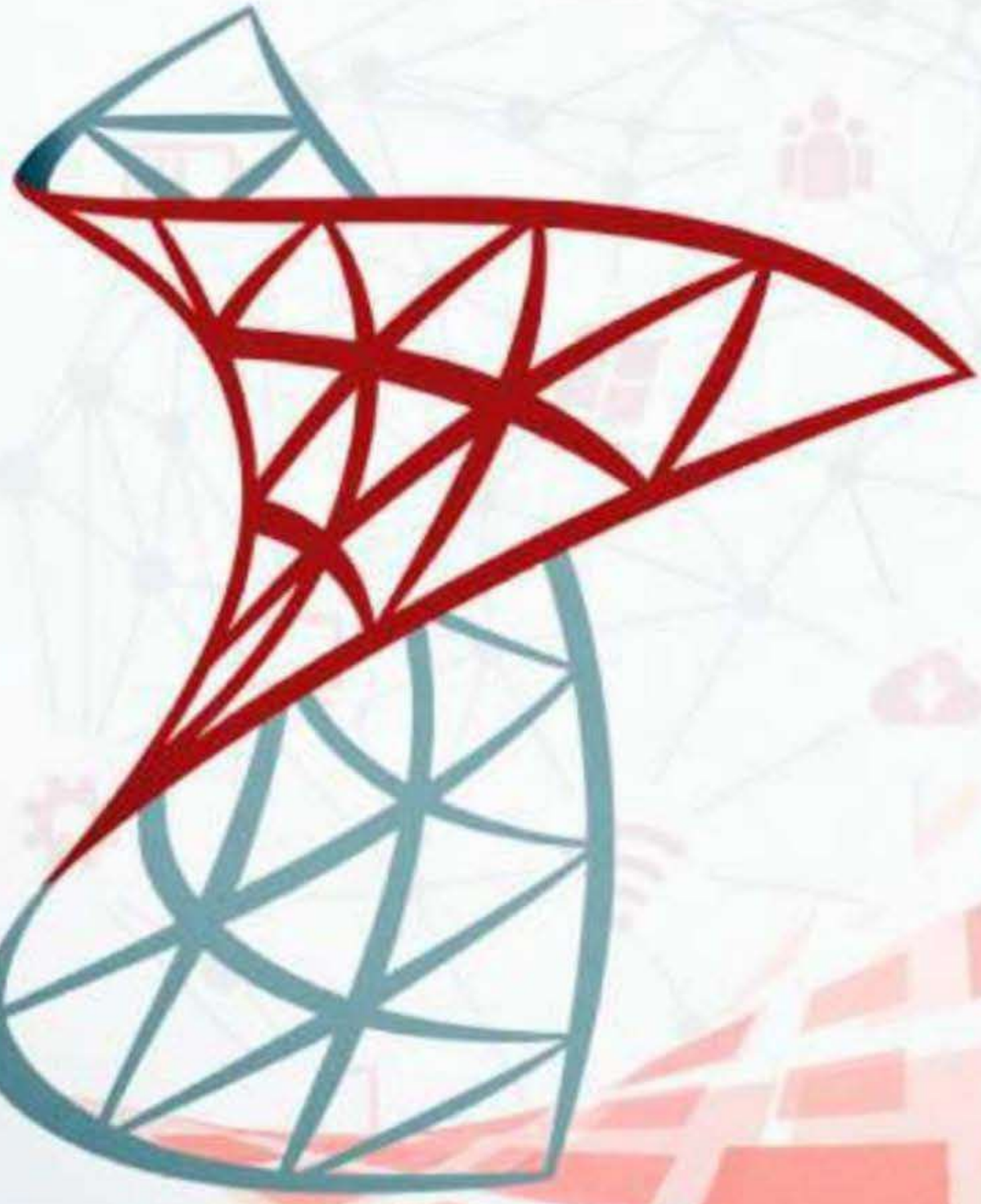
EDITORIAL MACRO

Manuel Torres Remon

// PROGRAMACIÓN TRANSACT CON SQL SERVER 2016



Esta publicación incluye:
Material de descarga

// PROGRAMACIÓN TRANSACT CON SQL SERVER 2016

MANUEL TORRES REMON

EDITORIAL MACRO®

Perú - México - Colombia - Chile - Ecuador - España - Bolivia - Uruguay - Guatemala - Costa Rica

EDITORIAL MACRO®

Programación Transact con SQL Server 2016
Autor: Manuel Torres Remon

© Derechos de autor registrados:
Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:
Empresa Editora Macro EIRL

Jefe de edición:
Magaly Ramon Quiroz

Diseño de portada:
Fernando Cavassa Repetto

Corrección de estilo:
Dina Dueñas Calderón

Diseño y diagramación:
Eduardo Siesquén Aquije

Edición a cargo de:
© Empresa Editora Macro EIRL
Av. Paseo de la República N.° 5613, Miraflores, Lima, Perú

Teléfono: (511) 748 0560
E-mail: proyectoeditorial@editorialmacro.com
Página web: www.editorialmacro.com

Primera edición: Agosto 2017
Tiraje: 1750 ejemplares

Impresión:
Talleres gráficos de la Empresa Editora Macro EIRL
Jr. San Agustín N.° 612-624, Surquillo, Lima, Perú
Agosto 2017

ISBN N.° 978-612-304-547-0
Hecho el Depósito Legal en la Biblioteca Nacional del Perú N.° 2017-10274

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.

Manuel Torres Remon

Licenciado en Informática. Se ha dedicado a la consultoría y docencia de cursos de tecnología desde hace 16 años. A su vez, ha brindado diversos talleres de capacitación en las instituciones más importantes de Lima (Perú).

Realizó sus estudios en el Instituto de Educación Superior Manuel Arévalo Cáceres, en la Universidad Alas Peruanas y en la Universidad Peruana del Norte. En todas estas instituciones recibió una esmerada formación profesional que ha podido demostrar ampliamente a lo largo de su trayectoria docente.

Actualmente, se desempeña como docente de tecnología en instituciones de educación superior como Cibertec, Manuel Arévalo Cáceres y UNIMASTER de la Universidad Nacional de Ingeniería (UNI); en todas ellas imparte los cursos de Programación, Base de Datos y Análisis de Sistemas.

Dedicatoria

Este libro está dedicado con mucho cariño y aprecio a mis pequeñas Ángela Victoria y Fernanda Ximena, quienes son y seguirán siendo mi fuente de inspiración, y mucho más a mi esposa Luz por comprenderme en todo lo que me propongo.

Agradecimientos

Todo libro escrito es un sacrificio que tiene que ser valorado y esto solo se dará si usted aprecia este material tal como lo aprecio yo mismo. Escuché, alguna vez, la siguiente frase: "La lectura de un libro enriquece de conocimientos y empobrece la ignorancia"; pienso que es cierto, aunque hoy un libro impreso es difícil de leer, pues son pocas las personas que cargan uno en la mano, ya que han sido reemplazados por los dispositivos móviles. Por eso, mi primer agradecimiento es para usted, amigo lector, por adquirir este material, que ha significado tiempo y esfuerzo para mí. Además, si tuviera alguna consulta o comentario sobre él, me gustaría invitarle a escribirme por *e-mail* a manuel.torresr@hotmail.com.

Asimismo, deseo agradecer a la gran familia MACRO por confiar nuevamente en mi persona para el desarrollo de este libro: *Programación Transact con SQL Server 2016.*

Índice

Introducción 17

■ Capítulo 1: Introducción a la base de datos 19

1.1 Introducción 21
1.2 Información 22
1.2.1 Concepto 22
1.2.2 Características 23
1.3 Base de datos 23
1.3.1 Historia 23
1.3.2 Definición 25
1.3.3 Objetivos 25
1.4 Sistema de base de datos 26
1.5 Usuarios 27
1.5.1 DBA (Administrador de base de datos) 27
1.5.2 Diseñador de base de datos 27
1.5.3 Usuario final 28
1.6 Representación de la información 28
1.6.1 Representación de la información desde el mundo real 29
1.6.2 Representación de la información desde el dominio de las ideas 30
1.6.3 Representación de la información desde los datos 31
1.7 Modelo Entidad-Relación 31
1.7.1 Entidades 32
1.7.2 Atributos 33
1.7.3 Relaciones de correspondencia 35
1.7.4 Generalización 38
1.7.5 Agregación 40
▸ Caso desarrollado: "SALES S. A." 41

■ Capítulo 2: Gestión de base de datos 53

2.1 SQL Server 2016 55
2.2 Características de SQL Server 2016 55
2.2.1 Tareas críticas 55
2.2.2 Seguridad y rendimiento 56
2.2.3 Inteligencia empresarial 56
2.2.4 Análisis de base de datos 56
2.3 Ediciones de SQL Server 2016 57
2.3.1 Enterprise 57
2.3.2 Estándar 57
2.3.3 Developer 57
2.3.4 Web 57
2.3.5 Express 57
2.4 Historial de revisiones de SQL Server 57
2.5 Instalación de SQL Server 2016 58
2.6 SQL Management Studio 63
2.6.1 Pantalla inicial del entorno 65
2.6.2 Configuración de fuente para el entorno de trabajo 67
2.7 Las bases de datos en SQL Server 69
2.8 Componentes de una base de datos 69
2.8.1 Archivo MDF 70
2.8.2 Archivo NDF 70
2.8.3 Archivo LDF 70
2.9 Motor de base de datos 70
2.10 Sentencias DDL 71
2.10.1 Create 71
2.10.2 Alter 71
2.10.3 Drop 71
2.11 Sentencias DML 71
2.11.1 Insert 72
2.11.2 Update 72
2.11.3 Delete 72
2.11.4 Select 72
2.12 Creación de una base de datos 72
2.13 Modificación de archivos de una base de datos 74
2.14 Activar la base de datos 75

2.15 Los tipos de datos en SQL Server 2016 75
2.16 Tipos de datos definidos por el usuario 77
- Caso desarrollado 1: Base de datos básica 79
- Caso desarrollado 2: Base de datos con especificación de archivo primario 80
- Caso desarrollado 3: Base de datos con especificación de archivos secundarios 81
- Caso desarrollado 4: Base de datos con especificación de archivos de transacciones 82
- Caso desarrollado 5: Añadir archivos a la base de datos 83
- Caso desarrollado 6: Modificar propiedades de la base de datos 84
- Caso desarrollado 7: Eliminar archivos de la base de datos 84
- Caso desarrollado 8: Integración 85

Capítulo 3: Gestión de tablas 87

3.1 Sentencias DDL para la gestión de tablas 89
3.1.1 Creación de una tabla 89
3.1.2 Opciones de la tabla 90
3.1.3 Modificación de una tabla 92
3.1.4 Eliminación de una tabla 93
3.2 Restricciones de valores nulos y no nulos 94
3.3 Restricciones de integridad referencial 95
3.3.1 Llave primaria 95
3.3.2 Llave secundaria 97
3.4 Manejo de esquemas 99
3.5 Restricciones para datos 103
3.5.1 Identity 103
3.5.2 Default 104
3.5.3 Check 106
3.5.4 Unique 108
3.6 Sentencias DML para datos 110
3.6.1 Insert 110
3.6.2 Update 113
3.6.3 Delete 117
3.7 Manejo de datos masivos en SQL Server 119
3.7.1 Instrucción Bulk Insert 119
3.8 Sentencia Merge 122
- Caso desarrollado 1 123
- Caso desarrollado 2 130

■ Capítulo 4: Recuperación de datos 135

4.1 Sentencia Select para la recuperación de registros 137
4.2 Consultas de registros básico 138
4.3 Consultas distinguidas 139
4.4 Consultas ordenadas 140
4.5 Consultas por cantidad de registros 142
4.6 Consultas con especificación de campos 143
4.7 Consultas con cabeceras 143
4.8 Consultas con campos calculados 145
4.9 Consultas que crean tablas de registros 146
4.10 Consultas condicionadas 148
4.11 Consultas internas 151
4.12 Consultas externas 153
4.12.1 Left Join 154
4.12.2 Right Join 155
4.12.3 Full Join 156
4.13 Consultas agrupadas 157
4.13.1 Funciones agregadas 158
4.13.2 Cláusula Group by 160
4.13.3 Cláusula Group by con resúmenes 164
4.14 Subconsultas 167
4.15 Vistas 168
4.15.1 Ventajas de las vistas 169
4.15.2 Creación de las vistas 169
4.15.3 Mantenimiento de datos mediante vistas 174
4.15.4 Simplificación de consultas complejas mediante el uso de vistas 175
▸ Caso desarrollado 177

■ Capítulo 5: Transact SQL 191

5.1 Introducción 193
5.2 Fundamentos de programación Transact SQL (T-SQL) 193
5.2.1 Variables e identificadores 193
5.2.2 Funciones Cast y Convert 197
5.3 Estructuras de control 199
5.3.1 Estructura selectiva IF 199
5.3.2 Estructura condicional múltiple CASE 203
5.3.3 Estructura de control While 207

5.4 Control de errores en Transact SQL 210
5.5 Función @@error 213
5.6 Función Raiserror 215
▸ Caso desarrollado 216

■ Capítulo 6: Procedimientos almacenados 223
6.1 Procedimientos almacenados 225
6.2 Tipos de procedimientos almacenados 225
6.3 Procedimientos almacenados del sistema 225
6.4 Instrucción Execute y SP_ExecuteSQL 228
6.5 Procedimientos almacenados definidos por el usuario 228
6.6 Opciones sobre los procedimientos almacenados 235
6.7 Transacciones en Transact SQL 236
6.8 Begin Transaction 237
▸ Caso desarrollado 239

■ Capítulo 7: Funciones 245
7.1 Funciones 247
7.2 Funciones del sistema 247
7.2.1 Funciones de conjuntos de filas 247
7.2.2 Funciones de agregado 248
7.2.3 Funciones de categoría 253
7.2.4 Funciones escalares 257
7.3 Funciones definidas por el usuario 277
7.3.1 Funciones escalares 277
7.3.2 Funciones tabla en línea 282
7.3.3 Funciones tabla multisentencia 284
▸ Caso desarrollado 286

■ Capítulo 8: Cursores 297
8.1 Cursores 299
8.2 Proceso de implementación de un cursor 299
8.3 Tipos de cursores 300
8.4 Declaración de cursor 300

■ Capítulo 9: Triggers 309

9.1 Triggers 311
9.2 Tipos de triggers 311
9.3 Opciones de triggers 312

■ Capítulo 10: XML con SQL e imágenes 321

10.1 Introducción 323
10.2 Modelo de datos relacionales o XML 323
10.3 Ventajas de almacenar valores en XML 323
10.4 Elección de la tecnología XML 324
10.5 Tipo de datos XML 327
10.6 Columnas y variables XML 328
10.7 FOR XML y OPENXML 332
10.8 Instrucción OpenRowSet 340

Referencias bibliográficas 351

Introducción

Server 2016 es un sistema integrado de gestión de base de datos. Hoy en día las organizaciones necesitan tener un control automatizado de sus archivos, es decir, una simple factura no necesariamente tendrá que ser física, en la actualidad se la puede enviar por *e-mail*. Por ejemplo, en Perú, mediante las leyes 26612 y 681, se aprueba el uso de las imágenes como medio de sustento legal, considerándose las mismas con el mismo valor legal que el documento original; con esto se puede dejar de lado la papelería para pasar a los archivos digitales. SQL puede tener el control de sus archivos administrándolos de manera eficaz, rápida y segura.

Programación Transact SQL propone realizar procesos de manera profesional por medio de scripts que se ejecutarán tanto en el cliente como en el servidor y que será de gran utilidad poder dominarlos. Los lenguajes de programación siembran la cultura de la programación nativa, mientras que SQL Server propone instrucciones o sentencias para la obtención de resultados. Transact SQL rompe ese esquema y le quita un poco de protagonismo a los lenguajes de programación usando sus estructuras, que son entendidas en el Motor de base de datos de SQL Server 2016.

El libro de *Programación Transact con SQL Server 2016* le permitirá conocer todas las sentencias que se necesitan para controlar la información dentro de una organización. Para cumplir con dicho objetivo, se ha dividido el libro en diez capítulos que son distribuidos de la siguiente manera.

En el primer capítulo se hace un estudio sobre los principales conceptos de las bases de datos, lo que permitirá obtener conocimientos básicos acerca del manejo de la información.

En el capítulo dos se presentan los componentes de una base de datos y, además, cómo pueden administrarse sus archivos. Asimismo, se conocerá el grupo de sentencias que se usarán: sentencias DML y DDL.

En el capítulo tres se desarrollan todas las sentencias que permitirán administrar de la mejor manera las tablas de una base de datos; además, se aprenderá a implementar restricciones a las tablas de una base de datos haciendo que la integridad de los datos sea la correcta al momento de ingresar valores a la base de datos.

El capítulo cuatro explica cómo se recupera la información a partir de datos almacenados en la base de datos; asimismo, muestra muchas técnicas para la recuperación de la información usando la integración de cláusulas dentro de la sentencia SELECT.

El capítulo cinco desarrolla nuevas sentencias que se pueden implementar en SQL como la programación Transact SQL, la cual permitirá ampliar nuestras posibilidades de recuperación de información, así como implementar sentencias de manera profesional.

En el capítulo seis se explica cómo se implementan los procedimientos almacenados y se analiza la importancia de estos en el proceso del desarrollo de un *software*.

El capítulo siete describe cómo se implementan las funciones dentro de SQL y el uso que se le puede dar cuando se integra con todas las sentencias vistas hasta este momento.

El capítulo ocho explica cómo se implementan los cursores y cuál es la importancia que tienen cuando se usa la información almacenada en una base de datos como reportes.

En el capítulo nueve se aprende a implementar políticas o reglas de negocio por medio de los disparadores o triggers dentro de la base de datos; estos pueden implementarse tanto a los datos almacenados, así como a los objetos que contiene la base de datos como tablas o la misma base de datos.

Finalmente, en el capítulo diez se entiende cómo se maneja la tecnología XML integrándola a los datos almacenados; asimismo, cómo poder manejar imágenes y cuál es la mejor manera de administrarlas.



Introducción a la base de datos

Capítulo 1

Competencia tecnológica:

- Reconocer los diferentes conceptos que asocian las bases de datos con el mundo real; asimismo, diseñar un modelo Entidad-Relación a partir de un problema.

Contenido:

1.1 Introducción
1.2 Información
1.3 Base de datos
1.4 Sistema de base de datos
1.5 Usuarios
1.6 Representación de la información
1.7 Modelo Entidad-Relación
- Caso desarrollado: "SALES S. A."

1.1 Introducción

En la actualidad, la expresión *base de datos* es extensamente usada en todos los ámbitos donde se desarrolla el ser humano; no necesariamente es una expresión computacional porque hoy en día la base de datos es considerada como la única solución para el manejo de todo tipo de información, sea grande o pequeña.

Se puede decir que las bases de datos han logrado un nivel de usabilidad extremadamente amplio con solo mencionar que la información que se almacena es usada y generada por los seres humanos así como por las máquinas. Se calcula que cada día Google procesa cerca de 25 *petabytes*, un aproximado de un millón de *gigabytes* de datos; la red social Facebook comparte más de 10 millones de fotografías, y YouTube sube una hora de video cada segundo. Esto genera un problema de superpoblación de información; es decir, se tiene toda la información posible, pero, sin una buena administración, dicha información no serviría para nada. Al respecto, se puede decir que, si se quiere buscar un dato particular, la búsqueda resultará un tanto difícil ante tanta información. Es aquí donde entran las bases de datos, las cuales permiten tener la información de alguna manera administrada.



Figura 1.1 Base de datos
Fuente: <http://conceptodefinicion.de/wp-content/uploads/2014/10/basededatos.jpg>

En la actualidad, las bases de datos se encuentran informatizadas y presentan una ventaja en la administración del espacio que ocupa una base de datos física, pues la información almacenada en la base de datos se puede encontrar en un servidor local o en un servidor web como lo es la nube; en ambos casos se ofrece un nivel de seguridad de la información por usuario propia del sistema de gestión de base de datos. Asimismo, la información se encontrará disponible en cualquier momento, ya que podrá ser accesible desde cualquier dispositivo que posea internet.

1.2 Información

1.2.1 Concepto

Existen diferentes conceptos relacionados con la *información*; pero según la historia se puede decir que, desde la Edad Media, se tenía como concepto de la *información* 'el dar forma a la materia y a la mente'; el cual da significado a muchas de las cosas y, principalmente, a los modelos del pensamiento humano. Aquí podría radicar la diferencia que existe entre los diferentes entes que generan información, ya que el ser humano considera la *información* como 'el significado que tiene las cosas'.

A este respecto, se puede mencionar que la información está formada por un grupo de datos previamente ordenados y que estos permiten construir un mensaje con significado. En ese sentido, la información permite resolver problemas en todos los ámbitos y, si se habla en una organización, esta puede determinar la toma de decisiones que necesita la gerencia.

Por otro lado, en términos informáticos la *información* es 'un conjunto de datos estrictamente organizados', los cuales pueden ser procesados para construir nueva información a través de mensajes, instrucciones, operaciones o funciones; que serán usados en diferentes dispositivos como la computadora, sitios web o dispositivos móviles. La información presenta el siguiente ciclo de vida:



Figura 1.2 Ciclo de vida de la información en términos informáticos

El ciclo de vida de la información inicia con la **generación de la información**; la cual se dijo puede ser generada tanto por los seres humanos como por las mismas máquinas. La **selección** hace referencia a que toda la información generada puede ser seleccionada a conveniencia del usuario. La **representación** es el entendimiento que se le da a la información seleccionada y que podrá **almacenarse** como parte de un proceso. Una vez almacenada la información, esta se podrá **recuperar** de tal forma que permita una **distribución** filtrada y ordenada. Todos estos elementos que componen el ciclo de vida de la información trabajan en comunión para poder **usar** dicha información en un determinado tiempo.

Finalmente, para cualquier otro tipo de ciencia la *información* es considerada como 'un conjunto de elementos que dan significado y sentido a las cosas', ya sean objetos o entidades del mundo a través de códigos y modelos.

1.2.2 Características

La información resulta muy importante para el ser humano y la sociedad; actualmente, esta podría darse al redactar un documento, escribir información por redes sociales, editar imágenes fotográficas y publicarlas en Facebook, grabar un video y subirlo a YouTube. Entonces, se puede decir que toda información presenta las siguientes características:

- **Exacta:** La información no debe presentar variaciones; al contrario, debe ser ordenada y previamente almacenada para que sea considerada como exacta.
- **Objetiva:** La información debe tener una base de criterios previamente establecidos para que cualquier usuario manipule la información en cualquier circunstancia, obteniendo resultados óptimos.
- **Dirección:** Toda información tiene una dirección específica, como la información solicitada por una compañía de teléfonos o la información necesaria en un centro de salud.
- **Válida y veraz:** La información, cuando presenta criterios ordenados y uniformes, es considerada como válida y veraz; ya que mide, de forma precisa, el resultado del uso de la información.
- **Continua:** La información debe mantenerse estable durante el tiempo; aquí es posible mencionar que la información puede variar por diferentes medios, quizás por una modificación, pero siempre mantendrá la información a pesar de los cambios.
- **Oportuna:** Los usuarios podrán obtener la información en el momento que la necesiten, ya sea en forma textual o gráfica.

1.3 Base de datos

1.3.1 Historia

Desde el inicio de la vida en la Tierra se ha generado mucha información que ha sido almacenada en una base de datos, la cual hasta cierto tiempo se encontraba en las bibliotecas formando un gran registro de información. Por ejemplo, se tienen registros de las grandes culturas de América, desde su fecha de creación, historia, costumbres, hasta su fecha de desaparición. Hasta hace poco tiempo acceder a dicha información era un tanto complicado, pues solo se podía leer desde libros, pero con la ayuda de la internet se abrió un gran camino a la información, accediendo a las bases de datos de todo el mundo y compartiendo información.

Entonces, ante la necesidad de registrar grandes volúmenes de información generadas en el mundo, y la aparición paralela de las primeras computadoras, el concepto de *base de datos* casi siempre ha estado ligado a la informática, que la define como 'el tratamiento automático de la información'. Algunas fechas importantes en la historia de la base de datos son las siguientes:

Herman Hollerith (1884)

Ante la necesidad de automatizar el proceso de censo, ideó una máquina tabuladora, mejor conocida como censadora, la cual permitía realizar dicho proceso en menos tiempo, usando tarjetas perforadas. Por eso, es considerado como el primer ingeniero estadístico de la historia.

Década de 1950

Se da origen a las cintas magnéticas, que representan una evolución de las tarjetas perforadas, con la diferencia que permitían automatizar la información y generar respaldos de seguridad. Su principal desventaja era el acceso de forma secuencial de la información.

Década de 1960

Con la reducción del costo de las computadoras, las empresas ya podían adquirirlas; por lo tanto, el manejo de la información se realizaba mediante el uso de los discos ante el crecimiento del mismo. Con este tipo de manejo de la información se podía realizar una administración de los datos de forma directa y rápida. Una de las características más importantes de esta época fue el inicio de las primeras generaciones de bases de datos de red y las bases de datos jerárquicas usando para esto listas y árboles como estructura de datos.

Década de 1970

Edgar F. Codd, científico informático reconocido por sus aportaciones sobre las bases de datos relacionales, publica un artículo llamado "Un modelo relacional de datos para grandes bancos de datos compartidos", el cual propone un conjunto de reglas para las bases de datos relacionales que hasta hoy usamos.

A partir de las bases propuestas por Edgar F. Codd, es que Larry Ellison funda Oracle Corporation, en la cual desarrolla un sistema de gestión de bases de datos relacional con el mismo nombre de dicha compañía.

Década de 1980

Lo más resaltante de esta época es el desarrollo de SQL como un lenguaje de consulta de acceso a las bases de datos relacionales, el cual permitía manipular y consultar la información almacenada mediante sentencias simples y rápidas de entender; esa simplicidad es la que lo hace tan popular.

Década de 1990

Aparece muy fuerte el tema orientado a los objetos, de tal forma que la administración de los datos estaría asociada a los objetos, como son las nuevas expresiones para las consultas, la aplicación de la recursividad, los disparadores y la programación Transact.

Nace la competencia para SQL y Oracle con programas de administración de base de datos de licencia libre como PostgreSQL y MySQL, teniendo una fuerte inclinación por administrar datos en la web.

SIGLO XXI

Se establecen fuertemente, más que en años anteriores, las empresas IBM, Microsoft y Oracle, cada una con sus sistemas de gestión de base de datos totalmente integrados para distintas plataformas.

1.3.2 Definición

De forma general, se puede decir que una base de datos es una colección de datos almacenados de manera coherente y especialmente permanente; los cuales pueden ser manipulados en cualquier tiempo, así como compartidos entre múltiples usuarios. Por otro lado, para la informática es considerada como un conjunto de datos relacionados entre sí, con un objetivo específico.

Entonces, una base de datos puede proporcionar acceso a datos a todo tipo de usuarios, quienes podrán manipular, visualizar, registrar, actualizar, eliminar y todo lo que se pueda realizar con la información proporcionada.

Finalmente, se ha de mencionar que las bases de datos pueden ser de distintos tamaños y presentar diversos grados de dificultad. Además, pueden ser almacenadas localmente, ya sea en la unidad de almacenamiento de una computadora o en cualquier dispositivo; o pueden ser distribuidas, es decir, la información se almacena en equipos remotos y se puede acceder a ella a través de una red.



Figura 1.3 Representación de la información en una base de datos
Fuente: <http://www.imagensocial.es/wp-content/uploads/2014/06/gestion_bases_datos.jpg>

1.3.3 Objetivos

Cualquiera de las formas en que se use una base de datos siempre contará con ventajas, ya que el uso de ellas se da por una necesidad de tener la información coherente a la mano en cualquier tiempo. A continuación, se mencionan algunos objetivos que presentan las bases de datos:

- **Independencia de la información:** Toda aplicación que se desarrolle y use una base de datos lo podrá realizar mediante la política de independencia, eso quiere decir que cualquier aplicación podrá usar la base de datos al mismo tiempo.
- **Coherencia en los resultados:** Al recuperar información desde la base de datos, esta devolverá el valor esperado siempre y cuando se especifique correctamente los criterios de búsqueda.
- **Redundancia de datos:** Reduce la duplicidad de datos aprovechando al máximo el espacio en disco y la inconsistencia de los datos.
- **Información compartida:** Esto permite que cualquier usuario autorizado pueda tener acceso a la información en cualquier momento.

1.4 Sistema de base de datos

Es un sistema que permite administrar de manera correcta las bases de datos, así como a los usuarios que acceden a ella. Dicha administración de datos se realiza con un sistema llamado DBMS del inglés Database Management System (Sistema de administración de bases de datos), el cual posee un conjunto de servicios que permiten gestionar los datos de una base de datos:



Figura 1.4 Sistema de base de datos

Un sistema de administración de base de datos puede dividirse en tres subsistemas:

- **Sistema de control de archivos:** Permite almacenar la información contenida en la base de datos en un medio físico.
- **Sistema de administración interna:** Permite distribuir la información de la base de datos en un determinado orden.
- **Sistema de administración externa:** Permite mostrar la información por medio de una interfaz que permitirá interactuar al usuario con la información contenida.

Los más conocidos sistemas de administración de bases de datos son:

IBM DB2	IBM DB2
Microsoft SQL Server	Microsoft SQL Server 2016
Microsoft Access	A
Oracle	ORACLE DATABASE 11g
Sybase	SYBASE
MySQL	MySQL
PostgreSQL	PostgreSQL

1.5 Usuarios

El manejo de la información de una organización no solo es responsabilidad de una persona, pues cada vez que alguien tenga contacto con la base de datos se le llamará usuario y pueden ser de los siguientes tipos:

1.5.1 DBA (Administrador de base de datos)

Es el encargado de llevar a cabo todas las actividades relacionadas con la base de datos como la implementación, la administración y el *software* que usará la base de datos. Entre las actividades que realiza está la de autorizar el acceso a las bases de datos, es decir, es el responsable del nivel de seguridad que ofrece la información de una organización. La responsabilidad del administrador de base de datos incluye políticas y procedimientos relativos a la gestión, seguridad, mantenimiento y el uso del sistema de base de datos dentro de una organización.

1.5.2 Diseñador de base de datos

Los diseñadores de bases de datos tienen casi las mismas actividades que un administrador de base de datos; lo que podría diferenciarlos es el tema de la responsabilidad. El diseñador de la base de datos, como dice su nombre, diseña, prueba y pone en marcha la base de datos aplicando las normativas de la organización. Entre las principales tareas que realiza el diseñador de base de datos están:

- Implementar el modelo lógico de datos que permita representar correctamente las necesidades de los usuarios finales.
- Realizar el estudio de viabilidad del proyecto de base de datos.
- Implementar las bases de datos en un sistema de gestión de datos.
- Realizar las pruebas de datos.
- Desarrollar aplicaciones que permitan mostrar la información almacenada en la base de datos a los usuarios finales.
- Realizar el mantenimiento y la actualización de la estructura de una base de datos según la necesidad de los usuarios finales.

1.5.3 Usuario final

Es el que hace uso de la información almacenada en la base de datos, ya sea para consultarla, actualizarla o usarla en un lenguaje de programación. Se dice que las bases de datos existen gracias al uso que le dan los usuarios finales.

1.6 Representación de la información

Cuando se trató el concepto de *información* se vio la importancia que esta tiene en el proceso de la implementación de una base de datos; pues lo que se almacena en una base de datos es justamente datos que, cuando tienen un propósito específico, se les llama información.

Entonces, un dato es cualquier valor numérico, imagen, audio, símbolo, texto, etc., que por sí solo no tiene significado; para que los datos determinen una decisión primero deben convertirse en información. Ahora, para que el procesador de la computadora entienda un dato usa un lenguaje binario de unos y ceros. Así sea una imagen o video que se registre en la base de datos, esta debe ser convertida al lenguaje binario por algún medio; por ejemplo, podría ser utilizando serializaciones de los lenguajes de programación.

Por otro lado, las bases de datos tienen como objetivo almacenar información, la cual se registra en forma de datos para luego ser administrados como el usuario lo desee. El término *dato* proviene del latín *datum* que significa 'lo que se da'; hay que tener en cuenta que un dato solo no representa nada, solo en conjunto tiene un valor significativo.

Finalmente, se puede decir que un dato es un valor importante en la implementación de una base de datos, y que este puede estar representado de forma simbólica ya sea mediante letras, números o símbolos. Asimismo, se debe considerar la siguiente tabla de unidades de medida para el almacenado de datos en la base de datos:

Unidad de medida	Símbolo	Equivalencia
1 Byte	-	8 bits
1 KiloByte	KB	1,024 Bytes
1 MegaByte	MB	1,024 KB
1 GigaByte	GB	1,024 MB
1 TeraByte	TB	1,024 GB
1 PetaByte	PB	1,024 TB
1 ExaByte	EB	1,024 PB
1 ZettaByte	ZB	1,024 EB
1 YottaByte	YB	1,024 ZB

1.6.1 Representación de la información desde el mundo real

Se mencionó que las bases de datos almacenan información con un objetivo específico; y que la información proviene de los datos. Ahora bien, ¿de dónde provienen los datos? Estos provienen de las entidades u objetos que se encuentran en nuestro universo; por tanto, nuestra primera labor será encontrar estas entidades. Véase el siguiente caso:

"La empresa CLEAN-SERVICE oferta sus productos de limpieza a partir del informe que envía el departamento de estudio de mercado de la empresa; esta se contacta telefónicamente con los posibles clientes y se concierta con ellos una cita en la empresa para ofertarles algún producto.

Al contactar telefónicamente con los clientes, se les toman sus datos personales para posteriormente realizar mailings *de ofertas. También se guardan los datos referentes a la cita. Para todas las citas concertadas se debe de realizar un control de acceso de las visitas, que básicamente consiste en solicitar la identificación de cualquier persona que acceda al recinto. No se permitirá entrar a ninguna persona que no tenga cita previa. A las personas que tengan cita concertada se les entregará una tarjeta de entrada, la cual deberán entregar a la salida firmada por el empleado al que han visitado. Semanalmente los empleados de la empresa generan un informe detallado a partir de los resultados obtenidos en las visitas realizadas; este informe se envía al departamento de Marketing".*

A partir de este caso se identifica a las entidades que componen el problema:

Entidad	Descripción
Producto 	Son los productos que produce y vende la empresa.
Informe 	Informe que detalla los productos que la empresa oferta.
Departamento de estudio de mercado 	Es el encargado de generar el informe de los productos ofertados.
Cliente 	Son los posibles clientes de la empresa.
Cita 	Se genera a partir de una llamada telefónica.
Tarjeta de entrada	Se le asigna al cliente el día de la cita.

Empleado 	Es la persona que atenderá los requerimientos de los clientes.
Informe de resultados	Es el informe que realizaran los empleados de la empresa según las citas con los clientes.

Se puede notar que todas las entidades encontradas tienen una descripción y un objetivo bien claro en el problema. Los posibles datos que puede tener cada entidad serán las siguientes:

Entidad	Dato
Producto	Código, descripción, precio, *stock*.
Informe	Número, fecha de registro.
Departamento de estudio de mercado	Encargado del departamento, número de empleados a cargo.
Cliente	Código, nombres, dirección, teléfono, correo electrónico.
Cita	Número, fecha, hora.
Tarjeta de entrada	Número, fecha, hora de entrada, hora de salida.
Empleado	Código, nombres, sueldo.
Informe de resultados	Fecha, número de pedidos.

Si alguna entidad encontrada no contiene dato; entonces, no será considerada como entidad, quizás se esté frente a un dato.

1.6.2 Representación de la información desde el dominio de las ideas

Es la representación de la información que se debe almacenar en la base de datos; no obstante, hay que tener en cuenta que primero se debe definir claramente las entidades y atributos de un proceso de negocio desde el mundo real. El dominio de las ideas establece una relación entre las entidades encontradas en el mundo real y la información que se almacenará físicamente en la base de datos en forma de registro.



Figura 1.5 Atributos de la entidad CLIENTE

Hay que tener en cuenta que la entidad fue determinada en la representación de la información desde el mundo real; esta debe ser considerada en forma singular, mientras que los atributos representan las características particulares de la entidad.

Según el caso expuesto anteriormente, se detalla la relación entre las entidades y los atributos:

Entidad	Atributo
PRODUCTO	CÓDIGO_PRO, DESCRIPCIÓN_PRO, PRECIO_PRO, STOCK_PRO
INFORME	NÚMERO_INF, FECHA_INF
DEPARTAMENTO_MERCADO	CÓDIGO_DEP, NOMBRE_DEP, NUM_EMP_DEP
CLIENTE	CÓDIGO_CLI, NOMBRES_CLI, DIRECCIÓN_CLI, TELÉFONO_CLI, CORREO_CLI
CITA	NÚMERO_CIT, FECHA_CIT, HORA_CIT
TARJETA_ENTRADA	NÚMERO_TAR, FECHA_TAR, HORA_ENT_TAR, HORA_SAL_TAR
EMPLEADO	CÓDIGO_EMP, NOMBRES_EMP, SUELDO_EMP
RESULTADOS	FECHA_RES, NÚMERO_PED_RES

1.6.3 Representación de la información desde los datos

La información desde los datos considera tanto el valor como su tipo de datos; es así que los atributos encontrados en la representación anterior deberán ser especificados por medio de información real y relacionados entre sí, formando un conjunto de registros como se observa en la siguiente imagen:



Figura 1.6 Representación de la información desde los datos

Entonces, la entidad se obtiene del mundo real y los atributos se obtienen desde el dominio de las ideas; esto forma una tabla que finalmente deberá tener información en forma de datos. En resumen, se puede decir que la entidad será conocida como tabla; los atributos, como columnas o campos de una tabla, y la información registrada, como registro o *tuplas*.

1.7 Modelo Entidad-Relación

Es un modelo que permite representar el diseño conceptual de una base de datos. A partir de un modelo E-R se puede determinar el diagrama de una base de datos en un gestor de base de datos como SQL Server.

Este modelo usa un conjunto de elementos gráficos que permiten representar la estructura general de una base de datos en forma lógica. Entre los principales requisitos para implementar un modelo de Entidad-Relación (E-R) se tiene las entidades, los atributos y las relaciones.

1.7.1 Entidades

Es considerado como un objeto obtenido del mundo real que tiene existencia propia y que a su vez se distingue de otros objetos. Una entidad puede ser un objeto físico como un libro, producto, persona; o de existencia conceptual como la venta, el robo. En el modelo E-R se representa gráficamente por un rectángulo:



Existen dos tipos de entidades:

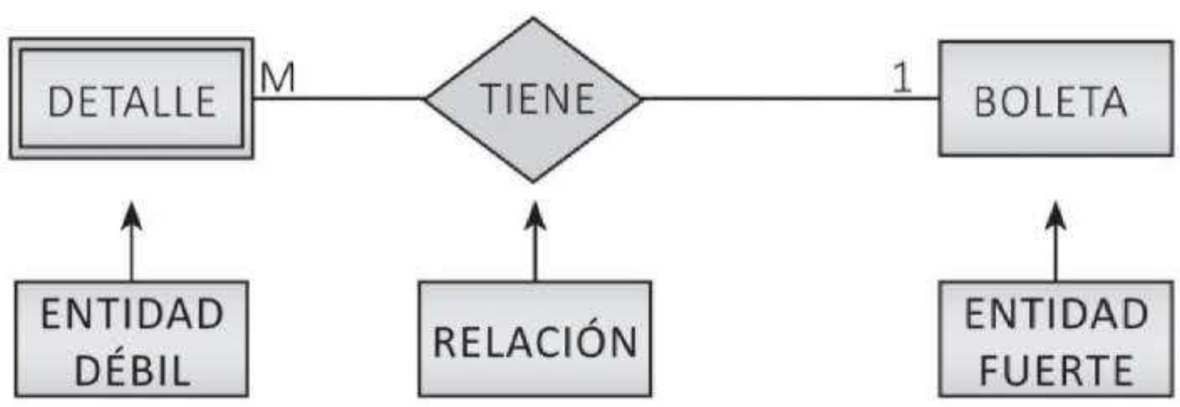
Entidad fuerte: Se caracteriza porque no depende de otra entidad; asimismo, cuenta con una clave primaria. Su forma gráfica se representa por medio de un rectángulo simple.



Entidad débil: Se caracteriza porque su existencia depende de la existencia de otra entidad. Y que la llave primaria de la entidad fuerte también será llave dentro de la entidad débil. Su representación gráfica se realiza mediante un rectángulo de doble línea.



En la siguiente imagen se distinguen los tipos de entidades en una relación:



Se puede interpretar la relación de la siguiente manera:

- Varios detalles están asociados a una sola boleta.
- Una boleta puede tener varios detalles registrados.

Entonces, la entidad BOLETA representa a la entidad fuerte porque los valores que contiene son necesarios para la especificación de los detalles; mientras que la entidad DETALLE depende de la existencia de la entidad BOLETA, porque si no existe una boleta, entonces no se puede registrar sus detalles.

A continuación, se muestra un ejemplo de especificación de atributos de ambas entidades en una relación:

Entidad: Boleta

COD_BOL	FEC_BOL
B0001	12/10/2018
B0002	13/10/2018
B0003	14/10/2018
B0004	15/10/2018

CAMPO CLAVE

Entidad: Detalle_Boleta

COD_BOL	COD_PRO	DES_PRO	CAN_PRO
B0001	P001	Yogurt 1 litro	10
B0001	P002	Gaseosa ½ litro	15
B0002	P001	Azúcar blanca 1 kilo	12
B0003	P002	Caramelos variados 1 kilo	8

CAMPO CLAVE (DÉBIL)

Finalmente, se ve que la entidad BOLETA cuenta con un campo clave llamado COD_BOL que será también especificado en la entidad débil como campo clave añadiendo a su clave original de la entidad débil. Cuando hay dos campos claves en una misma entidad se le llama llave compuesta. También hay que tener en cuenta que el campo COD_BOL de la entidad DETALLE_BOLETA es la clave foránea y por este medio se podrán asociar ambas entidades.

1.7.2 Atributos

Es considerado como una propiedad de una entidad. Estos describen la estructura de la base de datos; cuando se defina los atributos de una determinada entidad se debe pensar también con el objetivo que este tiene dentro de la base de datos, pues podría ser que se definan atributos que no tienen relevancia en una base de datos. Se puede representar de varias formas como se visualiza en la siguiente imagen:

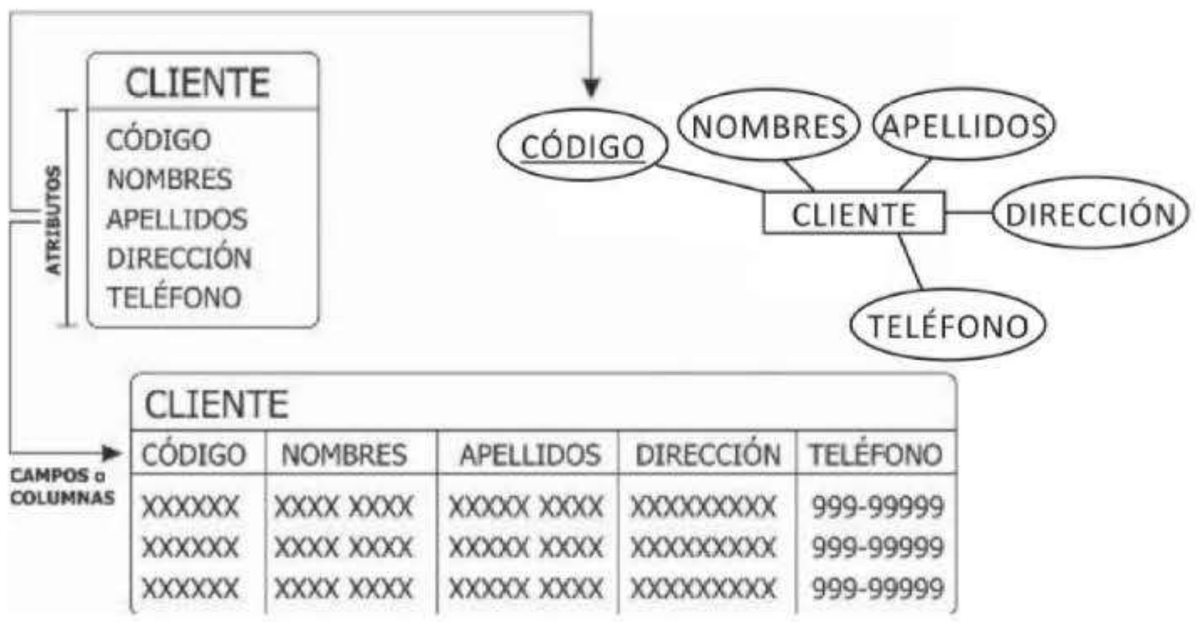


Figura 1.7 Atributos de una entidad

En el diagrama Entidad-Relación (DER) se pueden distinguir porque se especifican dentro de óvalos, mientras que en el modelo conceptual se le conoce como atributos, y en el esquema de base de datos de SQL Server se le conoce como campos o columnas de una tabla.

Los atributos cuentan con las siguientes partes:

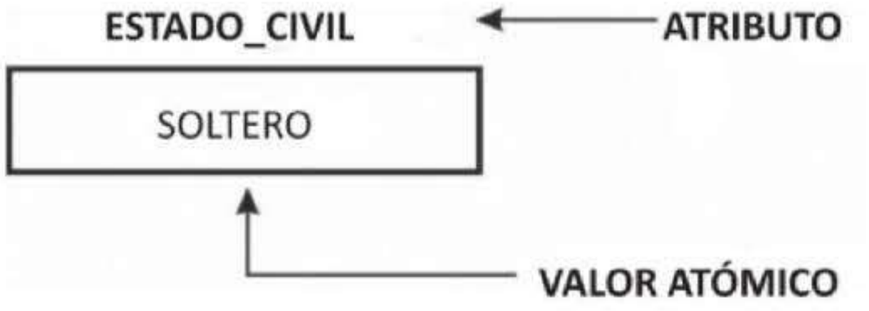


Se le llama ocurrencias a los posibles valores que puede tener un atributo; asimismo, se debe considerar qué valores se agregarán, ya que en SQL Server se debe definir exactamente el tipo de datos por cada atributo; por ejemplo, el código del cliente es de tipo cadena y solo debe almacenar cinco caracteres.

Existen tipos de atributos dentro de la colección contenida en una entidad:

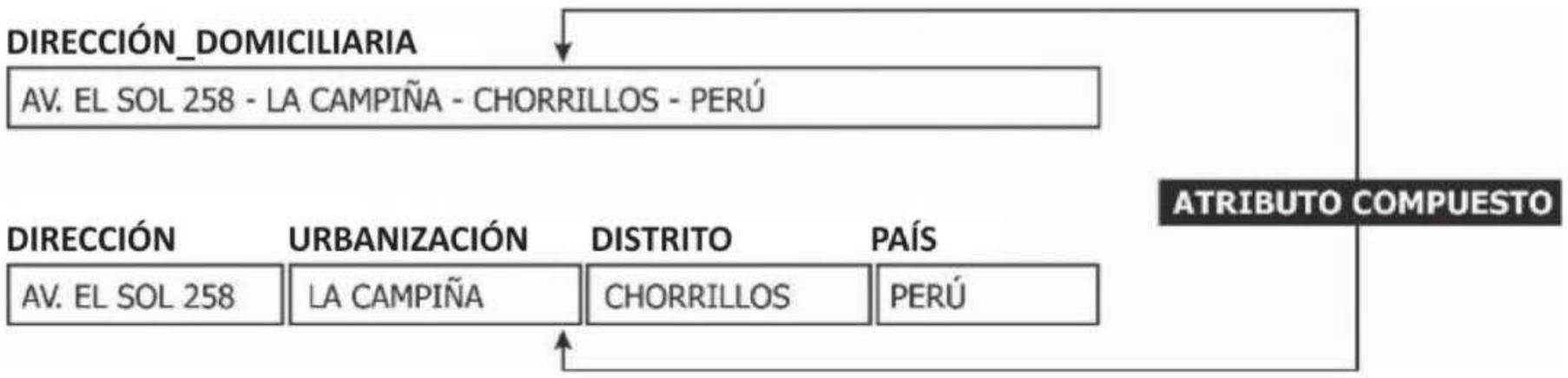
- Simples, atómicos o monovalorados

  Son valores que no tienen subdivisiones, es decir, representan a un solo valor, como por ejemplo el estado civil de una persona.



- Compuestos o multivalorados

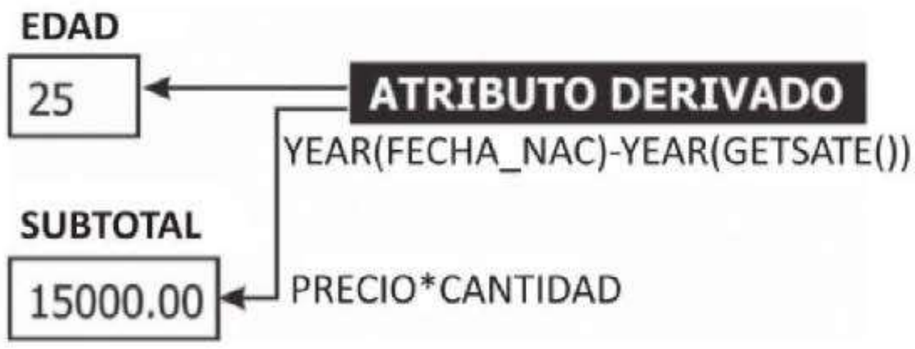
  Son atributos que contienen más de una información en un mismo valor.



Por ejemplo, la dirección domiciliaria de un empleado se puede registrar en un solo atributo y, como se visualiza, dentro del valor se especifica la dirección, la urbanización, el distrito y el país. Sin embargo, lo recomendable sería generar más atributos para almacenar la información compuesta como se observa en la imagen.

- Derivados

  Son llamados también calculados porque se obtienen a partir de otros atributos existentes; por ejemplo, el subtotal de una venta o la edad de un empleado. En el caso del subtotal, este se obtiene a partir de la cantidad comprada y el precio del producto; mientras que la edad se calcula a partir de la fecha de nacimiento del estudiante.



1.7.3 Relaciones de correspondencia

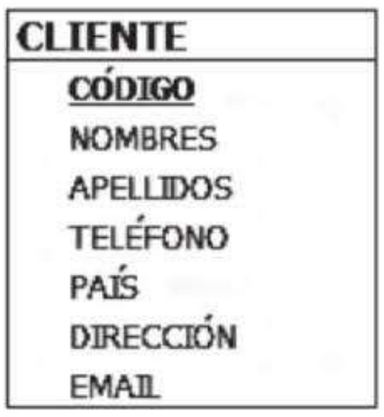
Es también conocido como cardinalidad y ocurre por el proceso de asociación entre dos entidades. Se representan en el diagrama Entidad-Relación mediante flechas y rombos.

El concepto básico de cardinalidad es que un número de instancias de una entidad puede asociarse a un elemento de la otra entidad relacionada. Su definición se realiza especificando cardinalidades mínimas y máximas, teniendo el siguiente juego de relaciones (0,1), (1,1), (0,n), (1,n), (m,n). Pero en el diseño de los diagramas de Entidad-Relación no es necesario ser tan específico, ya que solo se puede tomar los valores máximos del juego de relaciones.

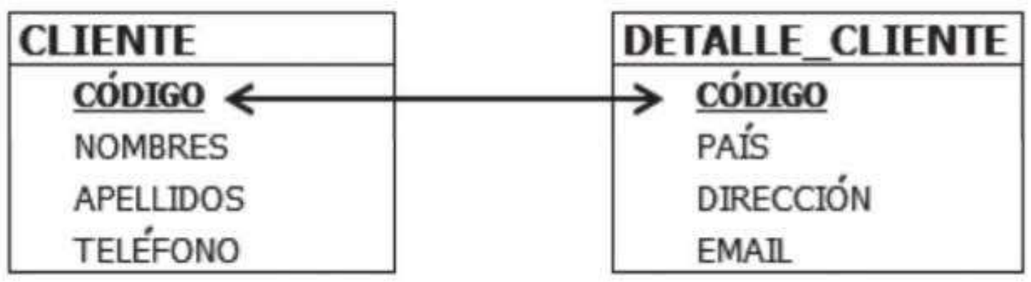
Algunos tipos de relaciones son:

- Uno a uno

  En una relación de entidades, uno de los registros se asocia de forma única a un solo registro de la tabla asociada. Estos casos podrían darse solo cuando la entidad tiene muchos atributos y es necesario crear una nueva entidad con los atributos sobrantes; por ejemplo, sea el caso que necesita registrar los siguientes datos de los clientes:



Y se toma la decisión de crear una entidad llamada DETALLE_CLIENTE, entonces se observa una relación uno a uno de la siguiente manera:



El código de la tabla CLIENTE se asocia con el código de DETALLE_CLIENTE una sola vez, ya que son detalles del cliente y solo le pertenece a uno de ellos. En forma de registros, se mostraría de la siguiente manera:

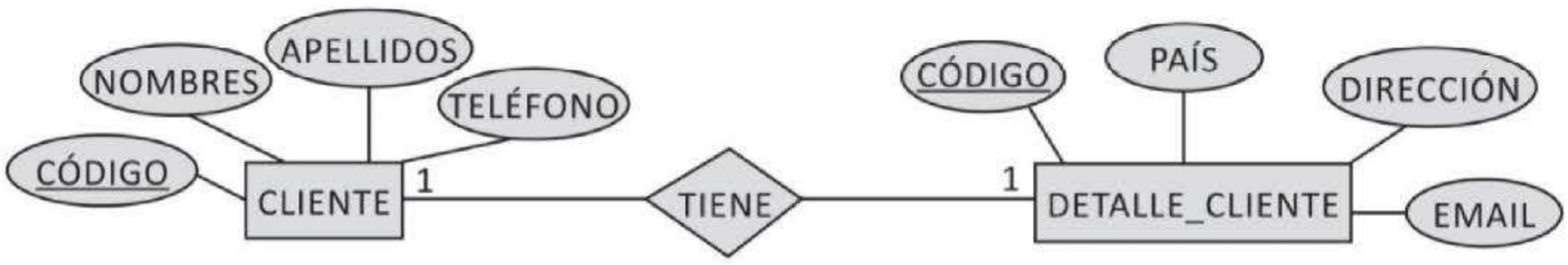
CLIENTE

CÓDIGO	NOMBRES	APELLIDOS	TELÉFONO
C0001	FERNANDA	TORRES LAZARO	521-8956
C0002	LUZ	LAZARO MENOR	452-5654
C0003	ANGELA VICTORIA	TORRES LAZARO	362-9856

DETALLE_CLIENTE

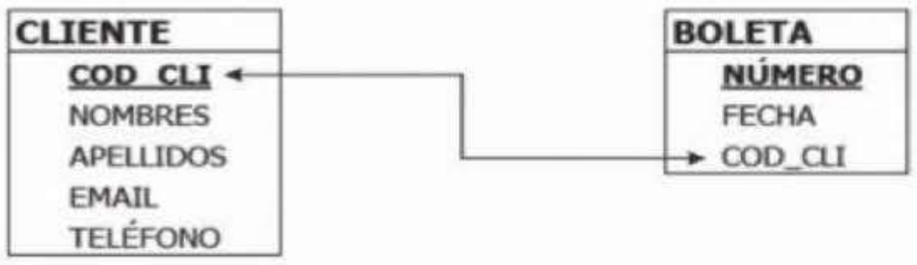
CÓDIGO	PAÍS	DIRECCIÓN	EMAIL
C0001	PERÚ	AV. EL SOL 563	FTORRES@GMAIL.COM
C0002	ECUADOR	AV. CIUDAD CENTRAL 545	LLAZARO@GMAIL.COM
C0003	CHILE	AV. ANTOFAGASTA 3455	ATORRES@HOTMAIL.COM

En el modelo Entidad-Relación, las entidades CLIENTE y DETALLE_CLIENTE se graficarían de la siguiente manera:



- Uno a muchos

En una relación de entidades, uno de los registros se asocia de forma única a muchos registros de la tabla asociada. La relación uno a muchos también se puede referenciar de forma inversa como muchos a uno.



El código de la tabla CLIENTE se asocia con el código de cliente especificado en la entidad BOLETA; entonces, se podría decir que un cliente puede tener registrado muchas boletas:

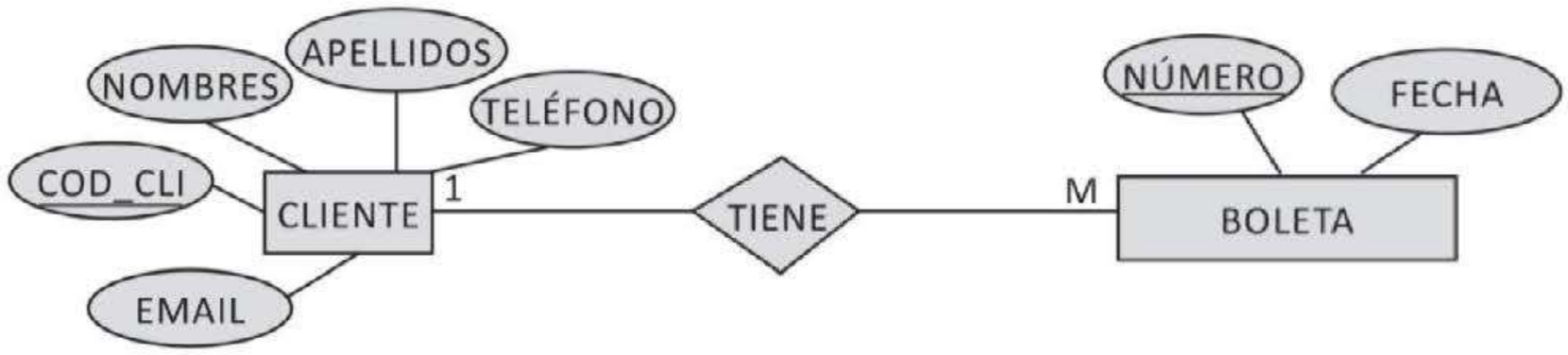
CLIENTE

COD_CLI	NOMBRES	APELLIDOS	EMAIL	TELÉFONO
C0001	FERNANDA	TORRES LAZARO	FTORRES@HOTMAIL.COM	963-955225
C0002	LUZ	LAZARO MENOR	LLAZARO@GMAIL.COM	985-968569
C0003	ANGELA	TORRES LAZARO	ATORRES@GMAIL.COM	963-965874
C0004	MANUEL	TORRES REMON	MTORRES@GMAIL.COM	925-963258

BOLETA

NÚMERO	FECHA	COD_CLI
1	10/10/2018	C0002
2	10/10/2018	C0001
3	11/10/2018	C0001
4	11/10/2018	C0003

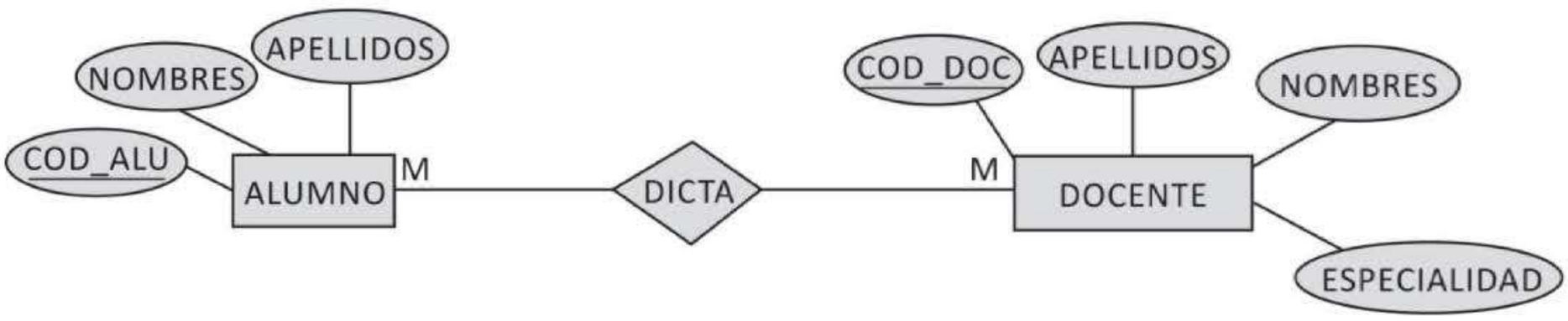
En la imagen se ve que el cliente con código C0001 tiene registrado las boletas dos y tres respectivamente. Hay que tener en cuenta que este tipo de relación cumple la cardinalidad uno a uno y, asimismo, el uno a muchos, ya que en determinado momento uno de los clientes podría registrar más boletas. En el modelo Entidad-Relación, las entidades CLIENTE y BOLETA se graficarían de la siguiente manera:



- Muchos a muchos

Uno de los registros de la entidad se asocia a varios registros de otra entidad y viceversa. El tipo de relación muchos a muchos genera una nueva entidad que contiene inicialmente los campos claves de ambas entidades. Esta relación es algo particular frente a las relaciones anteriores, pues físicamente no es implementable porque desglosa una nueva entidad; por ejemplo, podría darse el caso de que varios docentes dicten un mismo curso a varios alumnos.

En el modelo Entidad-Relación, las entidades ALUMNO y DOCENTE se graficarían de la siguiente manera:



La forma adecuada de representar la relación entre el docente y los alumnos es implementar una nueva entidad llamada DICTADO, tal como se muestra en la siguiente imagen:



La representación de los registros se muestra de la siguiente manera:

DOCENTE

COD_DOC	APELLIDOS	NOMBRES	ESPECIALIDAD
D0001	GARCIA ROMERO	LOURDES	ANÁLISIS Y DISEÑO
D0002	HURTADO YUCRA	ANALIA	DISEÑO GRÁFICO
D0003	FIGUEROA RAMIREZ	ROSA	PROGRAMACIÓN I

DICTADO

COD_DOC	COD_ALU	FECHA
D0001	A0001	10/5/2018
D0001	A0003	10/5/2018
D0002	A0002	11/6/2018
D0001	A0002	10/5/2018

ALUMNO

COD_ALU	APELLIDOS	NOMBRES
A0001	ALCALA ROJAS	JUAN PEDRO
A0002	DIAZ LOPEZ	GUADALUPE
A0003	DURAND FREITAS	FERNANDA
A0004	CASAS MENDOZA	DIANA
A0005	ROJAS ROJAS	GINA
A0006	PEREZ MIRANDA	VIVIANA

Tanto el código del docente y del alumno son necesarios para la implementación de la entidad DICTADO, ya que DICTADO se comporta como una entidad débil por la dependencia que existe con otras entidades.

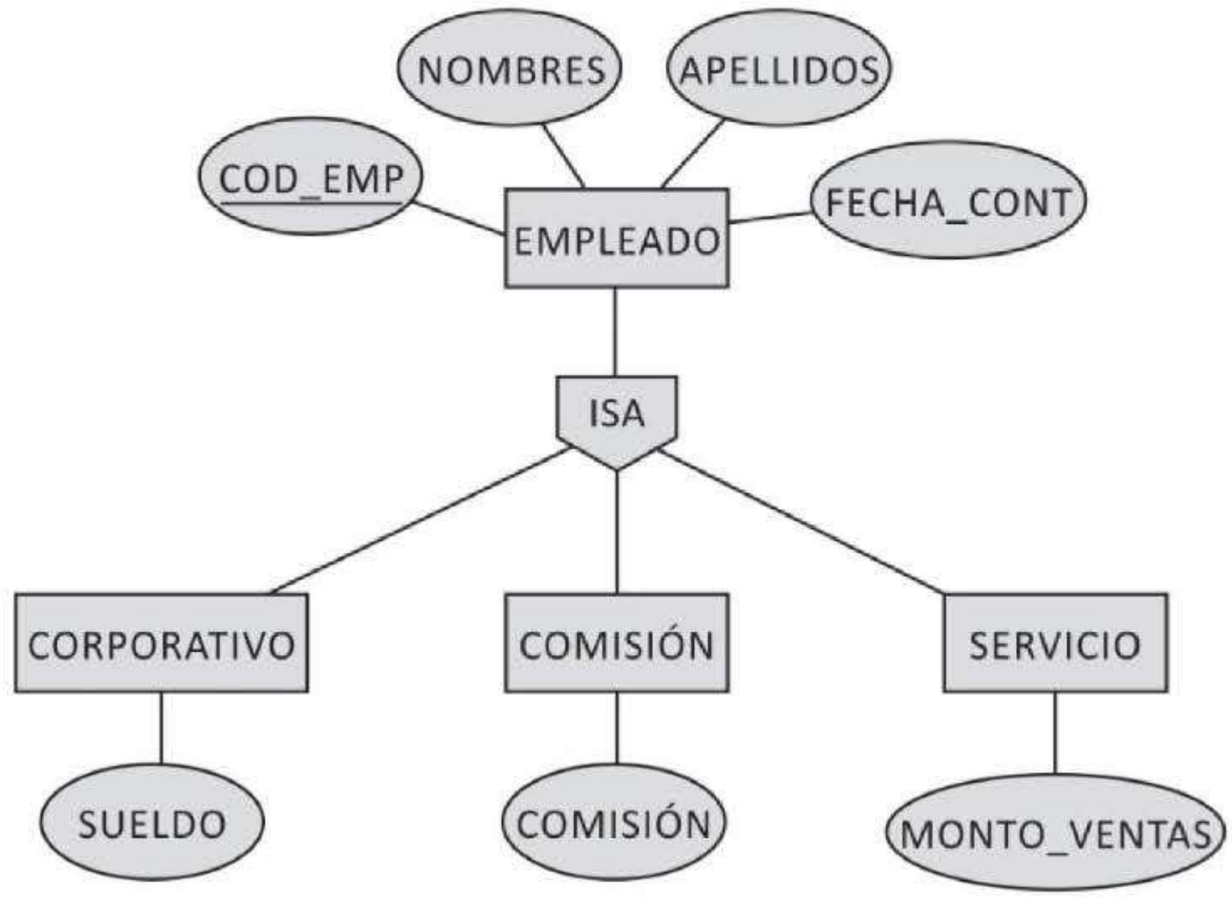
1.7.4 Generalización

Es un proceso de abstracción que permite organizar las entidades por características comunes. Si se nota que existen entidades con atributos similares, entonces se está frente a una generalización.

Por ejemplo, en una empresa existen tres tipos de empleados: el empleado corporativo, el cual tiene un sueldo fijo en la empresa; el empleado por comisión, el cual gana un porcentaje de comisión según las ventas que realiza, y el empleado por servicio, el cual es contratado para un determinado proyecto en un determinado tiempo. Veamos la siguiente imagen que refleja la generalización de los empleados:



Al generalizar a los empleados, los atributos también sufren una alteración como se muestra en la siguiente imagen:



La forma de visualizarlo con registros en cada una de las entidades sería de la siguiente manera:

EMPLEADO

COD_EMP	NOMBRES	APELLIDOS	FECHA_CONT
EM0001	FERNANDA XIMENA	TORRES LAZARO	10/06/2000
EM0002	ANGELA VICTORIA	TORRES LAZARO	11/05/2001
EM0003	LUZ VICTORIA	LAZARO MENOR	25/06/2002
EM0004	TORRES REMON	MANUEL ANGEL	06/06/2003
EM0005	MENDOZA MENOR	LUCERO	20/02/2004

CORPORATIVO

COD_EMP	SUELDO
EM0004	$ 3,500.00
EMP005	$ 2,500.00

COMISIÓN

COD_EMP	COMISIÓN
EM0001	$ 250.00
EM0003	$ 360.00

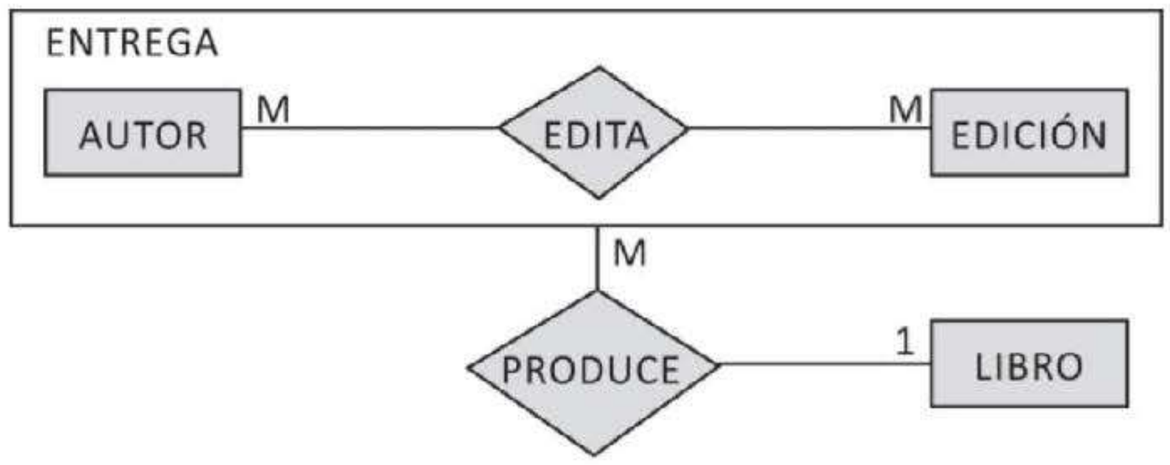
SERVICIO

COD_EMP	MONTO_VENTAS
EM0002	$ 25,000.00

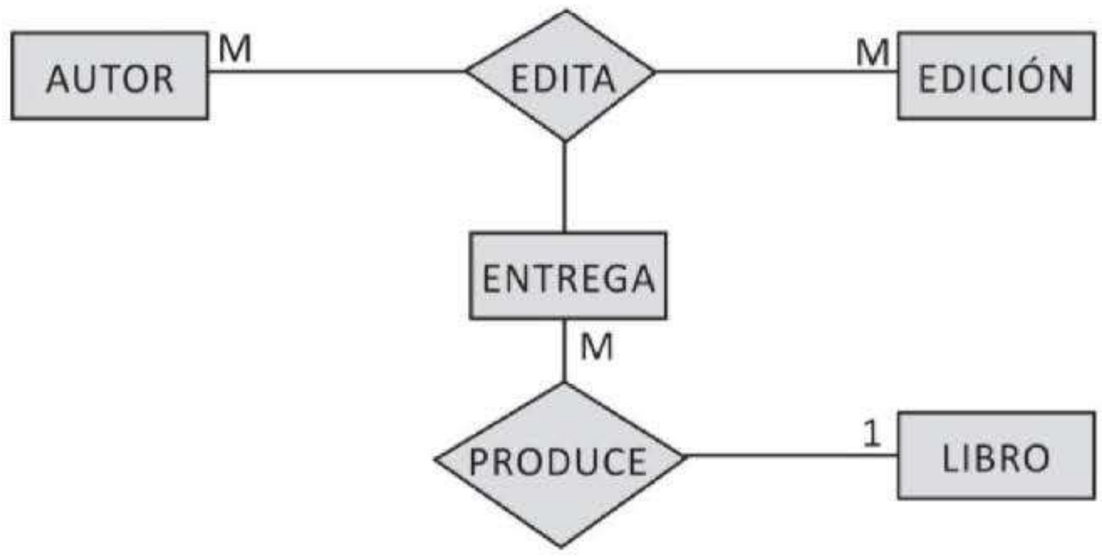
La entidad EMPLEADO lista todos los datos de los empleados, mientras que las entidades de bajo nivel muestran la información que diferencia a cada empleado. Hay que tener en cuenta que las entidades generadas a partir de la generalización tienen una relación de uno a uno con la entidad padre o base.

1.7.5 Agregación

Es una forma particular de relación entre entidades, pues un grupo de ellas puede formar una nueva entidad llamada entidad de alto nivel. La intención de formar una nueva entidad es que dicha entidad se asociará a otra por alguna acción particular.



En la imagen se observa que uno o más autores realizan un trabajo bibliográfico que es entregado al área de edición para una edición del material; todo esto se realiza mediante un contrato, y esto produce uno o más libros. Si no se aplica la regla de la agregación, se tendría un diseño similar al siguiente modelo:



A nivel de registros, se puede observar la siguiente imagen:

AUTOR

COD_AUT	NOMBRES	ESPECIALIDAD
A0001	JUAN COTRINA LOPEZ	INFORMATICA
A0002	ANDRES HIJAR NARRA	GESTION
A0003	JESUS JIRAF URQUIAGA	INGENIERIA
A0004	MILAGROS ROCA RUIZ	INFORMATICA

EDICIÓN

COD_EDI	FECHA
ED0001	10/10/2018
ED0002	10/11/2018
ED0003	10/12/2018

ENTREGA

COD_AUT	COD_EDI	COD_LIB	NUM_HOJAS
A0001	ED0002	L0001	550
A0001	ED0002	L0002	350
A0003	ED0001	L0003	150

LIBRO

COD_LIB	NOMBRE	CATEGORÍA
L0001	INFORMATICA BASICA	FISICO
L0002	EXCEL PARA INGENIERIOS	FISICO
L0003	PROGRAMACION	VIRTUAL

Caso desarrollado: "SALES S. A."

SALES S. A. es una empresa dedicada a la venta de productos al por mayor y menor; debido a la alta demanda de los productos que ofrece necesita tener un mejor control de la información que se genera por el proceso de venta.

Se necesita construir una base de datos que permita gestionar las ventas, vendedores, facturas, productos, orden de compra y los datos del cliente. Para ello, se necesita información del proveedor del producto como un identificador: razón social, dirección, teléfono, distrito y el nombre del representante legal. Asimismo, se deben registrar los datos del cliente como un identificador: nombre o razón social, dirección, teléfono, número de RUC, distrito, fecha de registro, tipo y condición del cliente. Por otro lado, se deben registrar los datos de los vendedores como un identificador: nombres, apellidos, sueldo, fecha de inicio, distrito y tipo de vendedor. Tanto los proveedores, clientes y vendedores se asocian a un determinado distrito, el cual cuenta con un identificador y una descripción.

Además, los vendedores registrarán la venta por medio de una factura, la cual cuenta con un número, fecha de registro, cliente, fecha de cancelación, estado, vendedor y porcentaje de IGV. Así, por cada factura registrada se genera un detalle del mismo, el cual cuenta con datos como número de factura, código del producto, cantidad y precio de venta. Dentro del detalle de la factura se especifica los productos que se registran por la venta con la siguiente información: un identificador de producto, descripción, precio, *stock* actual, *stock* mínimo, unidades, línea de producto y si el producto es importado.

Finalmente, cuando el proveedor ingresa productos a la empresa lo debe realizar mediante una orden de compra, la cual cuenta con un número identificador, fecha de registro, proveedor, fecha de atención y estado. Asimismo, por cada orden de compra se genera un detalle de orden, el cual cuenta con el número de orden, código del producto y la cantidad solicitada. Todo este proceso se registra en el proceso de abastecimiento, el cual cuenta con el código del proveedor, código de producto y el precio.

Solución:

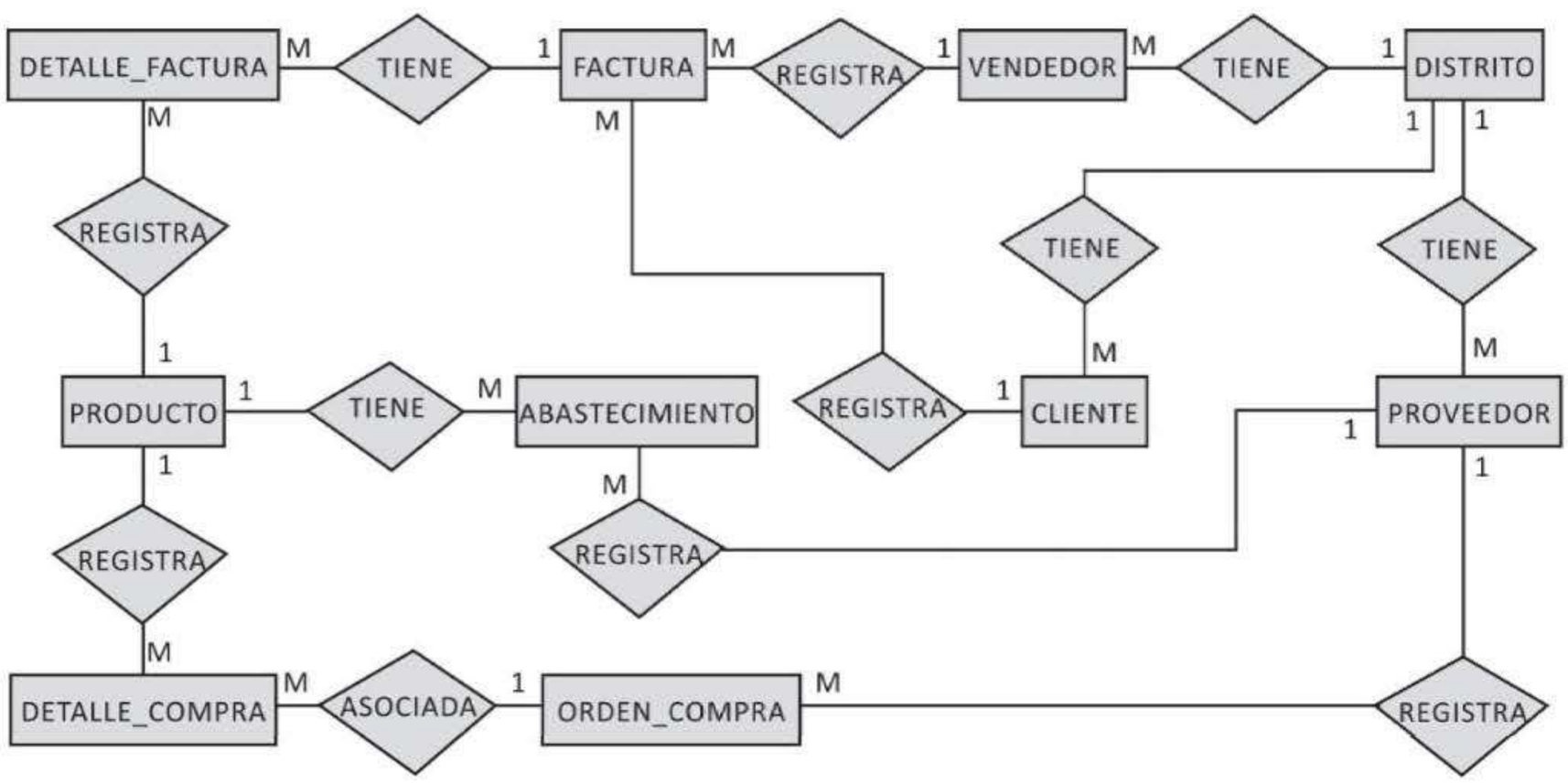
1. Listado de las entidades:

DISTRITO	Registra toda la información referente a los distritos que estarán asociados a un cliente, vendedor o proveedor.
PRODUCTO	Registra toda la información referente a los productos que ofrece la empresa.
CLIENTE	Registra toda la información referente a los clientes que por lo menos generaron una venta de producto.
VENDEDOR	Registra toda la información referente a los vendedores que permiten generar el proceso de venta de productos.
PROVEEDOR	Registra toda la información referente a los proveedores que ofrecen sus productos a la empresa.
FACTURA	Registra toda la información de la venta.
DETALLE_FACTURA	Registra el detalle de la venta, como el nombre y la cantidad de producto solicitado.
ORDEN_COMPRA	Registra los datos de la orden de compra de productos que estarán asociados a un determinado proveedor.
DETALLE_COMPRA	Registra la especificación del producto que genera una orden de compra.
ABASTECIMIENTO	Registra datos referentes al proveedor y el producto adquirido.

2. Listado de entidades y sus atributos:

DISTRITO	**COD_DIS**, NOM_DIS
PRODUCTO	**COD_PRO**, DES_PRO, PRE_PRO, SAC_PRO, SMI_PRO, UNI_PRO, LIN_PRO, IMP_PRO
CLIENTE	**COD_CLI**, RSO_CLI, DIR_CLI, TLF_CLI, RUC_CLI, COD_DIS, FEC_REG, TIP_CLI, CON_CLI
VENDEDOR	**COD_VEN**, NOM_VEN, APE_VEN, SUE_VEN, FIN_VEN, TIP_VEN, **COD_DIS**
PROVEEDOR	**COD_PRV**, RSO_PRV, DIR_PRV, TEL_PRV, **COD_DIS**, REP_PRV
FACTURA	**NUM_FAC**, FEC_FAC, **COD_CLI**, FEC_CAN, EST_FAC, COD_VEN, POR_IGV
DETALLE_FACTURA	**NUM_FAC**, **COD_PRO**, CAN_VEN, PRE_VEN
ORDEN_COMPRA	**NUM_OCO**, FEC_OCO, **COD_PRV**, FAT_OCO, EST_OCO
DETALLE_COMPRA	**NUM_OCO**, **COD_PRO**, CAN_DET
ABASTECIMIENTO	**COD_PRV**, **COD_PRO**, PRE_ABA

3. Diseño del diagrama de Entidad-Relación:



4. Script de creación de la base de datos en SQL Server:

```
USE MASTER
GO

SET DATEFORMAT YMD
GO

IF DB_ID('BDVENTAS') IS NOT NULL
        DROP DATABASE BDVENTAS
GO

CREATE DATABASE BDVENTAS
GO

USE BDVENTAS
GO

CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)         NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)     NOT NULL
)
GO

CREATE TABLE VENDEDOR(
        COD_VEN CHAR(3)         NOT NULL PRIMARY KEY ,
        NOM_VEN VARCHAR(20)     NOT NULL,
        APE_VEN VARCHAR(20)     NOT NULL,
        SUE_VEN MONEY           NOT NULL,
        FIN_VEN DATE            NOT NULL,
        TIP_VEN VARCHAR(10)     NOT NULL,
        COD_DIS CHAR(5)         NOT NULL REFERENCES DISTRITO
)
GO

CREATE TABLE CLIENTE(
        COD_CLI CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_CLI CHAR(30)        NOT NULL,
        DIR_CLI VARCHAR(100)    NOT NULL,
        TLF_CLI CHAR(9)         NOT NULL,
        RUC_CLI CHAR(11)        NULL,
        COD_DIS CHAR(5)         NOT NULL REFERENCES DISTRITO,
        FEC_REG DATE            NOT NULL,
        TIP_CLI VARCHAR(10)     NOT NULL,
        CON_CLI VARCHAR(30)     NOT NULL
)
GO

CREATE TABLE PROVEEDOR(
        COD_PRV CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)     NOT NULL,
        DIR_PRV VARCHAR(100)    NOT NULL,
        TEL_PRV CHAR(15)        NULL,
        COD_DIS CHAR(5)         NOT NULL REFERENCES DISTRITO,
        REP_PRV VARCHAR(80)     NOT NULL
)
GO
```

```
CREATE TABLE FACTURA(
        NUM_FAC VARCHAR(12)   NOT NULL PRIMARY KEY,
        FEC_FAC DATE          NOT NULL,
        COD_CLI CHAR(5)       NOT NULL REFERENCES CLIENTE,
        FEC_CAN DATE          NOT NULL,
        EST_FAC VARCHAR(10)   NOT NULL,
        COD_VEN CHAR(3)       NOT NULL REFERENCES VENDEDOR,
        POR_IGV DECIMAL       NOT NULL
)
GO

CREATE TABLE ORDEN_COMPRA(
        NUM_OCO CHAR(5)       NOT NULL PRIMARY KEY,
        FEC_OCO DATE          NOT NULL,
        COD_PRV CHAR(5)       NOT NULL REFERENCES PROVEEDOR,
        FAT_OCO DATE          NOT NULL,
        EST_OCO CHAR(1)       NOT NULL
)
GO

CREATE TABLE PRODUCTO(
        COD_PRO CHAR(5)       NOT NULL PRIMARY KEY,
        DES_PRO VARCHAR(50)   NOT NULL,
        PRE_PRO MONEY         NOT NULL,
        SAC_PRO INT           NOT NULL,
        SMI_PRO INT           NOT NULL,
        UNI_PRO VARCHAR(30)   NOT NULL,
        LIN_PRO VARCHAR(30)   NOT NULL,
        IMP_PRO VARCHAR(10)   NOT NULL
)
GO

CREATE TABLE DETALLE_FACTURA(
        NUM_FAC VARCHAR(12)   NOT NULL  REFERENCES FACTURA,
        COD_PRO CHAR(5)       NOT NULL  REFERENCES PRODUCTO,
        CAN_VEN INT           NOT NULL,
        PRE_VEN MONEY  NOT NULL
        PRIMARY KEY (NUM_FAC,COD_PRO)
)
GO

CREATE TABLE DETALLE_COMPRA(
        NUM_OCO CHAR(5)       NOT NULL  REFERENCES ORDEN_COMPRA,
        COD_PRO CHAR(5)       NOT NULL  REFERENCES PRODUCTO,
        CAN_DET INT           NOT NULL
        PRIMARY KEY (NUM_OCO,COD_PRO)
)
GO

CREATE TABLE ABASTECIMIENTO(
        COD_PRV CHAR(5)       NOT NULL  REFERENCES PROVEEDOR,
        COD_PRO CHAR(5)       NOT NULL  REFERENCES PRODUCTO,
        PRE_ABA MONEY  NOT NULL
        PRIMARY KEY (COD_PRV,COD_PRO)
)
GO
```

```
INSERT INTO DISTRITO VALUES('D01','SURCO')
INSERT INTO DISTRITO VALUES('D02','JESUS MARIA')
INSERT INTO DISTRITO VALUES('D03','SAN ISIDRO')
INSERT INTO DISTRITO VALUES('D04','LA MOLINA')
INSERT INTO DISTRITO VALUES('D05','SAN MIGUEL')
INSERT INTO DISTRITO VALUES('D06','MIRAFLORES')
INSERT INTO DISTRITO VALUES('D07','BARRANCO')
INSERT INTO DISTRITO VALUES('D08','CHORRILLOS')
INSERT INTO DISTRITO VALUES('D09','SAN BORJA')
INSERT INTO DISTRITO VALUES('D10','LINCE')
INSERT INTO DISTRITO VALUES('D11','BREÑA')
INSERT INTO DISTRITO VALUES('D12','MAGDALENA')
INSERT INTO DISTRITO VALUES('D13','RIMAC')
INSERT INTO DISTRITO VALUES('D14','SURQUILLO')
INSERT INTO DISTRITO VALUES('D15','PUEBLO LIBRE')
INSERT INTO DISTRITO VALUES('D16','BELLAVISTA')
INSERT INTO DISTRITO VALUES('D17','CALLAO')
INSERT INTO DISTRITO VALUES('D18','SAN MARTIN DE PORRES')
INSERT INTO DISTRITO VALUES('D19','SANTA ANITA')
INSERT INTO DISTRITO VALUES('D20','LOS OLIVOS')
INSERT INTO DISTRITO VALUES('D21','INDEPENDENCIA')
INSERT INTO DISTRITO VALUES('D22','LIMA - CERCADO')
INSERT INTO DISTRITO VALUES('D24','SAN LUIS')
INSERT INTO DISTRITO VALUES('D25','EL AGUSTINO')
INSERT INTO DISTRITO VALUES('D26','SAN JUAN DE LURIGANCHO')
INSERT INTO DISTRITO VALUES('D27','ATE - VITARTE')
INSERT INTO DISTRITO VALUES('D28','SAN JUAN DE MIRAFLORES')
INSERT INTO DISTRITO VALUES('D29','CARMEN DE LA LEGUA')
INSERT INTO DISTRITO VALUES('D30','COMAS')
INSERT INTO DISTRITO VALUES('D31','VILLA MARIA DEL TRIUNFO')
INSERT INTO DISTRITO VALUES('D32','VILLA EL SALVADOR')
INSERT INTO DISTRITO VALUES('D33','LA PERLA')
INSERT INTO DISTRITO VALUES('D34','VENTANILLA')
INSERT INTO DISTRITO VALUES('D35','PUENTE PIEDRA')
INSERT INTO DISTRITO VALUES('D36','CARABAYLLO')
INSERT INTO DISTRITO VALUES('D37','SANTA MARIA')

INSERT INTO VENDEDOR VALUES('V01','JUANA','ALVA',1000,'07/12/2003','1','D08')
INSERT INTO VENDEDOR VALUES('V02','JUAN','SOTO',1200,'10/03/2002','2','D03')
INSERT INTO VENDEDOR
VALUES('V03','CARLOS','AREVALO',1500,'02/10/1992','2','D09')
INSERT INTO VENDEDOR VALUES('V04','CESAR','OJEDA',850,'08/11/1999','1','D01')
INSERT INTO VENDEDOR VALUES('V05','JULIO','VEGA',1500,'11/05/1997','1','D01')
INSERT INTO VENDEDOR VALUES('V06','ANA','ORTEGA',1200,'11/05/1994','1','D05')
INSERT INTO VENDEDOR VALUES('V07','JOSE','PALACIOS',2500,'12/02/1997','1','D02')
INSERT INTO VENDEDOR VALUES('V08','RUBEN','GOMEZ',1450,'12/08/1996','2','D04')
INSERT INTO VENDEDOR
VALUES('V09','PATRICIA','ARCE',1800,'12/05/1997','2','D04')
```

```
INSERT INTO VENDEDOR
VALUES('V10','RENATO','PEREZ',1550,'11/08/1998','2','D01')

INSERT INTO CLIENTE VALUES('C001','FINSETH','AV. LOS VIÑEDOS
150','4342318','48632081','D05','12/10/2008','1','ALICIA BARRETO')
INSERT INTO CLIENTE VALUES('C002','ORBI','AV. EMILIO CAVENECIA
225','4406335','57031642','D04','02/01/2013','2','ALFONSO BELTRAN')
INSERT INTO CLIENTE VALUES('C003','SERVIEMSA','JR. COLLAGATE
522','75012403',NULL,'D05','06/03/2011','2','CHRISTIAN LAGUNA')
INSERT INTO CLIENTE VALUES('C004','ISSA','CALLE LOS AVIADORES
263','3725910','46720159','D01','9/12/2012','1','LUIS APUMAYTA')
INSERT INTO CLIENTE VALUES('C005','MASS','AV. TOMAS MARSANO
880','4446177','83175942','D14','10/01/2013','1','KATIA ARMEJO')
INSERT INTO CLIENTE VALUES('C006','BERKER','AV. LOS PROCERES
521','3810322','54890124','D05','7/05/2013','1','JUDITH ASTE')
INSERT INTO CLIENTE VALUES('C007','FIDENZA','JR. EL NIQUEL
282','5289034','16204790','D20','10/02/2012','2','HECTOR VIVANCO')
INSERT INTO CLIENTE VALUES('C008','INTECH','AV. SAN LUIS 2619  5TO P','
2249493','34021824','D09','01/07/2011','2','CARLOS VILLANUEVA')
INSERT INTO CLIENTE VALUES('C009','PROMINENT','JR. IQUIQUE
132','43233519',NULL,'D11','06/11/2010','1','JORGE VALDIVIA')
INSERT INTO CLIENTE VALUES('C010','LANDU','AV.NICOLAS DE AYLLON
1453','3267840','30405261','D05','11/06/2013','2','RAQUEL ESPINOZA')
INSERT INTO CLIENTE VALUES('C011','FILASUR','AV. EL SANTUARIO
1189','4598175','70345201','D26','08/04/2012','1','ANGELICA VIVAS')
INSERT INTO CLIENTE VALUES('C012','SUCERTE','JR. GRITO DE HUAURA
114','4206434','62014503','D05','11/08/2011','1','KARINA VEGA')
INSERT INTO CLIENTE VALUES('C013','HAYASHI','JR. AYACUCHO
359','42847990',NULL,'D22','03/09/2013','2','ERNESTO UEHARA')
INSERT INTO CLIENTE VALUES('C014','KADIA','AV.SANTA CRUZ 1332 OF.201','
4412418','22202915','D06','10/05/2011','1','MIGUEL ARCE')
INSERT INTO CLIENTE VALUES('C015','MEBA','AV. ELMER FAUCETT 1638','
4641234','50319542','D16','06/11/2011','2','RICARDO GOMEZ')
INSERT INTO CLIENTE VALUES('C016','CARDELI','JR. BARTOLOME HERRERA
451','2658853','26403158','D10','5/04/2013','2','GIANCARLO BONIFAZ')
INSERT INTO CLIENTE VALUES('C017','PAYET','CALLE JUAN FANNING 327','
4779834','70594032','D07','5/12/2013','1','PAOLA URIBE')
INSERT INTO CLIENTE VALUES('C018','DASIN','AV. SAENZ PEÑA 338 - B','
4657574','35016752','D17','12/03/2011','1','ANGELA BARRETO')
INSERT INTO CLIENTE VALUES('C019','COREFO','AV. CANADA 3894 -
3898','4377499','57201691','D24','01/03/2010','2','ROSALYN CORTEZ')
INSERT INTO CLIENTE VALUES('C020','CRAMER','JR. MARISCAL MILLER
1131','4719061','46031783','D02','8/11/2010','1','CHRISTOPHER RAMOS')

INSERT INTO PROVEEDOR VALUES('PR01','FABER CASTELL','AV. ISABEL LA CATOLICA
1875','4330895','D13','CARLOS AGUIRRE')
INSERT INTO PROVEEDOR VALUES('PR02','ATLAS','AV. LIMA
471','5380926','D13','CESAR TORRES')
INSERT INTO PROVEEDOR VALUES('PR03','3M','AV. VENEZUELA
3018','2908165','D16','OMAR INJOQUE')
```

```
INSERT INTO PROVEEDOR VALUES('PR04','DITO','AV. METROPOLITANA
376',NULL,'D19','RAMON FLORES')
INSERT INTO PROVEEDOR VALUES('PR05','ACKER','CALLE LAS DUNAS
245','4780143','D27','JULIO ACUÑA')
INSERT INTO PROVEEDOR VALUES('PR06','DEDITEC','CALLE PICHINCHA
644','5662848','D11','JAVIER GONZALES')
INSERT INTO PROVEEDOR VALUES('PR07','OFFICETEC','CALLE LAS PERDICES 225
OF. 204','4216184','D03','CARLOS ROBLES')
INSERT INTO PROVEEDOR VALUES('PR08','INVICTA','AV. LOS FRUTALES
564','4364247','D27','ALBERTO RODRIGUEZ')
INSERT INTO PROVEEDOR VALUES('PR09','DIPROPOR','AV. DEL AIRE
901','4742046','D24','ROBERTO RONCEROS')
INSERT INTO PROVEEDOR VALUES('PR10','MIURA','AV. LA PAZ
257','4459710','D06','JORGE VASQUEZ')
INSERT INTO PROVEEDOR VALUES('PR11','PRAXIS','AV. JOSE GALVEZ 1820 -
1844','4703944','D10','ERICKA ZEGARRA')
INSERT INTO PROVEEDOR VALUES('PR12','SUMIGRAF','AV. MANCO CAPAC
754','3320343','D13','KARINNA PAREDES')
INSERT INTO PROVEEDOR VALUES('PR13','LIMMSA','PROLG. HUAYLAS
670','2546995','D08','LAURA ORTEGA')
INSERT INTO PROVEEDOR VALUES('PR14','VENINSA','AV. TEJADA
338','2473832','D07','MELISA RAMOS')
INSERT INTO PROVEEDOR VALUES('PR15','CROSLAND','AV. ARGENTINA 3206 -
 3250','4515272','D17','JUAN RAMIREZ')
INSERT INTO PROVEEDOR VALUES('PR16','PETRAMAS','CALLE JOAQUIN MADRID 141 2DO
P',NULL,'D09','ROCIO GUERRERO')
INSERT INTO PROVEEDOR VALUES('PR17','REAWSE','AV. SANTA ROSA 480
',NULL,'D19','MARIA PEREZ')
INSERT INTO PROVEEDOR VALUES('PR18','EDUSA','AV. MORALES DUAREZ
1260','4525536','D29','PABLO MARTEL')
INSERT INTO PROVEEDOR VALUES('PR19','OTTMER','URB.PRO INDUSTRIAL MZ B6 LT
16','5369893','D18','ANGELA RENDILLA')
INSERT INTO PROVEEDOR VALUES('PR20','BARI','AV. ARNALDO MARQUEZ
1219',NULL,'D02','VANESA QUINTANA')

INSERT INTO FACTURA VALUES('FA001','06-07-2013','C001','05-08-
2013','2','V01','0.19')
INSERT INTO FACTURA VALUES('FA002','06-07-2013','C019','05-08-
2013','3','V02','0.19')
INSERT INTO FACTURA VALUES('FA003','01-09-2013','C003','03-10-
2013','2','V04','0.19')
INSERT INTO FACTURA VALUES('FA004','06-09-2013','C016','05-10-
2013','2','V07','0.19')
INSERT INTO FACTURA VALUES('FA005','01-10-2013','C015','12-10-
2013','2','V08','0.19')
INSERT INTO FACTURA VALUES('FA006','01-08-2013','C009','05-08-
2013','3','V05','0.19')
INSERT INTO FACTURA VALUES('FA007','05-10-2013','C019','10-10-
2013','1','V09','0.19')
INSERT INTO FACTURA VALUES('FA008','04-10-2013','C012','06-10-
2013','2','V10','0.19')
INSERT INTO FACTURA VALUES('FA009','03-10-2013','C008','10-10-
2013','2','V09','0.19')
```

```
INSERT INTO FACTURA VALUES('FA010','01-10-2013','C017','06-10-
2013','2','V02','0.19')
INSERT INTO FACTURA VALUES('FA011','10-09-2013','C019','10-09-
2013','2','V05','0.19')
INSERT INTO FACTURA VALUES('FA012','01-12-2013','C014','01-12-
2013','1','V04','0.19')
INSERT INTO FACTURA VALUES('FA013','01-12-2013','C011','01-12-
2013','3','V08','0.19')
INSERT INTO FACTURA VALUES('FA014','01-12-2012','C020','01-12-
2013','2','V09','0.19')
INSERT INTO FACTURA VALUES('FA015','08-12-2012','C015','06-01-
2013','2','V07','0.19')
INSERT INTO FACTURA VALUES('FA016','06-01-2013','C016','09-01-
2013','2','V05','0.19')
INSERT INTO FACTURA VALUES('FA017','06-01-2013','C015','06-01-
2013','1','V06','0.19')
INSERT INTO FACTURA VALUES('FA018','03-02-2013','C016','04-02-
2013','2','V10','0.19')
INSERT INTO FACTURA VALUES('FA019','07-02-2013','C008','01-03-
2013','2','V03','0.19')
INSERT INTO FACTURA VALUES('FA020','06-02-2013','C013','10-03-
2013','2','V02','0.19')

INSERT INTO ORDEN_COMPRA VALUES('OC001','05-03-2013','PR08',' 12-03-2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC002','08-04-2013','PR16',' 10-04-2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC003','02-08-2013','PR10' ,'02-08-2013',
'3')
INSERT INTO ORDEN_COMPRA VALUES('OC004','05-04-2013','PR01','5-04-2013', '3')
INSERT INTO ORDEN_COMPRA VALUES('OC005','06-03-2013','PR07',' 10-03-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC006','02-01-2013','PR19',' 2-01-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC007','06-02-2013','PR20' ,'5-04-2013',
'3')
INSERT INTO ORDEN_COMPRA VALUES('OC008','02-06-2013','PR04',' 01-07-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC009','03-08-2013','PR11',' 10-09-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC010','05-09-2013','PR01',' 5-09-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC011','02-10-2013','PR03',' 3-10-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC012','04-10-2013','PR14' ,'5-10-
2013','3')
INSERT INTO ORDEN_COMPRA VALUES('OC013','02-11-2013','PR05','6-11-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC014','03-11-2013','PR19',' 5-12-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC015','03-11-2013','PR18',' 10-12-
2013','1')
INSERT INTO ORDEN_COMPRA VALUES('OC016','06-12-2013','PR06' ,'06-12-
2013', '3')
```

```
INSERT  INTO  ORDEN_COMPRA  VALUES('OC017','08-01-2013','PR09','  8-01-
2013','1')
INSERT  INTO  ORDEN_COMPRA  VALUES('OC018','01-02-2013','PR20','  8-02-
2013','1')
INSERT  INTO  ORDEN_COMPRA  VALUES('OC019','03-03-2013','PR11','  6-03-
2013','1')
INSERT  INTO  ORDEN_COMPRA  VALUES('OC020','02-04-2013','PR05'  ,'2-04-
2013','2')

INSERT INTO  PRODUCTO VALUES('P001','PAPEL BOND A-4',35.0,
200,1500,'MLL','2','V')
INSERT INTO  PRODUCTO VALUES('P002','PAPEL BOND OFICIO ',35,
50,1500,'MLL','2','F')
INSERT INTO  PRODUCTO VALUES('P003','PAPEL BULKY ',10
,498,1000,'MLL','2','V')
INSERT INTO  PRODUCTO VALUES('P004','PAPEL
PERIODICO',9,4285,1000,'MLL','2','F')
INSERT INTO  PRODUCTO VALUES('P005','CARTUCHO TINTA NEGRA', 40
,50,30,'UNI','1','F')
INSERT INTO  PRODUCTO VALUES('P006','CARTUCHO TINTA COLOR ',45 ,58,35,
'UNI','1','F')
INSERT INTO  PRODUCTO VALUES('P007','PORTA
DISKETTES',3.5,300,100,'UNI','1','V')
INSERT INTO  PRODUCTO VALUES('P008','CAJA DE DISKETTES * 10
',30,125,180,'UNI','1','F')
INSERT INTO  PRODUCTO VALUES('P009','BORRADOR DE TINTA ',10
,100,500,'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P010','BORRADOR BLANCO  ',8
,2000,400,'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P011','TAJADOR METAL ',20
,1120,300,'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P012','TAJADOR PLASTICO',12
,608,300,'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P013','FOLDER MANILA
OFICIO',20,200,150,'CIE','3','F')
INSERT INTO  PRODUCTO VALUES('P014','FOLDER MANILA A-4',20
,150,150,'CIE','3','V')
INSERT INTO  PRODUCTO VALUES('P015','SOBRE MANILA
OFICIO',15,300,130,'CIE','3','F')
INSERT INTO  PRODUCTO VALUES('P016','SOBRE MANILA
A-4',18,200,100,'CIE','3','F')
INSERT INTO  PRODUCTO VALUES('P017','LAPICERO NEGRO',10,3000,1000,
'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P018','LAPICERO AZUL
',10 ,2013,1500,'DOC','3','F')
INSERT INTO  PRODUCTO VALUES('P019','LAPICERO
ROJO',8,1900,1000,'DOC','3','V')
INSERT INTO  PRODUCTO VALUES('P020','FOLDER PLASTICO A-4',50
,3080,1100,'CIE','3','F')
INSERT INTO  PRODUCTO VALUES('P021','PROTECTOR DE
PANTALLA',50,20,5,'UNI','1','F')
```

```
INSERT INTO DETALLE_FACTURA VALUES('FA001','P007','6'  ,'5')
INSERT INTO DETALLE_FACTURA VALUES('FA001','P011','25'  ,'25')
INSERT INTO DETALLE_FACTURA VALUES('FA001','P013','11'  ,'20')
INSERT INTO DETALLE_FACTURA VALUES('FA002','P004','8'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA003','P002','10'  ,'40')
INSERT INTO DETALLE_FACTURA VALUES('FA003','P011','6'  ,'20')
INSERT INTO DETALLE_FACTURA VALUES('FA003','P017','21'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA003','P019','12'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA004','P004','3'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA004','P009','50'  ,'5')
INSERT INTO DETALLE_FACTURA VALUES('FA005','P003','20'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA005','P006','50'  ,'50')
INSERT INTO DETALLE_FACTURA VALUES('FA005','P020','5'  ,'60')
INSERT INTO DETALLE_FACTURA VALUES('FA006','P002','20'  ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA006','P003','15'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA006','P009','12'  ,'5')
INSERT INTO DETALLE_FACTURA VALUES('FA007','P003','20'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA007','P012','4'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA008','P004','15'  ,'9')
INSERT INTO DETALLE_FACTURA VALUES('FA008','P008','15'  ,'30')
INSERT INTO DETALLE_FACTURA VALUES('FA008','P020','50'  ,'50')
INSERT INTO DETALLE_FACTURA VALUES('FA009','P001','5'  ,'30')
INSERT INTO DETALLE_FACTURA VALUES('FA009','P002','15'  ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA009','P006','2'  ,'50')
INSERT INTO DETALLE_FACTURA VALUES('FA009','P019','1'  ,'8')
INSERT INTO DETALLE_FACTURA VALUES('FA010','P002','3'  ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA011','P002','20'  ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA012','P002','3'  ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA012','P006','1'  ,'50')
INSERT INTO DETALLE_FACTURA VALUES('FA013','P002','130' ,'35')
INSERT INTO DETALLE_FACTURA VALUES('FA013','P003','5'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA013','P015','4'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA014','P009','10'  ,'8')
INSERT INTO DETALLE_FACTURA VALUES('FA015','P008','5'  ,'30')
INSERT INTO DETALLE_FACTURA VALUES('FA015','P016','3'  ,'18')
INSERT INTO DETALLE_FACTURA VALUES('FA016','P006','15'  ,'50')
INSERT INTO DETALLE_FACTURA VALUES('FA016','P008','2'  ,'30')
INSERT INTO DETALLE_FACTURA VALUES('FA017','P002','120' ,'40')
INSERT INTO DETALLE_FACTURA VALUES('FA017','P005','120' ,'40')
INSERT INTO DETALLE_FACTURA VALUES('FA018','P003','4'  ,'12')
INSERT INTO DETALLE_FACTURA VALUES('FA018','P005','6'  ,'40')
INSERT INTO DETALLE_FACTURA VALUES('FA019','P002','150' ,'40')
INSERT INTO DETALLE_FACTURA VALUES('FA019','P003','6'  ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA019','P006','2'  ,'45')
INSERT INTO DETALLE_FACTURA VALUES('FA019','P008','10'  ,'30')
INSERT INTO DETALLE_FACTURA VALUES('FA020','P009','120' ,'10')
INSERT INTO DETALLE_FACTURA VALUES('FA020','P015','5'  ,'15')

INSERT INTO DETALLE_COMPRA VALUES('OC001','P006','100')
INSERT INTO DETALLE_COMPRA VALUES('OC001','P016','20')
```

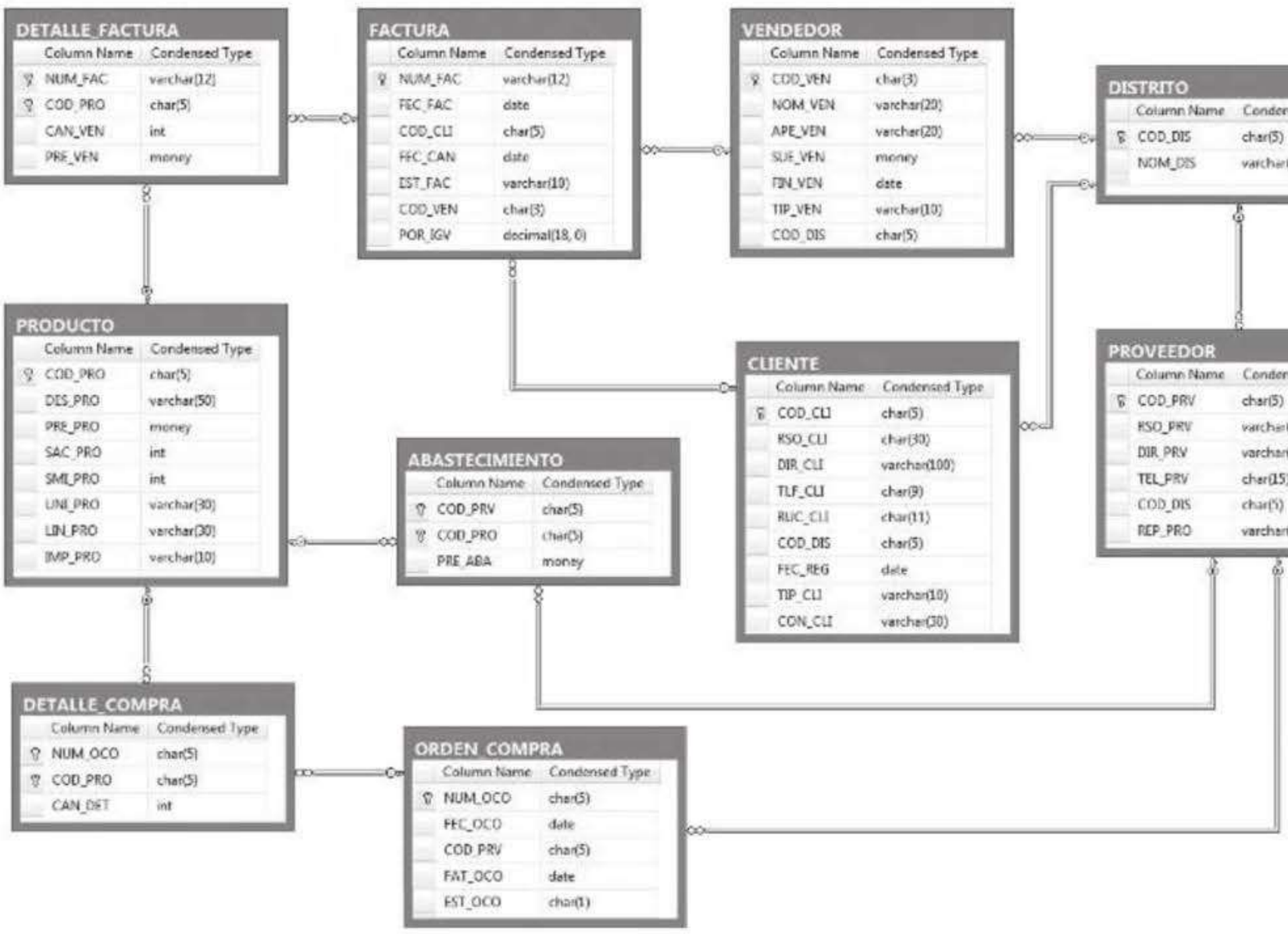
```
INSERT INTO DETALLE_COMPRA VALUES('OC002','P003','200')
INSERT INTO DETALLE_COMPRA VALUES('OC002','P005','500')
INSERT INTO DETALLE_COMPRA VALUES('OC003','P005','50')
INSERT INTO DETALLE_COMPRA VALUES('OC004','P009','10')
INSERT INTO DETALLE_COMPRA VALUES('OC004','P013','50')
INSERT INTO DETALLE_COMPRA VALUES('OC005','P007','150')
INSERT INTO DETALLE_COMPRA VALUES('OC005','P008','100')
INSERT INTO DETALLE_COMPRA VALUES('OC008','P002','10')
INSERT INTO DETALLE_COMPRA VALUES('OC008','P012','100')
INSERT INTO DETALLE_COMPRA VALUES('OC009','P009','50')
INSERT INTO DETALLE_COMPRA VALUES('OC009','P011','50')
INSERT INTO DETALLE_COMPRA VALUES('OC010','P001','100')
INSERT INTO DETALLE_COMPRA VALUES('OC011','P008','5')
INSERT INTO DETALLE_COMPRA VALUES('OC011','P016','10')
INSERT INTO DETALLE_COMPRA VALUES('OC012','P007','50')
INSERT INTO DETALLE_COMPRA VALUES('OC012','P011','100')
INSERT INTO DETALLE_COMPRA VALUES('OC013','P013','50')
INSERT INTO DETALLE_COMPRA VALUES('OC014','P004','50')
INSERT INTO DETALLE_COMPRA VALUES('OC014','P008','50')
INSERT INTO DETALLE_COMPRA VALUES('OC014','P020','50')
INSERT INTO DETALLE_COMPRA VALUES('OC016','P015','100')
INSERT INTO DETALLE_COMPRA VALUES('OC017','P012','100')
INSERT INTO DETALLE_COMPRA VALUES('OC017','P014','100')
INSERT INTO DETALLE_COMPRA VALUES('OC019','P006','100')
INSERT INTO DETALLE_COMPRA VALUES('OC020','P005','500')
INSERT INTO DETALLE_COMPRA VALUES('OC020','P011','100')

INSERT INTO ABASTECIMIENTO VALUES('PR01','P003','8')
INSERT INTO ABASTECIMIENTO VALUES('PR01','P005','35')
INSERT INTO ABASTECIMIENTO VALUES('PR01','P007','3')
INSERT INTO ABASTECIMIENTO VALUES('PR01','P009','8')
INSERT INTO ABASTECIMIENTO VALUES('PR01','P011','18')
INSERT INTO ABASTECIMIENTO VALUES('PR02','P002','30')
INSERT INTO ABASTECIMIENTO VALUES('PR02','P007','3')
INSERT INTO ABASTECIMIENTO VALUES('PR03','P002','32')
INSERT INTO ABASTECIMIENTO VALUES('PR03','P004','7')
INSERT INTO ABASTECIMIENTO VALUES('PR04','P001','28')
INSERT INTO ABASTECIMIENTO VALUES('PR04','P006','40')
INSERT INTO ABASTECIMIENTO VALUES('PR05','P018','9')
INSERT INTO ABASTECIMIENTO VALUES('PR06','P009','7')
INSERT INTO ABASTECIMIENTO VALUES('PR06','P017','8')
INSERT INTO ABASTECIMIENTO VALUES('PR07','P016','15')
INSERT INTO ABASTECIMIENTO VALUES('PR07','P019','7')
INSERT INTO ABASTECIMIENTO VALUES('PR08','P006','42')
INSERT INTO ABASTECIMIENTO VALUES('PR08','P010','6')
INSERT INTO ABASTECIMIENTO VALUES('PR09','P002','30')
INSERT INTO ABASTECIMIENTO VALUES('PR09','P014','17')
INSERT INTO ABASTECIMIENTO VALUES('PR11','P001','27')
INSERT INTO ABASTECIMIENTO VALUES('PR11','P006','44')
INSERT INTO ABASTECIMIENTO VALUES('PR12','P002','33')
```

```
INSERT INTO ABASTECIMIENTO VALUES('PR12','P010','7')
INSERT INTO ABASTECIMIENTO VALUES('PR13','P005','35')
INSERT INTO ABASTECIMIENTO VALUES('PR14','P016','15')
INSERT INTO ABASTECIMIENTO VALUES('PR15','P020','45')
INSERT INTO ABASTECIMIENTO VALUES('PR16','P008','25')
INSERT INTO ABASTECIMIENTO VALUES('PR16','P012','9')
INSERT INTO ABASTECIMIENTO VALUES('PR16','P013','15')
INSERT INTO ABASTECIMIENTO VALUES('PR19','P008','28')
INSERT INTO ABASTECIMIENTO VALUES('PR19','P016','16')
INSERT INTO ABASTECIMIENTO VALUES('PR20','P012','10')
INSERT INTO ABASTECIMIENTO VALUES('PR20','P020','43')

SELECT * FROM CLIENTE
SELECT * FROM PROVEEDOR
SELECT * FROM FACTURA
SELECT * FROM ORDEN_COMPRA
SELECT * FROM PRODUCTO
SELECT * FROM DETALLE_FACTURA
SELECT * FROM DETALLE_COMPRA
SELECT * FROM VENDEDOR
SELECT * FROM DISTRITO
SELECT * FROM ABASTECIMIENTO
```

5. Diagrama de base de datos en SQL Server:





Capítulo 2

Gestión de base de datos

Competencia tecnológica:

- Reconocer las principales funciones de SQL Server 2016.
- Identificar los componentes de una base de datos usando sentencias de programación.
- Crear y administrar una base de datos a partir de sentencias DDL.

Contenido:

2.1 SQL Server 2016
2.2 Características de SQL Server 2016
2.3 Ediciones de SQL Server 2016
2.4 Historial de revisiones de SQL Server
2.5 Instalación de SQL Server 2016
2.6 SQL Management Studio
2.7 Las bases de datos en SQL Server
2.8 Componentes de una base de datos
2.9 Motor de base de datos
2.10 Sentencias DDL
2.11 Sentencias DML
2.12 Creación de una base de datos
2.13 Modificación de archivos de una base de datos
2.14 Activar la base de datos
2.15 Los tipos de datos en SQL Server 2016
2.16 Tipos de datos definidos por el usuario
- Caso desarrollado 1: Base de datos básica
- Caso desarrollado 2: Base de datos con especificación de archivo primario
- Caso desarrollado 3: Base de datos con especificación de archivos secundarios
- Caso desarrollado 4: Base de datos con especificación de archivos de transacciones
- Caso desarrollado 5: Añadir archivos a la base de datos
- Caso desarrollado 6: Modificar propiedades de la base de datos
- Caso desarrollado 7: Eliminar archivos de la base de datos
- Caso desarrollado 8: Integración

2.1 SQL Server 2016



Es considerado un sistema de gestión de base de datos relacionales diseñado por Microsoft, que no solo almacena datos sino también implementa soluciones de comercio electrónico, soluciones para inteligencia de negocios e informe de datos.

La versión 2016 presenta un diseño mejorado, ya que permite implementar aplicaciones eficientes para procesamientos de transacciones en línea mejor llamado OLTP con alta escalabilidad, rendimiento y disponibilidad. La seguridad ha sido uno de los puntos fuertes, pues protege la información en reposo y en movimiento. Mantiene un nivel de integración de datos entre diferentes dispositivos tenga o no conexión. El nivel de análisis permite obtener información para la toma de decisiones sin necesidad de mover los datos. Y con respecto a la disposición de la información en la nube, proporciona herramientas coherentes para el manejo de la información en cualquier lugar, ya sea su centro de trabajo o nube privada.

SQL Server toma mucha importancia en la actualidad, pues existe mucha información (millones) en el mundo y esta crece de manera exponencial, tanto por factor humano como por máquina; por ello, las bases de datos de SQL suelen considerarse como la "columna vertebral" de la arquitectura TI en una organización.

2.2 Características de SQL Server 2016

Con respecto a sus antecesores, SQL Server 2016 presenta mejoras, las cuales se verán a continuación:

2.2.1 Tareas críticas

Mediante nuevas funciones en el tratamiento de tareas críticas, mejora en flexibilidad, rendimiento y disponibilidad para sus procesamientos de transacciones, así como en las cargas de trabajo de almacenamiento de datos.

- Permite ampliar la memoria del servidor hasta 12 *terabytes*; con esto consigue ejecutar transacciones hasta 30 veces más rápidas, y hasta 100 veces más rápidas con respecto a las consultas.
- Mejora en la multiprogramación con el uso de más de 640 procesadores lógicos.
- Realiza análisis operacionales en tiempo real sobre datos de transacciones.

2.2.2 Seguridad y rendimiento

Considerado como la base de datos más usado en el mundo, presenta un alto nivel de seguridad gracias al trabajo que se realiza mediante multicapas.

- Es considerado un sistema de gestión de base de datos con menos vulnerabilidades entre muchas plataformas.
- Presenta un mejorado cifrado de datos transparentes (EDT), aplicando nuevas técnicas de protección de datos almacenados y en movimiento.
- Presenta un enmascaramiento de datos dinámico (DDM) que permite ocultar información de nivel confidencial; solo será accesible para usuarios con privilegios.
- Mejora la gestión del acceso de usuarios al servidor, así como la administración de la información.

2.2.3 Inteligencia empresarial

Usa una plataforma de inteligencia de negocios (BI) completa, la cual permite gestionar información compleja y convertirla en herramientas de ayuda para la toma de decisiones.

- Mejora los modelos de informes y adiciona un conjunto de gráficos nuevos para la presentación de los informes.
- Presenta el portal web de servicios de informes, el cual accede a los indicadores claves de rendimiento para informes móviles y paginados.
- Agiliza y mejora el rendimiento en servicios de análisis con procesamiento paralelo.
- Usa nuevos modelos multidimensionales en los servicios de análisis.
- Permite una amigable configuración de la vista previa de las herramientas de datos de SQL Server en Visual Studio.

2.2.4 Análisis de base de datos

Los análisis avanzados integrados proporcionan escalabilidad y rendimiento para construir y ejecutar los algoritmos de análisis directamente en el núcleo de la base de datos transaccional de SQL Server.

- Mejora la administración de datos proveniente de otras fuentes.
- Transforma datos complejos, procedentes de otras fuentes, en modelos de datos de confianza, utilizando un lenguaje de modelado estadístico.
- Los modelos creados son implementados en cualquier plataforma como en la misma base de datos, la nube o para Linux.

2.3 Ediciones de SQL Server 2016

SQL Server es un sistema de manejo de bases de datos del modelo relacional. Las ediciones que presenta para la versión 2016 varían según la necesidad de las aplicaciones que pretende desarrollar una determinada organización; entre ellas, se tiene:

2.3.1 Enterprise

Es la versión más completa de SQL Server 2016, ya que gestiona, de manera eficiente y confiable, las bases de datos e inteligencia de negocios. Asimismo, proporciona los niveles más altos en servicio y rendimiento para cargas de trabajo de nivel uno. Tiene un modelo de licencia propietario y se realiza pagos por volúmenes.

2.3.2 Estándar

Presenta funcionalidad para la administración de base de datos e inteligencia de negocios básica, principalmente, con carga de trabajo no críticas. Tiene un modelo de licencia propietario con un menor costo que el Enterprise, e igualmente se paga por volúmenes.

2.3.3 Developer

Es la versión funcional de SQL Server, la cual permite a desarrolladores probar todas sus capacidades mediante el desarrollo de aplicaciones. Tienen un modelo de licencia gratuito.

2.3.4 Web

Es la versión web de SQL Server, la cual está disponible solo para proveedores de servicios de *software*. Tiene un modelo de licencia propietario y el costo se basa en la cantidad de *hosting*.

2.3.5 Express

Es la versión básica de SQL Server desde la cual se podrá implementar aplicaciones de mediana escala con un margen de datos de hasta 10GB. Su modelo de licencia es gratuita.

2.4 Historial de revisiones de SQL Server

Cada versión que se presenta en SQL Server mejora a la anterior, principalmente, en temas de seguridad y, como en el caso de la versión 2016, resalta su Always Encrypted y Role Level Security.

Versión	Año de lanzamiento	Nombre del proyecto
1.0	1989	SQL SQL Server 1.0
4.21	1993	SEQUEL SQL Server 4.21
6.0	1995	SQL95 SQL Server 2.6
6.5	1996	Hydra SQL Server 2.6.5

7.0	1998	Sphinx SQL Server 7.0
8.0	2000 2003	Shiloh (SQL Server 2000) Liberty (SQL Server 2000 64 bit)
9.0	2005	Yukon SQL Server 2005
10.0	2008	Katmai SQL Server 2008
10.5	2010	Kilimanjaro SQL Server 2008 R2
11.0	2012	Denali SQL Server 2012
12.0	2014	Hekaton SQL Server 2014
13.0	2016	SQL Server 2016

2.5 Instalación de SQL Server 2016

Antes de iniciar la instalación, se debe considerar algunos aspectos principales con respecto a esta nueva versión y, principalmente, a sus ediciones. A continuación, se detallan algunas recomendaciones con respecto al *hardware* y *software*:

- Se recomienda instalar SQL Server 2016 en discos locales con formatos de archivo NTFS por presentar un nivel de seguridad aceptable y seguro.

> Tipo: Disco local
> Sistema de archivos: NTFS

- Si la instalación se realiza en unidades de disco de solo lectura o comprimidas automáticamente, la instalación se bloqueará.
- Si la instalación se realiza por medio de conexión remota a escritorio, esta no se llevará a cabo.
- La instalación correcta se origina si se realiza la instalación a partir de un recurso compartido de red, unidad asignada, unidad local o como una imagen ISO de una máquina virtual.
- Para la instalacion del entorno SQL Server Management Studio es necesario el framework .NET 4.6.1, el cual se puede descargar desde la URL <https://www.microsoft.com/es-es/download/details.aspx?id=49982>.

Los requisitos mínimos para la instalación de SQL Server 2016, tanto en *hardware* como en *software*, son los siguientes:

Componente	Requisito
.NET Framework	Mínimamente se necesita el .NET Framework 4.6 para el Motor de base de datos, Master Data Services o la replicación. La instalación de dicho framework se puede realizar de forma manual mediante la URL <https://www.microsoft.com/es-es/download/details.aspx?id=49982>.
Software de red	Los sistemas operativos en la actualidad presentan un *software* de red integrado y no será necesario actualizarlos, ya que todos los sistemas permiten protocolos de red como memoria compartida, canalizaciones con nombre y TCP/IP.
Disco duro	El espacio en disco varía según los componentes de instalación de SQL Server 2016. Mínimamente se requiere 6GB de espacio disponible en disco.
Memoria	Para la versión Express se requiere un mínimo de 512MB; para las demás versiones, un mínimo de 4GB. Además, se recomienda que, conforme aumente la información de la base de datos, también se aumente la memoria.
Procesador	Mínimo recomendado x64 de 2Ghz o superior.

Proceso de instalación

Primero se debe obtener el *software* de instalación, el cual se puede descargar desde la siguiente URL: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>.



Fuente: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

El proceso de instalación presenta un asistente que proporciona un árbol de características para instalar componentes de SQL Server, las cuales son:

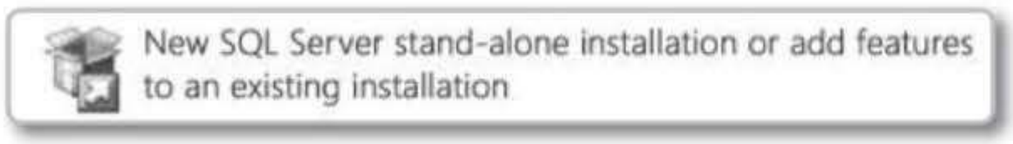
- Motor de base de datos
- Analysis Services
- Reporting Services
- Integration Services
- Master Data Services
- Data Quality Services
- Componentes de conectividad

En la versión 2016, las herramientas de administración de SQL Server ya no se instalan desde el árbol de características; estas se pueden instalar individualmente o seleccionar una combinación de ellas. En el proceso de instalación se presentan las siguientes pantallas:

- Seleccione el tipo de instalación: **Básica**, **Personalizado** o **Descargar medios**.



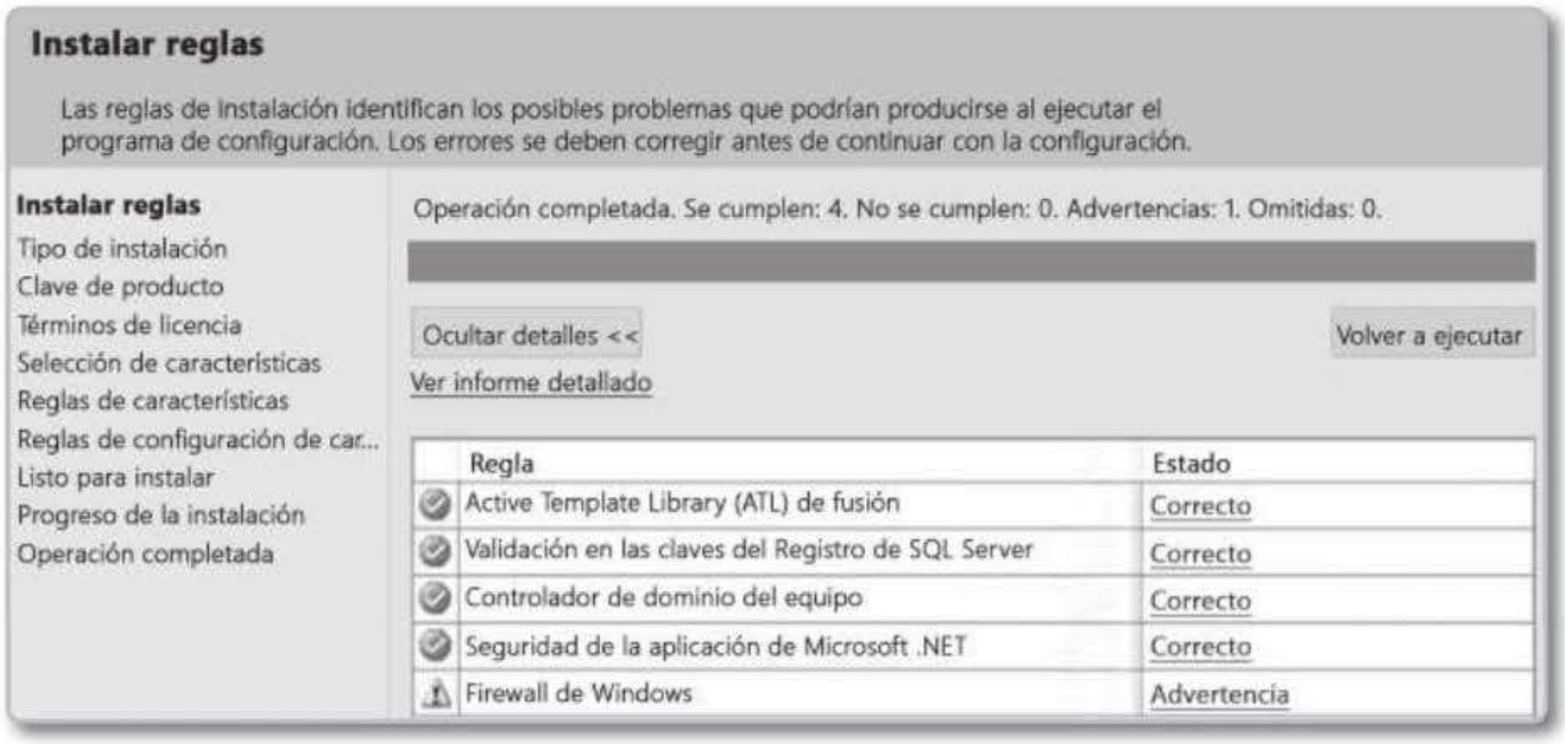
- Luego, desde el asistente, seleccione **New SQL Server stand-alone installation or add features to an existing installation** 'Nueva instalación independiente de SQL Server'.



- Con respecto a las actualizaciones, en **Microsoft Update** se recomienda que no active el *check* del control y continúe con la instalación.



- En **Instalar reglas**, actívelas todas, pues la instalación se detendrá si no se cumple con una de ellas.



- En **Tipo de instalación**, seleccione **Realizar una nueva instalación de SQL Server 2016** y continúe con la instalación.



- En la solicitud de **Clave de producto**, seleccione **Developer** porque en la revisión que se realiza al inicio de este capítulo se indica que Developer es una versión gratuita.



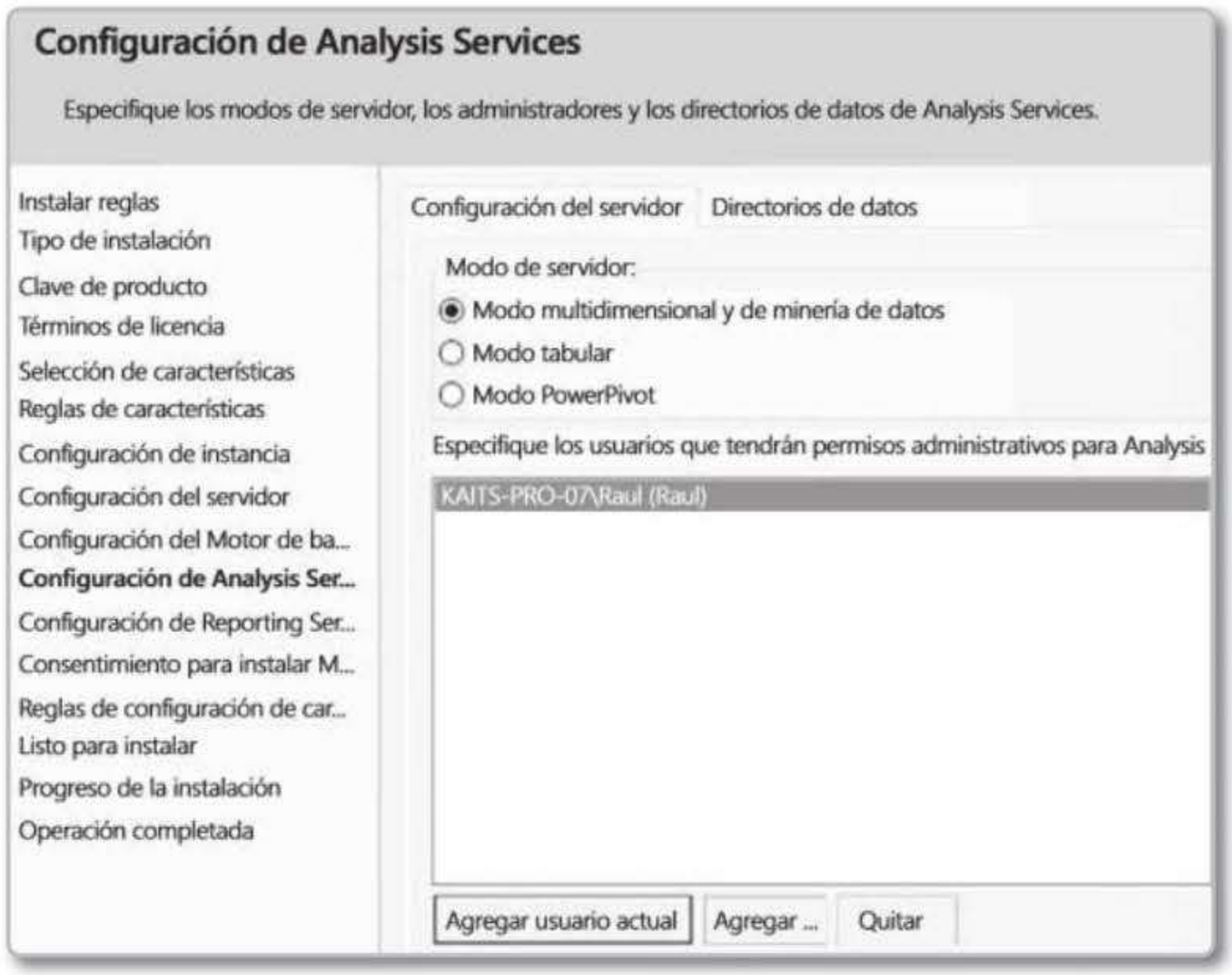
- En **Términos de licencia**, solo active **Acepto los términos de licencia**.



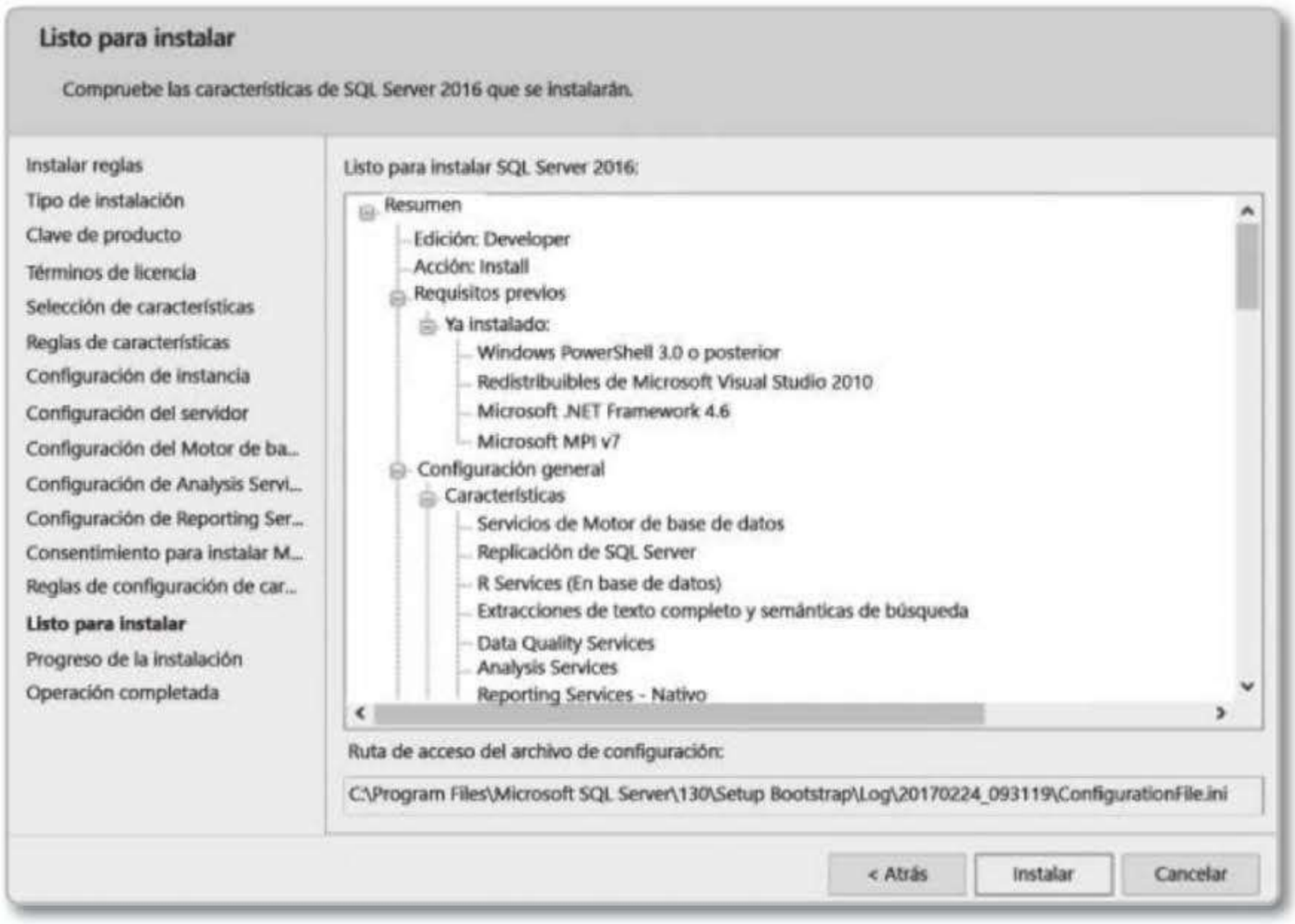
- Con respecto a la **Selección de características**, el mismo asistente marcará los check necesarios para su instalación; mientras que, en **Configuración de instancia**, se recomienda que seleccione la opción **Instancia predeterminada**.



- Con respecto a la **Configuración de Analysis Services** (Configuración de servicio de análisis), solo presione el botón **Agregar usuario actual**; el **Modo de servidor** puede ser **Modo multidimensional y de minería de datos**, ya que agrupa el trabajo de modo tabular y PowerPivot. Así, continúe con la instalación.

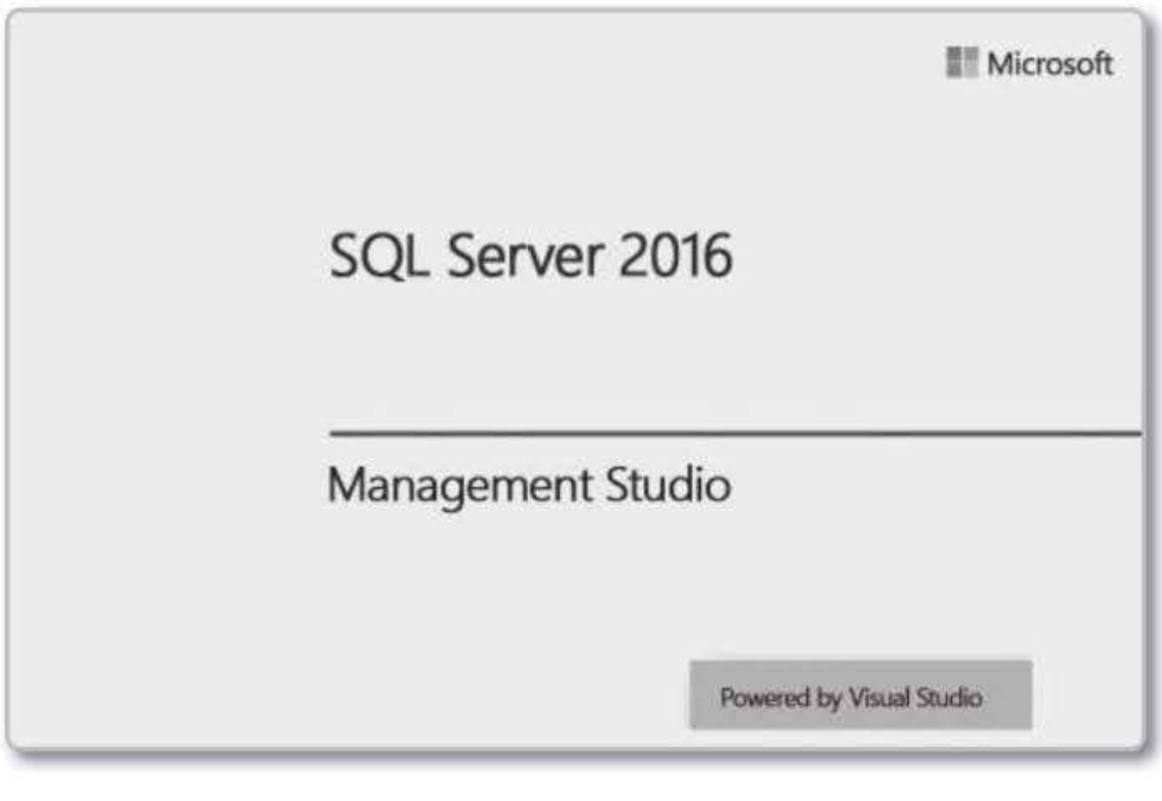


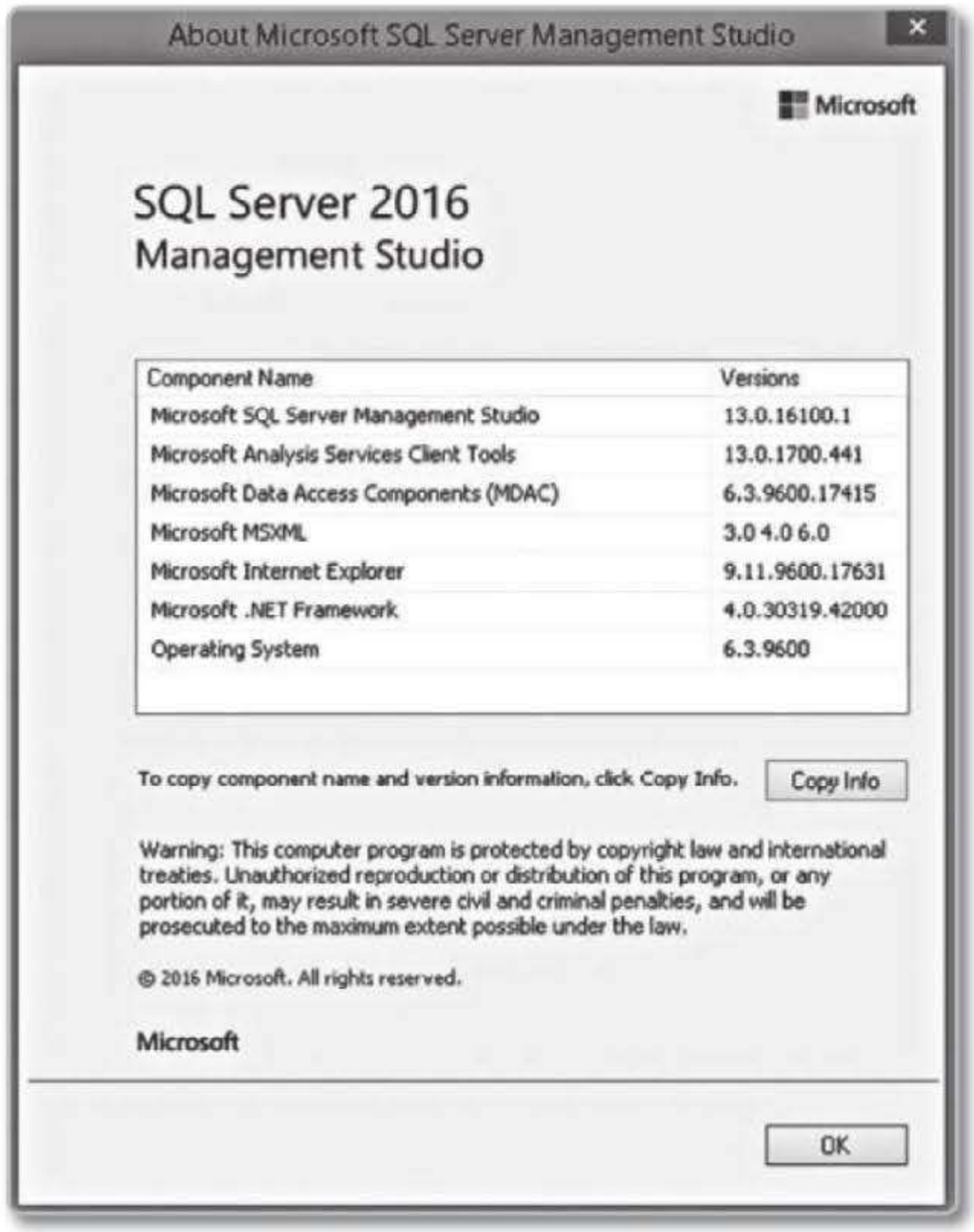
- Finalmente, para iniciar la instalación de componentes presione el botón **Instalar**.



2.6 SQL Management Studio

Es un entorno integrado que permite administrar y gestionar todos los componentes instalados por SQL Server 2016. Es una integración de herramientas gráficas con una serie de editores de scripts que permiten acceder a SQL Server. Para descargar la versión de SQL Management Studio, use la siguiente URL: <https://msdn.microsoft.com/es-es/library/mt238290.aspx>.





Al iniciar el SQL Management Studio se presenta la ventana de conexión al servidor, desde la cual se podrá acceder a todos los servicios de SQL Server.



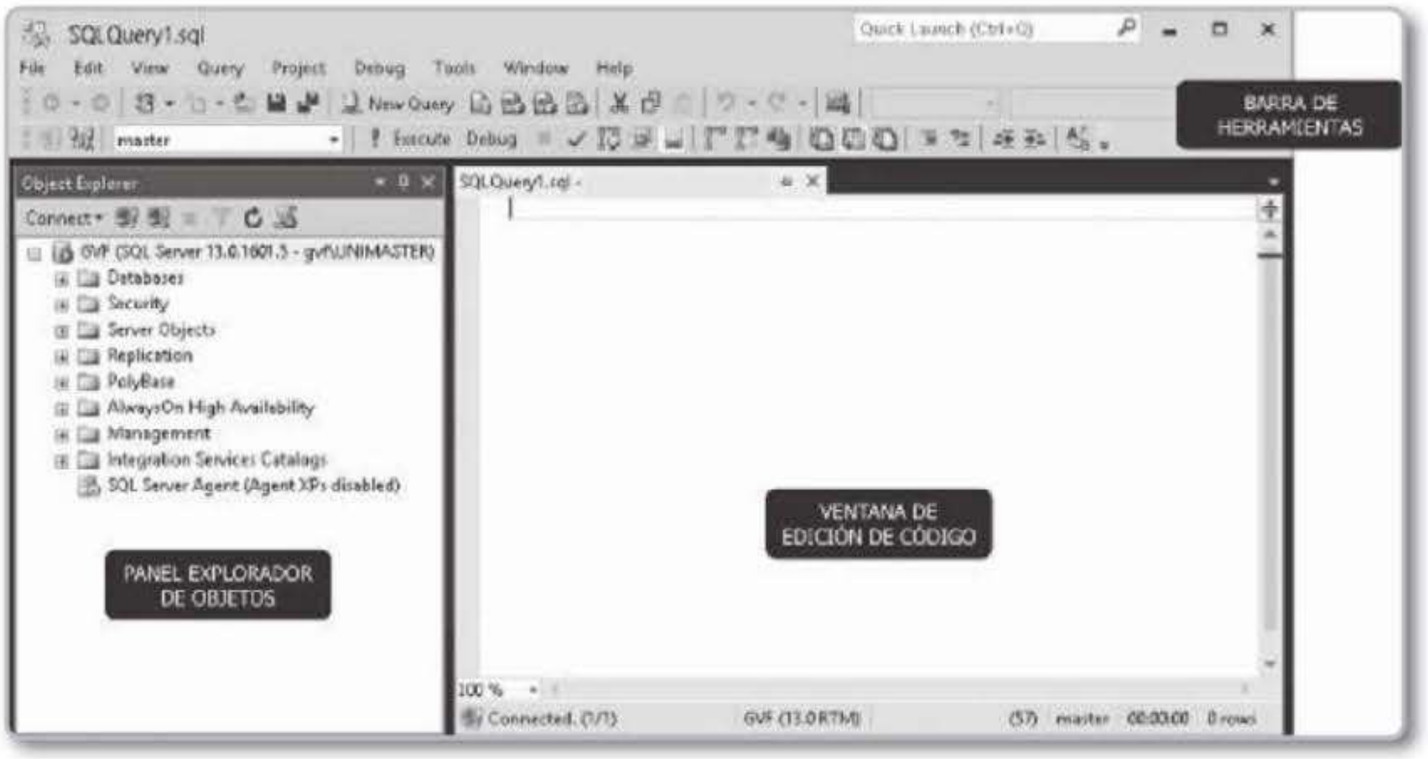
El modo de autenticación se puede realizar de dos maneras:

- **Windows Authentication:** Permite identificar a un usuario de la base de datos por medio de inicio de sesión de Windows; por tanto, no solicitará nombre de usuario o clave.

- **SQL Server Authentication:** Permite identificar a un usuario de la base de datos por medio de un nombre de usuario y una clave. Esta forma de autenticación es muy estricta con respecto a la administración de la información que se muestra a los usuarios.

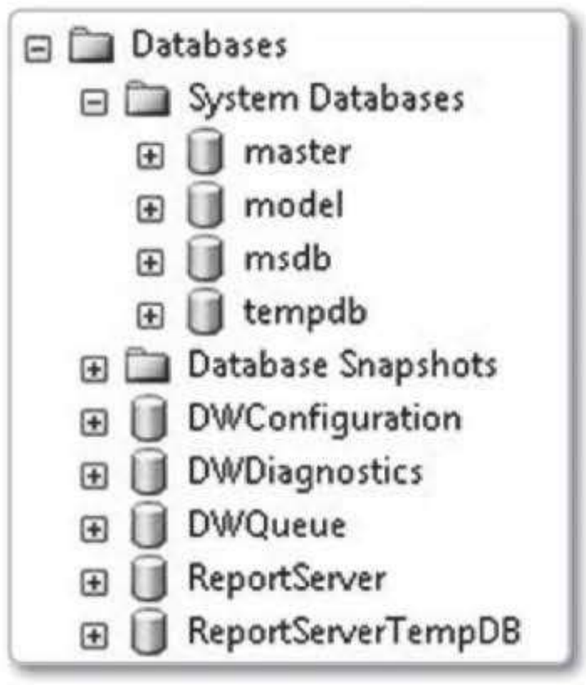
2.6.1 Pantalla inicial del entorno

La pantalla inicial del entorno se muestra de la siguiente manera:



Panel explorador de objetos:

Presenta en forma de listado todos los objetos contenidos en el servidor. Se debe prestar especial atención en la carpeta Databases, ya que allí se encontrarán las bases de datos del sistema y las que se implementarán.



Inicialmente, las bases de datos que se presentan son Master, Model, Msdb y Tempdb; es a partir de Master que se podrá crear nuevas bases de datos usando como plantilla la base de datos Model.

A continuación, se ve algunas opciones presentadas dentro del Explorador de Objetos:

- Para conectarse a un nuevo servidor, se debe seleccionar **Connect > Database Engine...**
- Si se quiere visualizar las bases de datos del sistema, se debe seleccionar **Databases > System Databases...**
- Para visualizar los procedimientos almacenados en la base de datos, se debe seleccionar **Databases > NombreBaseDatos > Programmability > Stored Procedures**.

Barra de Herramientas

Presenta controles que permiten administrar el código SQL.



master	Muestra una lista de base de datos que pueden ser activadas por el usuario; al seleccionar una base de datos desde este control esta quedará activa.
Execute	Las sentencias implementadas en SQL Server pueden ejecutarse presionando el botón **Execute** o la tecla **<F5>**.
	Las sentencias implementadas en SQL Server se pueden verificar en el servidor mediante **<Ctrl>** + **<F5>**.
	Cuando se ejecuta una sentencia, el resultado se puede mostrar en tres entornos; por ejemplo, si se ejecuta la sentencia: **SELECT * FROM SYS.SYSDATABASES** **GO** Se ejecuta la sentencia presionando la tecla **<F5>** y, para las formas de visualizaciones, se cuenta con las siguientes alternativas: **- Texto** (<Ctrl> + <T>): Se muestran los resultados parecidos a la salida por consola.

Results

```
tempdb
model
msdb
ReportServer
ReportServerTempDB

(6 row(s) affected)
```

- **Grilla** (<Ctrl> + <D>): Presenta los resultados en forma de cuadrículas; esta forma de mostrarse es la más común.

	name	dbid	sid
1	master	1	0x01
2	tempdb	2	0x01
3	model	3	0x01
4	msdb	4	0x01
5	ReportServer	5	0x01050
6	ReportServerTempDB	6	0x01050

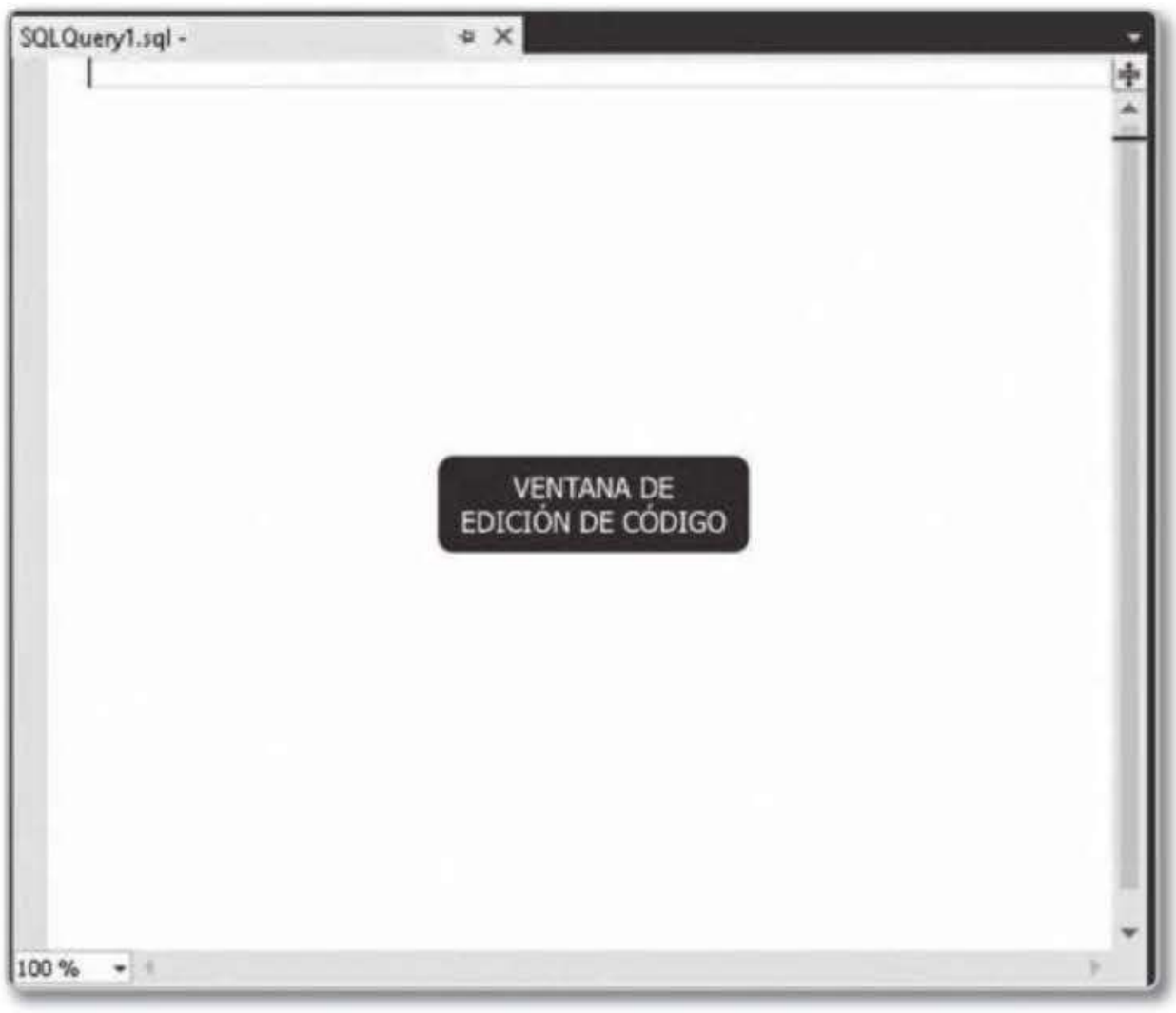
- **Reporte:** Permite grabar, en forma de reporte, los resultados obtenidos. La extensión de este archivo es RPT.

	Permiten colocar y eliminar asignación de comentarios sobre instrucciones seleccionadas por el usuario.

Ventana de edición de código

SQL Server se caracteriza por implementar scripts dentro del editor de código, ya que desde allí se podrá tener acceso a todos los objetos de una base. Gracias a los comandos que se pueden implementar, desde versiones anteriores al SQL Server 2016, se viene utilizando la administración de ficheros; eso quiere decir que se podrán implementar scripts desde diferentes hojas del editor de consultas.

Para agregar una nueva hoja de edición, se debe seleccionar el botón **New Query** desde la barra de herramientas o desde el menú **File > New > Database Engine Query**.

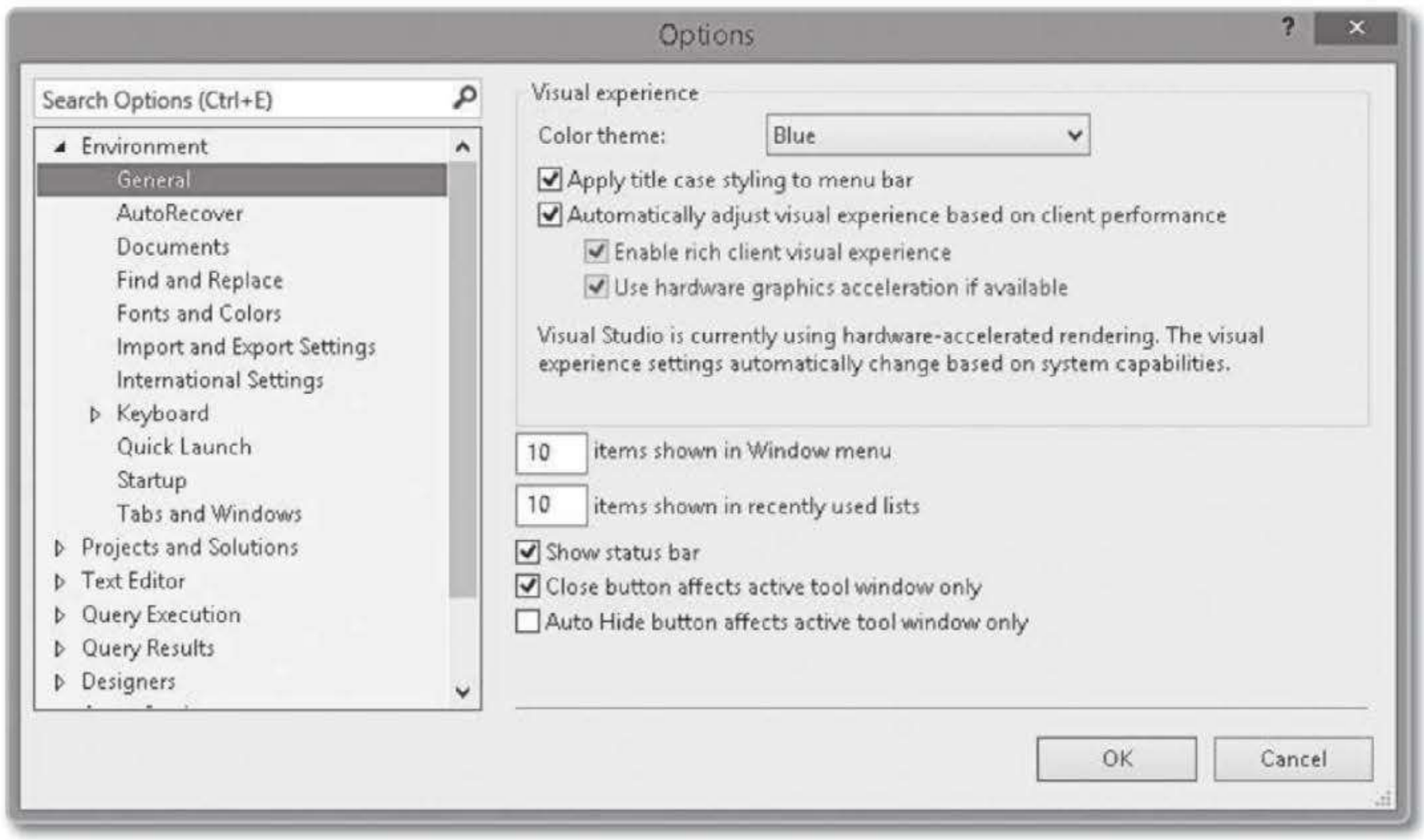


2.6.2 Configuración de fuente para el entorno de trabajo

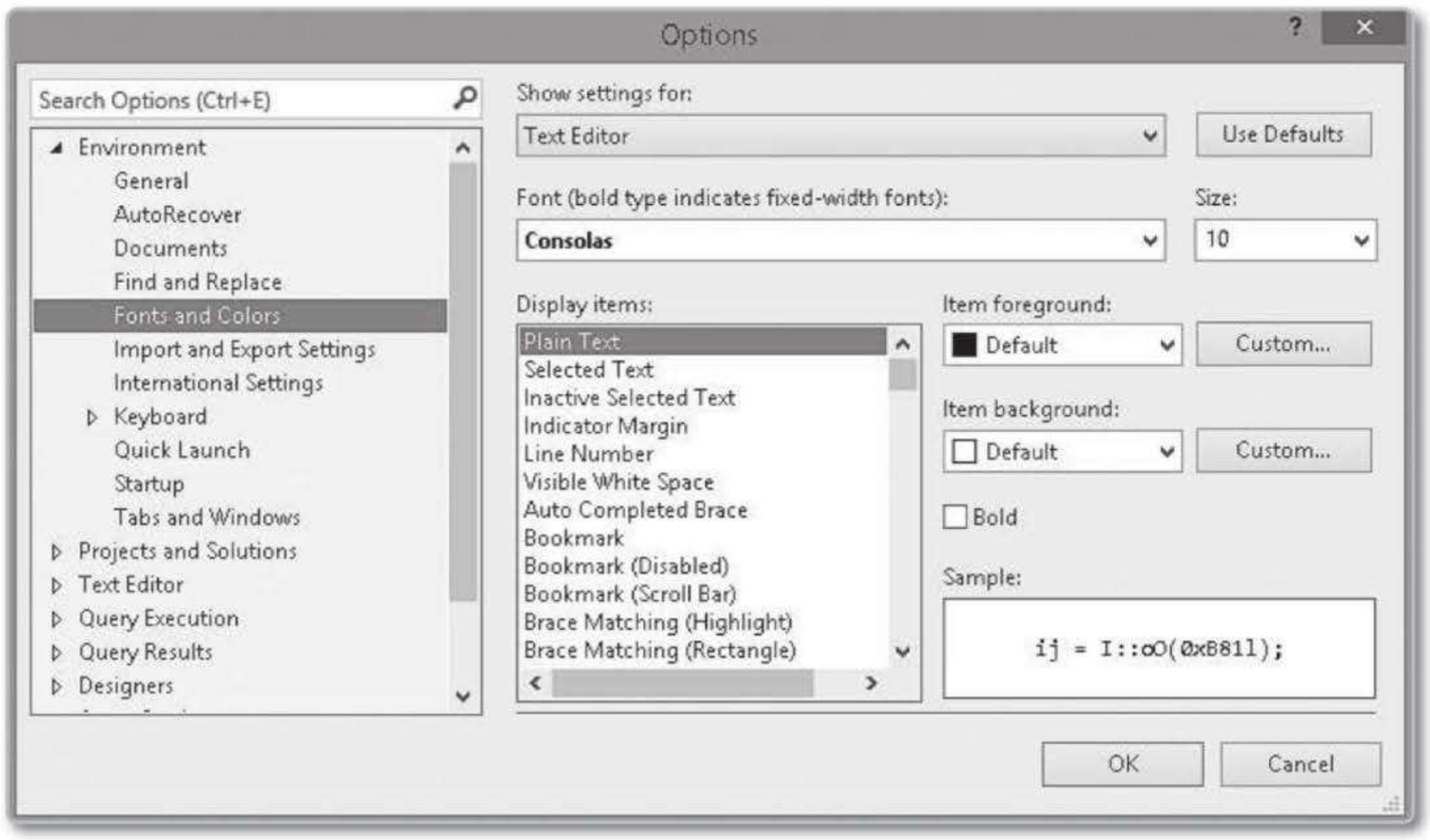
Transact-SQL se caracteriza por generar procesos en una base de datos por medio de scripts que se ejecutan dentro de un editor de consultas. En muchas ocasiones, es posible confundir algunos operadores símbolos en los scripts; por ejemplo, la letra *o* con el número cero (O - 0), ambos pueden ocasionar errores lógicos cuando se ejecute algún script; por lo tanto, se recomienda que use la fuente Consolas, con un tamaño establecido por el usuario para el mejor desempeño de los scripts dentro de Transact-SQL. A continuación, se observa el cambio entre la *o* y el cero (O- 0).

Acceso:

- En herramientas, seleccione **Tools > Options** (Opciones).



- Seleccione **Fonts and Colors**.
- Cambie el tipo de fuente. En **Font** seleccione **Consolas** con un tamaño adecuado.



- Presione **OK**.

2.7 Las bases de datos en SQL Server

Al instalar SQL Server también se registran bases de datos que administran diferentes aspectos del servidor de base de datos. SQL cuenta con las siguientes bases de datos del sistema:

Master	Master es el núcleo de toda instancia en SQL Server; eso quiere decir que esta base contiene información vital sobre una instancia de SQL Server. Se compone de tablas de sistema que registran el seguimiento de la instalación del servidor y de la administración de todas las bases de datos que se generen posteriormente. La información que contiene es bases de datos, opciones de configuración, triggers a nivel de servidor y servidores con enlace.
Tempdb	Es una base de datos temporal; técnicamente es informal para SQL Server y su uso es, generalmente, cuando se realiza operaciones de clasificación y agregación. En ocasiones, las aplicaciones usan las tablas, los procesos y los cursores de esta tabla, ya que provee de un espacio de trabajo temporal y tiene la capacidad de regenerarse cada vez que arranca SQL Server.
Model	Model es una plantilla para todas las bases de datos creadas en el servidor. Se podrá implementar cuando ejecute el comando CREATE DATABASE, dando lugar a la creación de una nueva base de datos basada en un modelo predefinido por SQL Server; este le añade páginas vacías que serán propias de la nueva base de datos.
Msdb	Es empleada por el servicio SQL Server Agent y por los servicios de integración de SQL Server en la cual guarda información con respecto a tareas de automatización, historial de copias de seguridad, información de rastreo de registros, tareas, alertas, cuentas proxy, planes de mantenimiento y registros de mensajería de la base de datos.

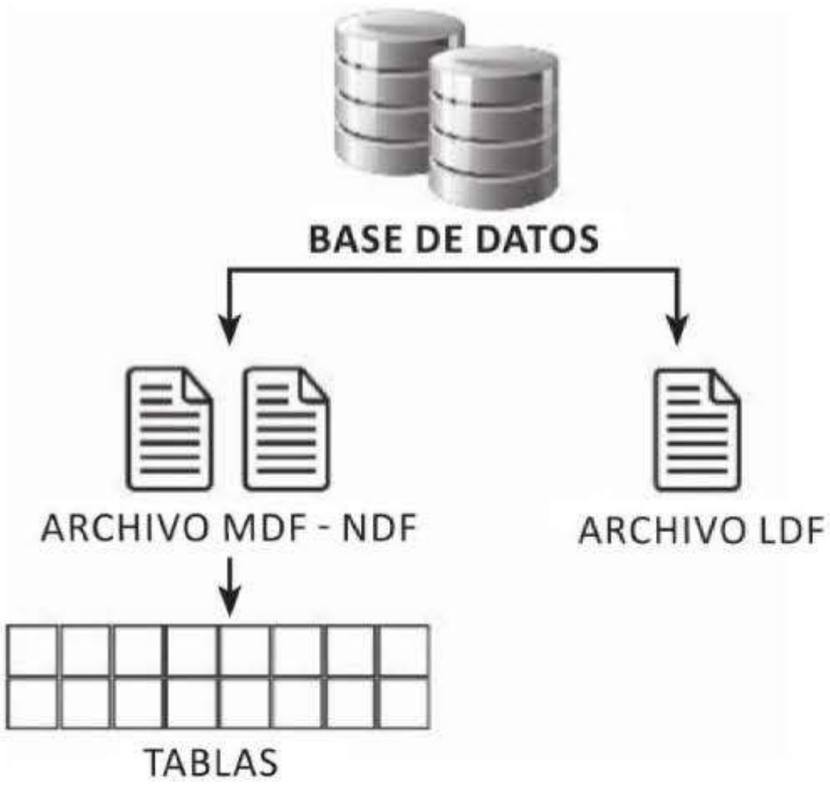
2.8 Componentes de una base de datos

Cuando se crea una base de datos se debe tener en cuenta el nombre de la base de datos, el propietario (el usuario que crea la base de datos), su tamaño, archivos y grupos de archivos.

Antes de crear una base de datos, se debe considerar lo siguiente:

- De forma predeterminada, tienen permiso para crear una base de datos las funciones fijas del servidor sysadmin y dbcreator, aunque se puede otorgar permisos a otros usuarios.
- El usuario que crea la base de datos se convierte en su propietario.
- En un servidor, pueden crearse hasta 32 767 bases de datos.

Se utilizan tres tipos de archivos para almacenar una base de datos:



2.8.1 Archivo MDF

También llamado archivo principal o master; por cada base de datos solo se tiene un archivo MDF y este puede contener toda la información referente a la base de datos; dicha información se almacena en tablas.



2.8.2 Archivo NDF

También llamado archivo secundario; estos archivos contienen todos los datos que no caben en el archivo MDF. No es necesario que las bases de datos tengan archivos de datos secundarios si el archivo principal es lo suficientemente grande como para contener todos los datos.



2.8.3 Archivo LDF

También llamado archivo de transacción; contiene la información interna como la fecha de creación y otras características propias de la base de datos. El tamaño mínimo para un archivo de transacciones es de 512 *kilobytes* (KB).



2.9 Motor de base de datos

Es el servicio principal de SQL Server; se usa para crear las bases de datos, tablas, almacenar información y crear objetos de base de datos como índices, vistas, procedimientos almacenados, funciones, triggers. SQL Server usa el Management Studio para dicha administración.

Principalmente, el Motor de base de datos permite:

- Diseñar y crear una base de datos que contenga las tablas relacionales o los documentos XML que el sistema necesita.
- Implementar sistemas para obtener acceso y cambiar los datos almacenados en la base de datos, lo que incluye implementar los sitios web o las aplicaciones que funcionan con los datos, así como crear procedimientos que utilicen las herramientas y utilidades de SQL Server para trabajar con los datos.
- Aplicar los sistemas implementados en la organización o en los clientes.
- Proporcionar soporte técnico administrativo diario para optimizar el rendimiento de la base de datos.

2.10 Sentencias DDL

Son sentencias del lenguaje de definición de datos, el cual permite gestionar objetos dentro de la base de datos como la creación, modificación y eliminación. Se cuenta con las siguientes sentencias:

2.10.1 Create

Sentencia que permite crear un determinado objeto en la base de datos e, inclusive, crear la base de datos. Algunos casos de creación de objetos son:

- Crear una base de datos CREATE DATABASE
- Crear una tabla CREATE TABLE
- Crear un procedimiento almacenado CREATE PROCEDURE
- Crear una función CREATE FUNCTION
- Crear un disparador CREATE TRIGGER

2.10.2 Alter

Sentencia que permite modificar la estructura de datos de un objeto de la base de datos e, inclusive, modificar la base de datos. Algunas modificaciones sobre objetos son:

- Modificar una base de datos ALTER DATABASE
- Modificar una tabla ALTER TABLE
- Modificar un procedimiento almacenado ALTER PROCEDURE
- Modificar una función ALTER FUNCTION
- Modificar un disparador ALTER TRIGGER

2.10.3 Drop

Sentencia que permite eliminar un determinado objeto de la base de datos e, inclusive, eliminar la base de datos. Algunos casos de eliminación de objetos son:

- Eliminar una base de datos DROP DATABASE
- Eliminar una tabla DROP TABLE
- Eliminar un procedimiento almacenado DROP PROCEDURE
- Eliminar una función DROP FUNCTION
- Eliminar un disparador DROP TRIGGER

2.11 Sentencias DML

Son sentencias del lenguaje de manipulación de datos que permiten administrar la información contenida en una tabla de datos; así, se tiene la inserción, modificación, eliminación y consulta de registros.

2.11.1 Insert

Sentencia que permite insertar registros en una tabla de base de datos.

2.11.2 Update

Sentencia que permite modificar registros contenidos en una tabla de base de datos.

2.11.3 Delete

Sentencia que permite eliminar registros contenidos en una tabla de base de datos.

2.11.4 Select

Sentencia que permite mostrar información en forma de registros desde una tabla.

2.12 Creación de una base de datos

Una base de datos en SQL Server es una réplica de la base de datos Model, ya que esta base proporciona páginas vacías en las que se puede crear archivos lógicos dentro de la base de datos para una implementación. Úsese la sentencia DDL Create para la creación de la base de datos, la cual cuenta con el siguiente formato:

Básico:

```
CREATE DATABASE <NOMBREBASEDATOS>
GO
```

Especificación de archivo maestro:

```
CREATE DATABASE <NOMBREBASEDATOS>
ON PRIMARY (
NAME ='NOMBREARCHIVOLOGICO',
FILENAME ='NOMBREARCHIVOFISICO',
SIZE =TAMAÑOENDISCO,
MAXSIZE =MAXIMOTAMAÑO,
FILEGROWTH =FACTORDECRECIMIENTO
)
GO
```

Especificación de archivo secundario:

```
CREATE DATABASE <NOMBREBASEDATOS>
ON PRIMARY (
NAME ='NOMBREARCHIVOLOGICO',
FILENAME ='NOMBREARCHIVOFISICO',
SIZE =TAMAÑOENDISCO,
MAXSIZE =MAXIMOTAMAÑO,
FILEGROWTH =FACTORDECRECIMIENTO
), (
NAME ='NOMBREARCHIVOLOGICO',
FILENAME ='NOMBREARCHIVOFISICO',
SIZE =TAMAÑOENDISCO,
MAXSIZE =MAXIMOTAMAÑO,
FILEGROWTH =FACTORDECRECIMIENTO
)
```

Especificación de archivo de transacciones:

```
CREATE DATABASE <NOMBREBASEDATOS>
ON PRIMARY (
NAME ='NOMBREARCHIVOLOGICO',
FILENAME ='NOMBREARCHIVOFISICO',
SIZE =TAMAÑOENDISCO,
MAXSIZE =MAXIMOTAMAÑO,
FILEGROWTH =FACTORDECRECIMIENTO
)
LOG ON (
NAME ='NOMBREARCHIVOLOGICO',
FILENAME ='NOMBREARCHIVOFISICO',
SIZE =TAMAÑOENDISCO,
MAXSIZE =MAXIMOTAMAÑO,
FILEGROWTH =FACTORDECRECIMIENTO
      )
```

Donde:

- NOMBREBASEDATOS: Es el nombre de la nueva base de datos; deben ser únicos en un servidor y pueden tener hasta 128 caracteres, a menos que no se especifique ningún nombre lógico para el registro. Si no se especifica ningún nombre, SQL genera un nombre lógico al anexar un sufijo a NombreBaseDatos.
- ON PRIMARY: Especifica que la lista de archivos está asociada al grupo principal. Este grupo contiene todas las tablas del sistema de base de datos. También contiene todos los objetos no asignados a los grupos de archivos de usuario.

- LOG ON: Especifica que los archivos de registro de la base de datos (archivo de registro) se han definido explícitamente. La palabra clave va seguida de una lista delimitada por comas, la cual define las características de los archivos de registro.
- NAME: Especifica el nombre lógico del archivo. Este archivo es utilizado para referenciar al archivo en las sentencias del Transact-SQL que se ejecutan después.
- FILENAME: Especifica el nombre de archivo en el sistema operativo. Se debe precisar la ruta de acceso y el nombre del archivo que el sistema operativo reconocerá; por ejemplo, MDF es reconocido como archivo principal de base de datos. La ruta de acceso debe especificar un directorio en el servidor.
- SIZE: Especifica el tamaño para el archivo. Cuando este parámetro no es especificado para un archivo de registro, SQL le asigna automáticamente 1MB, el mínimo predeterminado.
- MAXSIZE: Especifica el tamaño máximo de crecimiento del archivo; se pueden utilizar sufijos KB y MB; el valor predeterminado es MB; solo se pueden especificar números enteros.
- FILEGROWTH: Especifica el incremento de crecimiento del archivo; este valor no puede exceder el valor de MAXSIZE. Se debe emplear un número entero. Un valor 0 indica que no hay creciminento. El valor se puede especificar en MB, KB o %. El valor predeterminado es de 10%.

Por buenas prácticas siempre será bueno validar la existencia de la base de datos usando la siguiente sentencia:

```
IF DB_ID('<NOMBREBASEDATOS>')IS NOT NULL
DROP DATABASE <NOMBREBASEDATOS>
GO
```

La sentencia IF permite condicionar la existencia de la base de datos siempre y cuando la función DB_ID emita el valor FALSE; se le asigna una comparación IS NOT NULL para validar la existencia de la base de datos. Entonces, se podría decir que, si la base de datos existe, se ejecuta la sentencia DROP DATABASE que eliminará la base.

2.13 Modificación de archivos de una base de datos

Permite agregar, modificar o eliminar archivos contenidos en la base de datos. Se usará la sentencia DDL Alter para la modificación o alteración de los archivos contenidos en una base de datos; se cuenta con el siguiente formato:

Agregar un archivo:

```
ALTER DATABASE <NOMBREBASEDATOS>
ADD FILE(
 NAME=<NOMBRELOGICOARCHIVO>,
 FILENAME=<NOMBREFISICOARCHIVO>,
 SIZE=<TAMAÑO>, MAXSIZE=<TAMAÑO>, FILEGROWTH=<TAMAÑO>
)
GO
```

Modificar las propiedades de los archivos:

```
ALTER DATABASE <NOMBREBASEDATOS>
 MODIFY FILE (
  NAME=<NOMBREARCHIVO>,
  PROPIEDAD=VALOR
)
GO
```

Eliminar archivos:

```
ALTER DATABASE <NOMBREBASEDATOS>
 REMOVE FILE <NOMBREARCHIVO>
GO
```

2.14 Activar la base de datos

Cuando se ejecutan sentencias, estas siempre se realizan sobre una base de datos activa. Eso quiere decir que dichas sentencias afectarán toda vez a los objetos contenidos en la base de datos. El formato de activación es:

```
USE <NOMBREBASEDATOS>
GO
```

Cuando se crea una base de datos, siempre se inicia con la activación de la base de datos maestra, con la sentencia USE MASTER.

2.15 Los tipos de datos en SQL Server 2016

SQL brinda una serie de datos para almacenar la información; la correcta selección del tipo de dato implica un determinado valor a almacenar, por ejemplo: Carácter, Enteros, Binario, Fechas, etc.

Carácter: Es una combinación de letras, número y caracteres. Se tiene los siguientes tipos:

Char	Tipo de datos carácter de longitud fija; tiene como requisito indispensable la especificación del total de caracteres permitidos. Se menciona algunas definiciones: **DNI CHAR(8)** Especifica que el campo DNI solo permitirá un máximo de ocho caracteres, como, por ejemplo, 09899445. **CATEGORÍA CHAR(1)** Especifica que el campo CATEGORÍA solo permitirá el registro de un carácter, como, por ejemplo, la categoría de un empleado A, B, C o D. **TELÉFONO CHAR(15)** Especifica que el campo TELÉFONO solo permitirá quince caracteres, entre números y caracteres especiales, como, por ejemplo, (051)-563-8958. **SEXO CHAR(1)** Especifica que el campo SEXO solo permitirá el registro de un carácter, como, por ejemplo, M (masculino) o F (femenino).

Varchar	Tipo de datos de longitud variable, es decir, el valor asignado no es fijo. Aquí SQL administra los espacios en blanco y los optimiza, haciendo que el campo se comporte dinámicamente con respecto al valor de la columna. Se menciona algunas definiciones: **DIRECCIÓN VARCHAR(50)** Especifica que el campo DIRECCIÓN permitirá el registro de un máximo de 50 caracteres como por ejemplo "Av. Las Palmeras 6345 Urb. Villa Norte – Los Olivos". **EMAIL VARCHAR(60)** Especifica que el campo EMAIL permitirá el registro de un máximo de 60 caracteres, como, por ejemplo, "manuel.torresr@gmail.com".
Text	Tipo de datos de longitud extensa el cual permite almacenar información masiva inclusive que excedan los 8KB. **COMENTARIO TEXT** Especifica que el campo COMENTARIO permitirá el registro de un texto masivo.

Fecha: Los datos fecha y hora consisten en combinaciones válidas de estos datos. Se tiene los siguientes tipos:

DateTime	Fechas en el Rango del 01 Enero del 1753 al 31 Diciembre del 9999. 08-05-2007 12.35:29.123 **FECHA DATETIME** Especifica que el campo FECHA acepta valores de tipo fecha y hora en el mismo campo.
SmallDateTime	Fechas en el Rango del 01 Enero 1900 al 06 Junio 2079. 08-05-2007 12:35.00 **FECHA SMALLDATETIME** Especifica que el campo FECHA acepta valores de tipo fecha y hora en el mismo campo con un rango corto.
DateTime2	08-05-2007 12:35:29.1234567 **FECHA DATETIME2** Especifica que el campo FECHA acepta valores de tipo fecha y hora en el mismo campo con un rango de milisegundos ampliado.
Time	12:35:29.1234567 **HORA_ENTRADA TIME** Especifica que el campo HORA_ENTRADA acepta solo valores de tipo hora.
Date	08-05-2007 **FECHA DATE** Especifica que el campo FECHA acepta solo valores de tipo fecha.

Número decimal: Consiste en almacenar valores numéricos con fracción decimal. Se tiene los siguientes tipos:

Decimal	Los datos pueden tener hasta 38 dígitos, podrán estar en el lado derecho del punto decimal. **PROMEDIO DECIMAL(5,2)** Especifica que el campo PROMEDIO almacena información con dos valores numéricos y dos valores decimales.
Numeric	Es equivalente a Decimal.

Número entero: Consiste en almacenar valores numéricos sin fracción decimal. Se tiene los siguientes tipos:

BigInt	De -2 ^ 63 (-9.223.372.036.854.775.808) a 2 ^ 63-1 (9.223.372.036.854.775.807) **DISTANCIA BIGINT** Especifica que el campo DISTANCIA almacena información de valores numéricos enteros de gran volumen.
Int	De -2 ^ 31 (-2.147.483.648) a 2 ^ 31-1 (2.147.483.647) **EVALUACIÓN INT** Especifica que el campo EVALUACIÓN almacena información de valores numéricos enteros de capacidad mediana.
Smallint	De -2 ^ 15 (-32.768) a 2 ^ 15-1 (32.767) **EDAD SMALLINT** Especifica que el campo EDAD almacena información de valores numéricos enteros de capacidad corta.
Tinyint	De 0 a 255 **OPCIÓN TINYINT** Especifica que el campo OPCIÓN almacena información de valores numéricos enteros de capacidad mínima.

Monetario: Consiste en almacenar valores de tipo moneda. Se tiene los siguientes tipos:

Money	De -922,337,203,685.477,5808 a 922,337,203,685.477,5807 **SUELDO MONEY** Especifica que el campo SUELDO almacena información de valores monetarios largos.
Smallmoney	De - 214.748,3648 a 214.748,3647 **PRECIO SMALLMONEY** Especifica que el campo PRECIO almacena información de valores monetarios cortos.

2.16 Tipos de datos definidos por el usuario

SQL nos permite agregar nuevos tipos de datos llamados "Tipos de datos definidos por el usuario", los cuales podrán ser usados como un tipo de datos dentro de la creación o modificación de una tabla de base de datos. Asimismo, se debe mencionar que un tipo de datos definido usará los tipos de datos nativos de SQL.

Formato para agregar un tipo definido:

```
SP_ADDTYPE NOMBRE_TIPO,'TIPO_DATOS','RESTRICCION'
GO
```

Formato para eliminar un tipo definido:

```
SP_DROPTYPE 'NOMBRE_TIPO'
GO
```

A continuación, se visualizan los tipos de datos permitidos dentro de la base de datos SALES. En este caso se usa las siguientes sentencias:

```
SELECT * FROM SYS.SYSTYPES
GO
```

	name	xtype	status	xusertype	length
1	image	34	0	34	16
2	text	35	0	35	16
3	uniqueidentifier	36	0	36	16
4	date	40	0	40	3
5	time	41	0	41	5
6	datetime2	42	0	42	8
7	datetimeoffset	43	0	43	10
8	tinyint	48	0	48	1
9	smallint	52	0	52	2
10	int	56	0	56	4
11	smalldatetime	58	0	58	4
12	real	59	0	59	4
13	money	60	0	60	8
14	datetime	61	0	61	8
15	float	62	0	62	8
16	sql_variant	98	0	98	8016
17	ntext	99	0	99	16
18	bit	104	0	104	1
19	decimal	106	0	106	17
20	numeric	108	0	108	17
21	smallmoney	122	0	122	4
22	bigint	127	0	127	8
23	hierarchyid	240	0	128	892
24	geometry	240	0	129	-1
25	geography	240	0	130	-1
26	varbinary	165	0	165	8000
27	varchar	167	0	167	8000
28	binary	173	0	173	8000
29	char	175	0	175	8000
30	timestamp	189	1	189	8
31	nvarchar	231	0	231	8000
32	nchar	239	0	239	8000
33	xml	241	0	241	-1
34	sysname	231	1	256	256

Con la siguiente sentencia, se puede agregar el tipo de datos DNI de ocho caracteres y con la restricción de campo obligatorio:

```
SP_ADDTYPE DNI,'CHAR(8)','NOT NULL'
GO
```

Si se ejecuta la sentencia SELECT * FROM SYS.SYSTYPES, el resultado se mostraría de la siguiente manera:

34	sysname	231	1	256	256
35	DNI	175	1	257	8

Caso desarrollado 1: Base de datos básica

Cree la base de datos VENTAS de forma predeterminada. Tenga en cuenta lo siguiente:

- Visualizar la base de datos mediante un listado.
- Mostrar los archivos que componen la base de datos.

Solución:

```
--Activamos la base de datos maestra
USE MASTER
GO

--Validando la existencia de la base de datos
IF DB_ID('VENTAS')IS NOT NULL
        DROP DATABASE VENTAS
GO

--Creando la base de datos VENTAS
CREATE DATABASE VENTAS
GO
```

Visualizar la base de datos mediante un listado:

```
SELECT * FROM SYS.sysdatabases S
WHERE S.name='VENTAS'
GO
```

	name	dbid	sid	mode	status	status2	crdate
1	VENTAS	54	0x0105000000000005150000065A4682F59DF37CAB3D91B...	0	65536	1627389952	2017-03-09 17:57:46.257

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS	1	C:\Program Files\Microsoft SQL Server\MSSQL11.PC\MSSQL\DATA\VENTAS.mdf	PRIMARY	4160 KB	Unlimited	1024 KB	data only
2	VENTAS_log	2	C:\Program Files\Microsoft SQL Server\MSSQL11.PC\MSSQL\DATA\VENTAS_log.ldf	NULL	1040 KB	2147483648 KB	10%	log only

Caso desarrollado 2: Base de datos con especificación de archivo primario

Cree la base de datos VENTAS en la carpeta C:\VENTAS. Tenga en cuenta lo siguiente:

- Archivo primario con un tamaño inicial de 50MB, máximo de 150MB y una tasa de crecimiento de 20%.
- Visualizar la base de datos mediante un listado.
- Mostrar los archivos que componen la base de datos.

Solución:

Recuerde que antes de ejecutar las sentencias, debe crear la carpeta VENTAS en la unidad C:\

```
--Activamos la base de datos maestra
USE MASTER
GO
--Validando la existencia de la base de datos
IF DB_ID('VENTAS')IS NOT NULL
       DROP DATABASE VENTAS
GO
--Creando la base de datos VENTAS
CREATE DATABASE VENTAS
ON PRIMARY(
       NAME='VENTAS',
       FILENAME='C:\VENTAS\VENTAS.MDF',
       SIZE=50MB,
       MAXSIZE=150MB,
       FILEGROWTH=20%
)
GO
```

Visualizar la base de datos mediante un listado:

```
SELECT * FROM SYS.sysdatabases S
WHERE S.name='VENTAS'
GO
```

	name	dbid	sid	mode	status	status2	crdate
1	VENTAS	54	0x0105000000000005150000065A4682F59DF37CAB3D91B...	0	65536	1627389952	2017-03-09 18:19:55.897

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS	1	C:\VENTAS\VENTAS.MDF	PRIMARY	51200 KB	153600 KB	20%	data only
2	VENTAS_log	2	C:\VENTAS\VENTAS_log.ldf	NULL	12800 KB	2147483648 KB	10%	log only

Caso desarrollado 3: Base de datos con especificación de archivos secundarios

Cree la base de datos VENTAS en la carpeta C:\VENTAS. Tenga en cuenta lo siguiente:

- Archivo primario con un tamaño inicial de 50MB, máximo de 150MB y una tasa de crecimiento de 20%.
- Archivo secundario con un tamaño inicial de 10MB, máximo de 50MB y una tasa de crecimiento de 2MB.
- Visualizar la base de datos mediante un listado.
- Mostrar los archivos que componen la base de datos.

Solución:

Recuerde que antes de ejecutar las sentencias, debe crear la carpeta VENTAS en la unidad C:\

```
--Activamos la base de datos maestra
USE MASTER
GO
--Validando la existencia de la base de datos
IF DB_ID('VENTAS')IS NOT NULL
        DROP DATABASE VENTAS
GO
--Creando la base de datos VENTAS
CREATE DATABASE VENTAS
ON PRIMARY(
        NAME='VENTAS_PRI',
        FILENAME='C:\VENTAS\VENTAS.MDF',
        SIZE=50MB,
        MAXSIZE=150MB,
        FILEGROWTH=20%
),(
        NAME='VENTAS_SEC',
        FILENAME='C:\VENTAS\VENTAS.NDF',
        SIZE=10MB,
        MAXSIZE=50MB,
        FILEGROWTH=2MB
)
GO
```

Visualizar la base de datos mediante un listado:

```
SELECT * FROM SYS.sysdatabases S
        WHERE S.name='VENTAS'
GO
```

	name	dbid	sid	mode	status	status2	crdate
1	VENTAS	54	0x0105000000000005150000065A4682F59DF37CAB3D91B...	0	65536	1627389952	2017-03-09 18:19:55.897

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS_PRI	1	C:\VENTAS\VENTAS.MDF	PRIMARY	51200 KB	153600 KB	20%	data only
2	VENTAS_log	2	C:\VENTAS\VENTAS_log.ldf	NULL	15360 KB	2147483648 KB	10%	log only
3	VENTAS_SEC	3	C:\VENTAS\VENTAS.NDF	PRIMARY	10240 KB	51200 KB	2048 KB	data only

Caso desarrollado 4: Base de datos con especificación de archivos de transacciones

Cree la base de datos VENTAS en la carpeta C:\VENTAS. Tenga en cuenta lo siguiente:

- Archivo primario con un tamaño inicial de 50MB, máximo de 150MB y una tasa de crecimiento de 20%.
- Archivo de transacciones con un tamaño inicial de 5MB, máximo de 50MB y una tasa de crecimiento de 10%.
- Visualizar la base de datos mediante un listado.
- Mostrar los archivos que componen la base de datos.

Solución:

Recuerde que antes de ejecutar las sentencias, debe crear la carpeta VENTAS en la unidad C:\

```
--Activamos la base de datos maestra
USE MASTER
GO
--Validando la existencia de la base de datos
IF DB_ID('VENTAS')IS NOT NULL
        DROP DATABASE VENTAS
GO
--Creando la base de datos VENTAS
CREATE DATABASE VENTAS
ON PRIMARY(
        NAME='VENTAS_PRI',
        FILENAME='C:\VENTAS\VENTAS.MDF',
        SIZE=50MB,
        MAXSIZE=150MB,
        FILEGROWTH=20%
)
LOG ON (
        NAME='VENTAS_LOG',
        FILENAME='C:\VENTAS\VENTAS.LDF',
        SIZE=5MB,
        MAXSIZE=50MB,
        FILEGROWTH=10%
)
GO
```

Visualizar la base de datos mediante un listado:

```
SELECT * FROM SYS.sysdatabases S
       WHERE S.name='VENTAS'
GO
```

	name	dbid	sid	mode	status	status2	crdate
1	VENTAS	54	0x0105000000000005150000065A4682F59DF37CAB3D91B...	0	65536	1627389952	2017-03-09 18:19:55.897

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS_PRI	1	C:\VENTAS\VENTAS.MDF	PRIMARY	51200 KB	153600 KB	20%	data only
2	VENTAS_LOG	2	C:\VENTAS\VENTAS.LDF	NULL	5120 KB	51200 KB	10%	log only

Caso desarrollado 5: Añadir archivos a la base de datos

El administrador evalúa que su base de datos se llenará pronto, por lo que debe agregar tres archivos secundarios llamados VENTAS_SEC1, VENTAS_SEC2, VENTAS_SEC3, respectivamente; cuyo tamaño inicial es 10MB; su tamaño máximo, 500MB, y su crecimiento, 10%. Visualice los archivos de la base de datos.

Solución:

Recuerde que antes de ejecutar las sentencias, debe crear la carpeta VENTAS en la unidad C:\

```
ALTER DATABASE VENTAS
ADD FILE(
       NAME=VENTAS_SEC1,
       FILENAME='C:\VENTAS\SEC\VENTAS_SEC1.NDF',
       SIZE=10MB, MAXSIZE=500MB, FILEGROWTH=10%
)
GO
ALTER DATABASE VENTAS
ADD FILE(
       NAME=VENTAS_SEC2,
       FILENAME='C:\VENTAS\SEC\VENTAS_SEC2.NDF',
       SIZE=10MB, MAXSIZE=500MB, FILEGROWTH=10%
)
GO
ALTER DATABASE VENTAS
ADD FILE(
       NAME=VENTAS_SEC3,
       FILENAME='C:\VENTAS\SEC\VENTAS_SEC3.NDF',
       SIZE=10MB, MAXSIZE=500MB, FILEGROWTH=10%
)
GO
```

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS_PRI	1	C:\VENTAS\VENTAS.MDF	PRIMARY	51200 KB	153600 KB	20%	data only
2	VENTAS_LOG	2	C:\VENTAS\VENTAS.LDF	NULL	5120 KB	51200 KB	10%	log only
3	VENTAS_SEC1	3	C:\VENTAS\SEC\VENTAS_SEC1.ndf	PRIMARY	10240 KB	512000 KB	10%	data only
4	VENTAS_SEC2	4	C:\VENTAS\SEC\VENTAS_SEC2.ndf	PRIMARY	10240 KB	512000 KB	10%	data only
5	VENTAS_SEC3	5	C:\VENTAS\SEC\VENTAS_SEC3.ndf	PRIMARY	10240 KB	512000 KB	10%	data only

Caso desarrollado 6: Modificar propiedades de la base de datos

El administrador de la base de datos necesita modificar el tamaño inicial a 20MB y la tasa de crecimiento a 15% del archivo secundario VENTAS_SEC1. Visualice las propiedades del archivo secundario VENTAS_SEC1.

Solución:

```
ALTER DATABASE VENTAS
MODIFY FILE(
        NAME=VENTAS_SEC1,
        SIZE=20MB, FILEGROWTH=15%
)
GO
```

Mostrar los archivos que componen la base de datos:

```
USE VENTAS
GO
SP_HELPFILE VENTAS_SEC1
GO
```

	name	filename	filegroup	size	maxsize	growth	usage
1	VENTAS_SEC1	C:\VENTAS\SEC\VENTAS_SEC1.ndf	PRIMARY	20480 KB	512000 KB	15%	data only

Caso desarrollado 7: Eliminar archivos de la base de datos

El administrador de la base de datos necesita eliminar el archivo secundario VENTAS_SEC2. Visualice los archivos que componen la base de datos VENTAS.

Solución:

```
ALTER DATABASE VENTAS
REMOVE FILE VENTAS_SEC2
GO
```

Mostrar los archivos que componen la base de datos:

```
SP_HELPDB VENTAS
GO
```

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	VENTAS_PRI	1	C:\VENTAS\VENTAS.MDF	PRIMARY	51200 KB	153600 KB	20%	data only
2	VENTAS_LOG	2	C:\VENTAS\VENTAS.LDF	NULL	5120 KB	51200 KB	10%	log only
3	VENTAS_SEC1	3	C:\VENTAS\SEC\VENTAS_SEC1.ndf	PRIMARY	20480 KB	512000 KB	15%	data only
4	VENTAS_SEC3	5	C:\VENTAS\SEC\VENTAS_SEC3.ndf	PRIMARY	10240 KB	512000 KB	10%	data only

Caso desarrollado 8: Integración

Cree la base de datos BD_PROYECTOS en la carpeta C:\PROYECTOS\DATA\ con la siguiente configuración:

- Archivo de datos: Un tamaño inicial de 100MB, máximo de 250MB y un factor de crecimiento de 5MB.
- Archivo secundario: Un tamaño inicial de 20MB, máximo de 70MB y un factor de crecimiento de 10%.
- Archivo de transacciones: Un tamaño inicial de 10MB, máximo de 40MB y un factor de crecimiento de 2MB.

```
--Activamos la base de datos maestra
USE MASTER
GO
--Validando la existencia de la base de datos
IF DB_ID('BD_PROYECTOS')IS NOT NULL
        DROP DATABASE BD_PROYECTOS
GO
--Creando la base de datos BD_PROYECTOS
CREATE DATABASE BD_PROYECTOS
ON PRIMARY(
        NAME='BD_PROYECTOS_PRI',
        FILENAME='C:\PROYECTOS\PROYECTOS.MDF',
        SIZE=100,
        MAXSIZE=250,
        FILEGROWTH=5
),(
        NAME='BD_PROYECTOS_SEC',
        FILENAME='C:\PROYECTOS\PROYECTOS.NDF',
        SIZE=20,
        MAXSIZE=70,
        FILEGROWTH=10%
)
LOG ON (
        NAME='BD_PROYECTOS_LOG',
        FILENAME='C:\PROYECTOS\PROYECTOS.LDF',
        SIZE=10,
        MAXSIZE=40,
        FILEGROWTH=2
)
GO
```



CAPÍTULO 3

GESTIÓN DE TABLAS

Competencia tecnológica:

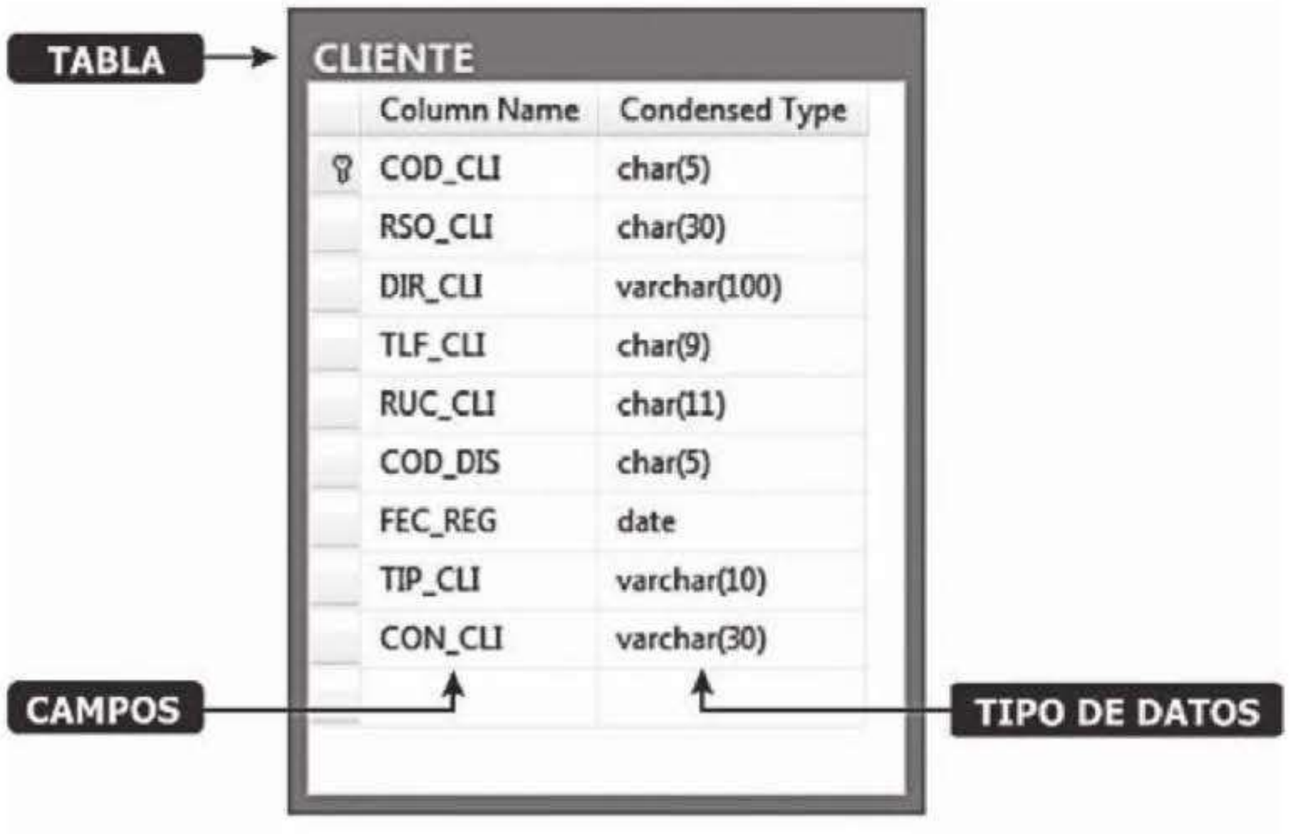
- Crear una base de datos en base a sentencias.
- Implementar sentencias DML sobre una base de datos.
- Implementar sentencias de manejo de información con Merge.

Contenido:

3.1 Sentencias DDL para la gestión de tablas
3.2 Restricciones de valores nulos y no nulos
3.3 Restricciones de integridad referencial
3.4 Manejo de esquemas
3.5 Restricciones para datos
3.6 Sentencias DML para datos
3.7 Manejo de datos masivos en SQL Server
3.8 Sentencia Merge
- Caso desarrollado 1
- Caso desarrollado 2

3.1 Sentencias DDL para la gestión de tablas

Las sentencias de definición de datos para tablas permitirán agregar, modificar o eliminar tablas dentro de una base de datos activa. Las tablas son consideradas como objetos de la base de datos y los principales componentes dentro del sistema de datos; asimismo, son las que guardan información que resulta importante en una base de datos.



La tabla presenta como estructura un nombre, el cual será definido con la sentencia DDL; la especificación de los campos se realizará dentro del bloque de construcción de la tabla y, por cada campo, se deberá especificar el tipo de datos.

3.1.1 Creación de una tabla

La tabla representa una entidad dentro de la base de datos; por lo tanto, se deberán crear las tablas necesarias para un sistema de base; asimismo, no podrá haber duplicidad de tablas y no existirán tablas que no se asocien a otras, es decir, todas las tablas deberán estar asociadas de alguna manera.

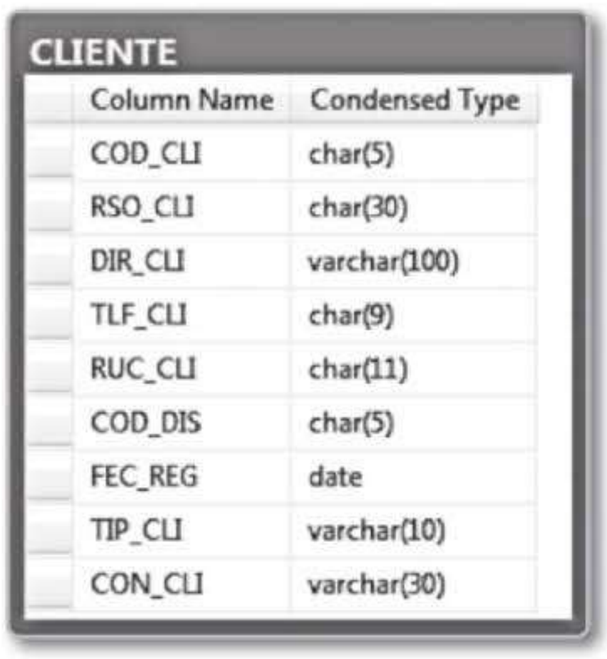
Formato general:

```
CREATE TABLE <NOMBRETABLA> (
 CAMPO1  TIPO,
 CAMPO2  TIPO,
 CAMPO3  TIPO
)
GO
```

Por ejemplo, si se crea la tabla CLIENTE, con los campos específicos, se podría tener el siguiente código:

```
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5),
        RSO_CLI CHAR(30),
        DIR_CLI VARCHAR(100),
        TLF_CLI CHAR(9),
        RUC_CLI CHAR(11),
        COD_DIS CHAR(5),
        FEC_REG DATE,
        TIP_CLI VARCHAR(10),
        CON_CLI VARCHAR(30)
)
GO
```

Al ejecutar la sentencia dentro de una base de datos, el resultado en forma de diagrama se mostraría de la siguiente manera:



3.1.2 Opciones de la tabla

Cuando se crean tablas dentro de una base de datos se presentan algunas opciones que se deben conocer para administrar de la mejor manera las tablas:

A. Validar la existencia de la tabla

Cuando la tabla es creada por primera vez dentro de la base de datos, no habrá problemas de validación, pero si se crea o modifica una tabla, será necesario controlar la existencia de la misma. Se puede usar el siguiente código de validación:

```
IF OBJECT_ID('CLIENTE')IS NOT NULL
        DROP TABLE CLIENTE
GO

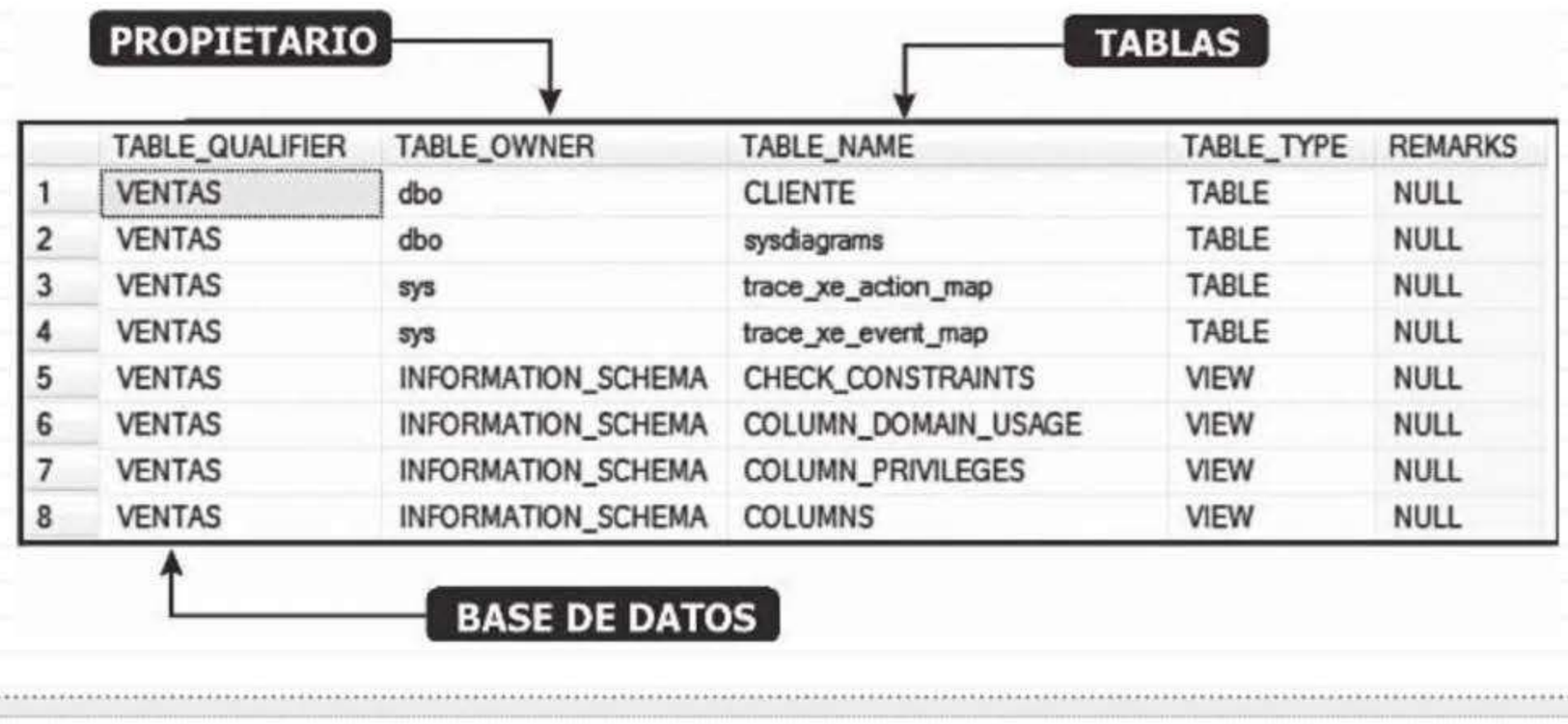
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5),
        RSO_CLI CHAR(30),
        DIR_CLI VARCHAR(100),
        TLF_CLI CHAR(9),
        RUC_CLI CHAR(11),
        COD_DIS CHAR(5),
        FEC_REG DATE,
        TIP_CLI VARCHAR(10),
        CON_CLI VARCHAR(30)
)
GO
```

La sentencia IF pregunta sobre la existencia del objeto Cliente; en caso exista, lo elimina usando la sentencia DROP TABLE; caso contrario, continúa con el código de creación de tabla.

B. Comprobar la existencia de la tabla

Cuando se ejecuta la sentencia CREATE TABLE de manera correcta, se envía como respuesta del servidor: "Comandos completados correctamente", pero no se podrá visualizar si la tabla realmente existe. Por tanto, se puede ejecutar la siguiente sentencia para listar todas las tablas de una base de datos activa:

```
SP_TABLES
GO
```

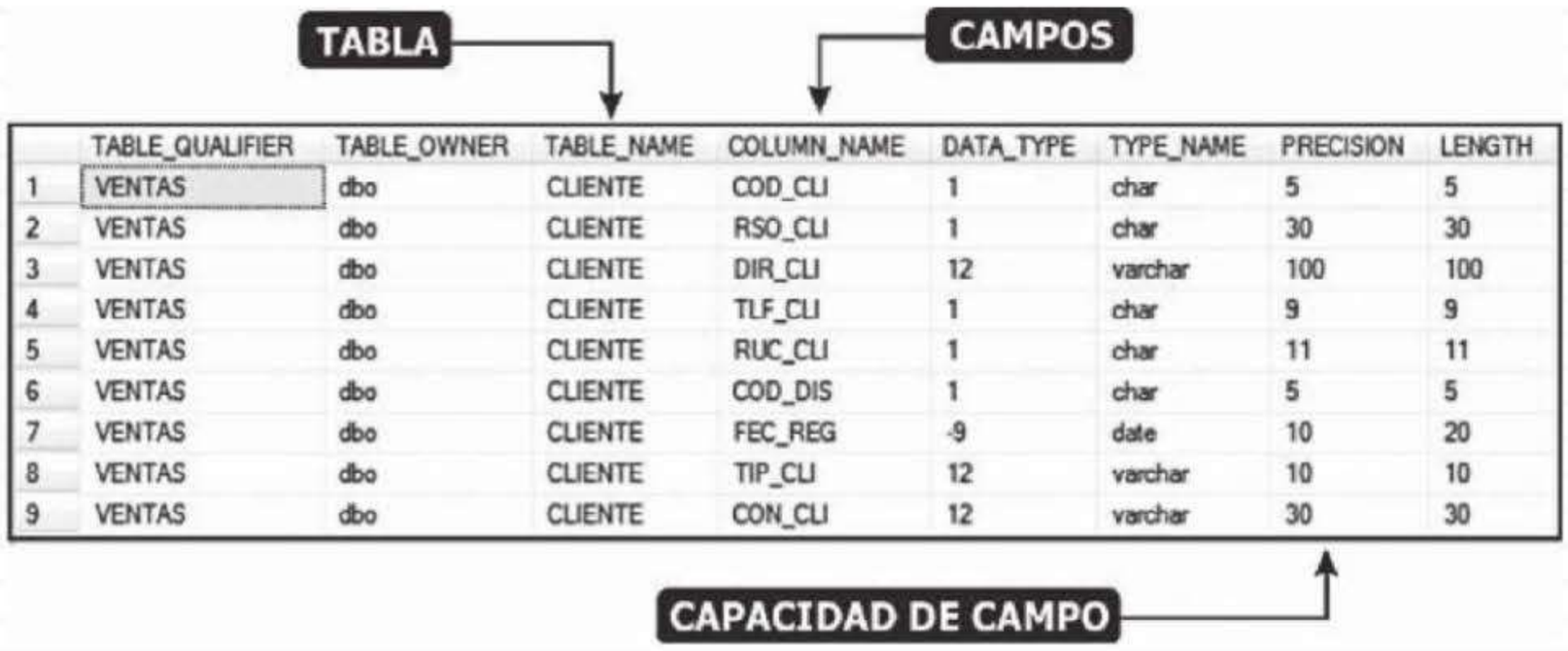


	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	REMARKS
1	VENTAS	dbo	CLIENTE	TABLE	NULL
2	VENTAS	dbo	sysdiagrams	TABLE	NULL
3	VENTAS	sys	trace_xe_action_map	TABLE	NULL
4	VENTAS	sys	trace_xe_event_map	TABLE	NULL
5	VENTAS	INFORMATION_SCHEMA	CHECK_CONSTRAINTS	VIEW	NULL
6	VENTAS	INFORMATION_SCHEMA	COLUMN_DOMAIN_USAGE	VIEW	NULL
7	VENTAS	INFORMATION_SCHEMA	COLUMN_PRIVILEGES	VIEW	NULL
8	VENTAS	INFORMATION_SCHEMA	COLUMNS	VIEW	NULL

C. Visualizar las columnas de la tabla

Al visualizar la estructura de una tabla, se podrá observar el nombre de la tabla, el nombre de las columnas y la capacidad de almacenamiento de cada campo. Para esto, se usa la siguiente sentencia:

```
SP_COLUMNS CLIENTE
GO
```



	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH
1	VENTAS	dbo	CLIENTE	COD_CLI	1	char	5	5
2	VENTAS	dbo	CLIENTE	RSO_CLI	1	char	30	30
3	VENTAS	dbo	CLIENTE	DIR_CLI	12	varchar	100	100
4	VENTAS	dbo	CLIENTE	TLF_CLI	1	char	9	9
5	VENTAS	dbo	CLIENTE	RUC_CLI	1	char	11	11
6	VENTAS	dbo	CLIENTE	COD_DIS	1	char	5	5
7	VENTAS	dbo	CLIENTE	FEC_REG	-9	date	10	20
8	VENTAS	dbo	CLIENTE	TIP_CLI	12	varchar	10	10
9	VENTAS	dbo	CLIENTE	CON_CLI	12	varchar	30	30

3.1.3 Modificación de una tabla

La modificación de la estructura de la tabla presenta varias opciones; como, por ejemplo, modificar, agregar o eliminar campos. Se presenta los siguientes formatos:

Modificación de la estructura:

```
ALTER TABLE <NOMBRETABLA>
 ALTER COLUMN <COLUMNA TIPO>
)
GO
```

Agregar un campo:

```
ALTER TABLE <NOMBRETABLA>
 ADD COLUMNANUEVA1 TIPO,
    COLUMNANUEVA2 TIPO
GO
```

Eliminar un campo:

```
ALTER TABLE <NOMBRE_TABLA>
 DROP COLUMN <COLUMNA>
)
GO
```

Algunos casos de modificación de la estructura de la tabla son:

```
Por ejemplo, a la tabla CLIENTE se le modifica el campo teléfono de cliente
"TLF_CLI" con tamaño 9 a tamaño 15.
ALTER TABLE CLIENTE
        ALTER COLUMN TLF_CLI CHAR(15)
GO

Si se agrega el campo FECHA DE NACIMIENTO, el código sería como sigue:
ALTER TABLE CLIENTE
 ADD FNA_CLI DATE
GO

Si se elimina el campo contacto del cliente "CON_CLI", el código sería
como sigue:
ALTER TABLE CLIENTE
 DROP COLUMN CON_CLI
GO

Para todos los casos, se debe comprobar las modificaciones mediante la
sentencia SP_COLUMNS CLIENTE.
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH
1	VENTAS	dbo	CLIENTE	COD_CLI	1	char	5	5
2	VENTAS	dbo	CLIENTE	RSO_CLI	1	char	30	30
3	VENTAS	dbo	CLIENTE	DIR_CLI	12	varchar	100	100
4	VENTAS	dbo	CLIENTE	TLF_CLI	1	char	15	15
5	VENTAS	dbo	CLIENTE	RUC_CLI	1	char	11	11
6	VENTAS	dbo	CLIENTE	COD_DIS	1	char	5	5
7	VENTAS	dbo	CLIENTE	FEC_REG	-9	date	10	20
8	VENTAS	dbo	CLIENTE	TIP_CLI	12	varchar	10	10
9	VENTAS	dbo	CLIENTE	FNA_CLI	-9	date	10	20

3.1.4 Eliminación de una tabla

Eliminar una tabla conlleva también a eliminar toda la información que contenga; por eso, se debe tener cuidado al elegir borrar una tabla.

Eliminación básica:

```
DROP TABLE <NOMBRETABLA>
GO
```

Eliminación por validación:

```
IF OBJECT_ID('<NOMBRETABLA>')IS NOT NULL
        DROP TABLE <NOMBRETABLA>
GO
```

Algunos casos de eliminación de tablas son:

```
Eliminar la tabla CLIENTE de forma básica:
DROP TABLE CLIENTE
GO

Eliminar la tabla CLIENTE validando su existencia:
IF OBJECT_ID('CLIENTE')IS NOT NULL
        DROP TABLE CLIENTE
GO

Para ambos casos, se debe comprobar si la tabla se eliminó correctamente
con la sentencia SP_TABLES.
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	REMARKS
1	VENTAS	dbo	sysdiagrams	TABLE	NULL
2	VENTAS	sys	trace_xe_action_map	TABLE	NULL
3	VENTAS	sys	trace_xe_event_map	TABLE	NULL

3.2 Restricciones de valores nulos y no nulos

Cuando se crea una tabla de base de datos, se debe especificar si el campo podrá recibir valores nulos o no. Por ejemplo, los campos claves o llaves siempre serán de restricción no nula, pues no existen llaves vacías; en otros campos, se debe analizar si será necesario dejarlo vacío o no, como en el caso de que algún cliente no cuente con correo electrónico.

Formato de restricción nulo o no nulo a partir de la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NULL | NOT NULL,
        CAMPO2 TIPO NULL | NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL
)
GO
```

Formato de restricción nulo o no nulo a partir de una modificación a la tabla:

```
ALTER TABLE <NOMBRETABLA>
 ALTER COLUMN TIPO NULL | NOT NULL
GO
```

Algunos casos de especificación de campo nulo o no nulo son:

```
Asignar valores nulos y no nulos en la creación de la tabla CLIENTE:
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5) NOT NULL,
        RSO_CLI CHAR(30) NOT NULL,
        DIR_CLI VARCHAR(100) NULL,
        TLF_CLI CHAR(9) NULL,
        RUC_CLI CHAR(11) NOT NULL,
        COD_DIS CHAR(5) NOT NULL,
        FEC_REG DATE NOT NULL,
        TIP_CLI VARCHAR(10) NOT NULL,
        CON_CLI VARCHAR(30) NULL
)

Añadir una columna ESTADO DEL CLIENTE (EST_CLI) con asignación de valor
nulo:
ALTER TABLE CLIENTE
ALTER COLUMN EST_CLI CHAR(1) NULL
```

3.3 Restricciones de integridad referencial

Las restricciones de integridad permiten asignar llaves primarias o foráneas a una tabla; es necesario dicha referencia para que las tablas se logren asociar a otras, generando la integridad referencial entre tablas. Se presentan dos restricciones para la integridad: la llave primaria y secundaria.

3.3.1 Llave primaria

También es llamada clave primaria, campo clave, llave principal o PRIMARY KEY, desde la cual se podrá asignar a una columna dicha restricción; esto restringe el ingreso de valor a datos únicos, el cual permite identificar un registro frente a los demás. Por buenas prácticas, el tipo de datos de un campo clave debe ser número entero (Int) o de tipo carácter simple (Char). Los formatos de asignación son:

Formato de asignación de campo clave a partir de la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPOCLAVE TIPO NOT NULL PRIMARY KEY,
        CAMPO2 TIPO NULL | NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL
)
GO
```

Formato de asignación de campo clave a partir de la modificación de la tabla:

```
ALTER TABLE <NOMBRETABLA>
 ADD PRIMARY KEY (CAMPOCLAVE)
GO
```

Formato de asignación de campo clave compuesto a partir de la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL,
        CAMPO2 TIPO NULL | NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL,
        PRIMARY KEY (<CAMPOCLAVE1>,<CAMPOCLAVE2>)
)
GO
```

Formato de asignación de campo clave compuesto a partir de la modificación de la tabla:

```
ALTER TABLE <NOMBRETABLA>
 ADD PRIMARY KEY (CAMPOCLAVE1, CAMPOCLAVE2)
GO
```

Algunos casos de especificación de campo clave simple y compuesto son:

Implementar la tabla DISTRITO asignando a COD_DIS como campo clave:

DISTRITO

	Column Name	Condensed Type	Nullable
🔑	COD_DIS	char(5)	No
	NOM_DIS	varchar(50)	No

```
Primera forma:
CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)        NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)    NOT NULL
)
GO

Segunda forma:
ALTER TABLE DISTRITO
 ADD PRIMARY KEY (COD_DIS)
GO
```

```
Tercera forma:
CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)       NOT NULL,
        NOM_DIS VARCHAR(50)   NOT NULL,
       PRIMARY KEY (COD_DIS)
)
GO

Implementar la tabla DETALLE_COMPRA asignando llaves compuestas a NUM_OCO
y COD_PRO como campos claves:
```

DETALLE_COMPRA

	Column Name	Condensed Type	Nullable
🔑	NUM_OCO	char(5)	No
🔑	COD_PRO	char(5)	No
	CAN_DET	int	No

```
Primera forma:
CREATE TABLE DETALLE_COMPRA(
        NUM_OCO CHAR(5)                 NOT NULL,
        COD_PRO CHAR(5)                 NOT NULL,
        CAN_DET INT                     NOT NULL
        PRIMARY KEY (NUM_OCO,COD_PRO)
)
GO

Segunda forma:
ALTER TABLE DETALLE_COMPRA
 ADD PRIMARY KEY (NUM_OCO,COD_PRO)
GO
```

3.3.2 Llave secundaria

También es llamada clave secundaria, campo foráneo, llave foránea, desde la cual se podrá asignar a una columna dicha restricción; esto permitirá asociar una tabla a otra. Los formatos de asignación son:

Formato de asignación de llave secundaria a partir de la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA1>(
        CAMPO1 TIPO NULL | NOT NULL,
        CAMPO2 TIPO NOT NULL REFERENCES <NOMBRETABLA2>,
        CAMPO3 TIPO NULL | NOT NULL
)
GO
```

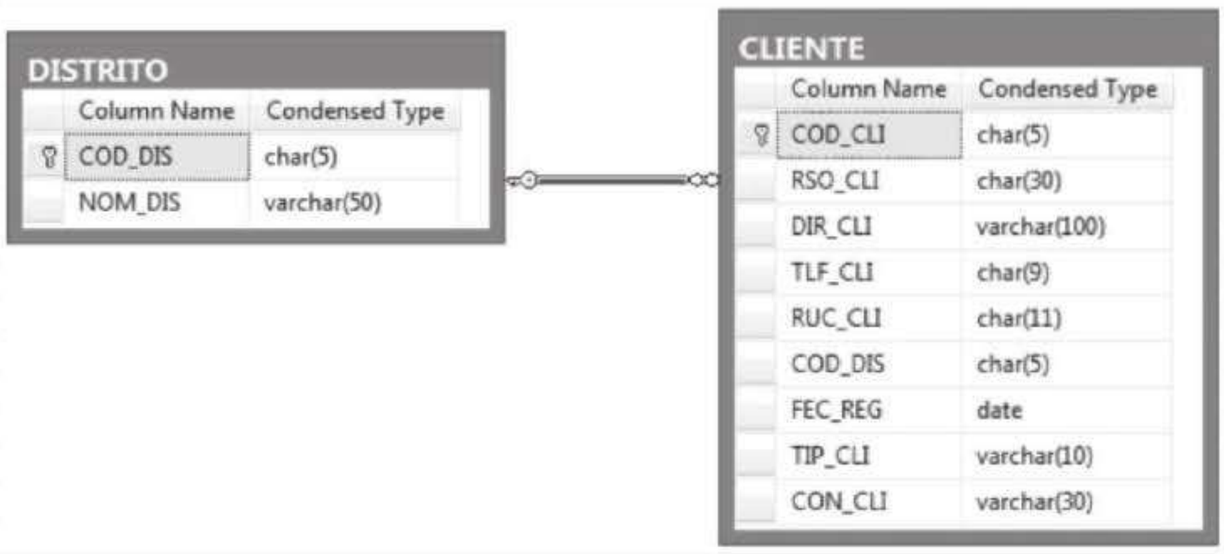
```
CREATE TABLE <NOMBRETABLA1>(
        CAMPO1 TIPO NULL | NOT NULL,
        CAMPO2 TIPO NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL,
       FOREIGN KEY (CAMPOSECUNDARIO) REFERENCES <NOMBRETABLA2>
)
GO
```

Formato de asignación de llave secundaria a partir de la modificación de la tabla:

```
ALTER TABLE <NOMBRETABLA1>
 ADD FOREIGN KEY (CAMPOSECUNDARIO) REFERENCES <NOMBRETABLA2>
GO
```

Algunos casos de especificación de campo clave simple y compuesto son:

```
Implementar la tabla DISTRITO asignando a COD_DIS como campo clave:
```




```
Primera forma:
CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)        NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)    NOT NULL
)
GO
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5)        NOT NULL PRIMARY KEY,
        RSO_CLI CHAR(30)       NOT NULL,
        DIR_CLI VARCHAR(100)NOT NULL,
        TLF_CLI CHAR(9)        NOT NULL,
        RUC_CLI CHAR(11)       NULL,
        COD_DIS CHAR(5)        NOT NULL REFERENCES DISTRITO,
        FEC_REG DATE           NOT NULL,
        TIP_CLI VARCHAR(10) NOT NULL,
        CON_CLI VARCHAR(30) NOT NULL
)
GO
```

```
Segunda forma:
CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)         NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)     NOT NULL
)
GO
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_CLI CHAR(30)        NOT NULL,
        DIR_CLI VARCHAR(100)    NOT NULL,
        TLF_CLI CHAR(9)         NOT NULL,
        RUC_CLI CHAR(11)        NULL,
        COD_DIS CHAR(5)         NOT NULL,
        FEC_REG DATE            NOT NULL,
        TIP_CLI VARCHAR(10)     NOT NULL,
        CON_CLI VARCHAR(30)     NOT NULL,
       FOREIGN KEY (COD_DIS)    REFERENCES DISTRITO
)
GO

Tercera forma:
CREATE TABLE DISTRITO(
        COD_DIS CHAR(5)         NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)     NOT NULL
)
GO
CREATE TABLE CLIENTE(
        COD_CLI CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_CLI CHAR(30)        NOT NULL,
        DIR_CLI VARCHAR(100)    NOT NULL,
        TLF_CLI CHAR(9)         NOT NULL,
        RUC_CLI CHAR(11)        NULL,
        COD_DIS CHAR(5)         NOT NULL,
        FEC_REG DATE            NOT NULL,
        TIP_CLI VARCHAR(10)     NOT NULL,
        CON_CLI VARCHAR(30)     NOT NULL
)
GO
ALTER TABLE CLIENTE
 ADD FOREIGN KEY (COD_DIS) REFERENCES DISTRITO
GO
```

3.4 Manejo de esquemas

Es un contenedor que puede contener tablas, vistas o procedimientos almacenados dentro de una misma base de datos. Es un trabajo de organización dentro de la base de datos; el esquema con más alto grado es la base de datos y los esquemas son secundarios; así es que, no es necesaria su implementación pero, por temas de organización, se pueden implementar.

Formato:

```
CREATE SCHEMA <NOMBRE_ESQUEMA> AUTHORIZATION DBO
GO
```

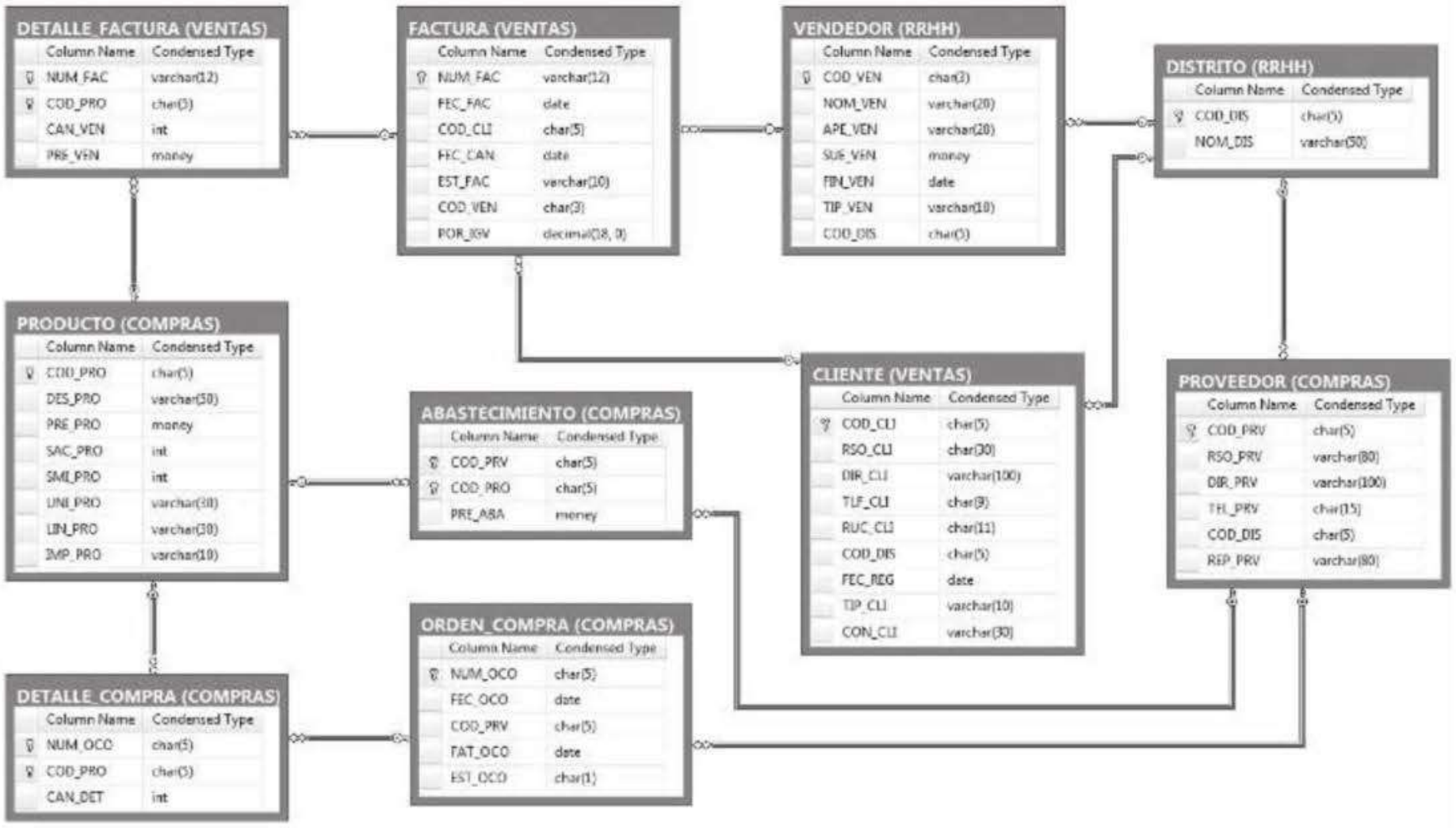
Formato para la creación de una tabla basada en esquemas:

```
CREATE TABLE <ESQUEMA>.<NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL,
        CAMPO2 TIPO NULL | NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL,
        PRIMARY KEY (<CAMPOCLAVE1>,<CAMPOCLAVE2>)
)
GO
```

A continuación, se muestra la implementación de la base de datos VENTAS con el uso de esquemas; inicialmente, se debe planear cómo será la distribución de las tablas por la cantidad de esquemas:

Esquema	Tablas
COMPRAS	Producto Proveedor Orden de compra Detalle de la orden de compra Abastecimiento
VENTAS	Cliente Factura Detalle de factura
RRHH	Vendedor Distrito

El diagrama de la base de datos se muestra de la siguiente manera:



El código SQL que permitirá crear los esquemas y las tablas se muestra a continuación:

```
USE MASTER
GO

SET DATEFORMAT YMD
GO

IF DB_ID('BDVENTAS') IS NOT NULL
        DROP DATABASE BDVENTAS
GO

CREATE DATABASE BDVENTAS
GO

USE BDVENTAS
GO

CREATE SCHEMA COMPRAS AUTHORIZATION DBO
GO
CREATE SCHEMA VENTAS AUTHORIZATION DBO
GO
CREATE SCHEMA RRHH AUTHORIZATION DBO
GO

CREATE TABLE RRHH.DISTRITO(
        COD_DIS CHAR(5)         NOT NULL PRIMARY KEY,
        NOM_DIS VARCHAR(50)     NOT NULL
)
GO

CREATE TABLE RRHH.VENDEDOR(
        COD_VEN CHAR(3)         NOT NULL PRIMARY KEY ,
        NOM_VEN VARCHAR(20)     NOT NULL,
        APE_VEN VARCHAR(20)     NOT NULL,
        SUE_VEN MONEY           NOT NULL,
        FIN_VEN DATE            NOT NULL,
        TIP_VEN VARCHAR(10)     NOT NULL,
        COD_DIS CHAR(5)         NOT NULL REFERENCES RRHH.DISTRITO
)
GO

CREATE TABLE VENTAS.CLIENTE(
        COD_CLI CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_CLI CHAR(30)        NOT NULL,
        DIR_CLI VARCHAR(100)    NOT NULL,
        TLF_CLI CHAR(9)         NOT NULL,
        RUC_CLI CHAR(11)        NULL,
        COD_DIS CHAR(5)         NOT NULL REFERENCES RRHH.DISTRITO,
        FEC_REG DATE            NOT NULL,
        TIP_CLI VARCHAR(10)     NOT NULL,
        CON_CLI VARCHAR(30)     NOT NULL
)
GO
```

```
CREATE TABLE COMPRAS.PROVEEDOR(
        COD_PRV CHAR(5)       NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)   NOT NULL,
        DIR_PRV VARCHAR(100)NOT NULL,
        TEL_PRV CHAR(15)      NULL,
        COD_DIS CHAR(5)       NOT NULL REFERENCES RRHH.DISTRITO,
        REP_PRV VARCHAR(80) NOT NULL
)
GO

CREATE TABLE VENTAS.FACTURA(
        NUM_FAC VARCHAR(12)   NOT NULL PRIMARY KEY,
        FEC_FAC DATE          NOT NULL,
        COD_CLI CHAR(5)       NOT NULL REFERENCES VENTAS.CLIENTE,
        FEC_CAN DATE          NOT NULL,
        EST_FAC VARCHAR(10)   NOT NULL,
        COD_VEN CHAR(3)       NOT NULL REFERENCES RRHH.VENDEDOR,
        POR_IGV DECIMAL       NOT NULL
)
GO

CREATE TABLE COMPRAS.ORDEN_COMPRA(
        NUM_OCO CHAR(5)       NOT NULL PRIMARY KEY,
        FEC_OCO DATE          NOT NULL,
        COD_PRV CHAR(5)       NOT NULL REFERENCES COMPRAS.PROVEEDOR,
        FAT_OCO DATE          NOT NULL,
        EST_OCO CHAR(1)       NOT NULL
)
GO

CREATE TABLE COMPRAS.PRODUCTO(
        COD_PRO CHAR(5)       NOT NULL PRIMARY KEY,
        DES_PRO VARCHAR(50)   NOT NULL,
        PRE_PRO MONEY         NOT NULL,
        SAC_PRO INT           NOT NULL,
        SMI_PRO INT           NOT NULL,
        UNI_PRO VARCHAR(30)   NOT NULL,
        LIN_PRO VARCHAR(30)   NOT NULL,
        IMP_PRO VARCHAR(10)   NOT NULL
)
GO

CREATE TABLE VENTAS.DETALLE_FACTURA(
        NUM_FAC VARCHAR(12)   NOT NULL  REFERENCES VENTAS.FACTURA,
        COD_PRO CHAR(5)       NOT NULL  REFERENCES COMPRAS.PRODUCTO,
        CAN_VEN INT           NOT NULL,
        PRE_VEN MONEY         NOT NULL
        PRIMARY KEY (NUM_FAC,COD_PRO)
)
GO

CREATE TABLE COMPRAS.DETALLE_COMPRA(
        NUM_OCO CHAR(5)       NOT NULL  REFERENCES COMPRAS.ORDEN_COMPRA,
        COD_PRO CHAR(5)       NOT NULL  REFERENCES COMPRAS.PRODUCTO,
        CAN_DET INT           NOT NULL
        PRIMARY KEY (NUM_OCO,COD_PRO)
)
GO
```

```
CREATE TABLE COMPRAS.ABASTECIMIENTO(
        COD_PRV CHAR(5)        NOT NULL  REFERENCES COMPRAS.PROVEEDOR,
        COD_PRO CHAR(5)        NOT NULL  REFERENCES COMPRAS.PRODUCTO,
        PRE_ABA MONEY          NOT NULL
        PRIMARY KEY (COD_PRV,COD_PRO)
)
GO
```

3.5 Restricciones para datos

3.5.1 Identity

La propiedad Identity solo es aplicable a columnas de tipo numérico, ya que define un autoincremento de valores que pueden representar una numeración de valores en forma automática por cada registro dentro de la tabla. Por ejemplo, podría tratarse de una tabla de facturas y su columna NUMFACTURA tenga asignada la propiedad Identity para identificar el número de factura registrada.

Formato para la asignación de Identity:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL PRIMARY KEY IDENTITY(VALOR1, VALOR2),
        CAMPO2 TIPO NULL | NOT NULL,
        CAMPO3 TIPO NULL | NOT NULL
)
GO
```

El primer valor de Identity determina el punto de inicio y el segundo valor indica la forma de incremento.

La especificación de la restricción Identity sobre la tabla FACTURA es la siguiente:

```
Implementar la tabla FACTURA asignando a NUM_FAC autoincrementable desde
el número 1000 con un aumento de uno:
```

FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	int
	FEC_FAC	date
	FEC_CAN	date
	EST_FAC	varchar(10)
	POR_IGV	decimal(18, 0)

```
CREATE TABLE FACTURA(
        NUM_FAC INT             NOT NULL PRIMARY KEY IDENTITY(1000,1),
        FEC_FAC DATE            NOT NULL,
        FEC_CAN DATE            NOT NULL,
        EST_FAC VARCHAR(10)     NOT NULL,
        POR_IGV DECIMAL         NOT NULL
)
GO

Se registran cinco facturas para comprobar las numeraciones automáticas:

INSERT INTO FACTURA VALUES('06-07-2013','05-08-2013','2','0.19')
INSERT INTO FACTURA VALUES('06-07-2013','05-08-2013','3','0.19')
INSERT INTO FACTURA VALUES('01-09-2013','03-10-2013','2','0.19')
INSERT INTO FACTURA VALUES('06-09-2013','05-10-2013','2','0.19')
INSERT INTO FACTURA VALUES('01-10-2013','12-10-2013','2','0.19')

Se visualiza sus registros usando la sentencia SELECT * FROM FACTURA:
```

	NUM_FAC	FEC_FAC	FEC_CAN	EST_FAC	POR_IGV
1	1000	2013-06-07	2013-05-08	2	0
2	1001	2013-06-07	2013-05-08	3	0
3	1002	2013-01-09	2013-03-10	2	0
4	1003	2013-06-09	2013-05-10	2	0
5	1004	2013-01-10	2013-12-10	2	0

3.5.2 Default

Permite asignar un determinado valor por defecto según el tipo de datos; dicho valor debe ser especificado antes de registrar valores a la tabla.

Formato de asignación de Default en la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL PRIMARY KEY IDENTITY(VALOR1, VALOR2),
        CAMPO2 TIPO NULL | NOT NULL DEFAULT <VALOR>,
        CAMPO3 TIPO NULL | NOT NULL DEFAULT <VALOR>
)
GO
```

Formato de asignación de Default en la modificación de la tabla:

```
ALTER TABLE <NOMBRE_TABLA>
 ADD DEFAULT 'VALOR'
 FOR <CAMPO>
```

La especificación de la restricción Default sobre la tabla PROVEEDOR es la siguiente:

```
Implementar la tabla PROVEEDOR asignando como valor por defecto a DIR_PRV
el término "NO REGISTRA" y en el campo TEL_PRV asignar 000-0000000:
```

PROVEEDOR

	Column Name	Condensed Type
🔑	COD_PRV	char(5)
	RSO_PRV	varchar(80)
	DIR_PRV	varchar(100)
	TEL_PRV	char(15)
	REP_PRV	varchar(80)

```
Primera forma:
CREATE TABLE PROVEEDOR(
        COD_PRV CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)     NOT NULL,
        DIR_PRV VARCHAR(100)    NOT NULL DEFAULT 'NO REGISTRA',
        TEL_PRV CHAR(15)        NULL     DEFAULT '000-0000000',
        REP_PRV VARCHAR(80)     NOT NULL
)
GO

Segunda forma:
CREATE TABLE PROVEEDOR(
        COD_PRV CHAR(5)         NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)     NOT NULL,
        DIR_PRV VARCHAR(100)    NOT NULL,
        TEL_PRV CHAR(15)        NULL,
        REP_PRV VARCHAR(80)     NOT NULL
)
GO

ALTER TABLE PROVEEDOR
 ADD DEFAULT 'NO REGISTRA'
 FOR DIR_PRV
GO

ALTER TABLE PROVEEDOR
 ADD DEFAULT '000-000000'
 FOR TEL_PRV
GO

Se registran cinco datos de proveedores para comprobar los datos por
defecto.

INSERT INTO PROVEEDOR VALUES('PR01','FABER CASTELL',DEFAULT,'4330895','CARLOS
AGUIRRE')
INSERT INTO PROVEEDOR VALUES('PR02','ATLAS','AV. LIMA 471',DEFAULT,'CESAR
TORRES')
INSERT INTO PROVEEDOR VALUES('PR03','3M',DEFAULT,DEFAULT,'OMAR INJOQUE')
INSERT INTO PROVEEDOR VALUES('PR04','DITO','AV. METROPOLITANA 376',NULL,'RAMON
FLORES')
INSERT INTO PROVEEDOR VALUES('PR05','ACKER','CALLE LAS DUNAS 245','4780143',
'JULIO ACUÑA')
```

Se visualiza sus registros usando la sentencia SELECT * FROM PROVEEDOR:

	COD_PRV	RSO_PRV	DIR_PRV	TEL_PRV	REP_PRV
1	PR01	FABER CASTELL	NO REGISTRA	4330895	CARLOS AGUIRRE
2	PR02	ATLAS	AV. LIMA 471	000-000000	CESAR TORRES
3	PR03	3M	NO REGISTRA	000-000000	OMAR INJOQUE
4	PR04	DITO	AV. METROPOLITANA 376	NULL	RAMON FLORES
5	PR05	ACKER	CALLE LAS DUNAS 245	4780143	JULIO ACUÑA

Para visualizar las restricciones aplicadas a la tabla PROVEEDOR, se puede ejecutar el siguiente código:

```
SP_HELPCONSTRAINT PROVEEDOR
GO
```

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	DEFAULT on column DIR_PRV	DF__PROVEEDOR__DIR_P__182C9B23	(n/a)	(n/a)	(n/a)	(n/a)	('NO REGISTRA')
2	DEFAULT on column TEL_PRV	DF__PROVEEDOR__TEL_P__1920BF5C	(n/a)	(n/a)	(n/a)	(n/a)	('000-000000')
3	PRIMARY KEY (clustered)	PK__PROVEEDO__28BE23B29ED24C1A	(n/a)	(n/a)	(n/a)	(n/a)	COD_PRV

3.5.3 Check

Permite restringir el rango de valores que pueden estar permitidos ingresar en una o más columnas de una tabla. Se debe tener en cuenta que Check evalúa el valor a registrar en la tabla; por lo tanto, se debe implementar una condición sobre cada campo evaluado.

Formato de asignación de Check en la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL PRIMARY KEY IDENTITY(VALOR1, VALOR2),
        CAMPO2 TIPO NULL | NOT NULL CHECK <CONDICION>,
        CAMPO3 TIPO NULL | NOT NULL CHECK <CONDICION>
)
GO
```

Formato de asignación de Check en la modificación de la tabla:

```
ALTER TABLE <NOMBRETABLA>
ADD CONSTRAINT <NOMBRE>
CHECK <CONDICION>
```

La especificación de la restricción Check sobre la tabla PRODUCTO es la siguiente:

Implementar la tabla PRODUCTO haciendo que su precio solo registre valores entre 1 y 100, el *stock* actual sea mayor a cero y que solo se permita registrar los valores bolsa, caja o paquete en la unidad de producto:




```
Primera forma:
CREATE TABLE PRODUCTO(
 COD_PRO CHAR(5)       NOT NULL PRIMARY KEY,
 DES_PRO VARCHAR(50)   NOT NULL,
 PRE_PRO MONEY         NOT NULL CHECK (PRE_PRO BETWEEN 1 AND 100),
 SAC_PRO INT           NOT NULL CHECK (SAC_PRO>0),
 SMI_PRO INT           NOT NULL,
 UNI_PRO VARCHAR(30)   NOT NULL CHECK (UNI_PRO IN('BOLSA','CAJA','PAQUETE')),
 LIN_PRO VARCHAR(30)   NOT NULL,
 IMP_PRO VARCHAR(10)   NOT NULL
)
GO

Segunda forma:
CREATE TABLE PRODUCTO(
        COD_PRO CHAR(5)       NOT NULL PRIMARY KEY,
        DES_PRO VARCHAR(50)   NOT NULL,
        PRE_PRO MONEY         NOT NULL,
        SAC_PRO INT           NOT NULL,
        SMI_PRO INT           NOT NULL,
        UNI_PRO VARCHAR(30)   NOT NULL,
        LIN_PRO VARCHAR(30)   NOT NULL,
        IMP_PRO VARCHAR(10)   NOT NULL
)
GO

ALTER TABLE PRODUCTO
 ADD CONSTRAINT CHK_PRECIO
 CHECK (PRE_PRO BETWEEN 1 AND 100)
GO
```

```
ALTER TABLE PRODUCTO
 ADD CONSTRAINT CHK_STOCKACTUAL
 CHECK (SAC_PRO>0)
GO

ALTER TABLE PRODUCTO
 ADD CONSTRAINT CHK_UNIDAD
 CHECK (UNI_PRO IN ('BOLSA','CAJA','PAQUETE'))
GO
```

Se comprueba la restricción en la unidad del producto:

```
INSERT INTO  PRODUCTO
 VALUES('P001','PAPEL BOND A-4',35.0, 200,1500,'OTRO','2','V')
GO
```

El mensaje mostrado desde el servidor:

```
Messages
Msg 547, Level 16, State 0, Line 1
The INSERT statement conflicted with the CHECK constraint "CHK_UNIDAD".
The conflict occurred in database "BDVENTAS3", table "dbo.PRODUCTO", column 'UNI_PRO'.
The statement has been terminated.
```

Para visualizar las restricciones aplicadas a la tabla PRODUCTO, se puede ejecutar el siguiente código:

```
SP_HELPCONSTRAINT PRODUCTO
GO
```

3.5.4 Unique

Permite determinar que los valores registrados en una misma columna no sean idénticos, es decir, se mantengan únicos; como se vio, anteriormente, el campo clave también tiene la misma restricción, pero existe una diferencia entre ellos: el campo clave solo se puede definir en campos específicos, mientras que con Unique se puede restringir a todas las columnas restantes. Se debe tener en cuenta que asignar Unique a un campo no clave no significa que se le esté asignando como llave. Una columna definida como Unique acepta a lo más un valor NULL, mientras que una columna restringida con PRIMARY KEY no permite dicha acción.

Formato de asignación de Unique en la creación de la tabla:

```
CREATE TABLE <NOMBRETABLA>(
        CAMPO1 TIPO NOT NULL PRIMARY KEY,
        CAMPO2 TIPO NULL | NOT NULL UNIQUE,
        CAMPO3 TIPO NULL | NOT NULL UNIQUE
)
GO
```

Formato de asignación de Unique en la modificación de la tabla:

```
ALTER TABLE <NOMBRE_TABLA>
 ADD CONSTRAINT 'NOMBRE_RESTRINCCION'
 UNIQUE <CAMPO>
```

La especificación de la restricción Unique sobre la tabla PROVEEDOR es la siguiente:

Implementar la tabla PROVEEDOR asignando como valor único a la columna teléfono:

PROVEEDOR

	Column Name	Condensed Type
🔑	COD_PRV	char(5)
	RSO_PRV	varchar(80)
	DIR_PRV	varchar(100)
	TEL_PRV	char(15)
	REP_PRV	varchar(80)

Primera forma:

```
CREATE TABLE PROVEEDOR(
        COD_PRV CHAR(5)        NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)    NOT NULL,
        DIR_PRV VARCHAR(100)   NOT NULL,
        TEL_PRV CHAR(15)       NULL UNIQUE,
        REP_PRV VARCHAR(80)    NOT NULL
)
GO
```

Segunda forma:

```
CREATE TABLE PROVEEDOR(
        COD_PRV CHAR(5)        NOT NULL PRIMARY KEY,
        RSO_PRV VARCHAR(80)    NOT NULL,
        DIR_PRV VARCHAR(100)   NOT NULL,
        TEL_PRV CHAR(15)       NULL,
        REP_PRV VARCHAR(80)    NOT NULL
)
GO

ALTER TABLE PROVEEDOR
 ADD CONSTRAINT UNI_TELEFONO
 UNIQUE (TEL_PRV)
GO
```

Para visualizar las restricciones aplicadas a la tabla PROVEEDOR, se puede ejecutar el siguiente código:

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'PROVEEDOR'
```

3.6 Sentencias DML para datos

3.6.1 Insert

La sentencia INSERT permite agregar una nueva fila a una tabla o vista de una determinada base de datos.

Consideraciones generales al insertar una fila nueva:

- Si se desea reemplazar la fila registrada, primero se la debe eliminar con la sentencia DELETE o TRUNCATE TABLE, ya que ocasionaríamos un error de duplicidad de registro debido al código del mismo.
- Si se desea actualizar la fila registrada, se debe usar la sentencia UPDATE, condicionando el campo clave para una actualización personalizada.
- Para insertar un conjunto de registros a una nueva tabla, se debe combinar las sentencias SELECT e INTO.
- Cuando la sentencia INSERT es ejecutada y detecta un error aritmético, el Motor de base de datos de SQL muestra un mensaje de error desde la activación Set Arithabort On y detiene el resto del código ejecutado.
- No es posible registrar subconsultas dentro de una misma implementación de registros de una tabla.

Formato básico:

La especificación de valores debe ser exactamente igual a la cantidad de columnas que tiene la tabla.

```
INSERT INTO <NOMBRE_TABLA>
VALUES (VALOR1, VALOR2, VALOR3, VALORN)
```

Formato de especificación de campos:

La especificación de las columnas determina el orden de registro de valores; solo hay que considerar las columnas que tienen valor de tipo Identity, ya que estos, así se especifiquen o no, siempre registrarán un valor.

```
INSERT INTO <NOMBRE_TABLA> (CAMPO1, CAMPO2, CAMPO3)
VALUES (VALOR1, VALOR2, VALOR3)
```

Formato de inserción múltiple:

Se puede utilizar el formato básico o de especificación de campos para una inserción de múltiples registros.

```
INSERT INTO <NOMBRE_TABLA> VALUES
(VALOR1, VALOR2, VALOR3),
(VALOR1, VALOR2, VALOR3),
(VALOR1, VALOR2, VALOR3),
(VALOR1, VALOR2, VALOR3)
GO

INSERT INTO <NOMBRE_TABLA> (CAMPO1, CAMPO2, CAMPO3) VALUES
 (VALOR1, VALOR2, VALOR3),
 (VALOR1, VALOR2, VALOR3),
 (VALOR1, VALOR2, VALOR3)
GO
```

Formato de llenado por una tabla externa:

Hay que tener en cuenta que la tabla especificada en el INSERT es la tabla DESTINO y se llenará de registros provenientes de otra tabla a la cual se accede por medio de una consulta.

```
INSERT INTO <NOMBRE_TABLA>
 SELECT * FROM <TABLA>
```

Formato de llenado con variables:

Este tipo de llenado usa variables locales; para el registro de valores, se usa la sentencia DECLARE para declarar variables locales, y la sentencia SET para asignar un valor. Se debe tener en cuenta que en SQL las variables siempre empiezan con arroba.

```
-- Declarando las variables locales
DECLARE @VAR1 <TIPO>, @VAR2 <TIPO>, @VAR3 <TIPO>

-- Asignando los valores a las variables
SELECT @VAR1=VALOR , @VAR2 =VALOR, @VAR3=VALOR

-- Enviando los valores a la tabla desde las variables
INSERT INTO <NOMBRE_TABLA>
VALUES (@VAR1, @VAR2, @VAR3)
```

Formato de llenado en la creación de la tabla:

Hay que tener en cuenta que la tabla de donde provienen los datos se especifica en la cláusula FROM y que ella también determina la estructura que tendrá la tabla DESTINO. Esto se puede controlar por medio del símbolo asterisco (*) o especificando los campos.

```
SELECT * INTO <TABLA_DESTINO>
 FROM <TABLA_ORIGEN>
```

Donde:

- **INTO:** Es una palabra clave opcional; normalmente, se usa entre el INSERT y la especificación de la tabla destino.
- **TABLA DESTINO O VISTA:** Es el nombre de la tabla o vista destino, es decir, la que recibirá los datos. También es posible especificar una variable de tipo TABLE como origen de tabla.
- **ASTERISCO (*):** Representa a todas las columnas que compondrán la estructura de la nueva tabla. Aquí también se pueden especificar las columnas necesarias.
- **TABLA ORIGEN:** Desde aquí se obtendrán todos los datos necesarios para la nueva tabla, tanto en información como en estructura.

La especificación de la sentencia INSERT sobre la tabla PRODUCTO es la siguiente:

```
Usando la tabla PRODUCTO, se realizarán los siguientes registros de
valores con los distintos formatos de inserción:
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
Realizando la inserción de forma básica
INSERT INTO  PRODUCTO VALUES('P001','PAPEL BOND A-4',35.0,
200,1500,'MLL','2','V')
INSERT INTO  PRODUCTO VALUES('P002','PAPEL BOND OFICIO ',35,
50,1500,'MLL','2','F')

Realizando la inserción con especificación de campos
INSERT  INTO   PRODUCTO  (COD_PRO,DES_PRO,PRE_PRO,SAC_PRO,SMI_PRO,UNI_PRO,
LIN_PRO,IMP_PRO)
       VALUES('P003','PAPEL BULKY ',10 ,498,1000,'MLL','2','V')
INSERT INTO  PRODUCTO (COD_PRO,DES_PRO,PRE_PRO,SAC_PRO,SMI_PRO,UNI_PRO,
LIN_PRO,IMP_PRO)
       VALUES('P004','PAPEL PERIODICO',9,4285,1000,'MLL','2','F')

Realizando la inserción múltiple
INSERT INTO  PRODUCTO VALUES
 ('P005','CARTUCHO TINTA NEGRA', 40 ,50,30,'UNI','1','F'),
 ('P006','CARTUCHO TINTA COLOR ',45  ,58,35, 'UNI','1','F')
```

```
INSERT INTO  PRODUCTO (COD_PRO,DES_PRO,PRE_PRO,SAC_PRO,SMI_PRO,UNI_PRO,
LIN_PRO,IMP_PRO) VALUES
 ('P007','PORTA DISKETTES',3.5,300,100,'UNI','1','V'),
 ('P008','CAJA DE DISKETTES * 10 ',30,125,180,'UNI','1','F')

Realizando la inserción desde una tabla externa
CREATE TABLE BK_PRODUCTO(
        COD_PRO CHAR(5)        NOT NULL PRIMARY KEY,
        DES_PRO VARCHAR(50)    NOT NULL,
        PRE_PRO MONEY          NOT NULL,
        SAC_PRO INT            NOT NULL,
        SMI_PRO INT            NOT NULL,
        UNI_PRO VARCHAR(30)    NOT NULL,
        LIN_PRO VARCHAR(30)    NOT NULL,
        IMP_PRO VARCHAR(10)    NOT NULL
)
GO
--Realizar la inserción
INSERT INTO  BK_PRODUCTO
        SELECT * FROM PRODUCTO
GO

--Comprobar
SELECT * FROM BK_PRODUCTO
GO

Realizando la inserción con uso de variables
DECLARE @COD CHAR(4),@DES VARCHAR(40),@PRE MONEY,@SAC INT,@SMI INT,@UNI
VARCHAR(30),@LIN INT,@IMP CHAR(1)

SET @COD='P010'
SET @DES='BORRADOR BLANCO'
SET @PRE=8
SET @SAC=2000
SET @SMI=400
SET @UNI='DOC'
SET @LIN='3'
SET @IMP='F'
INSERT INTO  PRODUCTO VALUES(@COD,@DES,@PRE,@SAC,@SMI,@UNI,@LIN,@IMP)
GO

Para visualizar los registros de la tabla PRODUCTO, se usa la sentencia
SELECT * FROM PRODUCTO.
```

3.6.2 Update

La sentencia UPDATE permite modificar o actualizar un conjunto de registros de una tabla o vista, dependiendo de una condición.

Formato:

```
UPDATE <TABLA>
 SET<CAMPO = VALOR>,<CAMPO = VALOR>
 WHERE <CONDICION>
```

Donde:

- TABLA: Aquí se especifica el nombre de la tabla o vista que necesite ser actualizada.
- SET: Aquí se especifica la lista de nombres de campos que se actualizarán en una determinada tabla.
- <CAMPO=VALOR>: Aquí se especifica la columna que desea actualizar; solo hay que tener en cuenta que no se pueden actualizar las columnas con restricciones de identidad (Identity) o los campos claves. Con respecto al valor asignado también podrá ser una expresión.
- WHERE: Especifica las condiciones que limitarán los valores que se actualizarán. Todo dependerá de la condición planteada; los operadores dentro de la instrucción Where dependerán del tipo de datos de la columna.

Consideraciones generales al actualizar registros:

- Si la sentencia de actualización infringe una restricción, una regla, o si el nuevo valor es de un tipo de datos incompatible al declarado en la tabla; se cancela la instrucción dando como resultado un mensaje de error y no actualiza ningún registro de la tabla.
- La sentencia UPDATE es susceptible a errores aritméticos que podría ser un error de desbordamiento o división por cero durante la evaluación de la expresión; en este caso, la actualización no se lleva a cabo, mostrando un mensaje de error y cortando la actualización desde el punto de error. A diferencia del primer punto, este sí actualiza los registros hasta donde se ocasionó el error.
- Se puede implementar la sentencia UPDATE dentro de funciones definidas por el usuario, solo hay que tener en cuenta que la tabla que se va a modificar sea una variable de tipo TABLE.
- Si en la actualización participan columnas con definición de tipo Char y Nchar, se rellenan con espacios vacíos a la derecha hasta la longitud definida en la tabla.

La especificación de la sentencia UPDATE sobre la tabla PRODUCTO es la siguiente:

```
Usando la tabla PRODUCTO, se realizarán las siguientes actualizaciones:
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

Actualizando todos los registros de la tabla PRODUCTO

Inicialmente los registros se muestran de la siguiente manera:

	COD_PRO	DES_PRO	PRE_PRO
1	P001	PAPEL BOND A-4	35.00
2	P002	PAPEL BOND OFICIO	35.00
3	P003	PAPEL BULKY	10.00
4	P004	PAPEL PERIODICO	9.00
5	P005	CARTUCHO TINTA NEGRA	40.00
6	P006	CARTUCHO TINTA COLOR	45.00
7	P007	PORTA DISKETTES	3.50
8	P008	CAJA DE DISKETTES * 10	30.00
9	P009	BORRADOR DE TINTA	10.00
10	P010	BORRADOR BLANCO	8.00

Se actualizarán los precios aumentando $2.0 a todos los productos:

```
UPDATE PRODUCTO
       SET PRE_PRO=PRE_PRO+2
GO
```

También se podría usar el siguiente código:

```
UPDATE PRODUCTO
       SET PRE_PRO+=2
GO
```

Se verifica que la actualización sea correcta:

	COD_PRO	DES_PRO	PRE_PRO
1	P001	PAPEL BOND A-4	37.00
2	P002	PAPEL BOND OFICIO	37.00
3	P003	PAPEL BULKY	12.00
4	P004	PAPEL PERIODICO	11.00
5	P005	CARTUCHO TINTA NEGRA	42.00
6	P006	CARTUCHO TINTA COLOR	47.00
7	P007	PORTA DISKETTES	5.50
8	P008	CAJA DE DISKETTES * 10	32.00
9	P009	BORRADOR DE TINTA	12.00
10	P010	BORRADOR BLANCO	10.00

Actualizando registros condicionados de la tabla PRODUCTO

Actualizar el *stock* actual de los productos al doble solo si dicho *stock* es menor o igual a 50.

```
UPDATE PRODUCTO
       SET SAC_PRO*=2
       WHERE SAC_PRO<=50
GO
```

Actualizar el precio de los productos descontando un 50% del precio actual solo para los productos con código P001, P005 y P010.

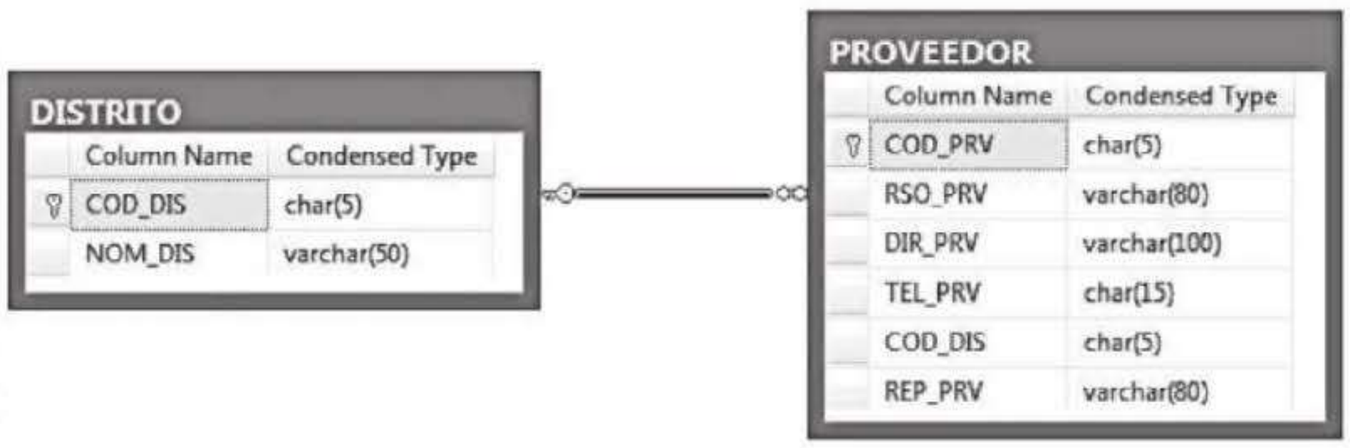
```
UPDATE PRODUCTO
       SET PRE_PRO=PRE_PRO*0.5
       WHERE COD_PRO IN ('P001','P005','P010')
```

```
GO

Actualizar la descripción del producto con código P004 por "PAPEL PERIODICO
75 GRAMOS".
UPDATE PRODUCTO
        SET DES_PRO='PAPEL PERIODICO 75 GRAMOS'
        WHERE COD_PRO='P004'
GO
```

Actualizando registros con unión de tablas

Actualizar el teléfono por 0000000 a todos los proveedores cuyo distrito se encuentre registrado con RIMAC.



Inicialmente se tiene los siguientes registros en la tabla PROVEEDOR:

	COD_PRV	RSO_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_PRV
1	PR01	FABER CASTELL	AV. ISABEL LA CATOLICA 1875	4330895	D13	CARLOS AGUIRRE
2	PR02	ATLAS	AV. LIMA 471	5380926	D13	CESAR TORRES
3	PR12	SUMIGRAF	AV. MANCO CAPAC 754	3320343	D13	KARINNA PAREDES

El código de distrito D13 pertenece al RIMAC como se muestra a continuación:

	COD_DIS	NOM_DIS
1	D13	RIMAC

```
Por tanto, la siguiente sentencia permite actualizar con 0000000 a todos
los proveedores del distrito del RIMAC.
UPDATE PROVEEDOR
        SET TEL_PRV='0000000'
        FROM PROVEEDOR AS P
        JOIN DISTRITO AS D ON P.COD_DIS=D.COD_DIS
        AND P.COD_DIS= (SELECT COD_DIS
                        FROM DISTRITO
                        WHERE NOM_DIS='RIMAC')
GO
```

La cláusula JOIN permite unir dos o más tablas; se recomienda usar alias para cada tabla relacionada, porque en la cláusula ON se deberá especificar la forma en que se unen ambas tablas, y siempre se agregará una condición de búsqueda tipo subconsulta. En el código anterior, la sentencia:

```
SELECT COD_DIS
 FROM DISTRITO
 WHERE NOM_DIS='RIMAC'

Emite como resultado el código D13 que permitirá actualizar a todos los
proveedores que cumplan con dicha condición. Si no usamos la unión de
tablas, el código de actualización sería de la siguiente manera:

UPDATE PROVEEDOR
        SET TEL_PRV='0000000'
        WHERE COD_DIS='D13'
GO

O también se podría optar por la siguiente sentencia:

UPDATE PROVEEDOR
SET TEL_PRV='0000000'
        WHERE COD_DIS = (SELECT COD_DIS
                         FROM DISTRITO
                         WHERE NOM_DIS='RIMAC')
GO


Actualizando registros usando variables
Actualizar el valor importado del producto por F solo para aquellos
productos cuyo tipo de unidad sea DOC (docena).

DECLARE @IMPORTADO CHAR(1)='F'
UPDATE PRODUCTO
        SET IMP_PRO=@IMPORTADO
        WHERE UNI_PRO='DOC'
GO
```

3.6.3 Delete

La sentencia DELETE permite eliminar todos los registros especificados en una determinada tabla. Hay que tener en cuenta que, para eliminar registros, estos no deben encontrarse asociados a otra tabla, ya que rompería la regla de integridad.

Formato:

```
DELETE <FROM><TABLA>
 WHERE <CONDICION>
```

Donde:

- **TABLA:** Aquí se especifica el nombre de la tabla o vista que necesite eliminar sus registros.
- **WHERE:** Condiciona la eliminación de los registros dependiendo de la condición especificada.

Consideraciones generales al eliminar una fila:

- La sentencia DELETE se puede bloquear cuando se ocasione un error de restricción FOREIGN KEY, es decir, cuando se intente eliminar registros que se encuentran asociados a otras tablas. De encontrarse este caso, el Motor de base de datos envía un error al usuario y no elimina ningún registro del script.
- Cuando la sentencia DELETE determina un error aritmético en su proceso, el Motor de base de datos cancela todo el proceso y envía un mensaje de error al usuario, ya que activa la librería SET ARITHABORT ON.
- También se puede optar por la sentencia TRUNCATE TABLE cuando se necesite eliminar todas las filas de una determinada tabla sin especificar la cláusula WHERE. En este caso, TRUNCATE usa menos recursos que DELETE, por tanto, es mucho más rápida la transacción.

La especificación de la sentencia DELETE sobre la tabla PRODUCTO es la siguiente:

```
Usando la tabla PRODUCTO se realizarán las siguientes eliminaciones:
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
Eliminando todos los registros de la tabla PRODUCTO
DELETE PRODUCTO
GO

Eliminando todos los registros de la tabla PRODUCTO cuyo stock actual sea
inferior a 50
DELETE PRODUCTO
WHERE SAC_PRO<50
GO
```

3.7 Manejo de datos masivos en SQL Server

El Motor de base de datos de Microsoft SQL Server permite importar y exportar masivamente datos entre una tabla de SQL Server y un archivo de datos que podría ser un texto plano txt.

SQL Server hace referencia a la importación y exportación de la siguiente manera:

- **Exportación masiva:** Se refiere a la copia de datos de una tabla de SQL Server en un archivo de datos.
- **Importación masiva:** Significa cargar datos de un archivo de datos a una tabla de SQL Server.

Los métodos básicos disponibles en el Motor de base de datos son:

- BCP: Utilidad de línea de comandos que permite la importación y exportación de datos masivos y genera archivos de formato.
- BULK INSERT: Instrucción de Transact SQL que importa datos directamente de un archivo de datos en una tabla de una base de datos. Esta instrucción no permite la exportación de datos.
- OPENROWSET (BULK): Instrucción Transact SQL que usa el proveedor de conjunto de filas BULK OPENROWSET para importar masivamente desde un archivo externo hacia una tabla de SQL Server; es especialmente usado para la importación de imágenes masivas.

3.7.1 Instrucción Bulk Insert

La instrucción BULK INSERT importa un archivo de datos en una tabla con un formato definido por el usuario.

```
BULK INSERT <NOMBRE_TABLA>
      FROM 'ARCHIVO_DATOS'
     WITH (
       FIELDTERMINATOR = <SIMBOLO_CAMPO_TERMINADOR>
       FIRSTROW = <NUMERO_FILA>
       ROWTERMINATOR = <SIMBOLO_FILA_TERMINADOR>
      )
```

- <NOMBRE_TABLA>: Es el nombre de la tabla o vista en la que se va a realizar una importación masiva de datos. Solo se pueden utilizar vistas en las que todas las columnas hagan referencia a la misma tabla base.
- <ARCHIVO_DATOS>: Es la ruta de acceso completa al archivo de datos que contiene los datos que se van a importar en la tabla o vista especificada. BULK INSERT puede importar datos desde un disco (incluidos una ubicación de red, disquete, disco duro, etc.). Por ejemplo, si el archivo se encuentra en la unidad C: y se llama Fuente.txt, entonces se puede especificar de la siguiente manera: FROM 'C:\FUENTE.TXT'.
- FIELDTERMINATOR: Especifica el terminador de campo que se va a utilizar para archivos de datos de tipo texto; hay que considerar que los datos dentro del archivo de texto siempre deben contar con el mismo separador de campos, el cual puede ser cualquier carácter.
- FIRSTROW: Especifica el número de la primera fila que se va a cargar. El valor predeterminado es la primera fila del archivo de datos especificado.

- **ROWTERMINATOR:** Especifica el terminador de fila que se va a utilizar para archivos de datos de tipo Char y Varchar. El terminador de fila predeterminado es \n (carácter de nueva línea).

La especificación de la sentencia BULK INSERT sobre la tabla PRODUCTO es la siguiente:

Usando la tabla PRODUCTO, se realizará el volcado de la información:

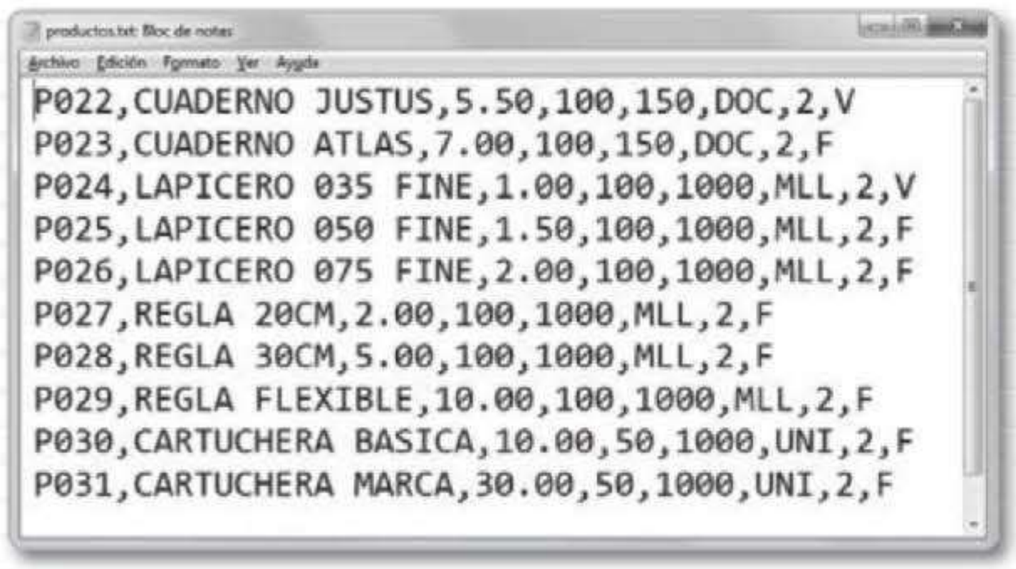
PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

Inicialmente, los registros de la tabla PRODUCTO se muestran de la siguiente manera:

	COD_PRO	DES_PRO	PRE_PRO	SAC_PRO	SMI_PRO	UNI_PRO	LIN_PRO	IMP_PRO
1	P001	PAPEL BOND A-4	18.50	200	1500	MLL	2	V
2	P002	PAPEL BOND OFICIO	37.00	100	1500	MLL	2	F
3	P003	PAPEL BULKY	12.00	498	1000	MLL	2	V
4	P004	PAPEL PERIODICO 75 GRAMOS	11.00	4285	1000	MLL	2	F
5	P005	CARTUCHO TINTA NEGRA	21.00	100	30	UNI	1	F
6	P006	CARTUCHO TINTA COLOR	47.00	58	35	UNI	1	F
7	P007	PORTA DISKETTES	5.50	300	100	UNI	1	V
8	P008	CAJA DE DISKETTES * 10	32.00	125	180	UNI	1	F
9	P009	BORRADOR DE TINTA	12.00	100	500	DOC	3	F
10	P010	BORRADOR BLANCO	5.00	2000	400	DOC	3	F
11	P011	TAJADOR METAL	22.00	1120	300	DOC	3	F
12	P012	TAJADOR PLASTICO	14.00	608	300	DOC	3	F
13	P013	FOLDER MANILA OFICIO	22.00	200	150	CIE	3	F
14	P014	FOLDER MANILA A-4	22.00	150	150	CIE	3	V
15	P015	SOBRE MANILA OFICIO	17.00	300	130	CIE	3	F
16	P016	SOBRE MANILA A-4	20.00	200	100	CIE	3	F
17	P017	LAPICERO NEGRO	12.00	3000	1000	DOC	3	F
18	P018	LAPICERO AZUL	12.00	2013	1500	DOC	3	F
19	P019	LAPICERO ROJO	10.00	1900	1000	DOC	3	F
20	P020	FOLDER PLASTICO A-4	52.00	3080	1100	CIE	3	F
21	P021	PROTECTOR DE PANTALLA	52.00	40	5	UNI	1	F

Se cuenta con un archivo de texto llamado Productos.txt, el cual debe encontrarse guardado en la unidad C: de la computadora. Los datos que contiene son:



Con la siguiente sentencia, se aplica el volcado desde el archivo de texto hacia la tabla PRODUCTO.

```
BULK INSERT PRODUCTO
        FROM 'C:\PRODUCTOS.TXT'
        WITH(
                FIELDTERMINATOR=',',
                ROWTERMINATOR='\n',
                FIRSTROW=1
        )
GO
```

También se podría usar la siguiente sentencia:

```
BULK INSERT PRODUCTO
        FROM 'C:\PRODUCTOS.TXT'
        WITH(
                FIELDTERMINATOR=',',
                ROWTERMINATOR='\n'
      )
GO
```

Finalmente, los registros de la tabla PRODUCTO se muestran de la siguiente manera:

	COD_PRO	DES_PRO	PRE_PRO	SAC_PRO	SMI_PRO	UNI_PRO	LIN_PRO	IMP_PRO
1	P001	PAPEL BOND A-4	18.50	200	1500	MLL	2	V
2	P002	PAPEL BOND OFICIO	37.00	100	1500	MLL	2	F
3	P003	PAPEL BULKY	12.00	498	1000	MLL	2	V
4	P004	PAPEL PERIODICO 75 GRAMOS	11.00	4285	1000	MLL	2	F
5	P005	CARTUCHO TINTA NEGRA	21.00	100	30	UNI	1	F
6	P006	CARTUCHO TINTA COLOR	47.00	58	35	UNI	1	F
7	P007	PORTA DISKETTES	5.50	300	100	UNI	1	V
8	P008	CAJA DE DISKETTES * 10	32.00	125	180	UNI	1	F
9	P009	BORRADOR DE TINTA	12.00	100	500	DOC	3	F
10	P010	BORRADOR BLANCO	5.00	2000	400	DOC	3	F
11	P011	TAJADOR METAL	22.00	1120	300	DOC	3	F
12	P012	TAJADOR PLASTICO	14.00	608	300	DOC	3	F
13	P013	FOLDER MANILA OFICIO	22.00	200	150	CIE	3	F
14	P014	FOLDER MANILA A-4	22.00	150	150	CIE	3	V
15	P015	SOBRE MANILA OFICIO	17.00	300	130	CIE	3	F
16	P016	SOBRE MANILA A-4	20.00	200	100	CIE	3	F

17	P017	LAPICERO NEGRO	12.00	3000	1000	DOC	3	F
18	P018	LAPICERO AZUL	12.00	2013	1500	DOC	3	F
19	P019	LAPICERO ROJO	10.00	1900	1000	DOC	3	F
20	P020	FOLDER PLASTICO A-4	52.00	3080	1100	CIE	3	F
21	P021	PROTECTOR DE PANTALLA	52.00	40	5	UNI	1	F
22	P022	CUADERNO JUSTUS	5.50	100	150	DOC	2	V
23	P023	CUADERNO ATLAS	7.00	100	150	DOC	2	F
24	P024	LAPICERO 035 FINE	1.00	100	1000	MLL	2	V
25	P025	LAPICERO 050 FINE	1.50	100	1000	MLL	2	F
26	P026	LAPICERO 075 FINE	2.00	100	1000	MLL	2	F
27	P027	REGLA 20CM	2.00	100	1000	MLL	2	F
28	P028	REGLA 30CM	5.00	100	1000	MLL	2	F
29	P029	REGLA FLEXIBLE	10.00	100	1000	MLL	2	F
30	P030	CARTUCHERA BASICA	10.00	50	1000	UNI	2	F
31	P031	CARTUCHERA MARCA	30.00	50	1000	UNI	2	F

3.8 Sentencia Merge

Realiza múltiples acciones sobre una misma tabla; para determinar qué acción tomar se debe condicionar e implementar una serie de criterios. MERGE puede realizar operaciones de inserción, actualización o eliminación en una misma tabla. Muchos administradores de base de datos usan la sentencia MERGE para sincronizar información de las tablas.

Las características de la sentencia MERGE son:

- **Sincronizar los datos de dos tablas**

  Permite realizar una comparación entre dos tablas con la misma estructura, quizás en una copia de seguridad, ya que MERGE puede igualar registros entre dos tablas.

- **Registrar la información**

  Cuando se maneja nueva información en la tabla, se debe estar seguro de que la información es nueva; si es así, se inserta la información; caso contrario, se puede actualizar con los datos. Por ejemplo, cuando alguien se registra en una web muchas veces, la primera vez lo registra como un nuevo valor; las veces siguientes, simplemente actualizará sus valores.

La especificación de la sentencia MERGE sobre la tabla PRODUCTO es la siguiente:

```
Usando la tabla PRODUCTO, se aplica la sentencia MERGE:
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
Implementar un MERGE que permita registrar un nuevo producto. Si este
producto ya se encuentra registrado, solo actualizará sus datos; caso
contrario, lo registrará como un nuevo registro.

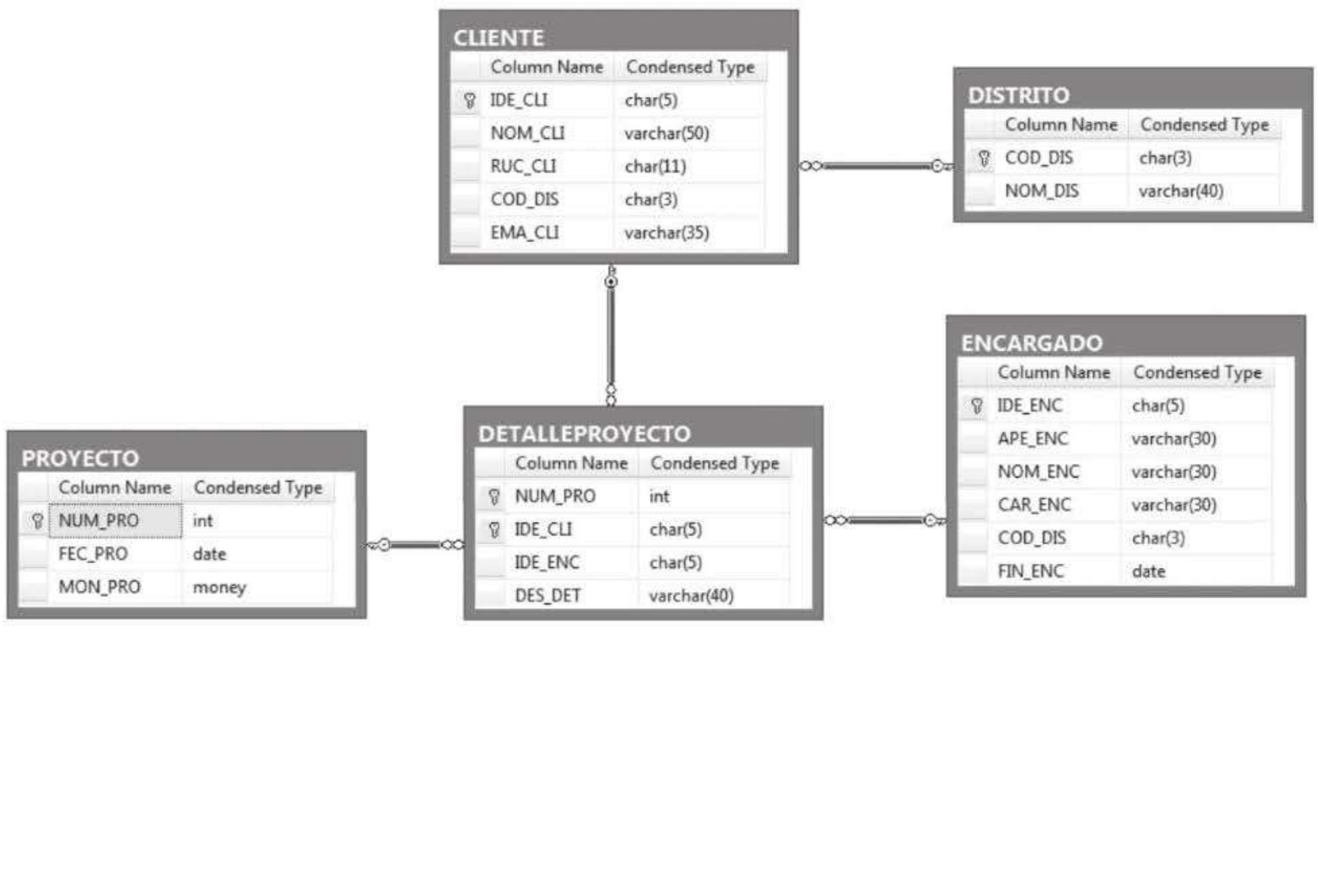
--Declaración de variables
DECLARE @COD CHAR(4),@DES VARCHAR(40),@PRE MONEY,@SAC INT,@SMI INT,@UNI
VARCHAR(30),@LIN INT,@IMP CHAR(1)
SET @COD='P001'
SET @DES='PAPEL BOND A-4 ATLAS'
SET @PRE=18.50
SET @SAC=200
SET @SMI=1500
SET @UNI='MLL'
SET @LIN='2'
SET @IMP='V'

--Aplicando la sentencia MERGE
MERGE PRODUCTO AS TARGET
       USING (SELECT @COD,@DES,@PRE,@SAC,@SMI,@UNI,@LIN,@IMP)
       AS  SOURCE  (COD_PRO,DES_PRO,PRE_PRO,SAC_PRO,SMI_PRO,UNI_PRO,LIN_
PRO,IMP_PRO)
       ON (TARGET.COD_PRO = SOURCE.COD_PRO)
       WHEN MATCHED THEN
       UPDATE SET DES_PRO=SOURCE.DES_PRO,PRE_PRO=SOURCE.PRE_PRO,
                          SAC_PRO=SOURCE.SAC_PRO,SMI_PRO=SOURCE.SMI_PRO,
                          UNI_PRO=SOURCE.UNI_PRO,LIN_PRO=SOURCE.LIN_PRO,
                          IMP_PRO=SOURCE.IMP_PRO
       WHEN NOT MATCHED THEN
       INSERT VALUES(SOURCE.COD_PRO, SOURCE.DES_PRO,SOURCE.PRE_PRO,
                     SOURCE.SAC_PRO,SOURCE.SMI_PRO,SOURCE.UNI_PRO,
                          SOURCE.LIN_PRO,SOURCE.IMP_PRO);
GO
```

Caso desarrollado 1

Cree la base de datos PROYECTOS_INDUSTRIALES teniendo en cuenta los siguientes aspectos:

- Validar la base de datos y crear el siguiente modelo de base de datos:



- Aplicar las siguientes restricciones:
  - Aplicar la propiedad Identity a la columna NUM_PRO de la tabla PROYECTO con un inicio de uno.
  - En el correo del cliente, mostrar el mensaje "No cuenta" si el usuario no registra dicho dato.
  - En el monto del proyecto, el valor debe ser superior a cero.
  - En la carrera del encargado, solo se podrán registrar valores como "Jefe-Contador-Supervisor-Vendedor".
  - El número de RUC del cliente debe ser un valor único por cliente.
- Modificar la capacidad de caracteres a 100 de la descripción del detalle de proyecto.
- Agregar el campo teléfono "TEL_CLI" y número de hijos "HIJ_CLI" a la tabla CLIENTE.
- Eliminar el campo HIJ_CLI de la tabla CLIENTE.
- Listar las tablas implementadas en la base de datos.
- Listar los campos de cada tabla implementada.
- Realizar la inserción de registros a cada tabla de la base de datos PROYECTOS INDUSTRIALES.
- Realizar las siguientes actualizaciones:
  - Modificar el correo electrónico del cliente "María López" por maria_lopez@gmail.com.
  - Modificar los números de teléfono de todos los clientes por '0000000'.
  - Disminuir en $5000.00 a todos los proyectos registrados en el primer semestre del año 2012.
- Implementar la sentencia MERGE para un distrito; en caso existiera, actualizar la información; caso contrario, insertarlo como un nuevo distrito.

Solución:

- Validar la base de datos y crear el siguiente modelo de base de datos:

```
--Activando la base de datos maestra
USE MASTER
GO
--Validando la existencia de la base de datos
IF DB_ID('PROYECTOSINDUSTRIALES') IS NOT NULL
BEGIN
        DROP DATABASE PROYECTOSINDUSTRIALES
END
--Creando la base de datos
CREATE DATABASE PROYECTOSINDUSTRIALES
GO
--Activando la base de datos
USE PROYECTOSINDUSTRIALES
GO
```

```
--Definir el formato de fecha
SET DATEFORMAT DMY
GO

--Creación de tablas
CREATE TABLE CLIENTE(
        IDE_CLI         CHAR(5)         NOT NULL,
        NOM_CLI         VARCHAR(50)     NOT NULL,
        RUC_CLI         CHAR(11)        NOT NULL,
        COD_DIS         CHAR(3)         NOT NULL,
        EMA_CLI         VARCHAR(35)     NULL
)
GO

CREATE TABLE DISTRITO(
        COD_DIS         CHAR(3)         NOT NULL,
        NOM_DIS         VARCHAR(40)     NOT NULL
)
GO

CREATE TABLE ENCARGADO(
        IDE_ENC         CHAR(5)         NOT NULL,
        APE_ENC         VARCHAR(30)     NOT NULL,
        NOM_ENC         VARCHAR(30)     NOT NULL,
        CAR_ENC         VARCHAR(30)     NOT NULL,
        COD_DIS         CHAR(3)         NOT NULL,
        FIN_ENC         DATE            NOT NULL
)
GO

CREATE TABLE PROYECTO(
        NUM_PRO INT     NOT NULL IDENTITY(1,1),
        FEC_PRO DATE    NOT NULL,
        MON_PRO MONEY   NOT NULL
)
GO

CREATE TABLE DETALLEPROYECTO(
        NUM_PRO         INT             NOT NULL,
        IDE_CLI         CHAR(5)         NOT NULL,
        IDE_ENC         CHAR(5)         NOT NULL,
        DES_DET         VARCHAR(40)     NOT NULL
)
GO

--Asignando llaves primarias
ALTER TABLE PROYECTO
        ADD PRIMARY KEY NONCLUSTERED (NUM_PRO)
ALTER TABLE ENCARGADO
        ADD PRIMARY KEY NONCLUSTERED (IDE_ENC)
ALTER TABLE CLIENTE
        ADD PRIMARY KEY NONCLUSTERED (IDE_CLI)
ALTER TABLE DETALLEPROYECTO
        ADD PRIMARY KEY NONCLUSTERED (NUM_PRO,IDE_CLI)
ALTER TABLE DISTRITO
        ADD PRIMARY KEY NONCLUSTERED (COD_DIS)
```

```
--Asignando llaves foráneas
ALTER TABLE DETALLEPROYECTO
       ADD FOREIGN KEY (IDE_CLI) REFERENCES CLIENTE
ALTER TABLE DETALLEPROYECTO
       ADD FOREIGN KEY (NUM_PRO) REFERENCES PROYECTO
ALTER TABLE DETALLEPROYECTO
       ADD FOREIGN KEY (IDE_ENC) REFERENCES ENCARGADO
ALTER TABLE CLIENTE
       ADD FOREIGN KEY (COD_DIS) REFERENCES DISTRITO
GO
```

- Aplicar las siguientes restricciones:
  - Aplicar la propiedad Identity a la columna NUM_PRO de la tabla PROYECTO con un inicio de uno.

```
CREATE TABLE PROYECTO(
       NUM_PRO INT     NOT NULL IDENTITY(1,1),
       FEC_PRO DATE    NOT NULL,
       MON_PRO MONEY   NOT NULL
)
GO
```

  - En el correo del cliente, mostrar el mensaje "No cuenta" si el usuario no registra dicho dato.

```
ALTER TABLE CLIENTE
 ADD DEFAULT 'NO CUENTA'
 FOR EMA_CLI
GO
```

  - En el monto del proyecto, el valor debe ser superior a cero.

```
ALTER TABLE PROYECTO
 ADD CONSTRAINT CHK_MONTO
 CHECK (MON_PRO>0)
GO
```

  - En la carrera del encargado, solo se podrán registrar valores como "Jefe-Contador -Vendedor".

```
ALTER TABLE ENCARGADO
 ADD CONSTRAINT CHK_CARRERA
 CHECK (CAR_ENC IN('JEFE','CONTADOR','SUPERVISOR','VENDEDOR'))
GO
```

- El número de RUC del cliente debe ser un valor único por cliente.

```
ALTER TABLE CLIENTE
 ADD CONSTRAINT UNI_RUC
 UNIQUE (RUC_CLI)
GO
```

- Modificar la capacidad de caracteres a 100 de la descripción del detalle de proyecto.

```
ALTER TABLE DETALLEPROYECTO
 ALTER COLUMN DES_DET VARCHAR(100)
GO
```

- Agregar el campo teléfono "TEL_CLI" y número de hijos "HIJ_CLI" a la tabla CLIENTE.

```
ALTER TABLE CLIENTE
 ADD TEL_CLI CHAR(15),HIJ_CLI INT
GO
```

- Eliminar el campo HIJ_CLI de la tabla CLIENTE.

```
ALTER TABLE CLIENTE
 DROP COLUMN HIJ_CLI
GO
```

- Listar las tablas implementadas en la base de datos.

```
SP_TABLES
GO
```

- Listar los campos de cada tabla implementada.

```
SP_COLUMNS CLIENTE
GO
SP_COLUMNS PROYECTO
GO
SP_COLUMNS DISTRITO
GO
SP_COLUMNS DETALLEPROYECTO
GO
SP_COLUMNS ENCARGADO
GO
```

- Realizar la inserción de registros a cada tabla de la base de datos PROYECTOS INDUSTRIALES.

```
INSERT INTO DISTRITO VALUES('L01','LIMA')
INSERT INTO DISTRITO VALUES('L02','LINCE')
INSERT INTO DISTRITO VALUES('L03','BREÑA')
INSERT INTO DISTRITO VALUES('L04','LOS OLIVOS')
INSERT INTO DISTRITO VALUES('L05','RIMAC')
INSERT INTO DISTRITO VALUES('L06','SAN MARTIN DE PORRES')
INSERT INTO DISTRITO VALUES('L07','SAN LUIS')
INSERT INTO DISTRITO VALUES('L08','SAN BORJA')

INSERT INTO CLIENTE VALUES('CL001','LOPEZ
MARIA','10856985252','L03','MLOPEZ@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL002','GOMEZ
JOSE','10856111252','L02','JGOMEZ@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL003','ACOSTA
JESUS','10856222252','L01','JACOSTA@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL004','ARIAS
GUADSLUPE','10833385252','L04','GARIAS@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL005','CARLOS
MANUEL','10854445252','L08','MCARLOS@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL006','GERZ
BRIGITTE','10856555252','L06','BGERZ@HOTMAIL.COM',DEFAULT)
INSERT INTO CLIENTE VALUES('CL007','BARBOSA
MARISOL','10555985252','L04','MBARBOSA@HOTMAIL.COM',DEFAULT)

INSERT INTO ENCARGADO
VALUES('E0001','LUIS','GRIMALDO','JEFE','L01','01/02/2010')
INSERT INTO ENCARGADO VALUES
('E0002','ROJAS','FILIMON','CONTADOR','L02','04/05/2010')
INSERT INTO ENCARGADO
VALUES('E0003','SALAZAR','RODOLFO','SUPERVISOR','L01','05/05/2010')
INSERT INTO ENCARGADO
VALUES('E0004','VILCHEZ','ALEJANDRO','VENDEDOR','L03','01/03/2011')
INSERT INTO ENCARGADO
VALUES('E0005','JACINTO','MARTIN','JEFE','L04','04/07/2011')
INSERT INTO ENCARGADO
VALUES('E0006','JUAREZ','CARLOS','CONTADOR','L01','01/08/2011')
INSERT INTO ENCARGADO
VALUES('E0007','LAMAS','EDUARDO','VENDEDOR','L06','03/09/2011')
INSERT INTO ENCARGADO
VALUES('E0008','PALACIOS','ELVIS','JEFE','L04','01/02/2012')
INSERT INTO ENCARGADO
VALUES('E0009','PRIETO','CARLOS','SUPERVISOR','L01','03/04/2012')
INSERT INTO ENCARGADO
VALUES('E0010','CONTRERAS','MARIA','SUPERVISOR','L07','02/06/2012')
INSERT INTO ENCARGADO
VALUES('E0011','LAZARO','JUANA','VENDEDOR','L08','02/07/2012')

INSERT INTO PROYECTO VALUES('01/05/2012',100000)
INSERT INTO PROYECTO VALUES('11/05/2012',300000)
INSERT INTO PROYECTO VALUES('23/05/2012',140000)
INSERT INTO PROYECTO VALUES('21/06/2012',150000)
INSERT INTO PROYECTO VALUES('21/06/2012',10000)
INSERT INTO PROYECTO VALUES('16/07/2012',330000)
INSERT INTO PROYECTO VALUES('15/07/2012',220000)
```

```
INSERT INTO PROYECTO VALUES('10/08/2012',260000)
INSERT INTO PROYECTO VALUES('08/08/2012',140000)
INSERT INTO PROYECTO VALUES('06/09/2012',160000)
INSERT INTO PROYECTO VALUES('02/10/2012',246000)

INSERT INTO DETALLEPROYECTO VALUES(1,'CL001','E0001','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(2,'CL002','E0011','OBRA AREQUIPA')
INSERT INTO DETALLEPROYECTO VALUES(3,'CL001','E0002','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(4,'CL001','E0002','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(5,'CL004','E0003','OBRA AREQUIPA')
INSERT INTO DETALLEPROYECTO VALUES(6,'CL007','E0003','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(7,'CL005','E0004','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(8,'CL002','E0005','OBRA AREQUIPA')
INSERT INTO DETALLEPROYECTO VALUES(9,'CL001','E0006','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(10,'CL002','E0007','OBRA LIMA')
INSERT INTO DETALLEPROYECTO VALUES(11,'CL003','E0005','OBRA AREQUIPA')

--Listando los registros
SELECT * FROM CLIENTE
SELECT * FROM PROYECTO
SELECT * FROM DISTRITO
SELECT * FROM DETALLEPROYECTO
SELECT * FROM ENCARGADO
```

- Realizar las siguientes actualizaciones:
  - Modificar el correo electrónico del cliente "María López" por maria_lopez@gmail.com.

```
UPDATE CLIENTE
        SET EMA_CLI = 'MARIA_LOPEZ@GMAIL.COM'
        WHERE NOM_CLI='LOPEZ MARIA'
GO
```

  - Modificar los números de teléfono de todos los clientes por '0000000'.

```
UPDATE CLIENTE
        SET TEL_CLI = '0000000'
        WHERE TEL_CLI IS NULL
GO
```

  - Disminuir en $5000.00 a todos los proyectos registrados en el primer semestre del año 2012.

```
UPDATE PROYECTO
        SET MON_PRO -= 5000
        WHERE YEAR(FEC_PRO) = 2012 AND MONTH(FEC_PRO) BETWEEN 1 AND 6
GO
```

- Implementar la sentencia MERGE para un distrito; en caso existiera, actualizar la información; caso contrario, insertarlo como un nuevo distrito.

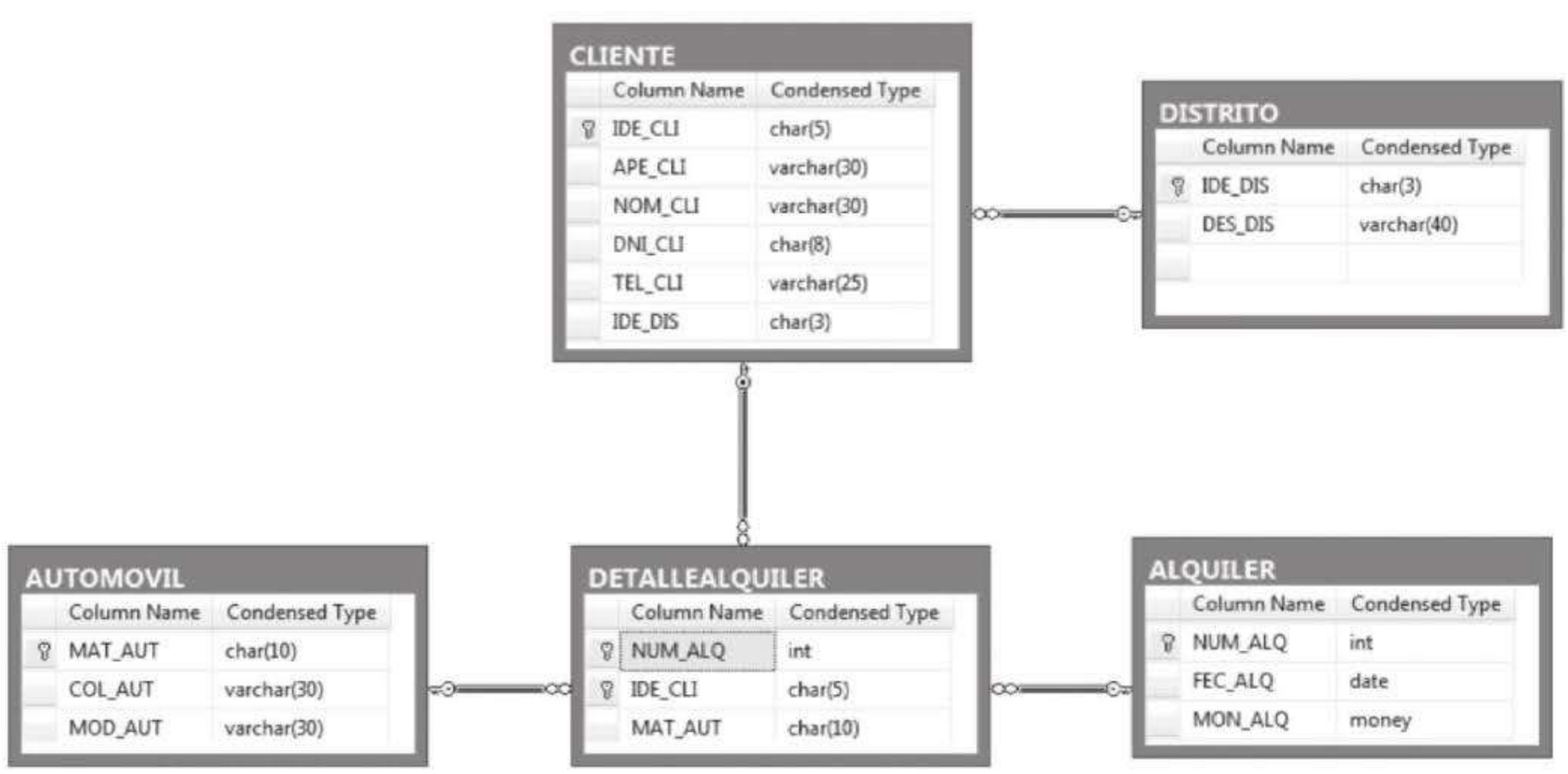
```
--Declaración de variables
DECLARE @COD CHAR(4),@DIS VARCHAR(40)
SET @COD='L09'
SET @DIS='LINCE'

--Aplicando la sentencia MERGE
MERGE DISTRITO AS TARGET
        USING (SELECT @COD,@DIS)
        AS SOURCE (COD_DIS,NOM_DIS)
        ON (TARGET.NOM_DIS = SOURCE.NOM_DIS)
        WHEN MATCHED THEN
        UPDATE SET NOM_DIS=SOURCE.NOM_DIS
        WHEN NOT MATCHED THEN
        INSERT VALUES(SOURCE.COD_DIS, SOURCE.NOM_DIS);
GO
```

Caso desarrollado 2

Cree la base de datos RENT-CAR teniendo en cuenta los siguientes aspectos:

- Validar la base de datos y crear el siguiente modelo de base de datos:



- Aplicar las siguientes restricciones:
  - En el teléfono del cliente, mostrar el mensaje "000-00000" si el usuario no registra dicho dato.
  - En el monto de alquiler, no permitir valores menores o iguales a cero.
  - Los colores permitidos para el registro del automóvil: "Rojo, Plata y Negro".

- Agregar el campo correo electrónico "COR_CLI" a la tabla cliente.
- Realizar la inserción de registros a cada tabla de la base de datos RENTCAR.
- Implementar la sentencia MERGE para un automóvil; en caso existiera, actualizar la información; caso contrario, insertarlo como un nuevo automóvil.

Solución:

```
--Activando la base de datos maestra
use master
go

--Validando la base de datos RENT-CAR
IF DB_ID('RENTCAR') IS NOT NULL
BEGIN
       DROP DATABASE RENTCAR
END

--Creando la base de datos RENT-CAR
CREATE DATABASE RENTCAR
GO

--Activando la base de datos RENTCAR
USE RENTCAR
GO

--Configurando el formato de fecha
SET DATEFORMAT DMY
GO

--Creando las tablas que componen la base de datos RENTCAR.
CREATE TABLE CLIENTE(
       IDE_CLI        CHAR(5)        NOT NULL,
       APE_CLI        VARCHAR(30)    NOT NULL,
       NOM_CLI        VARCHAR(30)    NOT NULL,
       DNI_CLI        CHAR(8)        NOT NULL,
       TEL_CLI        VARCHAR(25)    NULL DEFAULT '000-00000',
       IDE_DIS        CHAR(3)        NOT NULL
)
GO

CREATE TABLE DISTRITO(
       IDE_DIS        CHAR(3)        NOT NULL,
       DES_DIS        VARCHAR(40)    NOT NULL
)
GO

CREATE TABLE AUTOMOVIL(
       MAT_AUT        CHAR(10)       NOT NULL,
       COL_AUT        VARCHAR(30)    NOT  NULL  CHECK  (COL_AUT  IN('ROJO',
       'PLATA','NEGRO')),
       MOD_AUT        VARCHAR(30)    NOT NULL
)
GO
```

```
CREATE TABLE ALQUILER(
        NUM_ALQ         INT     NOT NULL,
        FEC_ALQ         DATE    NOT NULL,
        MON_ALQ         MONEY   NOT NULL CHECK (MON_ALQ>0)
)
GO

CREATE TABLE DETALLEALQUILER(
        NUM_ALQ         INT             NOT NULL,
        IDE_CLI         CHAR(5)         NOT NULL,
        MAT_AUT         CHAR(10)        NOT NULL
)
GO

--4. Implementando las llaves primarias a las tablas
ALTER TABLE AUTOMOVIL
        ADD PRIMARY KEY (MAT_AUT)
ALTER TABLE ALQUILER
        ADD PRIMARY KEY (NUM_ALQ)
ALTER TABLE CLIENTE
        ADD PRIMARY KEY (IDE_CLI)
ALTER TABLE DISTRITO
        ADD PRIMARY KEY (IDE_DIS)
ALTER TABLE DETALLEALQUILER
        ADD PRIMARY KEY (NUM_ALQ,IDE_CLI)

--5. Implementando las llaves foráneas a las tablas relacionadas
ALTER TABLE DETALLEALQUILER
        ADD FOREIGN KEY (NUM_ALQ) REFERENCES ALQUILER
ALTER TABLE DETALLEALQUILER
        ADD FOREIGN KEY (IDE_CLI) REFERENCES CLIENTE
ALTER TABLE DETALLEALQUILER
        ADD FOREIGN KEY (MAT_AUT) REFERENCES AUTOMOVIL
ALTER TABLE CLIENTE
        ADD FOREIGN KEY (IDE_DIS) REFERENCES DISTRITO
GO


--Agregar el campo correo electrónico “COR_CLI” a la tabla cliente.
ALTER TABLE CLIENTE
        ADD COR_CLI VARCHAR(50) NULL
GO

--Realizar la inserción de registros a cada tabla de la base de datos
RENTCAR.
--Insertando los registros a la table DISTRITO
INSERT INTO DISTRITO VALUES('L01','LIMA')
INSERT INTO DISTRITO VALUES('L02','LINCE')
INSERT INTO DISTRITO VALUES('L03','BREÑA')
INSERT INTO DISTRITO VALUES('L04','LOS OLIVOS')
INSERT INTO DISTRITO VALUES('L05','RIMAC')
INSERT INTO DISTRITO VALUES('L06','SAN MARTIN DE PORRES')
INSERT INTO DISTRITO VALUES('L07','SAN LUIS')
INSERT INTO DISTRITO VALUES('L08','SAN BORJA')
```

```
--Insertando los registros a la table CLIENTE
INSERT INTO CLIENTE VALUES('CL001','LOPEZ','MARIA','10856985','526-
5254','L03','')
INSERT INTO CLIENTE VALUES('CL002','GOMEZ','JOSE','02188525','585-
8212','L02','')
INSERT INTO CLIENTE VALUES('CL003','ACOSTA','JESUS','07415482','485-
1598','L01','')
INSERT INTO CLIENTE VALUES('CL004','ARIAS','GUADSLUPE','08995255','258-
8569','L04','')
INSERT INTO CLIENTE VALUES('CL005','CARLOS','MANUEL','10418525','296-
9685','L08','')
INSERT INTO CLIENTE VALUES('CL006','GERZ','BRIGITTE','10952155','585-
1254','L06','')
INSERT INTO CLIENTE VALUES('CL007','BARBOSA','MARISOL','78514855','584-
5869','L04','')

--Insertando los registros a la table ALQUILER
INSERT INTO ALQUILER VALUES('1','12/03/10',185)
INSERT INTO ALQUILER VALUES('2','02/01/11',100)
INSERT INTO ALQUILER VALUES('3','01/05/11',54)
INSERT INTO ALQUILER VALUES('4','11/08/12',85)
INSERT INTO ALQUILER VALUES('5','22/06/12',18)
INSERT INTO ALQUILER VALUES('6','30/01/12',15)
INSERT INTO ALQUILER VALUES('7','16/04/12',165)
INSERT INTO ALQUILER VALUES('8','23/05/12',135)
INSERT INTO ALQUILER VALUES('9','21/03/12',225)

--Insertando los registros a la table AUTOMOVIL
INSERT INTO AUTOMOVIL VALUES('AF-456','ROJO','SEDAN')
INSERT INTO AUTOMOVIL VALUES('D4-243','NEGRO','SEDAN')
INSERT INTO AUTOMOVIL VALUES('H5-455','ROJO','STATION WAGON')
INSERT INTO AUTOMOVIL VALUES('GG-654','ROJO','CAMIONETA')
INSERT INTO AUTOMOVIL VALUES('FD-463','PLATA','STATION WAGON')
INSERT INTO AUTOMOVIL VALUES('SF-112','ROJO','LI')
INSERT INTO AUTOMOVIL VALUES('FG-654','PLATA','L10')
INSERT INTO AUTOMOVIL VALUES('GT-532','ROJO','SEDAN')
INSERT INTO AUTOMOVIL VALUES('YT-457','NEGRO','STATION WAGON')

--Insertando detalle de alquiler
INSERT INTO DETALLEALQUILER VALUES('1','CL002','GG-654')
INSERT INTO DETALLEALQUILER VALUES('2','CL002','GT-532')
INSERT INTO DETALLEALQUILER VALUES('3','CL006','FD-463')
INSERT INTO DETALLEALQUILER VALUES('3','CL007','GT-532')
INSERT INTO DETALLEALQUILER VALUES('4','CL005','H5-455')
INSERT INTO DETALLEALQUILER VALUES('5','CL004','SF-112')
INSERT INTO DETALLEALQUILER VALUES('6','CL004','YT-457')
INSERT INTO DETALLEALQUILER VALUES('7','CL003','GT-532')

--Listando los registros
select * from CLIENTE
select * from ALQUILER
select * from DISTRITO
select * from AUTOMOVIL
select * from DETALLEALQUILER
```

```
--Implementar la sentencia MERGE para un automóvil; en caso existiera,
actualizar la información; caso contrario, insertarlo como un nuevo auto-
móvil.

--Declaración de variables
DECLARE @MAT CHAR(10),@COL VARCHAR(30),@MOD VARCHAR(30)
SET @MAT='HY-345'
SET @COL='PLATA'
SET @MOD='CHERY'

--Aplicando la sentencia MERGE
MERGE AUTOMOVIL AS TARGET
        USING (SELECT @MAT,@COL,@MOD)
        AS SOURCE (MAT_AUT,COL_AUT,MOD_AUT)
        ON (TARGET.MAT_AUT = SOURCE.MAT_AUT)
        WHEN MATCHED THEN
        UPDATE SET COL_AUT=SOURCE.COL_AUT, MOD_AUT=SOURCE.MOD_AUT
        WHEN NOT MATCHED THEN
        INSERT VALUES(SOURCE.MAT_AUT, SOURCE.COL_AUT,SOURCE.MOD_AUT);
GO
```



Capítulo 4

Recuperación de datos

Competencia tecnológica:

- Implementar consultas que permitan recuperar información desde las tablas de una base de datos.
- Implementar subconsultas obteniendo información desde las tablas de una base de datos.
- Implementar vistas obteniendo información desde las tablas de una base de datos.

Contenido:

4.1 Sentencia Select para la recuperación de registros
4.2 Consultas de registros básico
4.3 Consultas distinguidas
4.4 Consultas ordenadas
4.5 Consultas por cantidad de registros
4.6 Consultas con especificación de campos
4.7 Consultas con cabeceras
4.8 Consultas con campos calculados
4.9 Consultas que crean tablas de registros
4.10 Consultas condicionadas
4.11 Consultas internas
4.12 Consultas externas
4.13 Consultas agrupadas
4.14 Subconsultas
4.15 Vistas
- Caso desarrollado

4.1 Sentencia Select para la recuperación de registros

Es una sentencia que permite obtener información a partir de un criterio; es una de las principales características que presenta SQL Server como servicio de base de datos. El servidor interpreta la consulta y recupera los datos especificados desde las tablas. Tiene el siguiente formato general:

```
SELECT <ALL> <*>
  <DISTINCT>
  <TOP VALOR [PERCENT]>
  <ALIAS.CAMPO>
  <AS CAMPO>
  <INTO TABLA_DESTINO>
FROM <TABLA> <AS ALIAS>
   <INNER | LEFT | RIGHT | CROSS JOIN >
   <WHERE CONDICION>
   <GROUP BY CAMPO>
   <HAVING CONDICION>
   <ORDER BY CAMPO ASC | DESC>
```

Donde:

- ALL: Es considerado como un predicado dentro de la sentencia SELECT, el cual hace referencia a todas las filas contenidas en una tabla.
- * (Asterisco): Hace referencia a todas las columnas de una tabla.
- DISTINCT: Especifica que el conjunto de filas devueltas por la consulta no sean repetidas; estas filas deben ser únicas. Los valores asignados con NULL se consideran iguales desde el punto de vista de la cláusula DISTINCT.
- TOP: Especifica que el conjunto de filas devueltas por la consulta puede ser controlado en número y en porcentaje. Se considera que la muestra de registros devueltos siempre será el primer conjunto de filas especificadas. Para especificar el porcentaje solo agregue la palabra PERCENT.
- ALIAS.CAMPO: La especificación alias hace referencia a una tabla, y campo, a uno de los atributos de la tabla. Si se hace referencia a una sola tabla en la consulta, no será necesario la referencia alias; ya que al referenciar dos o más tablas, las columnas podrían repetirse, y alias hace que estos campos se diferencien. Por otro lado, para poder especificar varios campos se debe separar por comas (,) y considerar como número máximo de campos 4096.
- INTO: Permite crear una nueva tabla y enviar las filas resultantes del proceso de consulta.
- FROM: Permite especificar la tabla que está involucrada en la consulta; hay que tener en cuenta que aquí es donde se define el alias a la tabla.
- INNER: Permite unir a la consulta una tabla, haciendo una combinación de campos que luego serán parte del proceso de la consulta.
- WHERE: Esta cláusula permite condicionar el resultado de una consulta.
- GROUP BY: Permite agrupar un conjunto de registros, en forma de resumen, con los valores especificados de una o más columnas.

- **HAVING:** Es la condicional que se presenta solo cuando se especifica un GROUP BY, es decir, condiciona los registros que se presentan agrupados.
- **ORDER BY:** Permite ordenar los registros de acuerdo a una columna específica; no hay límite de criterio de ordenamiento. Hay que tener en cuenta que la cláusula ORDER BY ordena en base a la intercalación de las columnas participantes en el orden. Existe dos formas de intercalación de columnas: la intercalación de Windows y la intercalación de SQL. La tradicional es la intercalación Latin1_General, pero también se podría especificar con Traditional_Spanish.

4.2 Consultas de registros básico

Este tipo de consultas permite mostrar todos los registros de una tabla. Su uso se basa en mostrar rápidamente todo el contenido de una tabla.

Formato:

```
SELECT <ALL> <*>
  FROM <TABLA> <AS ALIAS>
GO
```

A continuación, algunos casos sobre consultas:

```
Listar todos los registros desde la tabla PRODUCTO.
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
Primera forma:
Al especificar un asterisco, se entiende que se lista por un lado todas las
filas, y por otro lado, todas las columnas de la tabla PRODUCTO.
SELECT * FROM PRODUCTO
GO

Segunda forma:
Al especificar la cláusula ALL, se hace referencia a todas las filas,
mientras que asterisco hace referencia a todas las columnas de la tabla
PRODUCTO.
SELECT ALL * FROM PRODUCTO
GO
```

```
Tercera forma:
Se le asigna un alias a la tabla PRODUCTO llamado "P"; por lo tanto, al
referenciar p.* se hace referencia a que se muestren todos los registros
de la tabla PRODUCTO.
SELECT P.* FROM PRODUCTO P
GO

Cuarta forma:
Cuando no se asigna un alias a la tabla, se puede referenciar a todas las
filas y columnas anteponiendo el nombre de la tabla al asterisco.
SELECT PRODUCTO.* FROM PRODUCTO
GO
```

4.3 Consultas distinguidas

Este tipo de consultas permite mostrar información que no presente valores repetidos. Por otra parte, hay que tener en cuenta que las consultas distinguidas se pueden usar en cuadros combinados o listas, en un lenguaje de programación.

Formato:

```
SELECT <DISTINCT> <CAMPO>
  FROM <TABLA>
GO
```

A continuación, algunos casos sobre consultas distinguidas:

```
Listar todos los tipos de unidades (UNI_PRO) que presentan los productos.
```

PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

```
Inicialmente, se pueden mostrar todos los registros de la columna UNI_PRO;
el resultado presenta las unidades solicitadas, pero se encuentran repe-
tidas:

SELECT UNI_PRO FROM PRODUCTO
GO
```

	UNI_PRO
1	MLL
2	MLL
3	MLL
4	MLL
5	UNI
6	UNI
7	UNI
8	UNI
9	DOC
10	DOC
11	DOC
12	DOC
13	CIE
14	CIE
15	CIE
16	CIE
17	DOC

```
Para evitar la repitencia de valores, se pueden ejecutar las siguientes
consultas:

SELECT DISTINCT UNI_PRO  FROM PRODUCTO
GO
```

	UNI_PRO
1	CIE
2	DOC
3	MLL
4	UNI

4.4 Consultas ordenadas

Este tipo de consultas permite mostrar información ordenada a partir de la especificación de los campos.

Formato:

```
SELECT <ALL> <*>
  <DISTINCT>
 FROM <TABLA> <AS ALIAS>
 <ORDER BY CAMPO ASC | DESC>
```

A continuación, algunos casos sobre consultas ordenadas:

Si se tiene la información referente a la tabla CLIENTE:

CLIENTE

Column Name	Condensed Type
COD_CLI	char(5)
RSO_CLI	char(30)
DIR_CLI	varchar(100)
TLF_CLI	char(9)
RUC_CLI	char(11)
COD_DIS	char(5)
FEC_REG	date
TIP_CLI	varchar(10)
CON_CLI	varchar(30)

- Mostrar todos los registros de CLIENTE ordenados por la columna RAZÓN SOCIAL de forma ascendente.

Primera forma:
En la cláusula ORDER BY se especifica la forma de ordenamiento mediante las opciones ASC (Ascendente) y DESC (Descendente).

```
SELECT *
 FROM CLIENTE ORDER BY RSO_CLI ASC
GO
```

Segunda forma:
Cuando no se especifica la opción de ordenamiento, automáticamente se asume el valor por defecto de orden ascendente.

```
SELECT *
 FROM CLIENTE ORDER BY RSO_CLI
GO
```

Tercera forma:
También se puede especificar el número de columnas de la tabla; esto servirá cuando un campo sea calculado y no tenga un nombre específico.

```
SELECT *
 FROM CLIENTE ORDER BY 2
GO
```

- Mostrar todos los registros de CLIENTE ordenados por el año de registro del cliente (FEC_REG) en forma descendente y, ante la repitencia, ordenar por meses en forma ascendente.

Primera forma:
Dos criterios de ordenamiento se especifican mediante el operador coma (,). Hay que tener en cuenta que, al especificar dos criterios, el segundo criterio se ordena en base al primero. En este caso, el servidor ordena por año en forma descendente y, dentro de este resultado, ordena de forma ascendente por los meses.

```
SELECT *
        FROM CLIENTE
        ORDER BY YEAR(FEC_REG) DESC,MONTH(FEC_REG)
GO

Segunda forma:
Se usa alias en la tabla.
SELECT C.*
        FROM CLIENTE C
        ORDER BY YEAR(C.FEC_REG) DESC,MONTH(C.FEC_REG) ASC
GO
```

4.5 Consultas por cantidad de registros

Este tipo de consultas permite controlar la cantidad de filas mostradas en el resultado.

Formato:

```
SELECT <ALL> <*>
  <DISTINCT>
  <TOP VALOR [PERCENT]>
 FROM <TABLA> <AS ALIAS>
 <ORDER BY CAMPO ASC | DESC>
```

A continuación, algunos casos sobre consultas por cantidad de registros:

```
Si se tiene la tabla PRODUCTO:
```

PRODUCTO	
Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

```
Listar los 5 primeros productos registrados en la base de datos.
SELECT TOP 5 * FROM PRODUCTO
GO
Listar los 3 productos más caros que se registran en la tabla.
SELECT TOP 3 *
        FROM PRODUCTO
        ORDER BY PRE_PRO DESC
GO
Listar el 50% de los primeros productos.
SELECT TOP 50 PERCENT *
        FROM PRODUCTO
GO
```

4.6 Consultas con especificación de campos

Este tipo de consultas permite especificar qué columnas se mostrarán en el resultado.

Formato:

```
SELECT <CAMPO 1>, <CAMPO 2>,... <CAMPO N>
<ALIAS.CAMPO>
FROM <TABLA> <AS ALIAS>
```

A continuación, algunos casos sobre consultas con especificación de campos:

Si se tiene la tabla PRODUCTO:

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

Listar los registros de los productos especificando las columnas, el código, la descripción, el precio y el *stock* actual.

Primera forma:

Sin especificación de alias.

```
SELECT COD_PRO,DES_PRO,PRE_PRO,SAC_PRO
        FROM PRODUCTO
GO
```

Segunda forma:

Con especificación de alias.

```
SELECT P.COD_PRO, P.DES_PRO, P.PRE_PRO, P.SAC_PRO
        FROM PRODUCTO P
GO
```

4.7 Consultas con cabeceras

Este tipo de consultas permite asignar una cabecera a las columnas del resultado.

Formato:

```
SELECT <ALIAS.CAMPO> <AS CAMPO>
  FROM <TABLA> <AS ALIAS>
```

A continuación, algunos casos sobre consultas con cabeceras:

```
Si se tiene la tabla PRODUCTO:
```

PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

```
Realizar el siguiente listado:
CÓDIGO DESCRIPCIÓN    PRECIO   STOCK ACTUAL
XXXXX  XXXXXXXXXXXXX  99.99    99
XXXXX  XXXXXXXXXXXXX  99.99    99
XXXXX  XXXXXXXXXXXXX  99.99    99

Primera forma:
Usando la cláusula AS antes de especificar el nombre de la cabecera.
SELECT COD_PRO AS CODIGO,DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,SAC_PRO AS [STOCK ACTUAL]
       FROM PRODUCTO
GO

Segunda forma:
La cabecera se especifica luego de la columna sin ningún operador.
SELECT COD_PRO CODIGO,DES_PRO DESCRIPCION,
       PRE_PRO PRECIO,SAC_PRO [STOCK ACTUAL]
       FROM PRODUCTO
GO

Tercera forma:
La cabecera se especifica con el operador igual.
SELECT CODIGO = COD_PRO, DESCRIPCION = DES_PRO,
       PRECIO = PRE_PRO, [STOCK ACTUAL] = SAC_PRO
       FROM PRODUCTO
GO

Cuando la cabecera presenta más de dos palabras se tiene que especificar
usando los operadores corchetes ([]) o comillas simples ('').

SELECT COD_PRO AS CODIGO,DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,SAC_PRO AS 'STOCK ACTUAL'
       FROM PRODUCTO
GO
```

4.8 Consultas con campos calculados

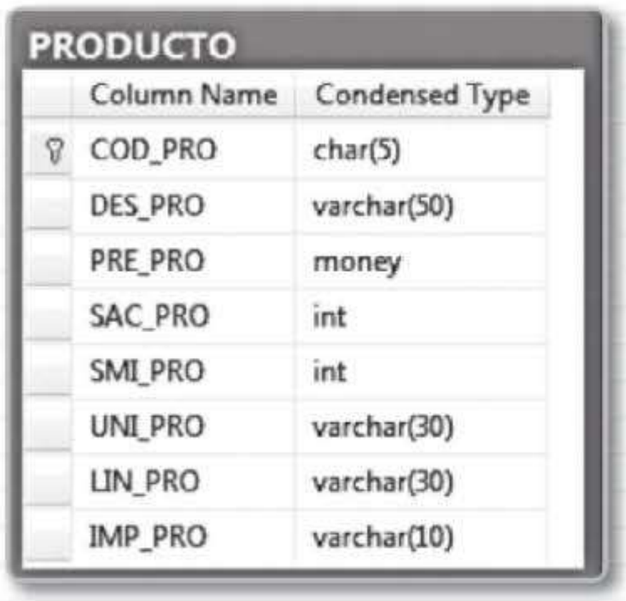
Este tipo de consultas permite mostrar columnas adicionales a la tabla; estas columnas se pueden obtener a partir de una expresión.

Formato:

```
SELECT <ALIAS.CAMPO><CAMPO CALCULADO>
  FROM <TABLA> <AS ALIAS>
```

A continuación, algunos casos sobre consultas con campos calculados:

- Si se tiene la tabla PRODUCTO:



PRODUCTO	
Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

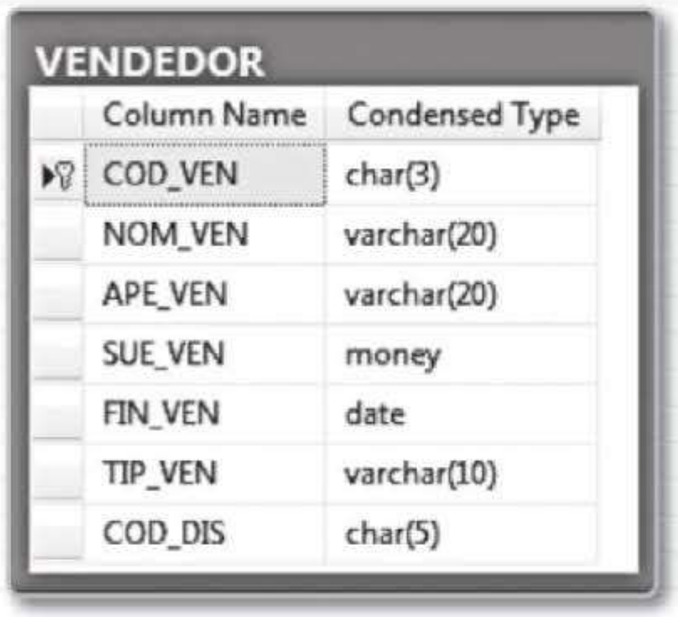
```
Realizar el siguiente listado:
CÓDIGO DESCRIPCIÓN       PRECIO      PRECIO VENTA (10% + DEL PRECIO)
XXXXX  XXXXXXXXXXXXXXX   99.99       99.99
XXXXX  XXXXXXXXXXXXXXX   99.99       99.99
XXXXX  XXXXXXXXXXXXXXX   99.99       99.99

SELECT COD_PRO AS CODIGO,DES_PRO AS DESCRIPCION,
 PRE_PRO AS PRECIO,(PRE_PRO*1.1) AS [PRECIO VENTA]
FROM PRODUCTO
GO

Realizar el siguiente listado:
CÓDIGO DESCRIPCIÓN      STOCK ACTUAL  STOCK MÍNIMO DIFERENCIA DE STOCK
XXXXX  XXXXXXXXXXXXXXX  99            99           99
XXXXX  XXXXXXXXXXXXXXX  99            99           99
XXXXX  XXXXXXXXXXXXXXX  99            99           99

SELECT COD_PRO AS CODIGO,DES_PRO AS DESCRIPCION,
 SAC_PRO AS [STOCK ACTUAL],SMI_PRO AS [STOCK MINIMO],
 (SAC_PRO-SMI_PRO) AS [DIFERENCIA DE STOCK]
FROM PRODUCTO
GO
```

- Si se tiene la tabla VENDEDOR:




```
Realizar el siguiente listado:
CÓDIGO VENDEDOR      (NOMBRE COMPLETO)     FECHA DE INICIO
XXXXX  XXXXXXX       XXXXXXXX              99/99/9999
XXXXX  XXXXXXX       XXXXXXXX              99/99/9999
XXXXX  XXXXXXX       XXXXXXXX              99/99/9999

Primera forma:
Para unir el nombre y apellido del vendedor se usa el operador más (+), el
cual permite unir o concatenar elementos de tipo de carácter. La función
SPACE permite asignar un espacio entre los campos; la cantidad de espacios
se especifica mediante un número como parámetro.

SELECT COD_VEN AS CODIGO, NOM_VEN+SPACE(1)+APE_VEN AS VENDEDOR,
       FIN_VEN AS [FECHA DE INICIO]
       FROM VENDEDOR
GO

Segunda forma:
Si se opta por no usar la función SPACE, se puede reemplazarla con una
pareja de comillas simples.

SELECT COD_VEN AS CODIGO, NOM_VEN+' '+APE_VEN AS VENDEDOR,
       FIN_VEN AS [FECHA DE INICIO]
       FROM VENDEDOR
GO
```

4.9 Consultas que crean tablas de registros

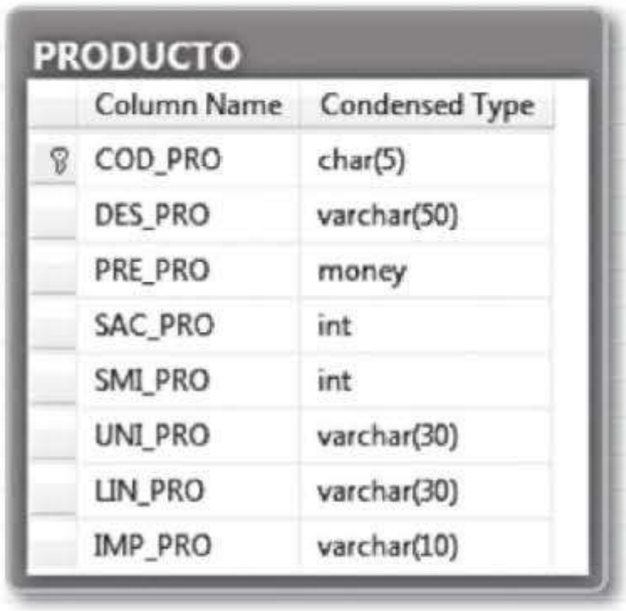
Este tipo de consultas permite crear una tabla con información resultante de una consulta, es decir, el resultado de la consulta se almacenará en la nueva tabla, formando de esta manera su estructura.

Formato:

```
SELECT <*>
  <ALIAS.CAMPO>
  <AS CAMPO>
  <INTO TABLA_DESTINO>
 FROM <TABLA> <AS ALIAS>
 <ORDER BY CAMPO ASC | DESC>
```

A continuación, algunos casos sobre consultas distinguidas:

Si se tiene la tabla PRODUCTO:



PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
Hacer una copia completa de la tabla PRODUCTO en una nueva tabla llamada
PRODUCTO_BAK.
SELECT * INTO PRODUCTO_BAK
       FROM PRODUCTO
GO

--Listando los productos de PRODUCTO_BAK
SELECT * FROM PRODUCTO_BAK
GO

Crear una tabla REPORTE_PRODUCTO a partir del código, descripción, precio
y tipo de unidad de los productos.
SELECT COD_PRO AS CODIGO, DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,UNI_PRO AS UNIDAD
       INTO REPORTE_PRODUCTO
       FROM PRODUCTO
GO

--Listando los productos del reporte
SELECT * FROM REPORTE_PRODUCTO
GO
```

4.10 Consultas condicionadas

Este tipo de consultas permite mostrar registros filtrados de una tabla; el filtro mostrará algunos registros que cumplan con una determinada condición.

Formato:

```
SELECT <ALL> <*>
   <ALIAS.CAMPO AS CAMPO>
FROM <TABLA> <AS ALIAS>
  <WHERE CONDICION>
  <ORDER BY CAMPO ASC | DESC>
```

Operadores de comparación

Cuando se implementa una condición, se tiene que hacer uso de ciertos operadores dependiendo del tipo de datos del valor a comparar.

Operador	Representacion
=	Comparación de igualdad entre dos valores
<>	Comparación de diferencia entre dos valores
>	Comparación del mayor valor
<	Comparación del menor valor
>=	Comparación del mayor o igual valor
<=	Comparación del menor o igual valor

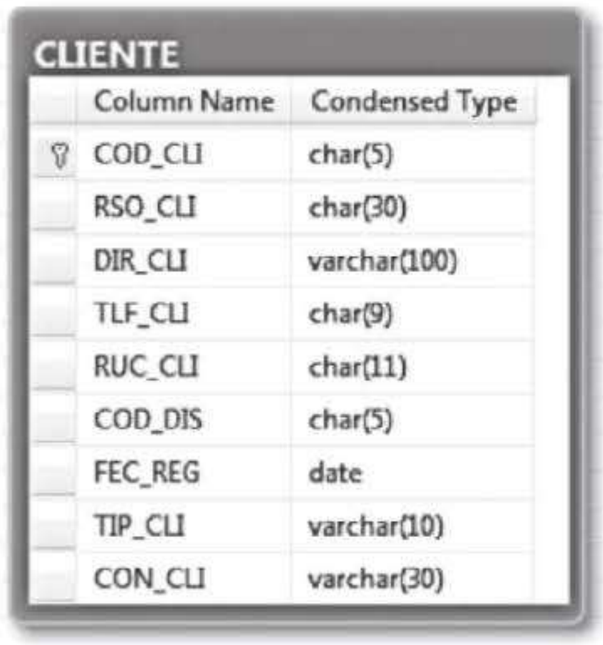
Operadores lógicos para el diseño de filtros

Los operadores lógicos tienen por misión comprobar la veracidad de alguna condición. Estos, como los operadores de comparación, devuelven el tipo de datos BOOLEAN con el valor Verdadero (TRUE), Falso (FALSE) o Desconocido (UNKNOWN).

AND	Representa a la Y lógica en la cual dos expresiones deben ser TRUE para poder devolver TRUE.
ANY	Devuelve TRUE si alguna expresión del conjunto de expresiones es TRUE.
BETWEEN	Devuelve TRUE si el valor se encuentra dentro de un rango, ya sea numérico o de cadena.
IN	Devuelve TRUE si el operando se encuentra dentro de una lista de valores específicos.
LIKE	Devuelve TRUE si el operando coincide a lo más con un patrón específico. Dicho patrón contiene la cadena de caracteres que se va a buscar en una expresión. Los comodines a usar son:
	% — Representa uno o más caracteres. Puede ser cualquier tipo de carácter, ya sea numérico, textual o símbolo.
	_ — Representa a un solo carácter de cualquier tipo.
	[ ] — Representa cualquier carácter individual dentro de un intervalo o conjunto de caracteres.
	[^] — Representa cualquier carácter individual fuera del intervalo especificado.
	IS NOT NULL — Representa que el contenido de una columna no esté vacía.
NOT	Invierte el valor booleano de una expresión del mismo tipo.
OR	Representa la O lógica, en la cual dos expresiones solo serán de tipo FALSE cuando ambas sean FALSE.
SOME	Devuelve TRUE si alguna de las comparaciones de un conjunto de comparaciones es TRUE.

A continuación, algunos casos sobre consultas condicionadas:

```
Si se tiene la información referente a la tabla CLIENTE:
```




```
Mostrar los registros de los clientes cuyo código de distrito sea D05 y,
además, el cliente se haya registrado en el año 2013.
SELECT C.*
       FROM CLIENTE C
       WHERE COD_DIS ='D05' AND YEAR(FEC_REG)=2013
GO

Mostrar los registros de los clientes que residen en San Miguel.
SELECT C.*
       FROM CLIENTE C
       WHERE C.COD_DIS = ANY(SELECT COD_DIS FROM DISTRITO
                                          WHERE NOM_DIS='SAN MIGUEL')
GO

Mostrar los registros de los clientes que se hayan registrado entre los
años 2008 al 2011.
SELECT C.*
       FROM CLIENTE C
       WHERE YEAR(C.FEC_REG) BETWEEN 2008 AND 2011
GO

Mostrar los registros de los clientes cuyo código de distrito sea D01,
D14 o D16.
SELECT C.*
       FROM CLIENTE C
       WHERE COD_DIS IN ('D01','D14','D16')
GO

Mostrar los registros de los clientes cuyo nombre de la razón social ini-
cie con la letra M.
SELECT C.*
       FROM CLIENTE C
       WHERE RSO_CLI LIKE 'M%'
GO
```

```
Mostrar los registros de los clientes cuyo nombre de contacto tenga como
segunda letra la A.
SELECT C.*
       FROM CLIENTE C
       WHERE CON_CLI LIKE '_A%'
GO

Mostrar los registros de los clientes cuyo nombre de razón social inicia
con las letras M, D o C.
SELECT C.*
       FROM CLIENTE C
       WHERE RSO_CLI LIKE '[MDC]%'
GO

Mostrar los registros de los clientes cuyo nombre de contacto tenga como
segunda letra la A, E o R.
SELECT C.*
       FROM CLIENTE C
       WHERE CON_CLI LIKE '_[AER]%'
GO

Mostrar los registros de los clientes cuya razón social no inicie con las
letras F, M o C.
SELECT C.*
       FROM CLIENTE C
       WHERE RSO_CLI LIKE '[^FMC]%'
GO

Mostrar los registros de los clientes cuyo número de RUC no presente un
valor nulo.
SELECT C.*
       FROM CLIENTE C
       WHERE RUC_CLI IS NOT NULL
GO

Mostrar los registros de los clientes cuyo año de registro sea diferente
al año 2012.
SELECT C.*
       FROM CLIENTE C
       WHERE NOT YEAR(FEC_REG) = 2012
GO


Mostrar los registros de los clientes con código C002 Y C007.
SELECT C.*
       FROM CLIENTE C
       WHERE C.COD_CLI='C002' OR C.COD_CLI='C007'
GO

Mostrar todos los distritos donde aún no se registran clientes.
SELECT *
       FROM DISTRITO
       WHERE COD_DIS = SOME(SELECT C.COD_DIS FROM CLIENTE C)
GO
```

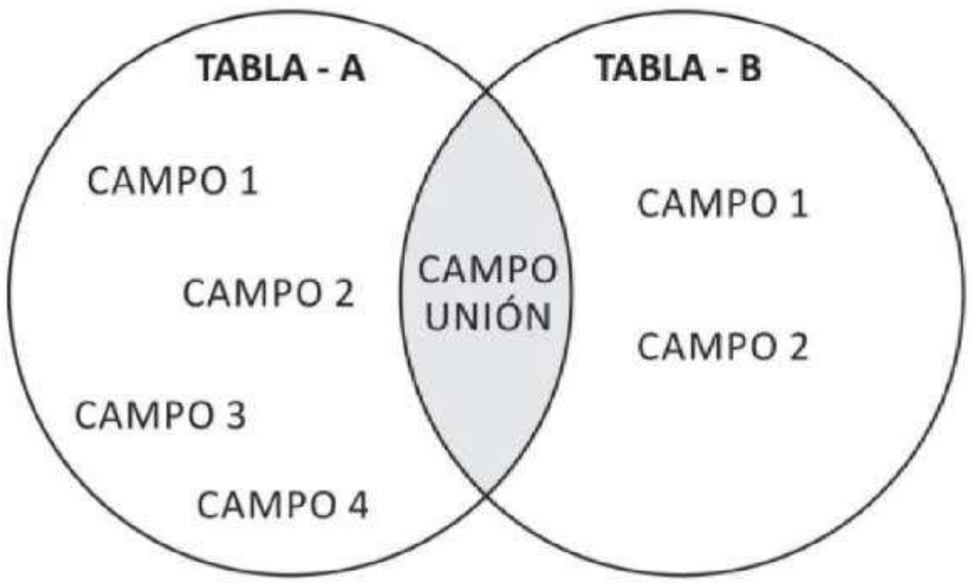
4.11 Consultas internas

Es conocida como combinación equivalente, en la cual los valores de las columnas de una tabla se comparan mediante el operador de igualdad a otros valores de una segunda tabla. Ante la coincidencia devuelve registros; es decir, si no encuentra valores iguales, no los muestra.

Cuando se asocian dos o más tablas, se debe tener en cuenta cuál es el campo de unión, ya que una tabla tiene un campo clave y la otra un campo secundario.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
FROM <TABLA-A> <AS ALIAS>
   <INNER JOIN TABLA-B ON CAMPO-A = CAMPO-B>
<ORDER BY CAMPO ASC | DESC>
```



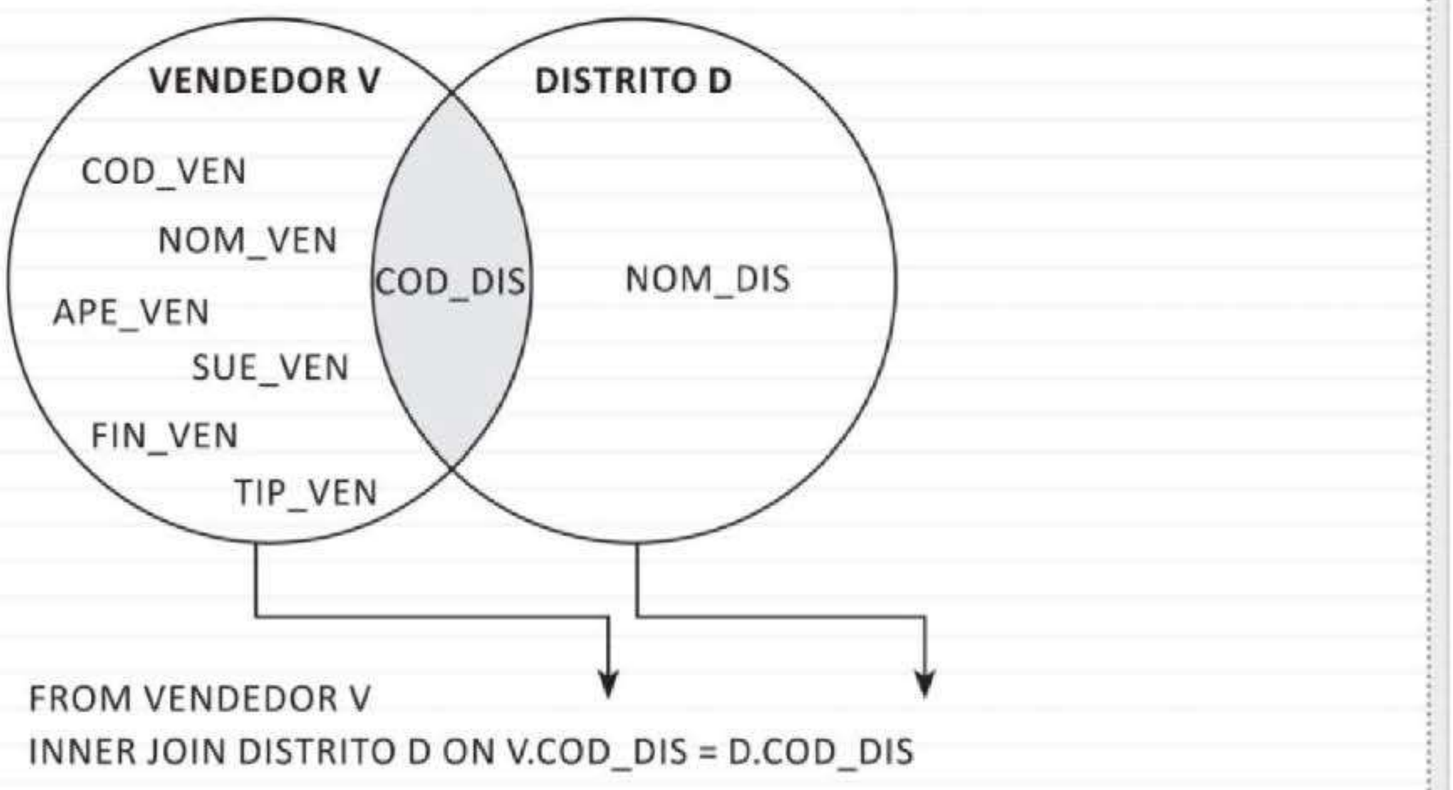
A continuación, algunos casos sobre consultas internas:

- Usando las tablas VENDEDOR y DISTRITO, realizar el siguiente listado:

```
CODIGO VENDEDOR                                SUELDO               FECHA DE INICIO DISTRITO
------ --------------------------------------- -------------------- --------------- --------------
V01    JUANA ALVA                              1000.00              2003-07-12      CHORRILLOS
V02    JUAN SOTO                               1200.00              2002-10-03      SAN ISIDRO
V03    CARLOS AREVALO                          1500.00              1992-02-10      SAN BORJA
V04    CESAR OJEDA                             850.00               1999-08-11      SURCO
V05    JULIO VEGA                              1500.00              1997-11-05      SURCO
V06    ANA ORTEGA                              1200.00              1994-11-05      SAN MIGUEL
V07    JOSE PALACIOS                           2500.00              1997-12-02      JESUS MARIA
V08    RUBEN GOMEZ                             1450.00              1996-12-08      LA MOLINA
V09    PATRICIA ARCE                           1800.00              1997-12-05      LA MOLINA
V10    RENATO PEREZ                            1550.00              1998-11-08      SURCO

(10 row(s) affected)
```

```
Primera forma:
Usando alias en las tablas.
SELECT V.COD_VEN AS CODIGO,V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       V.SUE_VEN AS SUELDO,V.FIN_VEN AS [FECHA DE INICIO],
       D.NOM_DIS AS DISTRITO
       FROM VENDEDOR V
       INNER JOIN DISTRITO D ON V.COD_DIS=D.COD_DIS
GO
```




```
FROM VENDEDOR V
INNER JOIN DISTRITO D ON V.COD_DIS = D.COD_DIS
```

Hay que tener en cuenta que la columna en común es COD_DIS, y es justamente esta la que determina la combinación.

Segunda forma:
Si se invierte la referencia de las tablas, el resultado es el mismo.

```
SELECT V.COD_VEN AS CODIGO,V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       V.SUE_VEN AS SUELDO,V.FIN_VEN AS [FECHA DE INICIO],
       D.NOM_DIS AS DISTRITO
       FROM DISTRITO D
       INNER JOIN VENDEDOR V ON V.COD_DIS=D.COD_DIS
GO
```

Tercera forma:
Sin usar alias en las tablas.

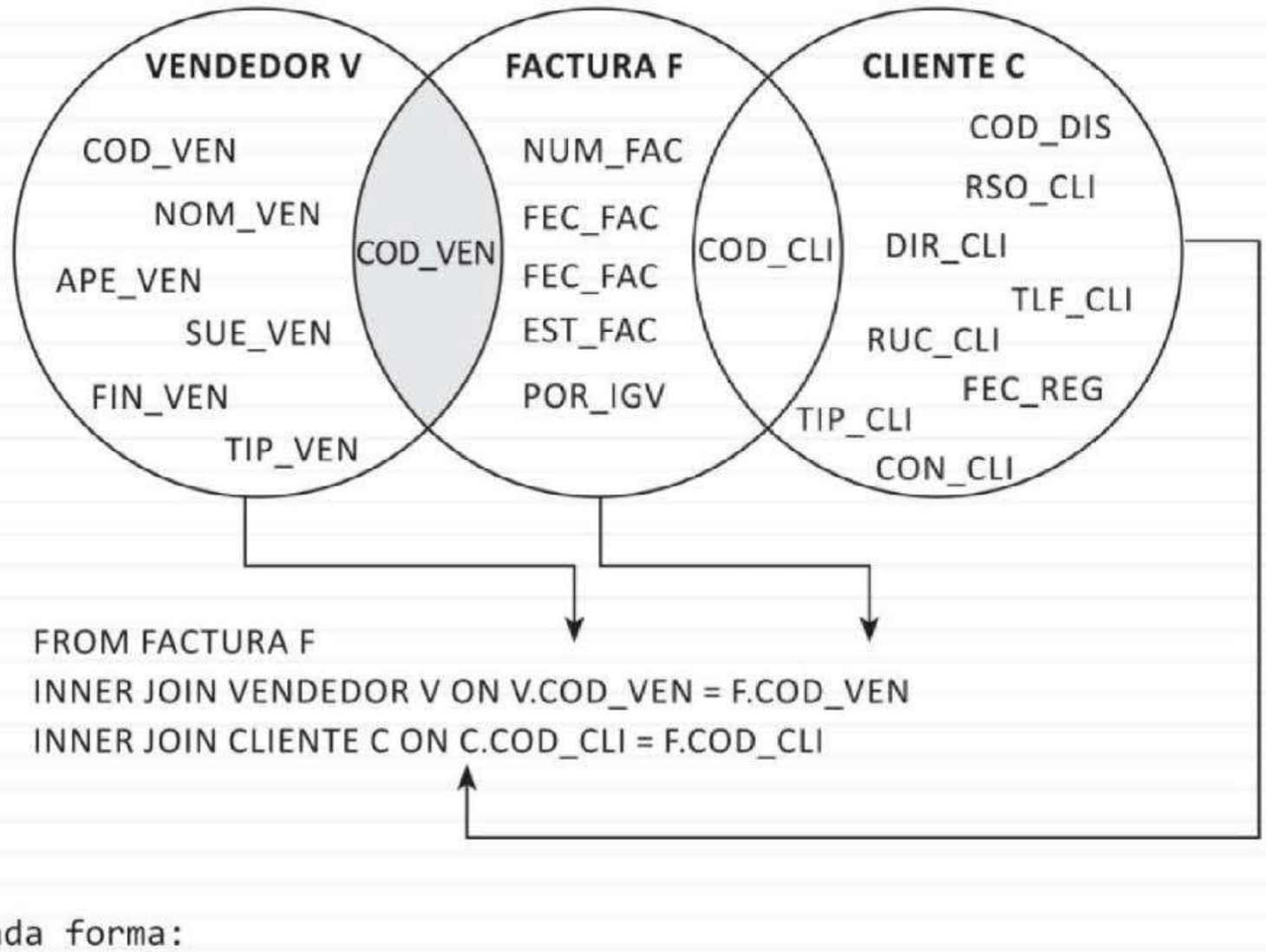
```
SELECT COD_VEN AS CODIGO,NOM_VEN+SPACE(1)+APE_VEN AS VENDEDOR,
       SUE_VEN AS SUELDO,FIN_VEN AS [FECHA DE INICIO],
       NOM_DIS AS DISTRITO
       FROM VENDEDOR
       INNER JOIN DISTRITO ON VENDEDOR.COD_DIS=DISTRITO.COD_DIS
GO
```

- Usando las tablas FACTURA, VENDEDOR y CLIENTE, realizar el siguiente listado:

```
FACTURA     FECHA FACTURADA VENDEDOR                                CLIENTE
----------- --------------- --------------------------------------- ----------
FA001       2013-06-07      JUANA ALVA                              FINSETH
FA002       2013-06-07      JUAN SOTO                               COREFO
FA003       2013-01-09      CESAR OJEDA                             SERVIEMSA
FA004       2013-06-09      JOSE PALACIOS                           CARDELI
FA005       2013-01-10      RUBEN GOMEZ                             MEBA
FA006       2013-01-08      JULIO VEGA                              PROMINENT
FA007       2013-05-10      PATRICIA ARCE                           COREFO
FA008       2013-04-10      RENATO PEREZ                            SUCERTE
FA009       2013-03-10      PATRICIA ARCE                           INTECH
FA010       2013-01-10      JUAN SOTO                               PAYET
FA011       2013-10-09      JULIO VEGA                              COREFO
FA012       2013-01-12      CESAR OJEDA                             KADIA
FA013       2013-01-12      RUBEN GOMEZ                             FILASUR
FA014       2012-01-12      PATRICIA ARCE                           CRAMER
FA015       2012-08-12      JOSE PALACIOS                           MEBA
FA016       2013-06-01      JULIO VEGA                              CARDELI
FA017       2013-06-01      ANA ORTEGA                              MEBA
FA018       2013-03-02      RENATO PEREZ                            CARDELI
FA019       2013-07-02      CARLOS AREVALO                          INTECH
FA020       2013-06-02      JUAN SOTO                               HAYASHI

(20 row(s) affected)
```

```
Primera forma:
SELECT F.NUM_FAC AS FACTURA,F.FEC_FAC AS [FECHA FACTURADA],
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       C.RSO_CLI AS CLIENTE
       FROM FACTURA F
       INNER JOIN VENDEDOR V ON V.COD_VEN = F.COD_VEN
       INNER JOIN CLIENTE C ON C.COD_CLI = F.COD_CLI
GO
```




```
Segunda forma:
Se usa la forma simplificada de Inner Join eliminando el término Inner.
SELECT F.NUM_FAC AS FACTURA,F.FEC_FAC AS [FECHA FACTURADA],
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       C.RSO_CLI AS CLIENTE
       FROM FACTURA F
       JOIN VENDEDOR V ON V.COD_VEN = F.COD_VEN
       JOIN CLIENTE C ON C.COD_CLI = F.COD_CLI
GO
```

4.12 Consultas externas

Se le llama así a una combinación en la cual se toma importancia a los valores que son equivalentes. Las combinaciones externas se especifican en la cláusula FROM con uno de los siguientes conjuntos de palabras clave:

- Left Join o Left Outer Join
- Right Join o Right Outer Join

Formato:

```
SELECT <ALL> <*>
<ALIAS.CAMPO AS CAMPO>
FROM <TABLA> <AS ALIAS>
<LEFT | RIGHT JOIN >
<ORDER BY CAMPO ASC | DESC>
```

4.12.1 Left Join

Muestra todas las filas de la tabla ubicada a la izquierda de la consulta, forzando a la otra tabla a mostrar valores tipo NULL para realizar una equivalencia entre las tablas.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
 FROM <TABLA> <AS ALIAS>
   <LEFT JOIN >
   <ORDER BY CAMPO ASC | DESC>
```

La aplicación de Left Join en la combinación de tablas es:

Usando las tablas VENDEDOR y DISTRITO, realizar el siguiente listado:

COD_DIS	NOM_DIS	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FIN_VEN	TIP_VEN	COD_DIS
D01	SURCO	V04	CESAR	OJEDA	850.00	1999-08-11	1	D01
D01	SURCO	V05	JULIO	VEGA	1500.00	1997-11-05	1	D01
D01	SURCO	V10	RENATO	PEREZ	1550.00	1998-11-06	2	D01
D02	JESUS MARIA	V07	JOSE	PALACIOS	2500.00	1997-12-02	1	D02
D03	SAN ISIDRO	V02	JUAN	SOTO	1200.00	2002-10-03	2	D03
D04	LA MOLINA	V08	RUBEN	GOMEZ	1450.00	1996-12-08	2	D04
D04	LA MOLINA	V09	PATRICIA	ARCE	1800.00	1997-12-05	2	D04
D05	SAN MIGUEL	V06	ANA	ORTEGA	1200.00	1994-11-05	1	D05
D06	MIRAFLORES	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D07	BARRANCO	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D08	CHORRILLOS	V01	JUANA	ALVA	1000.00	2003-07-12	1	D08
D09	SAN BORJA	V03	CARLOS	AREVALO	1500.00	1992-02-10	2	D09
D10	LINCE	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D11	BREÑA	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D12	MAGDALENA	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D13	RIMAC	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D14	SURQUILLO	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D15	PUEBLO LIBRE	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D16	BELLAVISTA	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D17	CALLAO	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D18	SAN MARTIN DE PORRES	NULL	NULL	NULL	NULL	NULL	NULL	NULL
D19	SANTA ANITA	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Las dos primeras columnas pertenecen a la tabla DISTRITO; por encontrarse en el lado izquierdo, muestran toda su información, así no tenga vendedores registrados. La tabla VENDEDOR se muestra a partir de la tercera columna, presentando el valor NULL solo para aquellos registros que no coinciden con la tabla Distrito. Por ejemplo, el distrito de código D06 no tiene clientes registrados, por eso se muestra como nulos.

```
SELECT *
        FROM DISTRITO D
        LEFT JOIN VENDEDOR V ON D.COD_DIS=V.COD_DIS
GO

Mostrar los distritos que no registran vendedor:
SELECT D.*
        FROM DISTRITO D
        LEFT JOIN VENDEDOR V ON D.COD_DIS=V.COD_DIS
        WHERE V.COD_DIS IS NULL
GO

Cuando se muestra la unión entre las tablas DISTRITO y VENDEDOR, se nota
que en la columna del vendedor se muestran valores nulos; por lo tanto,
para mostrar los distritos que tienen nulos en el vendedor se condiciona
a cualquier columna de la tabla vendedor; así,el resultado también podría
ser de la siguiente manera:

SELECT D.*
        FROM DISTRITO D
        LEFT JOIN VENDEDOR V ON D.COD_DIS=V.COD_DIS
        WHERE V.SUE_VEN IS NULL
GO
```

4.12.2 Right Join

Muestra todas las filas de la tabla ubicada en la derecha de la consulta, forzando a la otra tabla a mostrar valores tipo NULL para realizar una equivalencia entre las tablas.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
 FROM <TABLA> <AS ALIAS>
   <RIGHT JOIN >
   <ORDER BY CAMPO ASC | DESC>
```

La aplicación de Right Join en la combinación de tablas es:

Usando las tablas VENDEDOR y DISTRITO, realizar el siguiente listado:

COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FIN_VEN	TIP_VEN	COD_DIS	COD_DIS	NOM_DIS
V04	CESAR	OJEDA	850.00	1999-08-11	1	D01	D01	SURCO
V05	JULIO	VEGA	1500.00	1997-11-05	1	D01	D01	SURCO
V10	RENATO	PEREZ	1550.00	1998-11-08	2	D01	D01	SURCO
V07	JOSE	PALACIOS	2500.00	1997-12-02	1	D02	D02	JESUS MARIA
V02	JUAN	SOTO	1200.00	2002-10-03	2	D03	D03	SAN ISIDRO
V08	RUBEN	GOMEZ	1450.00	1996-12-08	2	D04	D04	LA MOLINA
V09	PATRICIA	ARCE	1800.00	1997-12-05	2	D04	D04	LA MOLINA
V06	ANA	ORTEGA	1200.00	1994-11-05	1	D05	D05	SAN MIGUEL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D06	MIRAFLORES
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D07	BARRANCO
V01	JUANA	ALVA	1000.00	2003-07-12	1	D08	D08	CHORRILLOS
V03	CARLOS	AREVALO	1500.00	1992-02-10	2	D09	D09	SAN BORJA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D10	LINCE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D11	BREÑA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D12	MAGDALENA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D13	RIMAC
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D14	SURQUILLO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D15	PUEBLO LIBRE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D16	BELLAVISTA

Las siete primeras columnas pertenecen a la tabla CLIENTE; por encontrarse en el lado izquierdo, muestran valores tipo NULL, ya que la tabla DISTRITO que se encuentra a la derecha muestra todos los distritos, forzando a la tabla de la izquierda a mostrar todos sus registros y rellenar con nulos.

```
SELECT *
       FROM VENDEDOR V
       RIGHT JOIN DISTRITO D ON D.COD_DIS=V.COD_DIS
GO

Mostrar los distritos que no registran vendedor:
SELECT D.*
       FROM VENDEDOR V
       RIGHT JOIN DISTRITO D ON D.COD_DIS=V.COD_DIS
       WHERE V.COD_DIS IS NULL
GO
```

4.12.3 Full Join

Combina los valores de la primera tabla con los valores de la segunda tabla, de forma que siempre devolverá filas de ambas tablas, aunque no cumplan la condición de correspondencia.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
 FROM <TABLA> <AS ALIAS>
   <FULL JOIN >
   <ORDER BY CAMPO ASC | DESC>
```

La aplicación de Full Join en la combinación de tablas es:

Mostrar el siguiente listado producto de la unión entre las tablas VENDEDOR y DISTRITO:

COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FIN_VEN	TIP_VEN	COD_DIS	COD_DIS	NOM_DIS
V01	JUANA	ALVA	1000.00	2003-07-12	1	D08	D08	CHORRILLOS
V02	JUAN	SOTO	1200.00	2002-10-03	2	D03	D03	SAN ISIDRO
V03	CARLOS	AREVALO	1500.00	1992-02-10	2	D09	D09	SAN BORJA
V04	CESAR	OJEDA	850.00	1999-08-11	1	D01	D01	SURCO
V05	JULIO	VEGA	1500.00	1997-11-05	1	D01	D01	SURCO
V06	ANA	ORTEGA	1200.00	1994-11-05	1	D05	D05	SAN MIGUEL
V07	JOSE	PALACIOS	2500.00	1997-12-02	1	D02	D02	JESUS MARIA
V08	RUBEN	GOMEZ	1450.00	1996-12-08	2	D04	D04	LA MOLINA
V09	PATRICIA	ARCE	1800.00	1997-12-05	2	D04	D04	LA MOLINA
V10	RENATO	PEREZ	1550.00	1998-11-08	2	D01	D01	SURCO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D06	MIRAFLORES
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D07	BARRANCO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D10	LINCE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D11	BREÑA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D12	MAGDALENA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D13	RIMAC
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D14	SURQUILLO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D15	PUEBLO LIBRE
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D16	BELLAVISTA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D17	CALLAO
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D18	SAN MARTIN DE PORRES
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D19	SANTA ANITA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D20	LOS OLIVOS
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D21	INDEPENDENCIA
NULL	NULL	NULL	NULL	NULL	NULL	NULL	D22	LIMA - CERCADO

Este tipo de unión mostrará las filas de VENDEDOR y DISTRITO, así no tengan asociación:

```
SELECT *
        FROM VENDEDOR V
        FULL JOIN DISTRITO D ON D.COD_DIS=V.COD_DIS
GO
```

Mostrar los distritos que no registran vendedor:

```
SELECT D.*
        FROM VENDEDOR V
        FULL JOIN DISTRITO D ON V.COD_DIS=D.COD_DIS
        WHERE V.APE_VEN IS NULL
GO
```

4.13 Consultas agrupadas

Este tipo de consultas permite mostrar información en forma de grupos con uno o más propósitos. Hay que tener en cuenta que las consultas agrupadas aumentan su productividad cuando se usan con funciones agregadas.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
  FROM <TABLA> <AS ALIAS>
   <INNER | LEFT | RIGHT | CROSS JOIN >
   <WHERE CONDICION>
   <GROUP BY CAMPO>
   <HAVING CONDICION>
   <ORDER BY CAMPO ASC | DESC>
```

4.13.1 Funciones agregadas

Son funciones que especifican un determinado valor sobre las columnas de una tabla; solo contamos con cinco funciones agregadas en SQL, los cuales son suma, máximo, mínimo, promedio y conteo.

A. **Count:** Permite devolver el total de registros o filas encontrados en una tabla dependiendo de una determinada condición.

Formato:

```
SELECT COUNT(CAMPO)
   FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Count:

```
Contar el total de registros de productos.

Primera forma:
Especificando una columna de la tabla.
SELECT COUNT(COD_PRO) AS [TOTAL DE PRODUCTOS]
       FROM PRODUCTO
GO

Segunda forma:
Usando el operador asterisco; este representa a cualquier columna de la
tabla.
SELECT COUNT(*) AS [TOTAL DE PRODUCTOS]
       FROM PRODUCTO
GO

Contar el total de registros de clientes cuyo registro sea del año 2012.
SELECT COUNT(*)  AS [TOTAL DE CLIENTES EN EL 2012]
       FROM CLIENTE C
       WHERE YEAR(C.FEC_REG)=2012
GO
```

B. **Sum:** Permite devolver el acumulado de una columna de tipo numérico bajo un determinado criterio.

Formato:

```
SELECT SUM(CAMPO)
   FROM <TABLA>
 WHERE <CONDICION>
```

4.13.1 Funciones agregadas

Son funciones que especifican un determinado valor sobre las columnas de una tabla; solo contamos con cinco funciones agregadas en SQL, los cuales son suma, máximo, mínimo, promedio y conteo.

A. **Count:** Permite devolver el total de registros o filas encontrados en una tabla dependiendo de una determinada condición.

Formato:

```
SELECT COUNT(CAMPO)
   FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Count:

```
Contar el total de registros de productos.

Primera forma:
Especificando una columna de la tabla.
SELECT COUNT(COD_PRO) AS [TOTAL DE PRODUCTOS]
       FROM PRODUCTO
GO

Segunda forma:
Usando el operador asterisco; este representa a cualquier columna de la
tabla.
SELECT COUNT(*) AS [TOTAL DE PRODUCTOS]
       FROM PRODUCTO
GO

Contar el total de registros de clientes cuyo registro sea del año 2012.
SELECT COUNT(*)  AS [TOTAL DE CLIENTES EN EL 2012]
       FROM CLIENTE C
       WHERE YEAR(C.FEC_REG)=2012
GO
```

B. **Sum:** Permite devolver el acumulado de una columna de tipo numérico bajo un determinado criterio.

Formato:

```
SELECT SUM(CAMPO)
   FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Sum:

```
Mostrar el total acumulado de los precios de productos.
SELECT SUM(P.PRE_PRO) AS [PRECIO ACUMULADO]
       FROM PRODUCTO P
GO

Mostrar el monto acumulado del stock actual solo de aquellos productos
importados.
SELECT SUM(P.SAC_PRO) AS [STOCK ACUMULADO]
       FROM PRODUCTO P
       WHERE IMP_PRO='V'
GO

Mostrar el monto acumulado del producto entre el precio y la cantidad
registrada en el detalle de la factura.
SELECT SUM(D.CAN_VEN*D.PRE_VEN) AS [SUBTOTAL ACUMULADO]
       FROM DETALLE_FACTURA D
GO
```

C. Max: Permite devolver el máximo valor de un campo numérico encontrado en una tabla dependiendo de una determinada condición.

Formato:

```
SELECT MAX(CAMPO)
  FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Max:

```
Mostrar el producto más caro que registra la tienda.
SELECT MAX(P.PRE_PRO) AS [PRECIO MAXIMO]
       FROM PRODUCTO P
GO

Mostrar el producto más caro que registra la tienda con respecto a los
productos importados.
SELECT MAX(P.PRE_PRO) AS [PRECIO MAXIMO]
       FROM PRODUCTO P
       WHERE IMP_PRO='V'
GO
```

D. Min: Permite devolver el mínimo valor de un campo numérico encontrado en una tabla dependiendo de una determinada condición.

Formato:

```
SELECT MIN(CAMPO)
  FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Min:

```
Mostrar el producto con el precio más bajo que registra la tienda.
SELECT MIN(P.PRE_PRO) AS [PRECIO MINIMO]
       FROM PRODUCTO P
GO

Mostrar el producto con el precio más bajo que registra la tienda con
respecto a los productos no importados.
SELECT MIN(P.PRE_PRO) AS [PRECIO MINIMO]
       FROM PRODUCTO P
       WHERE IMP_PRO='F'
GO
```

E. Avg: Permite devolver el promedio de una columna de tipo numérico bajo un determinado criterio.

Formato:

```
SELECT AVG(CAMPO)
   FROM <TABLA>
 WHERE <CONDICION>
```

A continuación, algunos casos del uso de Avg:

```
Mostrar el promedio de precios de los productos.
SELECT AVG(P.PRE_PRO) AS [PRECIO PROMEDIO]
       FROM PRODUCTO P
GO

Mostrar el promedio de stock actual solo de aquellos productos importados.
SELECT AVG(P.SAC_PRO) AS [STOCK PROMEDIO]
       FROM PRODUCTO P
       WHERE IMP_PRO='V'
GO
```

4.13.2 Cláusula Group by

Para recuperar información agrupada por algún criterio, se debe implementar la cláusula GROUP BY dentro de la consulta SELECT. El GROUP BY agrupa un conjunto de registros de acuerdo a los valores de una o más columnas de una tabla. Para una mejor implementación de las consultas agrupadas se puede usar las funciones agregadas como Sum, Count, Max, Min y Avg.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
  FROM <TABLA> <AS ALIAS>
   <GROUP BY CAMPO>
   <HAVING CONDICION>
```

A continuación, algunos casos del uso de GROUP BY básicos:

```
Mostrar el total de clientes registrados por distrito.
```

```
DISTRITO                                           TOTAL
-------------------------------------------------- -----------
BARRANCO                                           1
BELLAVISTA                                         1
BREÑA                                              1
CALLAO                                             1
JESUS MARIA                                        1
LA MOLINA                                          1
LIMA - CERCADO                                     1
LINCE                                              1
LOS OLIVOS                                         1
MIRAFLORES                                         1
SAN BORJA                                          1
SAN JUAN DE LURIGANCHO                             1
SAN LUIS                                           1
SAN MIGUEL                                         5
SURCO                                              1
SURQUILLO                                          1

(16 row(s) affected)
```

```
SELECT D.NOM_DIS AS DISTRITO, COUNT(*) AS [TOTAL]
       FROM CLIENTE C
       JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
       GROUP BY D.NOM_DIS
GO

Mostrar el total de facturas registradas por mes.
```

```
MES         TOTAL
----------- ----------
1           7
3           2
4           1
5           1
6           6
7           1
8           1
10          1

(8 row(s) affected)
```

```
SELECT MONTH(F.FEC_FAC) AS MES, COUNT(*) AS TOTAL
       FROM FACTURA F
       GROUP BY MONTH(F.FEC_FAC)
GO

Mostrar el monto acumulado (precio x cantidad) de las facturas.
```

```
FACTURA     ACUMULADO
----------- -----------
FA001       782
FA002       64
FA003       721
FA004       2509
FA005       2925
FA006       769
FA007       416
FA008       2950
FA009       255
FA010       9
FA011       400
FA012       10
FA013       16941
FA014       100
FA015       34
FA016       229
FA017       28800
FA018       52
FA019       22640
FA020       14425

(20 row(s) affected)
```

```
SELECT F.NUM_FAC AS FACTURA, SUM(DF.CAN_VEN*DF.CAN_VEN) AS ACUMULADO
       FROM FACTURA F
       JOIN DETALLE_FACTURA DF ON F.NUM_FAC=DF.NUM_FAC
       GROUP BY F.NUM_FAC
GO
```

Mostrar el total de facturas registradas por vendedor.

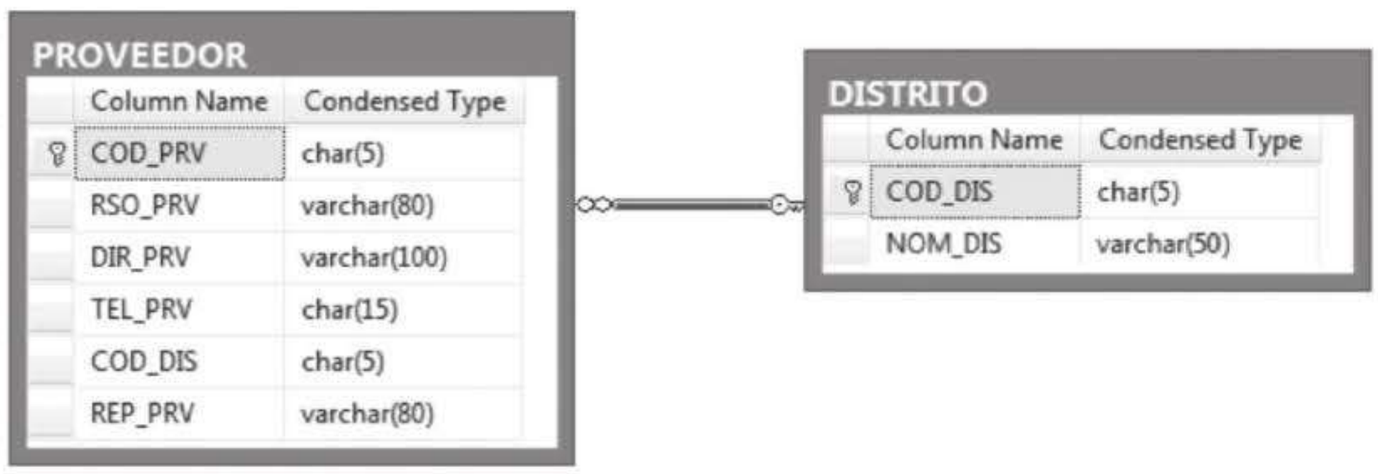
```
VENDEDOR                                TOTAL DE FACTURAS
--------------------------------------- -----------------
ANA ORTEGA                              1
CARLOS AREVALO                          1
CESAR OJEDA                             2
JOSE PALACIOS                           2
JUAN SOTO                               3
JUANA ALVA                              1
JULIO VEGA                              3
PATRICIA ARCE                           3
RENATO PEREZ                            2
RUBEN GOMEZ                             2

(10 row(s) affected)
```

```
SELECT V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       COUNT(*) AS [TOTAL DE FACTURAS]
       FROM FACTURA F
       JOIN VENDEDOR V ON V.COD_VEN=F.COD_VEN
       GROUP BY V.NOM_VEN+SPACE(1)+V.APE_VEN
GO
```

A continuación, algunos casos del uso de GROUP BY condicionados:

- Si se tiene la información de las tablas PROVEEDOR y DISTRITO:



Y se necesita mostrar el total de proveedores por distrito, pero solo para los distritos cuyo número de proveedores sea mayor o igual a dos.

```
DISTRITO                                                TOTAL
------------------------------------------------------- ---------
ATE - VITARTE                                           2
RIMAC                                                   3
SANTA ANITA                                             2

(3 row(s) affected)
```

```
SELECT D.NOM_DIS AS DISTRITO, COUNT(*) AS TOTAL
        FROM PROVEEDOR P
        JOIN DISTRITO D ON P.COD_DIS=D.COD_DIS
        GROUP BY D.NOM_DIS
        HAVING COUNT(*)>1
GO
```

• Si se tiene la información de la tabla FACTURA:

FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

Y se necesita mostrar información agrupada sobre los años y meses del registro de las facturas, condicionados a que sean solo del primer semestre.

```
AÑO         MES         TOTAL
----------- ----------- --------
2012        1           1
2013        1           6
2013        3           2
2013        4           1
2013        5           1
2013        6           6

(6 row(s) affected)
```

```
SELECT YEAR(F.FEC_FAC) AS AÑO, MONTH(F.FEC_FAC) AS MES,COUNT(*) AS TOTAL
        FROM FACTURA F
        GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
        HAVING MONTH(F.FEC_FAC) <=6
GO
```

4.13.3 Cláusula Group by con resúmenes

Los resúmenes solo se pueden aplicar a un grupo de información; esto permitirá visualizar montos o conteos realizados sobre un conjunto de información agrupada.

Formato:

```
SELECT <ALL> <*>
  <ALIAS.CAMPO AS CAMPO>
 FROM <TABLA> <AS ALIAS>
  <GROUP BY CAMPO>
   [ ROLLUP ]
   [ CUBE ] <CAMPOS>
  <HAVING CONDICION>
```

- ROLLUP: Genera filas de agregado en la cláusula GROUP BY más filas de subtotal y, también, una fila con un total general. El número de agrupaciones es igual al número de expresiones de la lista de elementos compuestos más uno; aquí justamente muestra los resultantes.
- CUBE: Genera filas de agregado en la cláusula GROUP BY más una fila de superagregado y filas de tabulación cruzada. El número de agrupaciones es igual a 2n, donde *n* es el número de expresiones de la lista de elementos compuestos.

A continuación, algunos casos del uso de GROUP BY con resúmenes:

- Si se cuenta con información de la tabla PRODUCTO:

PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

Se necesita mostrar el total de productos según el tipo de unidad, además del total de productos registrados, como se muestra en la siguiente imagen:

```
UNIDAD                          TOTAL
------------------------------- ---------
CIE                             5
DOC                             9
MLL                             10
UNI                             7
NULL                            31

(5 row(s) affected)
```

```
USANDO ROLLUP
SELECT UNI_PRO AS UNIDAD, COUNT(*) AS TOTAL
       FROM PRODUCTO
       GROUP BY UNI_PRO
       WITH ROLLUP
GO

USANDO CUBE
SELECT UNI_PRO AS UNIDAD, COUNT(*) AS TOTAL
       FROM PRODUCTO
       GROUP BY CUBE (UNI_PRO)
GO

Si se necesita modificar el valor NULL dentro de la consulta agrupada con
resúmenes, se podría usar el siguiente código:
SELECT CASE WHEN UNI_PRO IS NULL
                    THEN 'TOTAL PRODUCTOS: '
                    ELSE UNI_PRO
               END AS UNIDAD,
               COUNT(*) AS TOTAL
       FROM PRODUCTO
       GROUP BY CUBE (UNI_PRO)
GO

SELECT CASE WHEN UNI_PRO IS NULL
                    THEN 'TOTAL PRODUCTOS: '
                    ELSE UNI_PRO
               END AS UNIDAD,
               COUNT(*) AS TOTAL
       FROM PRODUCTO
       GROUP BY UNI_PRO
       WITH ROLLUP
GO
```

```
UNIDAD                          TOTAL
------------------------------ -----------
CIE                            5
DOC                            9
MLL                            10
UNI                            7
TOTAL PRODUCTOS:               31

(5 row(s) affected)
```

- Si se cuenta con información de la tabla FACTURA:

FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

Se necesita mostrar el total de facturas según el año en un determinado mes, además del total de facturas registradas, como se muestra en la siguiente imagen:

```
AÑO         MES         TOTAL
----------- ----------- -----------
2012        1           1
2012        8           1
2012        NULL        2
2013        1           6
2013        3           2
2013        4           1
2013        5           1
2013        6           6
2013        7           1
2013        10          1
2013        NULL        18
NULL        NULL        20

(12 row(s) affected)
```

```
SELECT YEAR(F.FEC_FAC) AS AÑO, MONTH(F.FEC_FAC) AS MES, COUNT(*) AS TOTAL
        FROM FACTURA F
        GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
        WITH ROLLUP
GO
```

Se puede modificar la impresión de NULL en la agrupación:

```
AÑO   MES            TOTAL
----- ------------- -----------
2012  1              1
2012  8              1
2012  TOTAL POR AÑO 2
2013  1              6
2013  3              2
2013  4              1
2013  5              1
2013  6              6
2013  7              1
2013  10             1
2013  TOTAL POR AÑO 18
TOTAL TOTAL POR AÑO 20

(12 row(s) affected)
```

```
SELECT CASE WHEN YEAR(F.FEC_FAC) IS NULL
                        THEN 'TOTAL'
                        ELSE CAST(YEAR(F.FEC_FAC) AS CHAR(4))
                END AS AÑO,
                CASE WHEN MONTH(F.FEC_FAC) IS NULL
                        THEN  'TOTAL POR AÑO'
                        ELSE CAST(MONTH(F.FEC_FAC) AS CHAR(2))
                END AS MES, COUNT(*) AS TOTAL
        FROM FACTURA F
        GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
        WITH ROLLUP
GO
```

Se necesita mostrar el total de facturas según el año en un determinado mes, además del total de facturas registradas, como se muestra en la siguiente imagen:

```
AÑO         MES         TOTAL
----------- ----------- -----------
2012        1           1
2012        8           1
2012        NULL        2
2013        1           6
2013        3           2
2013        4           1
2013        5           1
2013        6           6
2013        7           1
2013        10          1
2013        NULL        18
NULL        NULL        20

(12 row(s) affected)
```

```
SELECT YEAR(F.FEC_FAC) AS AÑO, MONTH(F.FEC_FAC) AS MES, COUNT(*) AS TOTAL
        FROM FACTURA F
        GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
        WITH ROLLUP
GO
```

Se puede modificar la impresión de NULL en la agrupación:

```
AÑO   MES            TOTAL
----- ------------- -----------
2012  1              1
2012  8              1
2012  TOTAL POR AÑO 2
2013  1              6
2013  3              2
2013  4              1
2013  5              1
2013  6              6
2013  7              1
2013  10             1
2013  TOTAL POR AÑO 18
TOTAL TOTAL POR AÑO 20

(12 row(s) affected)
```

```
SELECT CASE WHEN YEAR(F.FEC_FAC) IS NULL
                        THEN 'TOTAL'
                        ELSE CAST(YEAR(F.FEC_FAC) AS CHAR(4))
                END AS AÑO,
                CASE WHEN MONTH(F.FEC_FAC) IS NULL
                        THEN  'TOTAL POR AÑO'
                        ELSE CAST(MONTH(F.FEC_FAC) AS CHAR(2))
                END AS MES, COUNT(*) AS TOTAL
        FROM FACTURA F
        GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
        WITH ROLLUP
GO
```

4.14 SUBCONSULTAS

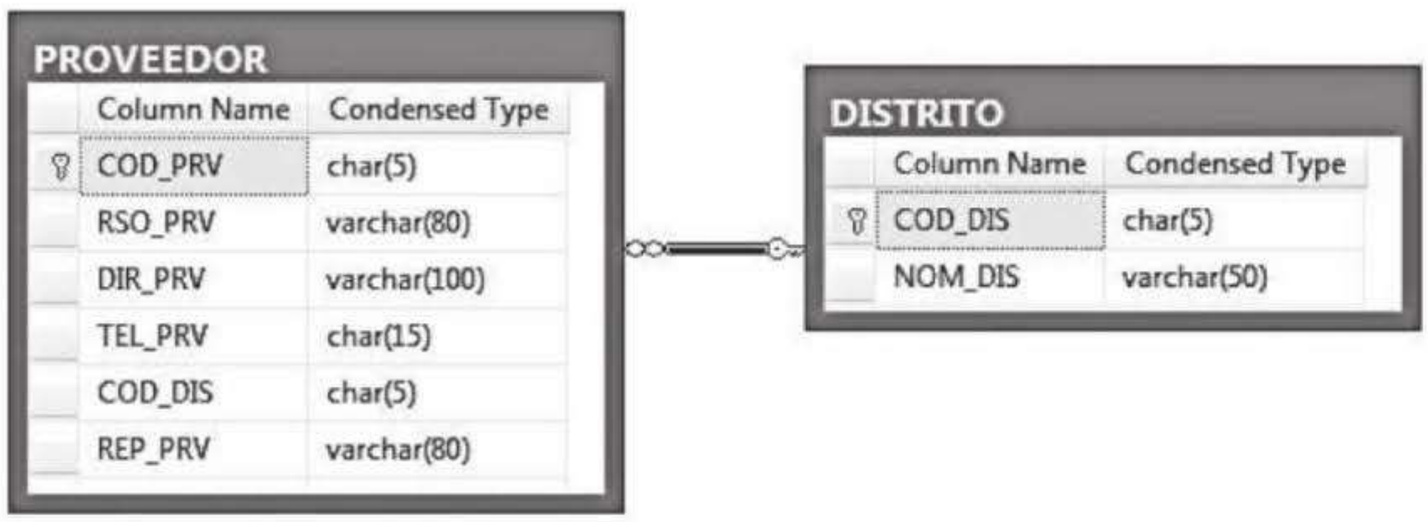
Es una sentencia SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT... INTO, DELETE, o UPDATE, o dentro de otra subconsulta.

Formato:

```
SELECT <ALL> <*> <SUBCONSULTA>
    FROM <TABLA> <AS ALIAS>
    <WHERE CONDICION - SUBCONSULTA>
```

A continuación, algunos casos del uso de subconsultas:

Si se tiene las tablas PROVEEDOR y DISTRITO:




```
Implementar una consulta que permita mostrar los datos del proveedor,
adicionando el nombre del distrito.

SELECT P.COD_PRV AS CODIGO,
       P.RSO_PRV AS PROVEEDOR,
       P.DIR_PRV AS DIRECCION,
       P.TEL_PRV AS TELEFONO,
       (SELECT D.NOM_DIS FROM DISTRITO D WHERE D.COD_DIS=P.COD_DIS) AS
DISTRITO,
       P.REP_PRV AS REPRESENTANTE
       FROM PROVEEDOR P
GO
```

CODIGO	PROVEEDOR	DIRECCION	TELEFONO	DISTRITO	REPRESENTANTE
PR01	FABER CASTELL	AV. ISABEL LA CATOLICA 1875	0000000	RIMAC	CARLOS AGUIRRE
PR02	ATLAS	AV. LIMA 471	0000000	RIMAC	CESAR TORRES
PR03	3M	AV. VENEZUELA 3018	2908165	BELLAVISTA	OMAR INJOQUE
PR04	DITO	AV. METROPOLITANA 376	NULL	SANTA ANITA	RAMON FLORES
PR05	ACKER	CALLE LAS DUNAS 245	4780143	ATE - VITARTE	JULIO ACUÑA
PR06	DEDITEC	CALLE PICHINCHA 644	5662848	BREÑA	JAVIER GONZALES
PR07	OFFICETEC	CALLE LAS PERDICES 225 OF. 204	4216184	SAN ISIDRO	CARLOS ROBLES
PR08	INVICTA	AV. LOS FRUTALES 564	4364247	ATE - VITARTE	ALBERTO RODRIGUEZ
PR09	DIPROPOR	AV. DEL AIRE 901	4742046	SAN LUIS	ROBERTO RONCEROS
PR10	MIURA	AV. LA PAZ 257	4459710	MIRAFLORES	JORGE VASQUEZ
PR11	PRAXIS	AV. JOSE GALVEZ 1820 - 1844	4703944	LINCE	ERICKA ZEGARRA
PR12	SUMIGRAF	AV. MANCO CAPAC 754	0000000	RIMAC	KARINNA PAREDES
PR13	LIMMSA	PROLG. HUAYLAS 670	2546995	CHORRILLOS	LAURA ORTEGA

```
Implementar una consulta que permita mostrar los proveedores de un
determinado distrito.

SELECT P.*
       FROM PROVEEDOR P
       WHERE P.COD_DIS=(SELECT D.COD_DIS FROM DISTRITO D WHERE D.NOM_
DIS='RIMAC')
GO
```

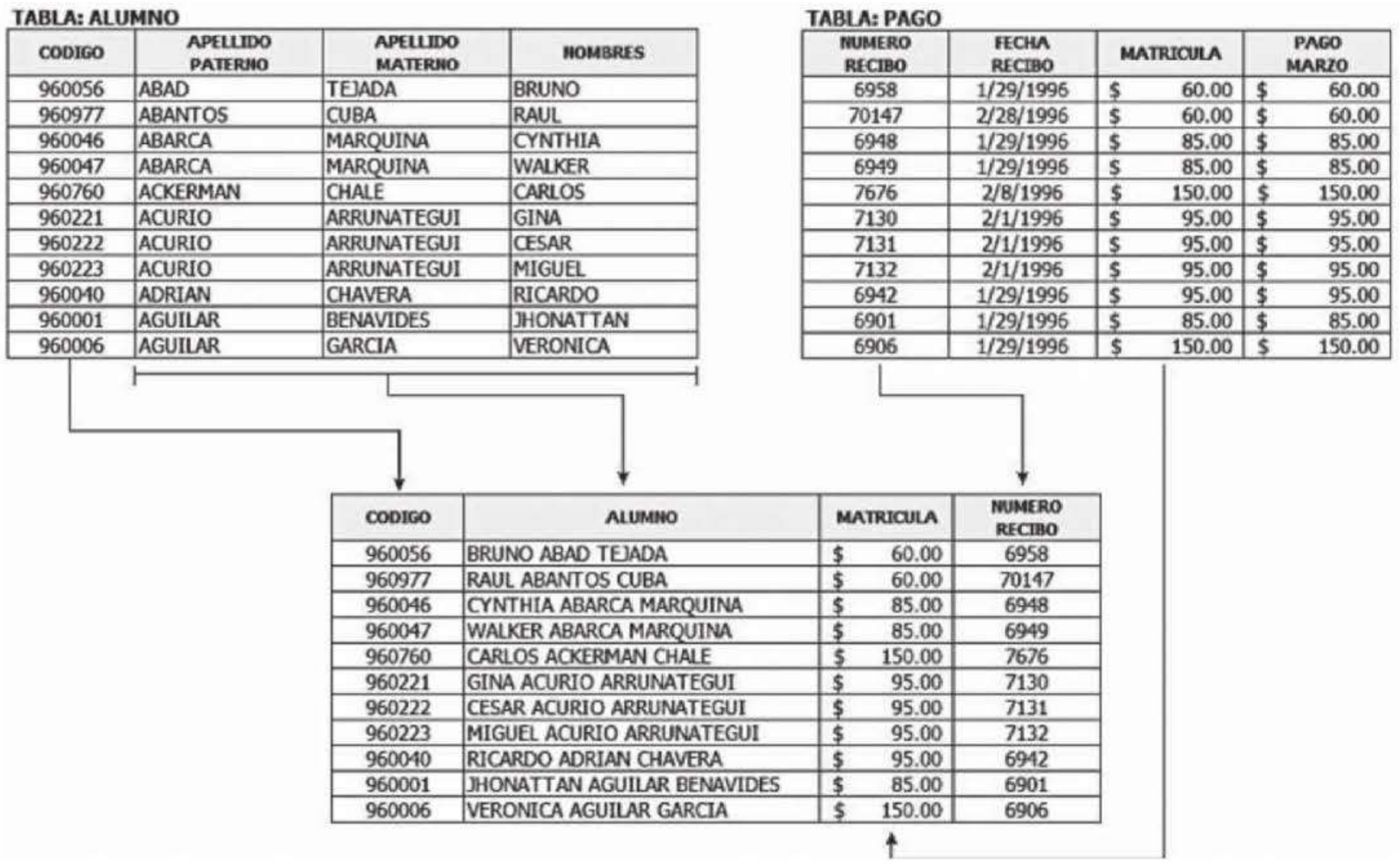
```
COD_PRV RSO_PRV         DIR_PRV                          TEL_PRV    COD_DIS REP_PRV
------- ------------------------------------------------------------ ----------
PR01    FABER CASTELL   AV. ISABEL LA CATOLICA 1875     0000000    D13     CARLOS AGUIRRE
PR02    ATLAS           AV. LIMA 471                     0000000    D13     CESAR TORRES
PR12    SUMIGRAF        AV. MANCO CAPAC 754              0000000    D13     KARINNA PAREDES


(3 row(s) affected)
```

4.15 Vistas

Es un mecanismo que permite almacenar, de forma permanente, una consulta en SQL Server; asimismo, genera un resultado a partir de una consulta almacenada en la vista, para finalmente ejecutar nuevos resultados como si fueran una nueva tabla.

Es considerado una tabla virtual, cuyo contenido se compone de columnas y filas provenientes de una consulta; esta consulta puede provenir de una o más tablas dentro de la misma base de datos.



A partir de las tablas ALUMNO y PAGO, se puede implementar una vista a partir de la información de ambas tablas. Asimismo, se debe considerar que una vista tiene como estructura la misma que proporciona sus fuentes; es decir, el nombre de la columna y tipo de datos de la vista serán idénticas a las de su tabla original.

4.15.1 Ventajas de las vistas

Algunas ventajas en el uso de las vistas son, por ejemplo, las siguientes:

- Permite centrar, simplificar y personalizar la forma de mostrar la información a cada usuario.
- Se usa como mecanismo de seguridad, el cual permite a los usuarios obtener acceso a la información proveniente de las tablas por medio de la vista.
- También proporciona una interfaz compatible con versiones anteriores al SQL para emular una tabla cuyo esquema ha cambiado debido a las nuevas versiones.

Algunas características importantes de las vistas son las siguientes:

- Las vistas solo pueden crearse en la base de datos activa.
- La instrucción CREATE VIEW permite crear una vista, y esta debe ser la primera instrucción dentro de un bloque de sentencias.
- Como máximo número de columnas permitidas en la vista se tiene 1024.

Cuando se realiza una consulta a través de una vista, el Motor de base de datos se asegura de que todos los objetos de base de datos a los que se hace referencia en algún lugar de la instrucción existen, que son válidos en el contexto de la instrucción y que las instrucciones de modificación de datos no infringen ninguna regla de integridad de los datos. Las comprobaciones que no son correctas devuelven un mensaje de error. Las comprobaciones correctas traducen la acción a una acción con las tablas subyacentes.

Cuando se crea una vista, la información sobre ella se almacena en las vistas de catálogo **sys.views**, **sys.columns** y **sys.sql_expression_dependencies**. El texto de la instrucción CREATE VIEW se almacena en la vista de catálogo **sys.sql_modules**.

4.15.2 Creación de las vistas

```
CREATE VIEW NOMBREVISTA
AS
  SENTENCIAS
GO
```

El nombre de la vista debe tener el mismo criterio con el que se le asigna un nombre a la tabla de una base de datos.

A continuación, se muestra cómo implementar una vista que permita mostrar información de las tablas VENDEDOR y DISTRITO.



```
--VALIDACIÓN DE LA VISTA
IF OBJECT_ID('V_LISTAVENDEDOR') IS NOT NULL
        DROP VIEW V_LISTAVENDEDOR
GO

--CREANDO LA VISTA
CREATE VIEW V_LISTAVENDEDOR
AS
        SELECT V.COD_VEN AS CODIGO,
                V.NOM_VEN+SPACE(1)+V.APE_VEN AS NOMBRE,
                V.SUE_VEN AS SUELDO,
                V.FIN_VEN AS [FECHA INICIO],
                D.NOM_DIS AS DISTRITO
                FROM VENDEDOR V
                JOIN DISTRITO D ON V.COD_DIS=D.COD_DIS
GO

--PRUEBA:
SELECT * FROM V_LISTAVENDEDOR
GO
```

Se inicia la implementación validando la existencia de la vista en la base de datos, usando la cláusula OBJECT_ID dentro de una sentencia selectiva. Luego, se crea la vista usando el mecanismo de unión Inner Join y especificando qué campos necesita nuestra vista; hay que tener en cuenta que todos los campos deben tener un nombre de cabecera, ya que este será parte de la vista como si fueran sus columnas que la componen. Finalmente, la prueba de la vista se realiza mediante una consulta SELECT.

A partir de una vista, se pueden realizar muchas consultas de los datos contenidos en ella, como si fuera una tabla. Algunos ejemplos de consultas sobre la vista son:

```
--Listar los vendedores del distrito "La Molina"
```

```
CODIGO NOMBRE              SUELDO      FECHA INICIO DISTRITO
------ ----------------------------------- ------------
V08    RUBEN GOMEZ         1450.00     1996-12-08   LA MOLINA
V09    PATRICIA ARCE       1800.00     1997-12-05   LA MOLINA

(2 row(s) affected)
```

```
SELECT V.*
        FROM V_LISTAVENDEDOR V
        WHERE V.DISTRITO='LA MOLINA'
GO
```

```
--VALIDACIÓN DE LA VISTA
IF OBJECT_ID('V_LISTAVENDEDOR') IS NOT NULL
        DROP VIEW V_LISTAVENDEDOR
GO

--CREANDO LA VISTA
CREATE VIEW V_LISTAVENDEDOR
AS
        SELECT V.COD_VEN AS CODIGO,
                V.NOM_VEN+SPACE(1)+V.APE_VEN AS NOMBRE,
                V.SUE_VEN AS SUELDO,
                V.FIN_VEN AS [FECHA INICIO],
                D.NOM_DIS AS DISTRITO
                FROM VENDEDOR V
                JOIN DISTRITO D ON V.COD_DIS=D.COD_DIS
GO

--PRUEBA:
SELECT * FROM V_LISTAVENDEDOR
GO
```

Se inicia la implementación validando la existencia de la vista en la base de datos, usando la cláusula OBJECT_ID dentro de una sentencia selectiva. Luego, se crea la vista usando el mecanismo de unión Inner Join y especificando qué campos necesita nuestra vista; hay que tener en cuenta que todos los campos deben tener un nombre de cabecera, ya que este será parte de la vista como si fueran sus columnas que la componen. Finalmente, la prueba de la vista se realiza mediante una consulta SELECT.

A partir de una vista, se pueden realizar muchas consultas de los datos contenidos en ella, como si fuera una tabla. Algunos ejemplos de consultas sobre la vista son:

```
--Listar los vendedores del distrito "La Molina"
```

```
CODIGO NOMBRE              SUELDO      FECHA INICIO DISTRITO
------ ----------------------------------- ------------
V08    RUBEN GOMEZ         1450.00     1996-12-08   LA MOLINA
V09    PATRICIA ARCE       1800.00     1997-12-05   LA MOLINA

(2 row(s) affected)
```

```
SELECT V.*
        FROM V_LISTAVENDEDOR V
        WHERE V.DISTRITO='LA MOLINA'
GO
```

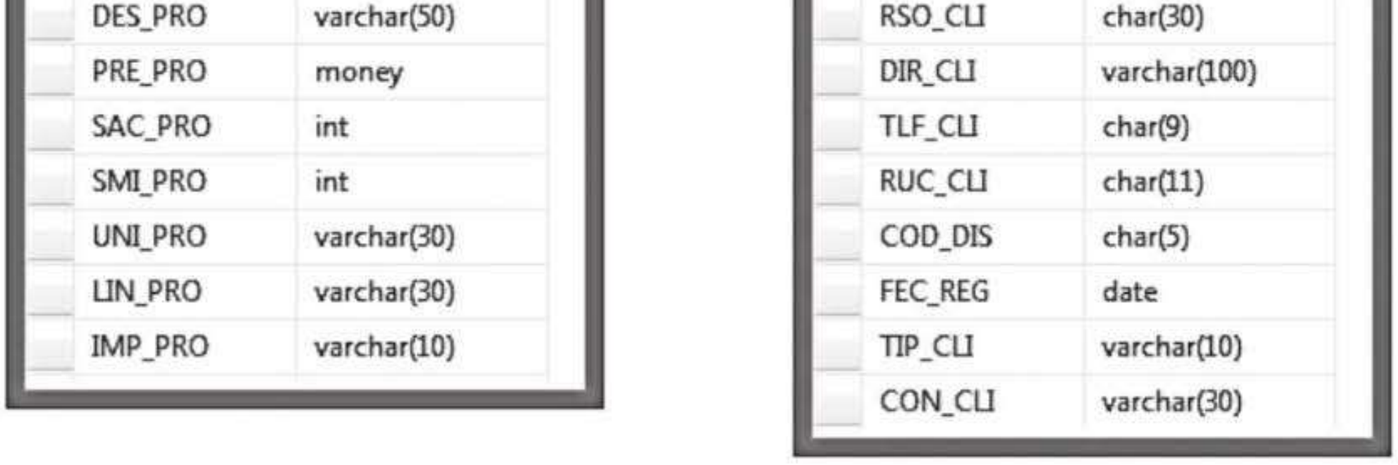
```
--Listar los vendedores cuyo año de inicio fue en los 90.
```

```
CODIGO NOMBRE              SUELDO                FECHA INICIO DISTRITO
------ ------------------------------------------ ----------- -------------
V03    CARLOS AREVALO      1500.00               1992-02-10   SAN BORJA
V04    CESAR OJEDA         850.00                1999-08-11   SURCO
V05    JULIO VEGA          1500.00               1997-11-05   SURCO
V06    ANA ORTEGA          1200.00               1994-11-05   SAN MIGUEL
V07    JOSE PALACIOS       2500.00               1997-12-02   JESUS MARIA
V08    RUBEN GOMEZ         1450.00               1996-12-08   LA MOLINA
V09    PATRICIA ARCE       1800.00               1997-12-05   LA MOLINA
V10    RENATO PEREZ        1550.00               1998-11-08   SURCO

(8 row(s) affected)
```

```
SELECT V.*
       FROM V_LISTAVENDEDOR V
       WHERE YEAR(V.[FECHA INICIO]) BETWEEN 1990 AND 1999
GO
```

A continuación, se observa el siguiente diagrama de base de datos:



Implementar una vista que permita mostrar información del número de factura, nombre del cliente, descripción del producto y subtotal facturado.

```
--VALIDACIÓN DE LA VISTA
IF OBJECT_ID('V_DETALLEFACTURA') IS NOT NULL
        DROP VIEW V_DETALLEFACTURA
GO

--CREANDO LA VISTA
CREATE VIEW V_DETALLEFACTURA
AS
        SELECT F.NUM_FAC AS NUMERO,
                C.RSO_CLI AS CLIENTE,
                P.DES_PRO AS PRODUCTO,
                D.CAN_VEN*D.PRE_VEN AS SUBTOTAL
                FROM FACTURA F
                JOIN CLIENTE C ON F.COD_CLI=C.COD_CLI
                JOIN DETALLE_FACTURA D ON D.NUM_FAC=F.NUM_FAC
                JOIN PRODUCTO P ON P.COD_PRO=D.COD_PRO
GO

--PRUEBA:
SELECT * FROM V_DETALLEFACTURA
GO
```

```
NUMERO      CLIENTE      PRODUCTO                                  SUBTOTAL
----------- ------------------------------------------------------------
FA001       FINSETH      PORTA DISKETTES                           30.00
FA001       FINSETH      TAJADOR METAL                             625.00
FA001       FINSETH      FOLDER MANILA OFICIO                      220.00
FA002       COREFO       PAPEL PERIODICO 75 GRAMOS                 80.00
FA003       SERVIEMSA    PAPEL BOND OFICIO                         400.00
FA003       SERVIEMSA    TAJADOR METAL                             120.00
FA003       SERVIEMSA    LAPICERO NEGRO                            252.00
FA003       SERVIEMSA    LAPICERO ROJO                             120.00
FA004       CARDELI      PAPEL PERIODICO 75 GRAMOS                 30.00
FA004       CARDELI      BORRADOR DE TINTA                         250.00
FA005       MEBA         PAPEL BULKY                               200.00
FA005       MEBA         CARTUCHO TINTA COLOR                      2500.00
FA005       MEBA         FOLDER PLASTICO A-4                       300.00
FA006       PROMINENT    PAPEL BOND OFICIO                         700.00
FA006       PROMINENT    PAPEL BULKY                               150.00
FA006       PROMINENT    BORRADOR DE TINTA                         60.00
FA007       COREFO       PAPEL BULKY                               240.00
FA007       COREFO       TAJADOR PLASTICO                          48.00
(50 row(s) affected)
```

```
--Listar las facturas cuyo subtotal se encuentre entre $30 y $100
SELECT D.*
       FROM V_DETALLEFACTURA D
       WHERE D.SUBTOTAL BETWEEN 30 AND 100
GO
```

```
NUMERO      CLIENTE      PRODUCTO                          SUBTOTAL
----------- ------------ --------------------------------- ---------
FA001       FINSETH      PORTA DISKETTES                   30.00
FA002       COREFO       PAPEL PERIODICO 75 GRAMOS         80.00
FA004       CARDELI      PAPEL PERIODICO 75 GRAMOS         30.00
FA006       PROMINENT    BORRADOR DE TINTA                 60.00
FA007       COREFO       TAJADOR PLASTICO                  48.00
FA009       INTECH       CARTUCHO TINTA COLOR              100.00
FA012       KADIA        CARTUCHO TINTA COLOR              50.00
FA013       FILASUR      PAPEL BULKY                       60.00
FA013       FILASUR      SOBRE MANILA OFICIO               48.00
FA014       CRAMER       BORRADOR DE TINTA                 80.00
FA015       MEBA         SOBRE MANILA A-4                  54.00
FA016       CARDELI      CAJA DE DISKETTES * 10            60.00
FA018       CARDELI      PAPEL BULKY                       48.00
FA019       INTECH       PAPEL BULKY                       60.00
FA019       INTECH       CARTUCHO TINTA COLOR              90.00
FA020       HAYASHI      SOBRE MANILA OFICIO               75.00

(16 row(s) affected)
```

```
--Listar las facturas del cliente COREFO
SELECT D.*
       FROM V_DETALLEFACTURA D
       WHERE D.CLIENTE = 'COREFO'
GO
```

```
NUMERO      CLIENTE     PRODUCTO                          SUBTOTAL
----------- ----------- --------------------------------- ---------
FA002       COREFO      PAPEL PERIODICO 75 GRAMOS         80.00
FA007       COREFO      PAPEL BULKY                       240.00
FA007       COREFO      TAJADOR PLASTICO                  48.00
FA011       COREFO      PAPEL BOND OFICIO                 700.00

(4 row(s) affected)
```

4.15.3 Mantenimiento de datos mediante vistas

Las vistas pueden gestionar el mantenimiento de registros como si se tratase de una tabla, para lo cual se debe tener en cuenta que solo se pueden ejecutar tres tipos de sentencias como son Insert, Update y Delete. Asimismo, se debe considerar que el mantenimiento se realiza a la vista así como a la tabla original de forma automática.

Implementar una vista que permita realizar el mantenimiento de la tabla CLIENTE.

```
--VALIDACIÓN DE LA VISTA
IF OBJECT_ID('V_MANTENIMIENTOPRODUCTO') IS NOT NULL
        DROP VIEW V_MANTENIMIENTOPRODUCTO
GO

--CREANDO LA VISTA
CREATE VIEW V_MANTENIMIENTOPRODUCTO
AS
        SELECT * FROM PRODUCTO
GO

--PRUEBA:
SELECT * FROM V_MANTENIMIENTOPRODUCTO
GO
```

	COD_PRO	DES_PRO	PRE_PRO	SAC_PRO	SMI_PRO	UNI_PRO	LIN_PRO	IMP_PRO
1	P001	PAPEL BOND A-4 ATLAS	18.50	200	1500	MLL	2	V
2	P002	PAPEL BOND OFICIO	37.00	100	1500	MLL	2	F
3	P003	PAPEL BULKY	12.00	498	1000	MLL	2	V
4	P004	PAPEL PERIODICO 75 GRAMOS	11.00	4285	1000	MLL	2	F
5	P005	CARTUCHO TINTA NEGRA	21.00	100	30	UNI	1	F
6	P006	CARTUCHO TINTA COLOR	47.00	58	35	UNI	1	F
7	P007	PORTA DISKETTES	5.50	300	100	UNI	1	V
8	P008	CAJA DE DISKETTES * 10	32.00	125	180	UNI	1	F
9	P009	BORRADOR DE TINTA	12.00	100	500	DOC	3	F
10	P010	BORRADOR BLANCO	5.00	2000	400	DOC	3	F
11	P011	TAJADOR METAL	22.00	1120	300	DOC	3	F
12	P012	TAJADOR PLASTICO	14.00	608	300	DOC	3	F
13	P013	FOLDER MANILA OFICIO	22.00	200	150	CIE	3	F
14	P014	FOLDER MANILA A-4	22.00	150	150	CIE	3	V

```
--PRUEBA DE INSERCIÓN DE REGISTROS
INSERT INTO V_MANTENIMIENTOPRODUCTO
        VALUES('P032','PORTALAPICEROS',5,100,10,'UNI',1,'V')
GO

--Actualizar el valor de importado a F (False) al producto con código P032.
UPDATE V_MANTENIMIENTOPRODUCTO
        SET COD_PRO='P032'
        WHERE IMP_PRO ='F'
GO

--PRUEBA DE ELIMINACIÓN DE REGISTROS
DELETE V_MANTENIMIENTOPRODUCTO
        WHERE COD_PRO='P032'
GO
```

4.15.4 Simplificación de consultas complejas mediante el uso de vistas

Clasificación de las vistas

Vistas horizontales

Un uso común de las vistas es restringir el acceso de un usuario a únicamente filas seleccionadas de una tabla. A continuación, se observa cómo implementar tres vistas horizontales que permitan separar las facturas por sus diferentes estados.

```
NUM_FAC     FEC_FAC    COD_CLI FEC_CAN    EST_FAC    COD_VEN POR_IGV
----------- ---------- ------- ---------- ---------- ------- -----------
FA007       2013-05-10 C019    2013-10-10 1          V09     0
FA012       2013-01-12 C014    2013-01-12 1          V04     0
FA017       2013-06-01 C015    2013-06-01 1          V06     0

(3 row(s) affected)

NUM_FAC     FEC_FAC    COD_CLI FEC_CAN    EST_FAC    COD_VEN POR_IGV
----------- ---------- ------- ---------- ---------- ------- -----------
FA001       2013-06-07 C001    2013-05-08 2          V01     0
FA003       2013-01-09 C003    2013-03-10 2          V04     0
FA004       2013-06-09 C016    2013-05-10 2          V07     0
FA005       2013-01-10 C015    2013-12-10 2          V08     0
FA008       2013-04-10 C012    2013-06-10 2          V10     0
FA009       2013-03-10 C008    2013-10-10 2          V09     0
FA010       2013-01-10 C017    2013-06-10 2          V02     0
FA011       2013-10-09 C019    2013-10-09 2          V05     0
FA014       2012-01-12 C020    2013-01-12 2          V09     0
FA015       2012-08-12 C015    2013-06-01 2          V07     0
FA016       2013-06-01 C016    2013-09-01 2          V05     0
FA018       2013-03-02 C016    2013-04-02 2          V10     0
FA019       2013-07-02 C008    2013-01-03 2          V03     0
FA020       2013-06-02 C013    2013-10-03 2          V02     0
FA021       2013-06-02 C013    2013-10-03 2          V02     0

(15 row(s) affected)

NUM_FAC     FEC_FAC    COD_CLI FEC_CAN    EST_FAC    COD_VEN POR_IGV
----------- ---------- ------- ---------- ---------- ------- -----------
FA002       2013-06-07 C019    2013-05-08 3          V02     0
FA006       2013-01-08 C009    2013-05-08 3          V05     0
FA013       2013-01-12 C011    2013-01-12 3          V08     0

(3 row(s) affected)
```

```
--VALIDACIÓN DE LA VISTA 1
IF OBJECT_ID('V_FACTURAS_EST1') IS NOT NULL
        DROP VIEW V_FACTURAS_EST1
GO

--CREANDO LA VISTA
CREATE VIEW V_FACTURAS_EST1
AS
        SELECT * FROM FACTURA F WHERE F.EST_FAC=1
GO

--VALIDACIÓN DE LA VISTA 2
IF OBJECT_ID('V_FACTURAS_EST2') IS NOT NULL
        DROP VIEW V_FACTURAS_EST2
GO
```

```
--CREANDO LA VISTA
CREATE VIEW V_FACTURAS_EST2
AS
	SELECT * FROM FACTURA F WHERE F.EST_FAC=2
GO

--VALIDACIÓN DE LA VISTA 3
IF OBJECT_ID('V_FACTURAS_EST3') IS NOT NULL
	DROP VIEW V_FACTURAS_EST3
GO

--CREANDO LA VISTA
CREATE VIEW V_FACTURAS_EST3
AS
	SELECT * FROM FACTURA F WHERE F.EST_FAC=3
GO


--PRUEBA DE VISTAS HORIZONTALES:
SELECT * FROM V_FACTURAS_EST1
SELECT * FROM V_FACTURAS_EST2
SELECT * FROM V_FACTURAS_EST3
GO
```

Vistas verticales

Una vista vertical se caracteriza por mostrar ciertas columnas que el diseñador desea que visualice el usuario. A continuación, se visualizan los registros de la tabla CLIENTE:

```
COD_CLI RSO_CLI       DIR_CLI                          TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------- ------------------------------------------------------- ----------- ------- ---------- ---------- ------------------
C001    FINSETH       AV. LOS VIÑEDOS 150              4342318   48632081    D05     2008-12-10 1          ALICIA BARRETO
C002    ORBI          AV. EMILIO CAVENECIA 225         4406335   57031642    D04     2013-02-01 2          ALFONSO BELTRAN
C003    SERVIEMSA     JR. COLLAGATE 522                75012403  NULL        D05     2011-06-03 2          CHRISTIAN LAGUNA
C004    ISSA          CALLE LOS AVIADORES 263          3725910   46720159    D01     2012-09-12 1          LUIS APUMAYTA
C005    MASS          AV. TOMAS MARSANO 880            4446177   83175942    D14     2013-10-01 1          KATIA ARMEJO
C006    BERKER        AV. LOS PROCERES 521             3810322   54890124    D05     2013-07-05 1          JUDITH ASTE
C007    FIDENZA       JR. EL NIQUEL 282                5289034   16204790    D20     2012-10-02 2          HECTOR VIVANCO
C008    INTECH        AV. SAN LUIS 2619  5TO P         2249493   34021824    D09     2011-01-07 2          CARLOS VILLANUEVA
C009    PROMINENT     JR. IQUIQUE 132                  43233519  NULL        D11     2010-06-11 1          JORGE VALDIVIA
C010    LANDU         AV.NICOLAS DE AYLLON 1453        3267840   30405261    D05     2013-11-06 2          RAQUEL ESPINOZA
C011    FILASUR       AV. EL SANTUARIO 1189            4598175   70345201    D26     2012-08-04 1          ANGELICA VIVAS
C012    SUCERTE       JR. GRITO DE HUAURA 114          4206434   62014503    D05     2011-11-08 1          KARINA VEGA
C013    HAYASHI       JR. AYACUCHO 359                 42847990  NULL        D22     2013-03-09 2          ERNESTO UEHARA
C014    KADIA         AV.SANTA CRUZ 1332 OF.201        4412418   22202915    D06     2011-10-05 1          MIGUEL ARCE

(22 row(s) affected)
```

Crear una vista de los clientes, el cual permita visualizar los siguientes datos: código, nombre completo, descripción del distrito y teléfono.

```
--VALIDACIÓN DE LA VISTA
IF OBJECT_ID('V_LISTACLIENTE') IS NOT NULL
	DROP VIEW V_LISTACLIENTE
GO

--CREANDO LA VISTA
CREATE VIEW V_LISTACLIENTE
AS
SELECT C.COD_CLI AS CODIGO,
	C.RSO_CLI AS CLIENTE,
```

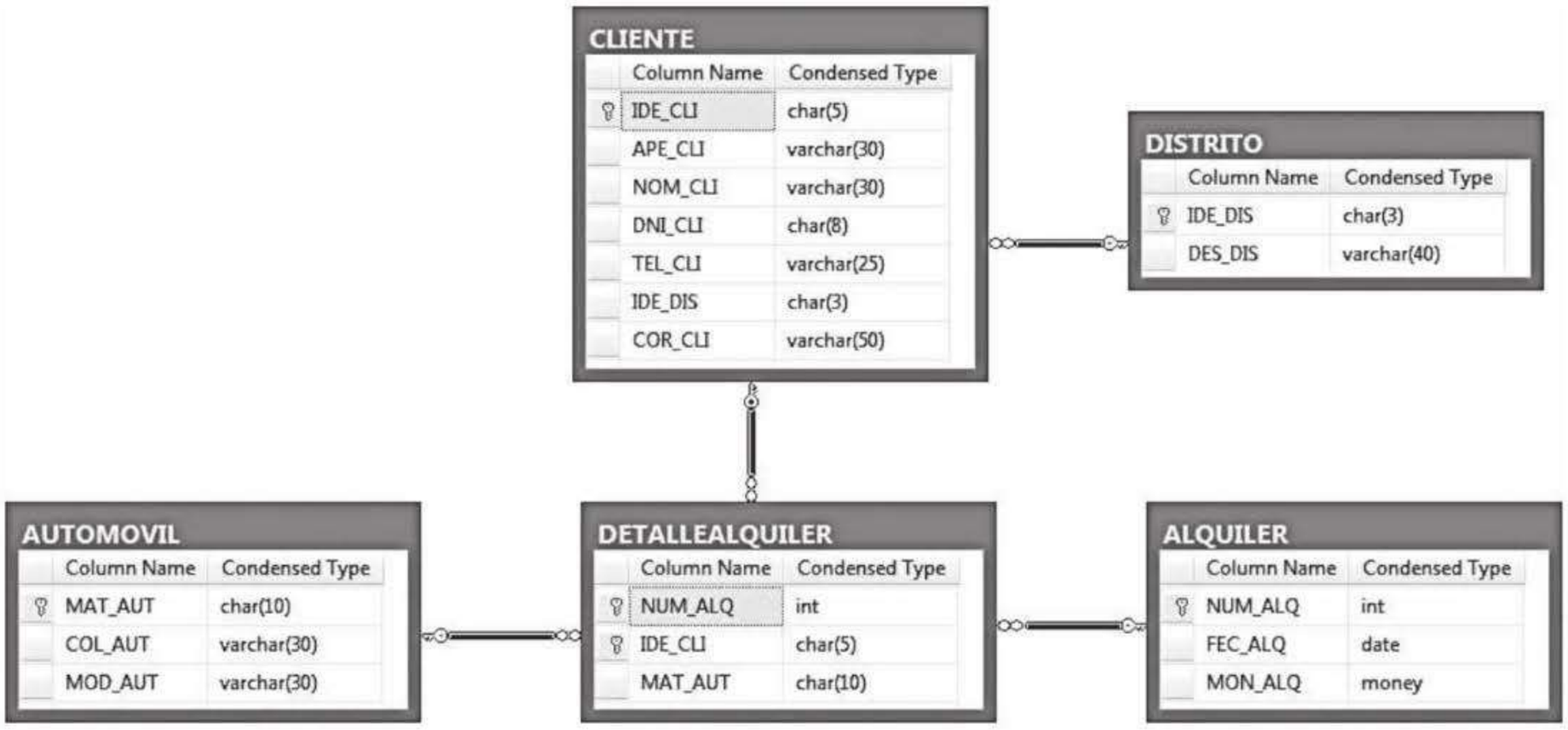
```
        C.DIR_CLI AS DIRECCION,
        D.NOM_DIS AS DISTRITO,
        C.FEC_REG AS [FECHA REG.]
        FROM CLIENTE C
        JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
GO

--PRUEBA:
SELECT * FROM V_LISTACLIENTE
GO
```

CODIGO	CLIENTE	DIRECCION	DISTRITO	FECHA REG.
C001	FINSETH	AV. LOS VIÑEDOS 150	SAN MIGUEL	2008-12-10
C002	ORBI	AV. EMILIO CAVENECIA 225	LA MOLINA	2013-02-01
C003	SERVIEMSA	JR. COLLAGATE 522	SAN MIGUEL	2011-06-03
C004	ISSA	CALLE LOS AVIADORES 263	SURCO	2012-09-12
C005	MASS	AV. TOMAS MARSANO 880	SURQUILLO	2013-10-01
C006	BERKER	AV. LOS PROCERES 521	SAN MIGUEL	2013-07-05
C007	FIDENZA	JR. EL NIQUEL 282	LOS OLIVOS	2012-10-02
C008	INTECH	AV. SAN LUIS 2619 5TO P	SAN BORJA	2011-01-07
C009	PROMINENT	JR. IQUIQUE 132	BREÑA	2010-06-11
C010	LANDU	AV.NICOLAS DE AYLLON 1453	SAN MIGUEL	2013-11-06
C011	FILASUR	AV. EL SANTUARIO 1189	SAN JUAN DE LURIGANCHO	2012-08-04
C012	SUCERTE	JR. GRITO DE HUAURA 114	SAN MIGUEL	2011-11-08
C013	HAYASHI	JR. AYACUCHO 359	LIMA - CERCADO	2013-03-09

Caso desarrollado

Si se cuenta con el siguiente diagrama de base de datos:



Y con la siguiente información por tablas:

```
SELECT * FROM CLIENTE
```

```
IDE_CLI APE_CLI    NOM_CLI      DNI_CLI  TEL_CLI     IDE_DIS COR_CLI
------- ---------- ------------ -------- ----------- ------- --------
CL001   LOPEZ      MARIA        10856985 526-5254    L03
CL002   GOMEZ      JOSE         02188525 585-8212    L02
CL003   ACOSTA     JESUS        07415482 485-1598    L01
CL004   ARIAS      GUADSLUPE    08995255 258-8569    L04
CL005   CARLOS     MANUEL       10418525 296-9685    L08
CL006   GERZ       BRIGITTE     10952155 585-1254    L06
CL007   BARBOSA    MARISOL      78514855 584-5869    L04

(7 row(s) affected)
```

```
SELECT * FROM DISTRITO
```

```
IDE_DIS DES_DIS
------- --------------------
L01     LIMA
L02     LINCE
L03     BREÑA
L04     LOS OLIVOS
L05     RIMAC
L06     SAN MARTIN DE PORRES
L07     SAN LUIS
L08     SAN BORJA

(8 row(s) affected)
```

```
SELECT * FROM AUTOMOVIL
```

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- --------------
AF-456     ROJO                             SEDAN
D4-243     NEGRO                            SEDAN
FD-463     PLATA                            STATION WAGON
FG-654     PLATA                            L10
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
HY-345     PLATA                            CHERY
SF-112     ROJO                             LI
YT-457     NEGRO                            STATION WAGON

(10 row(s) affected)
```

```
SELECT * FROM ALQUILER
```

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
1           2010-03-12 185.00
2           2011-01-02 100.00
3           2011-05-01 54.00
4           2012-08-11 85.00
5           2012-06-22 18.00
6           2012-01-30 15.00
7           2012-04-16 165.00
8           2012-05-23 135.00
9           2012-03-21 225.00

(9 row(s) affected)
```

```
SELECT * FROM DETALLEALQUILER
```

```
NUM_ALQ     IDE_CLI MAT_AUT
----------- ------- ----------
1           CL002   GG-654
2           CL002   GT-532
3           CL006   FD-463
3           CL007   GT-532
4           CL005   H5-455
5           CL004   SF-112
6           CL004   YT-457
7           CL003   GT-532

(8 row(s) affected)
```

Implemente las siguientes consultas:

- Listar los registros de todas las tablas.
- Listar los colores de autos permitidos en el registro.
- Listar los alquileres de automóviles ordenados por la fecha de alquiler en forma descendente; y de forma ascendente por el monto de alquiler ante la posible coincidencia de la fecha de alquiler.
- Listar los dos primeros registros de alquiler que cuenten con el menor valor en el monto de alquiler.
- Mostrar el siguiente listado:

NUMERO DE ALQUILER	FECHA ALQUILER	MONTO ALQUILER
999	99/99/9999	9999.99
999	99/99/9999	9999.99
999	99/99/9999	9999.99

- Mostrar el siguiente listado:

CÓDIGO	CLIENTE	TELÉFONO	DNI
XXXXX	XXXXX XXXXXX	9999999	99999999
XXXXX	XXXXX XXXXXX	9999999	99999999
XXXXX	XXXXX XXXXXX	9999999	99999999

- Crear una tabla llamada AUTOS_ROJOS que solo contenga información de los automóviles de color rojo a partir de la tabla AUTOMÓVIL.
- Listar solo los automóviles de color negro.
- Listar a los clientes cuyo código de distrito sea L04.
- Listar a los clientes cuya letra inicial de su nombre inicie con la letra "M".
- Listar los alquileres cuyo monto de alquiler se encuentre en el rango de $100 a $150.
- Listar todos los automóviles a excepción de los que tengan color plata.
- Mostrar el siguiente listado usando unión interna:

CÓDIGO	CLIENTE	DISTRITO	DNI
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999

- Mostrar el siguiente listado usando unión interna:

ALQUILER	FECHA	AUTOMOVIL	CLIENTE
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX

- Listar los registros de alquiler que aún no cuentan con detalle de alquiler.
- Listar a los clientes que aún no registran alquiler.
- Listar el total de clientes por distrito.
- Listar el total de automóviles por tipo de color.
- Listar el monto acumulado por año desde la tabla ALQUILER.
- Mostrar el siguiente listado usando subconsulta:

CÓDIGO	CLIENTE	DISTRITO	DNI
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999
XXXXX	XXXXX XXXXXX	XXXXXXXX	99999999

- Mostrar el siguiente listado usando subconsulta:

ALQUILER	FECHA	AUTOMOVIL	CLIENTE
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX
9999	99/99/9999	XXX-XXXX	XXXXXXXXX XXXXXXXXX

Solución:

```
--•    Listar los registros de todas las tablas.
```

```
IDE_CLI APE_CLI                        NOM_CLI                        DNI_CLI  TEL_CLI                   IDE_DIS COR_CLI
------- ------------------------------ ------------------------------ -------- ------------------------- ------- ---------
CL001   LOPEZ                          MARIA                          10856985 526-5254                  L03
CL002   GOMEZ                          JOSE                           02188525 585-8212                  L02
CL003   ACOSTA                         JESUS                          07415482 485-1598                  L01
CL004   ARIAS                          GUADSLUPE                      08995255 258-8569                  L04
CL005   CARLOS                         MANUEL                         10418525 296-9685                  L08
CL006   GERZ                           BRIGITTE                       10952155 585-1254                  L06
CL007   BARBOSA                        MARISOL                        78514855 584-5869                  L04
(7 row(s) affected)

NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ---------------------
1           2010-03-12 185.00
2           2011-01-02 100.00
3           2011-05-01 54.00
4           2012-08-11 85.00
5           2012-06-22 18.00
6           2012-01-30 15.00
7           2012-04-16 165.00
8           2012-05-23 135.00
9           2012-03-21 225.00
(9 row(s) affected)
MAT_AUT    COL_AUT                        MOD_AUT
---------- ------------------------------ ------------------------------
AF-456     ROJO                           SEDAN
D4-243     NEGRO                          SEDAN
FD-463     PLATA                          STATION WAGON
FG-654     PLATA                          L10
GG-654     ROJO                           CAMIONETA
GT-532     ROJO                           SEDAN
H5-455     ROJO                           STATION WAGON
HY-345     PLATA                          CHERY
SF-112     ROJO                           LI
YT-457     NEGRO                          STATION WAGON
(10 row(s) affected)
```

```
IDE_DIS DES_DIS
------- --------------------
L01     LIMA
L02     LINCE
L03     BREÑA
L04     LOS OLIVOS
L05     RIMAC
L06     SAN MARTIN DE PORRES
L07     SAN LUIS
L08     SAN BORJA

(8 row(s) affected)

NUM_ALQ     IDE_CLI MAT_AUT
----------- ------- ----------
1           CL002   GG-654
2           CL002   GT-532
3           CL006   FD-463
3           CL007   GT-532
4           CL005   H5-455
5           CL004   SF-112
6           CL004   YT-457
7           CL003   GT-532

(8 row(s) affected)
```

```
SELECT * FROM CLIENTE
SELECT * FROM ALQUILER
SELECT * FROM AUTOMOVIL
SELECT * FROM DISTRITO
SELECT * FROM DETALLEALQUILER

--•    Listar los colores de autos permitidos en el registro.
```

```
COL_AUT
--------------------
NEGRO
PLATA
ROJO

(3 row(s) affected)
```

```
SELECT DISTINCT A.COL_AUT
       FROM AUTOMOVIL A
GO

--•    Listar los alquileres de automóviles ordenados por la fecha de
alquiler en forma descendente; y de forma ascendente por el monto de
alquiler ante la posible coincidencia de la fecha de alquiler.
```

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ---------
4           2012-08-11 85.00
5           2012-06-22 18.00
8           2012-05-23 135.00
7           2012-04-16 165.00
9           2012-03-21 225.00
6           2012-01-30 15.00
3           2011-05-01 54.00
2           2011-01-02 100.00
1           2010-03-12 185.00

(9 row(s) affected)
```

```
SELECT A.*
       FROM ALQUILER A
       ORDER BY A.FEC_ALQ DESC, A.MON_ALQ ASC
GO

--•    Listar los dos primeros registros de alquiler que cuenten con el
menor valor en el monto de alquiler.
```

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
6           2012-01-30 15.00
5           2012-06-22 18.00

(2 row(s) affected)
```

```
SELECT TOP 2 *
       FROM ALQUILER A
       ORDER BY A.MON_ALQ ASC
GO

--•    Mostrar el siguiente listado:
--NUMERO DE ALQUILER  FECHA ALQUILER        MONTO ALQUILER
--999                 99/99/9999           9999.99
--999                 99/99/9999           9999.99
--999                 99/99/9999           9999.99
```

```
NUMERO DE ALQUILER FECHA ALQUILER MONTO ALQUILER
------------------ -------------- ---------------
1                  2010-03-12     185.00
2                  2011-01-02     100.00
3                  2011-05-01     54.00
4                  2012-08-11     85.00
5                  2012-06-22     18.00
6                  2012-01-30     15.00
7                  2012-04-16     165.00
8                  2012-05-23     135.00
9                  2012-03-21     225.00

(9 row(s) affected)
```

```
SELECT A.NUM_ALQ AS [NUMERO DE ALQUILER],
       A.FEC_ALQ AS [FECHA ALQUILER],
       A.MON_ALQ AS [MONTO ALQUILER]
       FROM ALQUILER A
GO

--•    Mostrar el siguiente listado:
--CÓDIGO      CLIENTE        TELÉFONO      DNI
--XXXXX       XXXXX XXXXXX   9999999       99999999
--XXXXX       XXXXX XXXXXX   9999999       99999999
--XXXXX       XXXXX XXXXXX   9999999       99999999
```

```
CODIGO CLIENTE                  TELEFNO                   DNI
------ ---------------------------------------------- --------
CL001  MARIA LOPEZ              526-5254                  10856985
CL002  JOSE GOMEZ               585-8212                  02188525
CL003  JESUS ACOSTA             485-1598                  07415482
CL004  GUADSLUPE ARIAS          258-8569                  08995255
CL005  MANUEL CARLOS            296-9685                  10418525
CL006  BRIGITTE GERZ            585-1254                  10952155
CL007  MARISOL BARBOSA          584-5869                  78514855

(7 row(s) affected)
```

```
SELECT C.IDE_CLI AS CODIGO,
       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
       C.TEL_CLI AS TELEFNO,
       C.DNI_CLI AS DNI
       FROM CLIENTE C
GO
```

```
--•    Crear una tabla llamada AUTOS_ROJOS que solo contenga información
de los automóviles de color rojo a partir de la tabla AUTOMÓVIL.
```

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- --------------
AF-456     ROJO                             SEDAN
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
SF-112     ROJO                             LI

(5 row(s) affected)
```

```
SELECT A.* INTO AUTOS_ROJOS
       FROM AUTOMOVIL A
       WHERE A.COL_AUT = 'ROJO'
GO
--•    Listar solo los automóviles de color negro.
```

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- --------------
D4-243     NEGRO                            SEDAN
YT-457     NEGRO                            STATION WAGON

(2 row(s) affected)
```

```
SELECT A.*
       FROM AUTOMOVIL A
       WHERE A.COL_AUT = 'NEGRO'
GO
--•    Listar a los clientes cuyo código de distrito sea L04.
```

```
IDE_CLI APE_CLI       NOM_CLI        DNI_CLI  TEL_CLI   IDE_DIS COR_CLI
------- ------------------------------------ ----------------------------
CL004   ARIAS         GUADSLUPE      08995255 258-8569  L04
CL007   BARBOSA       MARISOL        78514855 584-5869  L04

(2 row(s) affected)
```

```
SELECT C.*
       FROM CLIENTE C
       WHERE C.IDE_DIS = 'L04'
GO
--•    Listar a los clientes cuya letra inicial de su nombre inicie con
la letra "M".
```

```
IDE_CLI APE_CLI      NOM_CLI      DNI_CLI  TEL_CLI      IDE_DIS COR_CLI
------- ------------------------------- ------------------------------
CL001   LOPEZ        MARIA        10856985 526-5254     L03
CL005   CARLOS       MANUEL       10418525 296-9685     L08
CL007   BARBOSA      MARISOL      78514855 584-5869     L04

(3 row(s) affected)
```

```
SELECT C.*
        FROM CLIENTE C
        WHERE C.NOM_CLI LIKE 'M%'
GO
--•     Listar los alquileres cuyo monto de alquiler se encuentre en el
rango de $100 a $150.
```

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
2           2011-01-02 100.00
8           2012-05-23 135.00

(2 row(s) affected)
```

```
SELECT A.*
        FROM ALQUILER A
        WHERE A.MON_ALQ BETWEEN 100 AND 150
GO
SELECT A.*
        FROM ALQUILER A
        WHERE A.MON_ALQ>= 100 AND A.MON_ALQ<=150
GO
--•    Listar todos los automóviles a excepción de los que tengan color plata.
```

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- ----------------
AF-456     ROJO                             SEDAN
D4-243     NEGRO                            SEDAN
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
SF-112     ROJO                             LI
YT-457     NEGRO                            STATION WAGON

(7 row(s) affected)
```

```
SELECT A.*
        FROM AUTOMOVIL A
        WHERE NOT A.COL_AUT='PLATA'
GO

SELECT A.*
        FROM AUTOMOVIL A
        WHERE A.COL_AUT='NEGRO' OR A.COL_AUT='ROJO'
GO
--•     Mostrar el siguiente listado usando unión interna:
--CÓDIGO       CLIENTE                DISTRITO       DNI
--XXXXX        XXXXX  XXXXXX          XXXXXXXX       99999999
--XXXXX        XXXXX  XXXXXX          XXXXXXXX       99999999
--XXXXX        XXXXX  XXXXXX          XXXXXXXX       99999999
```

```
CODIGO CLIENTE              DISTRITO                  DNI
------ -------------------------------------------------
CL001  MARIA LOPEZ          BREÑA                     10856985
CL002  JOSE GOMEZ           LINCE                     02188525
CL003  JESUS ACOSTA         LIMA                      07415482
CL004  GUADSLUPE ARIAS      LOS OLIVOS                08995255
CL005  MANUEL CARLOS        SAN BORJA                 10418525
CL006  BRIGITTE GERZ        SAN MARTIN DE PORRES      10952155
CL007  MARISOL BARBOSA      LOS OLIVOS                78514855

(7 row(s) affected)
```

```
SELECT C.IDE_CLI AS CODIGO,
       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
       D.DES_DIS AS DISTRITO,
       C.DNI_CLI AS DNI
       FROM CLIENTE C
       JOIN DISTRITO D ON C.IDE_DIS = D.IDE_DIS
GO

--•    Mostrar el siguiente listado usando unión interna:
--ALQUILER    FECHA          AUTOMOVIL     CLIENTE
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
```

```
ALQUILER    FECHA      AUTOMOVIL  CLIENTE
----------- ---------- ---------- -----------------
1           2010-03-12 GG-654     JOSE GOMEZ
2           2011-01-02 GT-532     JOSE GOMEZ
3           2011-05-01 FD-463     BRIGITTE GERZ
3           2011-05-01 GT-532     MARISOL BARBOSA
4           2012-08-11 H5-455     MANUEL CARLOS
5           2012-06-22 SF-112     GUADSLUPE ARIAS
6           2012-01-30 YT-457     GUADSLUPE ARIAS
7           2012-04-16 GT-532     JESUS ACOSTA

(8 row(s) affected)
```

```
SELECT A.NUM_ALQ AS ALQUILER,
       A.FEC_ALQ AS FECHA,
       AU.MAT_AUT AS AUTOMOVIL,
       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE
       FROM ALQUILER A
       JOIN DETALLEALQUILER D ON A.NUM_ALQ=D.NUM_ALQ
       JOIN AUTOMOVIL AU ON AU.MAT_AUT=D.MAT_AUT
       JOIN CLIENTE C ON C.IDE_CLI=D.IDE_CLI
GO

--•    Listar los registros de alquiler que aún no cuentan con detalle de
alquiler.
```

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
8           2012-05-23 135.00
9           2012-03-21 225.00

(2 row(s) affected)
```

```
SELECT A.*
       FROM ALQUILER A
       LEFT JOIN DETALLEALQUILER D ON A.NUM_ALQ=D.NUM_ALQ
       WHERE D.IDE_CLI IS NULL
GO
SELECT A.*
       FROM DETALLEALQUILER D
       RIGHT JOIN ALQUILER A ON A.NUM_ALQ=D.NUM_ALQ
       WHERE D.IDE_CLI IS NULL
GO
--•    Listar a los clientes que aún no registran ALQUILER.
```

```
IDE_CLI APE_CLI        NOM_CLI     DNI_CLI  TEL_CLI     IDE_DIS COR_CLI
------- -------------------------------- ---------- ---------
CL001   LOPEZ          MARIA       10856985 526-5254    L03

(1 row(s) affected)
```

```
SELECT C.*
       FROM CLIENTE C
       LEFT JOIN DETALLEALQUILER D ON D.IDE_CLI=C.IDE_CLI
       WHERE D.IDE_CLI IS NULL
GO
SELECT C.*
       FROM DETALLEALQUILER D
       RIGHT JOIN CLIENTE C ON D.IDE_CLI=C.IDE_CLI
       WHERE D.IDE_CLI IS NULL
GO
--•    Listar el total de clientes por distrito.
```

```
DISTRITO                                  TOTAL DE CLIENTES
----------------------------------------- -----------------
BREÑA                                     1
LIMA                                      1
LINCE                                     1
LOS OLIVOS                                2
SAN BORJA                                 1
SAN MARTIN DE PORRES                      1

(6 row(s) affected)
```

```
SELECT D.DES_DIS AS DISTRITO, COUNT(*) AS [TOTAL DE CLIENTES]
       FROM CLIENTE C
       JOIN DISTRITO D ON C.IDE_DIS=D.IDE_DIS
       GROUP BY D.DES_DIS
GO
```

```
--•    Listar el total de automóviles por tipo de color.
```

```
COLOR                          TOTAL DE AUTOS
------------------------------ --------------
NEGRO                          2
PLATA                          3
ROJO                           5

(3 row(s) affected)
```

```
SELECT A.COL_AUT AS COLOR, COUNT(*) AS [TOTAL DE AUTOS]
       FROM AUTOMOVIL A
       GROUP BY A.COL_AUT
GO

--•    Listar el monto acumulado por año desde la tabla ALQUILER.
```

```
AÑO         ACUMULADO
----------- -----------
2010        185.00
2011        154.00
2012        643.00

(3 row(s) affected)
```

```
SELECT YEAR(A.FEC_ALQ) AS AÑO,
       SUM(A.MON_ALQ) AS ACUMULADO
       FROM ALQUILER A
       GROUP BY YEAR(A.FEC_ALQ)
GO

--•    Mostrar el siguiente listado usando subconsulta:
--CÓDIGO      CLIENTE        DISTRITO      DNI
--XXXXX       XXXXX XXXXXX   XXXXXXXX      99999999
--XXXXX       XXXXX XXXXXX   XXXXXXXX      99999999
--XXXXX       XXXXX XXXXXX   XXXXXXXX      99999999
```

```
CODIGO CLIENTE            DISTRITO                                          DNI
------ ------------------------------------------------------------------ --------
CL001  MARIA LOPEZ        BREÑA                                             10856985
CL002  JOSE GOMEZ         LINCE                                             02188525
CL003  JESUS ACOSTA       LIMA                                              07415482
CL004  GUADSLUPE ARIAS    LOS OLIVOS                                        08995255
CL005  MANUEL CARLOS      SAN BORJA                                         10418525
CL006  BRIGITTE GERZ      SAN MARTIN DE PORRES                              10952155
CL007  MARISOL BARBOSA    LOS OLIVOS                                        78514855

(7 row(s) affected)
```

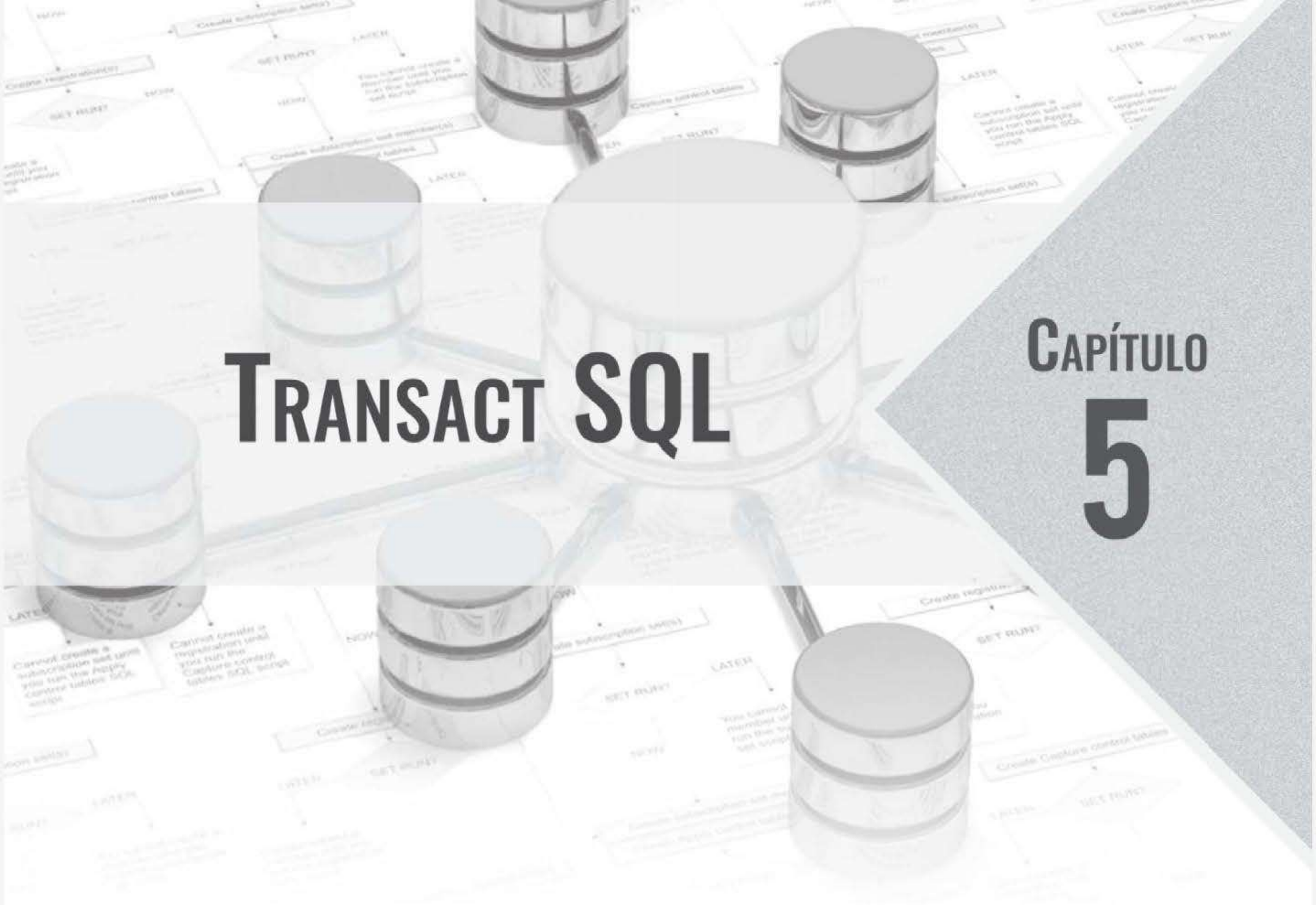
```
SELECT C.IDE_CLI AS CODIGO,
       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
       (SELECT D.DES_DIS FROM DISTRITO D WHERE D.IDE_DIS=C.IDE_DIS) AS
DISTRITO,
       C.DNI_CLI AS DNI
       FROM CLIENTE C
GO
```

```
--•    Mostrar el siguiente listado usando subconsulta:
--ALQUILER    FECHA          AUTOMOVIL     CLIENTE
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
--9999        99/99/9999     XXX-XXXX      XXXXXXXXX XXXXXXXXX
```

```
ALQUILER    FECHA      AUTOMOVIL  CLIENTE
----------- ---------- ---------- ------------------
1           2010-03-12 GG-654     JOSE GOMEZ
2           2011-01-02 GT-532     JOSE GOMEZ
3           2011-05-01 FD-463     BRIGITTE GERZ
3           2011-05-01 GT-532     MARISOL BARBOSA
4           2012-08-11 H5-455     MANUEL CARLOS
5           2012-06-22 SF-112     GUADSLUPE ARIAS
6           2012-01-30 YT-457     GUADSLUPE ARIAS
7           2012-04-16 GT-532     JESUS ACOSTA

(8 row(s) affected)
```

```
SELECT D.NUM_ALQ AS ALQUILER,
       (SELECT A.FEC_ALQ FROM ALQUILER A WHERE A.NUM_ALQ=D.NUM_ALQ) AS
FECHA,
       D.MAT_AUT AS AUTOMOVIL,
       (SELECT C.NOM_CLI+SPACE(1)+C.APE_CLI FROM CLIENTE C WHERE C.IDE_
CLI=D.IDE_CLI) AS CLIENTE
       FROM DETALLEALQUILER D
GO
```



CAPÍTULO 5

TRANSACT SQL

Competencia tecnológica:

- Implementar consultas usando programación Transact SQL.
- Integrar consultas con el uso de sentencias condicionales y repetitivas.

Contenido:

5.1 Introducción
5.2 Fundamentos de programación Transact SQL (T-SQL)
5.3 Estructuras de control
5.4 Control de errores en Transact SQL
5.5 Función @@error
5.6 Función Raiserror
- Caso desarrollado

5.1 Introducción

Transact SQL es el lenguaje de programación que proporciona Microsoft SQL Server para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación, ya que esta parte está limitada en SQL. Presenta las siguientes características:

- No tiene sensibilidad entre mayúsculas y minúsculas, es decir, las sentencias se podrán escribir tanto en mayúsculas como en minúsculas, y el servidor las interpretará de la misma manera.
- Se pueden incluir comentarios de una línea con el símbolo guion+guion y su comentario (--) o comentario de varias líneas con los símbolos /* */.
- El usuario no puede definir variables globales, ya que estas son propias de SQL Server y son identificadas con @@variableGlobal.
- En el caso de variables locales, se usa el operador DECLARE y siempre se empieza con @.

Con Transact SQL se pueden programar las siguientes unidades de SQL Server:

- procedimientos almacenados
- funciones
- triggers
- scripts

5.2 Fundamentos de programación Transact SQL (T-SQL)

Transact SQL amplía las sentencias SQL con una serie de extensiones que resultan de gran utilidad para la implementación de sentencias. Si no se toma en consideración la programación Transact SQL, se tendrá que seguir usando instrucciones simples, y la lógica condicional o repetitiva se tendrá que realizar desde un lenguaje de programación asociado a SQL Server.

5.2.1 Variables e identificadores

En Transact SQL se pueden declarar variables de tipo local, ya que las globales están implementadas y tienen como propietario a SQL Server. La visibilidad de dichas variables es limitada por las unidades de SQL Server, es decir, solo será accesible dentro de un proceso de lotes, un procedimiento almacenado, una función, etc. Características de las variables:

- Normalmente las variables se declaran al principio de un proceso por lotes, usando el operador DECLARE.
- El primer carácter de la variable debe ser un carácter alfabético (una letra); los demás caracteres pueden contener letras, símbolos o números.
- Los nombres de las variables pueden contener de uno a 128 caracteres.
- El identificador oficial de la variable local es @, el símbolo # representa a una tabla temporal o procedimiento temporal, y el símbolo ## indica un objeto global temporal.
- Para asignar un valor a una variable local se necesita el operador SET.

Formato para la declaración de variables locales:

```
Declarar una variable
DECLARE @nombreVariable TipoDatos

Múltiples declaraciones
DECLARE @nombreVariable1 TipoDatos,
                @nombreVariable1 TipoDatos,
              @nombreVariable1 TipoDatos
```

Donde:

- **@nombreVariable:** Es el nombre de la variable a emplear; debe empezar con @ obligatoriamente, ya que, si no se coloca, SQL lo interpretará como una columna de una tabla.
- **TipoDatos:** Es el ámbito de la variable, es decir, la capacidad que tendrá dicha variable como cadena, números, fechas, etc.

Formato para la asignación de valores a una variable local:

```
SET @nombreVariable = VALOR
```

Donde:

- **@nombreVariable:** Es el nombre de la variable previamente declarada con el operador DECLARE.
- **Valor:** Es el dato asignado a la variable; se debe considerar que dicho valor sea del mismo tipo con el que se declaró. Hay que tener en cuenta que, de acuerdo al tipo de datos, se deben asignar los valores; por ejemplo, para caracteres o fechas colocarlo entre dos comillas simples, y en los numéricos, sin comillas.

Formato para la declaración-asignación de una variable local:

```
DECLARE @nombreVariable TIPODATOS =  VALOR
```

La declaración-asignación permite declarar una variable local y asignar, en una misma línea, un valor adecuado a la variable; si necesita realizar lo mismo para más variables, entonces puede usar el siguiente formato:

Formato para la declaración-asignación de variables locales:

```
DECLARE  @nombreVariable TIPODATOS =  VALOR,
         @nombreVariable TIPODATOS =  VALOR,
         @nombreVariable TIPODATOS =  VALOR
```

A continuación, algunos casos sobre el uso de variables y asignaciones:

- Script que permita calcular el promedio de cuatro notas de un determinado alumno; dichas notas deberán estar inicializadas con valores predeterminados.

```
*** RESUMEN DE NOTAS ***
-----------------------------------
ESTUDIANTE: FERNANDA TORRES LAZARO
LA NOTA 1: 12
LA NOTA 2: 20
LA NOTA 3: 15
LA NOTA 4: 18
-----------------------------------
EL PROMEDIO ES: 16.25
```

```
Primera forma: Usando DECLARE y SET
--Declaración de variables
DECLARE @NOMBRE VARCHAR(30)
DECLARE @N1 INT,@N2 INT,@N3 INT,@N4 INT,@PROMEDIO DECIMAL(5,2)

--Asignación de valores
SET @NOMBRE = 'FERNANDA TORRES LAZARO'
SET @N1=12
SET @N2=20
SET @N3=15
SET @N4=18

--Calculando el promedio
SET @PROMEDIO=(@N1+@N2+@N3+@N4)/4.0

--Mostrando el resultado
PRINT '*** RESUMEN DE NOTAS ***'
PRINT '-----------------------------------'
PRINT 'ESTUDIANTE: '+@NOMBRE
PRINT 'LA NOTA 1: '+CAST(@N1 AS CHAR(2))
PRINT 'LA NOTA 2: '+CAST(@N2 AS CHAR(2))
PRINT 'LA NOTA 3: '+CAST(@N3 AS CHAR(2))
PRINT 'LA NOTA 4: '+CAST(@N4 AS CHAR(2))
PRINT '-----------------------------------'
PRINT 'EL PROMEDIO ES: '+CAST(@PROMEDIO AS CHAR(5))
GO

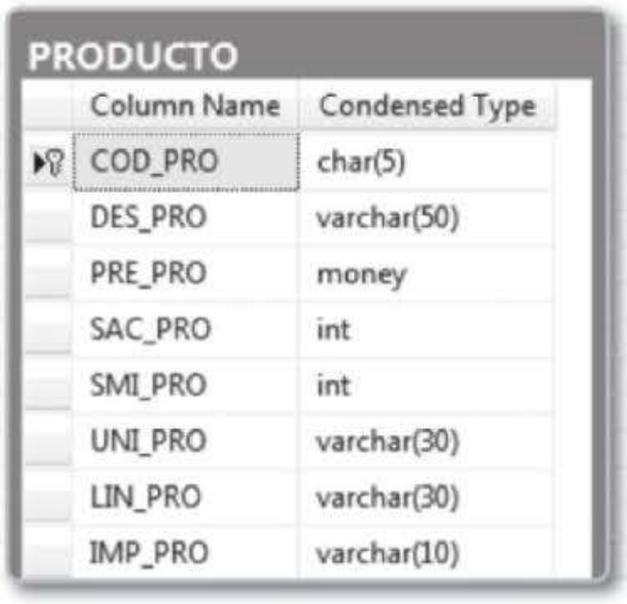
Segunda forma: Usando DECLARE

--Declaración de variables
DECLARE @NOMBRE VARCHAR(30) = 'FERNANDA TORRES LAZARO'
DECLARE @N1 INT=12,@N2 INT=20,@N3 INT=15,@N4 INT=18,@PROMEDIO DECIMAL(5,2)

--Calculando el promedio
SET @PROMEDIO=(@N1+@N2+@N3+@N4)/4.0
```

```
--Mostrando el resultado
PRINT '*** RESUMEN DE NOTAS ***'
PRINT '-----------------------------------'
PRINT 'ESTUDIANTE: '+@NOMBRE
PRINT 'LA NOTA 1: '+CAST(@N1 AS CHAR(2))
PRINT 'LA NOTA 2: '+CAST(@N2 AS CHAR(2))
PRINT 'LA NOTA 3: '+CAST(@N3 AS CHAR(2))
PRINT 'LA NOTA 4: '+CAST(@N4 AS CHAR(2))
PRINT '-------------------------------'
PRINT 'EL PROMEDIO ES: '+CAST(@PROMEDIO AS CHAR(5))
GO
```

- Si se usa la información de la tabla PRODUCTO:



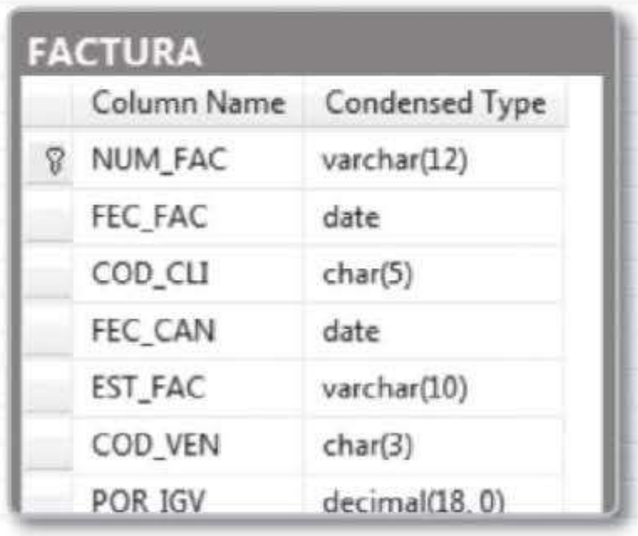
Se hará una consulta que permita mostrar la información de los productos con tipo de unidad MLL, haciendo uso de variables locales.

```
COD_PRO DES_PRO                                           PRE_PRO              UNI_PRO
------- ------------------------------------------------- -------------------- ---------
P001    PAPEL BOND A-4 ATLAS                              18.50                MLL
P002    PAPEL BOND OFICIO                                 37.00                MLL
P003    PAPEL BULKY                                       12.00                MLL
P004    PAPEL PERIODICO 75 GRAMOS                         11.00                MLL
P024    LAPICERO 035 FINE                                 1.00                 MLL
P025    LAPICERO 050 FINE                                 1.50                 MLL
P026    LAPICERO 075 FINE                                 2.00                 MLL
P027    REGLA 20CM                                        2.00                 MLL
P028    REGLA 30CM                                        5.00                 MLL
P029    REGLA FLEXIBLE                                    10.00                MLL

(10 row(s) affected)
```

```
DECLARE @UNI CHAR(3)='MLL'
SELECT P.COD_PRO, P.DES_PRO,P.PRE_PRO, P.UNI_PRO
        FROM PRODUCTO P
        WHERE UNI_PRO = @UNI
GO
```

- Si se usa la información de la tabla FACTURA:



FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

Se hará una consulta que permita mostrar la razón social del cliente asociado a una determinada factura, haciendo uso de variables locales; el resultado se debe mostrar de la siguiente manera:

```
LA FACTURA CON FA005 TIENE COMO CLIENTE A HAYASHI
```

```
DECLARE @FACT CHAR(5)= 'FA005'
DECLARE @CLIENTE VARCHAR(30)
SELECT  @CLIENTE = C.RSO_CLI
        FROM FACTURA F
        JOIN CLIENTE C ON F.COD_CLI=C.COD_CLI
PRINT 'LA FACTURA CON '+@FACT + ' TIENE COMO CLIENTE A '+@CLIENTE
GO
```

5.2.2 Funciones Cast y Convert

El uso de CAST o CONVERT es para pasar de un tipo de datos a otro; normalmente se realiza una conversión cuando una función requiere un tipo especial de datos como parámetro.

Formato:

```
CAST ( EXPRESION - VARIABLE AS TIPO_DATOS)

CONVERT ( TIPO_DATOS , EXPRESION - VARIABLE)
```

A continuación, algunos casos sobre el uso de CAST y CONVERT:

```
Se tiene un monto registrado en la variable local @MONTO de 1250.75
y se necesita mostrarlo al usuario por medio de la función PRINT. A
continuación, se ven cinco implementaciones.

Primera versión:
DECLARE @MONTO MONEY
SET @MONTO = 1250.75
PRINT 'EL MONTO INGRESADO ES: '+@MONTO
GO
```

Al no aplicar conversiones a la variable numérica @MONTO, se muestra el siguiente mensaje desde el servidor de SQL:

```
Msg 235, Level 16, State 0, Line 3
Cannot convert a char value to money.
The char value has incorrect syntax.
```

Segunda versión:

```
DECLARE @MONTO MONEY
SET @MONTO = 1250.75
SELECT @MONTO AS [VALOR DE MONTO]
GO
```

Al enviar la información de la variable a la sentencia SELECT, automáticamente se realiza una conversión implícita y no genera error; el resultado se muestra así:

```
VALOR DE MONTO
---------------------
1250.75

(1 row(s) affected)
```

Tercera versión:

```
DECLARE @MONTO MONEY
SET @MONTO = 1250.75
PRINT 'EL MONTO INGRESADO ES: '+CAST(@MONTO AS VARCHAR(10))
GO
```

La función CAST convierte el valor numérico 1250.75 en un valor de tipo cadena para que pueda concatenarse al texto “El monto ingresado es: ”.

Cuarta versión:

```
DECLARE @MONTO MONEY
SET @MONTO = 1250.75
PRINT 'EL MONTO INGRESADO ES: '+CONVERT(VARCHAR(10),@MONTO)
GO
```

La función CONVERT convierte el valor numérico 1250.75 en un valor de tipo cadena para que pueda concatenarse al texto “El monto ingresado es: ”.

Quinta versión:

```
DECLARE @MONTO MONEY
SET @MONTO = 1250.75
PRINT 'EL MONTO INGRESADO ES: '+STR(@MONTO)
GO
```

La función STR convierte el valor numérico 1250.75 en un valor de tipo cadena, pero adiciona espacios en blanco en el lado izquierdo del resultado y, además, no precisa los decimales. Se muestra de la siguiente manera:

```
EL MONTO INGRESADO ES:       1251
```

5.3 Estructuras de control

Existe un teorema de la estructura, o también llamado teorema de Böhm-Jacopini en honor de Corrado Böhm y Giuseppe Jacopini, donde se establece que toda función computable puede ser implementada en un lenguaje de programación; en este caso Transact SQL cuenta solo con tres estructuras lógicas, también llamadas estructuras de control, las cuales son:

- **Estructuras secuenciales:** Donde se ejecutan las instrucciones una a continuación de la otra.
- **Estructuras selectivas:** Donde se ejecutan las instrucciones según el valor lógico de una variable.
- **Estructuras repetitivas:** Donde se ejecutan las instrucciones en forma repetida, según una condición lógica llamada ciclo o bucle.

En Transact SQL las estructuras de control permiten modificar el flujo de ejecución de las expresiones de un script, es decir, ahora se podrá insertar registros, modificarlos o eliminarlos de acuerdo a una condición. Así, las estructuras de control en Transact SQL pueden:

- de acuerdo a una condición lógica, ejecutar un grupo u otro de sentencias (IF...ELSE);
- de acuerdo al valor de una variable, determinar uno u otro valor dentro de las columnas de una tabla (CASE);
- ejecutar un grupo de sentencias de forma repetida, mientras una condición sea verdadera o falsa (WHILE).

5.3.1 Estructura selectiva IF

La estructura IF evalúa una condición lógica y, en función del resultado, realiza una u otra expresión. Dicha condición se especifica mediante una expresión lógica, donde el resultado puede ser de tipo booleano, es decir, True o False. Presenta los siguientes formatos:

Formato simple:

```
IF <CONDICION_LOGICA>  <expresion_verdadera>

IF <CONDICION_LOGICA>
<BEGIN>
  <expresiones_verdaderas>
<END>
```

Formato doble:

La estructura IF doble permite controlar una expresión lógica, determinando así qué expresión ejecutar cuando la salida de dicha expresión sea True, o qué acciones se realizarán al resultar False.

```
IF <CONDICION_LOGICA>  <expresion_verdadera> else <expresion_falsa>

IF <CONDICION_LOGICA>
<BEGIN>
   <expresiones_verdaderas>
<END>
ELSE
<BEGIN>
   <expresiones_falsas>
<END>
```

Formato doblemente enlazado:

La estructura IF doblemente anidada, normalmente, se usa cuando la expresión lógica devuelve más de un resultado lógico, es decir, trabaja de acuerdo a las alternativas implementadas dentro de la estructura. Es similar a trabajar con CASE.

```
IF <CONDICION_LOGICA>
<BEGIN>
<expresiones_verdaderas>
<END>
ELSE IF <CONDICION_LOGICA>
<BEGIN>
<expresiones_verdaderas>
<END>
ELSE
<BEGIN>
<expresiones_falsas>
<END>
```

Donde:

- CONDICION_LOGICA: Es la expresión que el usuario debe determinar para ejecutar una o más expresiones. Hay que tener en cuenta que aquí es el momento de usar los operadores lógicos (< <= > >= <>) y relacionales (AND OR NOT EXISTS).
- BEGIN: Marca el inicio de las expresiones si la condición resultara True en su evaluación.
- END: Pone fin a las expresiones iniciadas desde el operador BEGIN.

A continuación, algunos casos sobre el uso de IF:

- Si se cuenta con la información de la tabla Distrito:

DISTRITO

	Column Name	Condensed Type
🔑	COD_DIS	char(5)
	NOM_DIS	varchar(50)

Se necesita implementar un código de programación Transact que permita registrar un nuevo distrito a partir de variables locales. Se debe evaluar que el nombre del distrito no exista en la tabla; en caso sea así, mostrar el mensaje "Distrito ya se encuentra registrado"; caso contrario, mostrar "Distrito registrado correctamente".

```
--Declaramos las variables y le asignamos valor
DECLARE @CODIGO CHAR(5)='D38',@NOMBRE VARCHAR(40)='NUEVO SAN JUAN'

--Buscamos la existencia de dicho distrito
IF EXISTS(SELECT * FROM DISTRITO WHERE NOM_DIS=@NOMBRE)
BEGIN
 PRINT 'DISTRITO YA SE ENCUENTRA REGISTRADO'
END
ELSE
BEGIN
 INSERT INTO DISTRITO VALUES(@CODIGO,@NOMBRE)
 PRINT 'DISTRITO REGISTRADO CORRECTAMENTE'
END
GO
```

```
(1 row(s) affected)
DISTRITO REGISTRADO CORRECTAMENTE
```

Si se ejecuta por segunda vez el mismo código, el resultado varía, puesto que el distrito Nuevo San Juan ya se encuentra registrado:

```
DISTRITO YA SE ENCUENTRA REGISTRADO
```

- Si se cuenta con la información de la tabla FACTURA:

FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

Se necesita implementar un código de programación Transact que permite mostrar el total de facturas registradas en un determinado año; si en dicho año no hay facturas registradas, mostrar el mensaje "No hay facturas registradas".

```
DECLARE @AÑO INT =2017
IF (SELECT COUNT(*) FROM FACTURA F
                        WHERE YEAR(F.FEC_FAC)=@AÑO) IS NULL
        PRINT 'NO HAY FACTURAS REGISTRADAS'
        ELSE
        BEGIN
                DECLARE @TOTAL INT
                SELECT @TOTAL=COUNT(*) FROM FACTURA F
                        WHERE YEAR(F.FEC_FAC)=@AÑO

                PRINT 'EN EL AÑO '+CAST(@AÑO AS CHAR(4))+' HAY '+
                   CAST(@TOTAL AS CHAR(2))+ ' FACTURAS REGISTRADAS'
        END
GO
```

En una primera prueba con el año 2017, el resultado se muestra de la siguiente manera:

```
EN EL AÑO 2017 HAY 0  FACTURAS REGISTRADAS
```

Si se cambia el valor de la variable año por 2013, el resultado se muestra de la siguiente manera:

```
EN EL AÑO 2013 HAY 18 FACTURAS REGISTRADAS
```

- Si se cuenta con la información de la tabla PRODUCTO:

PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

Se necesita implementar un código de programación Transact que permite mostrar una condición de acuerdo al total de productos por tipo de unidad. Si la cantidad es menor a 10, mostrar el mensaje "Condición: iniciar reporte"; en caso contrario, mostrar el mensaje "Condición: conforme".

```
DECLARE @UNIDAD CHAR(3)='UNI',@TOTAL INT
SELECT @TOTAL = COUNT(*) FROM PRODUCTO
        GROUP BY UNI_PRO
        HAVING UNI_PRO = @UNIDAD
PRINT 'EL TIPO DE UNIDAD '+@UNIDAD
PRINT 'TIENE UN TOTAL DE '+CAST(@TOTAL AS VARCHAR(5))+' PRODUCTOS'
IF @TOTAL>0 AND @TOTAL<10
                        PRINT 'CONDICION: INICIAR REPORTE'
                ELSE
                        PRINT 'CONDICION: CONFORME'
GO
```

En una primera prueba con el valor MLL para la unidad, el resultado se muestra de la siguiente manera:

```
EL TIPO DE UNIDAD MLL
TIENE UN TOTAL DE 10 PRODUCTOS
CONDICION: CONFORME
```

Si se cambia el valor de la variable unidad por UNI, el resultado se muestra de la siguiente manera:

```
EL TIPO DE UNIDAD UNI
TIENE UN TOTAL DE 7 PRODUCTOS
CONDICION: INICIAR REPORTE
```

5.3.2 Estructura condicional múltiple CASE

Con frecuencia es necesario que existan más de dos posibles acciones en una determinada condición, para esto se usan las estructuras de condicional múltiple. La estructura CASE evalúa una expresión que podrá tomar N valores distintos; según se elija uno de estos valores, se tomará N posibles acciones.

Formato múltiples valores para una sola condición:

```
CASE <CAMPO>
     WHEN <VALOR1> THEN expresion_resultado
     WHEN <VALOR2> THEN expresion_resultado
  ELSE expresion_falsa
END
```

Formato múltiples condiciones:

```
CASE
 WHEN < CONDICION 1>  THEN expresion_resultado
 WHEN < CONDICION 2>  THEN expresion_resultado
 WHEN < CONDICION 3>  THEN expresion_resultado
 ELSE expresion_falsa
END
```

Donde:

- **CAMPO:** Es el nombre de la columna que se va a comparar; también se pueden usar funciones SQL.
- **WHEN:** Especifica las expresiones que se van a buscar según la condición.
- **THEN:** Especifica las expresiones resultantes de una opción.

- **ELSE:** Determina qué acción se tomará cuando no se cumpla con ninguna de las condiciones u opciones.
- **END:** Especifica la finalización de la estructura CASE.

A continuación, algunos casos sobre el uso de CASE:

- Si se tiene información de la tabla PRODUCTO:



PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

Se necesita listar el código, nombre del vendedor, sueldo, fecha de inicio y fecha de inicio en letras:

```
MLL > MILLARES
UNI > UNIDADES
DOC > DOCEN
CIE > CIENTO
```

```
CODIGO DESCRIPCION                                    PRECIO               UNIDAD                           DESCRIPCION UNIDAD
------ ---------------------------------------------- -------------------- -------------------------------- ------------------
P001   PAPEL BOND A-4 ATLAS                           18.50                MLL                              MILLARES
P002   PAPEL BOND OFICIO                              37.00                MLL                              MILLARES
P003   PAPEL BULKY                                    12.00                MLL                              MILLARES
P004   PAPEL PERIODICO 75 GRAMOS                      11.00                MLL                              MILLARES
P005   CARTUCHO TINTA NEGRA                           21.00                UNI                              UNIDADES
P006   CARTUCHO TINTA COLOR                           47.00                UNI                              UNIDADES
P007   PORTA DISKETTES                                5.50                 UNI                              UNIDADES
P008   CAJA DE DISKETTES * 10                         32.00                UNI                              UNIDADES
P009   BORRADOR DE TINTA                              12.00                DOC                              DOCENAS
P010   BORRADOR BLANCO                                5.00                 DOC                              DOCENAS
P011   TAJADOR METAL                                  22.00                DOC                              DOCENAS
P012   TAJADOR PLASTICO                               14.00                DOC                              DOCENAS
P013   FOLDER MANILA OFICIO                           22.00                CIE                              CIENTO
P014   FOLDER MANILA A-4                              22.00                CIE                              CIENTO
P015   SOBRE MANILA OFICIO                            17.00                CIE                              CIENTO
P016   SOBRE MANILA A-4                               20.00                CIE                              CIENTO
```

```
SELECT P.COD_PRO AS CODIGO,
       P.DES_PRO AS DESCRIPCION,
       P.PRE_PRO AS PRECIO,
       P.UNI_PRO AS UNIDAD,
       CASE P.UNI_PRO
            WHEN 'MLL' THEN 'MILLARES'
            WHEN 'UNI' THEN 'UNIDADES'
            WHEN 'DOC' THEN 'DOCENAS'
            WHEN 'CIE' THEN 'CIENTO'
       END AS [DESCRIPCION UNIDAD]
       FROM PRODUCTO P
GO
```

- Si se tiene información de la tabla VENDEDOR:



VENDEDOR

	Column Name	Condensed Type
PK	COD_VEN	char(3)
	NOM_VEN	varchar(20)
	APE_VEN	varchar(20)
	SUE_VEN	money
	FIN_VEN	date
	TIP_VEN	varchar(10)
	COD_DIS	char(5)

Se necesita listar el código, el nombre del vendedor, el sueldo, la fecha de inicio y la fecha de inicio en letras:

```
CODIGO VENDEDOR                                SUELDO               FECHA INICIO FECHA
------ --------------------------------------- -------------------- ------------ -----------------
V01    JUANA ALVA                              1000.00              2003-07-12   12 JULIO 2003
V02    JUAN SOTO                               1200.00              2002-10-03   3  OCTUBRE 2002
V03    CARLOS AREVALO                          1500.00              1992-02-10   10 FEBRERO 1992
V04    CESAR OJEDA                             850.00               1999-08-11   11 AGOSTO 1999
V05    JULIO VEGA                              1500.00              1997-11-05   5  NOVIEMBRE 1997
V06    ANA ORTEGA                              1200.00              1994-11-05   5  NOVIEMBRE 1994
V07    JOSE PALACIOS                           2500.00              1997-12-02   2  DICIEMBRE 1997
V08    RUBEN GOMEZ                             1450.00              1996-12-08   8  DICIEMBRE 1996
V09    PATRICIA ARCE                           1800.00              1997-12-05   5  DICIEMBRE 1997
V10    RENATO PEREZ                            1550.00              1998-11-08   8  NOVIEMBRE 1998

(10 row(s) affected)
```

```
SELECT V.COD_VEN AS CODIGO,
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       V.SUE_VEN AS SUELDO,
       V.FIN_VEN AS [FECHA INICIO],
       CAST(DAY(V.FIN_VEN) AS CHAR(2))+
       CASE MONTH(V.FIN_VEN)
              WHEN 1 THEN ' ENERO '
              WHEN 2 THEN ' FEBRERO '
              WHEN 3 THEN ' MARZO '
              WHEN 4 THEN ' ABRIL '
              WHEN 5 THEN ' MAYO '
              WHEN 6 THEN ' JUNIO '
              WHEN 7 THEN ' JULIO '
              WHEN 8 THEN ' AGOSTO '
              WHEN 9 THEN ' SETIEMBRE '
              WHEN 10 THEN ' OCTUBRE '
              WHEN 11 THEN ' NOVIEMBRE '
              WHEN 12 THEN ' DICIEMBRE '

       END + CAST(YEAR(V.FIN_VEN) AS CHAR(4)) AS [FECHA]
       FROM VENDEDOR V
GO
```

- Si se tiene información de la tabla PROVEEDOR:

PROVEEDOR

	Nombre de columna	Tipo comprimido
🔑	COD_PRV	char(5)
	RSO_PRV	varchar(80)
	DIR_PRV	varchar(100)
	TEL_PRV	char(15)
	COD_DIS	char(5)
	REP_PRV	varchar(80)

Se necesita listar el total de proveedores por distrito, adicionando el mensaje "No cuenta" solo si la cantidad de proveedores por distrito es cero:

```
DISTRITO                                           TOTAL PROVEEDORES MENSAJE
-------------------------------------------------- ----------------- ---------
ATE - VITARTE                                      2
BARRANCO                                           1
BELLAVISTA                                         1
BREÑA                                              1
CALLAO                                             1
CARABAYLLO                                         0                 NO CUENTA
CARMEN DE LA LEGUA                                 1
CHORRILLOS                                         1
COMAS                                              0                 NO CUENTA
EL AGUSTINO                                        0                 NO CUENTA
INDEPENDENCIA                                      0                 NO CUENTA
JESUS MARIA                                        1
LA MOLINA                                          0                 NO CUENTA
LA PERLA                                           0                 NO CUENTA
LIMA - CERCADO                                     0                 NO CUENTA
LINCE                                              1
LOS OLIVOS                                         0                 NO CUENTA
MAGDALENA                                          0                 NO CUENTA
MIRAFLORES                                         1
NUEVO SAN JUAN                                     0                 NO CUENTA
PUEBLO LIBRE                                       0                 NO CUENTA
PUENTE PIEDRA                                      0                 NO CUENTA
```

```
SELECT  D.NOM_DIS AS DISTRITO,
                COUNT(P.COD_DIS) AS [TOTAL PROVEEDORES],
                CASE
                        WHEN COUNT(P.COD_DIS)=0 THEN 'NO CUENTA'
                        ELSE ''
                END AS [MENSAJE]
                FROM DISTRITO D
                LEFT JOIN PROVEEDOR P ON P.COD_DIS=D.COD_DIS
                GROUP BY D.NOM_DIS
GO
```

5.3.3 Estructura de control While

Desde que la computadora fue diseñada siempre tuvo como visión hacer que las aplicaciones reduzcan notablemente las líneas de código, haciendo inclusive que aquellas instrucciones que solían ser repetidas se tratasen dentro de un control de flujo repetitivo. Por esta razón, se implementa la estructura de control repetitiva como WHILE en Transact SQL.

Al respecto, se puede decir que la sentencia While hace repetir un conjunto de instrucciones en secuencia un número determinado de veces; a esto se le denomina bucle o ciclo. Si se desea acumular sumas o determinar el mayor o menor elemento de un conjunto de valores, entonces se está frente a un caso de repeticiones que puede ser controlado y administrado por la estructura repetitiva WHILE.

Por otro lado, While establece una condición para la ejecución repetida de una instrucción o bloque de instrucciones SQL. Las instrucciones se ejecutan repetidamente, siempre que la condición especificada sea verdadera. Se puede controlar la ejecución de instrucciones en el bucle WHILE con las palabras clave BREAK y CONTINUE.

Formato:

```
WHILE <condición>
     <expresion_repetida>
     [ BREAK ]
     <expresion_repetida>
     [ CONTINUE ]
     <expresion_repetida>
```

Donde:

- **Condición:** Se aplicará una evaluación lógica sobre dicha condición para poder determinar cuántos ciclos darán las expresiones colocadas dentro del WHILE. Hay que considerar que es posible usar los operadores lógicos y relaciones de SQL Server.
- **BREAK:** Permite cortar el ciclo de repeticiones y, según la secuencia de instrucciones, seguir con la aplicación.
- **CONTINUE:** Permite regenerar el ciclo de repeticiones no tomando en cuenta las instrucciones asignadas después del operador CONTINUE.
- Es mejor plantear la estructura WHILE con los operadores de inicio y fin (BEGIN END) para tener un mejor control de las expresiones internas del ciclo, además de poder usar las estructuras vistas anteriormente como IF o CASE.

A continuación, algunos casos sobre el uso de WHILE:

- Mostrar los diez primeros números usando la estructura WHILE:

```
VALOR I: 1
VALOR I: 2
VALOR I: 3
VALOR I: 4
VALOR I: 5
VALOR I: 6
VALOR I: 7
VALOR I: 8
VALOR I: 9
VALOR I: 10
```

```
--Declarar la variable
DECLARE @N INT=0

--Ciclo de repeticiones
WHILE (@N<10)
BEGIN
 SET @N+=1
 PRINT 'VALOR I: '+CAST(@N AS CHAR(2))
END
GO
```

- Si se tiene información de la tabla FACTURA:

FACTURA

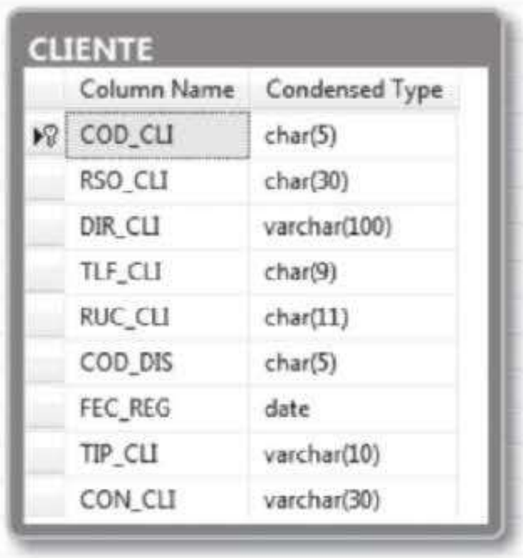
	Column Name	Condensed Type
PK	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

Mostrar los tres últimos registros de facturas.

```
--Declaración de variables
DECLARE @N INT=1,@TOPE INT=3

--Ciclo de repeticiones
WHILE @N<=100
BEGIN
                IF @N=@TOPE
                BEGIN
                   SELECT TOP(@N) * FROM FACTURA f ORDER BY f.NUM_FAC  DESC
                   BREAK
                END
                ELSE
                BEGIN
                        SET @N+=1
                        CONTINUE
                END
END
```

- Si se tiene información de la tabla CLIENTE:



Implementar un script que permita mostrar cierta cantidad de registro de los clientes usando valores iniciales y finales.

```
NUMERO    COD_CLI RSO_CLI      DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
--------- ------- ------------ -------------------------- --------- ----------- ------- ---------- ---------- --------------------
3         C003    SERVIEMSA    JR. COLLAGATE 522          75012403  NULL        D05     2011-06-03 2          CHRISTIAN LAGUNA
4         C004    ISSA         CALLE LOS AVIADORES 263    3725910   46720159    D01     2012-09-12 1          LUIS APUMAYTA
5         C005    MASS         AV. TOMAS MARSANO 880      4446177   83175942    D14     2013-10-01 1          KATIA ARMEJO

(3 row(s) affected)
```

```
DECLARE @VI INT=3,@VF INT=5
SELECT *
       FROM (SELECT ROW_NUMBER()
       OVER (ORDER BY C.COD_CLI) AS [NUMERO],* FROM CLIENTE C) X
       WHERE NUMERO BETWEEN @VI AND @VF
GO
```

Implementar un script que permita mostrar el registro de los clientes en forma de paginación de cinco en cinco.

```
ROWNUM COD_CLI RSO_CLI      DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------ ------- ------------ -------------------------- --------- ----------- ------- ---------- ---------- --------------------
1      C001    FINSETH      AV. LOS VIÑEDOS 150        4342318   48632081    D05     2008-12-10 1          ALICIA BARRETO
2      C002    ORBI         AV. EMILIO CAVENECIA 225   4406335   57031642    D04     2013-02-01 2          ALFONSO BELTRAN
3      C003    SERVIEMSA    JR. COLLAGATE 522          75012403  NULL        D05     2011-06-03 2          CHRISTIAN LAGUNA
4      C004    ISSA         CALLE LOS AVIADORES 263    3725910   46720159    D01     2012-09-12 1          LUIS APUMAYTA
5      C005    MASS         AV. TOMAS MARSANO 880      4446177   83175942    D14     2013-10-01 1          KATIA ARMEJO

(5 row(s) affected)

ROWNUM COD_CLI RSO_CLI      DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------ ------- ------------ -------------------------- --------- ----------- ------- ---------- ---------- --------------------
6      C006    BERKER       AV. LOS PROCERES 521       3810322   54890124    D05     2013-07-05 1          JUDITH ASTE
7      C007    FIDENZA      JR. EL NIQUEL 282          5289034   16204790    D20     2012-10-02 2          HECTOR VIVANCO
8      C008    INTECH       AV. SAN LUIS 2619  5TO P   2249493   34021824    D09     2011-01-07 2          CARLOS VILLANUEVA
9      C009    PROMINENT    JR. IQUIQUE 132            43233519  NULL        D11     2010-06-11 1          JORGE VALDIVIA
10     C010    LANDU        AV.NICOLAS DE AYLLON 1453  3267840   30405261    D05     2013-11-06 2          RAQUEL ESPINOZA

(5 row(s) affected)
```

```
DECLARE @PAGINA INT=1
WHILE(@PAGINA<=5)
BEGIN
               DECLARE @CANT_DEFILAS INT = 5
               SELECT * FROM (       SELECT ROWNUM = ROW_NUMBER()
                                            OVER (ORDER BY C.COD_CLI), *
                                            FROM CLIENTE C) X
                  WHERE
                  ROWNUM > (@CANT_DEFILAS * (@PAGINA - 1)) AND
                 ROWNUM <= (@CANT_DEFILAS * (@PAGINA - 1)) + @CANT_DEFILAS
               SET @PAGINA+=1
END
GO
```

5.4 Control de errores en Transact SQL

Los lenguajes de programación permiten controlar los errores que puedan generarse a nivel de código. SQL Server incorpora diferentes alternativas para controlar dichos errores, encargando esta tarea no al lenguaje de programación, sino al Transact SQL, desligándose un poco de las validaciones propias del SQL. A partir de la versión 2005 del SQL Server, se incorpora el bloque TRY-CATCH que permite gestionar los errores de manera adecuada.

Formato:

```
BEGIN TRY
     <Expresión_Sql>
END TRY
BEGIN CATCH
     <Expresion_Sql>
END CATCH
```

Donde:

- **BEGIN TRY:** Precisa el inicio del control de error en Transact SQL; en este bloque se coloca el script de trascendencia normal.
- **END TRY:** Indica la finalización del BEGIN TRY.
- **BEGIN CATCH:** Precisa el inicio del lado excepcional del script, es decir, el bloque BEGIN TRY le encomendará el trabajo a BEGIN CATCH al generarse cualquier tipo de error. Aquí entran a relucir las funciones especiales de errores que se verán a continuación.
- **END CATCH:** Indica la finalización del bloque BEGIN CATCH.

Consideraciones:

- El bloque TRY-CATCH detecta todos los errores de ejecución que tienen una gravedad mayor de 10 y que no cierran la conexión con la base de datos en SQL Server.
- La implementación de un bloque TRY-CATCH no puede abarcar varios bloques del mismo, es decir, no se podrá incluir dentro de un TRY-CATCH otro TRY-CATCH, ni mucho menos dentro de un bloque IF...ELSE, ya que generaría un error desde el Motor de base de datos de SQL Server. Pero dentro del bloque BEGIN CATCH sí se puede implementar un BEGIN TRY.
- La forma de trabajo del bloque TRY-CATCH consiste en que, si no hay errores en el script incluido dentro de un bloque BEGIN TRY, las sentencias se ejecutan inmediatamente posterior a la instrucción. En caso hubiera un error en el script incluido en un bloque BEGIN TRY, el control se transfiere a la primera instrucción del bloque BEGIN CATCH.
- Toda vez que la instrucción END CATCH sea la última de un *store procedure* o trigger, el control se devuelve a la instrucción que llamó al *store procedure* o activó el trigger.
- Los errores capturados por un bloque BEGIN CATCH no devuelven valor alguno a la aplicación que realiza la llamada. En caso necesite devolver alguna información, tendrá que implementarse funciones como RAISERROR o PRINT.
- La implementación de un bloque TRY...CATCH no se puede utilizar en una función definida por el usuario.

Funciones especiales de error:

- ERROR_NUMBER(): Devuelve el número de error detectado.
- ERROR_MESSAGE(): Devuelve el texto completo del mensaje de error. El texto incluye los valores suministrados para los parámetros sustituibles, como longitudes, nombres de objeto u horas.
- ERROR_SEVERITY(): Devuelve la gravedad del error.
- ERROR_STATE(): Devuelve el número de estado del error.
- ERROR_LINE(): Devuelve el número de línea dentro de la rutina en que se produjo el error.
- ERROR_PROCEDURE(): Devuelve el nombre del procedimiento almacenado o desencadenador en que se produjo el error.

A continuación, algunos casos sobre el uso del control de errores:

- Implementar un script que permita mostrar los valores que emiten las funciones de control de errores a partir de una división por cero.

```
(0 row(s) affected)

ErrorNumber ErrorSeverity ErrorState  ErrorProcedure    ErrorLine   ErrorMessage
----------- ------------- ----------- ------------------------------------------------------------------
8134        16            1           NULL              2           Divide by zero error encountered.

(1 row(s) affected)
```

```
BEGIN TRY
    SELECT 1/0
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage
END CATCH
GO
```

• Si se tiene información de la tabla DISTRITO:

DISTRITO

	Column Name	Condensed Type
🔑	COD_DIS	char(5)
	NOM_DIS	varchar(50)

Implementar un script que permita registrar un nuevo distrito; en caso ocurriera algún error en el registro, mostrar un mensaje.

```
BEGIN TRY
	DECLARE @COD CHAR(3)='D01',
		@DIS VARCHAR(40)='SAN JUAN DE ASIS'

	INSERT INTO DISTRITO VALUES(@COD,@DIS)
END TRY
BEGIN CATCH
	PRINT 'ERROR AL REGISTRAR EN LA TABLA DISTRITO'
END CATCH
```

El código anterior emite un error, dado que el código D01 ya se encuentra registrado en la tabla DISTRITO.

```
(0 row(s) affected)
ERROR AL REGISTRAR EN LA TABLA DISTRITO
```

Cuando no se controlan los errores, el servidor responde con un formato preestablecido; por ejemplo:

```
Msg 2627, Level 14, State 1, Line 4
Violation of PRIMARY KEY constraint 'PK__DISTRITO__2B2C6E897F80A102'.
Cannot insert duplicate key in object 'dbo.DISTRITO'. The duplicate key value is (D01  ).
The statement has been terminated.
```

Si en el script anterior se añade las funciones de control de error, se podría tener un informe de la siguiente manera:

```
(0 row(s) affected)
ERROR EN LA TABLA PASAJERO
Numero de Error     :2627
Numero de Severidad :14
Numero de Estado        :1
Linea de Error N°   :5
Mensaje de Error        :Violation of PRIMARY KEY constraint 'PK__DISTRITO__2B2C6E897F80A102'.
Cannot insert duplicate key in object 'dbo.DISTRITO'. The duplicate key value is (D01  ).
```

```
BEGIN TRY
	DECLARE @COD CHAR(3)='D01',
		@DIS VARCHAR(40)='SAN JUAN DE ASIS'

	INSERT INTO DISTRITO VALUES(@COD,@DIS)
END TRY
BEGIN CATCH
 PRINT 'ERROR EN LA TABLA PASAJERO'
 PRINT 'Numero de Error       :'+CAST(ERROR_NUMBER() AS VARCHAR(10))
 PRINT 'Numero de Severidad   :'+CAST(ERROR_SEVERITY() AS VARCHAR(10))
 PRINT 'Numero de Estado      :'+CAST(ERROR_STATE() AS VARCHAR(10))
 PRINT 'Linea de Error N°     :'+CAST(ERROR_LINE() AS VARCHAR(10))
 PRINT 'Mensaje de Error             :'+ERROR_MESSAGE()
END CATCH
```

5.5 Función @@error

La función del sistema @@ERROR devuelve 0 si la última sentencia Transact-SQL se ejecutó con éxito; si la sentencia causó un error, @@ERROR devuelve el número de error. Hay que tener en cuenta que el valor de @@ERROR cambia al finalizar cada sentencia Transact-SQL.

En vista de que @@ERROR obtiene un nuevo valor cuando se completa cada sentencia Transact-SQL, entonces se deberá considerar las siguientes opciones:

- Use la función @@ERROR después de una instrucción Transact-SQL.
- Guarde @@ERROR en una variable local de tipo entero, después de que se complete la instrucción Transact-SQL. El valor de la variable se puede usar inmediatamente después de la asignación.
- Para mostrar el número de error, SQL presenta una lista de posibles errores usando la siguiente sentencia: SELECT * FROM SYS.SYSMESSAGES

```
error       severity dlevel description
----------- -------- ------ ------------------------------------------------------------
21          20       0      Warning: Fatal error %d occurred at %S_DATE. Note the error and
101         15       0      Query not allowed in Waitfor.
102         15       0      Incorrect syntax near '%.*ls'.
103         15       0      The %S_MSG that starts with '%.*ls' is too long. Maximum length
104         15       0      ORDER BY items must appear in the select list if the statement
105         15       0      Unclosed quotation mark after the character string '%.*ls'.
106         16       0      Too many table names in the query. The maximum allowable is %d.
```

Consideraciones:

- @@ERROR debe comprobarse o guardarse después de cada instrucción Transact-SQL porque un programador no puede predecir qué instrucción puede generar un error. Esto dobla el número de instrucciones Transact-SQL que deben codificarse para implementar un fragmento de lógica dado.
- Las construcciones TRY...CATCH son mucho más simples. Un bloque de instrucciones Transact-SQL está delimitado por instrucciones BEGIN TRY y END TRY; después se escribe un bloque CATCH para gestionar errores que pueden generarse mediante el bloque de instrucciones.
- Fuera de un bloque CATCH, @@ERROR es la única parte de un error de Database Engine (Motor de base de datos) disponible en el lote, procedimiento almacenado o desencadenador que ha generado el error. El resto de las partes del error, como su gravedad, estado y texto del mensaje que contienen cadenas de sustitución (nombres de objeto, por ejemplo) solo se devuelven a la aplicación, donde pueden procesarse mediante los mecanismos de control de errores de las API. Si el error invoca un bloque CATCH, se pueden usar las funciones del sistema ERROR_LINE, ERROR_MESSAGE, ERROR_PROCEDURE, ERROR_NUMBER, ERROR_SEVERITY y ERROR_STATE.

A continuación, algunos casos sobre el uso del control de errores usando TRY CATCH:

Si se tiene información de la tabla VENDEDOR:

VENDEDOR

Column Name	Condensed Type
COD_VEN	char(3)
NOM_VEN	varchar(20)
APE_VEN	varchar(20)
SUE_VEN	money
FIN_VEN	date
TIP_VEN	varchar(10)
COD_DIS	char(5)

Implementar un script que permita actualizar el código del distrito de un determinado vendedor.

```
Msg 547, Level 16, State 0, Line 1
The UPDATE statement conflicted with the FOREIGN KEY constraint "FK__VENDEDOR__COD_DI__1273C1CD".
The conflict occurred in database "BDVENTAS", table "dbo.DISTRITO", column 'COD_DIS'.
The statement has been terminated.
OCURRIO UN ERROR AL ACTUALIZAR LA TABLA VENDEDOR
```

```
UPDATE VENDEDOR
                SET COD_DIS='D60'
                WHERE COD_VEN='V01'
IF @@ERROR<>0
                PRINT 'OCURRIO UN ERROR AL ACTUALIZAR LA TABLA VENDEDOR'
GO
```

Para poder mejorar el manejo del error sobre la actualización de un vendedor, se puede usar el siguiente código:

```
(0 row(s) affected)
OCURRIO UN ERROR AL ACTUALIZAR LA TABLA VENDEDOR
```

```
BEGIN TRY
                UPDATE VENDEDOR
                        SET COD_DIS='D60'
                        WHERE COD_VEN='V01'
END TRY
BEGIN CATCH
        IF @@ERROR<>0
                PRINT 'OCURRIO UN ERROR AL ACTUALIZAR LA TABLA VENDEDOR'
END CATCH
GO
```

5.6 Función Raiserror

Se usa para devolver mensajes definidos por el usuario a las aplicaciones con el mismo formato que un error del sistema o un mensaje de advertencia generado por el Motor de base de datos de SQL Server.

Formato:

```
RAISERROR <msg_id | msg_str | @local_variable,severity ,state>
```

Donde:

- **msg_id:** Es un número de mensaje de error definido por el usuario, almacenado en el catálogo SYS.MESSAGES mediante el procedimiento SP_ADDMESSAGE. Los números de los mensajes de error definidos por el usuario deben ser mayores que 50 000, ya que este número es el tope de los números asignados en forma nativa en el Motor de base de datos de SQL. Si no se especifica msg_id, RAISERROR genera un mensaje de error con el número 50 000.
- **msg_str:** Si se especifica msg_str, RAISERROR genera un mensaje de error con el número 50 000. msg_str es una cadena de caracteres que incluye especificaciones de conversión opcionales. Cada especificación de conversión define cómo se aplica formato a un valor de la lista de argumentos y cómo se coloca un campo en la posición indicada dentro de la especificación de conversión. La especificación de conversión tiene el formato siguiente:

```
% [[flag] [width] [. precision] [h | l]] type
```

- **severity:** Es el nivel de gravedad definido por el usuario asociado a este mensaje. Cuando se utiliza msg_id para generar un mensaje definido por el usuario, creado mediante sp_addmessage, la gravedad especificada en RAISERROR reemplaza la gravedad especificada en sp_addmessage.

  Todos los usuarios pueden especificar los niveles de gravedad del 0 al 18. Solo los miembros de la función fija de servidor sysadmin o los usuarios con permisos ALTER TRACE pueden especificar los niveles de gravedad del 19 al 25. Para los niveles de gravedad del 19 al 25, se necesita la opción WITH LOG.

- **state:** Es un número entero entre 0 y 255. Los valores negativos o mayores que 255 generan un error.

Consideraciones:

- RAISERROR permite asignar un número de error, gravedad y un estado específico.
- Puede solicitar que el error se guarde en el registro de errores del Motor de base de datos y en el registro de aplicación de Microsoft Windows.
- Puede sustituir valores de argumento en el texto del mensaje, de forma parecida a la función PRINT. Tanto RAISERROR como PRINT se pueden usar para devolver mensajes de información o advertencia a una aplicación.

- Cuando RAISERROR se usa con el msg_id de un mensaje definido por el usuario en sys.messages, msg_id se devuelve como el número de error de SQL Server o el código nativo del error. Cuando RAISERROR se usa con una msg_str en lugar de un msg_id, el número de error y el número nativo del error de SQL Server devuelto es 50 000.
- Cuando use RAISERROR para devolver un mensaje de error definido por el usuario, use un número de estado distinto en cada RAISERROR que haga referencia al error. Esto puede ayudar a diagnosticar los errores cuando se generan.

A continuación, algunos casos sobre el uso del control de errores usando RAISERROR:

Implementar un script que permita controlar el error de una división irreal.

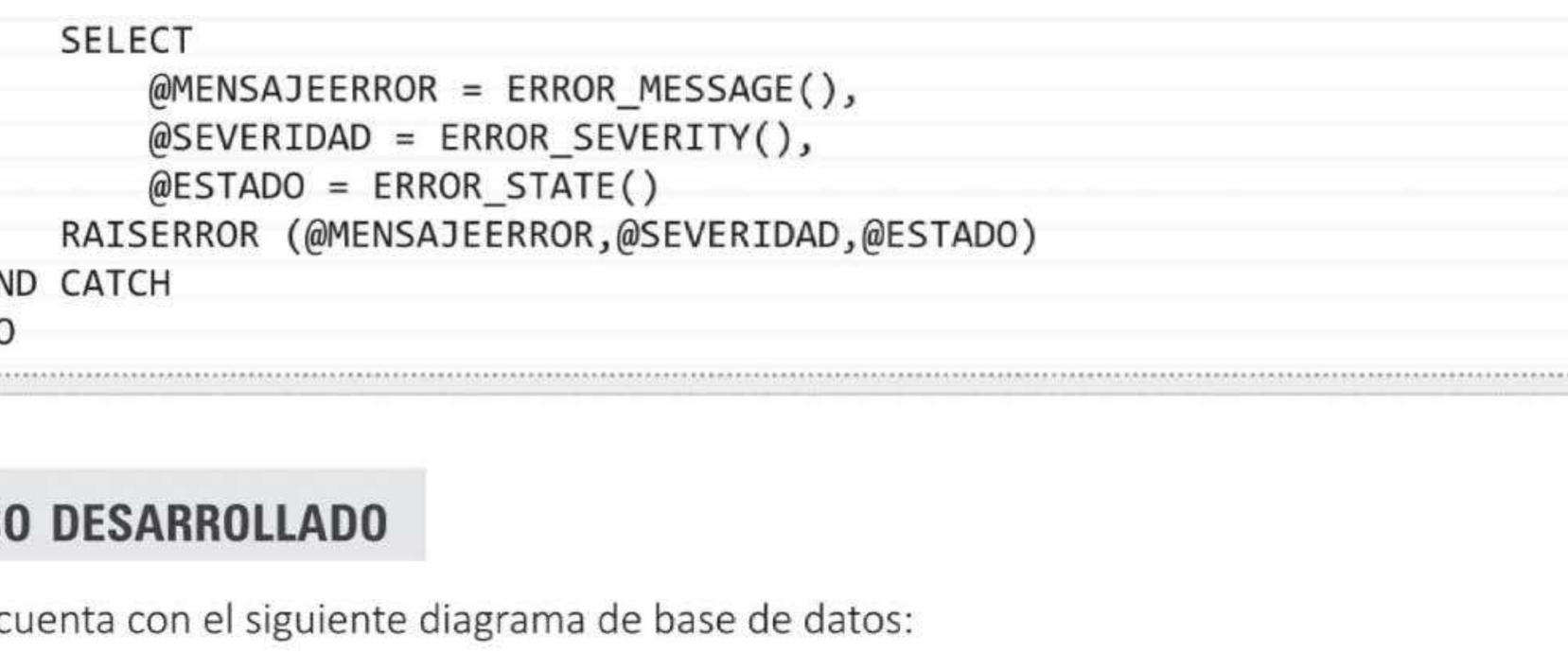
```
Msg 50000, Level 16, State 1, Line 13
Divide by zero error encountered.
```

```
BEGIN TRY
    PRINT 1/0
END TRY
BEGIN CATCH
    DECLARE @MENSAJEERROR VARCHAR(4000)
    DECLARE @SEVERIDAD INT
    DECLARE @ESTADO INT

    SELECT
        @MENSAJEERROR = ERROR_MESSAGE(),
        @SEVERIDAD = ERROR_SEVERITY(),
        @ESTADO = ERROR_STATE()
    RAISERROR (@MENSAJEERROR,@SEVERIDAD,@ESTADO)
END CATCH
GO
```

Caso desarrollado

Si se cuenta con el siguiente diagrama de base de datos:



Y con la siguiente información por tablas:

SELECT * FROM CLIENTE

```
IDE_CLI APE_CLI    NOM_CLI      DNI_CLI  TEL_CLI     IDE_DIS COR_CLI
------- ---------- ------------ -------- ----------- ------- --------
CL001   LOPEZ      MARIA        10856985 526-5254    L03
CL002   GOMEZ      JOSE         02188525 585-8212    L02
CL003   ACOSTA     JESUS        07415482 485-1598    L01
CL004   ARIAS      GUADSLUPE    08995255 258-8569    L04
CL005   CARLOS     MANUEL       10418525 296-9685    L08
CL006   GERZ       BRIGITTE     10952155 585-1254    L06
CL007   BARBOSA    MARISOL      78514855 584-5869    L04

(7 row(s) affected)
```

SELECT * FROM DISTRITO

```
IDE_DIS DES_DIS
------- --------------------
L01     LIMA
L02     LINCE
L03     BREÑA
L04     LOS OLIVOS
L05     RIMAC
L06     SAN MARTIN DE PORRES
L07     SAN LUIS
L08     SAN BORJA

(8 row(s) affected)
```

SELECT * FROM AUTOMOVIL

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- ---------------
AF-456     ROJO                             SEDAN
D4-243     NEGRO                            SEDAN
FD-463     PLATA                            STATION WAGON
FG-654     PLATA                            L10
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
HY-345     PLATA                            CHERY
SF-112     ROJO                             LI
YT-457     NEGRO                            STATION WAGON

(10 row(s) affected)
```

SELECT * FROM ALQUILER

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
1           2010-03-12 185.00
2           2011-01-02 100.00
3           2011-05-01 54.00
4           2012-08-11 85.00
5           2012-06-22 18.00
6           2012-01-30 15.00
7           2012-04-16 165.00
8           2012-05-23 135.00
9           2012-03-21 225.00

(9 row(s) affected)
```

SELECT * FROM DETALLEALQUILER

```
NUM_ALQ     IDE_CLI MAT_AUT
----------- ------- ----------
1           CL002   GG-654
2           CL002   GT-532
3           CL006   FD-463
3           CL007   GT-532
4           CL005   H5-455
5           CL004   SF-112
6           CL004   YT-457
7           CL003   GT-532

(8 row(s) affected)
```

Implemente las siguientes consultas:

- Mostrar la información de un determinado cliente a través de variables y el uso de PRINT.
- Mostrar la información del detalle de alquiler a través de variables y el uso de PRINT.
- Mostrar el nombre completo del cliente a partir de su código, a través de variables y el uso de PRINT.
- Mostrar una lista de automóviles de un determinado color, dicho valor debe ser ingresado por una variable.
- Script que permita registrar un nuevo automóvil; en caso exista, solo debe actualizarlo.
- Script que permita registrar un nuevo cliente validando la existencia del nombre del distrito. En caso el distrito exista, los datos del cliente se registrarán; caso contrario, mostrar el mensaje "Código de distrito no existe".
- Mostrar el siguiente listado de los alquileres, adicionando una columna que tome en cuenta el siguiente criterio:

MONTO ALQUILER	CRITERIO
MENOR A 50	BAJO
ENTRE 50 Y 100	MEDIO
MAYOR A 100	ALTO

Solución:

```
--•     Mostrar la información de un determinado cliente a través de variables
y el uso de PRINT.
```

```
 ** DATOS DEL CLIENTE **
-----------------------------------
CODIGO: CL002
-----------------------------------
APELLIDOS: GOMEZ
NOMBRES: GOMEZ
DNI: 02188525
TELEFONO: 585-8212
DISTRITO: L02
CORREO:
```

```
DECLARE @COD CHAR(5),@APE VARCHAR(30),@NOM VARCHAR(30),@DNI CHAR(8),@TEL
VARCHAR(25),@DIS CHAR(3),@COR VARCHAR(50)
SET @COD = 'CL002'
SELECT  @APE=C.APE_CLI, @NOM=C.APE_CLI, @DNI=C.DNI_CLI,
              @TEL=C.TEL_CLI, @DIS=C.IDE_DIS, @COR=C.COR_CLI
        FROM CLIENTE C
        WHERE C.IDE_CLI=@COD

PRINT ' ** DATOS DEL CLIENTE ** '
PRINT '-----------------------------------'
PRINT 'CODIGO: '+@COD
PRINT '-----------------------------------'
PRINT 'APELLIDOS: '+@APE
PRINT 'NOMBRES: '+@NOM
PRINT 'DNI: '+@DNI
PRINT 'TELEFONO: '+@TEL
PRINT 'DISTRITO: '+@DIS
PRINT 'CORREO: '+@COR
GO
```

```
--•     Mostrar la información del detalle de alquiler a través de variables
y el uso de PRINT.
```

```
ALQUILER    CLIENTE                                                    MATRICULA
----------- ---------------------------------------------------------- ----------
1           JOSE GOMEZ                                                 GG-654

(1 row(s) affected)
```

```
DECLARE @NUM INT=1
SELECT D.NUM_ALQ AS ALQUILER,
       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
       D.MAT_AUT AS MATRICULA
       FROM DETALLEALQUILER D
       JOIN CLIENTE C ON D.IDE_CLI=C.IDE_CLI
       WHERE D.NUM_ALQ=@NUM
GO
```

--• Mostrar el nombre completo del cliente a partir de su código a través de variables y el uso de PRINT.

```
CLIENTE
----------------------------------------
MARIA LOPEZ

(1 row(s) affected)

EL CODIGO CL001 PERTENECE AL CLIENTE MARIA LOPEZ
```

```
DECLARE @COD CHAR(5) = 'CL001'
SELECT C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE
       FROM CLIENTE C
       WHERE C.IDE_CLI=@COD
GO

DECLARE @COD CHAR(5) = 'CL001'
DECLARE @CLIENTE VARCHAR(40)
SELECT @CLIENTE = C.NOM_CLI+SPACE(1)+C.APE_CLI
       FROM CLIENTE C
       WHERE C.IDE_CLI=@COD

PRINT 'EL CODIGO '+@COD + ' PERTENECE AL CLIENTE '+ @CLIENTE
GO
```

--• Mostrar una lista de automóviles de un determinado color, dicho valor debe ser ingresado por una variable.

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- ---------------
AF-456     ROJO                             SEDAN
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
SF-112     ROJO                             LI
UH-5465    ROJO                             L30
UH-5475    ROJO                             L30

(7 row(s) affected)
```

```
DECLARE @COLOR VARCHAR(30)='ROJO'
SELECT A.*
        FROM AUTOMOVIL A
        WHERE A.COL_AUT=@COLOR
GO

--•     Script que permita registrar un nuevo automóvil; en caso exista,
solo debe actualizarlo.
```

```
(1 row(s) affected)
INSERCION CORRECTA
```

```
(1 row(s) affected)
ACTUALIZACION CORRECTA
```

```
DECLARE @MAT CHAR(10)='UH-5475',
                @COL VARCHAR(30)='ROJO',
                @MOD VARCHAR(30)='L30'
IF EXISTS(SELECT * FROM AUTOMOVIL A WHERE A.MAT_AUT=@MAT)
BEGIN
        UPDATE AUTOMOVIL
                SET COL_AUT=@COL,MOD_AUT=@MOD
                WHERE MAT_AUT=@MAT
        PRINT 'ACTUALIZACION CORRECTA'
END
ELSE
BEGIN
        INSERT INTO AUTOMOVIL VALUES(@MAT,@COL,@MOD)
        PRINT 'INSERCION CORRECTA'
END
GO

--•     Script que permita registrar un nuevo cliente validando la existencia
del nombre del distrito. En caso el distrito exista, los datos del cliente
se registrarán; caso contrario, mostrar el mensaje "Código de distrito no
existe".
```

```
(1 row(s) affected)
INSERCION CORRECTA
```

```
CODIGO DE DISTRITO NO EXISTE
```

```
DECLARE @COD CHAR(5)='CL008',
        @APE VARCHAR(30)='CARBAJAL',
        @NOM VARCHAR(30)='SANDRO',
        @DNI CHAR(8)='57856767',
        @TEL VARCHAR(25)='598-5667',
        @DIS CHAR(3)='L05',
        @COR VARCHAR(50)='SCARBAJA@HOTMAIL.COM'
```

```
IF EXISTS(SELECT * FROM DISTRITO D WHERE D.IDE_DIS=@DIS)
BEGIN
	INSERT INTO CLIENTE VALUES(@COD,@APE,@NOM,@DNI,@TEL,@DIS,@COR)
	PRINT 'INSERCION CORRECTA'
END
ELSE
BEGIN
	PRINT 'CODIGO DE DISTRITO NO EXISTE'
END
```

```
--•    Mostrar el siguiente listado de los alquileres, adicionando una
columna que tome en cuenta el siguiente criterio:
--MONTO ALQUILER      CRITERIO
--MENOR A 50          BAJO
--ENTRE 50 Y 100      MEDIO
--MAYOR A 100         ALTO
```

```
ALQUILER    FECHA      MONTO                CONDICION
----------- ---------- -------------------- ---------
1           2010-03-12 185.00               ALTO
2           2011-01-02 100.00               MEDIO
3           2011-05-01 54.00                MEDIO
4           2012-08-11 85.00                MEDIO
5           2012-06-22 18.00                BAJO
6           2012-01-30 15.00                BAJO
7           2012-04-16 165.00               ALTO
8           2012-05-23 135.00               ALTO
9           2012-03-21 225.00               ALTO

(9 row(s) affected)
```

```
SELECT A.NUM_ALQ AS ALQUILER, A.FEC_ALQ AS FECHA,
	A.MON_ALQ AS MONTO,
	CASE
		WHEN MON_ALQ<50 THEN 'BAJO'
		WHEN MON_ALQ BETWEEN 50 AND 100 THEN 'MEDIO'
		WHEN MON_ALQ>100 THEN 'ALTO'
	END AS CONDICION
	FROM ALQUILER A
GO
```



PROCEDIMIENTOS ALMACENADOS

CAPÍTULO 6

Competencia tecnológica:

- Implementar procedimientos almacenados con y sin parámetros.
- Implementar procedimientos almacenados con parámetros de salida.

Contenido:

6.1 Procedimientos almacenados
6.2 Tipos de procedimientos almacenados
6.3 Procedimientos almacenados del sistema
6.4 Instrucción Execute y SP_ExecuteSQL
6.5 Procedimientos almacenados definidos por el usuario
6.6 Opciones sobre los procedimientos almacenados
6.7 Transacciones en Transact SQL
6.8 Begin Transaction
- Caso desarrollado

6.1 Procedimientos almacenados

Un procedimiento almacenado es un conjunto de instrucciones de Transact-SQL o una referencia a un método de Common Language Runtime (CLR) de Microsoft .NET Framework que puede aceptar y devolver parámetros proporcionados por el usuario.

Los procedimientos se pueden crear para uso permanente o para uso temporal en una sesión, un procedimiento local temporal, o para su uso temporal en todas las sesiones, un procedimiento temporal global.

El Motor de base de datos considera al procedimiento almacenado como un objeto que contiene un conjunto de instrucciones SQL que se ejecutan en conjunto con un mismo nombre y una sola llamada.

Consideraciones generales sobre los procedimientos almacenados:

- Todo procedimiento almacenado se registra en el servidor actual.
- Pueden incluir atributos de seguridad como permisos y cadenas de propiedad; además, se les pueden asociar certificados.
- Los procedimientos almacenados mejoran la seguridad de una aplicación.
- Los procedimientos almacenados con parámetros pueden ayudar a proteger la aplicación ante ataques por inyección de código SQL.
- Permiten una programación modular.
- Puede crear el procedimiento una vez y llamarlo desde el programa tantas veces como desee. Así, puede mejorar el mantenimiento de la aplicación y permitir que las aplicaciones tengan acceso a la base de datos de manera uniforme.
- Pueden reducir el tráfico de red.
- Una operación que necesite centenares de líneas de código Transact-SQL puede realizarse mediante una sola instrucción que ejecute el código en un procedimiento, en vez de enviar cientos de líneas de código por la red.

6.2 Tipos de procedimientos almacenados

Se describen particularmente dos tipos de procedimientos almacenados:

- Procedimientos almacenados del sistema
- Procedimientos almacenados definidos por el usuario

6.3 Procedimientos almacenados del sistema

Los procedimientos almacenados del sistema generalmente son guardados en la base de datos Master y son típicamente identificados por el prefijo SP_. Ellos realizan una amplia variedad de tareas para soportar las funciones del SQL Server soportando:

- llamadas de aplicaciones externas para datos de las tablas del sistema;
- procedimientos generales para administración de las bases de datos;
- funciones de administración de seguridad.

Se nombrará algunos procedimientos almacenados del sistema y se propondrá algunos casos relevantes de estos procedimientos.

A continuación, algunos casos de implementación de procedimientos almacenados:

Procedimiento almacenado del sistema que muestre los privilegios de las columnas involucradas en la tabla PRODUCTO.



PRODUCTO	
Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

```
SP_COLUMN_PRIVILEGES PRODUCTO
GO
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	GRANTOR	GRANTEE	PRIVILEGE	IS_GRANTABLE
1	BDVENTAS	dbo	PRODUCTO	COD_PRO	dbo	dbo	INSERT	YES
2	BDVENTAS	dbo	PRODUCTO	COD_PRO	dbo	dbo	REFERENCES	YES
3	BDVENTAS	dbo	PRODUCTO	COD_PRO	dbo	dbo	SELECT	YES
4	BDVENTAS	dbo	PRODUCTO	COD_PRO	dbo	dbo	UPDATE	YES
5	BDVENTAS	dbo	PRODUCTO	DES_PRO	dbo	dbo	INSERT	YES
6	BDVENTAS	dbo	PRODUCTO	DES_PRO	dbo	dbo	REFERENCES	YES
7	BDVENTAS	dbo	PRODUCTO	DES_PRO	dbo	dbo	SELECT	YES
8	BDVENTAS	dbo	PRODUCTO	DES_PRO	dbo	dbo	UPDATE	YES
9	BDVENTAS	dbo	PRODUCTO	IMP_PRO	dbo	dbo	INSERT	YES
10	BDVENTAS	dbo	PRODUCTO	IMP_PRO	dbo	dbo	REFERENCES	YES

Procedimiento almacenado del sistema que muestre las columnas de la tabla PRODUCTO.

```
SP_COLUMNS PRODUCTO
GO
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION
1	BDVENTAS	dbo	PRODUCTO	COD_PRO	1	char	5
2	BDVENTAS	dbo	PRODUCTO	DES_PRO	12	varchar	50
3	BDVENTAS	dbo	PRODUCTO	PRE_PRO	3	money	19
4	BDVENTAS	dbo	PRODUCTO	SAC_PRO	4	int	10
5	BDVENTAS	dbo	PRODUCTO	SMI_PRO	4	int	10
6	BDVENTAS	dbo	PRODUCTO	UNI_PRO	12	varchar	30
7	BDVENTAS	dbo	PRODUCTO	LIN_PRO	12	varchar	30
8	BDVENTAS	dbo	PRODUCTO	IMP_PRO	12	varchar	10

Procedimiento almacenado que liste la información del producto con código P001.

```
COD_PRO DES_PRO                 PRE_PRO    SAC_PRO  SMI_PRO     UNI_PRO  LIN_PRO  IMP_PRO
------- ----------------------- ---------- -------- ----------- -------- -------- --------
P001    PAPEL BOND A-4 ATLAS    18.50      200      1500        MLL      2        V

(1 row(s) affected)
```

```
EXECUTE SP_EXECUTESQL
          N'SELECT * FROM PRODUCTO
          WHERE COD_PRO = @COD',
          N'@COD CHAR(5)',
          @COD = 'P001'
GO

Procedimiento almacenado del sistema que devuelva una lista de nombres de
atributos y sus valores correspondientes para SQL Server, la puerta de
enlace de la base de datos o el origen de datos subyacente del servidor
activo.
```

```
attribute_id attribute_name                                                attribute_value
------------ ------------------------------------------------------------ ----------------------------------------------------
1            DBMS_NAME                                                     Microsoft SQL Server
2            DBMS_VER                                                      Microsoft SQL Server 2012 - 11.0.2218.0 - 11.0.2218.0
10           OWNER_TERM                                                    owner
11           TABLE_TERM                                                    table
12           MAX_OWNER_NAME_LENGTH                                         128
13           TABLE_LENGTH                                                  128
14           MAX_QUAL_LENGTH                                               128
15           COLUMN_LENGTH                                                 128
16           IDENTIFIER_CASE                                               MIXED
17           TX_ISOLATION                                                  2
18           COLLATION_SEQ                                                 charset=iso_1 collation=Modern_Spanish_CI_AS
19           SAVEPOINT_SUPPORT                                             Y
20           MULTI_RESULT_SETS                                             Y
22           ACCESSIBLE_TABLES                                             Y
100          USERID_LENGTH                                                 128
101          QUALIFIER_TERM                                                database
```

```
SP_SERVER_INFO
GO

Procedimiento almacenado que lista las tablas creadas dentro de la base
de datos VENTAS.
```

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	TABLE_TYPE
1	BDVENTAS	dbo	ABASTECIMIENTO	TABLE
2	BDVENTAS	dbo	CLIENTE	TABLE
3	BDVENTAS	dbo	DETALLE_COMPRA	TABLE
4	BDVENTAS	dbo	DETALLE_FACTURA	TABLE
5	BDVENTAS	dbo	DISTRITO	TABLE
6	BDVENTAS	dbo	FACTURA	TABLE
7	BDVENTAS	dbo	ORDEN_COMPRA	TABLE
8	BDVENTAS	dbo	PRODUCTO	TABLE
9	BDVENTAS	dbo	PRODUCTO_BAK	TABLE
10	BDVENTAS	dbo	PROVEEDOR	TABLE
11	BDVENTAS	dbo	REPORTE_PRODUCTO	TABLE
12	BDVENTAS	dbo	sysdiagrams	TABLE
13	BDVENTAS	dbo	VENDEDOR	TABLE

```
SP_TABLES
GO
```

6.4 Instrucción Execute y SP_ExecuteSQL

La instrucción EXECUTE indica que SqlClient, ODBC, OLE DB o DB-Library deben ejecutar una o varias instrucciones Transact-SQL preparadas; claro está que dichas instrucciones deben cumplir con la sintaxis evaluada desde el Motor de base de datos de SQL.

Formato EXECUTE:

```
EXEC | EXECUTE <NOMBRE DEL PROCEDIMIENTO> <PARÁMETROS>
GO
```

Se especifica el nombre del procedimiento almacenado y, opcionalmente, se colocan los parámetros del procedimiento según su implementación.

SP_EXECUTESQL es un procedimiento almacenado que ejecuta una cadena Unicode en la cual admite la sustitución de parámetros. Hay que considerar que el valor deberá ser mínimamente de tipo Nvarchar. Este procedimiento deberá ejecutarse con la instrucción EXEC obligatoriamente.

A continuación, algunos casos sobre el uso de la sentencia EXECUTE y SP_EXECUTESQL:

```
Implementar un procedimiento almacenado que permita listar los registros
de la tabla PRODUCTO.
Usando Exec
DECLARE @X VARCHAR(MAX)
SET @X='SELECT * FROM PRODUCTO'
EXEC(@X)
GO

Usando SP_EXECUTESQL
DECLARE @X NVARCHAR(MAX)
SET @X='SELECT * FROM PRODUCTO'
EXEC SP_EXECUTESQL @X
GO
```

6.5 Procedimientos almacenados definidos por el usuario

Son procedimientos que se implementan de forma personalizada según la necesidad del usuario; aquí se podrá emplear cualquier instrucción vista hasta el momento.

Formato:

```
CREATE  PROCEDURE  <NOMBRE_PROCEDIMIENTO>  <(@PARAMETRO TIPO_DATOS) >
AS
BEGIN
   CUERPO_DEL_PROCEDIMIENTO
END
```

A continuación, algunos casos del uso de procedimientos almacenados sin definición de parámetros:

Procedimiento almacenado que permita devolver los 5 productos con más altos precios registrados en la tabla PRODUCTO.

```
COD_PRO DES_PRO                 PRE_PRO    SAC_PRO     SMI_PRO     UNI_PRO    LIN_PRO   IMP_PRO
------- ----------------------- ---------- ----------- ----------- ---------- --------- ---------
P020    FOLDER PLASTICO A-4     52.00      3080        1100        CIE        3         F
P021    PROTECTOR DE PANTALLA   52.00      40          5           UNI        1         F
P006    CARTUCHO TINTA COLOR    47.00      58          35          UNI        1         F
P002    PAPEL BOND OFICIO       37.00      100         1500        MLL        2         F
P008    CAJA DE DISKETTES * 10  32.00      125         180         UNI        1         F

(5 row(s) affected)
```

```
--Validando el procedimiento almacenado
IF OBJECT_ID('TOP5PRECIOS') IS NOT NULL
BEGIN
        DROP PROCEDURE TOP5PRECIOS
END
GO
--Creando el procedimiento almacenado
CREATE PROCEDURE TOP5PRECIOS
AS
        SET ROWCOUNT 5
        SELECT * FROM PRODUCTO ORDER BY PRE_PRO DESC
GO
--Ejecutando el procedimiento almacenado
EXECUTE TOP5PRECIOS
GO
```

Procedimiento almacenado que permita realizar el siguiente reporte:

```
CODIGO VENDEDOR                                SUELDO               FECHA DE INICIO DISTRITO
------ --------------------------------------- -------------------- --------------- --------------
V01    JUANA ALVA                              1000.00              2003-07-12      CHORRILLOS
V02    JUAN SOTO                               1200.00              2002-10-03      SAN ISIDRO
V03    CARLOS AREVALO                          1500.00              1992-02-10      SAN BORJA
V04    CESAR OJEDA                             850.00               1999-08-11      SURCO
V05    JULIO VEGA                              1500.00              1997-11-05      SURCO
V06    ANA ORTEGA                              1200.00              1994-11-05      SAN MIGUEL
V07    JOSE PALACIOS                           2500.00              1997-12-02      JESUS MARIA
V08    RUBEN GOMEZ                             1450.00              1996-12-08      LA MOLINA
V09    PATRICIA ARCE                           1800.00              1997-12-05      LA MOLINA
V10    RENATO PEREZ                            1550.00              1998-11-08      SURCO

(10 row(s) affected)
```

```
IF OBJECT_ID('SP_LISTADOVENDEDOR')IS NOT NULL
        DROP PROC SP_LISTADOVENDEDOR
GO
CREATE PROC SP_LISTADOVENDEDOR
AS
        SELECT V.COD_VEN AS CODIGO,
                V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
                V.SUE_VEN AS SUELDO,
                V.FIN_VEN AS [FECHA DE INICIO],
                D.NOM_DIS AS DISTRITO
                FROM VENDEDOR V
                JOIN DISTRITO D ON V.COD_DIS=D.COD_DIS
GO
```

```
--PRUEBA
EXEC SP_LISTADOVENDEDOR
GO

Procedimiento almacenado que permita mostrar el total de clientes
registrados por distrito.
```

```
DISTRITO                                           TOTAL DE CLIENTES
-------------------------------------------------- -----------------
BARRANCO                                           1
BELLAVISTA                                         1
BREÑA                                              1
CALLAO                                             1
JESUS MARIA                                        1
LA MOLINA                                          1
LIMA - CERCADO                                     1
LINCE                                              1
LOS OLIVOS                                         1
MIRAFLORES                                         1
SAN BORJA                                          1
SAN JUAN DE LURIGANCHO                             1
SAN LUIS                                           1
SAN MIGUEL                                         5
SURCO                                              1
SURQUILLO                                          1

(16 row(s) affected)
```

```
IF OBJECT_ID('SP_CLIENTESxDISTRITO')IS NOT NULL
        DROP PROC SP_CLIENTESxDISTRITO
GO
CREATE PROC SP_CLIENTESxDISTRITO
AS
        SELECT D.NOM_DIS AS DISTRITO,
                COUNT(*) AS [TOTAL DE CLIENTES]
                FROM CLIENTE C
                JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
                GROUP BY D.NOM_DIS
GO
--PRUEBA:
EXEC SP_CLIENTESxDISTRITO
GO
```

A continuación, algunos casos del uso de procedimientos almacenados con definición de parámetros:

```
Procedimiento almacenado que permita mostrar información de un determi-
nado producto.
```

```
CODIGO DESCRIPCION                                         PRECIO               STOCK ACTUAL STOCK MINIMO
------ -------------------------------------------------- -------------------- ------------ ------------
P001   PAPEL BOND A-4 ATLAS                                18.50                200          1500

(1 row(s) affected)
```

```
IF OBJECT_ID('SP_MUESTRAPRODUCTO')IS NOT NULL
        DROP PROC SP_MUESTRAPRODUCTO
GO
```

```
CREATE PROC SP_MUESTRAPRODUCTO(@COD CHAR(5))
AS
        SELECT P.COD_PRO AS CODIGO,
               P.DES_PRO AS DESCRIPCION,
               P.PRE_PRO AS PRECIO,
               P.SAC_PRO AS [STOCK ACTUAL],
               P.SMI_PRO AS [STOCK MINIMO]
               FROM PRODUCTO P
               WHERE P.COD_PRO=@COD
GO

--PRUEBA:
EXEC SP_MUESTRAPRODUCTO 'P001'
```

Procedimiento almacenado que permita mostrar el total de facturas registradas en un determinado año.

```
AÑO         TOTAL DE FACTURAS
----------- -----------------
2012        2

(1 row(s) affected)
```

```
IF OBJECT_ID('SP_FACTURASxAÑO')IS NOT NULL
        DROP PROC SP_FACTURASxAÑO
GO

CREATE PROC SP_FACTURASxAÑO(@AÑO INT)
AS
        SELECT YEAR(F.FEC_FAC) AS AÑO,
               COUNT(*) AS [TOTAL DE FACTURAS]
               FROM FACTURA F
               GROUP BY YEAR(F.FEC_FAC)
               HAVING YEAR(F.FEC_FAC) = @AÑO
GO

--PRUEBA:
EXEC SP_FACTURASxAÑO 2012
```

Procedimiento almacenado que permita listar los clientes de un determinado distrito.

```
CODIGO CLIENTE                        DIRECCION                                  TELEFONO  DISTRITO      FECHA DE REGISTRO
------ ------------------------------ ------------------------------------------ --------- ------------- ----------------------
C001   FINSETH                       AV. LOS VIÑEDOS 150                        4342318   SAN MIGUEL    2008-12-10
C003   SERVIEMSA                      JR. COLLAGATE 522                          75012403  SAN MIGUEL    2011-06-03
C006   BERKER                         AV. LOS PROCERES 521                       3810322   SAN MIGUEL    2013-07-05
C010   LANDU                          AV.NICOLAS DE AYLLON 1453                  3267840   SAN MIGUEL    2013-11-06
C012   SUCERTE                        JR. GRITO DE HUAURA 114                    4206434   SAN MIGUEL    2011-11-08

(5 row(s) affected)
```

```
IF OBJECT_ID('SP_CLIENTExDISTRITO')IS NOT NULL
        DROP PROC SP_CLIENTExDISTRITO
GO

CREATE PROC SP_CLIENTExDISTRITO(@DIS VARCHAR(30))
AS
        SELECT C.COD_CLI AS CODIGO,
                C.RSO_CLI AS CLIENTE,
                C.DIR_CLI AS DIRECCION,
                C.TLF_CLI AS TELEFONO,
                D.NOM_DIS AS DISTRITO,
                C.FEC_REG AS [FECHA DE REGISTRO]
                FROM CLIENTE C
                JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
                WHERE D.NOM_DIS=@DIS
GO

--PRUEBA:
EXEC SP_CLIENTExDISTRITO 'SAN MIGUEL'
GO
```

Procedimiento almacenado que permita listar las facturas en un determinado año y un determinado mes.

```
NUMERO      FECHA FACTURADA CLIENTE                         FECHA CANCELACION ESTADO     VENDEDOR
----------- --------------- ------------------------------- ----------------- ---------- --------------
FA001       2013-06-07      FINSETH                         2013-05-08        2          JUANA ALVA
FA002       2013-06-07      COREFO                          2013-05-08        3          JUAN SOTO
FA004       2013-06-09      CARDELI                         2013-05-10        2          JOSE PALACIOS
FA016       2013-06-01      CARDELI                         2013-09-01        2          JULIO VEGA
FA017       2013-06-01      MEBA                            2013-06-01        1          ANA ORTEGA
FA020       2013-06-02      HAYASHI                         2013-10-03        2          JUAN SOTO

(6 row(s) affected)
```

```
IF OBJECT_ID('SP_FACTURASxAÑOxMES')IS NOT NULL
        DROP PROC SP_FACTURASxAÑOxMES
GO
CREATE PROC SP_FACTURASxAÑOxMES(@AÑO INT,@MES INT)
AS
        SELECT F.NUM_FAC AS NUMERO,
                F.FEC_FAC AS [FECHA FACTURADA],
                C.RSO_CLI AS CLIENTE,
                F.FEC_CAN AS [FECHA CANCELACION],
                F.EST_FAC AS ESTADO,
                V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR
                FROM FACTURA F
                JOIN CLIENTE C ON C.COD_CLI=F.COD_CLI
                JOIN VENDEDOR V ON V.COD_VEN=F.COD_VEN
                WHERE YEAR(F.FEC_FAC)=@AÑO AND MONTH(F.FEC_FAC)=@MES
GO

--PRUEBA:
EXEC SP_FACTURASxAÑOxMES 2013,6
GO
```

Procedimiento almacenado que permita registrar un nuevo cliente; se debe tener en cuenta validar el código del distrito.

```
(1 row(s) affected)
CLIENTE REGISTRADO CORRECTAMENTE
```

```
IF OBJECT_ID('SP_NUEVOCLIENTE') IS NOT NULL
BEGIN
	DROP PROCEDURE SP_NUEVOCLIENTE
END
GO
--2.
CREATE PROCEDURE SP_NUEVOCLIENTE(@COD CHAR(5),@RSO VARCHAR(30),
                                 @DIR VARCHAR(100),@TLF VARCHAR(9),
                                 @RUC VARCHAR(11),@DIS CHAR(3),
                                 @REG DATE,@TIP VARCHAR(10),
                                 @CON VARCHAR(30))
AS
	BEGIN TRY
		IF EXISTS(SELECT * FROM DISTRITO D WHERE D.COD_DIS=@DIS)
		BEGIN
			INSERT INTO CLIENTE
				VALUES (@COD,@RSO,@DIR,@TLF,
					@RUC,@DIS,@REG,@TIP,@CON)
			PRINT 'CLIENTE REGISTRADO CORRECTAMENTE'
		END
		ELSE
			PRINT 'ERROR AL REGISTRAR'
	END TRY
	BEGIN CATCH
		PRINT 'OCURRIO UN ERROR AL REGISTRAR'
	END CATCH
GO
--PRUEBA:
EXEC	SP_NUEVOCLIENTE	'C021','TAI	LOY','AV.	UNIVERSITARIA
3424','4529636','51551585745','D01','25/10/2017','1','JUAN GARCIA'
GO
```

Procedimiento almacenado que permita registrar un nuevo distrito; el código del nuevo distrito debe generarse automáticamente y mostrar un mensaje de error si el distrito ya se encuentra registrado en la tabla.

```
(1 row(s) affected)
DISTRITO REGISTRADO CORRECTAMENTE
```

```
IF OBJECT_ID('SP_NUEVODISTRITO') IS NOT NULL
BEGIN
	DROP PROCEDURE SP_NUEVODISTRITO
END
GO
```

```
CREATE PROCEDURE SP_NUEVODISTRITO(@DIS VARCHAR(50))
AS
   SET ROWCOUNT 1
   DECLARE @COD CHAR(5)
   SELECT @COD=RIGHT(D.COD_DIS,2) FROM DISTRITO D ORDER BY D.COD_DIS DESC

   DECLARE @NUEVOCOD CHAR(5) = 'D'+CAST(@COD+1 AS CHAR(2))

   IF NOT EXISTS(SELECT D.NOM_DIS FROM DISTRITO D WHERE D.NOM_DIS=@DIS)
   BEGIN
        INSERT INTO DISTRITO VALUES (@NUEVOCOD,@DIS)
        PRINT 'DISTRITO REGISTRADO CORRECTAMENTE'
   END
   ELSE
        PRINT 'EL DISTRITO YA SE ENCUENTRA REGISTRADO'
GO
--Prueba:
SP_NUEVODISTRITO 'NUEVO CERCADO'
GO
```

A continuación, algunos casos del uso de procedimientos almacenados con parámetros de salida:

Procedimiento almacenado que permita mostrar el total de facturas registradas en un determinado año.

```
EL TOTAL DE FACTURAS REGISTRADAS EN EL AÑO 2012 ES 2
```

```
IF OBJECT_ID('SP_TOTALFACTURAS') IS NOT NULL
BEGIN
       DROP PROCEDURE SP_TOTALFACTURAS
END
GO

CREATE PROCEDURE SP_TOTALFACTURAS(@AÑO INT, @TOTAL INT OUTPUT)
AS
       SELECT @TOTAL= COUNT(*)
              FROM FACTURA F
              WHERE YEAR(F.FEC_FAC) = @AÑO
GO

--PRUEBA
DECLARE @T INT,@AÑO INT = 2012
EXEC SP_TOTALFACTURAS @AÑO,@TOTAL=@T OUTPUT
PRINT 'EL TOTAL DE FACTURAS REGISTRADAS EN EL AÑO ' +
      CAST(@AÑO AS CHAR(4)) + ' ES ' + CAST(@T AS VARCHAR(10))
GO
```

6.6 Opciones sobre los procedimientos almacenados

Los procedimientos almacenados representan objetos dentro de la base de datos, los cuales pueden ser administrados por medio de sentencias DDL.

Formato para la validación de un procedimiento almacenado:

```
IF OBJECT_ID('NOMBRE_PROCEDIMIENTO') IS NOT NULL
  DROP PROC NOMBRE_PROCEDIMIENTO
GO
```

Formato para crear un procedimiento almacenado:

```
CREATE PROCEDURE <NOMBRE_PROCEDIMIENTO>
AS
 --CONTENIDO DEL PROCEDIMIENTO
GO
```

Formato para modificar un procedimiento almacenado:

```
ALTER PROCEDURE <NOMBRE_PROCEDIMIENTO>
AS
  --MODIFICAR EL CONTENIDO
GO
```

Formato para eliminar un procedimiento almacenado:

```
DROP  PROCEDURE <NOMBRE_PROCEDIMIENTO>
GO
```

Formato para listar todos los procedimientos almacenados de una base de datos:

```
SELECT P.NAME,P.ID,P.CRDATE,P.TYPE FROM SYS.SYSOBJECTS P
                WHERE P.XTYPE='P'
GO
```

Por ejemplo, si se quiere visualizar los procedimientos almacenados de la base de datos VENTAS, el resultado se mostraría como sigue:

```
name                                   id          crdate                  type
-------------------------------------- ----------- ----------------------- ----
sp_upgraddiagrams                      757577737   2017-03-10 18:43:05.310 P
sp_helpdiagrams                        821577965   2017-03-10 18:43:05.317 P
sp_helpdiagramdefinition               837578022   2017-03-10 18:43:05.317 P
sp_creatediagram                       853578079   2017-03-10 18:43:05.317 P
sp_renamediagram                       869578136   2017-03-10 18:43:05.320 P
sp_alterdiagram                        885578193   2017-03-10 18:43:05.320 P
sp_dropdiagram                         901578250   2017-03-10 18:43:05.320 P
TOP5PRECIOS                            1525580473  2017-03-14 18:52:46.513 P
SP_LISTADOVENDEDOR                     1541580530  2017-03-14 18:59:16.023 P
SP_CLIENTESxDISTRITO                   1557580587  2017-03-14 19:03:12.643 P
SP_FACTURASxAÑO                        1573580644  2017-03-14 19:10:49.283 P
SP_MUESTRAPRODUCTO                     1589580701  2017-03-14 19:13:53.057 P
SP_FACTURASxAÑOxMES                    1605580758  2017-03-14 19:17:16.660 P
SP_CLIENTExDISTRITO                    1621580815  2017-03-14 19:21:22.140 P
SP_NUEVOCLIENTE                        1653580929  2017-03-14 19:38:40.913 P
SP_NUEVODISTRITO                       1685581043  2017-03-14 20:54:41.780 P
SP_TOTALFACTURAS                       1701581100  2017-03-14 21:03:40.070 P

(17 row(s) affected)
```

Formato para visualizar el contenido de un procedimiento almacenado:

```
SP_HELPTEXT NOMBRE_PROCEDIMIENTO
GO
```

6.7 Transacciones en Transact SQL

Una transacción es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, en forma indivisible o atómica.

Un SGBD se dice es transaccional si es capaz de mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

Para esto, el Motor de base de datos de SQL Server provee los mecanismos para especificar que un conjunto de acciones debe constituir una transacción. Se cuenta específicamente con tres:

- BEGIN TRAN: Especifica que va a empezar una transacción.
- COMMIT TRAN: Le indica al Motor de base de datos que puede considerar la transacción completada con éxito.
- ROLLBACK TRAN: Indica que se ha alcanzado un fallo y que se debe restablecer la base al punto de integridad.

6.8 BEGIN TRANSACTION

Representa un punto en el que los datos a los que hace referencia una conexión son lógica y físicamente coherentes. Si se producen errores, se pueden revertir todas las modificaciones realizadas en los datos después de BEGIN TRANSACTION para devolver los datos al estado conocido de coherencia. Cada transacción dura hasta que se completa sin errores y se emite COMMIT TRANSACTION para hacer que las modificaciones sean una parte permanente de la base de datos, o hasta que se produzcan errores y se borren todas las modificaciones con la instrucción ROLLBACK TRANSACTION.

Aunque BEGIN TRANSACTION inicia una transacción local, esta no se guardará en el registro de transacciones hasta que la aplicación realice, posteriormente, una acción que se deba almacenar en el registro, como la ejecución de una instrucción INSERT, UPDATE o DELETE.

Formato:

```
BEGIN  < TRAN | TRANSACTION > NOMBRE_TRANSACCION
  COMMIT TRAN <NOMBRE_TRANSACCION>
  ROLLBACK TRAN <NOMRE_TRANSACCION>
```

A continuación, algunos casos sobre el uso de transacciones:

```
Implementar un procedimiento almacenado que permita registrar un proveedor,
controlando dicha inserción por medio de una transacción. Emitir un
mensaje en cada ocasión; es decir, si todo es correcto, emitir "Proveedor
registrado con éxito"; en caso contrario, "Ocurrió un error al insertar".
```

PROVEEDOR

	Column Name	Condensed Type
▶🔑	COD_PRV	char(5)
	RSO_PRV	varchar(80)
	DIR_PRV	varchar(100)
	TEL_PRV	char(15)
	COD_DIS	char(5)
	REP_PRV	varchar(80)

```
(1 row(s) affected)
PROVEEDOR REGISTRADO CON EXITO
```

```
--Validar el procedimiento almacenado
IF OBJECT_ID('SP_NUEVOPROVEEDOR') IS NOT NULL
BEGIN
        DROP PROCEDURE SP_NUEVOPROVEEDOR
END
GO
```

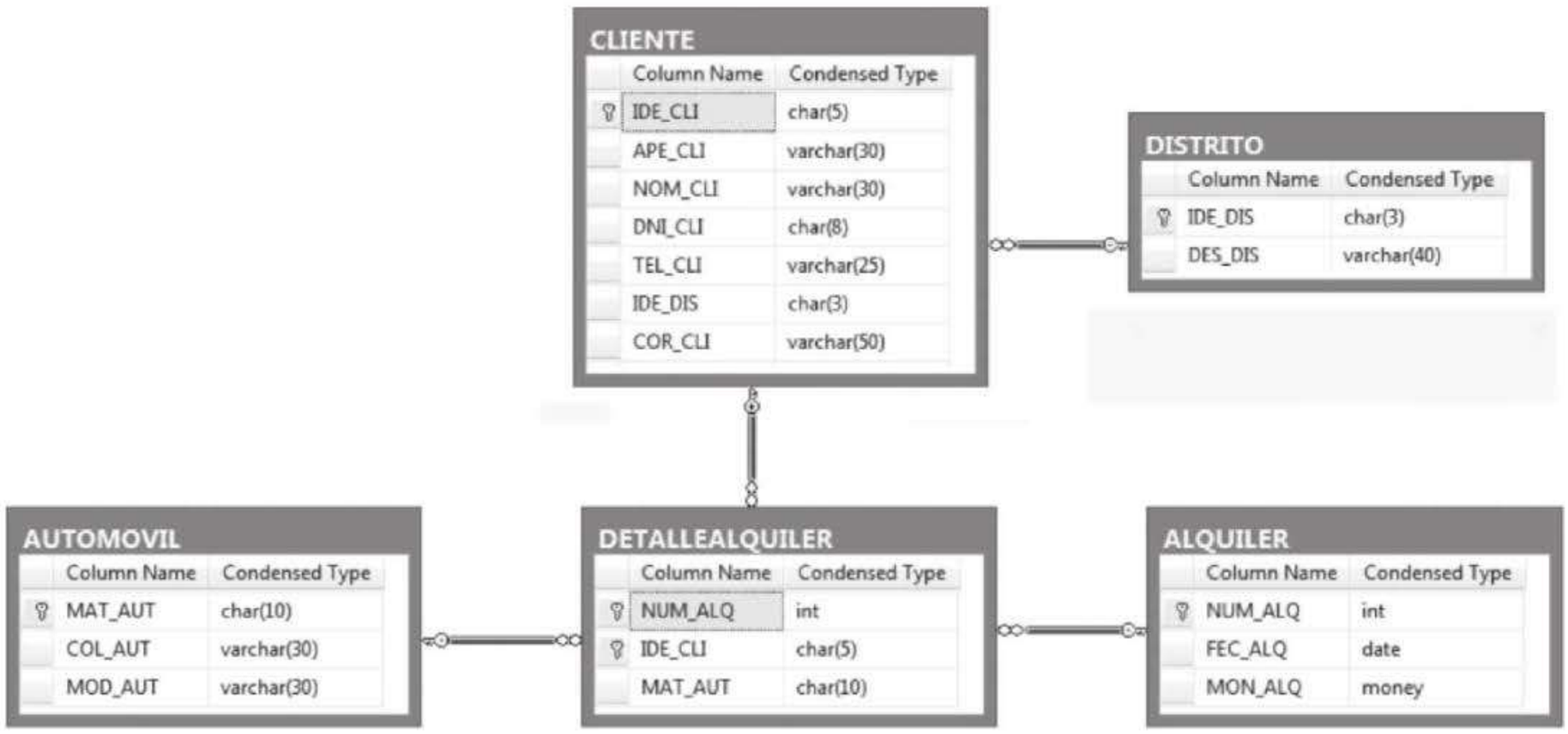
```
--Creando el procedimiento
CREATE PROCEDURE SP_NUEVOPROVEEDOR(  @COD CHAR(5),@RSO VARCHAR(80),
                                     @DIR VARCHAR(100),@TEL CHAR(15),
                                     @DIS CHAR(5),@REP VARCHAR(80))
AS
        BEGIN TRANSACTION T_PROVEEDOR
            INSERT INTO PROVEEDOR VALUES (@COD,@RSO,@DIR,@TEL,@DIS,@REP)
               IF @@ERROR=0
               BEGIN
                     PRINT 'PROVEEDOR REGISTRADO CON EXITO'
                     COMMIT TRAN T_PROVEEDOR
               END
               ELSE
               BEGIN
                     PRINT 'OCURRIO UN ERROR AL INSERTAR'
                     ROLLBACK TRAN T_PROVEEDOR
               END
GO

--Prueba:
SP_NUEVOPROVEEDOR 'PR21','TAI LOY','AV. UNIVERSITARIA','6544555',
                  'D02','JUAN FUIGUEROA'


También se puede usar la siguiente sentencia BEGIN TRY y BEGIN CATCH.
CREATE PROCEDURE SP_NUEVOPROVEEDOR(  @COD CHAR(5),@RSO VARCHAR(80),
                                     @DIR VARCHAR(100),@TEL CHAR(15),
                                     @DIS CHAR(5),@REP VARCHAR(80))
AS
        BEGIN TRANSACTION T_PROVEEDOR
               BEGIN TRY
                     INSERT INTO PROVEEDOR VALUES (@COD,@RSO,@DIR,
                                                   @TEL,@DIS,@REP)
                     COMMIT TRAN T_PROVEEDOR
                     PRINT 'PROVEEDOR REGISTRADO CON EXITO'
               END TRY
               BEGIN CATCH
                     PRINT 'OCURRIO UN ERROR AL INSERTAR'
                     ROLLBACK TRAN T_PROVEEDOR
               END CATCH
GO
```

CASO DESARROLLADO

Si se cuenta con el siguiente diagrama de base de datos:



Y con la siguiente información por tablas:

SELECT * FROM CLIENTE

```
IDE_CLI APE_CLI    NOM_CLI      DNI_CLI  TEL_CLI     IDE_DIS COR_CLI
------- ---------- ------------ -------- ----------- ------- -------
CL001   LOPEZ      MARIA        10856985 526-5254    L03
CL002   GOMEZ      JOSE         02188525 585-8212    L02
CL003   ACOSTA     JESUS        07415482 485-1598    L01
CL004   ARIAS      GUADSLUPE    08995255 258-8569    L04
CL005   CARLOS     MANUEL       10418525 296-9685    L08
CL006   GERZ       BRIGITTE     10952155 585-1254    L06
CL007   BARBOSA    MARISOL      78514855 584-5869    L04

(7 row(s) affected)
```

SELECT * FROM DISTRITO

```
IDE_DIS DES_DIS
------- --------------------
L01     LIMA
L02     LINCE
L03     BREÑA
L04     LOS OLIVOS
L05     RIMAC
L06     SAN MARTIN DE PORRES
L07     SAN LUIS
L08     SAN BORJA

(8 row(s) affected)
```

SELECT * FROM AUTOMOVIL

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- ---------------
AF-456     ROJO                             SEDAN
D4-243     NEGRO                            SEDAN
FD-463     PLATA                            STATION WAGON
FG-654     PLATA                            L10
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
HY-345     PLATA                            CHERY
SF-112     ROJO                             LI
YT-457     NEGRO                            STATION WAGON

(10 row(s) affected)
```

SELECT * FROM ALQUILER

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ----------
1           2010-03-12 185.00
2           2011-01-02 100.00
3           2011-05-01 54.00
4           2012-08-11 85.00
5           2012-06-22 18.00
6           2012-01-30 15.00
7           2012-04-16 165.00
8           2012-05-23 135.00
9           2012-03-21 225.00

(9 row(s) affected)
```

SELECT * FROM DETALLEALQUILER

```
NUM_ALQ     IDE_CLI MAT_AUT
----------- ------- ----------
1           CL002   GG-654
2           CL002   GT-532
3           CL006   FD-463
3           CL007   GT-532
4           CL005   H5-455
5           CL004   SF-112
6           CL004   YT-457
7           CL003   GT-532

(8 row(s) affected)
```

Implemente los siguientes procedimientos almacenados:

- Procedimiento almacenado que permita listar los datos de los clientes, dentro de los cuales se debe mostrar el nombre del distrito.
- Procedimientos almacenados que permitan dar mantenimiento a la tabla CLIENTE.
- Procedimiento almacenado que determine el último código del cliente registrado.
- Procedimiento almacenado que muestre el total de clientes por distrito.
- Procedimiento almacenado que permita listar los clientes de un determinado distrito.
- Procedimiento almacenado que permita mostrar información del detalle de alquiler, de acuerdo al número de alquiler.
- Procedimiento almacenado que permita mostrar los registros de automóviles de un determinado color.
- Procedimiento almacenado que muestra el detalle de alquiler de acuerdo al nombre de un determinado cliente.
- Procedimiento almacenado que permita listar los alquileres de un determinado año de registro.

Solución:

```
--•    Procedimiento almacenado que permita listar los datos de los
--clientes dentro de los cuales se debe mostrar el nombre del distrito.
IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
       DROP PROC SP_LISTACLIENTES
GO
CREATE PROC SP_LISTACLIENTES
AS
       SELECT C.IDE_CLI AS CODIGO, C.APE_CLI AS APELLIDOS,
              C.NOM_CLI AS NOMBRES, C.DNI_CLI AS DNI,
              C.TEL_CLI AS TELEFONO,
              D.DES_DIS AS DISTRITO,
              C.COR_CLI AS CORREO
              FROM CLIENTE C
              JOIN DISTRITO D ON C.IDE_DIS=D.IDE_DIS
GO
--PRUEBA:
SP_LISTACLIENTES
GO

--•    Procedimientos almacenados que permitan dar mantenimiento
--a la tabla CLIENTE.
IF OBJECT_ID('SP_NUEVOCLIENTE') IS NOT NULL
       DROP PROC SP_NUEVOCLIENTE
GO
CREATE PROC SP_NUEVOCLIENTE(@COD CHAR(5),@APE VARCHAR(30),
                            @NOM VARCHAR(30),@DNI CHAR(8),
                            @TEL VARCHAR(25),@DIS CHAR(3),
                            @COR VARCHAR(50))
```

```
AS
        INSERT INTO CLIENTE
                VALUES(@COD,@APE,@NOM,@DNI,@TEL,@DIS,@COR)
GO

IF OBJECT_ID('SP_MODIFICACLIENTE') IS NOT NULL
        DROP PROC SP_MODIFICACLIENTE
GO
CREATE PROC SP_MODIFICACLIENTE(@COD CHAR(5),@APE VARCHAR(30),
                               @NOM VARCHAR(30),@DNI CHAR(8),
                               @TEL VARCHAR(25),@DIS CHAR(3),
                               @COR VARCHAR(50))
AS
        UPDATE CLIENTE
                SET APE_CLI=@APE,NOM_CLI=@NOM,
                        DNI_CLI=@DNI,TEL_CLI=@TEL,
                        IDE_DIS=@DIS,COR_CLI=@COR
                WHERE IDE_CLI=@COD
GO

IF OBJECT_ID('SP_ELIMINACLIENTE') IS NOT NULL
        DROP PROC SP_ELIMINACLIENTE
GO
CREATE PROC SP_ELIMINACLIENTE(@COD CHAR(5))
AS
        DELETE CLIENTE
                WHERE IDE_CLI=@COD
GO

--PRUEBA
SP_NUEVOCLIENTE 'CL009','ROLF','CARLOS','45784598','541-5858','L05','CRO
LF@GAMIL.COM'
GO

SP_MODIFICACLIENTE 'CL009','ROLF','CARLOS','45784598','541-5858','L05','
CROLF@GAMIL.COM'
GO

SP_ELIMINACLIENTE 'CL009'
GO

--•    Procedimiento almacenado que determine el último código del cliente
registrado.
IF OBJECT_ID('SP_ULTIMOCODIGO') IS NOT NULL
        DROP PROC SP_ULTIMOCODIGO
GO
CREATE PROC SP_ULTIMOCODIGO
AS
        SELECT TOP 1 C.IDE_CLI
                FROM CLIENTE C
                ORDER BY 1 DESC
GO
--PRUEBA
SP_ULTIMOCODIGO
```

```
--•    Procedimiento almacenado que muestre el total de clientes por
distrito.
IF OBJECT_ID('SP_TOTALCLIENTESxDISTRITO') IS NOT NULL
       DROP PROC SP_TOTALCLIENTESxDISTRITO
GO
CREATE PROC SP_TOTALCLIENTESxDISTRITO
AS
       SELECT D.DES_DIS AS DISTRITO,
              COUNT(*) AS [TOTAL DE CLIENTES]
              FROM CLIENTE C
              JOIN DISTRITO D ON C.IDE_DIS=D.IDE_DIS
              GROUP BY D.DES_DIS
GO
--PRUEBA
SP_TOTALCLIENTESxDISTRITO
GO

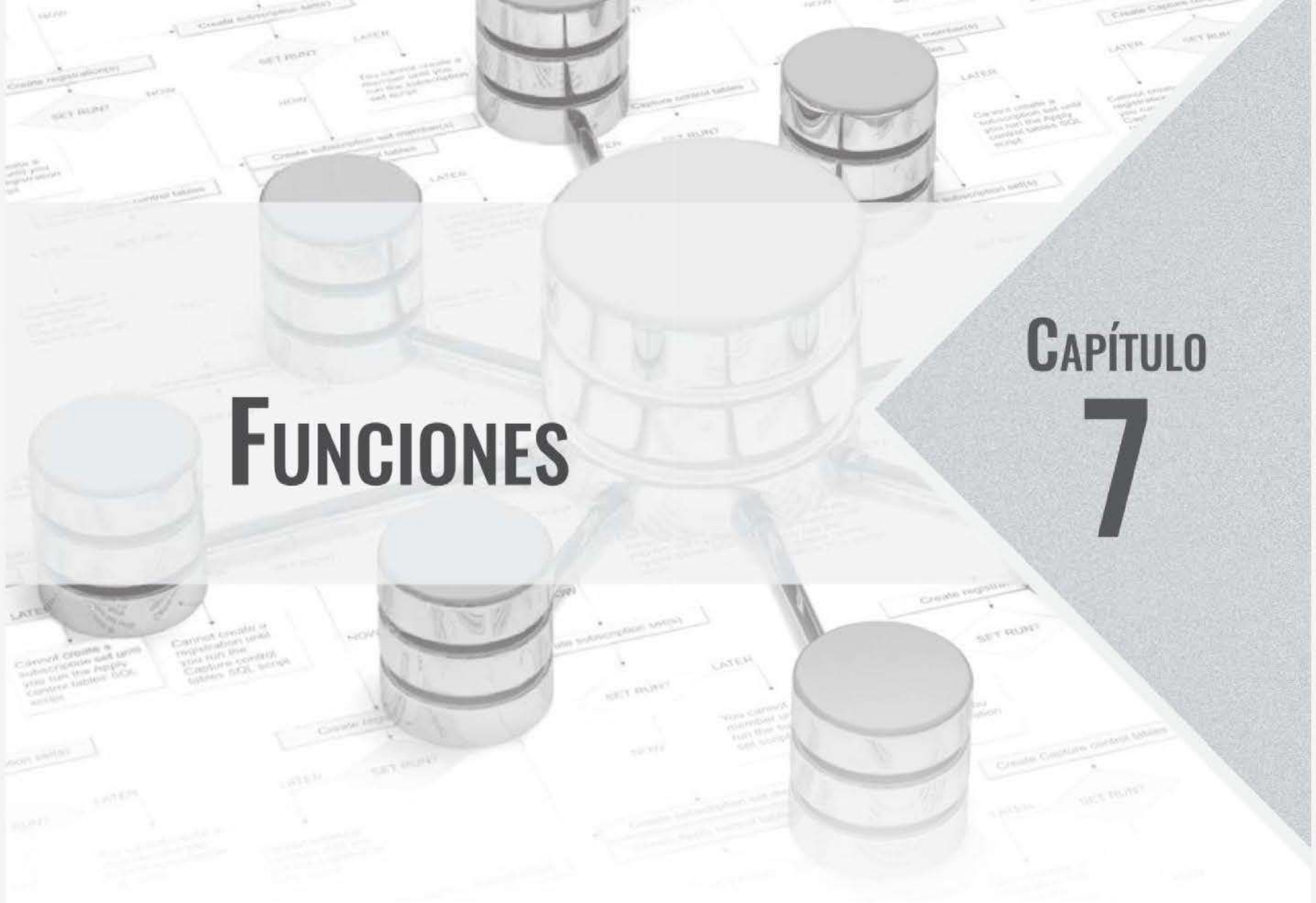
--•    Procedimiento almacenado que permita listar los clientes de un
determinado distrito.
IF OBJECT_ID('SP_CLIENTESxDISTRITO') IS NOT NULL
       DROP PROC SP_CLIENTESxDISTRITO
GO
CREATE PROC SP_CLIENTESxDISTRITO(@DISTRITO VARCHAR(30))
AS
       SELECT C.IDE_CLI AS CODIGO, C.APE_CLI AS APELLIDOS,
              C.NOM_CLI AS NOMBRES, C.DNI_CLI AS DNI,
              C.TEL_CLI AS TELEFONO,
              D.DES_DIS AS DISTRITO,
              C.COR_CLI AS CORREO
              FROM CLIENTE C
              JOIN DISTRITO D ON C.IDE_DIS=D.IDE_DIS
              WHERE D.DES_DIS=@DISTRITO
GO
--PRUEBA
SP_CLIENTESxDISTRITO 'LOS OLIVOS'
GO

--•    Procedimiento almacenado que permita mostrar información del
detalle de  alquiler, de acuerdo al número de alquiler.
IF OBJECT_ID('SP_DETALLEALQUILER') IS NOT NULL
       DROP PROC SP_DETALLEALQUILER
GO
CREATE PROC SP_DETALLEALQUILER (@NUMERO INT)
AS
       SELECT D.NUM_ALQ AS ALQUILER,
              C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
              D.MAT_AUT AS MATRICULA
              FROM DETALLEALQUILER D
              JOIN CLIENTE C ON D.IDE_CLI=C.IDE_CLI
              WHERE D.NUM_ALQ=@NUMERO
GO
--PRUEBA
SP_DETALLEALQUILER '2'
GO
```

```
--•    Procedimiento almacenado que permita mostrar los registros de
automóviles de un determinado color.
IF OBJECT_ID('SP_AUTOSxCOLOR') IS NOT NULL
       DROP PROC SP_AUTOSxCOLOR
GO
CREATE PROC SP_AUTOSxCOLOR(@COLOR VARCHAR(30))
AS
       SELECT A.MAT_AUT AS MATRICULA,
              A.COL_AUT AS COLOR,
              A.MOD_AUT AS MODELO
              FROM AUTOMOVIL A
              WHERE A.COL_AUT=@COLOR
GO
--PRUEBA
SP_AUTOSxCOLOR 'NEGRO'
GO

--•    Procedimiento almacenado que muestra el detalle de alquiler de
acuerdo al nombre de un determinado cliente.
IF OBJECT_ID('SP_DETALLExCLIENTE') IS NOT NULL
       DROP PROC SP_DETALLExCLIENTE
GO
CREATE PROC SP_DETALLExCLIENTE(@CLIENTE VARCHAR(30))
AS
       SELECT D.NUM_ALQ AS ALQUILER,
              C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
              D.MAT_AUT AS MATRICULO
              FROM CLIENTE C
              JOIN DETALLEALQUILER D ON C.IDE_CLI=D.IDE_CLI
              WHERE C.NOM_CLI+SPACE(1)+C.APE_CLI = @CLIENTE
GO
--PRUEBA
SP_DETALLExCLIENTE 'MANUEL CARLOS'
GO

--•    Procedimiento almacenado que permita listar los alquileres de un
determinado año de registro.
IF OBJECT_ID('SP_ALQUILERxAÑO') IS NOT NULL
       DROP PROC SP_ALQUILERxAÑO
GO
CREATE PROC SP_ALQUILERxAÑO(@AÑO INT)
AS
       SELECT A.NUM_ALQ AS ALQUILER,
              A.FEC_ALQ AS FECHA,
              A.MON_ALQ AS MONTO
              FROM ALQUILER A
              WHERE YEAR(A.FEC_ALQ)=@AÑO
GO
--PRUEBA:
SP_ALQUILERxAÑO 2012
GO
```



Capítulo 7

Funciones

Competencia tecnológica:

- Reconocer e implementar las principales funciones de fecha y cadena.
- Implementar sentencias que permitan controlar los datos por medio de funciones.

Contenido:

7.1 Funciones
7.2 Funciones del sistema
7.3 Funciones definidas por el usuario
- Caso desarrollado

7.1 Funciones

La solución a problemas complejos se facilita si se dividen en porciones de código llamados scripts, los cuales permiten al programador, en Transact SQL, dividir en unidades de programas o módulos que estén diseñados para una tarea específica; la ventaja de esto es que solo se escribirá una vez, pero pueden ser invocadas desde cualquier módulo de consulta, ya que estas funciones se crean dentro de una base de datos y se quedan registradas en él.

Hasta el momento se han usado varias funciones, estas tenían por misión devolver algún valor o ejecutar alguna acción; como, por ejemplo, GETDATE() en las fechas, o @@FETCH_STATUS visto en la implementación de cursores. Así, es posible encontrar muchas funciones preestablecidas. Entonces, se llega a la siguiente clasificación:

- Funciones del sistema
- Funciones definidas por el usuario
  - Funciones escalares
  - Funciones tabla en línea
  - Funciones tabla multisentencia

7.2 Funciones del sistema

SQL Server cuenta con una infinita variedad de funciones dependiendo de los valores, objetos o configuraciones que se desea realizar. Estas funciones normalmente se dividen en 4 grupos:

- Funciones de conjuntos de filas
- Funciones de agregado
- Funciones de categoría
- Funciones escalares

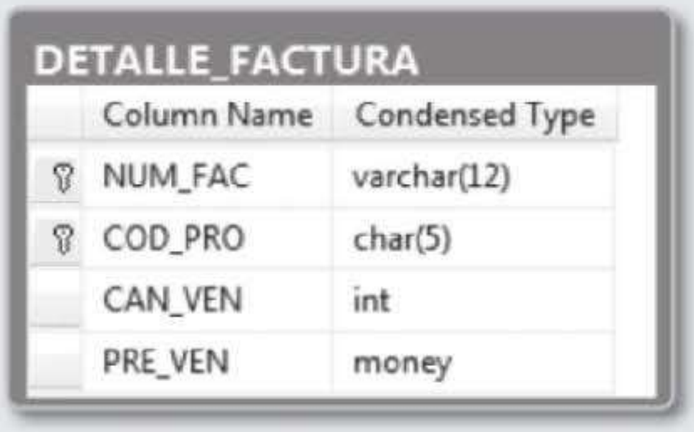
7.2.1 Funciones de conjuntos de filas

Son funciones que se caracterizan por devolver un objeto que se puede utilizar, como las referencias de tabla en una instrucción SQL.

- OPENDATASOURCE
- OPENROWSET
- OPENQUERY
- OPENXML

7.2.2 Funciones de agregado

Son funciones que se caracterizan por operar sobre una colección de valores y devolver un solo valor de resumen.

<table>
<tr><th>Función</th><th>Descripción</th></tr>
<tr><td>AVG</td><td>Determina el promedio de valores.
Script que permite determinar el promedio subtotal desde la tabla DETALLE_FACTURA.
 

SELECT AVG(D.CAN_VEN*D.PRE_VEN)
FROM DETALLE_FACTURA D
GO</td></tr>
<tr><td>MIN</td><td>Determina el mínimo valor encontrado.
Script que permite determinar el menor valor en el stock actual; luego, mostrar qué productos tienen el mismo stock que el menor valor.
 

--Determinar el mínimo stock actual
DECLARE @MENOR INT
SELECT @MENOR = MIN(P.SAC_PRO)
FROM PRODUCTO P
--Buscar los productos con igual stock al mínimo
SELECT P.COD_PRO AS CODIGO,
P.DES_PRO AS DESCRIPCION,
P.PRE_PRO AS PRECIO,
P.SAC_PRO AS [STOCK ACTUAL]
FROM PRODUCTO P
WHERE P.SAC_PRO=@MENOR
GO</td></tr>
</table>

CHECKSUM_AGG	Devuelve la suma de comprobación de los valores de un grupo; asimismo, omite los valores NULL que se presenten. Script que permite mostrar la suma de comprobación sobre la tabla DETALLE_FACTURA. SELECT CHECKSUM_AGG(CAST(CAN_VEN*PRE_VEN AS INT)) FROM DETALLE_FACTURA GO
SUM	Devuelve la suma acumulada de los valores numéricos de un campo de una tabla. Script que permite mostrar el monto subtotal de las facturas registradas: SELECT YEAR(F.FEC_FAC) AS AÑO,SUM(DF.CAN_VEN*DF.PRE_VEN) AS [SUBTOTAL POR AÑO] FROM DETALLE_FACTURA DF JOIN FACTURA F ON DF.NUM_FAC=F.NUM_FAC GROUP BY YEAR(F.FEC_FAC) GO
COUNT	Devuelve el total de registros de una tabla bajo un determinado criterio. Script que permite mostrar el total de facturas registradas por un determinado vendedor. SELECT V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR, COUNT(*) AS TOTAL FROM VENDEDOR V JOIN FACTURA F ON V.COD_VEN=F.COD_VEN GROUP BY V.NOM_VEN+SPACE(1)+V.APE_VEN GO
STDEV	Devuelve la desviación estándar. Script que permite mostrar el valor de la desviación estándar del campo cantidad desde la tabla DETALLE DE FACTURA. SELECT STDEV(D.CAN_VEN) AS [DESVIACION ESTANDAR] FROM DETALLE_FACTURA D GO
COUNT_BIG	Devuelve el total de elementos de un grupo. Tiene la misma funcionalidad que la función COUNT, solo que siempre devuelve un valor de tipo bigint. Script que determina el total de productos registrados. SELECT COUNT_BIG(P.COD_PRO) AS TOTAL FROM PRODUCTO P GO
STDEVP	Devuelve la desviación estándar poblacional. Script que permite mostrar el valor de la desviación estándar del campo cantidad desde la tabla DETALLE DE FACTURA. SELECT STDEVP(D.CAN_VEN) AS [DESVIACION ESTANDAR] FROM DETALLE_FACTURA D GO

GROUPING

Devuelve valores cero y uno. Cero cuando el valor no es agregado y uno cuando es agregado dentro de una consulta agrupada y con especificación de Rollup o Cube.

Script que permite determinar el total de facturas registradas por año y mes; además, añadir una columna llamada "tipo fila", la cual mostrará qué tipo de valor es cada fila del resultado.

```
AÑO         MES         Total       TipoFila
----------- ----------- ----------- -------------
2012        1           1           Fila Normal
2013        1           6           Fila Normal
NULL        1           7           Total Mes
2013        3           2           Fila Normal
NULL        3           2           Total Mes
2013        4           1           Fila Normal
NULL        4           1           Total Mes
2013        5           1           Fila Normal
NULL        5           1           Total Mes
2013        6           6           Fila Normal
NULL        6           6           Total Mes
2013        7           1           Fila Normal
NULL        7           1           Total Mes
2012        8           1           Fila Normal
NULL        8           1           Total Mes
2013        10          1           Fila Normal
NULL        10          1           Total Mes
NULL        NULL        20          Total General
2012        NULL        2           Total Año
2013        NULL        18          Total Año

(20 row(s) affected)
```

```
SELECT YEAR(F.FEC_FAC) AS AÑO, MONTH(F.FEC_FAC) AS MES, COUNT(*) AS Total,
  CASE
    WHEN GROUPING(YEAR(F.FEC_FAC))=0 AND GROUPING(MONTH(F.FEC_FAC))=1
THEN 'Total Año'
    WHEN GROUPING(YEAR(F.FEC_FAC))=1 AND GROUPING(MONTH(F.FEC_FAC))=0
THEN 'Total Mes'
    WHEN GROUPING(YEAR(F.FEC_FAC))=1 AND GROUPING(MONTH(F.FEC_FAC))=1
THEN 'Total General'
    ELSE 'Fila Normal'
  END AS TipoFila
FROM FACTURA F
GROUP BY CUBE(YEAR(F.FEC_FAC), MONTH(F.FEC_FAC))
GO
```

VAR

Devuelve la varianza estadística.

Script que permite mostrar el valor de la varianza estadística del campo cantidad desde la tabla DETALLE DE FACTURA.

```
SELECT VAR(D.CAN_VEN) AS [DESVIACION ESTANDAR]
            FROM DETALLE_FACTURA D
GO
```

GROUPING_ID

Tiene la misma funcionalidad que la función GROUPING.

Script que permite determinar el total de facturas registradas por año y mes; además, añadir una columna llamada "tipo fila", la cual mostrará qué tipo de valor es cada fila del resultado.

```
AÑO         MES         Total       TipoFila
----------- ----------- ----------- -------------
2012        1           1           Fila Normal
2013        1           6           Fila Normal
NULL        1           7           Total Mes
2013        3           2           Fila Normal
NULL        3           2           Total Mes
2013        4           1           Fila Normal
NULL        4           1           Total Mes
2013        5           1           Fila Normal
NULL        5           1           Total Mes
2013        6           6           Fila Normal
NULL        6           6           Total Mes
2013        7           1           Fila Normal
NULL        7           1           Total Mes
2012        8           1           Fila Normal
NULL        8           1           Total Mes
2013        10          1           Fila Normal
NULL        10          1           Total Mes
NULL        NULL        20          Total General
2012        NULL        2           Total Año
2013        NULL        18          Total Año

(20 row(s) affected)
```

```
SELECT YEAR(F.FEC_FAC) AS AÑO,
     MONTH(F.FEC_FAC) AS MES, COUNT(*) AS Total,
  CASE
    WHEN GROUPING(YEAR(F.FEC_FAC))=0 AND
       GROUPING(MONTH(F.FEC_FAC))=1 THEN 'Total Año'
    WHEN GROUPING(YEAR(F.FEC_FAC))=1 AND
       GROUPING(MONTH(F.FEC_FAC))=0 THEN 'Total Mes'
    WHEN GROUPING(YEAR(F.FEC_FAC))=1 AND
       GROUPING(MONTH(F.FEC_FAC))=1 THEN 'Total General'
    ELSE 'Fila Normal'
  END AS TipoFila
FROM FACTURA F
GROUP BY CUBE(YEAR(F.FEC_FAC), MONTH(F.FEC_FAC))
GO
```

VARP

Devuelve la varianza estadística poblacional.

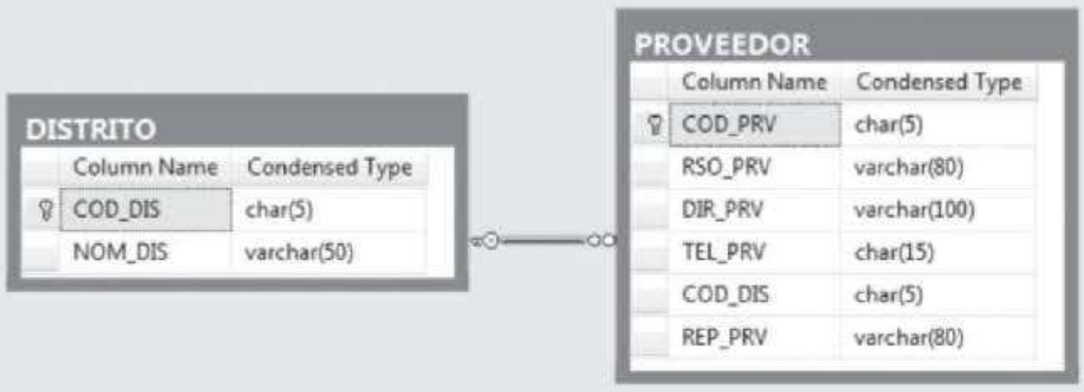
Script que permite mostrar el valor de la varianza estadística poblacional del campo cantidad desde la tabla DETALLE DE FACTURA.

```
SELECT VARP(D.CAN_VEN) AS [DESVIACION ESTANDAR]
            FROM DETALLE_FACTURA D
GO
```

MAX

Devuelve el valor máximo.

Script que permite mostrar los distritos con el máximo número de proveedores registrados.




```
--Obtener el máximo valor
DECLARE @MAXIMO INT
SELECT @MAXIMO = MAX(X.[TOTAL PROVEEDORES])
            FROM (    SELECT D.NOM_DIS AS DISTRITO,
                      COUNT(*) AS [TOTAL PROVEEDORES]
                      FROM DISTRITO D
                      JOIN PROVEEDOR P ON D.COD_DIS=P.COD_DIS
                      GROUP BY D.NOM_DIS
                      ) AS X

--Mostrar los distritos con máximo número de proveedores
SELECT X.DISTRITO,X.[TOTAL PROVEEDORES]
            FROM (    SELECT D.NOM_DIS AS DISTRITO,
                      COUNT(*) AS [TOTAL PROVEEDORES]
                      FROM DISTRITO D
                      JOIN PROVEEDOR P ON D.COD_DIS=P.COD_DIS
                      GROUP BY D.NOM_DIS
                      ) AS X
          WHERE X.[TOTAL PROVEEDORES]=@MAXIMO
GO
```

7.2.3 Funciones de categoría

Son funciones que se caracterizan por devolver un valor de categoría para cada fila de una partición.

Función	Descripción
RANK	Clasifica los elementos de un grupo en posiciones tipo ranking: primero, segundo, tercero. Si hay datos con el mismo valor, se colocan dentro de la misma posición, como si se tratase de un empate; pero el inmediato inferior tendrá posición en base al número de elementos existentes. Script que permite listar los productos mostrando la unidad, la descripción, el precio del producto y la escala que esta ocupa en base al precio más alto por unidad.

```
UNIDAD   DESCRIPCION                PRECIO                ESCALA
-------------------------------------------------------------------
CIE      FOLDER PLASTICO A-4        52.00                 1
CIE      FOLDER MANILA OFICIO       22.00                 2
CIE      FOLDER MANILA A-4          22.00                 2
CIE      SOBRE MANILA A-4           20.00                 4
CIE      SOBRE MANILA OFICIO        17.00                 5
DOC      TAJADOR METAL              22.00                 1
DOC      TAJADOR PLASTICO           14.00                 2
DOC      BORRADOR DE TINTA          12.00                 3
DOC      LAPICERO NEGRO             12.00                 3
DOC      LAPICERO AZUL              12.00                 3
DOC      LAPICERO ROJO              10.00                 6
DOC      CUADERNO ATLAS             7.00                  7
DOC      CUADERNO JUSTUS            5.50                  8
DOC      BORRADOR BLANCO            5.00                  9
MLL      PAPEL BOND OFICIO          37.00                 1
MLL      PAPEL BOND A-4 ATLAS       18.50                 2
MLL      PAPEL BULKY                12.00                 3
MLL      PAPEL PERIODICO 75 GRAMOS  11.00                 4
MLL      REGLA FLEXIBLE             10.00                 5
MLL      REGLA 30CM                 5.00                  6
MLL      LAPICERO 075 FINE          2.00                  7
MLL      REGLA 20CM                 2.00                  7
MLL      LAPICERO 050 FINE          1.50                  9
MLL      LAPICERO 035 FINE          1.00                  10
UNI      PROTECTOR DE PANTALLA      52.00                 1
UNI      CARTUCHO TINTA COLOR       47.00                 2
UNI      CAJA DE DISKETTES * 10     32.00                 3
UNI      CARTUCHERA MARCA           30.00                 4
UNI      CARTUCHO TINTA NEGRA       21.00                 5
UNI      CARTUCHERA BASICA          10.00                 6
UNI      PORTA DISKETTES            5.50                  7

(31 row(s) affected)
```

```
SELECT P.UNI_PRO AS UNIDAD, P.DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,
       RANK () OVER(PARTITION BY P.UNI_PRO
                          ORDER BY P.PRE_PRO DESC)  AS ESCALA
       FROM PRODUCTO P
GO
```

NTILE

Distribuye las filas de una partición ordenada en un número especificado de grupos. Los grupos se numeran a partir de uno. Para cada fila, NTILE devuelve el número del grupo al que pertenece la fila.

Script que permite mostrar el nivel de grupo de la tabla PRODUCTO en base al orden descendente de los precios y al tipo de unidad.

```
UNIDAD                         DESCRIPCION                                        PRECIO                GRUPO
------------------------------ -------------------------------------------------- --------------------- -------
UNI                            CARTUCHO TINTA NEGRA                               21.00                 1
UNI                            CARTUCHO TINTA COLOR                               47.00                 1
UNI                            PORTA DISKETTES                                    5.50                  1
UNI                            CAJA DE DISKETTES * 10                             32.00                 1
UNI                            PROTECTOR DE PANTALLA                              52.00                 1
UNI                            CARTUCHERA BASICA                                  10.00                 1
UNI                            CARTUCHERA MARCA                                   30.00                 1
MLL                            LAPICERO 035 FINE                                  1.00                  1
MLL                            LAPICERO 050 FINE                                  1.50                  2
MLL                            LAPICERO 075 FINE                                  2.00                  2
MLL                            REGLA 20CM                                         2.00                  2
MLL                            REGLA 30CM                                         5.00                  2
MLL                            REGLA FLEXIBLE                                     10.00                 2
MLL                            PAPEL BOND A-4 ATLAS                               18.50                 2
MLL                            PAPEL BOND OFICIO                                  37.00                 2
MLL                            PAPEL BULKY                                        12.00                 2
MLL                            PAPEL PERIODICO 75 GRAMOS                          11.00                 3
DOC                            LAPICERO NEGRO                                     12.00                 3
DOC                            LAPICERO AZUL                                      12.00                 3
DOC                            LAPICERO ROJO                                      10.00                 3
DOC                            BORRADOR DE TINTA                                  12.00                 3
DOC                            BORRADOR BLANCO                                    5.00                  3
DOC                            TAJADOR METAL                                      22.00                 3
DOC                            TAJADOR PLASTICO                                   14.00                 3
DOC                            CUADERNO JUSTUS                                    5.50                  4
DOC                            CUADERNO ATLAS                                     7.00                  4
CIE                            FOLDER MANILA OFICIO                               22.00                 4
CIE                            FOLDER MANILA A-4                                  22.00                 4
CIE                            SOBRE MANILA OFICIO                                17.00                 4
CIE                            SOBRE MANILA A-4                                   20.00                 4
CIE                            FOLDER PLASTICO A-4                                52.00                 4

(31 row(s) affected)
```

```
SELECT P.UNI_PRO AS UNIDAD, P.DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,
       NTILE(4) OVER(ORDER BY P.UNI_PRO DESC) AS GRUPO
       FROM PRODUCTO P
GO
```

DENSE_RANK

Clasifica los elementos de un grupo en posiciones tipo *ranking*: primero, segundo, tercero. Si hay datos con el mismo valor, se colocan dentro de la misma posición, como si se tratase de un empate; pero el inmediato inferior tendrá posición siguiente en el *ranking*.

Script que permite listar los productos mostrando la unidad, la descripción, el precio del producto y la escala que esta ocupa en base al precio más alto por unidad.

```
UNIDAD   DESCRIPCION                    PRECIO     ESCALA
-------------------------------------------------------------
CIE      FOLDER PLASTICO A-4            52.00      1
CIE      FOLDER MANILA OFICIO           22.00      2
CIE      FOLDER MANILA A-4              22.00      2
CIE      SOBRE MANILA A-4               20.00      3
CIE      SOBRE MANILA OFICIO            17.00      4
DOC      TAJADOR METAL                  22.00      1
DOC      TAJADOR PLASTICO               14.00      2
DOC      BORRADOR DE TINTA              12.00      3
DOC      LAPICERO NEGRO                 12.00      3
DOC      LAPICERO AZUL                  12.00      3
DOC      LAPICERO ROJO                  10.00      4
DOC      CUADERNO ATLAS                 7.00       5
DOC      CUADERNO JUSTUS                5.50       6
DOC      BORRADOR BLANCO                5.00       7
MLL      PAPEL BOND OFICIO              37.00      1
MLL      PAPEL BOND A-4 ATLAS           18.50      2
MLL      PAPEL BULKY                    12.00      3
MLL      PAPEL PERIODICO 75 GRAMOS      11.00      4
MLL      REGLA FLEXIBLE                 10.00      5
MLL      REGLA 30CM                     5.00       6
MLL      LAPICERO 075 FINE              2.00       7
MLL      REGLA 20CM                     2.00       7
MLL      LAPICERO 050 FINE              1.50       8
MLL      LAPICERO 035 FINE              1.00       9
UNI      PROTECTOR DE PANTALLA          52.00      1
UNI      CARTUCHO TINTA COLOR           47.00      2
UNI      CAJA DE DISKETTES * 10         32.00      3
UNI      CARTUCHERA MARCA               30.00      4
UNI      CARTUCHO TINTA NEGRA           21.00      5
UNI      CARTUCHERA BASICA              10.00      6
UNI      PORTA DISKETTES                5.50       7

(31 row(s) affected)
```

```
SELECT P.UNI_PRO AS UNIDAD, P.DES_PRO AS DESCRIPCION,
       PRE_PRO AS PRECIO,
       DENSE_RANK () OVER(PARTITION BY P.UNI_PRO
                          ORDER BY P.PRE_PRO DESC)  AS ESCALA
       FROM PRODUCTO P
GO
```

ROW_NUMBER Devuelve un número secuencial de una fila dentro de una partición de un conjunto de resultados.

Script que permite mostrar el número correlativo correspondiente a los productos, dependiendo del orden ascendente del código de producto.

```
NUMERO  CODIGO DESCRIPCION                          PRECIO
------- ------ ------------------------------------ -------
1       P001   PAPEL BOND A-4 ATLAS                 18.50
2       P002   PAPEL BOND OFICIO                    37.00
3       P003   PAPEL BULKY                          12.00
4       P004   PAPEL PERIODICO 75 GRAMOS            11.00
5       P005   CARTUCHO TINTA NEGRA                 21.00
6       P006   CARTUCHO TINTA COLOR                 47.00
7       P007   PORTA DISKETTES                      5.50
8       P008   CAJA DE DISKETTES * 10               32.00
9       P009   BORRADOR DE TINTA                    12.00
10      P010   BORRADOR BLANCO                      5.00
11      P011   TAJADOR METAL                        22.00
12      P012   TAJADOR PLASTICO                     14.00
13      P013   FOLDER MANILA OFICIO                 22.00
14      P014   FOLDER MANILA A-4                    22.00
15      P015   SOBRE MANILA OFICIO                  17.00
16      P016   SOBRE MANILA A-4                     20.00
17      P017   LAPICERO NEGRO                       12.00
18      P018   LAPICERO AZUL                        12.00
19      P019   LAPICERO ROJO                        10.00
20      P020   FOLDER PLASTICO A-4                  52.00
21      P021   PROTECTOR DE PANTALLA                52.00
22      P022   CUADERNO JUSTUS                      5.50
23      P023   CUADERNO ATLAS                       7.00
24      P024   LAPICERO 035 FINE                    1.00
25      P025   LAPICERO 050 FINE                    1.50
26      P026   LAPICERO 075 FINE                    2.00
27      P027   REGLA 20CM                           2.00
28      P028   REGLA 30CM                           5.00
29      P029   REGLA FLEXIBLE                       10.00
30      P030   CARTUCHERA BASICA                    10.00
31      P031   CARTUCHERA MARCA                     30.00

(31 row(s) affected)
```

7.2.4 Funciones escalares

Son funciones que se caracterizan por operar sobre un valor y después devolver otro valor. Las funciones escalares se pueden utilizar donde la expresión sea válida.

- **Funciones de configuración:** Devuelven información acerca de la configuración actual.

Función	Descripción

@@SERVERNAME	Muestra el nombre del servidor.
	SELECT @@SERVERNAME
	`----------`
	`USER-PC\PC`
	`(1 row(s) affected)`

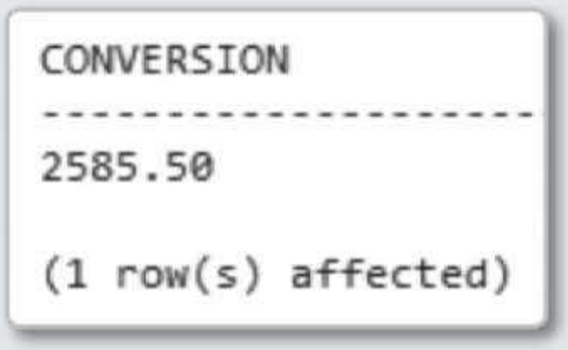
@@LANGUAGE	Muestra el idioma configurado en el servidor.
	SELECT @@LANGUAGE
	`----------`
	`us_english`
	`(1 row(s) affected)`

@@SERVICENAME	Muestra el nombre de la instancia del servidor SQL.
	SELECT @@SERVICENAME
	`----------`
	`PC`
	`(1 row(s) affected)`

@@VERSION	Muestra la versión de SQL Server actual.
	SELECT @@VERSION
	`Microsoft SQL Server 2016 - 13.0.2218.0 (X64)`
	`Jun 16 2016 13:05:25`
	`Copyright (c) Microsoft Corporation`
	`Enterprise Edition: Core-based Licensing (64-bit) on Windows NT 6.1 <X64> (Build 7601: Service Pack 1)`
	`(1 row(s) affected)`

- **Funciones de conversión:** Admiten conversión y conversión de tipos de datos.

Función	Descripción
CAST – CONVERT	Realizan conversiones de tipos; la diferencia entre CAST y CONVERT es la forma cómo especifican sus parámetros. Script que permite imprimir un total de clientes por distrito. --Usando CAST DECLARE @TOTAL INT SELECT @TOTAL=COUNT(*) FROM DISTRITO D JOIN CLIENTE C ON D.COD_DIS=C.COD_DIS WHERE D.NOM_DIS='SAN MIGUEL' PRINT 'TOTAL: '+CAST(@TOTAL AS CHAR(5)) GO --Usando CONVERT DECLARE @TOTAL INT SELECT @TOTAL=COUNT(*) FROM DISTRITO D JOIN CLIENTE C ON D.COD_DIS=C.COD_DIS WHERE D.NOM_DIS='SAN MIGUEL' PRINT 'TOTAL: '+CONVERT(CHAR(5),@TOTAL) GO
PARSE	Se recomienda su uso para convertir tipos de cadena a tipos de fecha y hora, y a número. Script que permite convertir un valor monetario expresado en caracteres a tipo moneda. CONVERSION --- 2585.50 (1 row(s) affected) DECLARE @MONTO VARCHAR(10) = '2585,50' SELECT PARSE(@MONTO AS money USING 'es-ES') AS CONVERSION GO

TRY_CAST	Devuelve una conversión de valor al tipo de datos especificado si la conversión se realiza correctamente; de lo contrario, devuelve NULL. Script que permite mostrar el valor de una fecha a un tipo de fecha y hora corta.  CONVERSION --- 2010-12-31 00:00:00 (1 row(s) affected) SELECT TRY_CAST('12/31/2010' AS SMALLDATETIME) AS CONVERSION GO
TRY_CONVERT	Devuelve una conversión de valor al tipo de datos especificado si la conversión se realiza correctamente; de lo contrario, devuelve NULL. Script que permite mostrar el valor de una fecha a un tipo de fecha y hora corta.  CONVERSION --- 2010-12-31 00:00:00 (1 row(s) affected) SELECT TRY_CONVERT(SMALLDATETIME,'12/31/2010') AS CONVERSION GO
TRY_PARSE	Se recomienda su uso para convertir tipos de cadena a tipos de fecha y hora, y a número. Script que permite convertir un valor monetario expresado en caracteres a tipo moneda.  CONVERSION --- 2585.50 (1 row(s) affected) DECLARE @MONTO VARCHAR(10) = '2585,50' SELECT TRY_PARSE(@MONTO AS money USING 'es-ES') AS CONVERSION GO

- **Funciones del cursor**: Devuelven información acerca de los cursores.

Función	Descripción
@@CURSOR_ROWS	Devuelve el número de filas certificadas que se encuentran en el último cursor abierto en la conexión. Valores que emite: • -1: El cursor es dinámico. Como los cursores dinámicos reflejan todos los cambios, el número de filas correspondientes al cursor cambia constantemente. Nunca se puede afirmar que se han recuperado todas las filas que correspondan. • 0: No se han abierto cursores, no hay filas calificadas para el último cursor abierto, o este se ha cerrado o su asignación se ha cancelado. • N: El cursor está completamente relleno. El valor devuelto (n) es el número total de filas del cursor.
CURSOR_STATUS	Determina si un procedimiento devuelve un cursor y un conjunto de resultados para un parámetro dado. Valores que emite: • 1: El conjunto de resultados del cursor tiene al menos una fila. • 0: El conjunto de resultados del cursor está vacío. • -1: El cursor está cerrado. • -2: No aplicable. • -3: No existe ningún cursor con el nombre indicado.
@@FETCH_STATUS	Devuelve el estado de la última instrucción FETCH del cursor emitida contra cualquier cursor abierto actualmente por la conexión. Valores que emite: • 0: La instrucción FETCH se ejecutó correctamente. • -1: La instrucción FETCH no se ejecutó correctamente o la fila estaba más allá del conjunto de resultados. • -2: Falta la fila capturada. • -9: El cursor no está realizando una operación de búsqueda.

- **Tipos de datos y funciones de fecha y hora**: Llevan a cabo operaciones sobre un valor de entrada de fecha y hora, y devuelven un valor numérico, de cadena o de fecha y hora.

Función	Descripción

SYSDATETIME	Devuelve el valor actual de la fecha y hora en el formato DateTime2.
	Script que permite mostrar la fecha y hora actual con el formato DateTime2.
	`----------`
	`2017-03-15 11:46:01.8572762`
	`(1 row(s) affected)`
	`SELECT SYSDATETIME()`
	`GO`

SYSUTCDATETIME

Devuelve el valor actual de la fecha y hora universal en el formato DateTime2.
Script que permite mostrar la fecha y hora actual con el formato DateTime2.

```
----------------------------
2017-03-15 16:48:31.0538098

(1 row(s) affected)
```

```
SELECT SYSUTCDATETIME()
GO
```

CURRENT_TIMESTAMP

Devuelve el valor actual de la fecha y hora en el formato DateTime.
Script que permite mostrar la fecha y hora actual con el formato DateTime.

```
-----------------------
2017-03-15 12:04:46.157

(1 row(s) affected)
```

```
SELECT CURRENT_TIMESTAMP
GO
```

GETUTCDATE

Devuelve el valor actual de la fecha y hora en el formato DateTime universal.
Script que permite mostrar la fecha y hora actual con el formato DateTime.

```
-----------------------
2017-03-15 17:08:22.670

(1 row(s) affected)
```

```
SELECT GETUTCDATE()
GO
```

DATENAME

Devuelve una cadena de caracteres que representa el parámetro especificado.
Los parámetros se muestran de la siguiente manera:

year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

Script que devuelve el mes en letras de la fecha actual.

```
SELECT DATENAME(MM,GETDATE())
GO
```

DAY

Devuelve el valor numérico del día de una fecha determinada.

Script que muestra el número del día de la fecha actual.

```
DIA ACTUAL
-----------
15

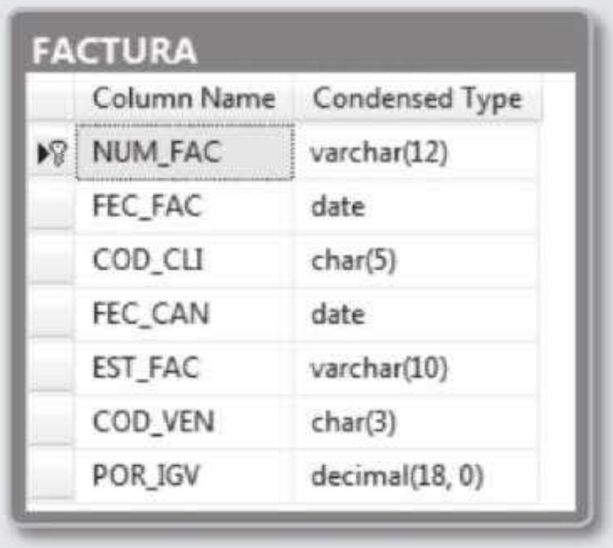
(1 row(s) affected)
```

```
SELECT DAY(GETDATE()) AS [DIA ACTUAL]
GO
```

YEAR

Devuelve el valor numérico del año de una fecha determinada.

Script que permite mostrar los años en que se registraron las facturas.




```
SELECT F.NUM_FAC AS FACTURA,
       F.FEC_FAC AS FECHA,
       YEAR(F.FEC_FAC) AS [AÑO DE REGISTRO]
       FROM FACTURA F
GO
```

```
FACTURA      FECHA      AÑO DE REGISTRO
------------ ---------- ---------------
FA001        2013-06-07 2013
FA002        2013-06-07 2013
FA003        2013-01-09 2013
FA004        2013-06-09 2013
FA005        2013-01-10 2013
FA006        2013-01-08 2013
FA007        2013-05-10 2013
FA008        2013-04-10 2013
FA009        2013-03-10 2013
FA010        2013-01-10 2013
```

DATEDIFF

Devuelve un valor numérico que representa la diferencia entre dos fechas.

Script que permite mostrar la cantidad de años que tiene registrada cada factura con respecto a la fecha actual.

FACTURA

Column Name	Condensed Type
NUM_FAC	varchar(12)
FEC_FAC	date
COD_CLI	char(5)
FEC_CAN	date
EST_FAC	varchar(10)
COD_VEN	char(3)
POR_IGV	decimal(18, 0)

```
SELECT F.NUM_FAC AS FACTURA,
       F.FEC_FAC AS FECHA,
       DATEDIFF(YY,F.FEC_FAC,GETDATE()) AS [AÑOS]
       FROM FACTURA F
GO
```

```
FACTURA     FECHA      AÑOS
----------- ---------- -----
FA001       2013-06-07 4
FA002       2013-06-07 4
FA003       2013-01-09 4
FA004       2013-06-09 4
FA005       2013-01-10 4
FA006       2013-01-08 4
FA007       2013-05-10 4
FA008       2013-04-10 4
FA009       2013-03-10 4
FA010       2013-01-10 4
FA011       2013-10-09 4
FA012       2013-01-12 4
FA013       2013-01-12 4
```

EOMONTH

Devuelve el último día del mes que contiene una determinada fecha.

Script que permite mostrar el número de factura, la fecha y el último día del mes con respecto a la fecha de facturación.

```
FACTURA     FECHA      ULTIMO DIA DEL MES FACTURADO
----------- ---------- ----------------------------
FA001       2013-06-07 2013-06-30
FA002       2013-06-07 2013-06-30
FA003       2013-01-09 2013-01-31
FA004       2013-06-09 2013-06-30
FA005       2013-01-10 2013-01-31
FA006       2013-01-08 2013-01-31
FA007       2013-05-10 2013-05-31
FA008       2013-04-10 2013-04-30
FA009       2013-03-10 2013-03-31
FA010       2013-01-10 2013-01-31
FA011       2013-10-09 2013-10-31
FA012       2013-01-12 2013-01-31
FA013       2013-01-12 2013-01-31
FA014       2012-01-12 2012-01-31
```

```
SELECT F.NUM_FAC AS FACTURA,
       F.FEC_FAC AS FECHA,
       EOMONTH(F.FEC_FAC)
      AS [ULTIMO DIA DEL MES FACTURADO]
       FROM FACTURA F
GO
```

SET DATEFORMAT

Define el formato de la fecha que se especifica en consultas o inserciones de registros. Se tiene las siguientes opciones:

```
SET DATEFORMAT DMY
SET DATEFORMAT YMD
SET DATEFORMAT DYM
```

Script que permite configurar el formato de la fecha de modo día, mes y año.

```
SET DATEFORMAT DMY
GO
```

GETDATE

Devuelve el valor de la fecha actual.

Script que muestra el número del día de la fecha actual.

```
FECHA ACTUAL
-----------------------
2017-03-15 12:29:45.980

(1 row(s) affected)
```

```
SELECT GETDATE() AS [FECHA ACTUAL]
GO
```

DATEPART

Devuelve un valor numérico correspondiente a una parte de la fecha. Los parámetros se muestran de la siguiente manera:

year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

Script que devuelve el mes de la fecha actual.

```
SELECT DATEPART(MM,GETDATE()) AS [MES ACTUAL]
GO
```

MONTH

Devuelve el valor numérico del mes de una fecha determinada.

Script que muestra el número del mes de la fecha actual.

```
MES ACTUAL
-----------
3

(1 row(s) affected)
```

```
SELECT MONTH(GETDATE()) AS [MES ACTUAL]
GO
```

DATEADD

Permite aumentar o disminuir valores en una fecha determinada.

Script que permite mostrar las diez fechas posteriores, de forma que los pagos se realicen de mes a mes.

```
LISTADO DE FECHAS DE PAGO
FECHA ACTUAL: 2017-03-15
--------------------------
2017-04-15
2017-05-15
2017-06-15
2017-07-15
2017-08-15
2017-09-15
2017-10-15
2017-11-15
2017-12-15
2018-01-15
```

```
DECLARE @FECHA DATE = GETDATE()
DECLARE @I INT = 1
PRINT 'LISTADO DE FECHAS DE PAGO'
PRINT 'FECHA ACTUAL: '+CAST(@FECHA AS VARCHAR(10))
PRINT '_____'
WHILE @I<=10
BEGIN
PRINT DATEADD(MONTH,@I,@FECHA)
SET @I+=1
END
GO
```

- **Funciones lógicas:** Realizan operaciones lógicas.

Función	Descripción
CHOOSE	Devuelve un valor de un conjunto de valores especificado por un índice. Script que permite mostrar los datos de las facturas, adicionando el día, mes en letras y el año desde la fecha de facturación.

```
FACTURA     FECHA      DIA         MES       AÑO
----------- ---------- ----------- --------- --------
FA001       2013-06-07 7           JUNIO     2013
FA002       2013-06-07 7           JUNIO     2013
FA003       2013-01-09 9           ENERO     2013
FA004       2013-06-09 9           JUNIO     2013
FA005       2013-01-10 10          ENERO     2013
FA006       2013-01-08 8           ENERO     2013
FA007       2013-05-10 10          MAYO      2013
FA008       2013-04-10 10          ABRIL     2013
FA009       2013-03-10 10          MARZO     2013
FA010       2013-01-10 10          ENERO     2013
FA011       2013-10-09 9           OCTUBRE   2013
FA012       2013-01-12 12          ENERO     2013
FA013       2013-01-12 12          ENERO     2013
FA014       2012-01-12 12          ENERO     2012
FA015       2012-08-12 12          AGOSTO    2012
FA016       2013-06-01 1           JUNIO     2013
FA017       2013-06-01 1           JUNIO     2013
FA018       2013-03-02 2           MARZO     2013
FA019       2013-07-02 2           JULIO     2013
FA020       2013-06-02 2           JUNIO     2013
FA021       2013-06-02 2           JUNIO     2013

(21 row(s) affected)
```

```
SELECT F.NUM_FAC AS FACTURA,
       F.FEC_FAC AS FECHA,
       DAY(F.FEC_FAC) AS DIA,
       CHOOSE(MONTH(F.FEC_FAC),'ENERO','FEBRERO','MARZO',
           'ABRIL','MAYO','JUNIO',
               'JULIO','AGOSTO','SETIEMBRE',
               'OCTUBRE','NOVIEMBRE','DICIEMBRE') AS MES,
       YEAR(F.FEC_FAC) AS AÑO
       FROM FACTURA F
GO
```

IIF

Permite devolver uno de dos valores, dependiendo de la evaluación de una condición.
Script que permite mostrar la descripción del tipo de vendedor: (1) Corporativo, (2) Por venta.

```
CODIGO VENDEDOR              SUELDO      TIPO        DESCRIPCION TIPO
------ --------------------------------- ----------- ----------------
V01    JUANA ALVA            1000.00     1           CORPORATIVO
V02    JUAN SOTO             1200.00     2           POR VENTA
V03    CARLOS AREVALO        1500.00     2           POR VENTA
V04    CESAR OJEDA           850.00      1           CORPORATIVO
V05    JULIO VEGA            1500.00     1           CORPORATIVO
V06    ANA ORTEGA            1200.00     1           CORPORATIVO
V07    JOSE PALACIOS         2500.00     1           CORPORATIVO
V08    RUBEN GOMEZ           1450.00     2           POR VENTA
V09    PATRICIA ARCE         1800.00     2           POR VENTA
V10    RENATO PEREZ          1550.00     2           POR VENTA
(10 row(s) affected)
```

```
SELECT  V.COD_VEN AS CODIGO,
        V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
        V.SUE_VEN AS SUELDO,
        V.TIP_VEN AS TIPO,
        IIF(TIP_VEN=1,'CORPORATIVO','POR VENTA')
              AS [DESCRIPCION TIPO]
        FROM VENDEDOR V
GO
```

- **Funciones matemáticas:** Realizan cálculos basados en valores de entrada, proporcionados como parámetros a las funciones, y devuelven valores numéricos.
  - ABS
  - DEGREES
  - RAND
  - ACOS
  - EXP
  - ROUND
  - ASIN
  - FLOOR
  - SIGN
  - ATAN
  - LOG
  - SIN
  - ATN2
  - LOG10
  - SQRT
  - CEILING
  - PI
  - SQUARE
  - COS
  - POWER
  - TAN
  - COT
  - RADIANS

- **Funciones de metadatos:** Devuelven información acerca de la base de datos y los objetos de la base de datos.
  - @@PROCID
  - INDEX_COL
  - APP_NAME
  - INDEXKEY_PROPERTY
  - APPLOCK_MODE
  - INDEXPROPERTY
  - APPLOCK_TEST
  - NEXT VALUE FOR
  - ASSEMBLYPROPERTY
  - OBJECT_DEFINITION
  - COL_LENGTH
  - OBJECT_ID
  - COL_NAME
  - OBJECT_NAME
  - COLUMNPROPERTY
  - OBJECT_SCHEMA_NAME
  - DATABASE_PRINCIPAL_ID
  - OBJECTPROPERTY
  - DATABASEPROPERTYEX
  - OBJECTPROPERTYEX
  - DB_ID
  - ORIGINAL_DB_NAME
  - DB_NAME
  - PARSENAME
  - FILE_ID
  - SCHEMA_ID
  - FILE_IDEX
  - SCHEMA_NAME
  - FILE_NAME
  - SCOPE_IDENTITY
  - FILEGROUP_ID
  - SERVERPROPERTY
  - FILEGROUP_NAME
  - STATS_DATE
  - FILEGROUPPROPERTY
  - TYPE_ID
  - FILEPROPERTY
  - TYPE_NAME
  - FULLTEXTCATALOGPROPERTY
  - TYPEPROPERTY
  - FULLTEXTSERVICEPROPERTY

- **Funciones de seguridad:** Devuelven información acerca de usuarios y roles.
  - CERTENCODED
  - PWDCOMPARE
  - CERTPRIVATEKEY
  - PWDENCRYPT
  - CURRENT_USER
  - SCHEMA_ID
  - DATABASE_PRINCIPAL_ID
  - SCHEMA_NAME
  - sys.fn_builtin_permissions
  - SESSION_USER
  - sys.fn_get_audit_file
  - SUSER_ID
  - sys.fn_my_permissions
  - SUSER_SID
  - HAS_PERMS_BY_NAME
  - SUSER_SNAME
  - IS_MEMBER
  - SYSTEM_USER
  - IS_ROLEMEMBER
  - SUSER_NAME
  - IS_SRVROLEMEMBER
  - USER_ID
  - ORIGINAL_LOGIN
  - USER_NAME
  - PERMISSIONS

- **Funciones de cadena:** Realizan operaciones en el valor de entrada de una cadena (Char o Varchar) y devuelven una cadena o un valor numérico.

Función	Descripción

ASCII	Devuelve el valor numérico ASCII que representa a un determinado carácter.
	Script que permite mostrar el código ASCII de las letras Ñ y ñ.
	`Ñ  ñ`
	`---------- ----------`
	`209  241`
	`(1 row(s) affected)`
	SELECT ASCII('Ñ') AS Ñ, ASCII('ñ') AS ñ
	GO

LTRIM

Devuelve una cadena de caracteres quitando los espacios en blanco en el lado izquierdo de la cadena.

Script que permite declarar una variable con diez espacios en blanco y otra variable con un código de cuatro caracteres; luego, muestra la unión de ambas variables sin espacios.

```
CODIGO          CODIGO SIN ESPACIO
--------------- ------------------
          C010 C010

(1 row(s) affected)
```

```
DECLARE @ESPACIO CHAR(10) = '          '
DECLARE @CODIGO CHAR(4)='C010'
SELECT @ESPACIO+@CODIGO AS CODIGO,
            LTRIM(@ESPACIO+@CODIGO) AS [CODIGO SIN ESPACIO]
GO
```

CHAR

Convierte un código ASCII en un carácter.

Script que permite mostrar los caracteres del valor 209 y 241.

```
209  241
---- ----
Ñ    ñ

(1 row(s) affected)
```

```
SELECT CHAR(209) AS '209', CHAR(241) AS '241'
GO
```

SPACE

Asigna una cantidad de espacios en blanco en una cadena de caracteres.

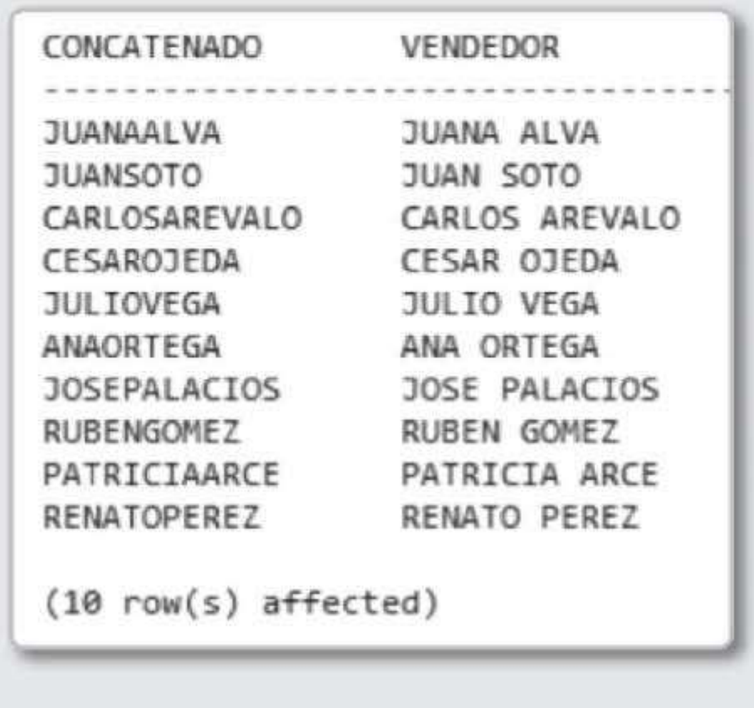
Script que permite mostrar el nombre del vendedor concatenado a su apellido.

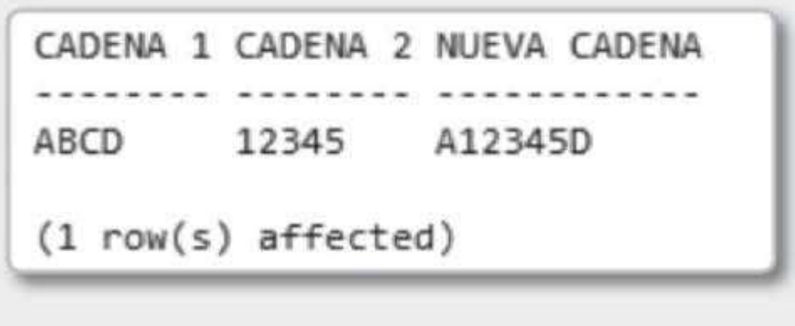
```
CONCATENADO         VENDEDOR
------------------------------------
JUANAALVA           JUANA ALVA
JUANSOTO            JUAN SOTO
CARLOSAREVALO       CARLOS AREVALO
CESAROJEDA          CESAR OJEDA
JULIOVEGA           JULIO VEGA
ANAORTEGA           ANA ORTEGA
JOSEPALACIOS        JOSE PALACIOS
RUBENGOMEZ          RUBEN GOMEZ
PATRICIAARCE        PATRICIA ARCE
RENATOPEREZ         RENATO PEREZ

(10 row(s) affected)
```

```
SELECT V.NOM_VEN+V.APE_VEN AS CONCATENADO,
            V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR
            FROM VENDEDOR V
GO
```

STR	Convierte un valor numérico en un valor de tipo cadena con espacios en blanco en el lado izquierdo. Script que permite imprimir un valor de tipo numérico.  `EL VALOR DE N ES:  1234` DECLARE @N INT = 1234 PRINT 'EL VALOR DE N ES: '+STR(@N) GO

CONCAT	Permite unir dos o más cadenas.
	Script que permite mostrar el nombre del vendedor concatenado a su apellido.
	 
	`CONCATENADO  VENDEDOR`
	`JUANAALVA  JUANA ALVA`
	`JUANSOTO  JUAN SOTO`
	`CARLOSAREVALO  CARLOS AREVALO`
	`CESAROJEDA  CESAR OJEDA`
	`JULIOVEGA  JULIO VEGA`
	`ANAORTEGA  ANA ORTEGA`
	`JOSEPALACIOS  JOSE PALACIOS`
	`RUBENGOMEZ  RUBEN GOMEZ`
	`PATRICIAARCE  PATRICIA ARCE`
	`RENATOPEREZ  RENATO PEREZ`
	`(10 row(s) affected)`
	SELECT CONCAT(V.NOM_VEN,V.APE_VEN) AS CONCATENADO,
	CONCAT(V.NOM_VEN,SPACE(1),V.APE_VEN) AS VENDEDOR
	FROM VENDEDOR V
	GO

STUFF	Devuelve una cadena formada por la inserción de una cadena en otra.
	Script que permite definir dos cadenas; en la cual la cadena 2 se inserta en la cadena 1 desde la posición con dos caracteres.
	 
	`CADENA 1  CADENA 2  NUEVA CADENA`
	`ABCD  12345  A12345D`
	`(1 row(s) affected)`
	DECLARE @CADENA1 VARCHAR(4)='ABCD'
	DECLARE @CADENA2 VARCHAR(5)='12345'
	SELECT @CADENA1 AS [CADENA 1], @CADENA2 AS [CADENA 2],
	STUFF(@CADENA1,2,2,@CADENA2) AS [NUEVA CADENA]
	GO

REPLACE

Permite reemplazar una cadena por otra dentro de una cadena base.

Script que permite mostrar el código de los clientes reemplazando el número cero por un carácter "x".

```
CODIGO NUEVO CODIGO
------ -------------
C001   CXX1
C002   CXX2
C003   CXX3
C004   CXX4
C005   CXX5
C006   CXX6
C007   CXX7
C008   CXX8
C009   CXX9
C010   CX1X
```

```
SELECT C.COD_CLI AS CODIGO,
        REPLACE(C.COD_CLI,'0','X') AS [NUEVO CODIGO]
        FROM CLIENTE C
GO
```

SUBSTRING

Devuelve una cadena de caracteres desde una posición inicial y una determinada cantidad de caracteres.

Script que permita separar el carácter y la parte numérica del código de cliente.

```
CODIGO LETRA NUMERO
------ ----- ------
C001   C     001
C002   C     002
C003   C     003
C004   C     004
C005   C     005
C006   C     006
C007   C     007
C008   C     008
C009   C     009
C010   C     010
C011   C     011
C012   C     012
```

```
SELECT C.COD_CLI AS CODIGO,
        SUBSTRING(C.COD_CLI,1,1) AS LETRA,
        SUBSTRING(C.COD_CLI,2,3) AS NUMERO
        FROM CLIENTE C
GO
```

FORMAT

Devuelve un valor con formato aplicado normalmente a valores de tipo fecha/hora, número y cadenas.

Script que permite la fecha actual en diferentes formatos:

	FORMATO AMERICANO	FORMATO ESPAÑOL	FORMATO ALEMAN
1	3/17/2017	17/03/2017	17.03.2017

	FORMATO AMERICANO	FORMATO ESPAÑOL	FORMATO ALEMAN
1	Friday, March 17, 2017	viernes, 17 de marzo de 2017	Freitag, 17. März 2017

```
DECLARE @FECHA DATETIME = GETDATE()
SELECT FORMAT ( @FECHA, 'd', 'en-US' )
                    AS 'FORMATO AMERICANO'
    ,FORMAT ( @FECHA, 'd', 'es-Es' )
                    AS 'FORMATO ESPAÑOL'
    ,FORMAT ( @FECHA, 'd', 'de-de' )
                    AS 'FORMATO ALEMAN'

SELECT FORMAT ( @FECHA, 'D', 'en-US' )
                    AS 'FORMATO AMERICANO'
    ,FORMAT ( @FECHA, 'D', 'es-Es' )
                    AS 'FORMATO ESPAÑOL'
    ,FORMAT ( @FECHA, 'D', 'de-de' )
                    AS 'FORMATO ALEMAN'
GO
```

Script que permite mostrar el número telefónico en el formato 99-999999.

```
NUMERO TELEFONICO
----------------------
982-360540
(1 row(s) affected)
```

```
DECLARE @TELEFONO INT='982360540'
SELECT FORMAT(@TELEFONO,'###-######') AS 'NUMERO TELEFONICO'
GO
```

Script que permite mostrar el sueldo de los vendedores con el formato de moneda.

```
SUELDO               FORMATO MONEDA
-------------------- ---------------
1000.00              $1,000.00
1200.00              $1,200.00
1500.00              $1,500.00
850.00               $850.00
1500.00              $1,500.00
1200.00              $1,200.00
2500.00              $2,500.00
1450.00              $1,450.00
1800.00              $1,800.00
1550.00              $1,550.00

(10 row(s) affected)
```

```
SELECT V.SUE_VEN AS SUELDO,
   FORMAT(V.SUE_VEN, 'C', 'en-us') AS 'FORMATO MONEDA'
FROM VENDEDOR V
```

REPLICATE

Permite asignar uno o más caracteres en una cadena.

Script que permite asignar números al lado izquierdo de un número entero, a partir del estado de la factura.

```
ESTADO     ESTADO CON FORMATO
---------- ------------------
2          0002
3          0003
2          0002
2          0002
2          0002
3          0003
1          0001
2          0002
2          0002
2          0002
```

```
SELECT F.EST_FAC AS ESTADO,
        REPLICATE('0', (4 - LEN(F.EST_FAC))) + F.EST_FAC
            AS [ESTADO CON FORMATO]
        FROM FACTURA F
GO
```

LEFT

Devuelve una cantidad de caracteres desde el lado izquierdo de la cadena.

Script que permite mostrar el primer carácter del código de cliente.

```
CODIGO LETRA INICIAL
------ -------------
C001   C
C002   C
C003   C
C004   C
C005   C
C006   C
C007   C
C008   C
C009   C
```

```
SELECT C.COD_CLI AS CODIGO,
        LEFT(C.COD_CLI,1) AS [LETRA INICIAL]
        FROM CLIENTE C
GO
```

REVERSE

Devuelve el valor inverso de una cadena de caracteres.

Script que permite mostrar el código del cliente de forma revés.

```
CODIGO NUEVO CODIGO
------ ------------
C001    100C
C002    200C
C003    300C
C004    400C
C005    500C
C006    600C
C007    700C
C008    800C
C009    900C
C010    010C
C011    110C
```

```
SELECT C.COD_CLI AS CODIGO,
        REVERSE(C.COD_CLI) AS [NUEVO CODIGO]
        FROM CLIENTE C
GO
```

UPPER

Devuelve en mayúsculas una cadena de caracteres.

Script que permite mostrar el correo electrónico en mayúsculas.

```
CORREO                                  CORREO MAYUSCULAS
--------------------------------------- ---------------------------------
manuel.torresr@hotmail.com              MANUEL.TORRESR@HOTMAIL.COM
(1 row(s) affected)
```

```
DECLARE @CORREO VARCHAR(40)='manuel.torresr@hotmail.com'
SELECT @CORREO AS [CORREO],
        UPPER(@CORREO) AS [CORREO MAYUSCULAS]
GO
```

LEN

Devuelve el número de caracteres que representa la longitud de la cadena.

Script que muestra la cantidad de caracteres que contiene los apellidos del vendedor.

```
APELLIDOS            TOTAL DE CARACTERES
-------------------- -------------------
ALVA                 4
SOTO                 4
AREVALO              7
OJEDA                5
VEGA                 4
ORTEGA               6
PALACIOS             8
GOMEZ                5
ARCE                 4
PEREZ                5
(10 row(s) affected)
```

```
SELECT V.APE_VEN AS APELLIDOS,
        LEN(V.APE_VEN) AS [TOTAL DE CARACTERES]
        FROM VENDEDOR V
GO
```

RIGHT

Obtiene caracteres desde el lado derecho de una cadena.

Script que permite mostrar el estado de la factura con ceros a la izquierda.

```
ESTADO      ESTADO CON FORMATO
----------- ------------------
2           0002
3           0003
2           0002
2           0002
2           0002
3           0003
1           0001
2           0002
2           0002
2           0002
```

```
SELECT F.EST_FAC AS ESTADO,
        RIGHT('0000' + LTRIM(RTRIM(F.EST_FAC)),4)
            AS [ESTADO CON FORMATO]
        FROM FACTURA F
GO
```

LOWER

Devuelve en minúsculas una cadena de caracteres.

Script que permite mostrar el correo electrónico en minúsculas.

```
CORREO                                   CORREO MINUSCULAS
---------------------------------------- ----------------------------
MANUEL.TORRES@HOTMAIL.COM                manuel.torres@hotmail.com

(1 row(s) affected)
```

```
DECLARE @CORREO VARCHAR(40)='MANUEL.TORRES@HOTMAIL.COM'
SELECT @CORREO AS [CORREO],
        LOWER(@CORREO) AS [CORREO MINUSCULAS]
GO
```

RTRIM

Devuelve una cadena de caracteres quitando los espacios en blanco en el lado derecho de la cadena.

Script que permite declarar una variable con diez espacios en blanco y otra variable con un código de cuatro caracteres; luego, muestra la unión de ambas variables sin espacios.

```
CODIGO          CODIGO SIN ESPACIO
--------------- ------------------
C010            C010

(1 row(s) affected)
```

```
DECLARE @ESPACIO CHAR(10) = '          '
DECLARE @CODIGO CHAR(4)='C010'
SELECT @CODIGO+@ESPACIO AS CODIGO,
        RTRIM(@CODIGO+@ESPACIO) AS [CODIGO SIN ESPACIO]
GO
```

- **Funciones del sistema:** Realizan operaciones y devuelven información acerca de valores, objetos y configuraciones de una instancia de SQL Server.
- **Funciones estadísticas del sistema:** Devuelven información estadística acerca del sistema.
  - @@CONNECTIONS
  - @@PACK_RECEIVED
  - @@CPU_BUSY
  - @@PACK_SENT
  - fn_virtualfilestats
  - @@TIMETICKS
  - @@IDLE
  - @@TOTAL_ERRORS
  - @@IO_BUSY
  - @@TOTAL_READ
  - @@PACKET_ERRORS
  - @@TOTAL_WRITE
- **Funciones de texto e imagen:** Realizan operaciones sobre los valores de entrada o columnas de texto o imagen, y devuelven información acerca del valor.
  - PATINDEX
  - TEXTVALID
  - TEXTPTR

7.3 Funciones definidas por el usuario

Es un tipo de implementación registrada por Transact-SQL que tiene por finalidad devolver un valor; en los lenguajes de programación este valor debe ser obtenido por algún medio, mientras que SQL Server lo puede mostrar a través de una consulta SELECT o la función PRINT, o asignarlo a una variable local mediante la sentencia SET.

Las funciones definidas por el usuario no se pueden utilizar para realizar acciones que modifican el estado de la base de datos. Las funciones definidas por el usuario, como las funciones de sistema, se pueden llamar desde una consulta. Las funciones escalares se pueden ejecutar con la instrucción EXECUTE, igual que los procedimientos almacenados.

7.3.1 Funciones escalares

Son funciones que devuelven un solo valor a quien lo invoque. Estas funciones luego se integran a sentencias como consultas, actualizaciones o eliminaciones.

Formato:

```
CREATE FUNCTION <PROPIETARIO. NOMBRE_FUNCION >(@PARAMETRO TIPODATO)
RETURNS <TIPODATO_RETORNADO>
< AS >
    BEGIN
        --CUERPO_FUNCION
        RETURN EXPRESION_SALIDA
    END
```

Donde:

- **PROPIETARIO:** SQL Server administra sus objetos por medio de su creador, esto es, un propietario para un grupo de objetos; aquellos objetos creados por cuenta del rol sysadmin automáticamente pertenecen a la cuenta DBO.

  DBO es una cuenta de usuario muy particular, ya que no puede ser utilizada para establecer conexiones a la base de datos, pero cuenta con privilegios especiales. La cuenta DBO es especial también porque es dueña de la base de datos. Por tanto, los objetos pertenecientes a la cuenta DBO tienen todos los permisos con dichos objetos y también se pueden derivar a otras cuentas de usuario. Cuando sucede esto se tiene que especificar el nombre del propietario y luego el del objeto, así como PROPIETARIO.OBJETO. En las funciones definidas por el usuario se tiene que especificar el propietario al ejecutarlas, ya que necesita comprobar qué tipo de permiso tiene dicho usuario, para esto se puede especificar de la siguiente manera: DBO.FN_CALCULA().

- **@PARAMETRO:** Se define las variables de entrada a la función, es decir, quien lo invoque tiene que enviar un valor por cada parámetro especificado en la función; hay que tener mucho cuidado en el tipo de datos de dichos valores.

- **RETURNS:** Permite especificar el tipo de datos de salida de la expresión en una función. Hay que tener en cuenta que una función de tipo escalar devuelve siempre un valor; por lo tanto, se debe especificar de qué tipo de datos es el valor de la salida.

- **RETURN:** Aquí se especifica el valor de salida de la función; se puede enviar directamente un valor, o por medio de una expresión.

A continuación, algunos casos para el uso de funciones escalares:

```
Función que permite calcular el 12% de un monto.
--Validando la existencia de la función
IF OBJECT_ID('FN_CALCULADESCUENTO') IS NOT NULL
        DROP FUNCTION FN_CALCULADESCUENTO
GO
--Creando la función
CREATE FUNCTION FN_CALCULADESCUENTO(@SUBTOTAL MONEY)
RETURNS MONEY
BEGIN
        RETURN @SUBTOTAL * 0.12
END
```

```
--Prueba 1
SELECT DBO.FN_CALCULADESCUENTO(1500) AS DESCUENTO
GO
```

```
DESCUENTO
---------------------
180.00
(1 row(s) affected)
```

```
--Prueba 2
SELECT F.NUM_FAC AS FACTURA,F.CAN_VEN*F.PRE_VEN AS SUBTOTAL,
       DBO.FN_CALCULADESCUENTO(F.PRE_VEN*F.CAN_VEN) AS DESCUENTO
       FROM DETALLE_FACTURA F
GO
```

FACTURA	SUBTOTAL	DESCUENTO
FA001	30.00	3.60
FA001	625.00	75.00
FA001	220.00	26.40
FA002	80.00	9.60
FA003	400.00	48.00
FA003	120.00	14.40
FA003	252.00	30.24
FA003	120.00	14.40
FA004	30.00	3.60
FA004	250.00	30.00
FA005	200.00	24.00
FA005	2500.00	300.00
FA005	300.00	36.00
FA006	700.00	84.00
FA006	150.00	18.00

```
Función que permite mostrar el nombre del distrito a partir del código
del mismo.
--Validando la existencia de la función
IF OBJECT_ID('FN_MUESTRADISTRITO') IS NOT NULL
       DROP FUNCTION FN_MUESTRADISTRITO
GO
--Creando la función
CREATE FUNCTION FN_MUESTRADISTRITO(@COD CHAR(3))
RETURNS VARCHAR(30)
BEGIN
       DECLARE @DIS VARCHAR(30)
       SELECT @DIS = D.NOM_DIS FROM DISTRITO D WHERE D.COD_DIS=@COD
       RETURN @DIS
END
--Prueba 1
SELECT DBO.FN_MUESTRADISTRITO('D01') AS DISTRITO
GO
```

```
DISTRITO
------------------------------
SURCO
(1 row(s) affected)
```

```
--Prueba 2
SELECT C.COD_CLI AS CODIGO,
       C.RSO_CLI AS CLIENTE,
       DBO.FN_MUESTRADISTRITO(C.COD_DIS) AS DISTRITO,
       C.RUC_CLI AS RUC
       FROM CLIENTE C
GO
```

```
CODIGO CLIENTE                        DISTRITO                        RUC
------ ------------------------------ ------------------------------ -----------
C001   FINSETH                        SAN MIGUEL                     48632081
C002   ORBI                           LA MOLINA                      57031642
C003   SERVIEMSA                      SAN MIGUEL                     NULL
C004   ISSA                           SURCO                          46720159
C005   MASS                           SURQUILLO                      83175942
C006   BERKER                         SAN MIGUEL                     54890124
C007   FIDENZA                        LOS OLIVOS                     16204790
C008   INTECH                         SAN BORJA                      34021824
C009   PROMINENT                      BREÑA                          NULL
C010   LANDU                          SAN MIGUEL                     30405261
C011   FILASUR                        SAN JUAN DE LURIGANCHO         70345201
C012   SUCERTE                        SAN MIGUEL                     62014503
C013   HAYASHI                        LIMA - CERCADO                 NULL
C014   KADIA                          MIRAFLORES                     22202915
C015   MEBA                           BELLAVISTA                     50319542
C016   CARDELI                        LINCE                          26403158
C017   PAYET                          BARRANCO                       70594032
C018   DASIN                          CALLAO                         35016752
C019   COREFO                         SAN LUIS                       57201691
C020   CRAMER                         JESUS MARIA                    46031783
C021   TAI LOY                        SURCO                          51551585745

(21 row(s) affected)
```

Script que permite mostrar la fecha de facturación en letras.

FACTURA

	Column Name	Condensed Type
🔑	NUM_FAC	varchar(12)
	FEC_FAC	date
	COD_CLI	char(5)
	FEC_CAN	date
	EST_FAC	varchar(10)
	COD_VEN	char(3)
	POR_IGV	decimal(18, 0)

```
IF OBJECT_ID('FN_MESLETRAS') IS NOT NULL
BEGIN
       DROP FUNCTION FN_MESLETRAS
END
GO

CREATE FUNCTION FN_MESLETRAS(@FECHA DATE)
RETURNS VARCHAR(20)
AS
BEGIN
```

```
DECLARE @NOMBRE VARCHAR(20)
   SET @NOMBRE=
    CASE DATENAME(MONTH,@FECHA)
      WHEN 'JANUARY' THEN 'ENERO'
      WHEN 'FEBRUARY' THEN 'FEBRERO'
      WHEN 'MARCH' THEN 'MARZO'
      WHEN 'APRIL' THEN 'ABRIL'
      WHEN 'MAY' THEN 'MAYO'
      WHEN 'JUNE' THEN 'JUNIO'
      WHEN 'JULY' THEN 'JULIO'
      WHEN 'AUGUST' THEN 'AGOSTO'
      WHEN 'SEPTEMBER' THEN 'SETIEMBRE'
      WHEN 'OCTOBER' THEN 'OCTUBRE'
      WHEN 'NOVEMBER' THEN 'NOVIEMBRE'
      WHEN 'DECEMBER' THEN 'DICIEMBRE'
    END
   RETURN @NOMBRE
END
GO

--PRUEBA 1
SELECT DBO.FN_MESLETRAS(GETDATE())
GO
```

```
--------------------
MARZO

(1 row(s) affected)
```

```
--PRUEBA 2
SELECT F.FEC_FAC AS NUMERO,
       F.FEC_FAC AS FECHA,
       CAST(DAY(F.FEC_FAC) AS CHAR(2))+' DE '
              +DBO.FN_MESLETRAS(F.FEC_FAC)+' DEL '+
              CAST(YEAR(F.FEC_FAC) AS CHAR(4)) AS [FECHA EN LETRAS],
       F.EST_FAC AS ESTADO
       FROM FACTURA F
GO
```

	NUMERO	FECHA	FECHA EN LETRAS	ESTADO
1	2013-06-07	2013-06-07	7 DE JUNIO DEL 2013	2
2	2013-06-07	2013-06-07	7 DE JUNIO DEL 2013	3
3	2013-01-09	2013-01-09	9 DE ENERO DEL 2013	2
4	2013-06-09	2013-06-09	9 DE JUNIO DEL 2013	2
5	2013-01-10	2013-01-10	10 DE ENERO DEL 2013	2
6	2013-01-08	2013-01-08	8 DE ENERO DEL 2013	3
7	2013-05-10	2013-05-10	10 DE MAYO DEL 2013	1
8	2013-04-10	2013-04-10	10 DE ABRIL DEL 2013	2
9	2013-03-10	2013-03-10	10 DE MARZO DEL 2013	2
10	2013-01-10	2013-01-10	10 DE ENERO DEL 2013	2
11	2013-10-09	2013-10-09	9 DE OCTUBRE DEL 2...	2

7.3.2 Funciones tabla en línea

Son funciones que devuelven los registros de una tabla como un conjunto.

Formato:

```
CREATE FUNCTION <PROPIETARIO. NOMBRE_FUNCION >(@PARAMETRO TIPODATO)
RETURNS TABLE
< AS >
    BEGIN
        RETURN (CONSULTA)
    END
```

Donde:

- **RETURNS TABLE:** Especifica que la devolución de la función se refiere a un conjunto de valores.
- **RETURN (CONSULTA):** Aquí se especifica la consulta que permite ser devuelto por la función; se debe tener en cuenta que los parámetros de la función pueden ser usados en la condición de la consulta.

A continuación, algunos casos para el uso de funciones tabla en línea:

```
Función que permite listar los productos.
```

PRODUCTO

	Column Name	Condensed Type
🔑	COD_PRO	char(5)
	DES_PRO	varchar(50)
	PRE_PRO	money
	SAC_PRO	int
	SMI_PRO	int
	UNI_PRO	varchar(30)
	LIN_PRO	varchar(30)
	IMP_PRO	varchar(10)

```
IF OBJECT_ID('FN_PRODUCTOS') IS NOT NULL
BEGIN
       DROP FUNCTION FN_PRODUCTOS
END
GO

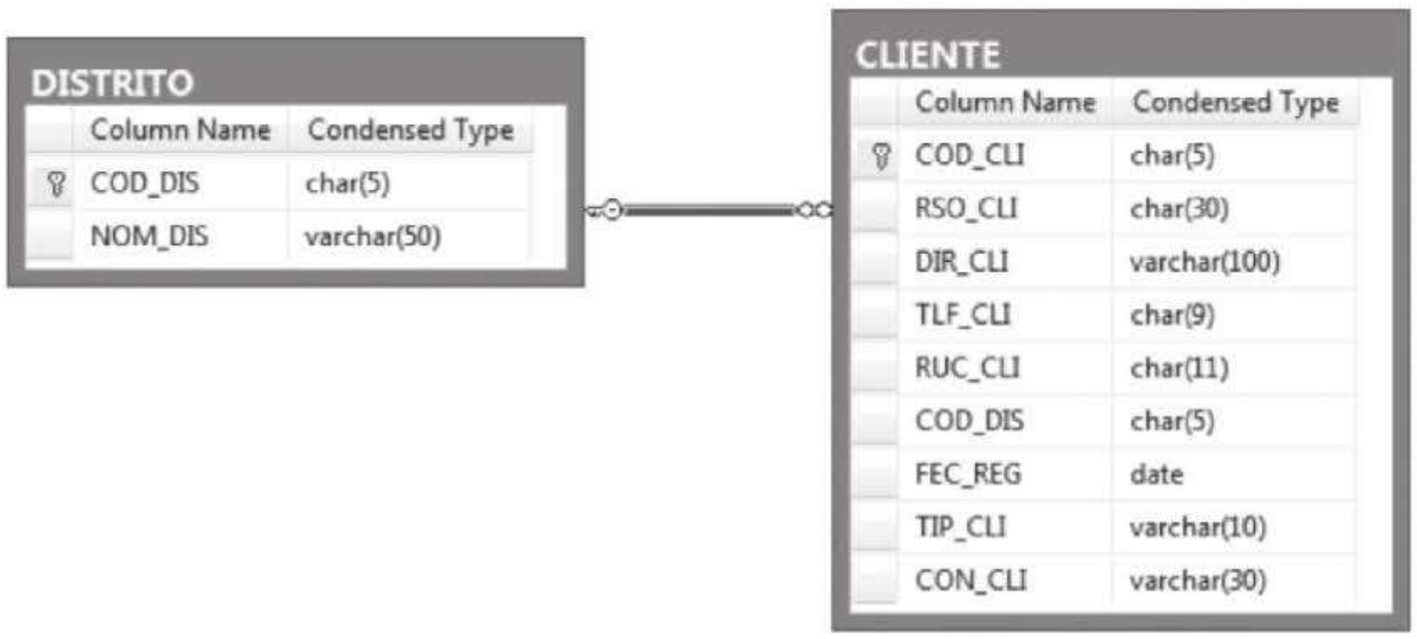
CREATE FUNCTION FN_PRODUCTOS()
RETURNS TABLE
AS

       RETURN (SELECT P.COD_PRO AS CODIGO, P.DES_PRO AS DESCRIPCION,
                      P.PRE_PRO AS PRECIO, P.SAC_PRO AS [STOCK ACTUAL],
                      P.SMI_PRO AS [STOCK MINIMO],P.UNI_PRO AS UNIDAD,
                      P.LIN_PRO AS LINEA, P.IMP_PRO AS IMPORTADO
                      FROM PRODUCTO P)
GO
```

```
--PRUEBA
SELECT * FROM FN_PRODUCTOS()
GO
```

CODIGO	DESCRIPCION	PRECIO	STOCK ACTUAL	STOCK MINIMO	UNIDAD	LINEA	IMPORTADO
P001	PAPEL BOND A-4 ATLAS	18.50	200	1500	MLL	2	V
P002	PAPEL BOND OFICIO	37.00	100	1500	MLL	2	F
P003	PAPEL BULKY	12.00	498	1000	MLL	2	V
P004	PAPEL PERIODICO 75 GRAMOS	11.00	4285	1000	MLL	2	F
P005	CARTUCHO TINTA NEGRA	21.00	100	30	UNI	1	F
P006	CARTUCHO TINTA COLOR	47.00	58	35	UNI	1	F
P007	PORTA DISKETTES	5.50	300	100	UNI	1	V
P008	CAJA DE DISKETTES * 10	32.00	125	180	UNI	1	F
P009	BORRADOR DE TINTA	12.00	100	500	DOC	3	F

Función que permita mostrar un listado de los clientes según el distrito.




```
IF OBJECT_ID('FN_CLIENTESXDISTRITO') IS NOT NULL

BEGIN
                DROP FUNCTION FN_CLIENTESXDISTRITO
END
GO

CREATE FUNCTION FN_CLIENTESXDISTRITO(@DIS VARCHAR(30))

RETURNS TABLE
AS
        RETURN (
                SELECT C.COD_CLI AS CODIGO, C.RSO_CLI AS CLIENTE,
                        C.DIR_CLI AS DIRECCION,
                        D.NOM_DIS AS DISTRITO,
                        C.TLF_CLI AS TELEFONO
                        FROM CLIENTE C
                        JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
                        WHERE D.NOM_DIS=@DIS
        )
GO
```

```
--PRUEBA
SELECT * FROM FN_CLIENTESXDISTRITO('SAN MIGUEL')
GO
```

```
CODIGO CLIENTE      DIRECCION                          DISTRITO      TELEFONO
------ ------------------------------------------------------------------
C001   FINSETH      AV. LOS VIÑEDOS 150                SAN MIGUEL    4342318
C003   SERVIEMSA    JR. COLLAGATE 522                  SAN MIGUEL    75012403
C006   BERKER       AV. LOS PROCERES 521               SAN MIGUEL    3810322
C010   LANDU        AV.NICOLAS DE AYLLON 1453          SAN MIGUEL    3267840
C012   SUCERTE      JR. GRITO DE HUAURA 114            SAN MIGUEL    4206434

(5 row(s) affected)
```

7.3.3 Funciones tabla multisentencia

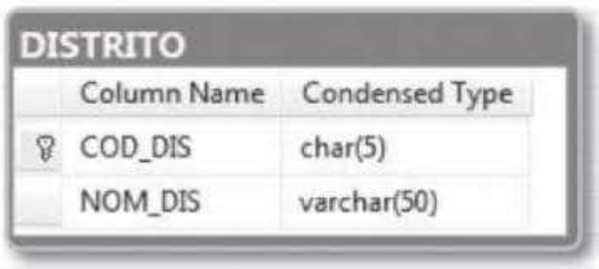
Son funciones que devuelven los registros de una tabla como un conjunto.

Formato:

```
CREATE FUNCTION <PROPIETARIO. NOMBRE_FUNCION >(@PARAMETRO TIPODATO)
RETURNS @VARIABLE TABLE (ARGUMENTO)
< AS >
    BEGIN
        --CUERPO DE LA FUNCION
        RETURN
    END
```

A continuación, algunos casos para el uso de funciones tabla multisentencia:

```
Función que permite mostrar los datos de la tabla DISTRITO.
```



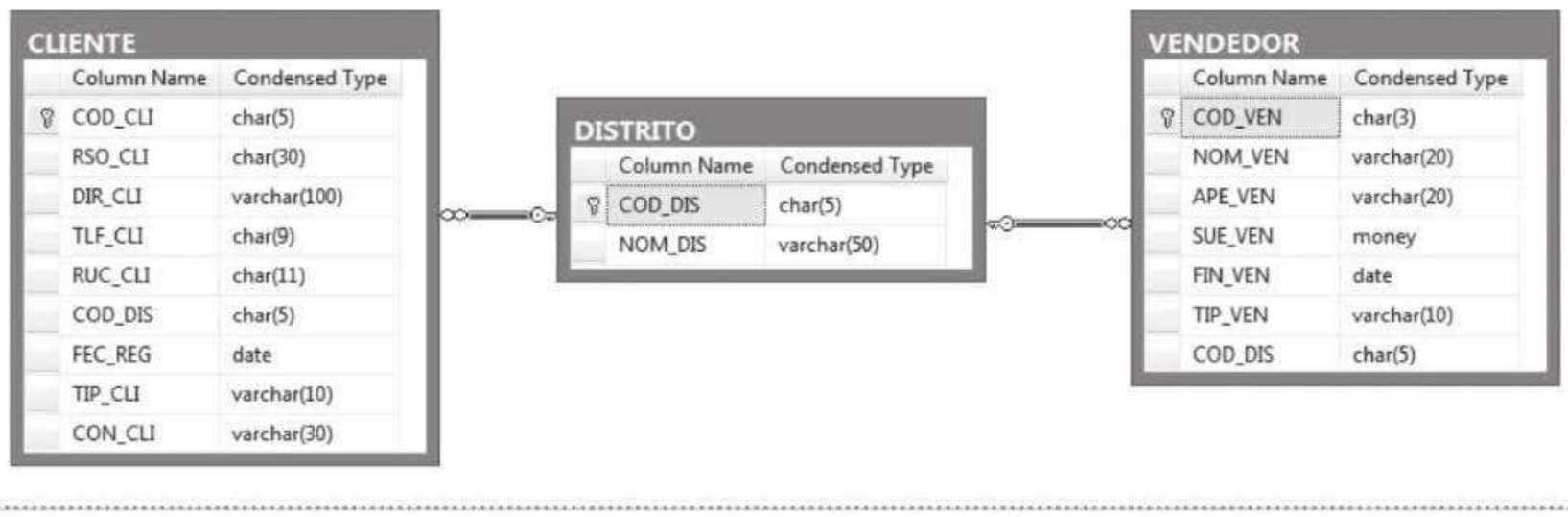

```
IF OBJECT_ID('FN_LISTADODISTRITO') IS NOT NULL
BEGIN
        DROP FUNCTION FN_LISTADODISTRITO
END
GO

CREATE FUNCTION FN_LISTADODISTRITO()
RETURNS @TABLA TABLE(
                COD CHAR(5),
                DIS VARCHAR(40))
AS
```

```
BEGIN
        INSERT INTO @TABLA
                SELECT F.COD_DIS AS CODIGO,
                        F.NOM_DIS AS DISTRITO
                        FROM DISTRITO F
        RETURN
END
GO
--PRUEBA:
SELECT * FROM FN_LISTADODISTRITO()
GO
```

COD	DIS
D01	SURCO
D02	JESUS MARIA
D03	SAN ISIDRO
D04	LA MOLINA
D05	SAN MIGUEL
D06	MIRAFLORES
D07	BARRANCO
D08	CHORRILLOS
D09	SAN BORJA
D1	NUEVO CERCADO
D10	LINCE
D11	BREÑA
D12	MAGDALENA
D13	RIMAC
D14	SURQUILLO
D15	PUEBLO LIBRE
D16	BELLAVISTA
D17	CALLAO
D18	SAN MARTIN DE PORRES

Función que permite mostrar dos columnas de las tablas DISTRITO, CLIENTE y VENDEDOR desde una sola función.



```
IF OBJECT_ID('FN_VISTATABLA') IS NOT NULL
BEGIN
	DROP FUNCTION DBO.FN_VISTATABLA
END
GO
CREATE FUNCTION FN_VISTATABLA(@TABLA VARCHAR(20))
RETURNS @MATRIZ TABLE (CODIGO CHAR(10),DESCRIPCION VARCHAR(50))
AS
BEGIN
	IF @TABLA='DISTRITO'
		INSERT INTO @MATRIZ
			SELECT D.COD_DIS, D.NOM_DIS
				FROM DISTRITO D
		ELSE IF @TABLA='CLIENTE'
			INSERT INTO @MATRIZ
			SELECT C.COD_CLI,C.RSO_CLI
				FROM CLIENTE C
		ELSE IF @TABLA='VENDEDOR'
			INSERT INTO @MATRIZ
			SELECT V.COD_VEN, V.NOM_VEN+SPACE(1)+V.APE_VEN
				FROM VENDEDOR V
	RETURN
END
GO
--PRUEBA 1
SELECT * FROM FN_VISTATABLA('DISTRITO')
GO
--PRUEBA 2
SELECT * FROM FN_VISTATABLA('CLIENTE')
GO
--PRUEBA 3
SELECT * FROM FN_VISTATABLA('VENDEDOR')
GO
```

Caso desarrollado

Si se cuenta con el siguiente diagrama de base de datos:



Y con la siguiente información por tablas:

SELECT * FROM CLIENTE

```
IDE_CLI APE_CLI    NOM_CLI      DNI_CLI  TEL_CLI     IDE_DIS COR_CLI
------- ---------- ------------ -------- ----------- ------- -------
CL001   LOPEZ      MARIA        10856985 526-5254    L03
CL002   GOMEZ      JOSE         02188525 585-8212    L02
CL003   ACOSTA     JESUS        07415482 485-1598    L01
CL004   ARIAS      GUADSLUPE    08995255 258-8569    L04
CL005   CARLOS     MANUEL       10418525 296-9685    L08
CL006   GERZ       BRIGITTE     10952155 585-1254    L06
CL007   BARBOSA    MARISOL      78514855 584-5869    L04

(7 row(s) affected)
```

SELECT * FROM DISTRITO

```
IDE_DIS DES_DIS
------- --------------------
L01     LIMA
L02     LINCE
L03     BREÑA
L04     LOS OLIVOS
L05     RIMAC
L06     SAN MARTIN DE PORRES
L07     SAN LUIS
L08     SAN BORJA

(8 row(s) affected)
```

SELECT * FROM AUTOMOVIL

```
MAT_AUT    COL_AUT                          MOD_AUT
---------- -------------------------------- ---------------
AF-456     ROJO                             SEDAN
D4-243     NEGRO                            SEDAN
FD-463     PLATA                            STATION WAGON
FG-654     PLATA                            L10
GG-654     ROJO                             CAMIONETA
GT-532     ROJO                             SEDAN
H5-455     ROJO                             STATION WAGON
HY-345     PLATA                            CHERY
SF-112     ROJO                             LI
YT-457     NEGRO                            STATION WAGON

(10 row(s) affected)
```

SELECT * FROM ALQUILER

```
NUM_ALQ     FEC_ALQ    MON_ALQ
----------- ---------- ---------
1           2010-03-12 185.00
2           2011-01-02 100.00
3           2011-05-01 54.00
4           2012-08-11 85.00
5           2012-06-22 18.00
6           2012-01-30 15.00
7           2012-04-16 165.00
8           2012-05-23 135.00
9           2012-03-21 225.00

(9 row(s) affected)
```

SELECT * FROM DETALLEALQUILER

```
NUM_ALQ     IDE_CLI MAT_AUT
----------- ------- ----------
1           CL002   GG-654
2           CL002   GT-532
3           CL006   FD-463
3           CL007   GT-532
4           CL005   H5-455
5           CL004   SF-112
6           CL004   YT-457
7           CL003   GT-532

(8 row(s) affected)
```

Implemente las siguientes funciones:

- Función que devuelva el promedio del monto de alquiler, para luego mostrar el siguiente reporte:

AÑO ALQUILER	PROMEDIO POR AÑO
9999	999.99
9999	999.99
9999	999.99

- Función que permita mostrar el menor monto de alquiler en un determinado año, para luego mostrar el siguiente reporte:

AÑO ALQUILER	MONTO MINIMO
9999	999.99
9999	999.99
9999	999.99

- Función que permita mostrar el monto acumulado de alquiler en un determinado año, para luego mostrar el siguiente reporte:

AÑO ALQUILER	MONTO ACUMULADO
9999	999.99
9999	999.99
9999	999.99

- Función que permita mostrar el total de automóviles de un determinado color, para luego mostrar el siguiente reporte:

COLOR DE AUTO	TOTAL
XXXXXXXXX	99
XXXXXXXXX	99

- Función que permita mostrar la diferencia de años entre la fecha de alquiler y el año actual, para luego mostrar el siguiente reporte:

ALQUILER	MONTO	FECHA ALQUILER	DIFERENCIA EN AÑOS
99	999.99	99/99/9999	99
99	999.99	99/99/9999	99
99	999.99	99/99/9999	99

- Implemente una función tabla en línea que permita mostrar todos los registros de los clientes. Luego, muestre a los clientes de un determinado distrito.
- Implemente una función tabla en línea que permita mostrar todos los registros de los automóviles de un determinado color. Luego, muestre una lista de los automóviles de un determinado color.
- Implemente una función tabla en línea que permita mostrar todos los registros de los alquileres según el año de registro. Luego, muestre el registro de los alquileres de un determinado año.
- Implemente una función multisentencia que permita mostrar todos los registros de los automóviles.
- Implemente una función multisentencia que permita mostrar todos los registros de los detalles de alquileres.

Solución:

```
--Función que devuelva el promedio del monto de alquiler,
--para luego mostrar el siguiente reporte:
--AÑO ALQUILER        PROMEDIO POR AÑO
--9999                999.99
--9999                999.99
--9999                999.99

--Implementación de la función
IF OBJECT_ID('FN_PROMEDIOxAÑO') IS NOT NULL
        DROP FUNCTION FN_PROMEDIOxAÑO
GO
```

```
CREATE FUNCTION FN_PROMEDIOxAÑO(@AÑO INT)
RETURNS MONEY
AS
BEGIN
        DECLARE @MONTO MONEY
        SELECT @MONTO = AVG(A.MON_ALQ)
                FROM ALQUILER A
                WHERE YEAR(A.FEC_ALQ) = @AÑO
        RETURN @MONTO
END
GO

--Reporte
SELECT DISTINCT YEAR(A.FEC_ALQ) AS [AÑO ALQUILER],
        DBO.FN_PROMEDIOxAÑO(YEAR(A.FEC_ALQ)) AS [PROMEDIO POR AÑO]
        FROM ALQUILER A
GO

--Función que permita mostrar el menor monto de alquiler en un determinado año,
--para luego mostrar el siguiente reporte:

--AÑO ALQUILER          MONTO MINIMO
--9999                  999.99
--9999                  999.99

--Implementación de la función
IF OBJECT_ID('FN_MENOR_MONTO') IS NOT NULL
        DROP FUNCTION FN_MENOR_MONTO
GO
CREATE FUNCTION FN_MENOR_MONTO(@AÑO INT)
RETURNS MONEY
AS
BEGIN
        DECLARE @MONTO MONEY
        SELECT @MONTO = MIN(A.MON_ALQ)
                FROM ALQUILER A
                WHERE YEAR(A.FEC_ALQ) = @AÑO
        RETURN @MONTO
END
GO

--Reporte
SELECT DISTINCT YEAR(A.FEC_ALQ) AS [AÑO ALQUILER],
        DBO.FN_MENOR_MONTO(YEAR(A.FEC_ALQ)) AS [MONTO MINIMO]
        FROM ALQUILER A
GO
```

```
--Función que permita mostrar el monto acumulado de alquiler en un
determinado año, para luego mostrar el siguiente reporte:
--AÑO ALQUILER        MONTO ACUMULADO
--9999                999.99
--9999                999.99
--9999                999.99

--Implementación de la función
IF OBJECT_ID('FN_MONTOACUMULADO') IS NOT NULL
        DROP FUNCTION FN_MONTOACUMULADO
GO
CREATE FUNCTION FN_MONTOACUMULADO(@AÑO INT)
RETURNS MONEY
AS
BEGIN
        DECLARE @MONTO MONEY
        SELECT @MONTO = SUM(A.MON_ALQ)
                FROM ALQUILER A
                WHERE YEAR(A.FEC_ALQ) = @AÑO
        RETURN @MONTO
END
GO

--Reporte
SELECT DISTINCT YEAR(A.FEC_ALQ) AS [AÑO ALQUILER],
        DBO.FN_MONTOACUMULADO(YEAR(A.FEC_ALQ)) AS [MONTO ACUMULADO]
        FROM ALQUILER A
GO

--Función que permita mostrar el total de automóviles de un determinado color,
--para luego mostrar el siguiente reporte:
--COLOR DE AUTO        TOTAL
--XXXXXXXXX            99
--XXXXXXXXX            99

--Implementación de la función
IF OBJECT_ID('FN_AUTOSxCOLOR') IS NOT NULL
        DROP FUNCTION FN_AUTOSxCOLOR
GO
CREATE FUNCTION FN_AUTOSxCOLOR(@COLOR VARCHAR(40))
RETURNS INT
AS
BEGIN
        DECLARE @TOTAL INT
        SELECT @TOTAL = COUNT(*)
                FROM AUTOMOVIL A
```

```
                WHERE A.COL_AUT=@COLOR
        RETURN @TOTAL
END
GO

--Reporte
SELECT DISTINCT A.COL_AUT AS [COLOR DE AUTO],
        DBO.FN_AUTOSxCOLOR(A.COL_AUT) AS [TOTAL]
        FROM AUTOMOVIL A
GO

--Función que permita mostrar la diferencia de años entre la fecha de al-
quiler y el año actual, para luego mostrar el siguiente reporte:
--ALQUILER     MONTO          FECHA ALQUILER         DIFERENCIA EN AÑOS
--99           999.99         99/99/9999             99
--99           999.99         99/99/9999             99
--99           999.99         99/99/9999             99

--Implementación de la función
IF OBJECT_ID('FN_DIFERENCIAxAÑOS') IS NOT NULL
        DROP FUNCTION FN_DIFERENCIAxAÑOS
GO
CREATE FUNCTION FN_DIFERENCIAxAÑOS(@FECHA DATE)
RETURNS INT
AS
BEGIN
        DECLARE @TOTAL INT
        SELECT @TOTAL = DATEDIFF(YY,@FECHA,GETDATE())
        RETURN @TOTAL
END
GO

--Reporte
SELECT A.NUM_ALQ AS ALQUILER,
        A.MON_ALQ AS MONTO,
        A.FEC_ALQ AS [FECHA ALQUILER],
        DBO.FN_DIFERENCIAxAÑOS(A.FEC_ALQ) AS [DIFERENCIA EN AÑOS]
        FROM ALQUILER A
GO

--Implemente una función tabla en línea que permita mostrar todos los
registros de los clientes. Luego, muestre a los clientes de un determinado
distrito.

--Implementación de la función
IF OBJECT_ID('FN_LISTACLIENTES') IS NOT NULL
        DROP FUNCTION FN_LISTACLIENTES
GO
CREATE FUNCTION FN_LISTACLIENTES()
RETURNS TABLE
AS
```

```
        RETURN (SELECT C.IDE_CLI AS CODIGO,
                       C.NOM_CLI+SPACE(1)+C.APE_CLI AS CLIENTE,
                       C.DNI_CLI AS DNI,
                       C.TEL_CLI AS TELEFONO,
                       D.DES_DIS AS DISTRITO,
                       C.COR_CLI AS CORREO
                       FROM CLIENTE C
                       JOIN DISTRITO D ON C.IDE_DIS=D.IDE_DIS)
GO

--Reporte de clientes por distrito
SELECT *
       FROM FN_LISTACLIENTES() C
       WHERE C.DISTRITO='LOS OLIVOS'
GO

--Implemente una función tabla en línea que permita mostrar todos los registros
--de los automóviles de un determinado color. Luego, muestre una lista de
los automóviles de un determinado color.

--Implementación de la función
IF OBJECT_ID('FN_LISTAAUTOMOVILES') IS NOT NULL
       DROP FUNCTION FN_LISTAAUTOMOVILES
GO
CREATE FUNCTION FN_LISTAAUTOMOVILES()
RETURNS TABLE
AS
       RETURN (SELECT A.MAT_AUT AS MATRICULA,
                      A.COL_AUT AS COLOR,
                      A.MOD_AUT AS MODELO
                      FROM AUTOMOVIL A
                )
GO

--Reporte de automóviles por color
SELECT *
       FROM FN_LISTAAUTOMOVILES() A
       WHERE A.COLOR='NEGRO'
GO

--Implemente una función tabla en línea que permita mostrar todos los registros
--de los alquileres según el año de registro.

--Implementación de la función
IF OBJECT_ID('FN_LISTAAALQUILER') IS NOT NULL
       DROP FUNCTION FN_LISTAAALQUILER
GO
```

```
CREATE FUNCTION FN_LISTAAALQUILER(@AÑO INT)
RETURNS TABLE
AS
        RETURN (SELECT A.NUM_ALQ AS ALQUILER,
                                A.FEC_ALQ AS FECHA,
                                A.MON_ALQ AS MONTO
                                FROM ALQUILER A
                                WHERE YEAR(A.FEC_ALQ) = @AÑO
                        )
GO

--Reporte de automóviles por color
DECLARE @AÑO INT = 2011
SELECT *
        FROM FN_LISTAAALQUILER(@AÑO)
GO

--Implemente una función multisentencia que permita mostrar todos los
registros de los automóviles.

--Implementación de la función
IF OBJECT_ID('FN_LISTADOAUTOS') IS NOT NULL
BEGIN
        DROP FUNCTION FN_LISTADOAUTOS
END
GO

CREATE FUNCTION FN_LISTADOAUTOS()
RETURNS @TABLA TABLE(
                CODIGO CHAR(10),
                COLOR  VARCHAR(30),
                MODELO VARCHAR(30))
AS
BEGIN
        INSERT INTO @TABLA
                SELECT A.MAT_AUT, A.COL_AUT, A.MOD_AUT
                        FROM AUTOMOVIL A
        RETURN
END
GO

--Prueba:
SELECT * FROM FN_LISTADOAUTOS()
GO

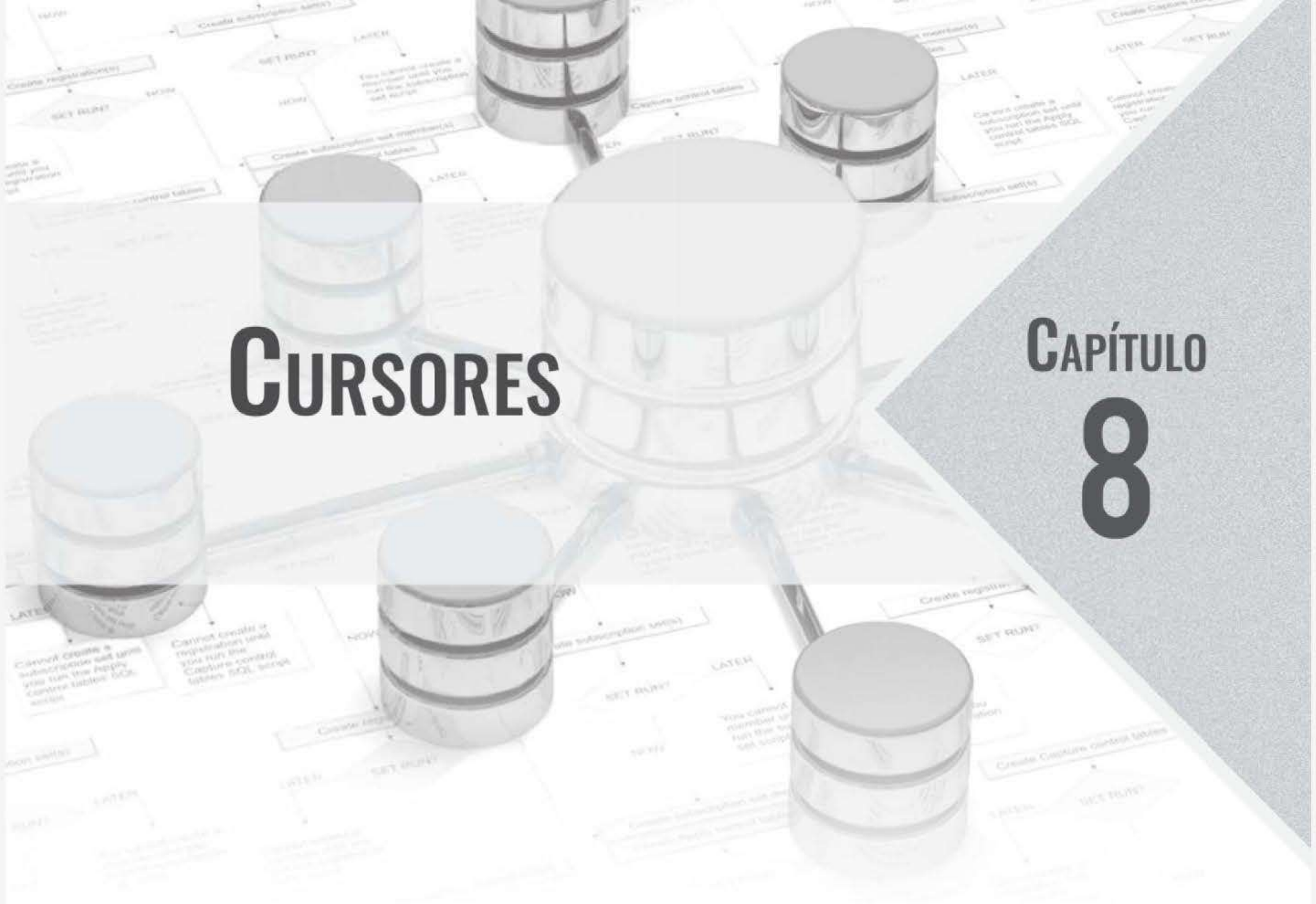
--Implemente una función multisentencia que permita mostrar todos los
registros de los detalles de alquileres.

--Implementación de la función
IF OBJECT_ID('FN_LISTADODETALLE') IS NOT NULL
BEGIN
```

```
        DROP FUNCTION FN_LISTADODETALLE
END
GO

CREATE FUNCTION FN_LISTADODETALLE()
RETURNS @TABLA TABLE(
                NUMERO          INT,
                CLIENTE         VARCHAR(30),
                AUTOMOVIL       CHAR(10))
AS
BEGIN
        INSERT INTO @TABLA
                SELECT D.NUM_ALQ,
                        C.NOM_CLI+SPACE(1)+C.APE_CLI,
                        D.MAT_AUT
                        FROM DETALLEALQUILER D
                        JOIN CLIENTE C ON D.IDE_CLI=C.IDE_CLI
        RETURN
END
GO

--Prueba:
SELECT * FROM FN_LISTADODETALLE()
GO
```



Capítulo 8

Cursores

Competencia tecnológica:

- Implementar consultas tipo reportes usando cursores.
- Implementar cursores con quiebres de segundo y tercer nivel.

Contenido:

8.1 Cursores
8.2 Proceso de implementación de un cursor
8.3 Tipos de cursores
8.4 Declaración de cursor

8.1 Cursores

Hasta el momento se ha visto que la sentencia SELECT ha trabajado para mostrar filas de registros de una determinada tabla o algún valor agrupado, o asignando valores a las variables locales; pero ya se debe tener en cuenta que esta instrucción controla a los registros de una tabla de forma matricial; de modo que, si en algún momento se deseaba almacenar información desde una tabla, se tenía que declarar una variable de tipo TABLE. Pero, a pesar de eso, hasta ahora no se ha visto el control de los registros fila por fila, ya que para esto se tendría que recorrer todo el conjunto de registros enviados desde SELECT y tratar de bajarlos a variables locales, aunque resultaría bastante engorroso trabajar de esa manera. Por ello, Transact SQL cuenta con CURSORES que permitirán tener el control fila por fila de un conjunto de registros.

Esto se debe a que hay algunas aplicaciones en línea que no pueden controlar información en conjunto. Estas aplicaciones necesitan un mecanismo que trabaje con una fila o un pequeño bloque de filas cada vez. Los cursores son una extensión de los conjuntos de resultados que proporciona dicho mecanismo.

Entonces, ¿en qué casos serán útiles los cursores? Si ya se tiene acceso a las filas de una tabla uno a uno, se podrá obtener dichos valores y controlarlos desde variables locales, y así poder acumular valores, contarlos, aplicar quiebres, etc.

Consideraciones:

- Los cursores permiten situarse en filas específicas del conjunto de resultados.
- Recuperan una fila o un bloque de filas de la posición actual en el conjunto de resultados.
- Aceptan modificaciones de los datos en la posición actual del conjunto de resultados.
- Aceptan diferentes grados de visibilidad para los cambios que realizan otros usuarios en la información de la base de datos que se presenta en el conjunto de resultados.

8.2 Proceso de implementación de un cursor

Para implementar un cursor se necesitan pasos específicos; siempre será recomendado implementarlo dentro de un procedimiento almacenado.

1. Declarar el cursor y asignar un conjunto de resultados provenientes de la instrucción **SELECT**.
2. Aperturar el cursor para que se active; usar el operador **OPEN**.
3. Recuperar las filas del cursor que se desea visualizar y usar la estructura repetitiva **WHILE** para dicho trabajo; aquí es el momento de usar la variable global **@@FETCH_STATUS**.
4. Hay que tener en cuenta que se pueden realizar operaciones de actualización o eliminación en la fila de la posición actual del cursor.
5. Cerrar el cursor usando el operador **CLOSE**.
6. Liberar el cursor usando el operador **DEALLOCATE**.

8.3 Tipos de cursores

Son cuatro tipos de cursores admitidos por Microsoft SQL Server. La instrucción DECLARE CURSOR se ha ampliado para que pueda especificar cuatro tipos para los cursores de Transact-SQL.

Estos cursores varían en su capacidad para detectar cambios en el conjunto de resultados y en los recursos que consumen, como la memoria y el espacio de Tempdb. Un cursor puede detectar cambios en las filas solo cuando intenta recuperarlas una segunda vez. El origen de datos no puede notificar al cursor las modificaciones realizadas en las filas recuperadas actualmente. El nivel de aislamiento de la transacción influye también en la capacidad de un cursor para detectar los cambios.

Los cuatro tipos de cursor de servidor de la API que admite el servidor SQL Server son:

- Cursores estáticos
- Cursores dinámicos
- Cursores de solo avance
- Cursores controlados por conjunto de claves

8.4 Declaración de cursor

Formato de la declaración de un cursor:

```
DECLARE NOMBRE_CURSOR CURSOR
FOR EXPRESION_SELECT
```

Donde:

- NOMBRE_CURSOR: Es el nombre definido al cursor.
- EXPRESION_SELECT: Es la consulta base del cursor.

Formato de la apertura de un CURSOR:

```
OPEN NOMBRE_CURSOR
```

Formato de la recuperación de registros desde el cursor:

```
FETCH
  [[NEXT|PRIOR|FIRST|LAST
      | ABSOLUTE N | @NVAR
      | RELATIVE N | @NVAR
   ] FROM ] NOMBRE_CURSOR | @VARIABLECURSOR
[ INTO @NOMBRE_VARIABLE[ ,...N ] ]
```

Donde:

- NEXT: Devuelve la fila de resultados inmediatamente posterior a la fila actual, y aumenta la fila actual a la fila devuelta. Si FETCH NEXT es la primera operación de recuperación en un cursor, se devuelve la primera fila del conjunto de resultados. NEXT es la opción predeterminada para la recuperación de cursores.
- PRIOR: Devuelve la fila de resultados inmediatamente anterior a la fila actual, y reduce la fila actual a la fila devuelta. Si FETCH PRIOR es la primera operación de recuperación en un cursor, no se devuelve ninguna fila y el cursor queda posicionado delante de la primera fila.
- FIRST: Devuelve la primera fila del cursor y la convierte en la fila actual.
- LAST: Devuelve la última fila del cursor y la convierte en la fila actual.
- ABSOLUTE N | @NVAR: Si N o @NVAR es positivo, se devuelve la fila N desde el principio del cursor y la fila devuelta se convierte en la nueva fila actual. Si N o @NVAR es negativo, se devuelve la fila N anterior al final del cursor y la fila devuelta se convierte en la nueva fila actual. Si N o @NVAR es 0, no se devuelven filas.
- RELATIVE N | @NVAR: Si N o @NVAR es positivo, se devuelve la fila N posterior a la fila actual y la fila devuelta se convierte en la nueva fila actual. Si N o @NVAR es negativo, se devuelve la fila N anterior a la fila actual y la fila devuelta se convierte en la nueva fila actual. Si N o @NVAR es 0, se devuelve la fila actual. Si se especifica FETCH RELATIVE con N o @NVAR establecidos en un número negativo o en 0 en la primera operación de recuperación que se realiza en un cursor, no se devuelven filas.

Formato de cierre de cursor:

Cierra un cursor abierto mediante la liberación del conjunto de resultados actual y todos los bloqueos de cursor mantenidos en las filas en las que está colocado. CLOSE deja las estructuras de datos accesibles para que se puedan volver a abrir, pero las recuperaciones y las actualizaciones posicionadas no se permiten hasta que se vuelva a abrir el cursor. CLOSE debe ejecutarse en un cursor abierto, por lo que no se permite en cursores que solo están declarados o que ya están cerrados.

```
CLOSE NOMBRE_CURSOR
```

Sintaxis de la liberación de recursos de un cursor:

```
DEALLOCATE NOMBRE_CURSOR
```

A continuación, algunos casos sobre el manejo de cursores:

Script que permita implementar un cursor básico donde se imprima el primer registro de la tabla VENDEDOR.

VENDEDOR

Column Name	Condensed Type
COD_VEN	char(3)
NOM_VEN	varchar(20)
APE_VEN	varchar(20)
SUE_VEN	money
FIN_VEN	date
TIP_VEN	varchar(10)
COD_DIS	char(5)

```
—DECLARANDO EL CURSOR
DECLARE C_VENDEDOR CURSOR
                FOR SELECT * FROM VENDEDOR
--ABRIR EL CURSOR
OPEN C_VENDEDOR
--MOSTRAR EL PRIMER REGISTRO
FETCH NEXT FROM C_VENDEDOR
GO
```

Al ejecutar la sentencia se muestra la siguiente imagen:

```
COD_VEN NOM_VEN              APE_VEN              SUE_VEN              FIN_VEN    TIP_VEN    COD_DIS
------- -------------------- -------------------- -------------------- ---------- ---------- -------
V04     CESAR                OJEDA                850.00               1999-08-11 1          D01

(1 row(s) affected)
```

Script que permita implementar un cursor donde se imprima todos los registros de la tabla CLIENTE.

CLIENTE

Column Name	Condensed Type
COD_CLI	char(5)
RSO_CLI	char(30)
DIR_CLI	varchar(100)
TLF_CLI	char(9)
RUC_CLI	char(11)
COD_DIS	char(5)
FEC_REG	date
TIP_CLI	varchar(10)
CON_CLI	varchar(30)

```
--DECLARACION DE VARIABLES
DECLARE @COD CHAR(3),@CLI CHAR(15),@DIR CHAR(30),
               @TEL CHAR(15),@DIS CHAR(20),@FEC DATE
--DECLARANDO EL CURSOR
DECLARE C_CLIENTES CURSOR
        FOR SELECT C.COD_CLI, C.RSO_CLI,
                        C.DIR_CLI,C.TLF_CLI,D.NOM_DIS,C.FEC_REG
                        FROM CLIENTE C
                        JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
```

```
--ABRIR EL CURSOR
OPEN C_CLIENTES
--OBTENER EL PRIMER REGISTRO
FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
PRINT 'CODIGO    CLIENTE         DIRECCION     TELEFONO       FECHA REG.'
PRINT '-----------------------------------------------------------------'
--RECORRER POR EL CURSOR
WHILE @@FETCH_STATUS=0
BEGIN
        PRINT @COD+SPACE(5)+@CLI+SPACE(5)+@DIR+SPACE(5)+
             @TEL+SPACE(5)+CAST(@FEC AS CHAR(10))
        FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
END
--CERRAR E INHABILITAR EL CURSOR
CLOSE C_CLIENTES
DEALLOCATE C_CLIENTES
GO
```

```
CODIGO    CLIENTE          DIRECCION                    TELEFONO          FECHA REG.
-------------------------------------------------------------------------------------
C00     FINSETH            AV. LOS VIÑEDOS 150          4342318           2008-12-10
C00     ORBI               AV. EMILIO CAVENECIA 225     4406335           2013-02-01
C00     SERVIEMSA          JR. COLLAGATE 522            75012403          2011-06-03
C00     ISSA               CALLE LOS AVIADORES 263      3725910           2012-09-12
C00     MASS               AV. TOMAS MARSANO 880        4446177           2013-10-01
C00     BERKER             AV. LOS PROCERES 521         3810322           2013-07-05
C00     FIDENZA            JR. EL NIQUEL 282            5289034           2012-10-02
C00     INTECH             AV. SAN LUIS 2619  5TO P     2249493           2011-01-07
C00     PROMINENT          JR. IQUIQUE 132              43233519          2010-06-11
C01     LANDU              AV.NICOLAS DE AYLLON 1453    3267840           2013-11-06
C01     FILASUR            AV. EL SANTUARIO 1189        4598175           2012-08-04
C01     SUCERTE            JR. GRITO DE HUAURA 114      4206434           2011-11-08
C01     HAYASHI            JR. AYACUCHO 359             42847990          2013-03-09
```

Script que permita implementar un cursor donde se imprima todos los registros de la tabla PRODUCTO dependiendo de la unidad.

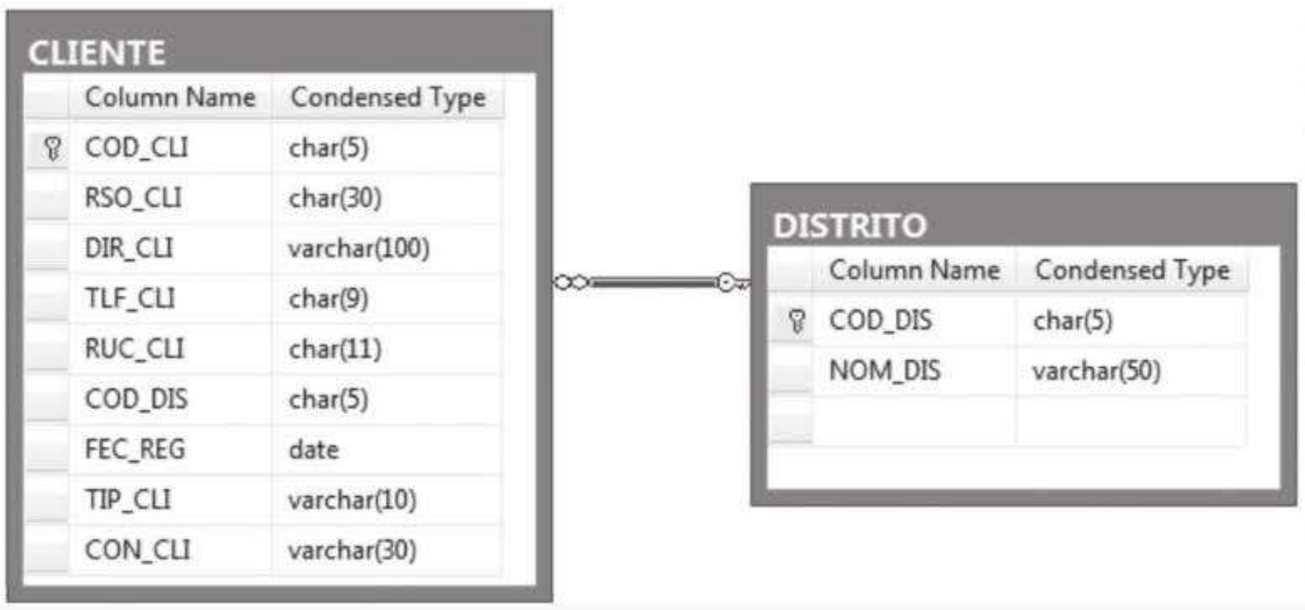
PRODUCTO

Column Name	Condensed Type
COD_PRO	char(5)
DES_PRO	varchar(50)
PRE_PRO	money
SAC_PRO	int
SMI_PRO	int
UNI_PRO	varchar(30)
LIN_PRO	varchar(30)
IMP_PRO	varchar(10)

```
DECLARE @COD CHAR(5),@DES CHAR(20),
                @PRE MONEY,@UNI CHAR(5),@TOTAL INT=0,@UNIDAD CHAR(5)='MLL'
DECLARE MICURSOR CURSOR
       FOR SELECT P.COD_PRO,P.DES_PRO,P.PRE_PRO,P.UNI_PRO
                        FROM PRODUCTO P
                        WHERE P.UNI_PRO=@UNIDAD
OPEN MICURSOR
FETCH MICURSOR INTO @COD,@DES,@PRE,@UNI
PRINT 'CODIGO    PRODUCTO                       PRECIO      UNIDAD'
PRINT '-------------------------------------------------------------'
WHILE @@FETCH_STATUS=0
BEGIN
   SET @TOTAL+=1
   PRINT @COD+SPACE(5)+@DES+SPACE(5)+CAST(@PRE AS CHAR(8))+SPACE(5)+@UNI
       FETCH MICURSOR INTO @COD,@DES,@PRE,@UNI
END
PRINT '-------------------------------------------------------------'
PRINT 'EL TOTAL DE PRODUCTOS DE LA '+
                @UNIDAD+' ES:'+CAST(@TOTAL AS CHAR(6))
CLOSE MICURSOR
DEALLOCATE MICURSOR
GO
```

```
CODIGO    PRODUCTO                       PRECIO      UNIDAD
-------------------------------------------------------------
P001      PAPEL BOND A-4 ATLAS            18.50      MLL
P002      PAPEL BOND OFICIO               37.00      MLL
P003      PAPEL BULKY                     12.00      MLL
P004      PAPEL PERIODICO 75 G            11.00      MLL
P024      LAPICERO 035 FINE                1.00      MLL
P025      LAPICERO 050 FINE                1.50      MLL
P026      LAPICERO 075 FINE                2.00      MLL
P027      REGLA 20CM                       2.00      MLL
P028      REGLA 30CM                       5.00      MLL
P029      REGLA FLEXIBLE                  10.00      MLL
-------------------------------------------------------------
EL TOTAL DE PRODUCTOS DE LA MLL   ES:10
```

Script que permita implementar un cursor donde se imprima los clientes por distrito.



```
--DECLARACION DE VARIABLES
DECLARE @COD CHAR(3),@CLI CHAR(15),@DIR CHAR(30),
                @TEL CHAR(15),@DIS CHAR(20),@FEC DATE
--DECLARANDO EL CURSOR
DECLARE C_CLIENTES CURSOR
        FOR SELECT C.COD_CLI, C.RSO_CLI,
                        C.DIR_CLI,C.TLF_CLI,D.NOM_DIS,C.FEC_REG
                        FROM CLIENTE C
                        JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS

--ABRIR EL CURSOR
OPEN C_CLIENTES
--OBTENER EL PRIMER REGISTRO
FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
PRINT 'CODIGO    CLIENTE      DIRECCION         TELEFONO      FECHA REG.'
PRINT '-----------------------------------------------------------------'

--RECORRER POR EL CURSOR
WHILE @@FETCH_STATUS=0
BEGIN
        PRINT
@COD+SPACE(5)+@CLI+SPACE(5)+@DIR+SPACE(5)+@TEL+SPACE(5)+CAST(@FEC AS
CHAR(10))
        FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
END
--CERRAR E INHABILITAR EL CURSOR
CLOSE C_CLIENTES
DEALLOCATE C_CLIENTES
GO

---------------CURSOR DISTRITO ----------------------------------------
DECLARE @IDE CHAR(3),@DIST CHAR(15)
DECLARE C_DISTRITO CURSOR FOR
        SELECT D.COD_DIS,D.NOM_DIS FROM DISTRITO D
OPEN C_DISTRITO
FETCH C_DISTRITO INTO @IDE,@DIST
WHILE @@FETCH_STATUS=0
BEGIN

------------------CURSOR CLIENTES ----------------------------------
        PRINT ''
        PRINT 'DISTRITO: '+@DIST
        DECLARE @COD CHAR(3),@CLI CHAR(15),@DIR CHAR(30),
                @TEL CHAR(15),@DIS CHAR(20),@FEC DATE
        DECLARE C_CLIENTES CURSOR FOR
                SELECT C.COD_CLI, C.RSO_CLI,
                        C.DIR_CLI,C.TLF_CLI,D.NOM_DIS,C.FEC_REG
                        FROM CLIENTE C
                        JOIN DISTRITO D ON C.COD_DIS=D.COD_DIS
                        WHERE C.COD_DIS=@IDE
        OPEN C_CLIENTES
```

```
FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
PRINT 'CODIGO    CLIENTE    DIRECCION    TELEFONO   DISTRITO FECHAREG'
PRINT '-----------------------------------------------------------'
       WHILE @@FETCH_STATUS=0
       BEGIN
        PRINT SPACE(10)+@COD+SPACE(5)+@CLI+SPACE(5)+
        @DIR+SPACE(5)+@TEL+SPACE(5)+@DIS+SPACE(5)+CAST(@FEC AS CHAR(10))
              FETCH C_CLIENTES INTO @COD,@CLI,@DIR,@TEL,@DIS,@FEC
       END
       CLOSE C_CLIENTES
       DEALLOCATE C_CLIENTES
-------------------------------------------------------------------------
       FETCH C_DISTRITO INTO @IDE,@DIST
END
CLOSE C_DISTRITO
DEALLOCATE C_DISTRITO
GO
```

```
DISTRITO: SURCO
     CODIGO    CLIENTE    DIRECCION    TELEFONO    DISTRITO     FECHAREG
     ------------------------------------------------------------
     C00     ISSA               CALLE LOS AVIADORES 263          3725910          SURCO                2012-09-12
     C02     TAI LOY            AV. UNIVERSITARIA 3424           4529636          SURCO                2017-10-25

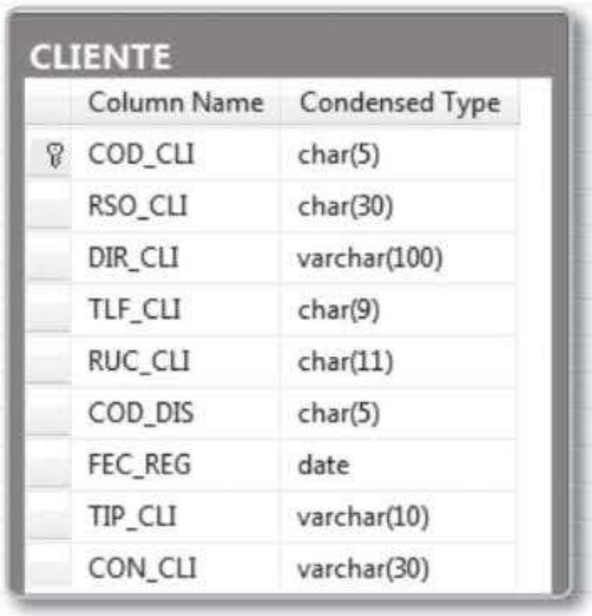
DISTRITO: JESUS MARIA
     CODIGO    CLIENTE    DIRECCION    TELEFONO    DISTRITO     FECHAREG
     ------------------------------------------------------------
     C02     CRAMER             JR. MARISCAL MILLER 1131         4719061          JESUS MARIA          2010-08-11

DISTRITO: SAN ISIDRO
     CODIGO    CLIENTE    DIRECCION    TELEFONO    DISTRITO     FECHAREG
     ------------------------------------------------------------

DISTRITO: LA MOLINA
     CODIGO    CLIENTE    DIRECCION    TELEFONO    DISTRITO     FECHAREG
     ------------------------------------------------------------
     C00     ORBI               AV. EMILIO CAVENECIA 225         4406335          LA MOLINA            2013-02-01

DISTRITO: SAN MIGUEL
     CODIGO    CLIENTE    DIRECCION    TELEFONO    DISTRITO     FECHAREG
     ------------------------------------------------------------
     C00     FINSETH            AV. LOS VIÑEDOS 150              4342318          SAN MIGUEL           2008-12-10
     C00     SERVIEMSA          JR. COLLAGATE 522                75012403         SAN MIGUEL           2011-06-03
     C00     BERKER             AV. LOS PROCERES 521             3810322          SAN MIGUEL           2013-07-05
     C01     LANDU              AV.NICOLAS DE AYLLON 1453        3267840          SAN MIGUEL           2013-11-06
     C01     SUCERTE            JR. GRITO DE HUAURA 114          4206434          SAN MIGUEL           2011-11-08
```

Script que permita implementar un cursor donde se imprima los clientes.



CLIENTE

	Column Name	Condensed Type
🔑	COD_CLI	char(5)
	RSO_CLI	char(30)
	DIR_CLI	varchar(100)
	TLF_CLI	char(9)
	RUC_CLI	char(11)
	COD_DIS	char(5)
	FEC_REG	date
	TIP_CLI	varchar(10)
	CON_CLI	varchar(30)

```
DECLARE @N INT=5
DECLARE MICURSOR CURSOR SCROLL
        FOR SELECT * FROM CLIENTE
OPEN MICURSOR
FETCH FIRST FROM MICURSOR
FETCH ABSOLUTE @N FROM MICURSOR
FETCH LAST FROM MICURSOR
CLOSE MICURSOR
DEALLOCATE MICURSOR
GO
```

```
COD_CLI RSO_CLI       DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------- ------------- -------------------------- --------- ----------- ------- ---------- ---------- ----------------
C001    FINSETH       AV. LOS VIÑEDOS 150        4342318   48632081    D05     2008-12-10 1          ALICIA BARRETO

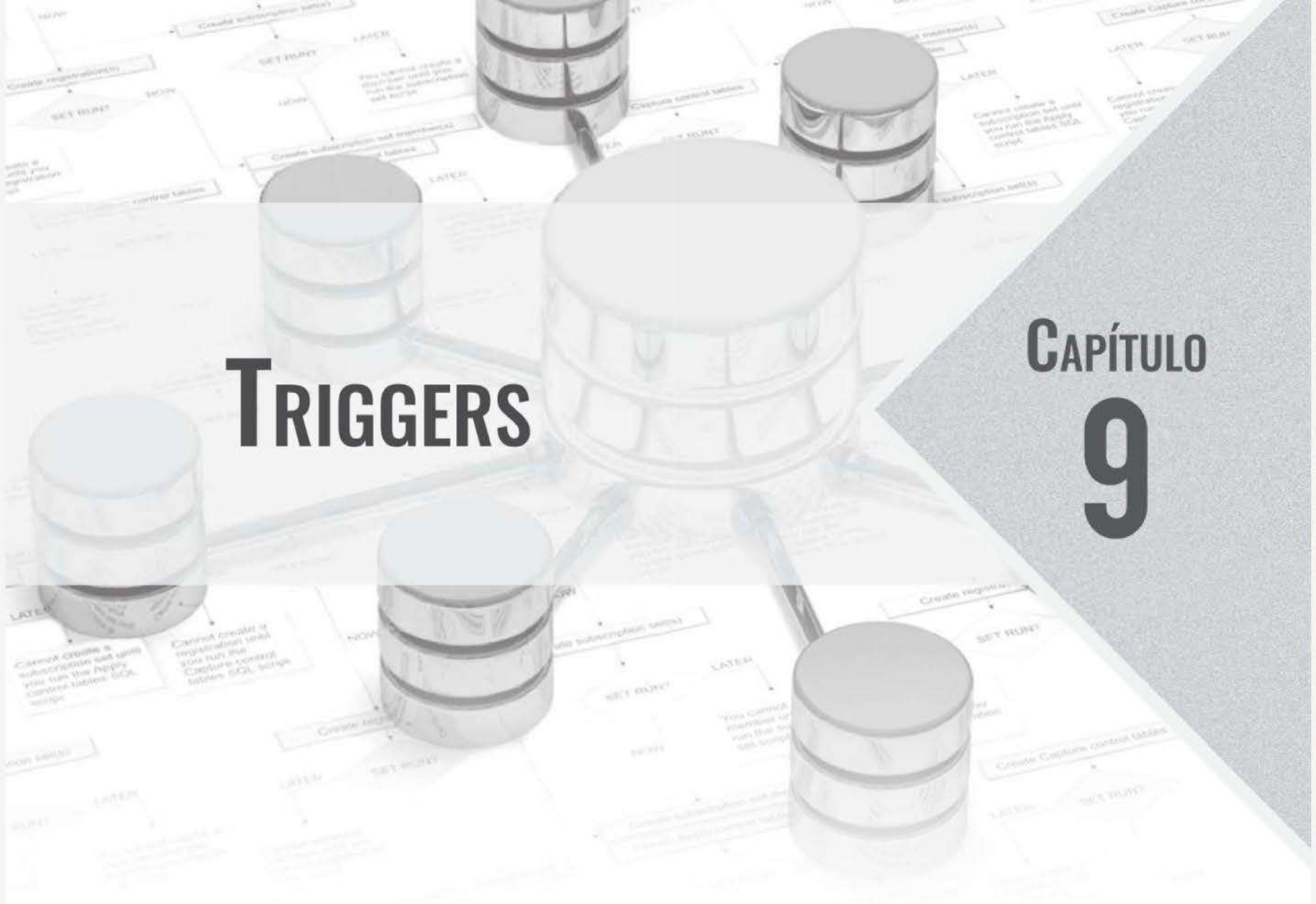
(1 row(s) affected)

COD_CLI RSO_CLI      DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------- ------------ -------------------------- --------- ----------- ------- ---------- ---------- ----------------
C005    MASS         AV. TOMAS MARSANO 880      4446177   83175942    D14     2013-10-01 1          KATIA ARMEJO

(1 row(s) affected)

COD_CLI RSO_CLI      DIR_CLI                    TLF_CLI   RUC_CLI     COD_DIS FEC_REG    TIP_CLI    CON_CLI
------- ------------ -------------------------- --------- ----------- ------- ---------- ---------- ----------------
C021    TAI LOY      AV. UNIVERSITARIA 3424     4529636   51551585745 D01     2017-10-25 1          JUAN GARCIA

(1 row(s) affected)
```



CAPÍTULO 9

TRIGGERS

Competencia tecnológica:

- Implementar disparadores sobre la base de datos.
- Implementar disparadores sobre los datos.

Contenido:

9.1 Triggers
9.2 Tipos de triggers
9.3 Opciones de triggers

9.1 Triggers

Un trigger (o disparador), en una base de datos, es un procedimiento que se ejecuta cuando se cumple una determinada condición al realizar una operación. Dependiendo de la base de datos, los triggers pueden ser de inserción (INSERT), actualización (UPDATE) o borrado (DELETE). Algunas bases de datos pueden ejecutar triggers al crear, borrar o editar usuarios, tablas, bases de datos u otros objetos.

Usos:

- Permiten registrar, auditar y monitorear los procesos de cambio de valores a las tablas de la base de datos activa.
- Pueden validar los valores aprobando o negando acciones realizadas por las sentencias de tipo DML.
- Pueden preservar la consistencia y claridad de los valores ejecutando acciones relacionadas con los objetos de tipo tabla de la base de datos activa.

Ventajas:

- Ofrecen chequeos de seguridad en valores con respecto a los objetos tablas de una base de datos.
- Fuerzan restricciones dinámicas de integridad de datos y de integridad referencial.
- Aseguran que las operaciones relacionadas se realicen juntas, en forma implícita.
- Ofrecen un mayor control sobre los objetos de una base de datos.

Desventajas:

- Se tiene que programar anticipadamente lo que debe realizar un trigger.
- Siendo un procedimiento no se puede invocar directamente.
- Los triggers siempre serán creados para un conjunto de registros y no para uno solo, ya que dispara por operación DML.
- Los triggers no son aplicables en tablas temporales.

9.2 Tipos de triggers

- **Triggers DML:** Se aplican a sentencias que controlan los registros de una tabla de datos.
- **Triggers DDL:** Se aplican a objetos contenidos del objeto de una base de datos.

Formato para triggers DML:

```
CREATE TRIGGER <NOMBRE_TRIGGER>
ON <TABLE>
   FOR | AFTER | INSTEAD OF
   [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ]
AS
   SENTENCIA_SQL
```

Formato para triggers DDL:

```
CREATE TRIGGER <NOMBRE_TRIGGER>
ON [ ALL SERVER | DATABASE ]
FOR | AFTER
AS
   SENTENCIA SQL
```

Donde:

- **TABLE / VIEW:** Es la tabla o vista en que se ejecuta el desencadenador DML; algunas veces se denomina tabla del desencadenador o vista del desencadenador. Especificar el nombre completo de la tabla o vista es opcional. Solo se puede hacer referencia a una vista mediante un desencadenador INSTEAD OF.
- **DATABASE:** Aplica el ámbito de un desencadenador DDL a la base de datos actual. Si se especifica, el desencadenador se activa cada vez que *event_type* o *event_group* tienen lugar en la base de datos actual.
- **FOR | AFTER:** AFTER especifica que el desencadenador DML solo se activa cuando todas las operaciones especificadas en la instrucción SQL desencadenadora se han ejecutado correctamente. Además, todas las acciones referenciales en cascada y las comprobaciones de restricciones deben ser correctas para que este desencadenador se ejecute. AFTER es el valor predeterminado cuando solo se especifica la palabra clave FOR. Los desencadenadores AFTER no se pueden definir en las vistas.
- **INSTEAD OF:** Especifica que se ejecuta el desencadenador DML en vez de la instrucción SQL desencadenadora, por lo que se suplantan las acciones de las instrucciones desencadenadoras. INSTEAD OF no se puede especificar para los desencadenadores DDL. Como máximo, se puede definir un desencadenador INSTEAD OF por cada instrucción INSERT, UPDATE o DELETE en cada tabla o vista. No obstante, en las vistas es posible definir otras vistas que tengan su propio desencadenador INSTEAD OF.

9.3 Opciones de triggers

```
Mostrar todos los triggers de la base de datos.
SELECT * FROM SYS.TRIGGERS
GO

Eliminar un trigger.
DROP TRIGGER <NOMBRE TRIGGER>
GO

Inhabilitar un trigger: La inhabilitación indica que la tabla podrá
ejecutar sus sentencias DML sin mayor control del trigger inhabilitado.
DISABLE TRIGGER <NOMBRE CURSOR>
 ON <TABLA>
GO
```

```
Habilitar un trigger.
ENABLE TRIGGER <NOMBRE CURSOR>
 ON <TABLA>
GO

Inhabilitar todos los trigger.
ALTER TABLE PASAJERO
DISABLE TRIGGER ALL
```

A continuación, algunos casos del uso de triggers para DML:

```
Implemente un trigger que permita mostrar un mensaje cada vez que se
inserte o actualice un registro en la tabla CLIENTE.
```

CLIENTE

Column Name	Condensed Type
COD_CLI	char(5)
RSO_CLI	char(30)
DIR_CLI	varchar(100)
TLF_CLI	char(9)
RUC_CLI	char(11)
COD_DIS	char(5)
FEC_REG	date
TIP_CLI	varchar(10)
CON_CLI	varchar(30)

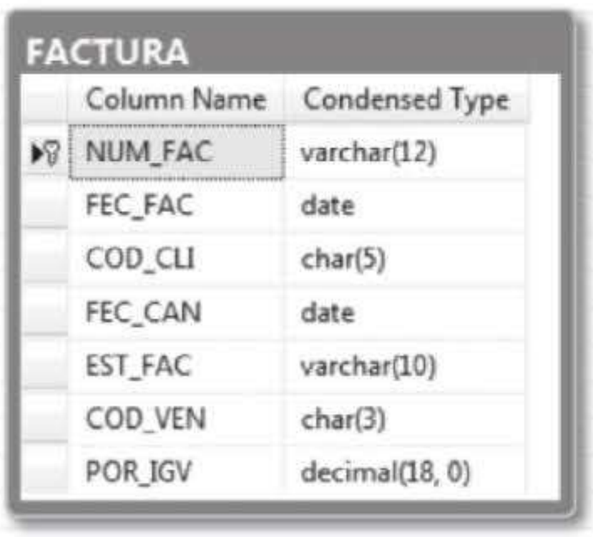
```
--Creando el trigger
CREATE TRIGGER TX_MENSAJE_CLIENTE
ON CLIENTE
FOR INSERT, UPDATE
AS
PRINT 'MENSAJE DISPARADO POR LA INSERCION O
                        ACTUALIZACION DE LA TABLA CLIENTE'
GO

--Probar el trigger con inserción
INSERT INTO CLIENTE
       VALUES('C022','HINOJOSA','AV. LIMA 131',
                   '5214521','42412155','D01','08/01/2015','1',
                   'JOSE RUPPER')
GO
--Probar el trigger con actualización
UPDATE CLIENTE
       SET FEC_REG='10/12/2015'
       WHERE COD_CLI='C022'
GO
```

```
MENSAJE DISPARADO POR LA INSERCION O
            ACTUALIZACION DE LA TABLA CLIENTE

(1 row(s) affected)
```

Implemente un trigger que permita crear un histórico de los registros realizados a la tabla FACTURA, en la cual por cada registro realizado por un cliente se deberá enviar el código del cliente y el conteo total de facturas realizadas por dicho cliente a una nueva llamada FACTURASXCLIENTE.




```
--Creando la tabla
CREATE TABLE FACTURASXCLIENTE(
        CLIENTE VARCHAR(30) NOT NULL,
        TOTAL   INT
)
GO

--Creando el trigger
CREATE TRIGGER TX_FACTURASxCLIENTE
ON FACTURA
AFTER INSERT
AS
        DECLARE @TOTAL INT,@CLI VARCHAR(30)
        SELECT @TOTAL=COUNT(*)
                FROM INSERTED,FACTURA
                WHERE FACTURA.COD_CLI=INSERTED.COD_CLI

        SELECT @CLI=INSERTED.COD_CLI FROM INSERTED


        IF EXISTS(SELECT * FROM FACTURASxCLIENTE WHERE CLIENTE=@CLI)
                UPDATE FACTURASXCLIENTE
                        SET TOTAL=@TOTAL
                        WHERE CLIENTE=@CLI
                ELSE
                        INSERT INTO FACTURASXCLIENTE VALUES(@CLI,@TOTAL)
GO
```

```
--PROBAR
INSERT INTO FACTURA VALUES('FA021','06-02-2013','C013',
                           '10-03-2013','2','V02','0.19')
GO
```

```
(1 row(s) affected)
(1 row(s) affected)
```

```
--COMPROBANDO LAS TABLAS
SELECT * FROM FACTURA
GO

SELECT * FROM FACTURASXCLIENTE
GO
```

CLIENTE	TOTAL
C013	2

(1 row(s) affected)

Implemente un trigger que permita controlar los registros de la tabla DISTRITO; si el nombre de dicho distrito fue registrado, entonces deberá mostrar con qué código fue registrado; en caso contrario, emitir un mensaje de "Distrito registrado correctamente".

DISTRITO

	Column Name	Condensed Type
🔑	COD_DIS	char(5)
	NOM_DIS	varchar(50)

```
CREATE TRIGGER TX_VALIDA_DISTRITO
ON DISTRITO
FOR INSERT
AS
 IF (SELECT COUNT(*) FROM INSERTED,DISTRITO
                          WHERE INSERTED.NOM_DIS=DISTRITO.NOM_DIS)>1
        BEGIN
        DECLARE @DISTRITO VARCHAR(30),@COD CHAR(5)
        SELECT @DISTRITO=NOM_DIS FROM INSERTED

        ROLLBACK

        SELECT @COD=COD_DIS FROM DISTRITO
                    WHERE DISTRITO.NOM_DIS=@DISTRITO

        PRINT 'NOMBRE DE DISTRITO YA REGISTRADO EN LA TABLA'
        PRINT ''
        PRINT 'EL DISTRITO '+@DISTRITO+
             ' SE ENCUENTRA REGISTRADO CON EL CODIGO: '+@COD

       END
ELSE
```

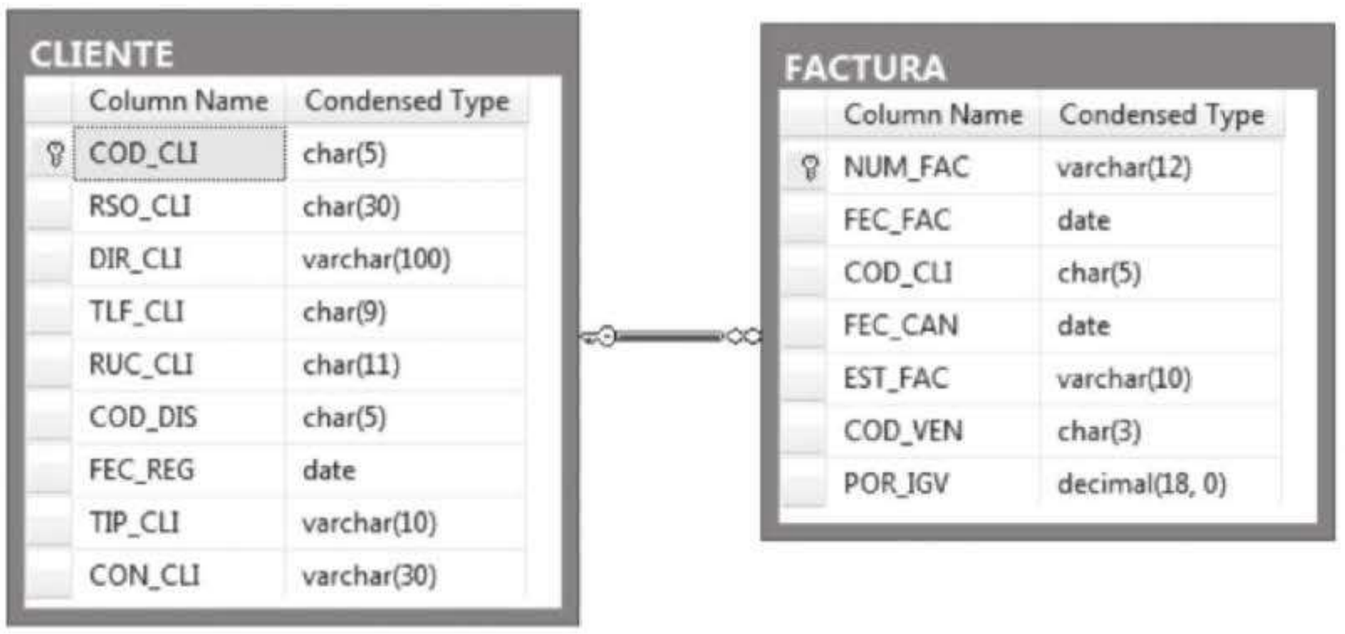
```
PRINT 'DISTRITO REGISTRADO CORRECTAMENTE'
GO

--PRUEBA
INSERT INTO DISTRITO VALUES('D38','SANTA MARIA')
GO
```

```
NOMBRE DE DISTRITO YA REGISTRADO EN LA TABLA

EL DISTRITO SANTA MARIA SE ENCUENTRA
REGISTRADO CON EL CODIGO: D37
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger.
The batch has been aborted.
```

Implemente un trigger que permita controlar la eliminación de un registro de la tabla CLIENTE; en la cual, si dicho cliente tiene facturas registradas, no permita su eliminación mostrando un mensaje; en caso contrario, mostrar el mensaje de eliminación correcta.

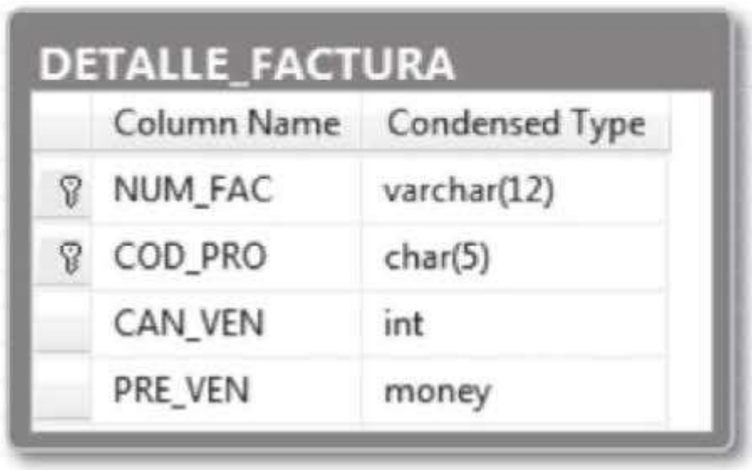



```
CREATE TRIGGER TX_ELIMINACLIENTE
ON CLIENTE
INSTEAD OF DELETE
AS
        IF EXISTS(SELECT * FROM FACTURA
                        WHERE FACTURA.COD_CLI=(SELECT DELETED.COD_CLI
                                                FROM DELETED))
        BEGIN
        ROLLBACK TRANSACTION
        PRINT 'EL CLIENTE TIENE FACTURAS REGISTRADAS, NO PUEDE ELIMINARSE'
        END
        ELSE
         PRINT 'CLIENTE ELIMINADO CORRECTAMENTE'
GO
```

```
--PRUEBA
DELETE CLIENTE WHERE COD_CLI='C001'
```

```
EL CLIENTE TIENE FACTURAS REGISTRADAS, NO PUEDE ELIMINARSE
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger.
The batch has been aborted.
```

Implemente un trigger que permita controlar el registro de un detalle de factura, en la cual se evalúe la cantidad registrada para que no se registre un valor inferior a cero en la columna cantidad de venta.

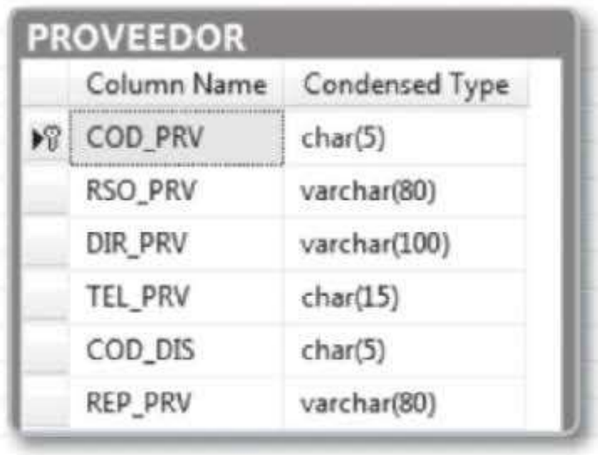



```
CREATE TRIGGER TX_VALIDADETALLE
ON DETALLE_FACTURA
FOR INSERT
AS
        IF (SELECT CAN_VEN FROM INSERTED)<=0
        BEGIN
                ROLLBACK TRANSACTION
                PRINT 'NO PUEDE REGISTRAR CANTIDADES
                           INFERIORES A CERO'
        END
        ELSE
                PRINT 'CANTIDAD REGISTRADA CORRECTAMENTE'
GO

--PRUEBA:
INSERT INTO DETALLE_FACTURA VALUES('FA020','P013',-5,15)
GO
```

```
NO PUEDE REGISTRAR CANTIDADES
INFERIORES A CERO
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

Implemente un trigger que permita crear una réplica de los registros insertados en la tabla PROVEEDOR; para dicho proceso debe implementar una nueva tabla llamada PROVEEDOR_BAK con las mismas columnas que la tabla PROVEEDOR.




```
--Validando la existencia de la tabla
IF OBJECT_ID('PROVEEDOR_BAK') IS NOT NULL
BEGIN
	DROP TABLE PROVEEDOR_BAK
END
GO

--Creando la tabla PROVEEDOR copia
CREATE TABLE PROVEEDOR_BAK(
	COD_PRV CHAR(5)		NOT NULL,
	RSO_PRV VARCHAR(80)	NOT NULL,
	DIR_PRV VARCHAR(100)	NOT NULL,
	TEL_PRV CHAR(15)	NULL,
	COD_DIS CHAR(5)		NOT NULL,
	REP_PRV VARCHAR(80)	NOT NULL
)
GO

--Creando el trigger
CREATE TRIGGER TX_REPLICAPROVEEDOR
ON PROVEEDOR
AFTER INSERT
AS
BEGIN
	INSERT PROVEEDOR_BAK
		SELECT * FROM INSERTED
END
GO

--PRUEBA
INSERT INTO PROVEEDOR
	VALUES('PR21','BARI','AV. ARNALDO MARQUEZ 1219',
		NULL,'D02','VANESA QUINTANA')
GO

--COMPROBAR LOS LISTADOS
SELECT * FROM PROVEEDOR
GO
SELECT * FROM PROVEEDOR_BAK
GO
```

A continuación, algunos casos del uso de triggers para DDL:

```
Implemente un trigger que permita bloquear la aplicación de las
instrucciones ALTER y DROP TABLE en la base de datos BDVENTAS.

--Validar la existencia del trigger
IF EXISTS (SELECT * FROM SYS.TRIGGERS
                        WHERE PARENT_CLASS = 0 AND
                        NAME = 'POLITICA')
        DROP TRIGGER POLITICA ON DATABASE
GO

--Creando el trigger
CREATE TRIGGER POLITICA
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    ROLLBACK TRANSACTION
        PRINT 'SI DESEA ELIMINAR O ALTERAR SUS TABLAS'
        PRINT 'PRIMERO DEBE INHABILITAR EL TRIGGER POLITICA'
GO

--PRUEBA
DROP TABLE DETALLE_FACTURA
GO
```

```
SI DESEA ELIMINAR O ALTERAR SUS TABLAS
PRIMERO DEBE INHABILITAR EL TRIGGER POLITICA
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger. The batch has been aborted.
```



XML CON SQL E IMÁGENES

CAPÍTULO 10

Competencia tecnológica:

- Entender e implementar scripts de operaciones con archivos XML, haciendo uso de variables de tipo XML y usando archivos de tipo XML para la manipulación de registros tanto internos como externos de XML.
- Administrar imágenes dentro de una tabla desde la aplicación SQL Image Viewer.

Contenido:

10.1 Introducción
10.2 Modelo de datos relacionales o XML
10.3 Ventajas de almacenar valores en XML
10.4 Elección de la tecnología XML
10.5 Tipo de datos XML
10.6 Columnas y variables XML
10.7 FOR XML y OPENXML
10.8 Instrucción OpenRowSet

10.1 Introducción

En ocasiones anteriores al SQL Server 2005, para almacenar un valor de tipo XML, se tenía que declarar como tipo Text, desde esa versión hacia adelante se puede implementar el tipo de datos XML.

El tipo de datos XML permite almacenar documentos y fragmentos XML en una base de datos de SQL Server. Un fragmento XML es una instancia XML en la que falta un solo elemento de nivel superior.

El tipo de datos XML y los métodos asociados ayudan a integrar el XML en el marco relacional de SQL Server.

10.2 Modelo de datos relacionales o XML

Si los datos están muy estructurados con un esquema conocido, el modelo relacional tiene más probabilidades de funcionar mejor para el almacenamiento de datos. SQL Server proporciona la funcionalidad y las herramientas necesarias. Por otra parte, si los datos están semiestructurados o no están estructurados, o no se conoce su estructura, se debe contemplar la posibilidad de crear un modelo para los datos.

XML es una buena opción si se desea un modelo independiente de la plataforma que garantice la portabilidad de los datos mediante el uso de marcado estructural y semántico. Además, es una opción apropiada si se cumplen algunas de las siguientes propiedades:

- Los datos están dispersos o no se conoce la estructura de los mismos, o la estructura de los datos puede cambiar de manera importante en el futuro.
- Los datos representan una jerarquía de inclusión en lugar de referencias entre entidades, y pueden ser recursivos.
- El orden es inherente a los datos.
- Se desea realizar consultas en los datos o actualizar parte de ellos basándose en su estructura.

Si no se cumple ninguna de estas condiciones, se debe utilizar el modelo de datos relacional. Por ejemplo, si los datos tienen formato XML, pero la aplicación únicamente utiliza la base de datos para almacenar y recuperar los datos, solo necesitará una columna [N]VARCHAR(MAX).

10.3 Ventajas de almacenar valores en XML

Siempre presentará muchas ventajas mientras se sepa qué hacer con los resultados que puede emitir un trabajo con XML. Así, se tiene:

- El Motor de base de datos puede determinar si los datos contenidos en una columna XML tienen un formato correcto.
- Se pueden realizar consultas y actualizaciones detalladas en los datos XML.
- Se pueden compartir los datos XML, hacer consultas en ellos y modificarlos de forma eficaz y con transacciones.
- Se puede extraer alguna sección de un documento XML, o insertar una nueva sección sin reemplazar todo el documento.

- Se puede operar con datos relacionales y datos XML dentro de una misma aplicación.
- En la actualidad, el XML es totalmente compatible con la mayoría de lenguajes de programación como NET, Java o PHP.
- Un XML tiene acceso a SOAP, ADO.NET y OLE DB.
- También se puede usar para realizar copias de seguridad, recuperaciones y réplicas de los registros de las tablas en una base de datos.

10.4 Elección de la tecnología XML

La elección de la tecnología XML, XML nativo frente a vista XML, en general depende de los siguientes factores:

- **Opciones de almacenamiento:** Los datos XML pueden ser más apropiados para el almacenamiento de objetos grandes (por ejemplo, el manual de un producto) o más sensibles al almacenamiento en columnas relacionales (por ejemplo, un elemento de línea convertido a XML). Cada opción de almacenamiento preserva la fidelidad del documento en distinta medida.
- **Funciones de consultas:** Es posible que una opción de almacenamiento parezca más apropiada que otra en función de la naturaleza de las consultas y del nivel de detalle con que se desea consultar los datos XML. La consulta detallada de los datos XML (por ejemplo, evaluación de predicados en nodos XML) se admite en diversos grados en las dos opciones de almacenamiento.
- **Indizar datos XML:** Tal vez se desee indizar los datos XML para acelerar el rendimiento de las consultas XML. Las opciones de indización varían según las opciones de almacenamiento; por ello, es necesario hacer la elección apropiada para optimizar la carga de trabajo.
- **Funciones para la modificación de datos:** Algunas cargas de trabajo implican una modificación detallada de los datos XML. Este es el caso de agregar una sección nueva a un documento. Sin embargo, otras cargas de trabajo, como el contenido web, no implican la modificación detallada de los datos XML. La compatibilidad con el lenguaje de modificación de datos puede ser importante para la aplicación.
- **Compatibilidad con esquemas:** Los datos XML se pueden describir mediante un esquema que puede ser o no un documento de esquema XML. La compatibilidad con XML enlazado a un esquema depende de la tecnología XML.

A continuación, algunos casos para el uso de XML:

- Script que permita crear una tabla llamada CLIENTEXML en donde se registren el CÓDIGO y sus tipos de razón social dentro de una columna de tipo XML.

CLIENTEXML

Column Name	Condensed Type
CODIGO	int
RAZON_SOCIAL	xml

```
IF OBJECT_ID('CLIENTEXML')IS NOT NULL
        DROP TABLE CLIENTEXML
GO
```

```
--Creando la tabla con tipo XML
CREATE TABLE CLIENTEXML(
        CODIGO          INT IDENTITY NOT NULL PRIMARY KEY,
        RAZON_SOCIAL    XML
)
GO

--Insertar un registro
INSERT INTO CLIENTEXML
        VALUES('<rsocial>MIYASATO HNOS.</rsocial>')
GO

--Listar la tabla XML
SELECT * FROM CLIENTEXML
GO
```

Se inicia creando la tabla CLIENTEXML, definiendo en este caso la columna CÓDIGO que será autonumérico y llave de los registros, y la columna RAZÓN SOCIAL que tendrá el tipo de datos XML. La imagen siguiente muestra la consulta realizada a la tabla CLIENTEXML:

```
(1 row(s) affected)
CODIGO      RAZON_SOCIAL
----------- ------------------------------------
1           <rsocial>MIYASATO HNOS.</rsocial>

(1 row(s) affected)
```

Si se necesita registrar dos tipos de razón social por cada registro de clientes, entonces se debe ejecutar el siguiente script:

```
DECLARE @PXML AS XML
SET @PXML = '<rsocial>MIYASATO HNOS SA.</rsocial>
                <rsocial>MIYASATO HNOS srltda.</rsocial>
                <rsocial>MIYASATO SAC.</rsocial>'
INSERT INTO CLIENTEXML(RAZON_SOCIAL) VALUES (@PXML)

SET @PXML = '<rsocial>ELECTRAPERU NOS srltda.</rsocial>
                <rsocial>ELECTRAPERU SAC.</rsocial>'
INSERT INTO CLIENTEXML(RAZON_SOCIAL) VALUES (@PXML)
GO
--PRUEBA
SELECT * FROM CLIENTEXML
GO
```

El resultado de visualizar los registros de la tabla CLIENTEXML lo muestra la siguiente imagen:

```
CODIGO      RAZON_SOCIAL
----------- ------------------------------------------------------------------------------------------------------------
1           <rsocial>MIYASATO HNOS.</rsocial>
2           <rsocial>MIYASATO HNOS SA.</rsocial><rsocial>MIYASATO HNOS srltda.</rsocial><rsocial>MIYASATO SAC.</rsocial>
3           <rsocial>ELECTRAPERU NOS srltda.</rsocial><rsocial>ELECTRAPERU SAC.</rsocial>

(3 row(s) affected)
```

Ahora se procederá a visualizar el contenido de cada valor de la columna razón social para comprobar si se cumplió con registrar la razón social de cada cliente en una misma fila:

```
<rsocial>MIYASATO HNOS SA.</rsocial>
<rsocial>MIYASATO HNOS srltda.</rsocial>
<rsocial>MIYASATO SAC.</rsocial>
```

Para lograr visualizar los valores XML registrados a un determinado cliente será cuestión de dar un click sobre el link del valor registrado en la columna RAZÓN SOCIAL.

- Script que permita crear una tabla llamada CLIENTEXML en donde se registren dos filas al mismo tiempo; esta vez se implementará dentro del XML la columna CÓDIGO y RAZÓN SOCIAL, por lo menos dos tipos de razón social por cada cliente.

CLIENTEXML

Column Name	Condensed Type
CODIGO	int
RAZON_SOCIAL	xml

```
--Validando la existencia de la tabla
IF OBJECT_ID('CLIENTEXML')IS NOT NULL
	DROP TABLE CLIENTEXML
GO

--Creando la tabla con tipo XML
CREATE TABLE CLIENTEXML(
	CODIGO			INT IDENTITY NOT NULL PRIMARY KEY,
	RAZON_SOCIAL	XML
)
GO

DECLARE @PXML AS XML
SET @PXML = '<clientes>
				<cliente>
					<codigo>P0001</codigo>
					<rsocial>mtorres peru SAC</rsocial>
					<rsocial>mtorres SRLTDA</rsocial>
				</cliente>
				<cliente>
					<codigo>P0002</codigo>
					<rsocial>llazaro SAC </rsocial>
					<rsocial>llazaro SRTLDA</rsocial>
				</cliente>
			</clientes>'
INSERT INTO CLIENTEXML(RAZON_SOCIAL) VALUES (@PXML)
GO

--PRUEBA
SELECT * FROM CLIENTEXML
GO
```

```
--PRUEBA
SELECT * FROM CLIENTEXML
GO
```

Se inicia por validar la existencia de la tabla CLIENTEXML; luego, se crea la tabla y, finalmente, se implementa el script que permite insertar dos registros en una sola sentencia; esta vez se usará una variable local del tipo XML que tenga asignado todo el script del XML y este será enviado por la instrucción INSERT INTO de la tabla CLIENTEXML.

La imagen siguiente muestra el *linkeo* realizado al valor de la columna RAZÓN SOCIAL de la tabla CLIENTEXML.

```
(1 row(s) affected)
CODIGO      RAZON_SOCIAL
----------- ------------------------------------------------------------------------------------
1           <clientes><cliente><codigo>P0001</codigo><rsocial>mtorres peru SAC</rsocial>
            <rsocial>mtorres SRLTDA</rsocial></cliente><cliente><codigo>P0002</codigo>
            <rsocial>llazaro SAC </rsocial><rsocial>llazaro SRTLDA</rsocial></cliente></clientes>

(1 row(s) affected)
```

```
<clientes>
  <cliente>
    <codigo>P0001</codigo>
    <rsocial>mtorres peru SAC</rsocial>
    <rsocial>mtorres SRLTDA</rsocial>
  </cliente>
  <cliente>
    <codigo>P0002</codigo>
    <rsocial>llazaro SAC </rsocial>
    <rsocial>llazaro SRTLDA</rsocial>
  </cliente>
</clientes>
```

10.5 Tipo de datos XML

Un tipo de datos XML permite almacenar documentos y fragmentos XML en una base de datos de SQL Server y también puede asociar una colección de esquemas XML con una columna, un parámetro o una variable del tipo de datos XML. Los esquemas de la colección se utilizan para validar y asignar un tipo a las instancias XML. En este caso, se dice que el XML tiene un tipo.

El tipo de datos XML y los métodos asociados ayudan a integrar el XML en el marco relacional de SQL Server. La lista de métodos que presenta este tipo de datos es la siguiente:

- **Query()**: Permite mostrar el contenido de una columna de tipo XML.
- **Value()**: Permite recuperar un valor de tipo SQL de una instancia XML.
- **Exists()**: Permite determinar si una consulta de tipo XML devuelve un resultado vacío.
- **Nodes()**: Permite dividir en varias filas, es decir, se propaga partes de documentos XML en conjunto de filas.

10.6 Columnas y variables XML

El tipo de datos XML es un tipo de datos integrado de SQL Server y es similar a otros tipos integrados como Int o Varchar. Al igual que ocurre con otros tipos integrados, se puede utilizar el tipo de datos XML como un tipo de columna al crear una tabla con un tipo de variable, un tipo de parámetro, un tipo de valor devuelto por una función CAST o CONVERT.

Formato:

```
CREATE TABLE NOMBRE_TABLA(COLUMNA XML)
DECLARE @X XML
```

Tanto en la creación de una columna o una variable se especifica con XML, ya que es un tipo integrado del Motor de base de datos.

A continuación, algunos casos del uso de XML:

- Script que permita listar los registros de la tabla CLIENTEXML. Use el método QUERY.

CLIENTEXML

	Column Name	Condensed Type
🔑	CODIGO	int
	RAZON_SOCIAL	xml

```
Primero se debe analizar la estructura XML con la que se compone un
registro de la tabla CLIENTEXML.

<clientes>
        <cliente>
                <codigo>P0001</codigo>
                <rsocial>mtorres peru SAC</rsocial>
                <rsocial>mtorres SRLTDA</rsocial>
        </cliente>
        <cliente>
                <codigo>P0002</codigo>
                <rsocial>llazaro SAC </rsocial>
                <rsocial>llazaro SRTLDA</rsocial>
        </cliente>
</clientes>

Como se notará, el esquema principal del XML está conformado por CLIENTES
como primer nivel, luego viene CLIENTE como un nodo de CLIENTES; aquí sí
hay que tener muchísimo cuidado con la sensibilidad de las mayúsculas
y minúsculas, ya que por lo general todo script XML está compuesto en
minúsculas, esto último referido a la estructura XML, no a los valores
contenidos entre sus etiquetas.

SELECT CODIGO,RAZON_SOCIAL.query('clientes') AS [RAZON SOCIAL]
        FROM CLIENTEXML
GO
```

El resultado de la consulta se muestra en la siguiente imagen:

```
CODIGO      RAZON SOCIAL
----------- ------------------------------------------------------------------------------------
1           <clientes><cliente><codigo>P0001</codigo><rsocial>mtorres peru SAC</rsocial>
            <rsocial>mtorres SRLTDA</rsocial></cliente><cliente><codigo>P0002</codigo>
            <rsocial>llazaro SAC </rsocial><rsocial>llazaro SRTLDA</rsocial></cliente></clientes>

(1 row(s) affected)
```

- Script que permita mostrar el código del cliente 1 y su segunda razón social registrado en la tabla CLIENTEXML. Use la función VALUE del tipo de datos XML.

CLIENTEXML

	Column Name	Condensed Type
🔑	CODIGO	int
	RAZON_SOCIAL	xml

```
DECLARE @PXML AS XML
SET @PXML = '<clientes>
                        <cliente>
                                <codigo>P0001</codigo>
                                <rsocial>mtorres peru SAC</rsocial>
                                <rsocial>mtorres SRLTDA</rsocial>
                        </cliente>
                        <cliente>
                                <codigo>P0002</codigo>
                                <rsocial>llazaro SAC </rsocial>
                                <rsocial>llazaro SRTLDA</rsocial>
                        </cliente>
                </clientes>'
DECLARE @CODIGO CHAR(5),@RAZON VARCHAR(30)

SELECT  @CODIGO=@PXML.value('(clientes/cliente/codigo)[1]','char(5)'),
        @RAZON=@PXML.value('(clientes/cliente/rsocial)[2]','varchar(30)')

SELECT @CODIGO AS [CODIGO],@RAZON AS [RAZON SOCIAL]
GO
```

Se inicia declarando la variable @PXML que tendrá asignado un esquema XML que corresponde a dos códigos de clientes con sus respectivos tipos de razón social.

Luego se declara la variable @CODIGO que tendrá la misión de capturar el valor del código del cliente, y @RAZON obtendrá una razón social especificada desde la variable @PXML. Aquí mismo hay que notar que se usa la función VALUE para poder obtener dichos valores; su formato especifica que se tiene que referenciar la razón exacta del valor a capturar. El valor especificado en los corchetes indica a qué elemento apunta la captura del valor; por ejemplo, el nodo cliente solo tiene dos índices (P0001 y P0002). Finalmente, se muestran los valores capturados mediante la sentencia SELECT.

La imagen siguiente muestra el resultado de ejecutar el script anterior:

```
CODIGO RAZON SOCIAL
------ -----------------
P0001  mtorres SRLTDA
(1 row(s) affected)
```

- Script que permita mostrar el registro según el código de un cliente ingresado por el usuario desde la tabla CLIENTEXML. Se debe usar la función EXIST del tipo de datos XML.

CLIENTEXML

Column Name	Condensed Type
CODIGO	int
RAZON_SOCIAL	xml

```
DECLARE @N INT=1
SELECT X.CODIGO,X.RAZON_SOCIAL
        FROM CLIENTEXML X
        WHERE X.RAZON_SOCIAL.exist('/clientes/cliente/rsocial[2]')=@N
GO
```

El resultado de la ejecución se muestra en la siguiente imagen:

```
CODIGO      RAZON_SOCIAL
----------- ------------------------------------------------------------------------------
1           <clientes><cliente><codigo>P0001</codigo><rsocial>mtorres peru SAC</rsocial>
            <rsocial>mtorres SRLTDA</rsocial></cliente><cliente><codigo>P0002</codigo>
            <rsocial>llazaro SAC </rsocial><rsocial>llazaro SRTLDA</rsocial></cliente></clientes>

(1 row(s) affected)
```

- Script que permita insertar los registros de los clientes almacenados en un archivo XML y volcarlo a la tabla CLIENTESXML. Se debe usar la sentencia BULK.

CLIENTEXML

Column Name	Condensed Type
CODIGO	int
RAZON_SOCIAL	xml

Para lograr insertar registros desde un XML a una tabla se necesita tener un archivo XML preparado; esto se puede realizar en el Bloc de Notas de Windows, colocando el siguiente script dentro de cualquier editor.

cliente.xml: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
<clientes>
        <cliente>
                <codigo>CL001</codigo>
                <rsocial>LIMA-TEEN SAC</rsocial>
                <rsocial>LIMA-TEEN SRLTDA</rsocial>
        </cliente>
        <cliente>
                <codigo>CL002</codigo>
                <rsocial>PERU TEEN SAC </rsocial>
                <rsocial>PERU TEEN SRTLDA</rsocial>
        </cliente>
</clientes>
```

Al ejecutar o arrastrarlo a una página web el resultado es el siguiente:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<clientes>
  ▼<cliente>
     <codigo>CL001</codigo>
     <rsocial>LIMA-TEEN SAC</rsocial>
     <rsocial>LIMA-TEEN SRLTDA</rsocial>
   </cliente>
  ▼<cliente>
     <codigo>CL002</codigo>
     <rsocial>PERU TEEN SAC</rsocial>
     <rsocial>PERU TEEN SRTLDA</rsocial>
   </cliente>
 </clientes>
```

El script que da solución al problema es como sigue:

```
INSERT INTO CLIENTEXML
        SELECT RAZON_SOCIAL
                FROM (SELECT *
                        FROM OPENROWSET(BULK 'C:\cliente.xml',
                                        SINGLE_CLOB) AS CLIENTEXML)
                AS P(RAZON_SOCIAL)
GO
```

También se debe visualizar el contenido de la tabla CLIENTEXML y comprobar si se completó el registro desde el archivo **clientes.xml**.

	CODIGO	RAZON_SOCIAL
1	1	<clientes><cliente><codigo>P0001</codigo><rsocia...
2	2	<clientes><cliente><codigo>CL001</codigo><rsocia...

La siguiente imagen muestra lo que SQL Server interpreta de la columna XML:

```
1  <clientes>
2    <cliente>
3      <codigo>CL001</codigo>
4      <rsocial>LIMA-TEEN SAC</rsocial>
5      <rsocial>LIMA-TEEN SRLTDA</rsocial>
6    </cliente>
7    <cliente>
8      <codigo>CL002</codigo>
9      <rsocial>PERU TEEN SAC </rsocial>
10     <rsocial>PERU TEEN SRTLDA</rsocial>
11   </cliente>
12 </clientes>
```

10.7 FOR XML Y OPENXML

Es posible ejecutar consultas SQL para obtener resultados en formato XML en lugar de conjuntos de filas estándar. Estas consultas pueden ejecutarse directamente o desde procedimientos almacenados y funciones definidas por el usuario. Para obtener los resultados directamente, se utiliza primero la cláusula FOR XML de la instrucción SELECT. A continuación, dentro de la cláusula FOR XML, se especifica un modo de XML: RAW, AUTO, EXPLICIT o PATH.

Formato:

```
[ FOR BROWSE | <XML> ]
<XML> ::= XML
    {
      { RAW [ ('ELEMENTNAME') ] | AUTO }
        [
           <COMMONDIRECTIVES>
           [ , XMLDATA | XMLSCHEMA [ ('TARGETNAMESPACEURI') ] ]
           [ , ELEMENTS [ XSINIL | ABSENT ]
        ]
      | EXPLICIT
        [
           <COMMONDIRECTIVES>
           [ , XMLDATA ]
        ]
      | PATH [ ('ELEMENTNAME') ]
        [
           <COMMONDIRECTIVES>
           [ , ELEMENTS [ XSINIL | ABSENT ] ]
      ]
```

RAW [('ElementName')]: Obtiene el resultado de la consulta y transforma cada fila del conjunto de resultados en un elemento XML con un identificador genérico <row /> como etiqueta del elemento. Opcionalmente, puede especificar un nombre para el elemento de fila cuando se utiliza esta directiva. El XML resultante utilizará el ElementName especificado como el elemento de fila generado para cada fila.

AUTO: Devuelve los resultados de la consulta en un árbol anidado XML sencillo. Cada tabla en la cláusula FROM, de la que al menos se presenta una columna en la cláusula SELECT, se representa como un elemento XML. A las columnas presentadas en la cláusula SELECT se les asignan los atributos de elementos apropiados.

EXPLICIT: Especifica que la forma del árbol XML resultante está definida explícitamente. Con este modo, es necesario escribir las consultas de una cierta manera, de modo que se pueda especificar explícitamente información adicional acerca de la anidación deseada.

PATH: Facilita la mezcla de elementos y atributos, así como la especificación de anidación adicional para representar propiedades complejas. Puede utilizar consultas FOR XML de modo EXPLICIT para construir XML a partir de un conjunto de filas; pero el modo PATH supone una alternativa más simple a las consultas de modo EXPLICIT potencialmente complicadas. El modo PATH, junto con la posibilidad de escribir consultas FOR XML anidadas y la directiva TYPE para devolver instancias de tipo XML, permite escribir consultas de forma más fácil. Ofrece una alternativa para

escribir la mayoría de las consultas de modo EXPLICIT. De forma predeterminada, el modo PATH genera un contenedor de elementos <row> para cada fila del conjunto de resultados. También se puede especificar un nombre de elemento. En este caso, el nombre especificado se utilizará como nombre del elemento contenedor. Si se proporciona una cadena vacía (FOR XML PATH ('')), no se generará ningún elemento contenedor.

XMLDATA: Especifica que se debe devolver un esquema XDR (XML-Data Reduced) insertado. El esquema se antepone al documento como un esquema insertado.

XMLSCHEMA: Devuelve un esquema XML W3C (XSD) insertado. Opcionalmente, puede especificar un URI de espacio de nombres de destino al especificar esta directiva. De este modo, se devuelve el espacio de nombres especificado en el esquema. Para obtener más información, vea Generación de esquemas XSD insertados.

ELEMENTS: Si se especifica la opción ELEMENTS, las columnas se devuelven como subelementos. Sin embargo, se les asignan atributos XML. Esta opción solo se admite en los modos RAW, AUTO y PATH. También puede especificar XSINIL o ABSENT cuando utilice esta directiva. XSINIL especifica que se puede crear un elemento con un atributo xsi:nil establecido en True para los valores de columna NULL. De forma predeterminada, o cuando se especifica ABSENT junto con ELEMENTS, no se crea ningún elemento para los valores NULL.

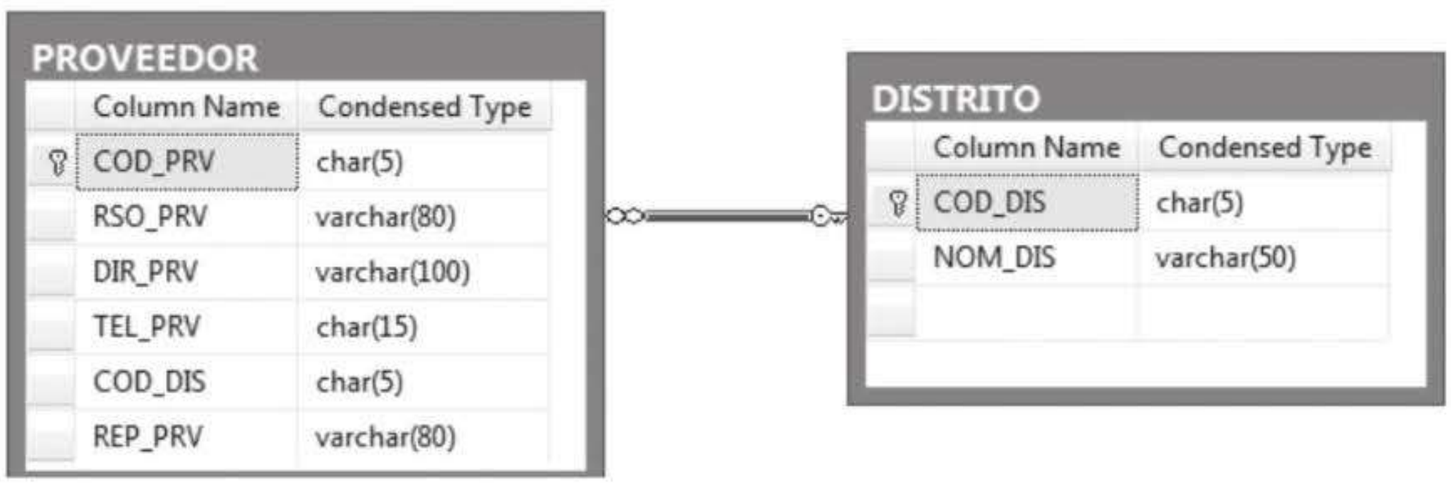
BINARY BASE64: Si se especifica la opción BINARY Base64, todos los datos binarios que devuelve la consulta se representan en formato codificado base64. Para recuperar datos binarios mediante el modo RAW y EXPLICIT, se debe especificar esta opción. En modo AUTO, de forma predeterminada, se devuelven datos binarios como una referencia.

TYPE: Especifica que la consulta devuelve los resultados como el tipo XML.

ROOT [('RootName')]: Especifica que se puede agregar un solo elemento de nivel superior al XML resultante. También se puede especificar el nombre del elemento raíz que se generará. El valor predeterminado es "root".

A continuación, algunos casos del uso de FOR XML:

- Script que permita mostrar los registros de la tabla PROVEEDOR en un elemento XML genérico. Se usará la cláusula FOR XML en modo RAW.



```
SELECT P.COD_PRV AS CODIGO, P.RSO_PRV AS PROVEEDOR,
       P.DIR_PRV AS DIRECCION,
       P.TEL_PRV AS TELEFONO,
       D.NOM_DIS AS DISTRITO,
       P.REP_PRV AS REPRESENTANTE
       FROM PROVEEDOR P
       JOIN DISTRITO D ON P.COD_DIS=D.COD_DIS
       FOR XML RAW
GO
```

El resultado de la ejecución de la consulta se muestra en la siguiente imagen:

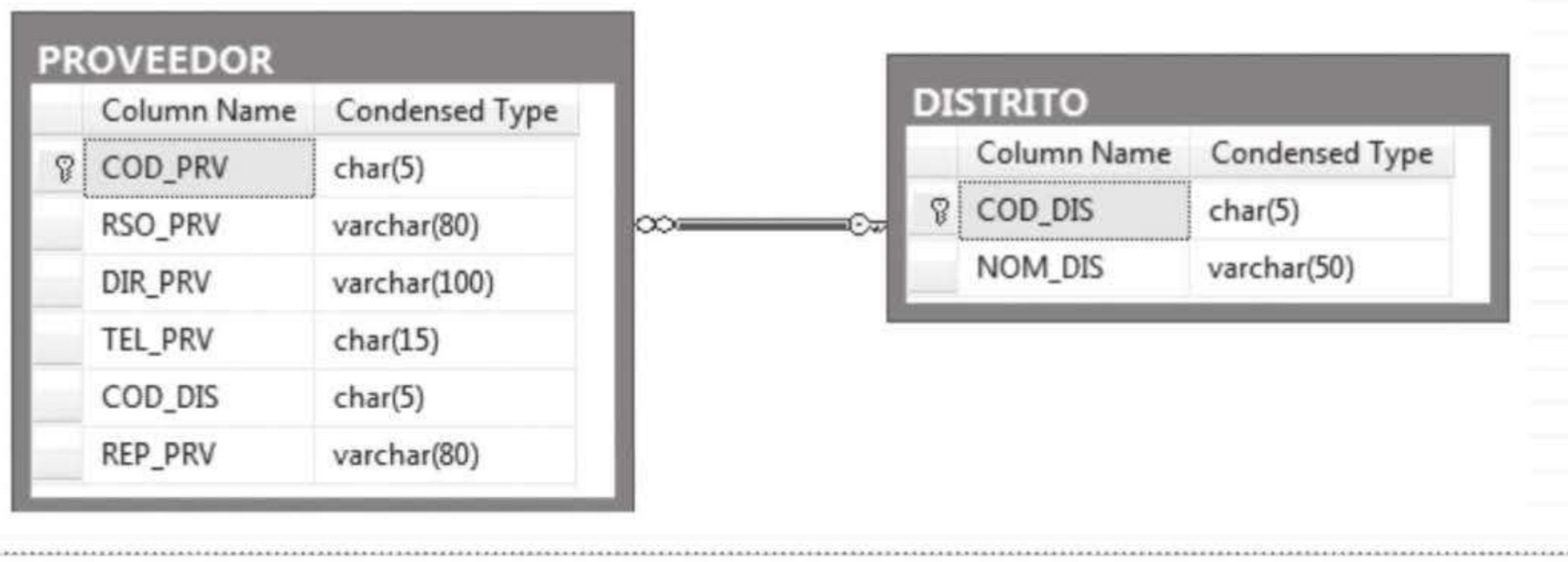
	XML_F52E2B61-18A1-11d1-B105-00805F49916B
1	<row CODIGO="PR01 " PROVEEDOR="FABER CASTELL" DIR...

La imagen siguiente muestra el *linkeo* hacia el registro mostrado en la consulta.

```
<row CODIGO="PR01 " PROVEEDOR="FABER CASTELL" DIRECCION="AV. ISABEL LA CATOLICA 1875" TELEFONO="0000000       " DISTRITO="RIMAC" REPRESENTANTE="CARLOS AGUIRRE" />
<row CODIGO="PR02 " PROVEEDOR="ATLAS" DIRECCION="AV. LIMA 471" TELEFONO="0000000       " DISTRITO="RIMAC" REPRESENTANTE="CESAR TORRES" />
<row CODIGO="PR03 " PROVEEDOR="3M" DIRECCION="AV. VENEZUELA 3018" TELEFONO="2908165       " DISTRITO="BELLAVISTA" REPRESENTANTE="OMAR INJOQUE" />
<row CODIGO="PR04 " PROVEEDOR="DITO" DIRECCION="AV. METROPOLITANA 376" DISTRITO="SANTA ANITA" REPRESENTANTE="RAMON FLORES" />
<row CODIGO="PR05 " PROVEEDOR="ACKER" DIRECCION="CALLE LAS DUNAS 245" TELEFONO="4780143       " DISTRITO="ATE - VITARTE" REPRESENTANTE="JULIO ACUÑA" />
<row CODIGO="PR06 " PROVEEDOR="DEDITEC" DIRECCION="CALLE PICHINCHA 644" TELEFONO="5662848       " DISTRITO="BREÑA" REPRESENTANTE="JAVIER GONZALES" />
<row CODIGO="PR07 " PROVEEDOR="OFFICETEC" DIRECCION="CALLE LAS PERDICES 225 OF. 204" TELEFONO="4216184       " DISTRITO="SAN ISIDRO" REPRESENTANTE="CARLOS ROBLES" />
<row CODIGO="PR08 " PROVEEDOR="INVICTA" DIRECCION="AV. LOS FRUTALES 564" TELEFONO="4364247       " DISTRITO="ATE - VITARTE" REPRESENTANTE="ALBERTO RODRIGUEZ" />
<row CODIGO="PR09 " PROVEEDOR="DIPROPOR" DIRECCION="AV. DEL AIRE 901" TELEFONO="4742046       " DISTRITO="SAN LUIS" REPRESENTANTE="ROBERTO RONCEROS" />
<row CODIGO="PR10 " PROVEEDOR="MIURA" DIRECCION="AV. LA PAZ 257" TELEFONO="4459710       " DISTRITO="MIRAFLORES" REPRESENTANTE="JORGE VASQUEZ" />
<row CODIGO="PR11 " PROVEEDOR="PRAXIS" DIRECCION="AV. JOSE GALVEZ 1820 - 1844" TELEFONO="4703944       " DISTRITO="LINCE" REPRESENTANTE="ERICKA ZEGARRA" />
<row CODIGO="PR12 " PROVEEDOR="SUMIGRAF" DIRECCION="AV. MANCO CAPAC 754" TELEFONO="0000000       " DISTRITO="RIMAC" REPRESENTANTE="KARINNA PAREDES" />
<row CODIGO="PR13 " PROVEEDOR="LIMMSA" DIRECCION="PROLG. HUAYLAS 670" TELEFONO="2546995       " DISTRITO="CHORRILLOS" REPRESENTANTE="LAURA ORTEGA" />
<row CODIGO="PR14 " PROVEEDOR="VENINSA" DIRECCION="AV. TEJADA 338" TELEFONO="2473832       " DISTRITO="BARRANCO" REPRESENTANTE="MELISA RAMOS" />
<row CODIGO="PR15 " PROVEEDOR="CROSLAND" DIRECCION="AV. ARGENTINA 3206 - 3250" TELEFONO="4515272       " DISTRITO="CALLAO" REPRESENTANTE="JUAN RAMIREZ" />
<row CODIGO="PR16 " PROVEEDOR="PETRAMAS" DIRECCION="CALLE JOAQUIN MADRID 141 2DO P" DISTRITO="SAN BORJA" REPRESENTANTE="ROCIO GUERRERO" />
<row CODIGO="PR17 " PROVEEDOR="REAWSE" DIRECCION="AV. SANTA ROSA 480 " DISTRITO="SANTA ANITA" REPRESENTANTE="MARIA PEREZ" />
<row CODIGO="PR18 " PROVEEDOR="EDUSA" DIRECCION="AV. MORALES DUAREZ 1260" TELEFONO="4525536       " DISTRITO="CARMEN DE LA LEGUA" REPRESENTANTE="PABLO MARTEL" />
<row CODIGO="PR19 " PROVEEDOR="OTTMER" DIRECCION="URB.PRO INDUSTRIAL MZ B6 LT 16" TELEFONO="5369893       " DISTRITO="SAN MARTIN DE PORRES" REPRESENTANTE="ANGELA RENDILLA" />
<row CODIGO="PR20 " PROVEEDOR="BARI" DIRECCION="AV. ARNALDO MARQUEZ 1219" DISTRITO="JESUS MARIA" REPRESENTANTE="VANESA QUINTANA" />
<row CODIGO="PR21 " PROVEEDOR="BARI" DIRECCION="AV. ARNALDO MARQUEZ 1219" DISTRITO="JESUS MARIA" REPRESENTANTE="VANESA QUINTANA" />
```

Como se notará, en la imagen se crea un elemento XML genérico llamado row por cada fila obtenida desde la tabla PROVEEDOR.

- Script que permita mostrar los registros de la tabla PROVEEDOR en un elemento XML genérico. Se usará la cláusula FOR XML en modo ELEMENTS.



```
SELECT P.COD_PRV AS CODIGO, P.RSO_PRV AS PROVEEDOR,
       P.DIR_PRV AS DIRECCION,
       P.TEL_PRV AS TELEFONO,
       D.NOM_DIS AS DISTRITO,
       P.REP_PRV AS REPRESENTANTE
       FROM PROVEEDOR P
       JOIN DISTRITO D ON P.COD_DIS=D.COD_DIS
       FOR XML RAW,ELEMENTS
GO
```

El resultado de la ejecución de la consulta se muestra en la siguiente imagen:

	XML_F52E2B61-18A1-11d1-B105-00805F49916B
1	<row CODIGO="PR01 " PROVEEDOR="FABER CASTELL" DIR...

La imagen siguiente muestra el *linkeo* hacia el registro mostrado en la consulta.

```
<row>
  <CODIGO>PR01 </CODIGO>
  <PROVEEDOR>FABER CASTELL</PROVEEDOR>
  <DIRECCION>AV. ISABEL LA CATOLICA 1875</DIRECCION>
  <TELEFONO>0000000        </TELEFONO>
  <DISTRITO>RIMAC</DISTRITO>
  <REPRESENTANTE>CARLOS AGUIRRE</REPRESENTANTE>
</row>
<row>
  <CODIGO>PR02 </CODIGO>
  <PROVEEDOR>ATLAS</PROVEEDOR>
  <DIRECCION>AV. LIMA 471</DIRECCION>
  <TELEFONO>0000000        </TELEFONO>
  <DISTRITO>RIMAC</DISTRITO>
  <REPRESENTANTE>CESAR TORRES</REPRESENTANTE>
</row>
<row>
  <CODIGO>PR03 </CODIGO>
  <PROVEEDOR>3M</PROVEEDOR>
  <DIRECCION>AV. VENEZUELA 3018</DIRECCION>
  <TELEFONO>2908165        </TELEFONO>
  <DISTRITO>BELLAVISTA</DISTRITO>
  <REPRESENTANTE>OMAR INJOQUE</REPRESENTANTE>
</row>
<row>
  <CODIGO>PR04 </CODIGO>
  <PROVEEDOR>DITO</PROVEEDOR>
  <DIRECCION>AV. METROPOLITANA 376</DIRECCION>
  <DISTRITO>SANTA ANITA</DISTRITO>
  <REPRESENTANTE>RAMON FLORES</REPRESENTANTE>
</row>
```

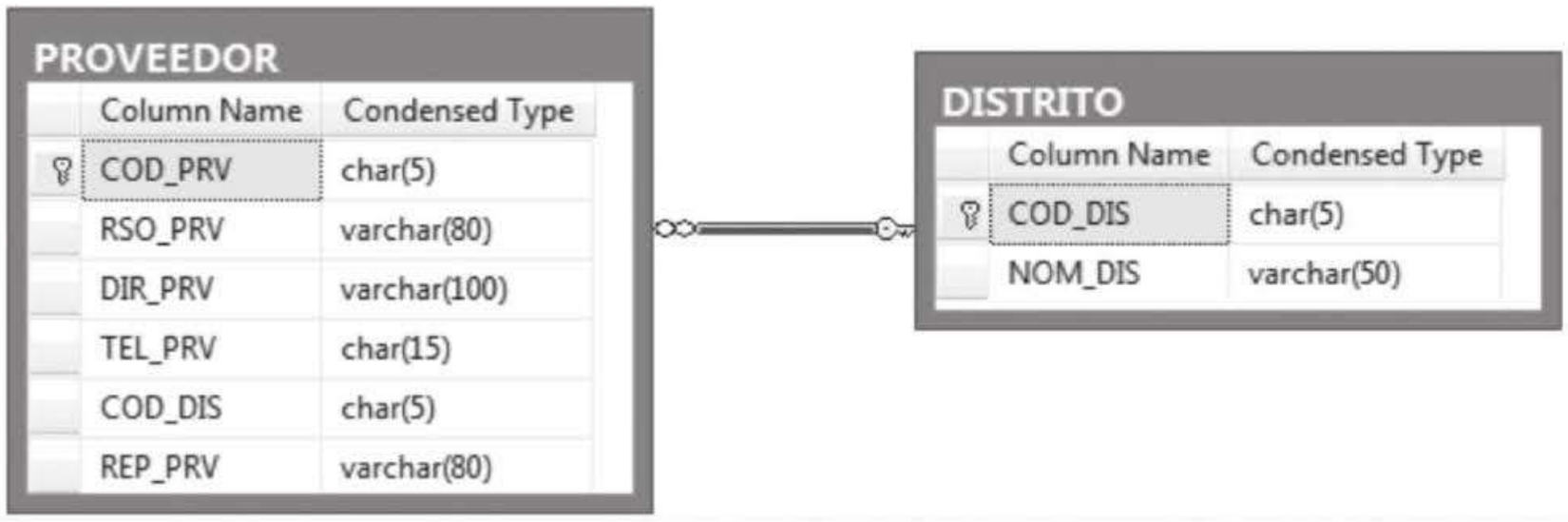
Como se notará, la generación del archivo XML se promueve con el nodo row y los elementos código, proveedor, dirección, teléfono, distrito; y se representa por cada cambio de fila. Otro punto que hay que recalcar es que sus subelementos formados en el archivo XML normalmente deben ser en minúsculas, pero como en la tabla se definió las columnas en mayúsculas también generó en mayúsculas parte del esquema XML. Para poder cambiar a minúsculas, se deberá ejecutar el siguiente script:

```
SELECT P.COD_PRV AS codigo, P.RSO_PRV AS proveedor,
       P.DIR_PRV AS direccion,
       P.TEL_PRV AS telefono,
       D.NOM_DIS AS distrito,
       P.REP_PRV AS representante
       FROM PROVEEDOR P
       JOIN DISTRITO D ON P.COD_DIS=D.COD_DIS
       FOR XML RAW,ELEMENTS
GO
```

Si se observa, el XML generado se muestra con los subelementos en minúsculas:

```
<row>
  <codigo>PR01 </codigo>
  <proveedor>FABER CASTELL</proveedor>
  <direccion>AV. ISABEL LA CATOLICA 1875</direccion>
  <telefono>0000000       </telefono>
  <distrito>RIMAC</distrito>
  <representante>CARLOS AGUIRRE</representante>
</row>
<row>
  <codigo>PR02 </codigo>
  <proveedor>ATLAS</proveedor>
  <direccion>AV. LIMA 471</direccion>
  <telefono>0000000       </telefono>
  <distrito>RIMAC</distrito>
  <representante>CESAR TORRES</representante>
</row>
<row>
  <codigo>PR03 </codigo>
  <proveedor>3M</proveedor>
  <direccion>AV. VENEZUELA 3018</direccion>
  <telefono>2908165       </telefono>
  <distrito>BELLAVISTA</distrito>
  <representante>OMAR INJOQUE</representante>
</row>
<row>
  <codigo>PR04 </codigo>
  <proveedor>DITO</proveedor>
  <direccion>AV. METROPOLITANA 376</direccion>
  <distrito>SANTA ANITA</distrito>
  <representante>RAMON FLORES</representante>
</row>
<row>
  <codigo>PR05 </codigo>
  <proveedor>ACKER</proveedor>
  <direccion>CALLE LAS DUNAS 245</direccion>
  <telefono>4780143       </telefono>
  <distrito>ATE - VITARTE</distrito>
  <representante>JULIO ACUÑA</representante>
</row>
```

- Script que permita insertar registros desde un archivo XML hacia la tabla PROVEEDOR.



Inicialmente, se necesita un archivo XML que contenga la siguiente información de los proveedores. Este archivo se debe encontrar en la unidad C:

proveedor.xml: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
<proveedores>
 <proveedor>
        <cod_prv>PR22</cod_prv>
        <rso_prv>MAYORSA</rso_prv>
        <dir_prv>AV. EL SOL 545</dir_prv>
        <tel_prv>586-9685</tel_prv>
        <cod_dis>D01</cod_dis>
        <rep_prv>JOSE MARTINEZ</rep_prv>
</proveedor>
<proveedor>
        <cod_prv>PR23</cod_prv>
        <rso_prv>MERCADOS PERU</rso_prv>
        <dir_prv>AV. LIMA 7856</dir_prv>
        <tel_prv>345-9354</tel_prv>
        <cod_dis>D04</cod_dis>
        <rep_prv>MARK TENERE</rep_prv>
</proveedor>
<proveedor>
        <cod_prv>PR24</cod_prv>
        <rso_prv>MOBIL SAC</rso_prv>
        <dir_prv>AV. LAS PALMERAS 545</dir_prv>
        <tel_prv>446-5445</tel_prv>
        <cod_dis>D03</cod_dis>
        <rep_prv>JUAN RYTORES</rep_prv>
</proveedor>
</proveedores>
```

En un documento web se muestra el archivo XML de la siguiente manera:



El código que permite obtener los registros desde el archivo XML y enviarlo a la tabla PROVEEDOR se muestra en el siguiente script:

```
INSERT INTO PROVEEDOR(COD_PRV,RSO_PRV,DIR_PRV,TEL_PRV,COD_DIS,REP_PRV)
       SELECT
              P.proveedores.query('cod_prv').value('.','CHAR(4)'),
              P.proveedores.query('rso_prv').value('.','VARCHAR(40)'),
              P.proveedores.query('dir_prv').value('.','VARCHAR(30)'),
              P.proveedores.query('tel_prv').value('.','VARCHAR(15)'),
              P.proveedores.query('cod_dis').value('.','CHAR(3)'),
              P.proveedores.query('rep_prv').value('.','VARCHAR(50)')
            FROM (SELECT CAST(P AS XML)
                        FROM OPENROWSET(BULK 'C:\proveedor.xml',
                                        SINGLE_BLOB) AS PRO(P)
                  ) AS PRO(P)
        CROSS APPLY P.nodes('proveedores/proveedor') AS P(proveedores);
GO
```

Se debe tener mucho cuidado con la sensibilidad de las funciones mostradas en el script, puesto que son estrictas con respecto a las minúsculas y mayúsculas. Es recomendable que lo digiten todo en minúsculas.

Luego de insertar los tres registros desde el archivo proveedor.xml, el Motor de base de datos envía el siguiente mensaje:

```
(3 row(s) affected)
(3 row(s) affected)
```

Si se revisa el contenido de los registros de la tabla PROVEEDOR, se observará que, efectivamente, los registros se han volcado desde el archivo XML hacia la tabla PROVEEDOR.

22	PR22	MAYORSA	AV. EL SOL 545	586-9685	D01	JOSE MARTINEZ
23	PR23	MERCADOS P...	AV. LIMA 7856	345-9354	D04	MARK TENERE
24	PR24	MOBIL SAC	AV. LAS PALMERAS 545	446-5445	D03	JUAN RYTORES

- Script que permita buscar un determinado proveedor mediante su CÓDIGO dentro de un archivo XML.

```
DECLARE @PXML XML
SET @PXML = '<proveedores>
                        <proveedor cod_prv=”PR22”>
                              <rso_prv>MAYORSA</rso_prv>
                              <dir_prv>AV. EL SOL 545</dir_prv>
                              <tel_prv>586-9685</tel_prv>
                              <cod_dis>D01</cod_dis>
                              <rep_prv>JOSE MARTINEZ</rep_prv>
                        </proveedor>
                        <proveedor cod_prv=”PR23”>
                              <rso_prv>MERCADOS PERU</rso_prv>
                              <dir_prv>AV. LIMA 7856</dir_prv>
                              <tel_prv>345-9354</tel_prv>
                              <cod_dis>D04</cod_dis>
                              <rep_prv>MARK TENERE</rep_prv>
                        </proveedor>
                        <proveedor cod_prv=”PR24”>
                              <rso_prv>MOBIL SAC</rso_prv>
                              <dir_prv>AV. LAS PALMERAS 545</dir_prv>
                              <tel_prv>446-5445</tel_prv>
                              <cod_dis>D03</cod_dis>
                              <rep_prv>JUAN RYTORES</rep_prv>
                        </proveedor>
                        </proveedores>'
SELECT @PXML.query('/proveedores/proveedor[./@cod_prv=”PR23”]')
GO
```

Si se presiona sobre el enlace emitido, se tendrá la siguiente imagen:

```
1 <proveedor cod_prv="PR23">
2   <rso_prv>MERCADOS PERU</rso_prv>
3   <dir_prv>AV. LIMA 7856</dir_prv>
4   <tel_prv>345-9354</tel_prv>
5   <cod_dis>D04</cod_dis>
6   <rep_prv>MARK TENERE</rep_prv>
7 </proveedor>
```

10.8 Instrucción OpenRowSet

Contiene toda la información de conexión necesaria para tener acceso a datos remotos desde un origen de datos OLE DB. Es un método alternativo para tener acceso a las tablas de un servidor vinculado y, al mismo tiempo, es un método *ad hoc* para conectarse y tener acceso a datos remotos utilizando OLE DB.

Características:

- Se puede hacer referencia a la función OPENROWSET en la cláusula FROM de una consulta, como si fuera el nombre de una tabla.
- También se puede hacer referencia a la función OPENROWSET como tabla de destino de una instrucción INSERT, UPDATE o DELETE, sujeta a la funcionalidad del proveedor OLE DB.
- Aunque la consulta puede devolver varios conjuntos de resultados, OPENROWSET solo devuelve el primero.
- OPENROWSET también admite operaciones masivas a través de un proveedor integrado BULK que permite que los datos se lean y se devuelvan como un conjunto de filas.

Formato:

```
OPENROWSET
(
  BULK 'ARCHIVO_DATOS' ,  SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB
)
```

Donde:

- **BULK:** Utiliza el proveedor de conjuntos de filas BULK para que OPENROWSET lea datos de un archivo. En SQL Server, OPENROWSET puede leer datos de un archivo sin necesidad de cargarlos en una tabla de destino. Esto permite utilizar OPENROWSET con una instrucción SELECT simple.

  Los argumentos de la opción BULK permiten elegir dónde empezar y acabar la lectura de datos, cómo abordar los errores y cómo interpretar los datos. Por ejemplo, se puede especificar que el archivo de datos se lea como un conjunto de filas de una sola fila y una sola columna de tipo Varbinary, Varchar o Nvarchar.
- **SINGLE_BLOB**: Devuelve el contenido de ARCHIVO_DATOS como un conjunto de filas de tipo Varbinary(max) de una sola fila y una sola columna.
- **SINGLE_CLOB**: Al leer ARCHIVO_DATOS como ASCII, se devuelve el contenido como un conjunto de filas de tipo Varchar(max) de una sola fila y una sola columna, mediante la intercalación de la base de datos actual.
- **SINGLE_NCLOB:** Al leer data_file como UNICODE, se devuelve el contenido como un conjunto de filas de tipo Nvarchar(max) de una sola fila y una sola columna, mediante la intercalación de la base de datos actual.

A continuación, algunos casos del uso de OPENROWSET:

- Script que permita volcar el contenido de un archivo de tipo XML en una sola fila de SQL Server.

```
      --1.
DECLARE @RUTA VARCHAR(100) = 'C:\PASAJEROS.XML'
DECLARE @SQL VARCHAR(500)

--2.
SET @SQL = 'SELECT BULKCOLUMN
                  FROM OPENROWSET(BULK ''' + @RUTA + ''', SINGLE_CLOB)
AS DATOS_XML'
EXEC(@SQL)
GO
```

En el punto uno se declaran las variables @RUTA que tienen por misión asignar la ruta del archivo fuente a volcar en este su archivo PASAJERO.XML; se encuentra en la unidad C:\. También se declara la variable @SQL que almacenará el volcado, convirtiéndose en la variable destino; por ese motivo, su declaración siempre será mínimamente VARCHAR o NVARCHAR de una capacidad extrema en el caso se prediga un máximo de 500 caracteres, pero podría ser MAX.

En el punto dos se asigna la consulta BULKCOLUMN a la variable @SQL proveniente de la instrucción OPENROWSET, especificando en este caso el volcado BULK, la ruta del archivo fuente y el tipo de valor devuelto; en este último se tiene que usar SINGLE_CLOB, ya que tiene que convertir los datos ASCII a texto legible.

Finalmente, se ejecuta la consulta por medio de la función EXEC, que tiene por misión ejecutar una instrucción preparada en Transact SQL.

El resultado de la ejecución se muestra en la siguiente imagen:

	BulkColumn
1	<pasajeros> <pasajero> <idpasajero>P0004</idpasajero> <email>ftorres@hotmail.com</email> </pasajero> <p...

Como se notará, se volcó todo el contenido del archivo externo PASAJERO.XML en una sola fila. A continuación, se muestra una segunda versión del mismo caso:

```
DECLARE @RUTA VARCHAR(100) = 'C:\PASAJEROS.XML'
DECLARE @SQL NVARCHAR(MAX)

SET @SQL = 'SELECT BULKCOLUMN
                  FROM OPENROWSET(BULK ''' + @RUTA + ''', SINGLE_CLOB)
                  AS DATOS_XML'

EXEC SP_EXECUTESQL @SQL
GO
```

El cambio realizado se produjo en la variable @SQL, se declaró como tipo NVARCHAR de un tamaño máximo permisible por el Motor de base de datos de SQL Server. Realizar este cambio repercute en la forma de ejecución del proceso; por tal motivo, se modificó la sentencia EXEC por EXEC SP_EXECUTESQL @SQL donde se hace referencia al procedimiento almacenado SP_EXECUTESQL que tiene por misión ejecutar una instrucción o lote de instrucciones Transact SQL.

- Script que permita volcar el contenido de un archivo de tipo TXT en una sola fila de SQL Server.

```
      DECLARE @RUTA VARCHAR(100) = 'C:\PASAJEROS.TXT'
DECLARE @SQL NVARCHAR(MAX)

SET @SQL = 'SELECT BULKCOLUMN
              FROM OPENROWSET(BULK ''' + @RUTA + ''', SINGLE_CLOB)
AS DATOS_TEXTO'

EXEC SP_EXECUTESQL @SQL
GO
```

Como se notará, el script es similar al mostrado anteriormente; la diferencia radica en el tipo de extensión especificada, ya que se hace referencia a un archivo de texto con extensión TXT; esto queda como precedente de que la instrucción OPENROWSET puede volcar cualquier tipo de archivo. En los ejercicios posteriores se verá cómo trabaja el OPENROWSET con el manejo de imágenes en SQL Server.

- Script que permita insertar archivos de tipo JPG dentro de la tabla FOTOPASAJERO con las columnas especificadas en la imagen.

FOTOPASAJERO

	Nombre de columna	Tipo comprimido
🔑	IDPASAJERO	char(5)
	FOTO	varbinary(MAX)

```
CREATE TABLE ALUMNOS(
CODIGO INT                    PRIMARY KEY,
FOTO          VARBINARY(MAX)        NOT NULL
)
GO

INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0001',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5001.JPG',
                       SINGLE_BLOB) AS PASAJEROS
INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0002',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5002.JPG',
                       SINGLE_BLOB) AS PASAJEROS
INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0003',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5003.JPG',
                       SINGLE_BLOB) AS PASAJEROS
```

```
INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0004',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5004.JPG',
                            SINGLE_BLOB) AS PASAJEROS
INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0005',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5005.JPG',
                            SINGLE_BLOB) AS PASAJEROS
INSERT INTO FOTOPASAJERO(IDPASAJERO,FOTO)
SELECT 'P0006',BULKCOLUMN
       FROM OPENROWSET(BULK 'C:\PASAJEROS\P5006.JPG',
                            SINGLE_BLOB) AS PASAJEROS
GO
```

Como producto de la inserción de seis registros con una columna para la fotografía, se muestra la conformidad del Motor de base de datos en la siguiente imagen:

```
(1 filas afectadas)

(1 filas afectadas)

(1 filas afectadas)

(1 filas afectadas)

(1 filas afectadas)

(1 filas afectadas)
```

La imagen siguiente muestra el resultado de consultar los registros de la tabla FOTOPASAJEROS.

	IDPASAJERO	FOTO
1	P0001	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10E0D4578696600004D4D002A0000000...
2	P0002	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10ECD4578696600004D4D002A0000000...
3	P0003	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10ECE4578696600004D4D002A0000000...
4	P0004	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10EDA4578696600004D4D002A0000000...
5	P0005	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10F3B4578696600004D4D002A0000000...
6	P0006	0xFFD8FFE000104A4649460001020201 2C012C0000FFE10D9F4578696600004D4D002A0000000...

Como se verá, no se muestra la fotografía, ya que su tipo de datos es Varbinary; pero para este caso se usará una aplicación que permitirá mostrar dichas imágenes; el *software* es SQL IMAGE VIEWER.

Para poder descargar la versión de prueba, entre a la siguiente dirección URL: <http://www.yohz.com/downloads.htm>, desde aquí deberá seleccionar **SQL Image Viewer**.



El proceso de instalación de SQL Image Viewer es el siguiente:

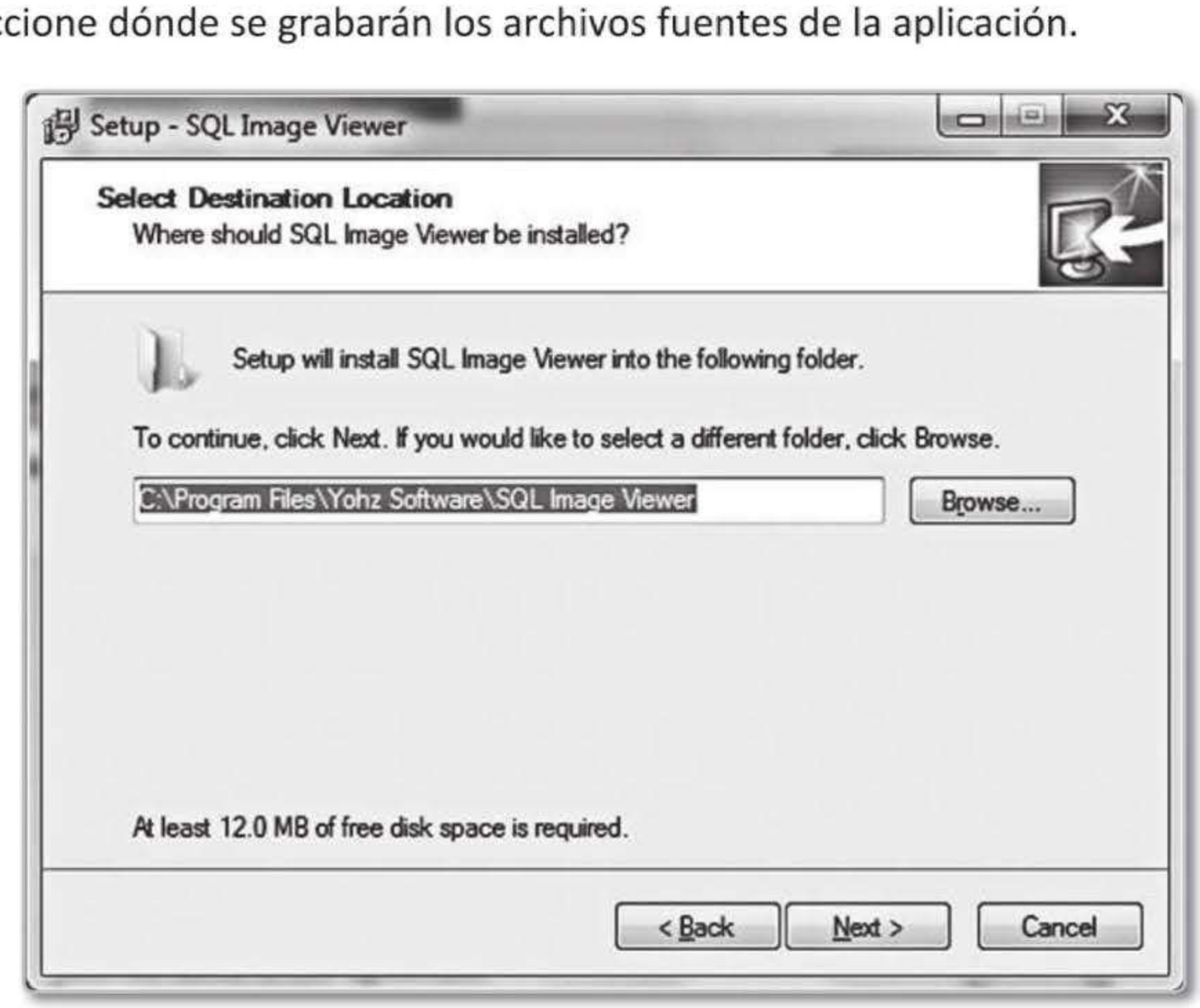
- Presione **Next** para iniciar el proceso de instalación.



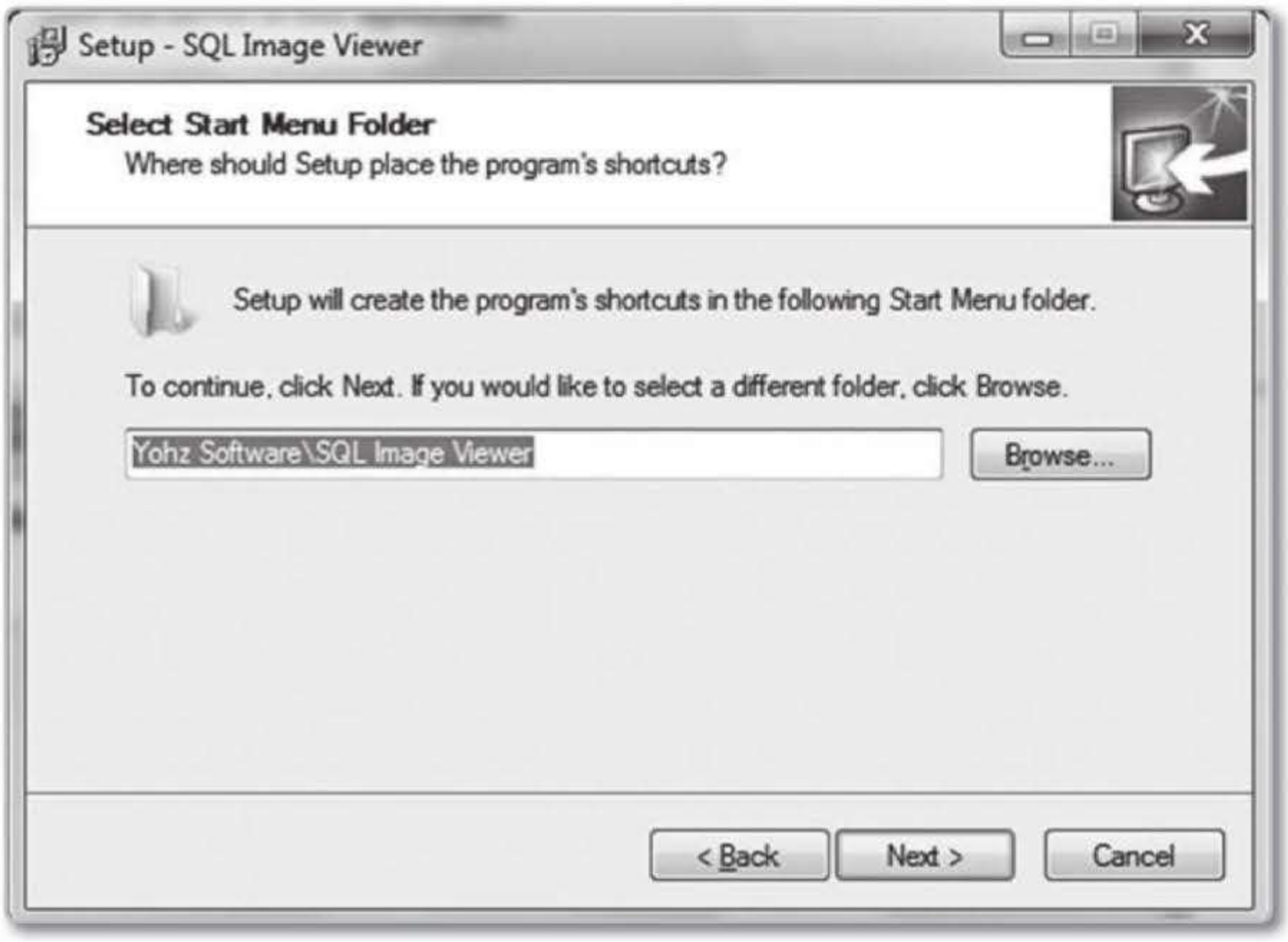
- Acepte la licencia de la aplicación, recuerde que solo es una versión de prueba.



- Direccione dónde se grabarán los archivos fuentes de la aplicación.

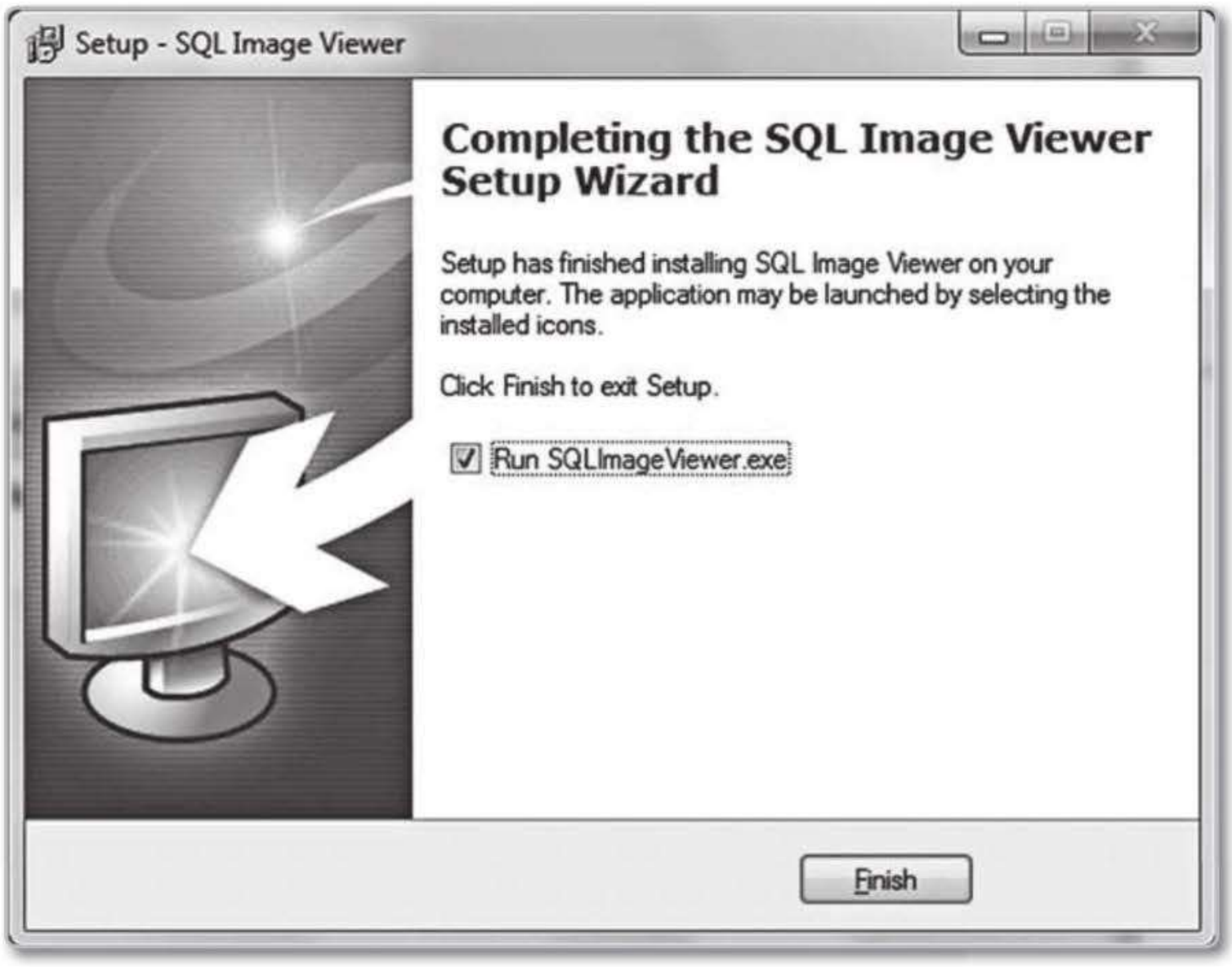


- Asigne un nombre adecuado a la aplicación, así será como Windows la reconocerá.

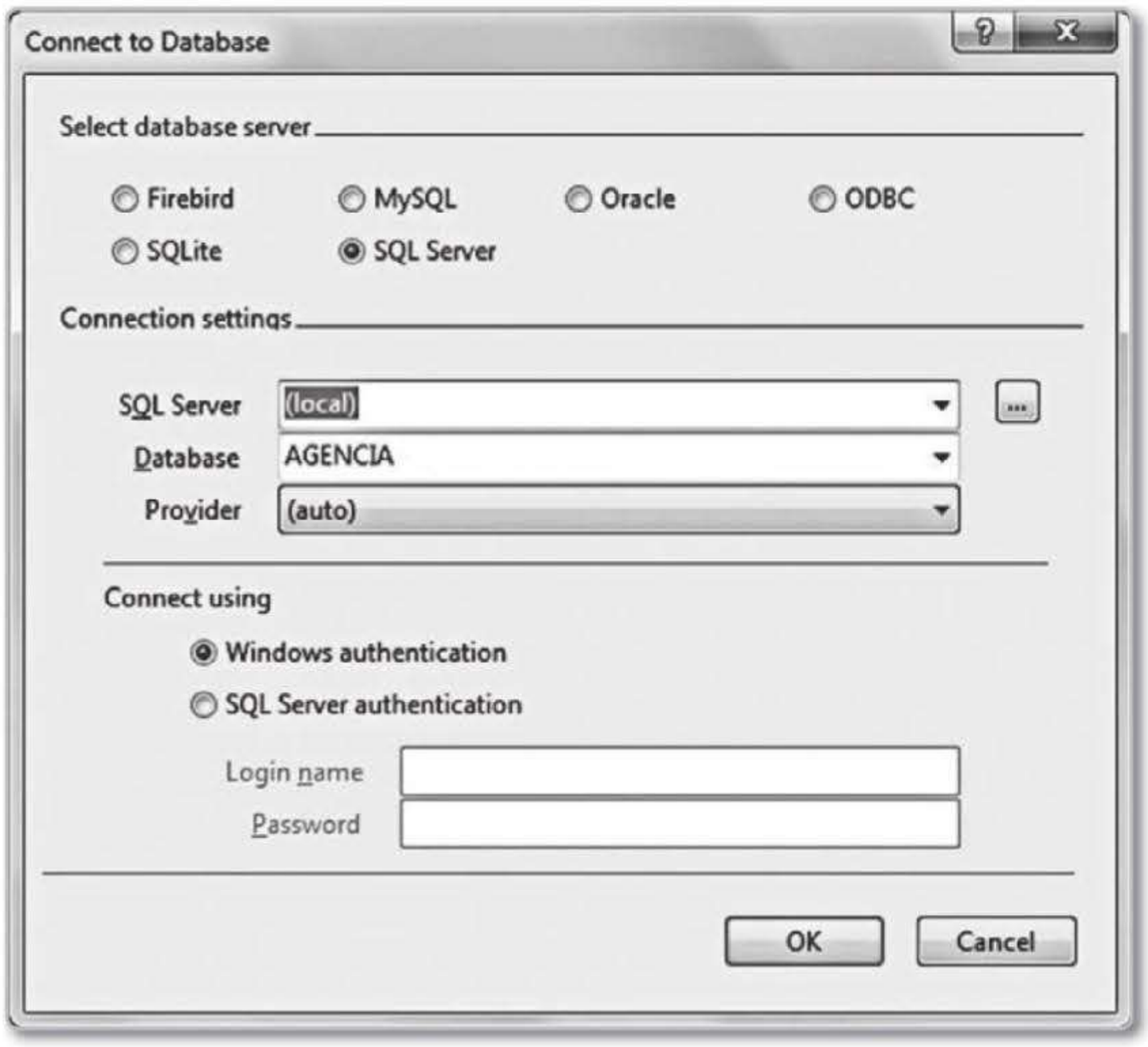


- Una vez finalizada la instalación, se procederá a ejecutar la misma con solo activar el *check* **Run SQLImageViewer.exe** y presionar **Finish**.

Como verá, esta aplicación presenta múltiples alternativas si usted maneja imágenes; en otro Motor de base de datos también podrá visualizar las imágenes por medio de esta aplicación.



- En la ventana **Connect to Database**, debe seleccionar **SQL Server**; en **SQL Server** seleccione **(local)**; en **Database**, **AGENCIA**; y no olvide usar la autenticación según la configuración inicial en SQL Server, en este caso usará el **Windows authentication**.



- Luego se mostrará la ventana principal de la aplicación que, como notará, presenta un **SQL editor** en donde usted podrá colocar el script que desea y podrá ejecutarlo como si se encontrase dentro de SQL Server. En el lado derecho de la aplicación se muestran todas las tablas que tienen la base de datos activa.

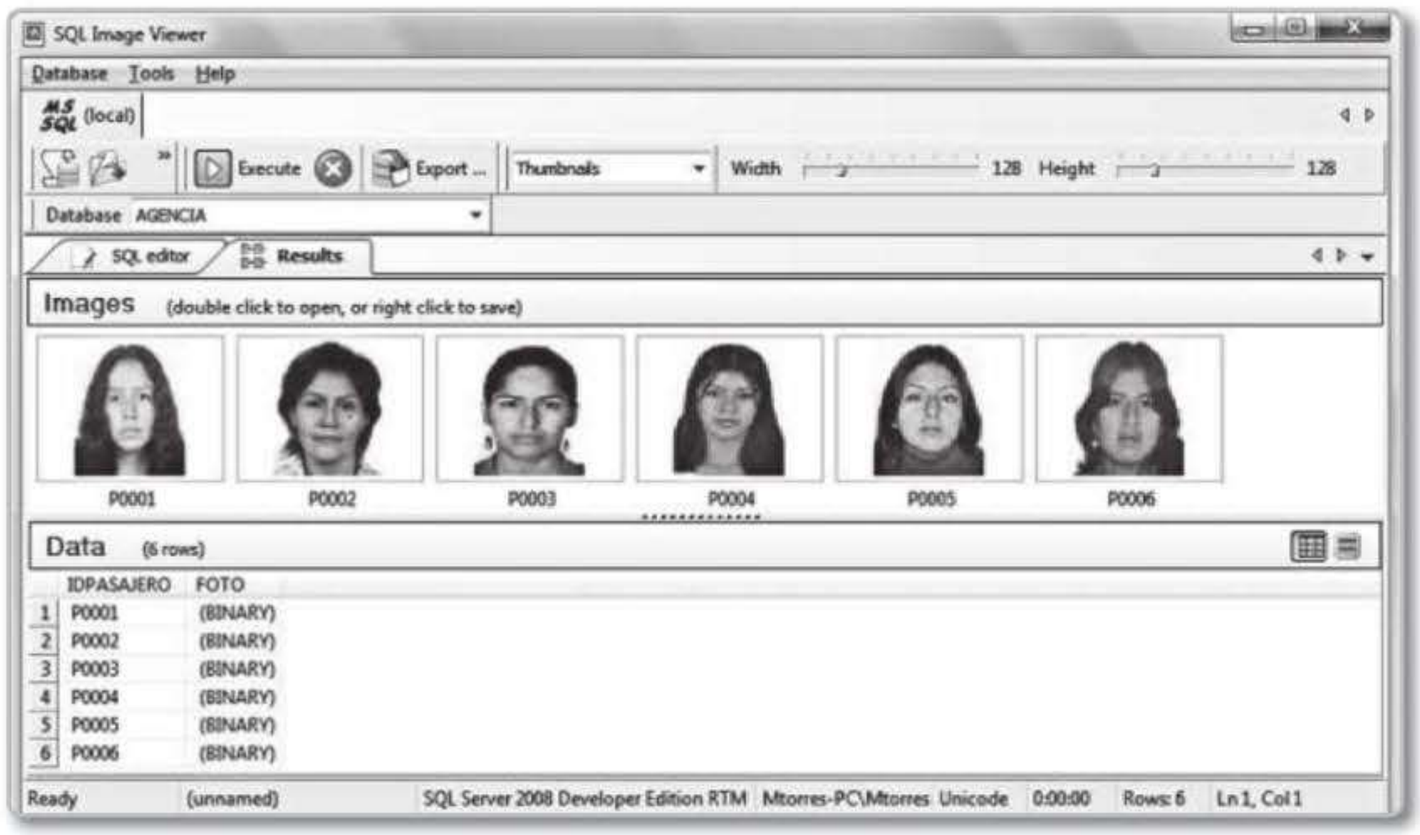


- Empezará por consultar la tabla FOTOPASAJERO que, como recordará, se implementó para almacenar fotografías de los pasajeros. En esta parte no debe añadir la sentencia GO, ya que no será reconocida como instrucción y generará un error.

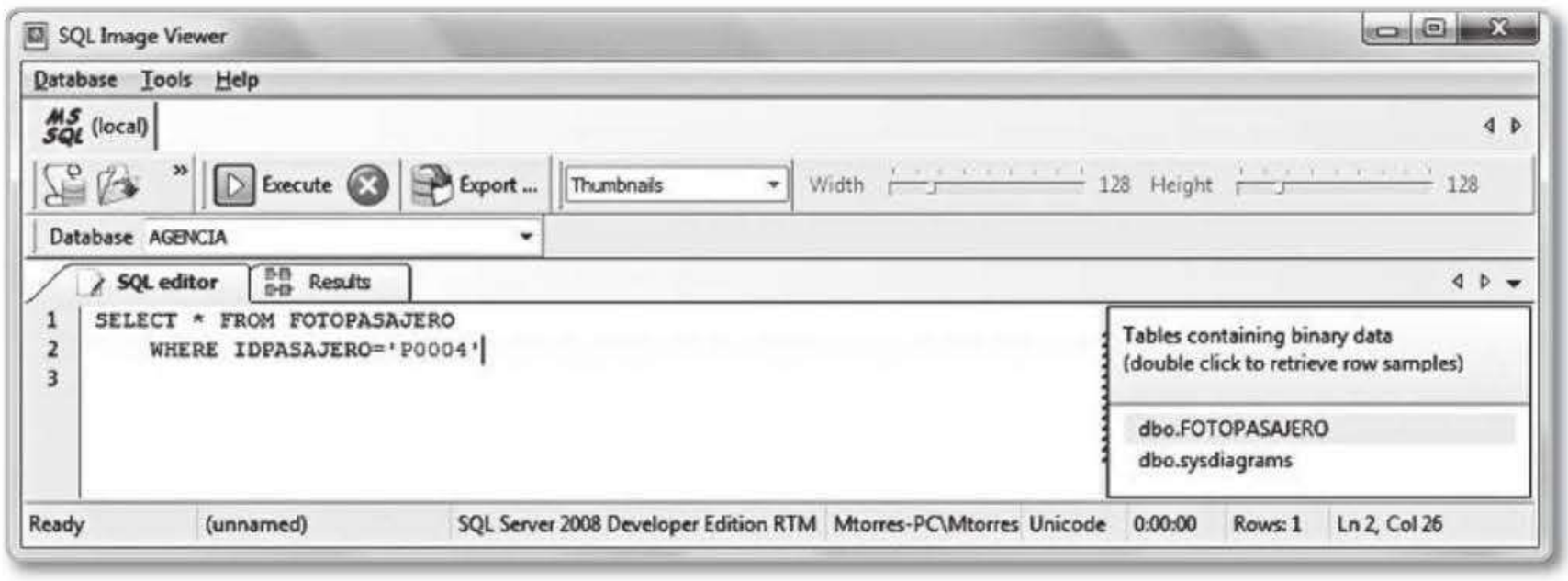
  El script se puede ejecutar seleccionándolo y presionando <F5> o, en su defecto, haciendo click sobre el botón **EXECUTE**.



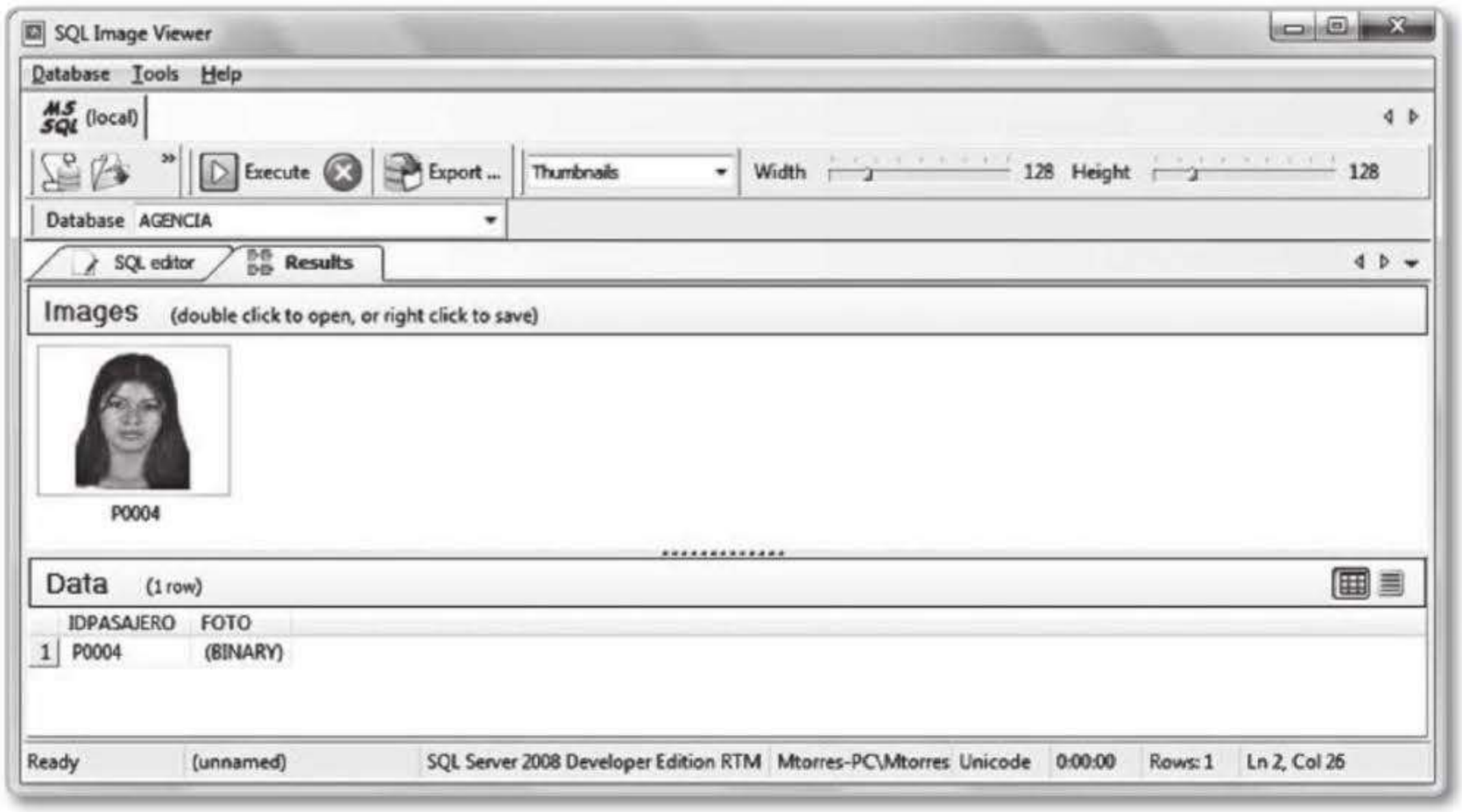
- Como verá, las imágenes de los pasajeros se podrán visualizar tal como se preparó el script de consulta; es decir, el orden de las imágenes responde al orden implementado en la sentencia SELECT.



- Así, se podrá realizar una consulta condicionada, como sucede en la siguiente imagen, donde se visualizará el pasajero cuyo código es P0004.



- La imagen siguiente muestra el resultado de la ejecución de la instrucción.



Referencias bibliográficas

- Connolly, T. M. y Begg C. E. (2005). *Sistemas de bases de datos*. 4.ª ed. Madrid: Addison Wesley.

- Elmasri, R. y Navathe, S. B. (2008). *Fundamentos de sistemas de bases de datos*. 5.ª ed. Madrid: Addison Wesley.

- Piattini, M., Marcos, E., Calero, C. y Vela, B. (2006). *Tecnología y diseño de bases de datos*. Madrid: Ediciones RA-MA.

- Silberschatz, A., Korth, H. F., Sudarshan, S. (2006). *Fundamentos de bases de datos*. 5.ª ed. Madrid: McGraw-Hill.

Impreso en los talleres gráficos de

EDITORIAL MACRO®

Surquillo

Agosto 2017