor nuevo.

Ejercicio 21: El ejemplo siguiente permite vaciar el archivo test1dat3 de la base de datos BdEjemplo_01 y usa la opción REMOVE FILE para eliminar el archivo de la base de datos.

```
USE BdEjemplo_01
GO
DBCC SHRINKFILE (test1dat3,EMPTYFILE)
GO
EXECUTE sp_helpdb BDEjemplo_01

--Eliminar el archivo de datos de la base de datos
ALTER DATABASE BDEjemplo_01
REMOVE FILE test1dat3
GO
```

Para verificar que se haya eliminado el archivo utilice el procedimiento almacenado de sistema sp_helpdb no olvide ejecutarlo en resultado en cuadrícula para observar al detalle el cambio.

```
EXECUTE sp_helpdb BDEjemplo_01
```



Cambiar el nombre de la base de datos

Para cambiar el nombre de una base de datos se utiliza el comando ALTER DATABASE

Ejercicio 22

```
ALTER DATABASE BDEjemplo_00
 MODIFY NAME =BDEjemplo_11
```

Eliminar una base de datos

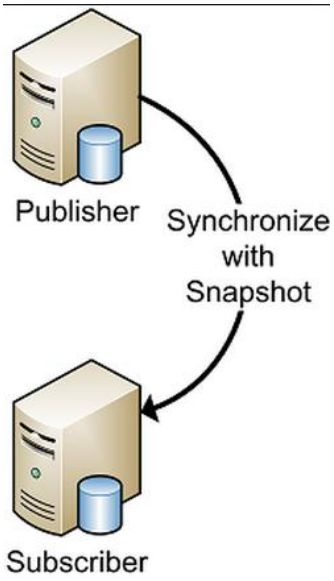
La instrucción DROP DATABASE

Quita una o varias bases de datos o instantáneas de la base de datos de una instancia de SQL Server

Ejercicio 23: En este ejemplo de eliminar la base de datos BDEjemplo_11.

```
DROP DATABASE   BDEjemplo_11
```

Creación de instantáneas (snapshot) de base de datos



Una instantánea de base de datos es una vista estática de sólo lectura de una base de datos denominada base de datos de origen. Una instantánea de base de datos es coherente en cuanto a las transacciones con la base de datos tal como existía en el momento de la creación de la instantánea. Al crear una instantánea de base de datos, normalmente la base de datos de origen tendrá transacciones abiertas. Antes de que la instantánea esté disponible, se revertirán las transacciones abiertas para que la instantánea de base de datos sea coherente en cuanto a las transacciones.

Usos

Los clientes pueden consultar una instantánea de base de datos, lo que resulta útil para escribir informes basados en los datos en el momento de la creación de la instantánea. Asimismo, si más adelante se daña la base de datos de origen, podrá devolverla al estado en el que se encontraba en el momento de la creación de la instantánea.

Sintaxis

```
CREATE DATABASE database_snapshot_name
   ON
      (
      NAME = logical_file_name,
      FILENAME = 'os_file_name'
      ) [ ,...n ]
   AS SNAPSHOT OF source_database_name
[;]
```

Ejercicio 24

A continuación se crea una instantánea de la base de datos Northwind

```
--01 Activar la base de datos del sistema master
USE master
GO
```

```
--02 Crear una instantánea de la base de datos Northwind
CREATE DATABASE Northwind_SNAPSHOT
 ON
 (
        NAME= Northwind,
        FILENAME='C:\DATA\Northwind_SNAPSHOT.MDF'
 )
AS SNAPSHOT OF  Northwind

--03 Activar la base de datos Northwind
USE Northwind
GO

--04 Consultar la tabla Products
SELECT * FROM Products
WHERE ProductID=1 --El precio unitario es 18.00

--05 Actualizar el precio del producto con ProductID a 28.00
UPDATE Products
SET UnitPrice=28.00
WHERE ProductID=1

--06 Verificar la modificación
SELECT * FROM Products
WHERE ProductID=1 --El precio unitario es 28.00

--07 Activar la instantanea Northwind_SNAPSHOT
USE Northwind_SNAPSHOT

--08 Consultar la tabla Products
SELECT * FROM Products
WHERE ProductID=1 --El precio unitario es 18.00
```

Creación de Esquemas

Un esquema de base de datos es un espacio de nombres separado de un usuario de base de datos. Un esquema se puede considerar como un contenedor de objetos. Los esquemas se pueden crear y modificar en una base de datos, y a los usuarios se les puede conceder

Acceso a un esquema. Un esquema puede ser propiedad de cualquier usuario y esta propiedad es transferible.

Sintaxis

```
CREATE SCHEMA schema_name_clause [ <schema_element> [ ...n ] ]
<schema_name_clause> ::=
   {
      schema_name
   | AUTHORIZATION owner_name
   | schema_name AUTHORIZATION owner_name
   }

<schema_element> ::=
   {
      table_definition | view_definition | grant_statement |
      revoke_statement | deny_statement
   }
```

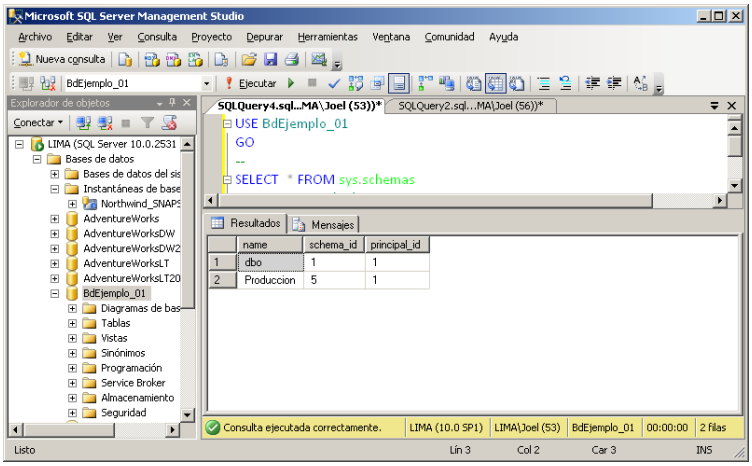
Ejercicio 25: Crear el esquema Producción en la base de datos BDEjemplo_01

```
USE BdEjemplo_01
GO
--  Crear  el esquema Producción
CREATE SCHEMA Producción
```

La vista sys.schemas del catálogo del sistema

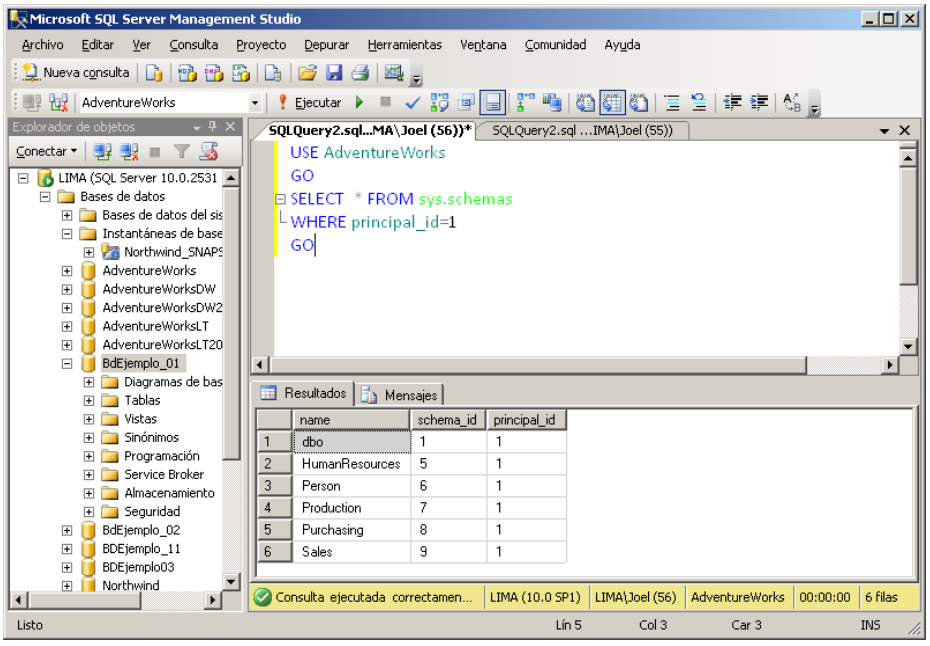
Contiene una fila por cada esquema de base de datos.

```
--  consultar la vista  sys.schemas del catálogo del sistema
SELECT  * FROM sys.schemas
WHERE principal_id=1
GO
```



La base de datos AdventureWorks presenta los siguientes esquemas:

- dbo
- HumanResources
- Person
- Production
- Purchasing
- Sales



3

Creación de Tablas

Las tablas son objeto de la base de datos que contiene todos sus datos. En las tablas, los datos se organizan con arreglo a un formato de filas y columnas, similar al de una hoja de cálculo. Cada fila representa a un registro único, y cada columna representa a un campo dentro de un registro. Por ejemplo, en una tabla que contenga los datos de los empleados de una compañía puede haber una fila para cada empleado y distintas columnas en las que figuren detalles de los empleados tales como el número de empleado, el nombre, la dirección, el puesto que ocupa y su número de teléfono particular.

Las tablas de SQL Server tienen dos componentes principales:

Columnas: Cada columna representa algún atributo del objeto representado por la tabla, por ejemplo, una tabla de Artículos tendría columnas para el IdArticulo descripción y precio.

Filas: Cada fila representa una ocurrencia individual del objeto representado por la tabla. Por ejemplo, la tabla de Artículos tendría una fila por cada una de los artículos comercializados por la empresa.

Directrices para crear tablas

Para crear una tabla en SQL-Server previamente se debe planificar lo siguiente:

- Los tipos de datos que debe contener la tabla.
- Los tipos de datos definidos por el usuario, y crearlos para poder utilizarlos.
- Las columnas de la tabla y los tipos de datos para cada columna (así como su longitud, si es preciso).
- Qué columnas aceptan valores NUL.
- Si deben utilizarse (y cuando) restricciones o valores predeterminados y reglas.
- Los tipos de índices necesarios, donde se necesitan y qué columnas son claves principales y claves externas.

También es posible crear una tabla básica y luego modificarla según las necesidades que se pueden establecer posteriormente.
Como cada columna representa un atributo de un objeto, los datos de cada ocurrencia de la columna son similares. Una de las propiedades de las columnas es su tipo de datos, que define el tipo de datos que la columna puede almacenar. A continuación se definen los diferentes tipos de datos que se pueden utilizar en SQL Server.

Tipos de datos

SQL Server tiene varios tipos de datos básicos, seleccionar el tipo de dato adecuado consiste en asociar el dominio de valores que necesite almacenar con el tipo de dato correspondiente. Al elegir tipos de datos para asignarlos a las diferentes columnas deberá evitar desperdiciar espacio de almacenamiento y a la vez proporcionar espacio suficiente para el rango de valores posibles en las columnas. Los tipos de datos disponibles en SQL Server 2014 se muestra a continuación:

A continuación se presenta un listado con los tipos de datos que se utilizan en SQL Server 2014

Numéricos exactos

bigint	numeric
bit	smallint
decimal	smallmoney
int	tinyint
money	

Numéricos aproximados

float	real

Fecha y hora

date	datetimeoffset
datetime2	smalldatetime
datetime	time

Cadenas de caracteres

char	varchar
text	

Cadenas de caracteres Unicode

nchar	nvarchar
ntext	

Cadenas binarias

binary	varbinary
image	

Otros tipos de datos

cursor	timestamp
hierarchyid	uniqueidentifier
sql_variant	xml
table	

Descripción de los principales tipos de datos

Tipos de datos Numéricos exactos

Los enteros son números completos y no contienen decimales ni fracciones.

Microsoft SQL Server tiene los siguientes tamaños de tipos de datos enteros:

- **bigint**

  Tiene una longitud de 8 bytes y almacena números de –2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807).

- **integer** o **int**

  Tiene una longitud de 4 bytes y almacena números entre -2.147.483.648 y 2.147.483.647.

- **smallint**

  Tiene una longitud de 2 bytes y almacena números entre -32.768 y 32.767.

- **tinyint**

  Tiene una longitud de 1 byte y almacena números entre 0 y 255.

Resumen

Tipo de datos	Intervalo	Almacenamiento
bigint	De -2^63 (-9.223.372.036.854.775.808) a 2^63-1 (9.223.372.036.854.775.807)	8 bytes
int	De -2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647)	4 bytes
smallint	De -2^15 (-32.768) a 2^15-1 (32.767)	2 bytes
tinyint	De 0 a 255	1 byte

Decimal y numeric

Son tipos de datos numéricos que tienen precisión y escala fijas.

decimal[(p[,s])] y numeric[(p[,s])]

Números de precisión y escala fijas. Cuando se utiliza la precisión máxima, los valores válidos se sitúan entre - 10^38 +1 y 10^38 - 1. Los sinónimos de ISO para decimal son de tipo dec y dec(p, s). numeric es funcionalmente equivalente a decimal.

p (precisión)

El número total máximo de dígitos decimales que se puede almacenar, tanto a la izquierda como a la derecha del separador decimal. La precisión debe ser un valor comprendido entre 1 y la precisión máxima de 38. La precisión predeterminada es 18.

S (escala)

El número máximo de dígitos decimales que se puede almacenar a la derecha del separador decimal. La escala debe ser un valor comprendido entre 0 y p. Sólo es posible especificar la escala si se ha especificado la precisión. La escala predeterminada es 0; por lo tanto, 0 <= s <= p. Los tamaños de almacenamiento máximo varían, según la precisión

Precisión	Bytes de almacenamiento
1 – 9	5
10-19	9
20-28	13
29-38	17

Números aproximados

float y real

Los tipos de datos **float** y **real** se conocen como tipos de datos aproximados. El comportamiento de **float** y **real** sigue la especificación IEEE 754 acerca de los tipos de datos numéricos aproximados.

Los tipos de datos numéricos aproximados no almacenan los valores exactos especificados para muchos números; almacenan una aproximación muy precisa del valor. Para muchas aplicaciones, la pequeña diferencia entre el valor especificado y la aproximación almacenada no es apreciable. Sin embargo, a veces la diferencia se hace notar. Debido a la naturaleza aproximada de los tipos de datos **float** y **real**, no los use cuando necesite un comportamiento numérico exacto, como, por ejemplo, en aplicaciones financieras, en operaciones que conlleven un redondeo o en comprobaciones de igualdad. En su lugar, use los tipos de datos enteros, **decimal**, **money** o **smallmoney**.

Resumen

Tipo de datos	Intervalo	Almacenamiento
Float	De - 1,79E+308 a -2,23E-308, 0 y de 2,23E-308 a 1,79E+308	Depende del valor de n.
Real	De - 3,40E + 38 a -1,18E - 38, 0 y de 1,18E - 38 a 3,40E + 38	4 Bytes

Tipo moneda

Tipos de datos que representan valores monetarios o de moneda.

Resumen

Tipo de datos	Intervalo	Almacenamiento
Money	De -922,337,203,685.477,5808 a 922,337,203,685.477,5807	8 bytes
smallmoney	De - 214.748,3648 a 214.748,3647	4 bytes

Tipo fecha hora

Date

Define una fecha.

Datetime

Define una fecha que se combina con una hora del día con fracciones de segundos basada en un reloj de 24 horas.

Smalldatetime

Define una fecha que se combina con una hora del día. La hora está en un formato de día de 24 horas, con segundos siempre a cero (: 00) y sin fracciones de segundo.

Datetime2

Define una fecha que se combina con una hora de día basada en el reloj de 24 horas. datetime2 puede considerarse una extensión del tipo datetime existente que tiene un intervalo de fechas mayor, un valor predeterminado mayor de precisión fraccionaria y precisión especificada por el usuario opcional.

Time

Define una hora de un día. La hora no distingue la zona horaria y está basada en un reloj de 24 horas.

Datetimeoffset

Define una fecha que se combina con una hora del día con reconocimiento de zona horaria y basado en un reloj de 24 horas

Resumen

Tipo de datos	Salida
time	12:35:29. 1234567
date	2007-05-08
smalldatetime	2007-05-08 12:35:00
datetime	2007-05-08 12:35:29.123
datetime2	2007-05-08 12:35:29. 1234567
datetimeoffset	2007-05-08 12:35:29.1234567 +12:15

Cadenas de caracteres

char y varchar

Son tipos de datos de caracteres que tienen longitud fija o variable.

char [(n)]

Datos de caracteres no Unicode de longitud fija, con una longitud de n bytes. n debe ser un valor entre 1 y 8.000. El tamaño de almacenamiento es n bytes. El sinónimo ISO de char es el tipo character.

varchar [(n | max)]

Datos de caracteres no Unicode de longitud variable. n puede ser un valor entre 1 y 8.000. max indica que el tamaño de almacenamiento máximo es de 2^31-1 bytes. El tamaño de almacenamiento es la longitud real de los datos especificados + 2 bytes. Los datos especificados pueden tener una longitud de 0 caracteres. Los sinónimos ISO de varchar son char varying o character varying.

Cadenas de caracteres Unicode

nchar y nvarchar

Tipos de datos de caracteres, para datos Unicode de longitud fija, nchar, o variable, nvarchar, y que utilizan el juego de caracteres UNICODE UCS-2.

nchar [(n)]

Datos de carácter Unicode de longitud fija, con n caracteres. n debe estar comprendido entre 1 y 4.000. El tamaño de almacenamiento es dos veces n bytes. Los sinónimos para nchar en ISO son national char y national character.

nvarchar [(n | max)]

Datos de carácter Unicode de longitud variable. N puede ser un valor comprendido entre 1 y 4.000. max indica que el tamaño máximo de almacenamiento es 2^31-1 bytes. El tamaño de almacenamiento en bytes es dos veces el número de caracteres especificado + 2 bytes. Los datos especificados pueden tener una longitud de 0 caracteres. Los sinónimos para nvarchar en ISO son national char varying y national character varying.

binary [(n)]

Datos binarios de longitud fija con una longitud de n bytes, donde n es un valor que oscila entre 1 y 8.000. El tamaño de almacenamiento es de n bytes.

varbinary [(n | max)]

Datos binarios de longitud variable. n puede ser un valor que oscila entre 1 y 8.000. max indica que el tamaño máximo de almacenamiento es de 2^31-1 bytes. El tamaño de almacenamiento es la longitud real de los datos especificados + 2 bytes. Los datos especificados pueden tener una longitud de 0 bytes. El sinónimo de ANSI SQL para varbinary es binary varying.

Cursor

Un tipo de datos para las variables o para los parámetros de resultado de los procedimientos almacenados que contiene una referencia a un cursor. Las variables creadas con el tipo de datos cursor aceptan NULL.

Las operaciones a las que pueden hacer referencia las variables y parámetros que tienen un tipo de datos cursor son:

- Las instrucciones DECLARE @local_variable y SET @local_variable.
- Las instrucciones del cursor OPEN, FETCH, CLOSE y DEALLOCATE.
- Los parámetros de resultado de procedimientos almacenados.
- La función CURSOR_STATUS.
- Los procedimientos almacenados del sistema sp_cursor_list, sp_describe_cursor, sp_describe_cursor_tables y sp_describe_cursor_columns.

hierarchyid

El tipo de datos del sistema de hierarchyid es de longitud variable. Use hierarchyid para representar la posición en una jerarquía. Una columna de tipo hierarchyid no representa automáticamente un árbol. Dependerá de la aplicación generar y asignar los valores hierarchyid de tal forma que la relación deseada entre las filas se refleje en los valores.

Un valor del tipo de datos hierarchyid representa una posición en una jerarquía de árbol.

uniqueidentifier

Es un GUID de 16 bytes

Una columna o una variable local de tipo de datos uniqueidentifier se puede inicializar en un valor de las siguientes formas:

- Mediante la función NEWID.
- Mediante la conversión a partir de una constante de cadena con el formato xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, donde cada x es un dígito hexadecimal en el intervalo 0-9 o a-f. Por ejemplo, 6F9619FF-8B86-D011-B42D-00C04FC964FF es un valor uniqueidentifier válido.

xml

Es el tipo de datos que almacena datos de XML. Puede almacenar instancias de xml en una columna o una variable de tipo xml.

Creación de tipos de datos

CREATE TYPE

Crea un tipo de datos de alias o un tipo definido por el usuario en la base de datos actual. La implementación de un tipo de datos de alias se basa en un tipo nativo del sistema de SQL Server. Un tipo definido por el usuario se implementa a través de una clase de un

Ensamblado de Common Language Runtime (CLR) de Microsoft .NET Framework. Para enlazar un tipo definido por el usuario a su implementación, el ensamblado CLR que contiene la implementación del tipo debe registrarse primero en SQL Server mediante CREATE ASSEMBLY.

Sintaxis

```
CREATE TYPE [ schema_name. ] type_name
{
    FROM base_type
    [ ( precision [ , scale ] ) ]
    [ NULL | NOT NULL ]
  | EXTERNAL NAME assembly_name [ .class_name ]
  | AS TABLE ( { <column_definition> | <computed_column_definition> }
        [ <table_constraint> ] [ ,...n ] )
} [ ; ]
```

Argumentos:

schema_name

Es el nombre del esquema al que pertenece el tipo de datos de alias o el tipo definido por el usuario.

type_name

Es el nombre del tipo de datos de alias o del tipo definido por el usuario. Los nombres de tipos deben cumplir las reglas de los identificadores.

base_type

Es el tipo de datos suministrado por SQL Server en el que se basa el tipo de datos de alias

Ejemplo:

Ejercicio 26: **Crear le tipo de dato DNI basado en CHAR(8)**

```
USE BdEjemplo_01
GO
CREATE TYPE DNI
FROM CHAR(8) NOT NULL ;
```

Ejercicio 27: **Crear le tipo de dato Iniciales basado en CHAR(3)**

```
USE BdEjemplo_01
GO
CREATE TYPE Iniciales
FROM CHAR(3) NOT NULL ;
```

Ejercicio 28: **Crear le tipo de dato Direccion basado en VARCHAR(30)**

```
CREATE TYPE Direccion
FROM VARCHAR(30) NOT NULL ;
GO
```

Ejercicio 29: **Crear le tipo de dato Telefono basado en CHAR(7) y que acepte NULL**

```
CREATE TYPE Telefono
FROM CHAR(7)  NULL ;
```

Eliminación de tipos de datos

DROP TYPE

Quita de la base de datos actual un tipo de datos de alias o un tipo definido por el usuario de Common Language Runtime (CLR).

Ejercicio 30:

```
USE BdEjemplo_01
GO
DROP  TYPE Teléfono
GO
```

Elimina el tipo de dato Teléfono de la base de datos BDEjemplo_01

La instrucción CREATE TABLE

Sintaxis

```
CREATE TABLE
   [ database_name . [ schema_name ] . | schema_name . ] table_name
      ( { <column_definition> | <computed_column_definition>
            | <column_set_definition> }
      [ <table_constraint> ] [ ,...n ] )
   [ ON { partition_scheme_name ( partition_column_name ) | filegroup
      | "default" } ]
   [ { TEXTIMAGE_ON { filegroup | "default" } ]
   [ FILESTREAM_ON { partition_scheme_name | filegroup
      | "default" } ]
   [ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]
```

Sintaxis reducida

```
CREATE TABLE nombre_tabla(
            nombre_columna1  tipo_dato1  [ NULL | NOT NULL ] ,
            nombre_columna2  tipo_dato2  [ NULL | NOT NULL ] ,
            nombre_columna3  tipo_dato3  [ NULL | NOT NULL ] ,
            ... )
```

Ejercicio 31: Creación de la tabla Producto

```
-- TABLA PRODUCTO
USE BdEjemplo_01
GO
IF EXISTS (SELECT * FROM SysObjects WHERE Type='U' and Name='Producto')
DROP TABLE Producto
GO

CREATE TABLE Producto
(
        IdProducto              CHAR(5) NOT NULL,
        Pro_Nombre              VARCHAR (40) NOT NULL,
        Pro_PrecioUnitario      DECIMAL (10,2) NULL,
```

```
        Pro_CantidadStock       INT NOT NULL,
        Pro_FechaCompra         DATETIME NOT NULL,
        Pro_Discontinuo         BIT NOT NULL
)
GO
```

Ejercicio 32: **Comprobar la existencia de la tabla Producto**

```
USE BDEjemplo_01
GO
SELECT table_name
FROM  information_schema.tables
WHERE table_name = 'Producto'
GO
```

Ejercicio 33: **Creación de la tabla Cliente utilizando los tipos de datos definidos por el usuario: dirección y teléfono**

```
-- TABLA CLIENTE
USE BdEjemplo_01
GO

IF EXISTS (SELECT * FROM SysObjects WHERE Type='U' and Name='Cliente')
DROP TABLE Cliente
GO

CREATE TABLE Cliente
(
        IdCliente           CHAR(4) NOT NULL,
        Cli_RazonSocial     VARCHAR (50) NOT NULL,
        Cli_Direccion       Direccion ,
        Cli_Telefono        Telefono
        )
GO
```

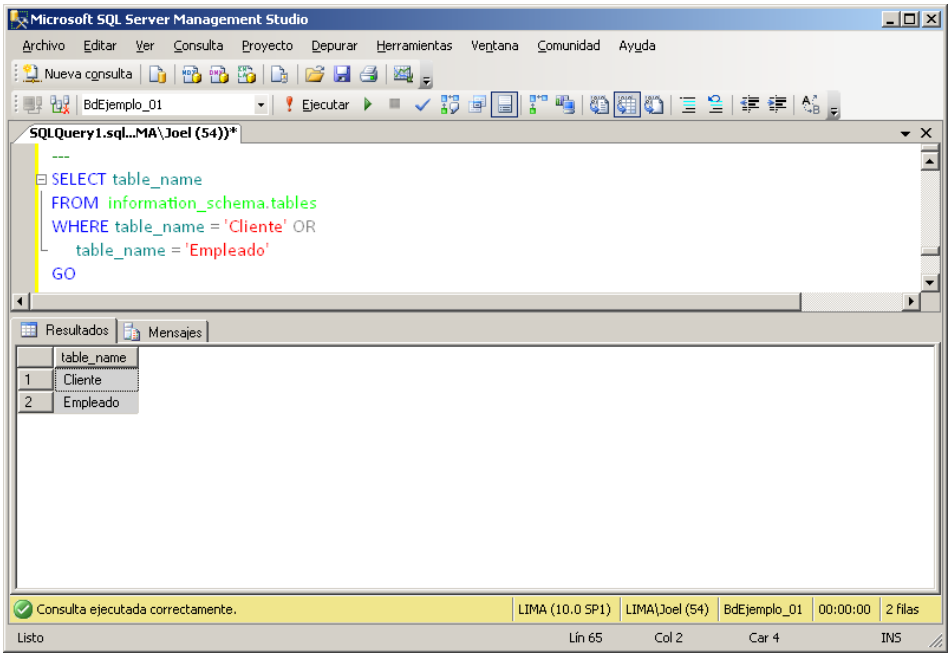
Ejercicio 34: Creación de la tabla Empleado utilizando los tipos de datos definidos por el usuario: dirección, teléfono y distrito

```
--TABLA EMPLEADO
USE BdEjemplo_01
GO

IF EXISTS (SELECT * FROM SysObjects WHERE Type='U' and Name='Empleado')
DROP TABLE Empleado
GO
```

```
CREATE TABLE Empleado
(
        IdEmpleado              CHAR(4) NOT NULL,
        Emp_Nombres             VARCHAR (20) NOT NULL,
        Emp_Apellidos           VARCHAR (30) NOT NULL,
        Emp_Direccion           Direccion,
        Emp_Telefono            Telefono

)
GO
```

Ejercicio 35: **Verificar la existencia de las tablas Cliente y Empleado**.

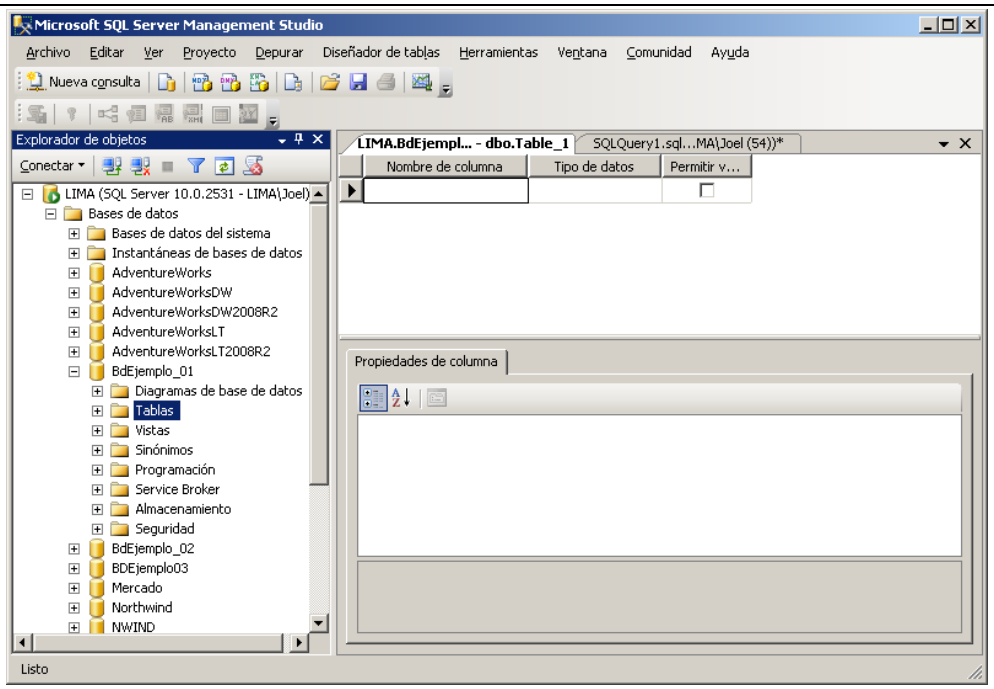
```
SELECT table_name
FROM  information_schema.tables
WHERE table_name = 'Cliente' OR
        table_name = 'Empleado'
GO
```



Creación de tablas utilizando Microsoft SQL Server Management Studio (SSMS)

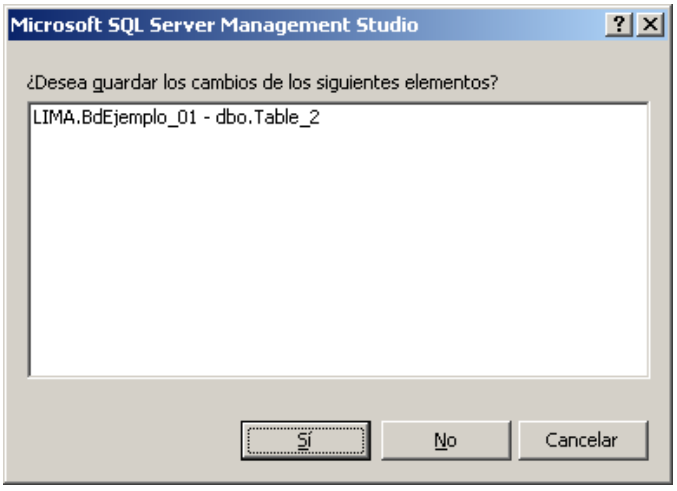
En el explorador de objetos seleccionar la base de datos BDEjemplo_01, luego seleccionar la carpeta Tablas con el botón derecho del mouse seleccionar Nueva Tabla.

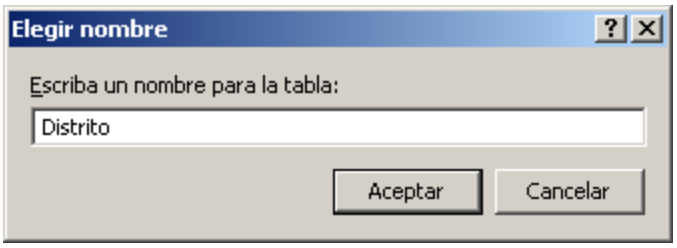
Para mostrar la ventana siguiente ventana:



Ingresar: IDDistrito CHAR(3) y Nombre VARCHAR(30)

Cerrar el editor para ingresar el nombre de la tabla: **Distrito**





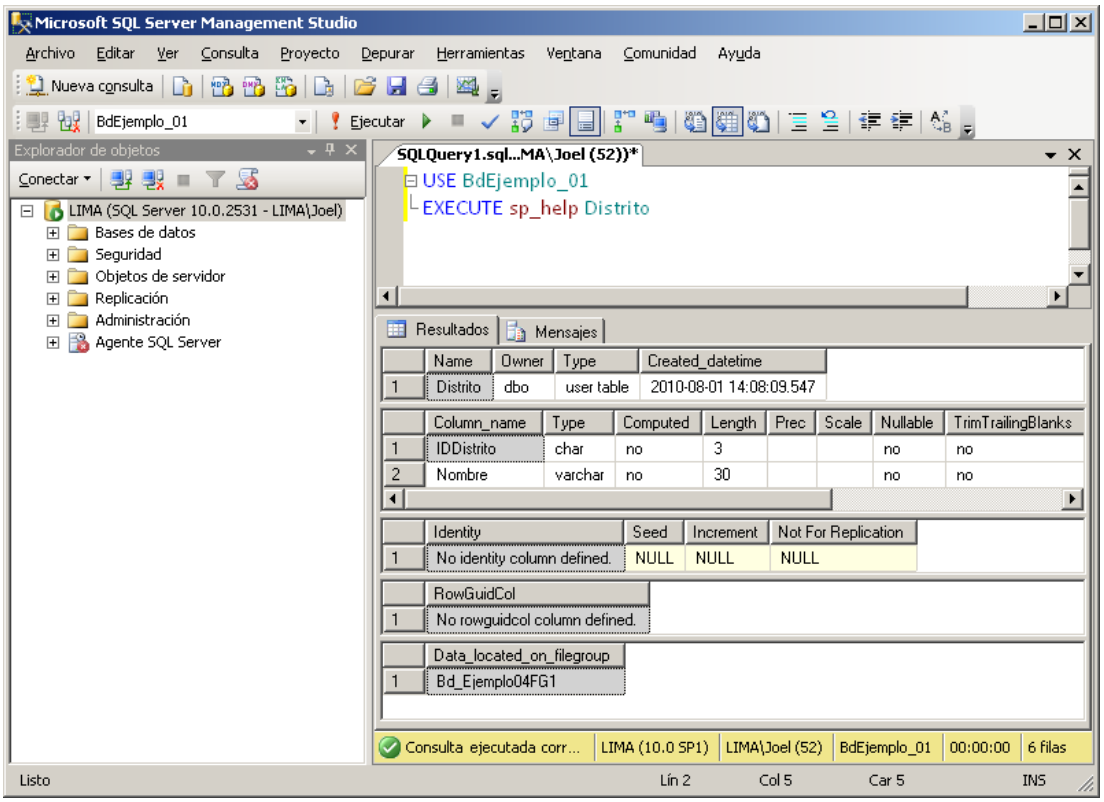
El procedimiento sp_help

Genera un reporte con información acerca de la definición de un objeto de la base de datos activa. Todos los usuarios de la base de datos pueden ejecutar este procedimiento.

Sintaxis

sp_help nombre_objeto_basedatos

Ejemplo: Verificación de la definición de la tabla Distrito



Modificación de la definición de una tabla

En una tabla podemos añadir nuevas columnas, eliminar columnas, cambiar las propiedades de una columna, añadir ó eliminar restricciones.

La instrucción ALTER TABLE

Permite modificar la definición de una tabla.

Sintaxis

```
ALTER TABLE nombre_tabla
        ADD nombre_columna propiedades_columna
        | DROP COLUMN nombre_columna
        | ALTER COLUMN nombre_columna
                                    nuevas_propiedades_columna

        | ADD CONSTRAINT nombre_restricción
                PRIMARY KEY… | UNIQUE… | FOREIGN KEY…
                | DEFAULT… | CHECK…
        | DROP CONSTRAINT nombre_restricción
```

- ADD nombre_columna permite añadir una nueva columna a la tabla.
- DROP COLUMN se usa para eliminar una columna.
- ALTER COLUMN permite modificar la definición de una columna.
- ADD CONSTRAINT permite añadir una restricción PRIMARY KEY, UNIQUE, FOREIGN KEY, DEFAULT ó CHECK a la definición de una tabla.
- DROP CONSTRAINT se usa para eliminar una restricción.

Ejercicio 36: Añadir la columna Iniciales CHAR (3) a la tabla Distrito

```
USE BdEjemplo_01
GO

ALTER TABLE Distrito
ADD Iniciales CHAR(3) NULL
```

Ejercicio 37: Añadir la columna IDDistrito a la tabla Cliente

```
USE BdEjemplo_01
GO

ALTER TABLE Cliente
ADD  IDDistrito CHAR(3) NOT  NULL
```

Ejercicio 38: Añadir la columna IDDistrito a la tabla Empleado

```
USE BdEjemplo_01
GO

ALTER TABLE Empleado
ADD  IDDistrito CHAR(3) NOT  NULL
```

Ejercicio 39: **Para el siguiente ejemplo crear una Tabla llamada Usuario.**

```
-- TABLA USUARIO
USE BdEjemplo_01
GO

IF EXISTS (SELECT * FROM SysObjects WHERE Type='U' and Name='Usuario')
DROP TABLE Usuario
GO
CREATE TABLE Usuario
(
        IdUsuario               CHAR(4) NOT NULL,
        Usu_Nombres              VARCHAR (20) NOT NULL,
        Usu_Telefono            Telefono,
        Usu_Direccion           VARCHAR (50) NOT NULL,
        Usu_FechaIngreso                DATETIME NOT NULL
)
GO
```

Ejercicio 40: **De la tabla Usuario eliminar la columna Usu_Direccion**

```
ALTER TABLE Usuario DROP COLUMN Usu_Direccion
GO
```

Ejercicio 41: Añadir la columna IDDistrito a la tabla Usuario

```
USE BdEjemplo_01
GO

ALTER TABLE Usuario
ADD  IDDistrito CHAR(3) NOT  NULL
```

Ejercicio 42: **Ejecutar el procedimiento almacenado del sistema so_help para verificar los cambios**.

```
EXEC sp_help Usuario
GO
```

4

Integridad de datos

La integridad de los datos se refiere a la consistencia y exactitud de los datos que se guardan en una base de datos.
Dos pasos importantes en el diseño de las tablas son la identificación de valores válidos para una columna y la determinación de cómo forzar la integridad de los datos en la columna.

Niveles de la integridad de datos

La integridad de datos se define en los siguientes niveles:

- Integridad de entidad
- Integridad de dominio
- Integridad referencial
- Integridad definida por el usuario

Integridad de entidad

Una tabla almacena los datos de cada una de las ocurrencias de una entidad. La entidad (o tabla) requiere que todas sus filas sean únicas. Esto se garantiza definiendo para cada fila de la entidad un identificador único (llave primaria). Por ejemplo, en la tabla Empleado, cada fila de la tabla debe representar a solo un empleado; un empleado no puede aparecer registrado en dos filas de la tabla.

La integridad de entidad se implementa utilizando:

- Restricciones PRIMARY KEY
- Restricciones UNIQUE
- La propiedad IDENTITY).

La restricción PRIMARY KEY

Las restricciones PRIMARY KEY identifican la columna o el conjunto de columnas cuyos valores identifican de forma unívoca cada una de las filas de una tabla.
Dos filas de la tabla no pueden tener el mismo valor de clave principal. No se pueden asignar valores NULL a ninguna de las columnas de una clave principal. NULL es un valor especial en las bases de datos que representa un valor desconocido, que no es lo mismo que un espacio en blanco o un 0. Como clave principal se recomienda el uso de columnas de tipo entero y de tamaño pequeño. Todas las tablas tienen que tener una clave principal.
Una tabla puede tener varias combinaciones de columnas que puedan identificar de forma unívoca las filas de la tabla; cada combinación es una clave candidata. El administrador de la base de datos elige una de las claves candidatas como clave principal.

Creación de clave primaria (PK)

Para crear la restricción de clave primaria de una tabla se utiliza el comando ALTER TABLE o al momento de crear la tabla.

Sintaxis

```
ALTER TABLE nombre_tabla

    ADD CONSTRAINT nombre_constraint

    PRIMARY KEY( columnaX, columnaP, … )
```

- PK_nombre_tabla es el nombre de la restricción clave primaria. Se recomienda definir como nombre de la clave primaria, el nombre de la tabla con el prefijo PK_. Si no se especifica el nombre de la restricción, SQL Server le asigna un nombre.

- columnaX, columnaP, es la columna ó combinación de columnas que se define como clave primaria. Las columnas involucradas no deben permitir valores nulos, y además, no deben tener valores duplicados. En el caso de una combinación de columnas, la combinación vista como una unidad no debe tener valores duplicados.

Ejercicio 43: Creación de la clave primaria de la tabla Cliente

```
USE BDEjemplo_01
GO
ALTER TABLE Cliente
ADD CONSTRAINT pk_cliente
PRIMARY KEY (IDCliente);
GO
```

Ejercicio 44: **Creación de una clave primaria de la tabla Empleado**

```
USE BDEjemplo_01
GO
ALTER TABLE Empleado
ADD CONSTRAINT pk_empleado
PRIMARY KEY (IDEmpleado);
GO
```

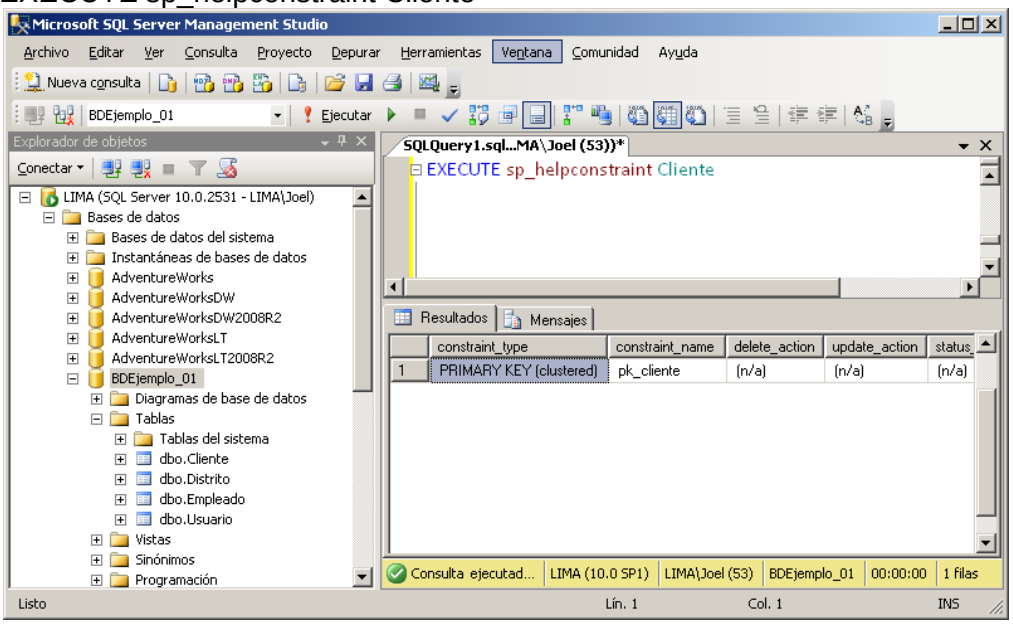
Ejercicio 45: Creación de una clave primaria de la tabla Producto

```
USE BDEjemplo_01
GO
ALTER TABLE Producto
ADD CONSTRAINT pk_producto
PRIMARY KEY (IDProducto);
GO
```

El procedimiento almacenado sp_helpconstraint

Devuelve la lista de todos los tipos de restricciones, el nombre definido por el usuario o proporcionado por el sistema, las columnas en que se han definido y la expresión que define la restricción (sólo para las restricciones DEFAULT y CHECK).

EXECUTE sp_helpconstraint Cliente



La Restricción UNIQUE

Puede utilizar las restricciones UNIQUE para asegurar que no se escriban valores duplicados en columnas específicas que no formen parte de una clave principal. Aunque tanto una restricción UNIQUE como PRIMARY KEY exigen que los elementos sean únicos, es preferible utilizar una restricción UNIQUE en vez de una restricción PRIMARY KEY cuando desee exigir la unicidad de:

- Una columna o una combinación de columnas que no sea la clave principal.
  En una tabla se puede definir varias restricciones UNIQUE, pero sólo una restricción PRIMARY KEY.
- Una columna que acepte valores NULL.

Las restricciones UNIQUE pueden definirse en columnas que aceptan valores NULL, mientras que las restricciones PRIMARY KEY sólo pueden definirse en columnas que no aceptan valores NULL.

Sintaxis

```
ALTER TABLE nombre_tabla
    ADD CONSTRAINT nombreRestriccion
    UNIQUE( columnaX, columnaP, … )
```

- Nombre Restriccion es el nombre de la restricción valor no duplicado ó UNIQUE. Se recomienda definir como nombre de la restricción, el nombre de la tabla seguido del nombre de la columna afectada, todo con el prefijo U_. Si no se especifica el nombre de la restricción, SQL Server le asigna un nombre.
- columnaX, columnaP, es la columna ó combinación de columnas a la que se aplica la restricción.

Ejercicio 46: **Creación de restricción UNIQUE para la columna Cli_RazonSocial en la tabla Cliente**

```
USE BDEjemplo_01
GO
ALTER TABLE Cliente
ADD CONSTRAINT u_nombre
UNIQUE (Cli_RazonSocial);
GO
```

La propiedad IDENTITY

Crea una columna de identidad en una tabla. Esta propiedad se utiliza con las instrucciones CREATE TABLE y ALTER TABLE de Transact-SQL.

Ejercicio 47: Creación de la tabla Categoria con el campo IDCategoria de tipo INT y que tenga la propiedad IDENTITY.

```
USE BDEjemplo_01
GO
--
CREATE TABLE Categoria(
IDCategoria     INT NOT NULL IDENTITY,
Nombre          VARCHAR(30) NOT NULL
);
GO
```

Cuando ingrese valores a la tabla Categoría solo debe ingresar valores en la columna Nombre, el sistema genera de forma automáticamente los valores en la columna IDCategoría; esto garantiza que dichos valores no se duplican en la tabla.

Ejercicio 48:

```
INSERT INTO Categoria(Nombre)
        VALUES ('Computadoras');
INSERT INTO Categoria(Nombre)
        VALUES ('Impresoras');
INSERT INTO Categoria(Nombre)
        VALUES ('Monitores');
```

Integridad de dominio

Se conoce como el dominio de un atributo al conjunto de valores aceptables para dicho atributo. La integridad de dominio establece qué condiciones deben cumplir los valores a insertar en una columna.

La integridad de dominio se define mediante reglas de validación, valores predeterminados, conjunto de valores permitidos en la columna (llave foránea), tipo y formato de los datos. Por ejemplo, en una tabla Empleado se puede requerir que el valor mínimo válido para la columna sueldoBase sea 420; cualquier valor menor a éste debe ser rechazado por la base de datos.
La integridad de dominio se puede implementar mediante:

- Restricciones CHECK
- Restricciones FOREIGN KEY
- Definiciones DEFAULT
- Definiciones NOT NULL y reglas.

Restricciones CHECK

Las restricciones CHECK exigen la integridad del dominio mediante la limitación de los valores que se pueden asignar a una columna.

Una restricción CHECK especifica una condición de búsqueda de tipo booleano (TRUE o FALSE) que se aplica a todos los valores que se escriben en la columna; los valores para los que el resultado no sea TRUE se rechazan. En una misma columna se pueden especificar varias restricciones CHECK. Como ejemplo se puede crear una restricción CHECK que compruebe que la fecha ingresada sea igual o mayor a la fecha del sistema.

Creación de regla de validación (CHECK)

Sintaxis

```
ALTER TABLE nombre_tabla
        ADD CONSTRAINT CK_nombre_tabla_nombre_columna
        CHECK( condición )
```

- CK_nombre_tabla_nombre_columna es el nombre de la restricción regla de validación ó CHECK. Se recomienda definir como nombre de la restricción, el nombre de la tabla

- seguido del nombre de la columna afectada, todo con el prefijo CK_. Si no se especifica el nombre de la restricción, SQL Server le asigna un nombre.
- condición es la expresión que determina cómo debe ser el valor a ingresar en la columna afectada por la restricción.

Ejercicio 49: Creación de restricción CHECK para la columna IDDistrito de la tabla Distrito.

El IDDistrito debe tener el siguiente formato L99, empezar con la letra L y luego dos dígitos.

```
SELECT * FROM Distrito
ALTER TABLE Distrito
ADD CONSTRAINT ck_IDDistrito
CHECK (IDDistrito LIKE '[L][0-1][0-1]')
--
INSERT INTO Distrito (IDDistrito,Nombre,Iniciales)
        VALUES('L01','Cercado de Lima', 'CER');
INSERT INTO Distrito (IDDistrito,Nombre,Iniciales)
        VALUES('L28','Independencia', 'IND');
```

Restricciones FOREIGN KEY

Las restricciones FOREIGN KEY identifican las relaciones entre las tablas.
Una clave externa de una tabla apunta a una clave candidata de otra tabla. Las claves externas evitan acciones que podrían dejar filas con valores de claves externas cuando no hay claves candidatas con ese valor.

Por ejemplo cuando se ingrese un valor en la columna IDDistrito en la tabla Cliente este valor debe existir en la tabla Distrito, en caso contrario se debe producir un error.

Creación de clave foránea

Sintaxis

```
ALTER TABLE nombre_tabla
    ADD CONSTRAINT FK_nombre_tabla_tabla_referenciada
    FOREIGN KEY( columnaX, columnaP, … )
    REFERENCES tabla_referenciada
```

- FK_nombre_tabla_tabla_referenciada es el nombre de la restricción clave foránea. Se recomienda definir como nombre de la clave foránea, el nombre de la tabla seguido del nombre de la tabla referenciada, todo con el prefijo FK_. Si no se especifica el nombre de la restricción, SQL Server le asigna un nombre.
- columnaX, columnaP, es la columna ó combinación de columnas que se define como clave foránea.
- tabla_referenciada es el nombre de la tabla primaria con la que se relaciona la tabla secundaria que tiene la clave foránea. De modo predeterminado, la clave foránea hace referencia a la clave primaria de la tabla primaria.

Ejercicio 50:

Nota: Antes de crear la clave foránea de la tabla Cliente se debe crear la clave primaria de la tabla Distrito

```
ALTER TABLE Distrito
    ADD CONSTRAINT pk_distrito
    PRIMARY KEY (IDDistrito);
```

Ejercicio 51: Creación de la clave foránea en la tabla Cliente que referencia a la tabla

Distrito.

```
ALTER TABLE Cliente
    ADD CONSTRAINT fk_IDDistrito
    FOREIGN KEY (IDDistrito)
    REFERENCES Distrito (IDDistrito)
```

La columna IDDistrito de la tabla Cliente establece la relación de esta tabla con la tabla Distrito. La clave foránea en IDDistrito de Cliente apunta a la clave primaria en IDDistrito de la tabla Distrito.

Ejercicio 52: **Creación de la clave foránea en la tabla Producto que referencia a la tabla Categoría.**

Nota: Antes de crear la clave foránea de la tabla Producto se debe añadir la columna IDCategoria a la tabla Producto.

```
ALTER TABLE Producto
ADD IDCategoria INT
GO
```

Ahora procedemos a crear la clave foránea de la tabla Producto.

```
ALTER TABLE Producto
    ADD CONSTRAINT fk_IDCategoria
    FOREIGN KEY (IDCategoria)
    REFERENCES Categoria (IDCategoria);
```

Definiciones DEFAULT

Los valores DEFAULT especifican qué valores se utilizan en una columna si no se especifica un valor al insertar las filas. Los valores predeterminados pueden ser cualquier expresión cuyo resultado sea una constante, como:

- Constante
- Función integrada
- Expresión matemática

Creación de valor predeterminado (DEFAULT)

Sintaxis

```
ALTER TABLE nombre_tabla
    ADD CONSTRAINT DF_nombre_tabla_nombre_columna
    DEFAULT valor_predeterminado FOR columnaX
```

- DF_nombre_tabla_nombre_columna es el nombre de la restricción valor predeterminado ó DEFAULT. Se recomienda definir como nombre de la restricción, el nombre de la tabla seguido del nombre de la columna afectada, todo con el prefijo DF_. Si no se especifica el nombre de la restricción, SQL Server le asigna un nombre.
- valor_predeterminado es el valor que se almacena en columnaX cuando al insertar una fila no se especifica el valor para esa columna.
- columnaX es la columna a la que se aplica la restricción.

Ejercicio 53: **Creación de restricción DEFAULT para la columna Discontinuo en la tabla Producto**

Para cada producto, el valor para la columna Discontinuo es 0.

```
ALTER TABLE Producto
        ADD CONSTRAINT DF_Pro_Discontinuo
        DEFAULT 0 FOR Pro_Discontinuo
GO
```

Pruebe la restricción insertando un Producto en el que no se establece de modo explícito el valor para la columna Pro_Discontinuo

Integridad referencial

La integridad referencial garantiza que la relación entre la llave primaria (en la tabla referenciada) y la llave foránea (en la tabla de referencia) siempre se mantiene. Una fila en una tabla referenciada no puede anularse, ni cambiar su valor de la llave primaria, si una llave foránea se refiere a la fila. Por ejemplo, si tenemos las tablas Dependencia y Empleado, cada empleado debe pertenecer a solo una dependencia; es decir, cada fila de la tabla Empleado debe estar relacionada con una fila de la tabla Dependencia, pero una fila de la tabla Dependencia puede estar relacionada con muchas filas de la tabla Empleado ya que una dependencia puede tener muchos empleados.

Integridad definida por el usuario

La integridad definida por el usuario le permite definir reglas de la compañía específicas que no pertenecen a ninguna otra categoría de integridad. Todas las categorías de integridad son compatibles con la integridad definida por el usuario (todas las restricciones en columnas y tablas de CREATE TABLE, procedimientos almacenados y desencadenadores).

Resumen

La tabla siguiente describe los diferentes tipos de restricciones.

Tipo de restricción	Descripción
PRIMARY KEY (clave primaria)	Garantiza que cada fila ó registro en una tabla es único(a). La columna ó combinación de columnas definida como clave primaria no permite valores duplicados.
UNIQUE (valor no duplicado)	Garantiza que cada valor en una columna es único. Permite valores únicos
FOREIGN KEY (clave foránea)	Define la columna ó combinación de columnas de una tabla secundaria cuyos valores dependen de la clave primaria de una tabla primaria.
DEFAULT (valor predeterminado)	Establece el valor predeterminado para una columna cuando al insertar una fila no se especifica el valor para dicha columna.
CHECK (regla de validación)	Establece la regla que debe cumplir un valor para que sea un valor aceptable en una columna.

5

Modificación de datos

Cuando se termine de crear las tablas y restricciones es necesario ingresar los datos, modificar los datos existentes o quizás eliminar algunos datos, para lograr estos objetivos SQL Server proporciona un conjunto de sentencias que forman lo que se denomina el Lenguaje de manipulación de datos (DML).

El Lenguaje de manipulación de datos (DML) se utiliza para seleccionar, insertar, actualizar y eliminar datos en los objetos definidos mediante sentencias de definición de datos (DDL).

Las sentencias utilizadas son:

- SELECT
- INSERT
- UPDATE
- DELETE

Inserción de filas (la instrucción INSERT)

El proceso de inserción de filas consiste en añadir a una tabla una o más filas y en cada fila todos o parte de sus campos.

Sintaxis

```
INSERT [INTO] nombre_tabla[ ( lista_de_columnas ) ]
        VALUES( lista_de_valores )
```

- lista_de_columnas es la relación de columnas en las que se almacenarán los valores especificados en lista_de_valores.
- lista_de_valores es la relación de los valores a almacenar en la fila a insertar.
- Los elementos de ambas listas van separados por comas

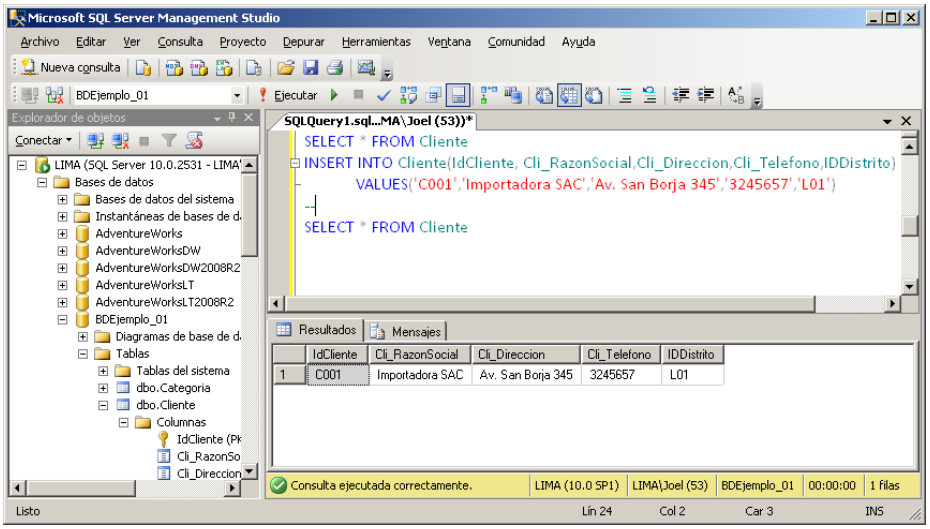
Ejercicio 54: Inserción de una fila con lista de valores completa a la tabla Cliente

```
INSERT INTO Cliente(IdCliente, Cli_RazonSocial ,  Cli_Direccion, Cli_Telefono, IDDistrito)
        VALUES('C001','Importadora SAC','Av. San Borja 345','3245657','L01')
```

Para comprobar la instrucción realizamos una consulta a la tabla Cliente

```
SELECT * FROM Cliente
```

Observe los resultados en el siguiente gráfico:



Ejercicio 55: Inserción de una fila en la tabla Categoría

La tabla Categoria tiene la columna IDCategoria con la propiedad IDENTITY por lo tanto no se debe incluir en la lista de campos y tampoco proporcionar un valor.

```
INSERT INTO Categoria (Nombre)
        VALUES ('Tarjetas de red')
--
SELECT * FROM Categoria
```

Resultados | Mensajes

	IDCategoria	Nombre
1	1	Computadoras
2	2	Impresoras
3	3	Monitores
4	4	Tarjetas de red

Cambiar datos

Una vez que se han creado las tablas y se han agregado los datos, cambiar o actualizar los datos de las tablas se convierte en uno de los procesos diarios de mantenimiento de una base de datos. Microsoft® SQL Server™ proporciona la instrucción UPDATE para cambiar los datos de una tabla existente

Actualización de datos (la instrucción UPDATE)

Sintaxis

```
UPDATE nombre_tabla
        SET     columnaX = expresiónX,
                columnaP = expresiónP, ...
        [ WHERE condición_de_las_filas_a_actualizar ]
```

- columnaX, columnaP son las columnas cuyo contenido se actualizará.
- expresiónX, expresiónP establecen los nuevos valores a almacenar en las columnas columnaX y columnaP respectivamente.
- condición_de_las_filas_a_actualizar es una expresión lógica que determina las filas en las que la actualización se debe llevar a cabo.

Ejercicio 56: Uso de UPDATE en la tabla Cliente cambiando la dirección y el teléfono del cliente con código C001

Av. San Borja 345 → Av. San Borja Norte 1345

3245657 → 6341223

```
UPDATE Cliente
        SET Cli_RazonSocial='Av. San Borja  Norte 1345',
```

```
        Cli_Telefono='6341223'
WHERE IdCliente='C001'
```

Eliminación de filas (la instrucción DELETE)

Sintaxis

```
DELETE [FROM] nombre_tabla
    [ WHERE condición_de_las_filas_a_eliminar ]
```

- condición_de_las_filas_a_eliminar es una expresión lógica que determina las filas en las que la eliminación se debe llevar a cabo.

Ejercicio 57: **Uso de DELETE Eliminar de la tabla Cliente al cliente con código C001**

```
USE BDEjemplo_01
GO
DELETE FROM Cliente
WHERE IdCliente='C001'
```

La instrucción TRUNCATE TABLE

La instrucción TRUNCATE TABLE es un método rápido y eficiente para eliminar todas las filas de una tabla. TRUNCATE TABLE es equivalente a la instrucción DELETE sin una cláusula WHERE. Sin embargo, TRUNCATE TABLE es más rápida y utiliza menos recursos de registro de sistema y de transacciones.

Ejemplo de TRUNCATE TABLE

```
TRUNCATE TABLE Cliente
```

La instrucción DROP TABLE

Quita una definición de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos de la tabla.

6

Consultas

Una consulta es una petición de datos almacenados en Microsoft® SQL Server. Aunque las consultas tienen varias formas de interactuar con un usuario, todas realizan la misma tarea: presentan al usuario el conjunto de resultados de una instrucción SELECT. La instrucción SELECT recupera los datos de SQL Server y los presenta de nuevo al usuario en uno o más conjuntos de resultados. Un conjunto de resultados es una organización tabular de los datos obtenidos de SELECT. Al igual que una tabla de SQL, el conjunto de resultados está formado por columnas y filas.

La sintaxis completa de la instrucción SELECT es compleja, aunque la mayor parte de las instrucciones SELECT describen las cuatro propiedades principales de un conjunto de resultados:

La instrucción SELECT

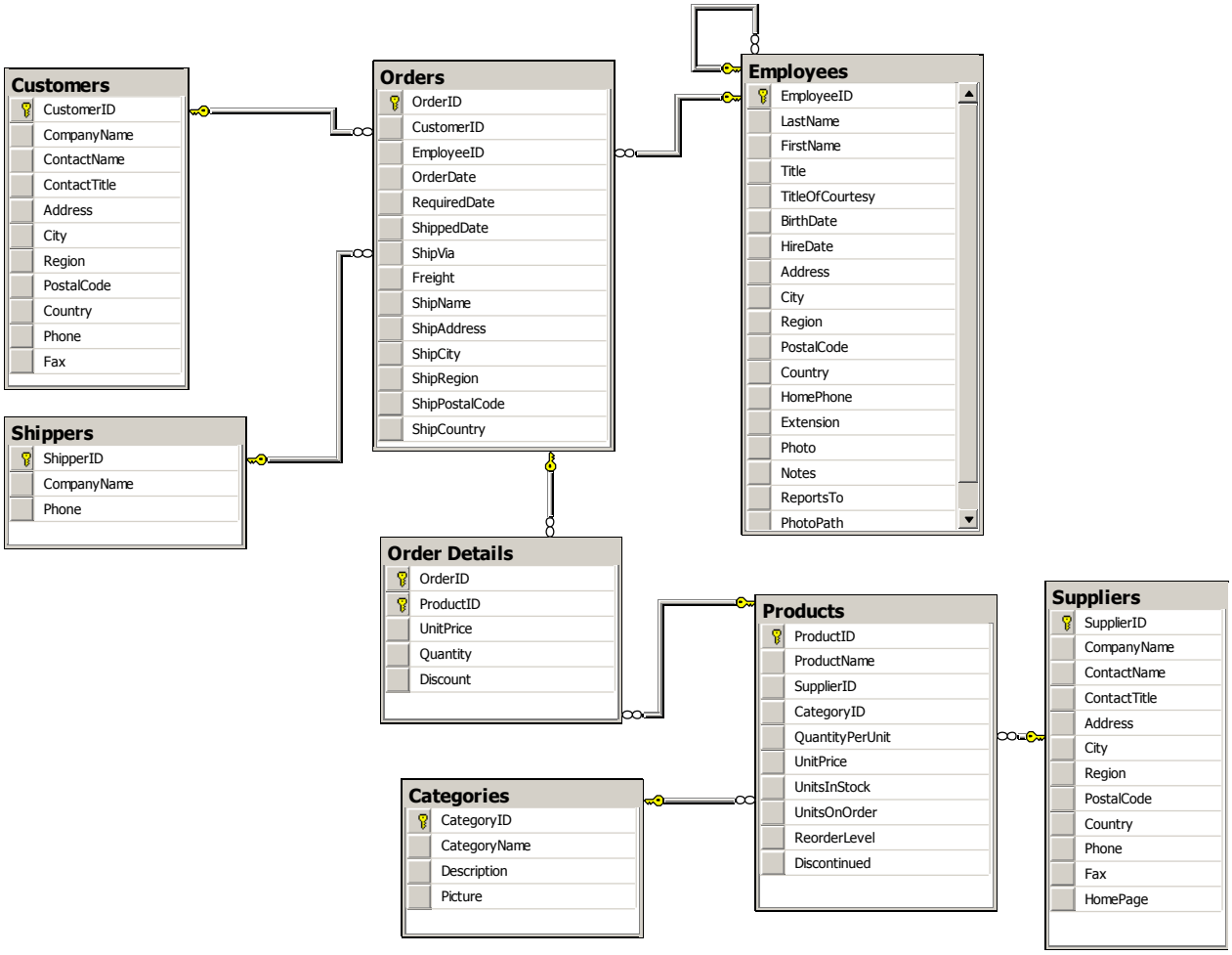
Hay tres componentes básicos en la instrucción SELECT: SELECT, FROM y WHERE. A continuación la Sintaxis básica.

Sintaxis básica

```
SELECT * | lista_columnas

FROM nombre_tabla

[ WHERE condición_filas ]
```

- lista_columnas es la lista de columnas a mostrar en el resultado de la consulta. Si se especifica * se mostrarán todas las columnas de la tabla.
- condición_filas es una expresión lógica que indica que las filas a mostrar son aquellas para las que el valor de la expresión es verdadero.

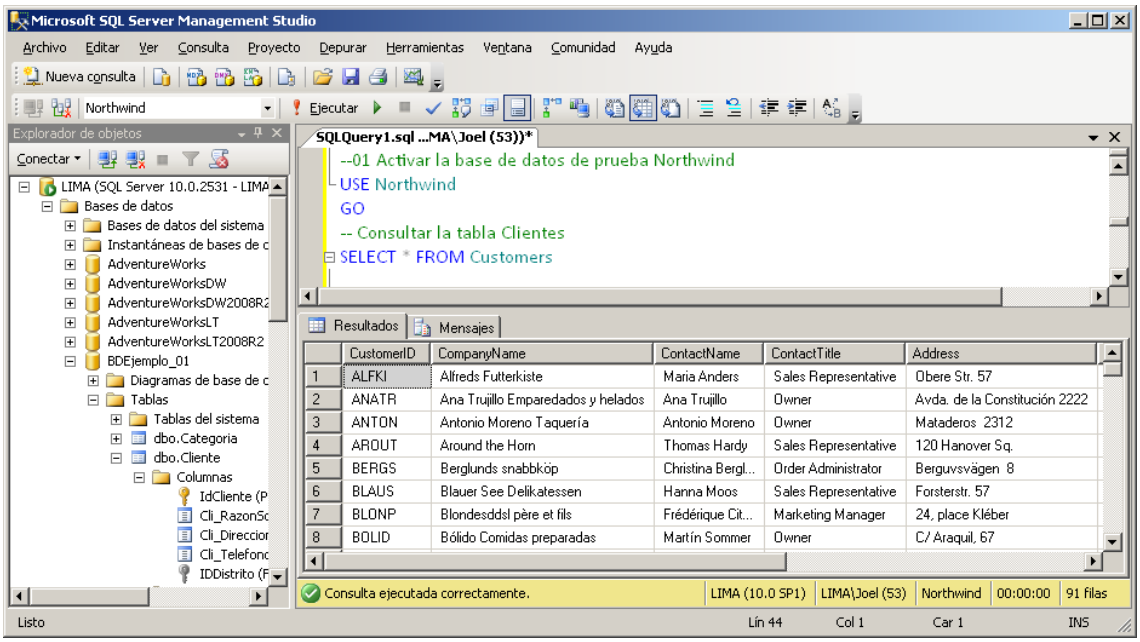
El siguiente diagrama muestra el modelo de la base de datos Northwind que se usará en los ejemplos.



Ejercicio 58: Lectura de todos los datos de la tabla Customers. Utilizar el carácter * para indicar que se solicitan todas las columnas.

```
--01 Activar la base de datos de prueba Northwind
USE Northwind
GO
-- Consultar la tabla Clientes
```

```
SELECT * FROM Customers
-- Consultar la tabla Productos
SELECT * FROM Products
```



Al dar un clic en la pestaña Mensajes se visualiza el número de filas devueltas como un conjunto de resultados.

(91 filas afectadas)

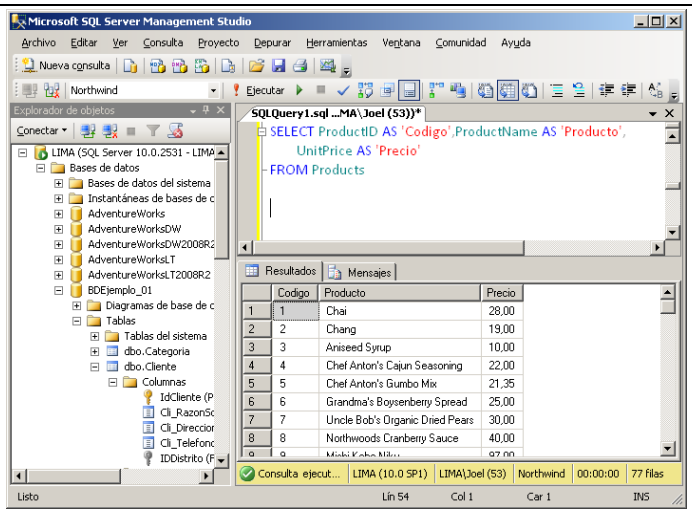
Lectura de columnas seleccionadas de una tabla

Ejercicio 59: Lectura de todos los datos de la tabla Customers mostrando las columnas ProductID,ProductName UnitPrice

```
SELECT ProductID ,ProductName,UnitPrice
FROM Products
```

Ejemplo: Utilizando un alias para las cabeceras de columnas

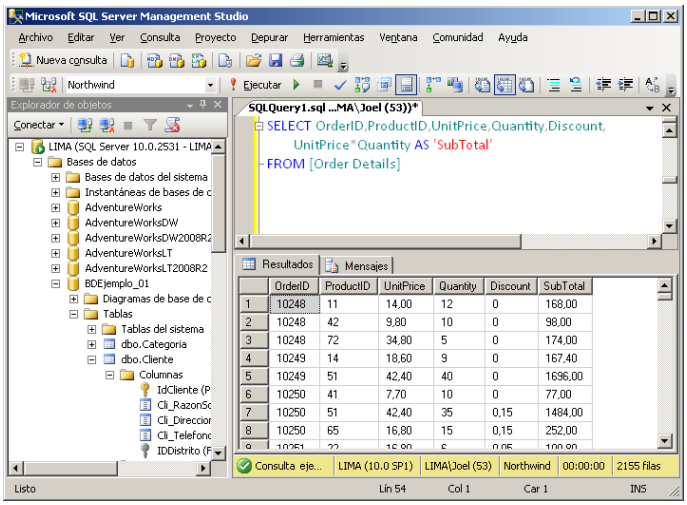
```
-- Utilizar un alias

SELECT ProductID AS 'Codigo',ProductName AS 'Producto',
      UnitPrice AS 'Precio'
FROM Products
```



Ejercicio 60: Definiendo columnas computadas o calculadas. En la siguiente consulta se crea la columna calculada SubTotal que se obtiene multiplicando UnitPrice*Quantity

Una columna computada es una columna que se muestra en el resultado de una consulta, pero que no existe físicamente como tal en la tabla. La columna computada muestra el resultado de ejecutar alguna operación con las columnas de la tabla.

```
SELECT OrderID,ProductID,UnitPrice,Quantity,Discount,
    UnitPrice*Quantity AS 'SubTotal'
FROM [Order Details]
```



Definición de filtros de fila en SELECT

En la cláusula WHERE, condición_filas es una expresión lógica que establece la condición que deben cumplir las filas a mostrar en el resultado de la consulta. Para construir la expresión lógica utilice operadores relacionales ó de comparación y operadores lógicos SQL como LIKE, BETWEEN e IN.

Ejercicio 61: Uso del operador de igualdad (=)

```
-- Listado de los clientes de Mexico
SELECT * FROM Customers
WHERE Country='Mexico'

--Listado de los clientes de France
SELECT * FROM Customers
WHERE Country='France'
```

Ejercicio 62:

```
-- Listado de los productos de la categoria 2
SELECT * FROM Products
WHERE CategoryID=2
```

Ejercicio 63: Uso del operador mayor que >

```
-- Listado de productos con precio mayor a 20
SELECT * FROM Products
WHERE UnitPrice> 20

-- Listado de productos con precio entre 20 y 30
SELECT * FROM Products
WHERE UnitPrice >= 20   AND UnitPrice <=30
```

Búsqueda basada en rango de valores – El operador BETWEEN

El operador BETWEEN permite ejecutar consultas que ejecutan búsquedas basadas en rango de valores numéricos, valores de cadena, ó valores de fecha.

Sintaxis

```
SELECT * | lista_columnas

FROM nombre_tabla

WHERE columna BETWEEN valor_inicial AND valor_final
```

BETWEEN indica que el valor en columna debe encontrarse en el rango definido por valor_inicial y valor_final.

- columna, es la columna en la que se busca según el rango especificado por valor_inicial y valor_final.
- valor_inicial, valor_final, establecen los límites del rango de valores en el que se basa la búsqueda.

Ejercicio 64: Uso del operador BETWEEN

```
-- Listado de productos con precio entre 20 y 30
-- operador BETWEEN
SELECT * FROM Products
WHERE UnitPrice BETWEEN 20 AND 30
```

Ejercicio 65: Uso del operador OR

```
-- Listado de  clientes de Mexico y Francia
SELECT * FROM Customers
WHERE Country='Mexico' OR Country='France'
ORDER BY Country -- 16 filas
```

Ejercicio 66: **Listado de productos de la categoria 1,2 y 3 -- 37 filas**

```
SELECT * FROM Products
WHERE CategoryID=1 OR
    CategoryID=2 OR
    CategoryID=3
```

Ejercicio 67: Uso del operador AND

```
-- Listado de productos de la categoria 1  con stock entre 10 y 30
SELECT * FROM Products
WHERE CategoryID=1 AND
    UnitsInstock BETWEEN 10 AND 30
```

Búsqueda basada en conjunto de valores – El operador IN

El operador IN permite ejecutar consultas que ejecutan búsquedas basadas en conjuntos de valores numéricos, valores de cadena, ó valores de fecha.

Sintaxis

```
SELECT * | lista_columnas

FROM nombre_tabla

WHERE columna [NOT]IN ( conjunto_de_valores )
```

IN indica que el valor en columna debe encontrarse (IN), ó no debe encontrarse (NOT IN) en el conjunto definido por conjunto_de_valores.

- columna, es la columna en la que se busca según el conjunto de valores especificado en conjunto_de_valores.
- conjunto_de_valores, establece el conjunto de valores en el que se basa la búsqueda.

Ejemplo: Uso del operador IN

Ejercicio 68: **Listado de productos de la categoria 1,2 y 3 -- 37 filas**

```
SELECT * FROM Products
WHERE CategoryID IN (1,2,3)
```

Ejercicio 69: **Uso del operador AND y OR**

-- Listado de las ordenes emitidas por los clientes ANTON y VINET y que fueron atendidos por los empleados con código 1 y 5

```
SELECT * FROM ORDERS
WHERE (CustomerID='ANTON' OR CustomerID='VINET') AND
    (EmployeeID=1 OR EmployeeID=5)
```

Consultas a cadenas de caracteres. El operador LIKE

Determina si una cadena de caracteres dada coincide o no con un determinado modelo. Un modelo puede incluir caracteres normales y caracteres comodín. Durante la coincidencia de patrones, los caracteres regulares deben coincidir exactamente con los caracteres especificados en la cadena de caracteres. Sin embargo, los caracteres comodín pueden coincidir con fragmentos arbitrarios de la cadena de caracteres. Con los caracteres comodín el operador LIKE es más flexible que los operadores de comparación de cadenas = y !=. Si alguno de los argumentos no es del tipo de datos de cadena de caracteres, Microsoft® SQL Server™ lo convierte al tipo de datos de cadena de caracteres, si es posible.

Sintaxis

```
SELECT * | lista_columnas
FROM nombre_tabla
WHERE columna LIKE expresión_cadena_a_buscar
```

- columna, es la columna en la que se busca la cadena de caracteres.
- expresión_cadena_a_buscar, indica como debe ser la cadena que se está buscando en columna. La expresión admite comodines.

Los comodines del operador LIKE

El siguiente cuadro muestra los comodines que puede utilizar con el operador LIKE.

Comodín	Descripción
%	Indica que en la posición del comodín puede ir cualquier cadena de caracteres, incluso una cadena nula.
_	Indica que en la posición del comodín puede ir cualquier carácter no nulo.
[abc]	Establece el conjunto de caracteres válidos en la posición del comodín.
[a-b]	Establece el rango de caracteres válidos en la posición del comodín.
^	Excluir. Indica que el carácter, conjunto de caracteres, ó rango de caracteres que sigue al símbolo **^** no debe figurar en el resultado de la consulta.

Ejercicio 70: Uso del comodín %

Lista de clientes con la letra C como primer carácter en el nombre

```
SELECT * FROM Customers
WHERE  CompanyName LIKE 'C%'
```

Lista de clientes con la letra a como último carácter en el nombre

```
SELECT * FROM Customers
WHERE CompanyName LIKE '%a'
```

Ejercicio 71: **Lista de clientes con la letra a como primer y último carácter en el nombre**

```
SELECT * FROM Customers
WHERE CompanyName LIKE 'a%a'
```

Ejercicio 72: Uso del comodín _

Listado de clientes con la letra c como tercer carácter en el nombre

```
SELECT * FROM Customers
WHERE CompanyName LIKE '__W%'
```

Ejercicio 73: **Uso del comodín [abc]**

Listado de clientes con la letra a,b,c,d como primer carácter en el nombre

```
SELECT * FROM Customers
WHERE CompanyName LIKE '[abcd]%'
```

Ejercicio 74: **Uso del comodín ^**

Listado de clientes que no tengan la letra a,b,c,d como primer carácter en el nombre

```
SELECT * FROM Customers
WHERE CompanyName LIKE '[^abcd]%'
```

Ejercicio 75: **Uso del comodín [a-c]**

Listado de productos que tengan como primer carácter a,b,c,d,e

```
SELECT * FROM Products
WHERE ProductName LIKE '[a-e]%'
```

Ejercicio 75: **Listado de productos que no tengan como primer carácter a,b,c,d,e**

```
SELECT * FROM Products
WHERE ProductName LIKE '[^a-e]%'
```

Ejercicios

Ejercicio 76: **Productos con la frase choco en el nombre**

```
SELECT * FROM Products
WHERE ProductName LIKE '%choco%'
```

Ejercicio 77: **Productos que tengan como primer y último carácter una vocal.**

```
SELECT * FROM Products
WHERE ProductName LIKE '[aeiou]%[aeiou]'
```

Ejercicio 78: **Clientes con primer carácter a o f y tercer carácter a**

```
SELECT * FROM Customers
WHERE CompanyName LIKE '[af]_a%'
```

Funciones de agregación

Son funciones que permiten efectuar una operación aritmética que resume los valores de una columna de toda la tabla, o que resume los valores de la columna agrupados según determinado criterio. La función produce un solo valor que es el resumen de la tabla, o de cada uno de los grupos.

Puede usar las funciones de agregación con la declaración SELECT o en combinación con la cláusula GROUP BY.

Con excepción de la función COUNT(*), todas las funciones de resumen retornan NULL si ninguna fila satisface la cláusula WHERE. La función COUNT (*) retorna un valor de cero si ninguna fila satisface la cláusula WHERE.

Las funciones de agregación son: AVG (), COUNT (), MAX (), MIN () y SUM ()

El siguiente cuadro presenta las funciones de agregación más utilizadas.

Función	Descripción
AVG()	Retorna el promedio de los valores de una columna ó expresión.
COUNT()	Retorna la cuenta del número de filas de una consulta.
MAX()	Retorna el valor máximo de una columna ó expresión.
MIN()	Retorna el valor mínimo de una columna ó expresión.
SUM()	Retorna la suma de los valores de una columna ó expresión.

Ejercicio 79: **Uso de la función AVG()**

Sintaxis

AVG([DISTINCT] expresión numérica)

- DISTINCT indica que debe eliminarse los valores duplicados de expresión_numérica antes de evaluar la función.

Ejercicio 80: Obtener el precio unitario promedio de todos los productos en la tabla Products de la base de datos Northwind.

```
SELECT AVG(UnitPrice) AS 'Precio promedio'
FROM Products
```

Ejercicio 81: Obtener el precio unitario máximo de todos los productos en la tabla Products de la base de datos Northwind.

```
SELECT * FROM Products
SELECT MAX(UnitPrice) AS 'Precio mayor'
FROM Products
```

Ejercicio 81: Obtener el precio unitario mínimo de todos los productos en la tabla Products de la base de datos Northwind.

```
SELECT MIN(UnitPrice) AS 'Precio menor'
FROM Products
```

Ejercicio 82: Obtener el stock valorado en la tabla Products de la base de datos Northwind.

```
 SELECT SUM(UnitPrice*UnitsInStock) AS 'stock valorado'
FROM Products
```

Ejemplo: Uso de la función COUNT()

Sintaxis

COUNT([DISTINCT] expresión)

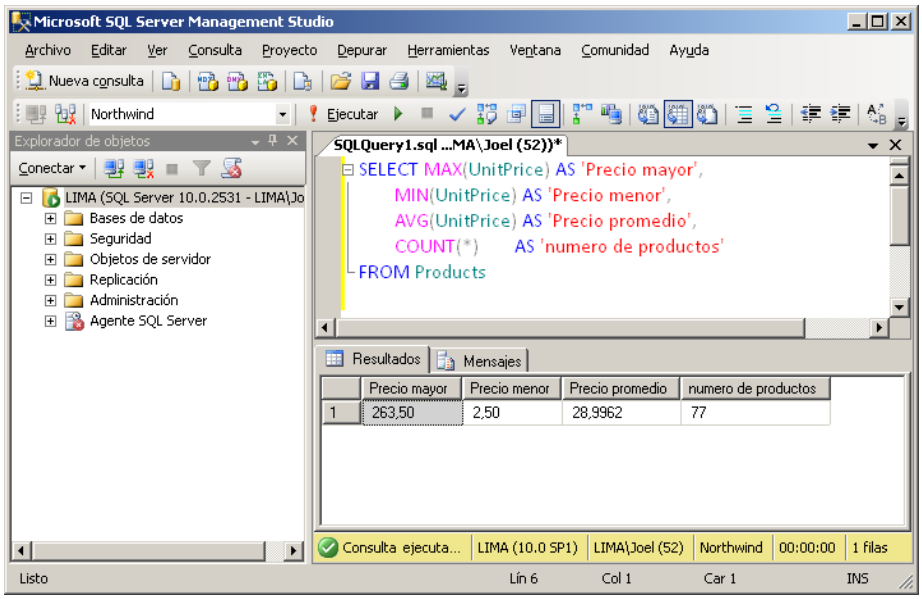
COUNT(*)

- DISTINCT indica que debe eliminarse los valores duplicados de expresión antes de evaluar la función.

- COUNT(expresión) ignora los valores NULL de expresión; COUNT(*) se utiliza para contar filas, por lo que no ignora los valores NULL.

Ejercicio 83: Obtener el precio unitario máximo, mínimo, promedio de todos los productos en la tabla Products de la base de datos Northwind. Además mostrar el número de productos.

```
SELECT MAX(UnitPrice) AS 'Precio mayor',
       MIN(UnitPrice) AS 'Precio menor',
       AVG(UnitPrice) AS 'Precio promedio',
       COUNT(*)       AS 'numero de productos'
FROM Products
```



La cláusula GROUP BY

La cláusula GROUP BY se utiliza para agrupar las filas en base a determinado criterio, y luego ejecutar una operación que resume un atributo para cada uno de los grupos así formados. Por ejemplo, puede utilizar GROUP BY para agrupar todas las facturas por cliente, y luego calcular el monto total facturado de cada cliente.

Sintaxis

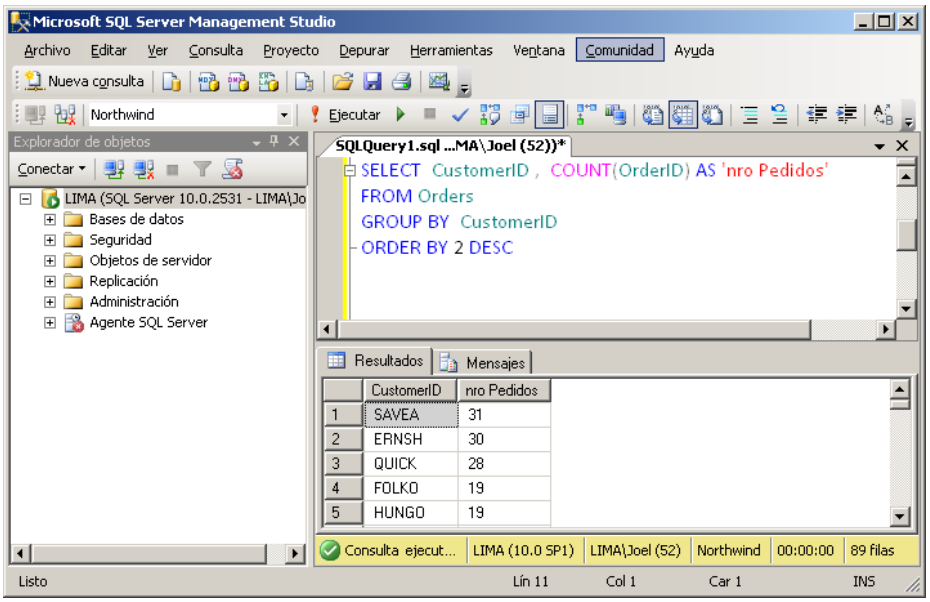
```
SELECT lista_columnas, función_agregación(columna),
        función_agregación(columna), ...
FROM nombre_tabla
[ WHERE condición_filas ]
GROUP BY lista_columnas
        [ HAVING condición_grupos ]
```

- Las columnas presentes en lista_columnas de la cláusula GROUP BY deben necesariamente estar presentes en la lista_columnas de SELECT.
- Cualquier columna presente en SELECT, y que no se encuentre en la lista_columnas de GROUP BY debe estar afectada por una función_agregación.
- condición_grupos en la cláusula HAVING permite establecer una expresión lógica que indica que los grupos a mostrar en el resultado son aquellos para los que el valor de la expresión es verdadero.
- Una consulta GROUP BY solo entrega una fila por cada grupo generado. Esta fila muestra el resultado de la función_agregación aplicada sobre el grupo. No muestra el contenido del grupo.
- Cuando se utiliza GROUP BY sobre una columna que contiene valores NULL, éstos se procesan como un grupo.

Ejercicio 84: Uso de la cláusula GROUP BY

Mostrar un listado que muestre el código del Cliente y el número de pedidos que realizo cada cliente.

```
SELECT  CustomerID ,  COUNT(OrderID) AS 'nro Pedidos'
FROM Orders
GROUP BY  CustomerID
ORDER BY 2 DESC
```



Ejercicio 85: **Mostrar un listado que muestre el código del Empleado y el número de pedidos que atendió cada empleado.**

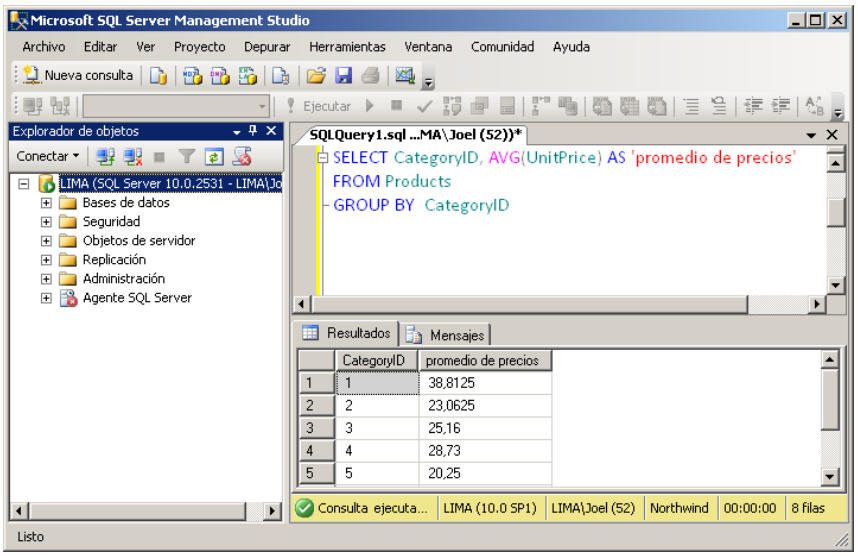
```
SELECT EmployeeID ,COUNT(OrderID) AS 'Nro. pedidos'
FROM Orders
GROUP BY  EmployeeID
```

Ejercicio 86: **Mostrar un listado que muestre el código de categoria y el número de productos que existe en cada categoria.**

```
SELECT CategoryID, COUNT(*) AS 'Nro.  de productos'
FROM Products
GROUP BY CategoryID
```

Ejercicio 87: **Mostrar un listado que muestre el código de categoria y el promedio de precios por categoría.**

```
SELECT CategoryID, AVG(UnitPrice) AS 'promedio de precios'
FROM Products
GROUP BY  CategoryID
```



Funciones para manipulación de fechas

El siguiente cuadro muestra una lista de las funciones para manipular fechas.

Función	Descripción
getdate()	Retorna la fecha y hora del sistema.
getucdate()	Retorna la fecha y hora del Meridiano de Greenwich. El valor se obtiene a partir de la configuración regional del sistema.
dateadd(*parte_fecha*, *n*, *fecha***)**	Genera una nueva fecha a partir de *fecha*, añadiéndole a fecha *n* unidades de *parte_fecha*.
datediff(*parte_fecha*, *fecha1*, *fecha2***)**	Entrega la diferencia entre *fecha1* y *fecha2* en las unidades indicadas en *parte_fecha*.
datepart(*parte_fecha*, *fecha***)**	Devuelve, a partir de *fecha*, un valor entero con la *parte_fecha* especificada.

datename(*parte_fecha*, *fecha***)**	Devuelve, a partir de *fecha*, una cadena con la *parte_fecha* especificada.
day(*fecha***)**	Devuelve un entero con la parte del día de *fecha*.
month(*fecha***)**	Devuelve un entero con la parte del mes de *fecha*.
year(*fecha***)**	Devuelve un entero con la parte del año de *fecha*.

El siguiente cuadro muestra las parte_fecha que podemos especificar en las funciones de fecha listadas arriba.

parte_fecha	Abreviatura	Descripción
Year	yy, yyyy	Año de la fecha
Quarter	q, qq	Trimestre del año de la fecha
Month	m, mm	Mes del año de la fecha
Week	ww, wk	Semana del año de la fecha
Day	d, dd	Día de la fecha
dayofyear	y, dy	Día del año de la fecha
weekday	dw	Día de la semana de la fecha
Hour	hh	Hora de la fecha
Minute	n, mi	Minutos de la hora de la fecha
Second	s, ss	Segundos de la hora de la fecha
millisecond	ms	Milisegundos de la hora de la fecha

Ejemplos de uso de funciones de fecha

Ejercicio 88: **GETDATE(): retorna la fecha del sistema**

```
PRINT GETDATE()
```

Ejercicio 89: **Obtener el año de la fecha**

```
PRINT DATEPART(YEAR,GETDATE())
```

Ejercicio 90: **Obtener el mes de la fecha**

```
PRINT DATEPART(MONTH,GETDATE())
```

Ejercicio 91: **Obtener el día de la fecha**

```
PRINT DATEPART(DAY,GETDATE())
```

Ejercicio 92: **Obtener el trimestre del año de la fecha**

```
PRINT DATEPART(QUARTER,GETDATE())
```

Ejercicio 93: **Obtener la semana del año de la fecha**

```
PRINT DATEPART(WEEK,GETDATE())
```

Ejercicio 94: **Obtener Día del año de la fecha**

```
PRINT DATEPART(DAYOFYEAR,GETDATE())
```

Ejercicio 95: **Obtener Día de la semana de la fecha**

```
PRINT DATEPART(WEEKDAY,GETDATE())
```

Ejercicio 96: **Listado de órdenes emitidas en el año 1996**

```
--Activar la base de datos de ejemplo Northwind
USE Northwind
GO
SELECT * FROM Orders
WHERE DATEPART(YEAR,OrderDate)= 1996
GO
```

Ejercicio 97: **Listado de ordenes emitidas en el mes de febrero del año 1997**

```
SELECT * FROM Orders
WHERE DATEPART(YEAR,OrderDate)= 1997 AND
    DATEPART(MONTH,OrderDate)= 2
```

Ejercicio 98: **Listado de ordenes emitidas el segundo trimestre del año 1996**

```
SELECT * FROM  Orders
```

```
WHERE DATEPART(YEAR,OrderDate)= 1996 AND
    DATEPART(QUARTER,OrderDate)=2
GO
```

Ejercicio 99: **Listado de órdenes emitidas en la primera quincena del mes de Agosto del año 1996**

```
SELECT * FROM Orders
WHERE DATEPART(YEAR,OrderDate)= 1996 AND
    DATEPART(MONTH,OrderDate)=8 AND
    (DATEPART(DAY,OrderDate) BETWEEN 1 AND 15)
```

Uso de la function DATENAME(*parte_fecha*, *fecha***)**

La función DATENAME devuelve, a partir de *fecha*, una cadena con la *parte_fecha* especificada.

Ejemplos de uso de la función DATENAME()

Ejercicio 100: **Mostrar el nombre del mes correspondiente a la fecha del sistema.**

```
SET LANGUAGE Spanish
PRINT DATENAME(MONTH,GETDATE())
```

```
Se cambió la configuración de idioma a Español.
Agosto
```

Ejercicio 101: Mostrar el nombre del día de la semana correspondiente a la fecha del sistema.

```
PRINT DATENAME(WEEKDAY,GETDATE())
```

Ejercicio 102: Listado de ordenes emitidas un martes 13 o domingo 7.

```
SELECT * FROM Orders
WHERE DATEPART(DAY,OrderDate)=13 AND
    DATENAME(WEEKDAY,OrderDate)='Martes' OR
    (DATEPART(DAY,OrderDate)=7 AND
    DATENAME(WEEKDAY,OrderDate)='Domingo')
```

Consultas a varias tablas

En la mayoría de los casos, la recuperación de los datos que los usuarios necesitan para trabajar implica la lectura de muchas tablas para que la información así obtenida sea de utilidad para ellos. En este capítulo veremos el diseño de las instrucciones SELECT que nos permiten recuperar datos de varias tablas en un solo conjunto de resultados.

Consultas correlacionadas

Un join, combinación ó **consulta correlacionada** es la consulta que selecciona columnas de dos tablas ó conjuntos de filas, y las entrega en un único conjunto de resultados. Las Filas de las tablas ó conjuntos de filas se combinan relacionando valores comunes, típicamente valores de clave primaria y clave foránea.

Sintaxis general

```
SELECT lista_columnas
FROM tabla1
tipo_join JOIN tabla2 ON condición_del_join
```

- **lista_columnas** es la lista de columnas a mostrar en el resultado de la consulta. Se recomienda que cada columna sea calificada con el alias de la tabla a la cual pertenece.
- **tipo_join** indica si el join es interior (INNER), exterior (OUTER) ó irrestricto (CROSS).
- **condición_del_join** es una expresión que indica en base a qué columnas de cada una de las tablas se establece la relación entre ellas.

Una combinación (join) puede ser de cualquiera de los siguientes tipos:

- INNER JOIN
- OUTER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

- CROSS JOIN

Los ejemplos a continuación se ejecutan sobre la base de datos Northwind.

INNER JOIN

Un INNER JOIN es la consulta correlacionada que combina todas las filas que están relacionadas de las dos tablas ó conjuntos de filas.

Ejemplo: Uso de INNER JOIN

Ejercicio 103: Generar un listado que presente los siguientes campos: OrderID, CompanyName y OrderDate. La información requerida proviene de dos tablas Orders y Customers.

```
SELECT Orders.OrderID,  Customers.Companyname, Orders.OrderDate
FROM Orders
INNER JOIN  Customers
ON  Orders.CustomerID=Customers.CustomerID
```

Ejercicio 104: Generar un listado que presente los siguientes campos: OrderID, Nombre del empleado y OrderDate. La información requerida proviene de dos tablas Orders y Employees.

```
SELECT Orders.OrderID, Employees. FirstName + ' '  +
        Employees. LastName AS 'Nombre',  Orders.OrderDate
FROM Orders
INNER JOIN  Employees
ON Orders.EmployeeID=Employees.EmployeeID
```

Ejercicio 105: Generar un listado que presente los siguientes campos: ProductName, CategoryName, UnitPrice. La información requerida proviene de dos tablas Products y Categories.

```
SELECT Products.ProductName, Categories. CategoryName,
        Products.UnitPrice
FROM Products
```

```
INNER JOIN Categories
ON  Products.CategoryID=Categories.CategoryID
```

Ejercicio 107: Generar un listado que presente los siguientes campos: ProductName y CompanyName. La información requerida proviene de dos tablas Products y Suppliers.

```
SELECT Products.ProductName, Suppliers.CompanyName
FROM Products
INNER JOIN Suppliers
N Products.SupplierID=Suppliers.SupplierID
```

Ejercicio 108: Generar un listado que presente los siguientes campos: CategoryName, y el promedio de precios por categoría. La información requerida proviene de dos tablas Categories y Products.

```
SELECT Categories.CategoryName,AVG(Products.UnitPrice) AS 'Promedio'
FROM Categories
INNER JOIN Products
ON Categories.CategoryID=Products.CategoryID
GROUP BY Categories.CategoryName
```

Ejercicio 109: Consulta a tres tablas. Generar un listado que presente los siguientes campos: ProductName, CategoryName, CompanyName y UnitPrice. La información requerida proviene de tres tablas Products, Categories y Supplies.

```
SELECT Products.ProductName, Categories.CategoryName,
        Suppliers.CompanyName, Products.UnitPrice
FROM Products
INNER JOIN Categories
ON Products.CategoryID  =Categories.CategoryID
INNER JOIN Suppliers
ON Products.SupplierID=Suppliers.SupplierID
```

Ejercicio 110: En PUBS se solicita presentar una lista de libros con los siguientes datos:

Nombre del Libro, categoría del libro, Nombre de la editorial,

País de origen y precio del libro.

```
USE pubs
GO
SELECT T.title AS Libro, T.type AS Categoria, P.pub_name AS Editorial,
        P.country as [Pais de origen], T.price AS [Precio del libro]
FROM        publishers  P
INNER JOIN  titles      T   ON P.pub_id = T.pub_id
GO
```

Ejercicio 111: Presentar el total de gastos de flete por compañía transportista en Northwind

```
USE Northwind
GO
-- Para averiguar cómo se llama la FK en ORDERS, aplicamos el SP:
SP_HELPCONSTRAINT ORDERS
GO
SELECT S.CompanyName   AS [Compañia transportista] ,
        SUM(O.Freight)     AS [Total de gastos de flete]
FROM        Shippers   S
INNER JOIN  Orders     O   ON     S.ShipperID = O.ShipVia
GROUP BY S.CompanyName
ORDER BY 2 DESC
GO
```

Ejercicio 112: En Edutec se solicita información de las matrículas de los alumnos,

Presentando los siguientes datos:

Apellidos y nombres de los alumnos,

Curso programado en que se matriculo,

El código del profesor que dicto el curso y

El promedio obtenido

```
USE EDUTEC
GO
SELECT      A.ApeAlumno, A.NomAlumno, M.IdCursoProg, CP.IdProfesor,
M.Promedio
FROM        Alumno              A
INNER JOIN  Matricula           M   ON   A.IdAlumno = M.IdAlumno
INNER JOIN  CursoProgramado     CP  ON       M.IdCursoProg = CP.IdCursoProg
GO
SELECT A.ApeAlumno + ',  ' + A.NomAlumno AS Alumno,
      M.IdCursoProg  AS [Curso programamdo],
      CP.IdProfesor   AS  [Codigo del profesor] ,   M.Promedio
FROM        Alumno              A
INNER JOIN  Matricula           M   ON   A.IdAlumno = M.IdAlumno
INNER JOIN  CursoProgramado     CP  ON       M.IdCursoProg = CP.IdCursoProg
GO
```

Ejercicio 113: En MarketPeru se solicita una lista de productos mostrando Nombre del producto,

Nombre del proveedor, ubicación y origen del proveedor, nombre de la categoría,

Precio y existencia, para las categorías consideradas de consumo humano */

```
USE MarketPERU
      GO
      -- Averiguando las Categorias existentes
      SELECT * FROM CATEGORIA
```

```
GO
SELECT PD.Nombre AS Producto, P.Nombre AS Proveedor,
P.Departamento + ' - ' + P.Direccion AS [ubicacion y origen] , C.Categoria,
PD.PrecioProveedor AS Precio, PD.StockActual AS Existencia
FROM        PROVEEDORP
INNER JOIN  PRODUCTO  PD    ON    P.IdProveedor = PD.IdProveedor
INNER JOIN  CATEGORIA C     ON    PD.IdCategoria = C.IdCategoria
WHERE
C.Categoria IN ('GOLOSINAS','EMBUTIDOS','LACTEOS','LICORES Y GASEOSAS')
GO
```

Ejercicio 114: En Pubs se solicita presentar los tres autores con mayor número de ejemplares vendidos.

```
USE pubs
GO
SELECT TOP 3 WITH TIES A.au_lname, A.au_fname,
SUM(S.qty) AS [ejemplares vendidos]
FROM        authors       A
INNER JOIN  titleauthor   TA    ON    A.au_id = TA.au_id
INNER JOIN  titles        T     ON    TA.title_id = T.title_id
INNER JOIN  sales         S     ON    T.title_id = S.title_id
GROUP BY A.au_lname, A.au_fname
ORDER BY 3 DESC
GO
```

Ejercicio 115: Presentar una lista de matrículas, mostrado la siguiente información:

Apellidos y nombres del alumno, Código del curso programado, Nombre del curso,

Apellidos y nombres del Profesor y el Promedio obtenido.

```
USE EduTec
GO
SELECT A.ApeAlumno, A.NomAlumno,  CP.IdCursoProg, C.NomCurso,
P.ApeProfesor, P.NomProfesor, M.Promedio
FROM          Alumno              A
INNER JOIN    Matricula           M   ON      A.IdAlumno = M.IdAlumno
INNER JOIN    CursoProgramado     CP  ON      M.IdCursoProg = CP.IdCursoProg
INNER JOIN    Curso               C   ON      CP.IdCurso = C.IdCurso
INNER JOIN    Profesor            P   ON      CP.IdProfesor = P.IdProfesor
ORDER BY 3
GO
```

Ejercicio 115: Presentar el monto total de ventas por región en Northwind, durante el primer semestre de 1997

```
USE Northwind
GO
SELECT  R.RegionDescription,
        SUM(OD.Quantity * OD.UnitPrice)  AS [Monto de ventas por Region]
FROM          Region              R
INNER JOIN    Territories         T     ON     R.RegionID = T.RegionID
INNER JOIN    EmployeeTerritories ET    ON     T.TerritoryID = ET.TerritoryID
INNER JOIN    Employees           E     ON    ET.EmployeeID = E.EmployeeID
INNER JOIN    Orders              O     ON     E.EmployeeID = O.EmployeeID
INNER JOIN    [Order Details]     OD    ON     O.OrderID = OD.OrderID
```

```
WHERE O.OrderDate BETWEEN '19970101'  AND '19970630'
GROUP BY R.RegionDescription
ORDER BY 2 DESC
GO
```

Ejercicio 116: Presentar el monto total de ventas de las regiones Norte y Sur en Northwind, durante el primer semestre de 1997 (Northern, Southern)

```
SELECT R.RegionDescription,
SUM(OD.Quantity * OD.UnitPrice)  AS [Monto de ventas por Region]
FROM          Region               R
INNER JOIN    Territories          T     ON    R.RegionID = T.RegionID
INNER JOIN    EmployeeTerritories  ET    ON    T.TerritoryID = ET.TerritoryID
INNER JOIN    Employees            E     ON    T.EmployeeID = E.EmployeeID
INNER JOIN    Orders               O     ON    E.EmployeeID = O.EmployeeID
INNER JOIN    [Order Details       OD    ON    O.OrderID = OD.OrderID
WHERE O.OrderDate BETWEEN '19970101'  AND '19970630'
GROUP BY R.RegionDescription
HAVING  R.RegionDescription IN ('Northern', 'Southern')
ORDER BY 2 DESC
GO
```

OUTER JOIN

Un OUTER JOIN es la consulta correlacionada que entrega todas las filas que están relacionadas, y además:
Las filas no relacionadas de la tabla izquierda (LEFT OUTER JOIN), ó
Las filas no relacionadas de la tabla derecha (RIGHT OUTER JOIN), ó
Las filas no relacionadas de ambas tablas (FULL OUTER JOIN)
Se considera como la tabla izquierda, a aquella que se menciona primero en la cláusula FROM.

Ejercicio 117: Uso de LEFT OUTER JOIN

```
SELECT Orders.OrderID,Customers.CompanyName
FROM Orders
LEFT OUTER  JOIN Customers
ON Orders.CustomerID=Customers.CustomerID—830 Filas
```

Ejercicio 118: Uso de RIGHT OUTER JOIN

```
SELECT Orders.OrderID,Customers.CompanyName
FROM Orders
RIGHT OUTER  JOIN Customers
ON Orders.CustomerID=Customers.CustomerID—832 filas
```

Ejercicio 119: Uso de FULL OUTER JOIN

```
SELECT Orders.OrderID,Customers.CompanyName
FROM Orders
FULL OUTER  JOIN Customers
ON Orders.CustomerID=Customers.CustomerID—832 filas
```

CROSS JOIN

Un CROSS JOIN es la consulta correlacionada que combina cada una de las filas de una de las tablas con todas las filas de la otra tabla.

No es necesario que exista una columna común para ejecutar cross join. Esta consulta también se conoce como el **producto cartesiano** de dos tablas.

Ejercicio 120: Uso de CROSS JOIN

```
SELECT Orders.OrderID,Customers.CompanyName
FROM Orders
CROSS  JOIN Customers –75530 filas
```

Enviando El Resultado De Una Consulta A Una Tabla

Ejercicio 121: Enviando los resultados a una tabla Temporal (#)

```
USE EduTec
GO
SELECT A.IdAlumno,A.ApeAlumno,A.NomAlumno,
CONVERT(CHAR(12),M.FecMatricula,103) AS 'Fecha de Matricula',
CP.IdCurso,CP.IdCiclo
FROM            Alumno          A
INNER JOIN  Matricula       M   ON   A.IdAlumno=M.IdAlumno
INNER JOIN  CursoProgramado CP  ON   M.IdcursoProg=CP.IdcursoProg
GO
SELECT A.IdAlumno,A.ApeAlumno,A.NomAlumno,
CONVERT(CHAR(12),M.FecMatricula,103) AS 'Fecha de Matricula',
CP.IdCurso,CP.IdCiclo
INTO #TabTempo1
FROM            Alumno          A
INNER JOIN  Matricula       M   ON   A.IdAlumno=M.IdAlumno
INNER JOIN  CursoProgramado CP  ON   M.IdcursoProg=CP.IdcursoProg
GO

SELECT * FROM #TabTempo1
GO
DROP TABLE #TabTempo1
GO
SELECT * FROM #TabTempo1 ORDER BY IdAlumno
GO
```

Ejercicio 122:

```
USE MarketPeru
GO
SELECT P.IdProveedor, P.Nombre AS Proveedor,
CONVERT(CHAR(12),G.FechaSalida,103) AS 'Fecha de Salida',
PR.Nombre AS Producto, GD.IdGuia
INTO #ProductosEnviados
FROM          PROVEEDORP
INNER JOIN  PRODUCTO          PR    ON    P.IdProveedor = PR.IdProveedor
INNER JOIN  GUIA_DETALLE      GD  ON    PR.IdProducto = GD.IdProducto
INNER JOIN  GUIA              G    ON    GD.IDGuia = G.IDGuia
GO
--
SELECT  *  FROM  #ProductosEnviados
GO
SELECT * FROM #ProductosEnviados ORDER BY IdProveedor
GO
DROP TABLE #ProductosEnviados
GO
```

Enviando los resultados a una tabla permanente

Ejercicio 123:

```
USE EduTec
GO
SELECT A.IdAlumno,A.ApeAlumno,A.NomAlumno,
        CONVERT(CHAR(12),M.FecMatricula,103) AS 'Fecha de Matricula',
        CP.IdCurso,CP.IdCiclo
INTO   TablaEjemplo1
```

```
FROM        Alumno              A
INNER JOIN  Matricula           M     ON    A.IdAlumno=M.IdAlumno
INNER JOIN  CursoProgramado     CP    ON    M.IdcursoProg=CP.IdcursoProg
GO
SELECT * FROM TablaEjemplo1 ORDER BY IdAlumno
GO
--
USE MarketPeru
GO
SELECT P.Nombre,OD.CantidadRecibida,
        CONVERT(CHAR(12),O.FechaEntrada,103) AS 'Fecha de Ingreso'
INTO  IngresoDeProductos
FROM        Producto            P
INNER JOIN  Orden_Detalle       OD    ON      P.IdProducto = OD.IdProducto
INNER JOIN  Orden               O     ON      OD.IdOrden = O.IdOrden
GO
SELECT * FROM IngresoDeProductos ORDER BY Nombre
GO
--
/* Presentar el total de ventas por región por año, en Northwind
y guardar los resultados en tablas históricas
*/
USE Northwind
GO
SELECT
        R.RegionDescription,
        SUM(OD.Quantity*OD.UnitPrice) AS MONTO
INTO Ventas_1996
```

```
FROM            Region              R
INNER JOIN      Territories         T     ON    R.RegionID=T.RegionID
INNER JOIN      EmployeeTerritories ET    ON    T.TerritoryID=ET.TerritoryID
INNER JOIN      Employees           E     ON    ET.EmployeeID=E.EmployeeID
INNER JOIN      Orders              O     ON    E.EmployeeID=O.EmployeeID
INNER JOIN      [Order Details]     OD    ON    O.OrderID=OD.OrderID
WHERE YEAR(O.OrderDate)=1996
GROUP BY R.RegionDescription
ORDER BY 2 DESC
GO
select * from Ventas_1996
go
SELECT
        R.RegionDescription,
        SUM(OD.Quantity*OD.UnitPrice) AS MONTO
INTO Ventas_1997
FROM            Region              R
INNER JOIN      Territories         T     ON    R.RegionID=T.RegionID
INNER JOIN      EmployeeTerritories ET    ON    T.TerritoryID=ET.TerritoryID
INNER JOIN      Employees           E     ON    ET.EmployeeID=E.EmployeeID
INNER JOIN      Orders              O     ON    E.EmployeeID=O.EmployeeID
INNER JOIN      [Order Details]     OD    ON    O.OrderID=OD.OrderID
WHERE YEAR(O.OrderDate)=1997
GROUP BY R.RegionDescription
ORDER BY 2 DESC
GO
select * from Ventas_1996
select * from Ventas_1997
```

```
go
SELECT
        R.RegionDescription,
        SUM(OD.Quantity*OD.UnitPrice) AS MONTO
INTO Ventas_1998
FROM                    Region                          R
INNER JOIN      Territories             T    ON      R.RegionID=T.RegionID
INNER JOIN      EmployeeTerritories ET   ON      T.TerritoryID=ET.TerritoryID
INNER JOIN      Employees       E     ON      ET.EmployeeID=E.EmployeeID
INNER JOIN      Orders          O     ON      E.EmployeeID=O.EmployeeID
INNER JOIN      [Order Details]     OD  ON      O.OrderID=OD.OrderID
WHERE YEAR(O.OrderDate)=1998
GROUP BY R.RegionDescription
ORDER BY 2 DESC
GO
select * from Ventas_1996
select * from Ventas_1997
select * from Ventas_1998
go
```

Subconsultas

Un subconsulta es una declaración SELECT anidada dentro una sentencia SELECT, INSERT, UPDATE o DELETE o dentro de otra subconsulta.

Si la respuesta a un requerimiento de datos requiere la ejecución de una serie de pasos lógicos, utilice subconsultas para tratar de resolver el requerimiento con una sola sentencia.

Las subconsultas son de los tipos siguientes:

- Subconsulta que entrega un solo valor (1 fila, 1 columna)
- Subconsulta que entrega un conjunto de valores (varias filas, 1 columna)

Una subconsulta se especifica entre paréntesis, y se puede especificar en cualquier donde la Sintaxis permite una expresión.

Cuando la subconsulta se especifica:

- en la lista de columnas del SELECT externo, ó
- en la cláusula WHERE del SELECT externo usando un operador relacional (test de comparación),

La subconsulta debe ser una que entregue un solo valor.

Ejercicio 105: Genere un listado que muestre el nombre y el precio de los productos con precio igual al producto más caro.

```
USE Northwind
GO
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice=(SELECT MAX(UnitPrice) FROM Products)
```

Ejercicio 106: Mostrar un listado con las siguientes columnas: ProductID, ProductName, y UnitPrice de los productos que abastece la empresa 'Tokyo Traders'

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
WHERE SupplierID =(SELECT SupplierID FROM Suppliers
                                    WHERE CompanyName='Tokyo Traders')
```

ProductID	ProductName	UnitPrice
9	Mishi Kobe Niku	97,00
10	Ikura	31,00
74	Longlife Tofu	10,00

Ejercicio 107: Genere la lista de los clientes que no se han registrado una orden. Mostrar el código y el nombre del Cliente.

```
SELECT Customers.CustomerID , Customers.CompanyName
FROM Customers
WHERE  NOT EXISTS (SELECT * FROM Orders
```

```
WHERE Customers.CustomerID=Orders.CustomerID)
```

FISSA FISSA Fabrica Inter. Salchichas S.A.

PARIS Paris spécialités

Ejercicios adicionales

En Northwind, presentar una lista de productos con el precio máximo

```
USE NORTHWIND
GO
SELECT MAX(UNITPRICE) AS COSTO FROM PRODUCTS
GO
SELECT ProductName, UnitPrice FROM PRODUCTS
    WHERE UnitPrice = (SELECT MAX(UNITPRICE) FROM PRODUCTS)
GO
```

En MarketPeru Presentar una lista de productos cuyos precios sean inferiores al precio promedio

```
USE MarketPERU
GO
SELECT  AVG(PrecioProveedor) FROM PRODUCTO
GO
SELECT Nombre,PrecioProveedor,
(SELECT  AVG(PrecioProveedor) FROM PRODUCTO) AS Promedio
FROM PRODUCTO
WHERE
PrecioProveedor < (SELECT  AVG(PrecioProveedor) FROM PRODUCTO)
ORDER BY 2 DESC
GO
```

USO DE CONVERT

```
USE EduTec
GO
```

```
SELECT GETDATE()AS [FECHA ACTUAL]
GO
SELECT
CONVERT(CHAR(12),GETDATE(),101) AS '101(USA)mm/dd/yy',
CONVERT(CHAR(12),GETDATE(),102) AS '102(ANSI)yy.mm.dd',
CONVERT(CHAR(12),GETDATE(),103) AS '103(BRITANICO/FRANCES)dd/mm/yy',
CONVERT(CHAR(12),GETDATE(),104) AS '104(ALEMAN)dd.mm.yy',
CONVERT(CHAR(12),GETDATE(),105) AS '105(ITALIANO)dd-mm-yy'
GO
SELECT A.ApeAlumno, A.NomAlumno,
 CONVERT(CHAR(12),M.FecMatricula,103) AS [Fecha de Matricula],
CP.IdCurso,CP.IdCiclo
FROM        Alumno              A
INNER JOIN  Matricula           M    ON    A.IdAlumno=M.IdAlumno
INNER JOIN  CursoProgramado     CP   ON    M.IdcursoProg=CP.IdcursoProg
GO
```

7

Vistas

Una vista es un objeto que almacena una consulta predefinida y que proporciona un modo alternativo de visualización de datos sin tener que redefinir la consulta. Las tablas requeridas en una vista se llaman tablas base. Con algunas excepciones, cualquier declaración SELECT puede nombrarse y guardarse como una vista. Los ejemplos comunes de vistas incluyen:

- Un subconjunto de filas o columnas de una tabla base.
- Una unión de dos o más tablas base.
- Un join de dos o más tablas base.
- Un resumen estadístico de una tabla base.
- Un subconjunto de otra vista, o alguna combinación de vistas y tablas base.

Ventajas de las vistas

El uso de las vistas proporciona las siguientes ventajas:

- El usuario accede a la data importante o apropiada para él. Limita el acceso a datos sensibles.
- Oculta la complejidad del modelo de datos. Un join de múltiples tablas se convierte en un simple SELECT para el usuario.
- Desde el punto de vista del usuario, una vista es una "tabla" pues puede ejecutar en ella todas las operaciones de datos: SELECT, INSERT, UPDATE y DELETE.
- Debido a que una vista es un objeto de la base de datos, puede asignarle permisos de usuario. Esto es mucho más eficiente que colocar los mismos permisos sobre columnas individuales en una tabla.

Sintaxis

CREATE VIEW [< *database_name* > **.**] [< *owner* > **.**] *view_name* [**(** *column* [**,**...*n*] **)**]
[WITH < view_attribute > [**,**...n]]
AS
select_statement
[WITH CHECK OPTION]

< view_attribute > ::=
{ ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

Ejemplos de vistas

Ejercicio 108: Creación de una vista sencilla

```
USE pubs
GO
CREATE VIEW titles_view
AS
SELECT title, type, price, pubdate
FROM titles
GO
```

Utilizar la vista

```
SELECT * FROM titles_view
```

Ejercicio 109: Crear una vista que muestre la categoría y el promedio de precios de la tabla Titles

```
USE pubs
GO
CREATE VIEW categories (category, average_price)
AS
SELECT type, AVG(price)
FROM titles
GROUP BY type
```

Ejercicio 110: Crear una vista que muestre un reporte del producto más vendido por cada categoría

```
CREATE VIEW v_VentasCategoriaProducto
AS
SELECT C.CategoryName,P.ProductName,
SUM(OD.quantity* OD.unitPrice*(1-OD.Discount)) AS Monto
FROM [Order Details] AS OD
INNER JOIN Products AS P ON OD.ProductID=P.ProductID
INNER JOIN Categories AS C ON P.CategoryID=C.CategoryID
GROUP BY C.CategoryNAme,P.ProductName
GO
```

Utilizar la vista v_VentasCategoriaProducto

```
SELECT * FROM v_VentasCategoriaProducto
ORDER BY 1,3 DESC
```

Ejercicio 111: Consulta que para cada categoria extrae el producto con más ventas desde la vista.

```
SELECT CAtegories.CategoryNAme,
(SELECT TOP 1 ProductName FROM v_VentasCategoriaProducto
WHERE CAtegories.CategoryNAme=v_VentasCategoriaProducto.CategoryNAme
ORDER BY Monto DESC) AS PRODUCTO,
(SELECT TOP 1 Monto FROM v_VentasCategoriaProducto
WHERE CAtegories.CategoryNAme=v_VentasCategoriaProducto.CategoryNAme
ORDER BY Monto DESC) AS Monto
FROM Categories
```

Ejercicios adicionales:

```
USE PUBS
GO
CREATE VIEW V_EDITORLIBRO
AS
SELECT P.PUB_NAME,P.COUNTRY,T.TITLE,T.PRICE
FROM PUBLISHERS P INNER JOIN TITLES T        ON  P.PUB_ID=T.PUB_ID
GO
SELECT * FROM V_EDITORLIBRO
GO
SELECT * FROM V_EDITORLIBRO ORDER BY PUB_NAME
GO
```

Si se alteran los datos en una vista los datos se actualizan en las tablas orígenes de datos e las vistas.

Vamos a cambiar el nombre de Algodata Infosystems por 'Algodata InfosystemsXXX'

```
UPDATE V_EDITORLIBRO SET pub_name= 'Algodata InfosystemsXXX'
WHERE pub_name ='Algodata Infosystems'
GO
SELECT * FROM V_EDITORLIBRO
GO
```

Verificando los cambios en la tabla origen de datos de la Vista:

```
SELECT * FROM publishers
GO
UPDATE publishers SET pub_name= 'Algodata Infosystems'
WHERE pub_name ='Algodata InfosystemsXXX'
GO
SELECT * FROM publishers
GO
```

```
SELECT * FROM V_EDITORLIBRO

GO
```

Vistas anidadas

```
USE Northwind

GO

CREATE VIEW V_CLIENTESSUDAMERICA

AS

SELECT * FROM CUSTOMERS

WHERE COUNTRY IN ('BRAZIL','ARGENTINA','VENEZUELA')

GO

--

SELECT * FROM V_CLIENTESSUDAMERICA ORDER BY COUNTRY

GO
```

Ahora vamos a crear otra Vista, que tiene como origen de datos a la vista anterior V_CLIENTESSUDAMERICA

```
CREATE VIEW V_CLISUDAMER2

AS

SELECT COMPANYNAME,COUNTRY,CITY,ADDRESS

FROM V_CLIENTESSUDAMERICA

GO

--

SELECT * FROM V_CLISUDAMER2

GO
```

En conclusión, se pueden generar vistas dentro de otras vistas. SQL Server permite anidar vistas. El anidamiento no debe superar los 32 niveles.

Es posible que el límite real del anidamiento de vistas sea inferior en función de la complejidad de la vista y de la memoria disponible.