



BASE DE DATOS

Laboratorio

ÍNDICE

Presentación	6
Red de contenidos	7
Unidad de Aprendizaje 1	
FUNDAMENTOS DE SQL SERVER 2014	
1.1 Tema 1 : Introducción a las bases de datos SQL SERVER 2014	11
1.1.1 : Lenguaje estructurado de consultas (SQL)	11
1.1.2 : Historia del lenguaje estructurado	11
1.1.3 : Importancia de la base de datos	12
1.1.4 : SQL Server 2014	14
1.2 Tema 2 : Comandos DDL y bases de datos	21
1.2.1 Comandos DDL	21
1.2.2 : Base de datos en SQL Server 2014: (Estructura, Creación, Modificación y Eliminación)	22
1.3 Tema 3 : Tipos de datos	35
1.3.1 : Identificación de los tipos de datos empleados en SQL SERVER 2014	36
1.4 Tema 4 : Tabla de datos	36
1.4.1 : Creación de una tabla de datos CREATE	37
1.4.2 : Modificación de una tabla de datos ALTER	39
1.4.3 : Eliminación de una tabla de datos DROP	40
1.5 Tema 5 : Restricciones parte I	40
1.5.1 : Asignación de PRIMARY KEY a la tabla de datos	43
1.5.2 : Asignación de FOREIGN KEY a la tabla de datos	44
1.5.3 : Implementación de una base de datos en SQL SERVER 2014	47
1.6 Tema 6 : Restricciones parte II	59
1.6.1 : Restricción DEFAULT	59
1.6.2 : Restricción CHECK	63
1.6.3 : Restricción UNIQUE	68
1.6.4 : Restricción IDENTITY	72
Unidad de Aprendizaje 2	
MANIPULACIÓN DE DATOS	
2.1 Tema 7 : Sentencias DML	91
2.1.1 : Inserción de datos: INSERT	91
2.1.2 : Actualización de datos: UPDATE	95
2.1.3 : Eliminación de datos: DELETE	96
2.1.4 : Integración de sentencias SQL	96
Unidad de Aprendizaje 3	
IMPLEMENTACIÓN DE CONSULTAS	
3.1 Tema 8 : Recuperación de datos	129
3.1.1 : Introducción al lenguaje de consultas SQL	129
3.1.2 : Uso de la sentencia: SELECT, FROM, WHERE, ORDER BY	129
3.1.3 : Manipulación de consultas condicionales: Operadores lógicos (AND, OR y NOT). Operadores de comparación <, <, =, <>,	133

<=, <=. Operador para el manejo de cadenas LIKE. Otros operadores IN, BETWEEN.	
3.1.4 : Funciones para el manejo de fechas (DAY (), MONTH (), YEAR (), DATEPART (), DATEDIFF())	143
3.1.5 : Ejercicios de aplicación	150
3.1.6 : Ejercicios de reforzamiento	152
Unidad de Aprendizaje 4	
INTRODUCCIÓN A LA PROGRAMACIÓN EN TRANSACT / SQL	
4.1 Tema 9 : Introducción a la programación EN SQL SERVER 2014	162
4.1.1 : Declaración de variables locales	162
4.1.2 : Procedimientos almacenados con una tabla	165
4.1.3 : Procedimientos almacenados con uno y dos parámetros de entrada	168
4.1.4 : Procedimientos almacenados de mantenimiento de datos. (INSERT, UPDATE y DELETE)	173
Unidad de Aprendizaje 5	
CONSULTAS MULTITABLA	
5.1 Tema 10 : Uniones Internas (INNER JOIN)	198
5.1.1 : Combinaciones internas con Inner Join	199
5.1.2 : Procedimientos almacenados con 2 o más tablas	201
5.1.3 : Procedimientos almacenados con 2 o más parámetros	203
5.1.4 : Ejercicios de aplicación	206
Unidad de Aprendizaje 6	
AGRUPAMIENTOS, SUBCONSULTAS Y VISTAS	
6.1 Tema 11 : Agrupamiento de datos	214
6.1.1 : Empleo de funciones agregadas	215
6.1.2 : Empleo de GROUP BY y HAVING	217
6.1.3 : Ejercicios con procedimiento almacenado y combinaciones internas	218
6.2 Tema 12 : Subconsultas	220
6.2.1 : Subconsultas	220
6.2.2 : Subconsultas usando procedimientos almacenados con un parámetro	221
6.2.3 : Subconsultas usando procedimientos almacenados con dos o más parámetros	222
6.2.4 : Ejercicios de aplicación	223
6.3 Tema 13 : Vistas	223
6.3.1 : Vistas multitable	224
6.3.2 : Clasificación de las vistas	226
Unidad de Aprendizaje 7	
DIAGRAMADOR DE BASE DE DATOS	
7.1 Tema 14 : Diagramador de SQL Server 2014	241
7.1.1 : Uso del diagramador de base de datos SQL Server 2014	241

Presentación

Base de Datos es un curso que pertenece a la línea de base de datos y se dicta en las carreras de Computación e Informática, Administración y Sistemas. Brinda los conceptos técnicos para diseñar y crear una base de datos relacional.

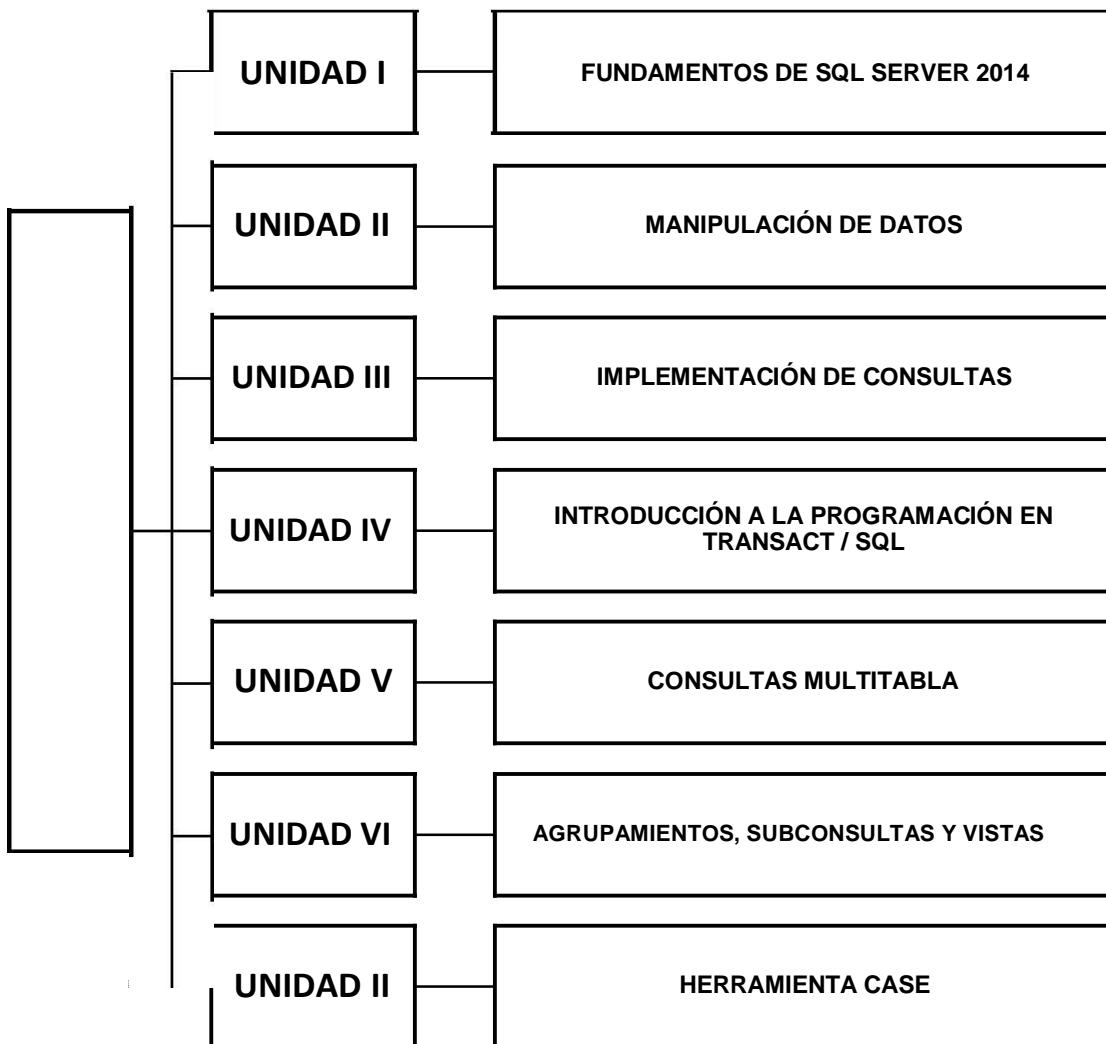
El curso es teórico y práctico. Consiste en la exposición de técnicas para diseño de base de datos que se complementan con casos reales empleados en las empresas del país. En primer lugar, se inicia con el reconocimiento del proceso de negocio. Luego, se determinan los actores (**entidades**) y la forma cómo se relacionan estos, lo cual se complementa con casos prácticos para su ejercitación. En segundo lugar, se efectúa la estandarización de datos a partir de una fuente de los mismos, que, aunado al diseño del proceso de negocio trabajado anteriormente, permite generar una estructura de una base de datos consistente. Finalmente, se concluye con el empleo de una serie de comandos que son la base del lenguaje estructurado de consultas (**SQL**), así como de la exposición de temas de interés.

El curso de Base de Datos (laboratorio), se desarrolla sobre la base de las siguientes unidades de aprendizaje: Fundamentos de SQL SERVER 2014, manipulación de datos, implementación de consultas, introducción a la programación en **Transact-SQL**, consultas multitable, agrupamientos – subconsultas y vistas, herramienta Case.

Al final de cada sesión, se añade un autoaprendizaje, en la cual se proponen ejercicios para que los estudiantes los realicen independientemente, es decir fuera de clases, de manera que puedan aumentar sus habilidades y comprobar sus conocimientos. Para recordar, brinda, de modo resumido, las conclusiones más importantes acerca de los contenidos tratados.

Al finalizar el curso, el alumno diseña, crea e implementa una base de datos para un proceso de negocio que contenga la implementación de reglas de negocio, vistas y procedimientos almacenados, haciendo uso del lenguaje de programación **Transact – SQL** y el gestor de base de datos **SQL SERVER 2014**.

Red de contenidos





FUNDAMENTOS DE SQL SERVER 2014

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la primera unidad, el estudiante construye una base de datos relacional utilizando el gestor de base de datos SQL – Server 2014 y los comandos del Lenguaje de Definición de Datos (DDL), asegurando la integridad de los datos mediante el empleo de restricciones tomando como caso un proceso de negocio real.

TEMARIO

1.1 Tema 1 : Introducción a las bases de datos SQL SERVER 2014

- 1.1.1 : Lenguaje estructurado de consultas (SQL)
- 1.1.2 : Historia del lenguaje estructurado
- 1.1.3 : Importancia de la base de datos
- 1.1.4 : SQL Server 2014

1.2 Tema 2 : Comandos DDL y base de datos

- 1.2.1 Comandos DDL
- 1.2.2 : Base de datos en SQL Server 2014: (Estructura, Creación, Modificación y Eliminación)

1.3 Tema 3 : Tipos de datos

- 1.3.1 : Identificación de los tipos de datos empleados en SQL SERVER 2014

1.4 Tema 4 : Tabla de datos

- 1.4.1 : Creación de una tabla de datos CREATE
- 1.4.2 : Modificación de una tabla de datos ALTER
- 1.4.3 : Eliminación de una tabla de datos DROP

1.5 Tema 5 : Restricciones parte I

- 1.5.1 : Asignación de PRIMARY KEY a la tabla de datos
- 1.5.2 : Asignación de FOREIGN KEY a la tabla de datos
- Implementación de una base de datos en SQL Server 2014

Casos propuestos y desarrollados

1.6 Tema 6 : Restricciones parte II

- 1.6.1 : Restricción DEFAULT
- 1.6.2 : Restricción CHECK
- 1.6.3 : Restricción UNIQUE
- 1.6.4 : Restricción IDENTITY

ACTIVIDADES PROPUESTAS

- Comprenden la visión general del curso.
Deducen la importancia de la existencia de las bases de datos.
Emplean los procedimientos necesarios para crear una base de datos.
Identifican los tipos de datos que se emplean en el SQL SERVER 2014.

1.1. Introducción a las bases de datos SQL SERVER 2014



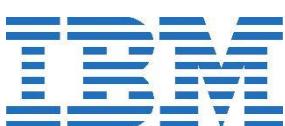
1.1.1. Lenguaje estructurado de consultas (SQL)

El lenguaje de consulta estructurado (**SQL**) es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos.

Pero como sucede con cualquier sistema de normalización, hay excepciones para casi todo. De hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor; por lo tanto, el lenguaje **SQL** normalizado (**ANSI**) no nos servirá para resolver todos los problemas, aunque sí se puede asegurar que cualquier sentencia escrita en **ANSI** será interpretable por cualquier motor de datos.

1.1.2. Historia del lenguaje estructurado

La historia de **SQL** empieza en **1974** con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de **IBM**, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba **SEQUEL** (Structured English Query Language) y se implementó en un prototipo llamado **SEQUEL-XRM** entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (**SEQUEL/2**) que, a partir de ese momento, cambió de nombre por motivos legales y se convirtió en **SQL**.



El prototipo (**System R**), basado en este lenguaje, se adoptó y utilizó internamente en **IBM** y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, otras compañías empezaron a desarrollar sus productos relacionales basados en **SQL**. A partir de 1981, **IBM** comenzó a entregar sus productos relacionales y, en 1983, empezó a vender **DB2**. En el curso de los años ochenta, numerosas compañías (por ejemplo, Oracle y Sybase, sólo por citar algunas) comercializaron productos basados en **SQL**, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó **SQL** (sustancialmente adoptó el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transformó en estándar **ISO**. Esta versión del estándar va con el nombre de **SQL/86**. En los años siguientes, este ha sufrido diversas revisiones que han conducido primero a la versión **SQL/89** y, posteriormente, a la actual **SQL/92**.

El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicación entre todos los productos que se basan en él. Desde el punto de vista práctico, por desgracia las cosas fueron de otro modo. Efectivamente, en general cada productor adopta e implementa, en la propia base de datos solo el corazón del lenguaje **SQL** (el así llamado Entry level o al máximo el Intermediate level), y lo extiende de manera individual según la propia visión que cada cual tenga del mundo de las bases de datos.

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987 .
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares , consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML , cambios en las funciones, estandarización del objeto sequence y de las columnas autonuméricas.
2005	SQL:2005		ISO/IEC 9075-14:2005 Define las maneras en las cuales SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL:2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursor. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.

Figura 1: Historia de versiones del lenguaje SQL

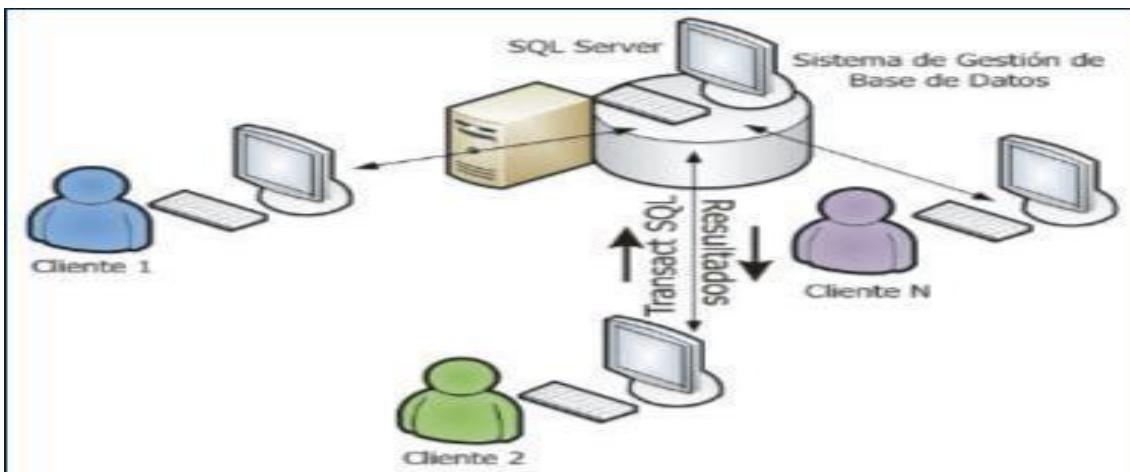
Fuente.- Tomado de https://es.wikipedia.org/wiki/Microsoft_SQL_Server

1.1.3. Importancia de la base de datos

Las bases de datos son importantes porque permiten almacenar grandes cantidades de información en forma estructurada, consistente e íntegra y dan la posibilidad a un desarrollador de utilizarlas mediante programas (aplicaciones); además, les proporciona a estos una herramienta bajo la cual puedan reducir considerablemente el tiempo del proceso de búsqueda en profundidad de los datos almacenados.

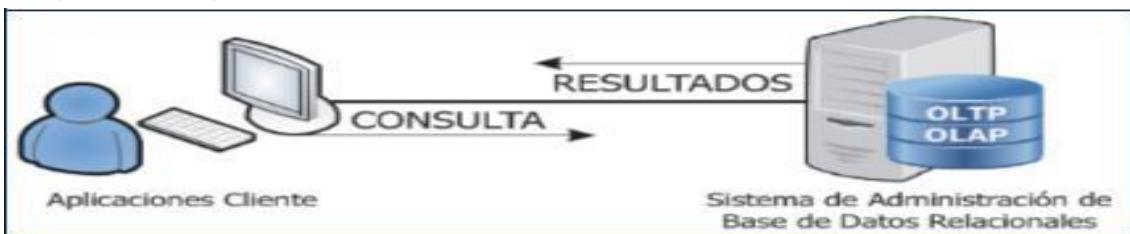
1.1.3.1 Implementación de las bases de datos con SQL SERVER

SQL Server es un sistema administrador para Bases de Datos relacionales basadas en la arquitectura **Cliente / Servidor (RDBMS)** que usa **Transact SQL** para mandar peticiones entre un cliente y el **SQL Server**.



1.1.3.2 Arquitectura Cliente / Servidor

SQL Server usa la arquitectura **Cliente / Servidor** para separar la carga de trabajo en tareas que se ejecuten en computadoras tipo Servidor y tareas que se ejecuten en computadoras tipo **Cliente**:



El **cliente** es responsable de la parte lógica y de presentar la información al usuario. Generalmente, el cliente ejecuta en una o más computadoras Cliente, aunque también puede ejecutarse en una computadora que cumple las funciones de Servidor con SQL Server.

El **servidor** SQL Server administra bases de datos y distribuye los recursos disponibles del servidor tales como memoria, operaciones de disco, etc. entre las múltiples peticiones.

La **arquitectura Cliente/Servidor** permite desarrollar aplicaciones para realizarlas en una variedad de ambientes.

SQL Server 2014 trabaja con dos (2) tipos de bases de datos:

OLTP	Online Transaction Processing Son bases de datos caracterizadas por mantener una gran cantidad de usuarios conectados concurrentemente realizando ingreso y/o modificación de datos. Por ejemplo: entrada de pedidos en línea, inventario, contabilidad o facturación.
OLAP	OnLine Analytical Processing Son bases de datos que almacenan grandes cantidades de datos que sirven para la toma de decisiones como, por ejemplo, las aplicaciones de análisis de ventas.

1.1.3.3 Transact SQL

Esta es una versión de **SQL** (Structured Query Language) usada como lenguaje de programación para **SQL Server**. **SQL** es un conjunto de comandos que permite especificar la información que se desea restaurar o modificar. Con **Transact SQL** se puede tener acceso a la información, realizar búsquedas, actualizar y administrar sistemas de bases de datos relacionales.

Aunque se denomine **SQL**, debido a que el propósito general es recuperar datos, realmente **SQL** nos brinda muchas más opciones. Es una herramienta mucho más interesante. Podemos utilizar más funciones de las que el **DBMS** (Database Management System - Sistema de Gestión de base de datos) nos proporciona.

1.1.4 SQL SERVER 2014



Figura 2: Logo SQL Server 2014

Fuente.- Tomado de <http://stephanefrechette.com/blog/wp-content/uploads/2014/04/sqlserver2014.png>

SQL Server 2014 se basa en las funciones críticas ofrecidas en la versión anterior, proporcionando un rendimiento, una disponibilidad y una facilidad de uso innovadores para las aplicaciones más importantes. **Microsoft SQL Server 2014** ofrece nuevas capacidades en memoria en la base de datos principal para el procesamiento de transacciones en línea (**OLTP**) y el almacenamiento de datos, que complementan nuestras capacidades de almacenamiento de datos en memoria y **BI** (Business Intelligence) existentes para lograr la solución de base de datos en memoria más completa del mercado.

SQL Server 2014 también proporciona nuevas soluciones de copia de seguridad y de recuperación ante desastres, así como de arquitectura híbrida con **Windows Azure**, lo que permite a los clientes utilizar sus actuales conocimientos con características locales que aprovechan los centros de datos globales de Microsoft. Además, **SQL Server 2014** aprovecha las nuevas capacidades de **Windows Server 2012** y **Windows Server 2012 R2** para ofrecer una escalabilidad sin parangón a las aplicaciones de base de datos en un entorno físico o virtual.

1.1.4.1 Ediciones del SQL Server 2014

SQL Server 2014 Enterprise (64 y 32 bits)

Proporciona capacidades de centro de datos de tecnología avanzada completas con un rendimiento ultrarrápido, virtualización ilimitada y Business Intelligence integral, que habilita los mayores niveles de servicio para las cargas de trabajo de gran importancia y el acceso del usuario final a ideas claras de los datos.



Figura 3: Logo SQL Server 2014

Fuente.- Tomado de http://s14.postimg.org/v3g5hljh/Microsoft_SQL_Server_2014_Enterprise_Edition_S.png

SQL Server 2014 Business Intelligence (64 y 32 bits)

Ofrece una plataforma completa que capacita a las organizaciones para crear e implementar soluciones de **BI** seguras, escalables y fáciles de administrar. Proporciona funcionalidad emocionante, como exploración y visualización de datos en un explorador; funciones eficaces de mezcla de datos y administración de integración mejorada.



Figura 4: Logo SQL Server 2014

Fuente.- Tomado de <https://empoweruy.files.wordpress.com/2013/11/sqlserver2012businessintelligence.png>

SQL Server 2014 Standard (64 y 32 bits)

Proporciona administración básica de bases de datos y base de datos de Business Intelligence para que los departamentos y pequeñas organizaciones ejecuten sus aplicaciones y admite las herramientas de desarrollo comunes, tanto locales como en la nube, que habilitan la administración eficaz de bases de datos con recursos de TI mínimos.



Figura 5: Logo SQL Server 2014

Fuente.- Tomado de <http://zdnet4.cbsistatic.com/hub/i/r/2014/09/18/74d1c2ba-3f02-11e4-b6a0-d4ae52e95e57/thumbnail/770x578/411aa89b0a072453faa3e046e11793db/microsoft-release-sql-server-2014-ctp2.png>

SQL Server 2014 Web (64 y 32 bits)

Es una opción con un costo total de propiedad bajo para los hosts de Web y los **VAP** de Web que proporciona capacidades asequibles de administración y escalabilidad para propiedades web, tanto de pequeña como de gran escala.

SQL Server 2014 Developer (64 y 32 bits)

Permite a los desarrolladores compilar cualquier tipo de aplicación en **SQL Server**. Incluye toda la funcionalidad de la edición Enterprise, pero tiene licencias para usarse como sistema de prueba y desarrollo, no como un servidor de producción.

SQL Server Developer es una opción ideal para las personas que compilan y prueban aplicaciones.

SQL Server 2014 Express (64 y 32 bits)

Es una base de datos gratuita para principiantes y es ideal para aprender a compilar pequeñas aplicaciones de servidor y de escritorio orientadas a datos. Es la mejor opción para los fabricantes de software independientes, los desarrolladores y los aficionados que compilan aplicaciones cliente. Si necesita características de base de datos más avanzadas, **SQL Server Express** se puede actualizar sin problemas a otras versiones superiores de SQL Server.

Express LocalDB de SQL Server es una versión ligera de Express que tiene todas sus características de capacidad de programación, pero se ejecuta en modo usuario y tiene una instalación rápida sin configuración y una lista reducida de requisitos previos.

1.1.4.2 Ejecutando SQL SERVER 2014

Para ejecutar SQL SERVER 2014 haga lo siguiente:

Botón Inicio de Windows, buscar la carpeta Microsoft SQL Server 2014 y luego clic en la opción **SQL Server Management Studio**, tal como muestra en la siguiente imagen:



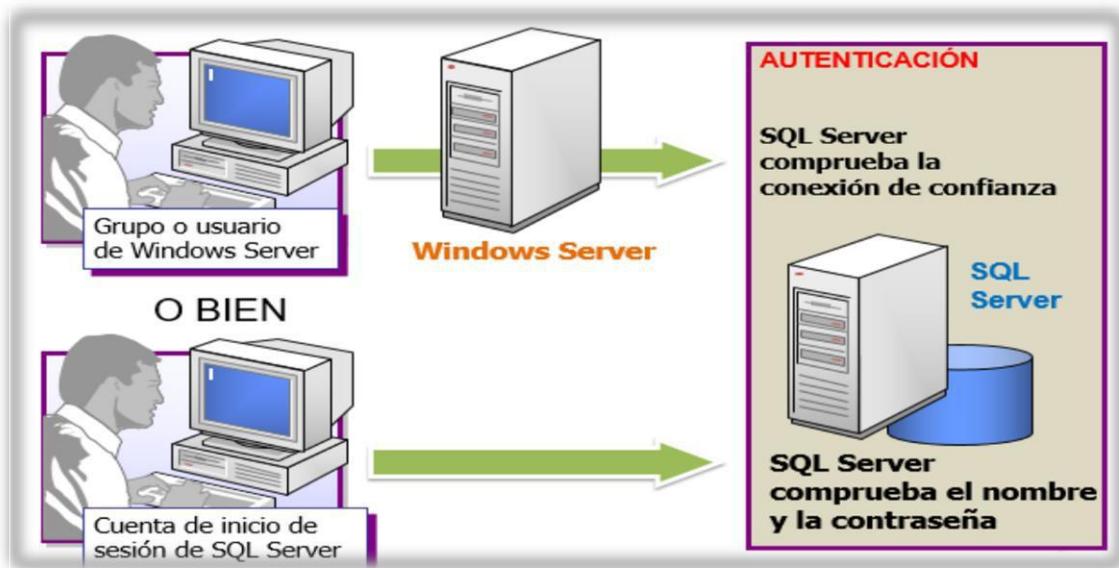
1.1.4.3 Inicio de Sesión en SQL SERVER 2014

Un inicio de sesión es una entidad de seguridad o una entidad que puede ser autenticada por un sistema seguro. Los usuarios necesitan iniciar sesión para conectarse a SQL Server. Puede crear un inicio de sesión basado en una entidad de seguridad de Windows (como un usuario de dominio o un grupo de dominio de Windows) o puede crear un inicio de sesión que no lo esté (como un inicio de sesión de SQL Server).

NOTA: Para usar la autenticación de SQL Server, el Motor de base de datos debe utilizar la autenticación de modo mixto.

Como entidad de seguridad, se pueden conceder permisos a los inicios de sesión. El ámbito de un inicio de sesión es todo el Motor de base de datos. Para establecer conexión con una base de datos concreta de la instancia de SQL Server, un inicio de sesión debe estar asignado a un usuario de la base de datos. Los permisos dentro de la base de datos se conceden y deniegan al usuario de la base de datos, no al inicio de sesión. Los permisos que tienen como ámbito la instancia completa de SQL Server (por ejemplo, el permiso **CREATE ENDPOINT**) se pueden conceder a un inicio de sesión.

NOTA: Cuando un inicio de sesión se conecta a SQL Server, la identidad se valida en la base de datos maestra. Use los usuarios de base de datos independiente para autenticar conexiones SQL Server/Base de datos SQL en el nivel de base de datos. No se necesita un inicio de sesión si se usan usuarios de base de datos independiente. Una base de datos independiente es una base de datos que está aislada de otras bases de datos y de la instancia de SQL Server/Base de datos SQL (y de la base de datos maestra) que hospeda la base de datos. SQL Server admite usuarios de base de datos independientes para la autenticación de Windows y SQL Server. Al usar Base de datos SQL, se combinan las reglas de usuarios de la base de datos independiente con las de firewall de nivel de base de datos.



1.1.4.4 Ingresando a SQL SERVER 2014

Antes de ingresar a SQL SERVER 2014 debemos tener en cuenta lo siguiente:

Modo de autenticación

Durante la instalación, debe seleccionar un modo de autenticación para Motor de base de datos. **Hay dos modos posibles:**

Modo de autenticación de Windows

Modo mixto.

El modo de autenticación de Windows habilita la autenticación de Windows y deshabilita la autenticación de SQL Server. El modo mixto habilita tanto la autenticación de Windows como la de SQL Server. La autenticación de Windows está disponible siempre y no se puede deshabilitar.

Conectarse a través de:

1. Modo de autenticación de Windows

Cuando un usuario se conecta a través de una cuenta de usuario de Microsoft Windows, SQL Server valida el nombre de cuenta y la contraseña con el token de la entidad de seguridad de Windows del sistema operativo. Esto significa que Windows confirma la identidad del usuario. SQL Server no pide la contraseña y no realiza la validación de identidad. La autenticación de Windows es el modo de autenticación predeterminado y es mucho más seguro que la autenticación de SQL Server. La autenticación de Windows usa el protocolo de seguridad de Kerberos, proporciona la aplicación de directivas de contraseñas en cuanto a la validación de la complejidad de las contraseñas seguras, ofrece compatibilidad para el bloqueo de cuentas y admite la expiración de las contraseñas. Una conexión realizada utilizando la autenticación de Windows se denomina a veces conexión de confianza, porque SQL Server confía en las credenciales proporcionadas por Windows.

Cuando se emplea la autenticación de Windows, se pueden crear grupos de Windows en el nivel de dominio y se puede crear un inicio de sesión en SQL Server para todo el

grupo. La administración del acceso desde el nivel de dominio puede simplificar la administración de cuentas.

Importante

Siempre que sea posible, utilice la autenticación de Windows.

2. Modo mixto

Cuando se utiliza la autenticación de SQL Server, los inicios de sesión se crean en SQL Server y no se basan en cuentas de usuario de Windows. El nombre de usuario y la contraseña se crean utilizando SQL Server y se almacenan en SQL Server. Los usuarios que se conectan usando la autenticación de SQL Server deben indicar sus credenciales (inicio de sesión y contraseña) cada vez que se conectan. Al utilizar la autenticación de SQL Server, debe establecer contraseñas seguras para todas las cuentas de SQL Server. Hay tres directivas de contraseñas opcionales para los inicios de sesión de SQL Server.

El usuario debe cambiar la contraseña en el siguiente inicio de sesión

Exigir expiración de contraseña

Exigir directivas de contraseñas

Desventajas de la autenticación de SQL Server

Si un usuario del dominio de Windows tiene un inicio de sesión y una contraseña para Windows, aún debe proporcionar otro inicio de sesión y contraseña (SQL Server) para conectarse. Hacer el seguimiento de varios nombres y contraseñas es difícil para muchos usuarios. Tener que proporcionar las credenciales de SQL Server cada vez que se conectan a la base de datos puede resultar molesto.

SQL Server no puede utilizar el protocolo de seguridad de Kerberos.

Windows proporciona directivas de contraseñas adicionales que no están disponibles para los inicios de sesión de SQL Server.

La contraseña de inicio de sesión cifrada de la autenticación de SQL Server se debe pasar a través de la red en el momento de la conexión. Algunas aplicaciones que se conectan automáticamente almacenarán la contraseña en el cliente. Estos puntos de ataque adicionales.

Ventajas de la autenticación de SQL Server

Permite a SQL Server admitir aplicaciones anteriores y aplicaciones proporcionadas por terceros que requieren la autenticación de SQL Server.

Permite que SQL Server admita entornos con sistemas operativos mixtos, en los que un dominio de Windows no autentica a todos los usuarios.

Permite a los usuarios conectarse desde dominios desconocidos o que no son de confianza. Por ejemplo, una aplicación en la que los clientes establecidos se conectan con los inicios de sesión de SQL Server asignados para recibir el estado de sus pedidos.

Permite que SQL Server admita aplicaciones basadas en web en las que los usuarios crean sus propias identidades.

Permite a los desarrolladores de software distribuir sus aplicaciones utilizando una jerarquía de permisos compleja basada en los inicios de sesión conocidos y preestablecidos de SQL Server.

Nota:

Al utilizar la autenticación de SQL Server, no se limitan los permisos de los administradores locales en el equipo donde se instala SQL Server.

Para empezar a interactuar ingresaremos seleccionando el **modo mixto**. Esto indica que debemos especificar el login y el password. Como ejemplo será: Login: **sa**

Password: **sql** (minúscula)



1.1.4.5 Tipos de base de datos de SQL SERVER 2014

Los tipos de bases de datos que maneja SQL Server 2014 son:

- Bases de datos del sistema
- Base de datos del usuario

Bases de datos del sistema

SQL Server incluye las siguientes bases de datos del sistema.

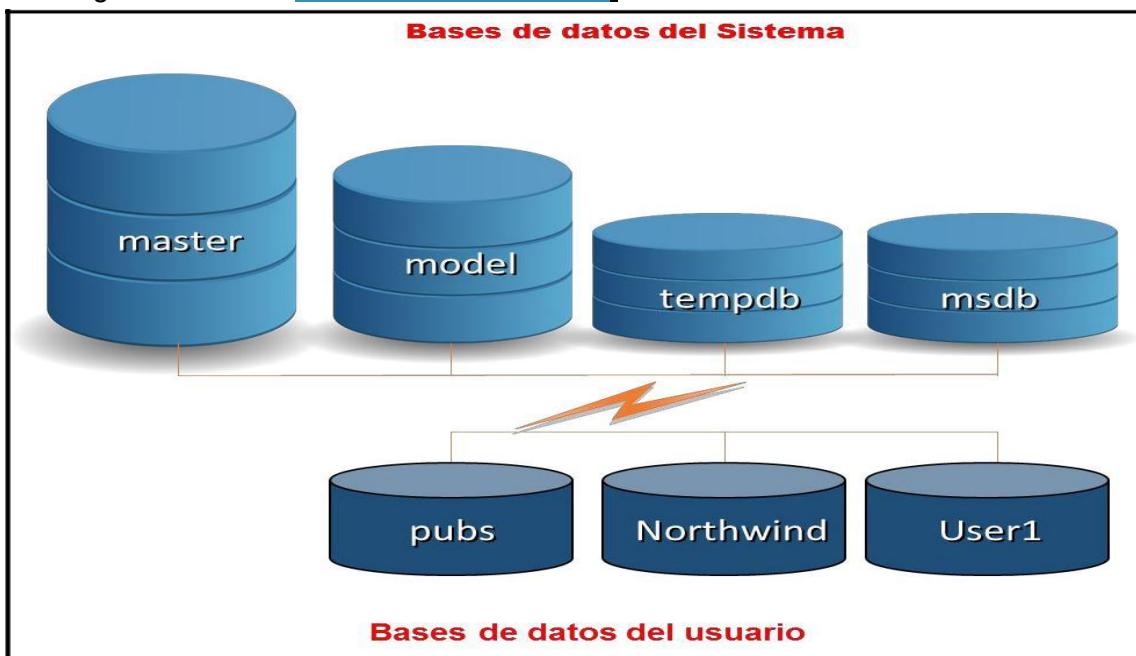
BASE DE DATOS DEL SISTEMA	DESCRIPCIÓN
MASTER	Registra toda la información del sistema para una instancia de SQL Server.
MSDB	La utiliza el Agente SQL Server para programar alertas y trabajos.

BASE DE DATOS DEL SISTEMA	DESCRIPCIÓN
MODEL	Se utiliza como plantilla para todas las bases de datos creadas en la instancia de SQL Server. Las modificaciones hechas a la base de datos model , como el tamaño de la base de datos, la intercalación, el modelo de recuperación y otras opciones de base de datos, se aplicarán a las bases de datos que se creen con posterioridad.
TEMPDB	Área de trabajo que contiene objetos temporales o conjuntos de resultados intermedios.

Bases de datos del usuario

BASE DE DATOS DEL USUARIO
PUBS
NORTHWIND
USER1

Las bases de datos de ejemplo **Northwind** y **pubs** no se instalan de forma predeterminada en Microsoft SQL Server 2014. Estas bases de datos se pueden descargar desde este [sitio Web de Microsoft](#).

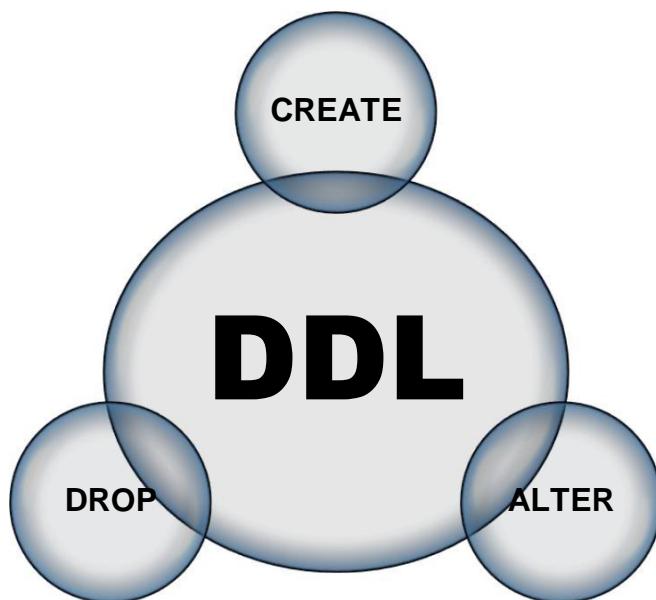


1.2. Comandos DDL y Bases de Datos

1.2.1. Comandos DDL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

El lenguaje de definición de datos (en inglés Data Definition Language, o **DDL**), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Los componentes son: **CREATE, ALTER, DROP**.



CREATE | CREAR

Este comando permite crear objetos de datos, como nuevas bases de datos, tablas, vistas y procedimientos almacenados.

Ejemplos:

```
CREATE TABLE PRODUCTOS  
CREATE DATABASE COMERCIO  
CREATE PROCEDURE SP_LISTAR_PRODUCTOS  
CREATE TRIGGER MENSAJE  
CREATE INDEX IDX_EMPLEADO
```

ALTER | MODIFICAR

Este comando permite modificar la estructura de una tabla u objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplos:

```
ALTER DATABASE BD_INSTITUTO  
MODIFY NAME=INSTITUTO  
GO
```

```
ALTER DATABASE BD_FARMACIA  
ADD FILE  
(  
)
```

```
ALTER TABLE PRODUCTO  
ADD STOCK_MAX INT NOT NULL  
GO
```

```
ALTER TABLE DETALLE_BOLETA  
ADD FOREIGN KEY (ID_PRODUCTO) REFERENCES  
PRODUCTO GO
```

DROP | ELIMINAR

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplos:

```
DROP TABLE ALUMNOS  
DROP DATABASE BD_FARMACIA  
DROP INDEX IDX_EMPLEADO  
DROP PROCEDURE SP_SP_LISTAR_PRODUCTOS
```

1.2.2. Base de datos en SQL Server 2014

1.2.2.1 ¿Qué es una Base de datos?

Una **base de datos** es una colección de información organizada de forma que un programa de ordenador pueda seleccionar rápidamente los fragmentos de datos que necesite. Una base de datos es un sistema de archivos electrónico.

Las bases de datos tradicionales se organizan por campos, registros y archivos. Un **campo** es una pieza única de información; un **registro** es un sistema completo de campos; y un **archivo** es una colección de registros. Por ejemplo, una guía de teléfono es análoga a un archivo. Contiene una lista de registros, cada uno de los cuales consiste en tres campos: nombre, dirección, y número de teléfono.

1.2.2.2 Estructura

Para crear una base de datos, determine el nombre de la base de datos, el propietario (el usuario que crea la base de datos), su tamaño, y los archivos y grupos de archivos utilizados para almacenarla.

Antes de crear una base de datos, considere 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 **32767** bases de datos.

Se utilizan tres (**03**) tipos de archivos para almacenar una base de datos:

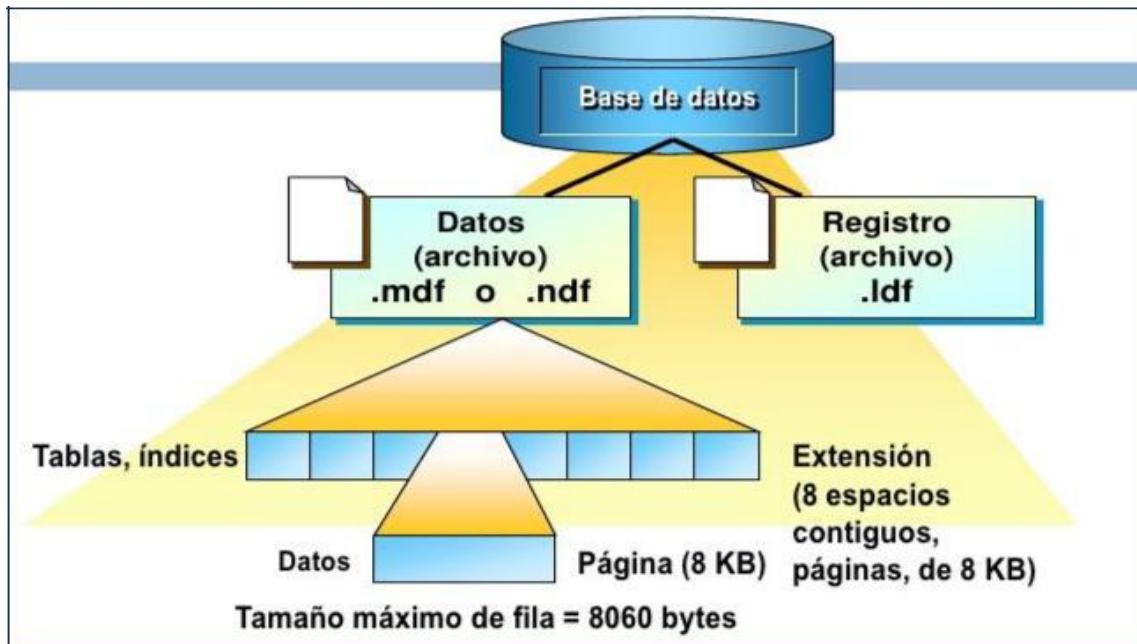


Figura 6: Componentes de una base de datos en SQL Server

Fuente.- Tomado de <http://image.slidesharecdn.com/transactionlog-091106221712-phpapp02/95/sql-server-como-se-almacenan-los-datos-2-728.jpg?cb=1257567466>

Definiremos los tipos de archivos que componen una base de datos:

Archivo Principal	Estos archivos contienen la información de inicio para la base de datos. Este archivo se utiliza también para almacenar datos. Cada base de datos tiene un único archivo principal. Tiene extensión .MDF .
Archivo Secundario	Estos archivos contienen todos los datos que no caben en el archivo de datos principal. 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. Algunas bases de datos pueden ser muy grandes y necesitar varios archivos de datos secundarios o utilizar archivos secundarios en unidades de disco distintas, de modo que los datos estén distribuidos en varios discos. Tiene extensión .NDF .
Archivo de Transacciones	Estos archivos contienen la información de registro que se utiliza para recuperar la base de datos. Debe haber al menos un archivo de registro de transacciones para cada base de datos, aunque puede haber más de uno. El tamaño mínimo para un archivo de registro es 512 kilobytes (KB) . Tiene extensión .LDF .

Algunas consideraciones:

Los archivos de datos y de registro de transacciones de **Microsoft® SQL Server™ 2014** no deben colocarse en sistemas de archivos comprimidos ni en una unidad de red remota, como un directorio de red compartido.

Cuando se crea una base de datos, todos los archivos que la componen se llenan con ceros que suplantan los datos de los archivos ya eliminados que hubieran quedado en el disco. Aunque esto provoque que el proceso de creación de los archivos sea más largo, es mejor, pues así se evita que el sistema operativo tenga que llenar los archivos con ceros cuando se escriban por primera vez datos en los archivos durante las operaciones habituales con la base de datos. De esta manera, se mejora el rendimiento de las operaciones cotidianas.

Es recomendable especificar el tamaño máximo de crecimiento del archivo. De ese modo se evita que se agote el espacio disponible en el disco al agregar datos. Para especificar un tamaño máximo para el archivo, utilice el parámetro **MAXSIZE** de la instrucción **CREATE DATABASE** o bien la opción Limitar crecimiento de archivo a **(MB)** cuando utilice el cuadro de diálogo Propiedades del Administrador corporativo de **SQL Server** para crear la base de datos.

Después de crear una base de datos, se recomienda crear una copia de seguridad de la base de datos **MASTER**.

1.2.2.3 Creación

Debemos tener en cuenta que una base de datos se compone mínimamente de un **archivo principal y un archivo de transacciones**. Al crear una base de datos **estándar** estos archivos se implementarán de manera automática, esto nos liberará de pensar en las especificaciones que pudiera tener dichos archivos, pero nos crea una desventaja en el control de los mismos. Por ejemplo, se presentan las siguientes cuestiones:

¿Dónde se guarda la base de datos?	La ubicación de los archivos de una base de datos dependerá del proceso de instalación del SQL Server; es decir, la dirección varía solo en la unidad de disco. Por ejemplo, veamos la dirección de una base de datos creada en SQL Server 2014 instalado en la unidad C: C:\Program Files\Microsoft SQL Server\ MSSQL12.SQLEXPRESS\ MSSQL\DATA
¿Cuál es el tamaño asignado a los archivos de la base de datos?	<p>Archivo Principal</p> <ul style="list-style-type: none"> - Tamaño: 4288 KB - Máximo Tamaño: Ilimitado - Tasa de Crecimiento: 1024 KB <p>Archivo de Transacciones</p> <ul style="list-style-type: none"> - Tamaño: 1072 KB - Máximo Tamaño: 2147483648 KB - Tasa de Crecimiento: 10%

Nota:

Antes de crear la base de datos en SQL SERVER 2014 tenga en cuenta lo siguiente:

Es posible crear una base de datos de manera **Estándar**

Es posible crear una base de datos de manera **Personalizada**

De manera estándar

Sintaxis para la creación de una base de datos estándar:

```
CREATE DATABASE [NOMBRE_BASE_DATOS]
GO
```

Donde:

CREATE DATABASE: Es la sentencia de creación de base de datos en el servidor.

[NOMBRE_BASE_DATOS]: Es el nombre que se le asigna a la base de datos; debemos tener en cuenta que dicho nombre no debe empezar con un numero o algún carácter especial y tampoco debe contener espacios en blanco.

GO: Es un comando que indica el final de un lote de sentencias.

De manera personalizada

Crear una base de datos personalizada nos permite especificar las propiedades de cada uno de los archivos que la compone e inclusive podemos agregar archivos adicionales como los archivos secundarios.

Sintaxis para la creación de una base de datos personalizada:

```
CREATE DATABASE [NOMBRE_BASE_DATOS]
ON
(
    NAME=NOMBRELOGICO_ARCHIVO,
    FILENAME='RUTA DEL ARCHIVO',
    SIZE=TAMAÑO INICIAL,
    MAXSIZE=MAXIMO TAMAÑO,
    FILEGROWTH=TASA DE CRECIMIENTO
)
GO
```

Describiendo las propiedades del archivo de una base de datos

Los archivos que componen una base de datos tienen características similares, lo que lo diferencia son los valores en cada una de las características:

NAME	Es el nombre lógico del archivo primario; no puede haber dos archivos lógicos con el mismo nombre en una misma base de datos. Por ejemplo: NAME = 'VENTAS'
FILENAME	Es la especificación de la ruta y el nombre del archivo físico; estos nombres seran visibles desde el explorador de archivos de windows. FILENAME='C:\base\ventas.mdf'

	Cuando se especifica una carpeta en la ruta, esta deberá estar creada antes de ejecutar la sentencia. Para este caso la carpeta “base” deberá existir en la unidad c.
SIZE	Es el tamaño inicial del archivo, debemos considerar que los tamaños se especifican en KB (Kilobyte), MB (Megabyte), GB (Gigabyte) y que cuando no se especifica se entiende que es MB. Por ejemplo: SIZE=2048KB SIZE=2MB SIZE=2 Todas las representaciones son similares; lo que lo diferencia es la forma de especificar la unidad de medida.
MAXSIZE	Es la definición del máximo tamaño que puede llegar a tener una base de datos. Estos se pueden dar en KB, MB, GB, etc. En caso no tengamos fijo el máximo tamaño podemos optar por asignar UNLIMITED, el cual usa el tamaño máximo permitido por el sistema operativo y la capacidad de disco con que se cuenta. Por ejemplo: MAXSIZE=300 (300 Megabyte) MAXSIZE=UNLIMITED (Límite permitido por el sistema)
FILEGROWTH	Es la definición de la tasa de crecimiento; esto puede darse en tamaños específicos (KB, MB o GB) o en porcentajes. Debemos tener en cuenta que esta definición solo será efectiva cuando la información almacenada cope el tamaño definido en la cláusula SIZE. FILEGROWTH=10 (Amplía el tamaño actual en 10MB). FILEGROWTH=10% (Amplía el tamaño actual en un 10% en el momento en que tiene lugar el incremento).

Actividades

Caso 1: BD_EJEMPLO

Crear la base de datos **BD_EJEMPLO** de forma estándar y visualizar las características de sus archivos:

```
--1. Creando la base de datos BD_EJEMPLO
CREATE DATABASE BD_EJEMPLO GO
```

```
--2. Visualizando las características de sus archivos
SP_HELPDB BD_EJEMPLO
GO
```

	name	db_size	owner	dbid	created	status	compatibility_level		
1	BD_EJEMPLO	5.23 MB	sa	6	Jun 22 2015	Status=ONLINE, Updateability=READ_WRITE, UserAccess=MULTI_USER, Recovery=SIMPLE, Version=782, Coll...	120		
	name	fileid	filename		filegroup	size	maxsize	growth	usage
1	BD_EJEMPLO	1	C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\BD_EJEMPLO.mdf		PRIMARY	4288 KB	Unlimited	1024 KB	data only
2	BD_EJEMPLO_log	2	C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\BD_EJEMPLO_log.ldf		NULL	1072 KB	2147483648 KB	10%	log only

FIN

Validando la existencia de la base de datos

Por conveniencia pedagógica validaremos la existencia de la base de datos; esto nos servirá para poder ejecutar las sentencias y realizar modificaciones cuando lo crea conveniente. Para esto usaremos la sentencia **IF** con el siguiente formato:

```
IF DB_ID('NOMBRE_BASE_DATOS') IS NOT NULL
BEGIN
    DROP DATABASE NOMBRE_BASE_DATOS
END
GO
```

Donde:

IF: Es la sentencia condicional que permitira condicionar la existencias de la base de datos.

DB_ID: Función que permite devolver el identificador de una base de datos.

IS NOT NULL: Determina si la evaluación especificada es no **NULA**, es decir, si existe.

BEGIN...END: Cláusula que permite determinar el inicio y fin de un conjunto de sentencias.

DROP DATABASE: Sentencia **DDL** que permite eliminar una base de datos.

Veamos como validar la existencia de la base de datos **BD_COMERCIO**:

```
IF DB_ID('BD_COMERCIO') IS NOT NULL
    DROP DATABASE BD_COMERCIO
GO
```

Podemos interpretar de la siguiente manera, si la base de datos **BD_COMERCIO** existe, entonces se eliminará dicha base, caso contrario se eliminará.

Caso 2: **BD_COMERCIAL**

Cree una base de datos llamada **BD_COMERCIAL** teniendo en cuenta las siguientes opciones:

Un archivo físico y el archivo lógico se creará automáticamente.

Valide la existencia de la base de datos.

Asigne al archivo físico el nombre (**NAME**) “**BD_COMERCIAL_PRI**” al archivo principal con un tamaño inicial de **50MB (SIZE)**, un tamaño máximo aceptado por el sistema (**MAXSIZE**), una tasa de crecimiento del **10% (FILEGROWTH)** y debe ser guardado en la carpeta **C:\COMERCIAL (FILENAME)**.

Verifique la existencia de los archivos implementados.

```

IF DB_ID('BD_COMERCIAL') IS NOT NULL
BEGIN
    DROP DATABASE BD_COMERCIAL
END
GO

CREATE DATABASE BD_COMERCIAL
ON
(
    NAME=BD_COMERCIAL_PRI,
    FILENAME='C:\COMERCIAL\BD_COMERCIAL_PRI.MDF',
    SIZE=50,
    MAXSIZE=UNLIMITED,
    FILEGROWTH=10%
)
GO
SP_HELPDB BD_COMERCIAL
GO

```

Caso 3: COMERCIAL2017

Cree una base de datos llamada **COMERCIAL2017** teniendo en cuenta las siguientes opciones:

Un archivo físico y el archivo lógico.

Valide la existencia de la base de datos.

Asigne al archivo físico principal de nombre “**COMERCIAL_PRI**”, con un tamaño inicial de **40MB**, un tamaño máximo de **250MB**, una tasa de crecimiento del **5MB** y debe ser guardado en la carpeta **C:\COMERCIAL\DATA**.

Asigne al archivo lógico el nombre “**COMERCIAL_TRA**” al archivo de transacciones con un tamaño inicial de **10MB**, un tamaño máximo de **50MB**, una tasa de crecimiento del **5%** y debe ser guardado en la carpeta **C:\COMERCIAL\LOG**.

Verifique la existencia de los archivos implementados.

```

IF DB_ID('COMERCIAL2017') IS NOT NULL
BEGIN
    DROP DATABASE COMERCIAL2017
END
GO

CREATE DATABASE COMERCIAL2017
ON
(
    NAME=COMERCIAL_PRI,
    FILENAME='C:\COMERCIAL\DATA\COMERCIAL_PRI.MDF',
    SIZE=40,
    MAXSIZE=250,
    FILEGROWTH=5MB
)

```

```

LOG ON (
    NAME=COMERCIAL_LOG,
    FILENAME='C:\COMERCIAL\LOG\COMERCIAL_TRA.LDF',
    SIZE=10,
    MAXSIZE=50,
    FILEGROWTH=5%
)
GO
SP_HELPDB COMERCIAL2017
GO

```

Caso 4: BD_TIENDA

Diseñe una base de datos llamada **BD_TIENDA** teniendo en cuenta las siguientes opciones:

Archivo físico principal, archivo físico secundario y el archivo lógico.

Valide la existencia de la base de datos.

Asigne al archivo físico principal el nombre “**BD_TIENDA_PRI**”, con un tamaño inicial de **100MB**, un tamaño máximo permitido por el sistema, una tasa de crecimiento del **15%** y debe ser guardado en la carpeta **C:\TIENDA\DATA**.

Asigne al archivo físico secundario el nombre “**BD_TIENDA_SEC**”, con un tamaño inicial de **50MB**, un tamaño máximo de **250MB**, una tasa de crecimiento del **10%** y debe ser guardado en la carpeta **C:\TIENDA\SEC**.

Asigne al archivo de transacciones el nombre “**BD_TIENDA_TRA**”, con un tamaño inicial de **5MB**, un tamaño máximo de **50MB**, una tasa de crecimiento del **5MB** y debe ser guardado en la carpeta **C:\TIENDA\LOG**.

Verifique la existencia de los archivos implementados.

```

IF DB_ID('BD_TIENDA') IS NOT NULL
BEGIN
    DROP DATABASE BD_TIENDA
END
GO
CREATE DATABASE BD_TIENDA
ON
(
    NAME=BD_TIENDA_PRI,
    FILENAME='C:\TIENDA\DATA\BD_TIENDA_PRI.MDF',
    SIZE=100,
    MAXSIZE=UNLIMITED,
    FILEGROWTH=15%
),
(
    NAME=BD_TIENDA_SEC,
    FILENAME='C:\TIENDA\SEC\BD_TIENDA_SEC.NDF',
    SIZE=50,
    MAXSIZE=250,
    FILEGROWTH=10%
)

```

```
LOG ON (
    NAME=BD_TIENDA_LOG,
    FILENAME='C:\TIENDA\LOG\BD_TIENDA_TRA.LDF',
    SIZE=5,
    MAXSIZE=50,
    FILEGROWTH=5
)
GO
SP_HELPDB BD_TIENDA
GO
```

1.2.2.4 Modificación

Cuando se trata de modificar en una base de datos SQL SERVER 2014, se utiliza el comando **ALTER**. En seguida realizaremos algunos ejemplos:

Caso 1: Modificar el nombre de la base de datos **BD_TIENDA** por **MINIMARKET**

```
USE MASTER
GO
ALTER DATABASE BD_TIENDA
MODIFY NAME=MINIMARKET
GO
```

Caso 2: Agregar 2 archivos secundarios a la base de datos COMERCIAL2017

```
ALTER DATABASE COMERCIAL2017
ADD FILE
(
    NAME=COMERCIAL_SEC1,
    FILENAME='C:\COMERCIAL\SEC\COMERCIAL_SEC1.NDF',
    SIZE=10,
    MAXSIZE=50,
    FILEGROWTH=10%
),
(
    NAME=COMERCIAL_SEC2,
    FILENAME='C:\COMERCIAL\SEC\COMERCIAL_SEC2.NDF',
    SIZE=10,
    MAXSIZE=100,
    FILEGROWTH=15%
)

GO

--VERIFICANDO LOS ARCHIVOS CREADOS
SP_HELPDB COMERCIAL2017
GO
```

Caso 3: Eliminar el archivo secundario COMERCIAL_SEC2 de la base de datos COMERCIAL2017

```
ALTER DATABASE COMERCIAL2017  
REMOVE FILE COMERCIAL_SEC2  
GO
```

```
--VERIFICANDO LOS ARCHIVOS  
SP_HELPDB COMERCIAL2017  
GO
```

1.2.2.5 Eliminación

A continuación, utilizaremos el comando **DROP**.

Caso 1: Eliminar la base de datos BD_EJEMPLO

```
DROP DATABASE BD_FARMACIA  
GO
```

Caso 2: Eliminar la base de datos BD_COMERCIAL validando la existencia del mismo.

```
USE MASTER  
GO  
IF DB_ID('BD_COMERCIAL') IS NOT NULL  
DROP DATABASE BD_COMERCIAL  
GO
```

Ejercicios Integrador

CASO DESARROLLADO: MAQUINARIAS

Cree la base de datos **BD_MAQUINARIAS** en la carpeta C:\MAQUINARIAS\DATA\ con la siguiente configuración:

Archivo de datos: Un tamaño inicial de 10MB, máximo de 100MB y un factor de crecimiento de 10%.

Archivo secundario: Un tamaño inicial de 5MB, máximo de 50MB y un factor de crecimiento de 10%.

Archivo de transacciones: Un tamaño inicial de 5MB, máximo de 40MB y un factor de crecimiento de 10%.

El administrador de la base de datos realiza las siguientes modificaciones:

Al archivo de transacciones:

Ampliar el tamaño inicial del archivo de transacciones a 10MB.

Modificar la tasa de crecimiento en un 15%.

Agregar archivos:

Agregar tres archivos secundarios llamados BD_MAQUINARIAS1, BD_MAQUINARIAS2 y BD_MAQUINARIAS3 de tamaño inicial 5MB y

máximo tamaño 20MB, los cuales estarán ubicados en C:\MAQUINARIAS\SEC.

Eliminar archivos:

Eliminar el archivo BD_MAQUINARIAS2.

Solución:

```
USE MASTER
GO
IF DB_ID('BD_MAQUINARIAS') IS NOT
NULL DROP DATABASE BD_MAQUINARIAS GO
CREATE DATABASE BD_MAQUINARIAS
ON PRIMARY (
    NAME='BD_MAQUINARIAS_PRI',
    FILENAME='C:\MAQUINARIAS\DATA\BD_MAQUINARIAS.MDF',
    SIZE=10,
    MAXSIZE=100,
    FILEGROWTH=10%),
(
    NAME='BD_MAQUINARIAS_SEC',
    FILENAME='C:\MAQUINARIAS\DATA\BD_MAQUINARIAS.NDF',
    SIZE=5,
    MAXSIZE=50,
    FILEGROWTH=10%
)
LOG ON (
    NAME='BD_MAQUINARIAS_TRA',
    FILENAME='C:\MAQUINARIAS\DATA\BD_MAQUINARIAS.LDF',
    SIZE=5,
    MAXSIZE=40,
    FILEGROWTH=10%
)
GO
--Verificando los archivos creados en la base de datos BD_MAQUINARIAS
SP_HELPDB BD_MAQUINARIAS
GO
/* A. Al archivo de transacciones:
   Ampliar el tamaño inicial del archivo de transacciones a 10MB.
   Modificar la tasa de crecimiento en un 15%. */
ALTER DATABASE BD_MAQUINARIAS
    MODIFY FILE (NAME='BD_MAQUINARIAS_TRA', SIZE=10, FILEGROWTH=15%)
GO
/* B. Agregar archivos:
Aregar tres archivos secundarios llamados BD_MAQUINARIAS1, BD_MAQUINARIAS2
y BD_MAQUINARIAS3 de tamaño inicial 5MB y máximo tamaño 20MB, los cuales
estarán ubicados en C:\MAQUINARIAS\SEC. */
```

```
ALTER DATABASE BD_MAQUINARIAS
ADD FILE (
    NAME='BD_MAQUINARIAS1',
    FILENAME='C:\MAQUINARIAS\SEC\BD_MAQUINARIAS1.NDF',
    SIZE=5, MAXSIZE=20, FILEGROWTH=10%)
GO
ALTER DATABASE BD_MAQUINARIAS
ADD FILE (
    NAME='BD_MAQUINARIAS2',
    FILENAME='C:\MAQUINARIAS\SEC\BD_MAQUINARIAS2.NDF',
    SIZE=5,
    MAXSIZE=20,
    FILEGROWTH=10%
)
GO
ALTER DATABASE BD_MAQUINARIAS
ADD FILE (
    NAME='BD_MAQUINARIAS3',
    FILENAME='C:\MAQUINARIAS\SEC\BD_MAQUINARIAS3.NDF',
    SIZE=5, MAXSIZE=20,
    FILEGROWTH=10%
)
GO

/* C. Eliminar archivos:
   Eliminar el archivo BD_MAQUINARIAS2. */

ALTER DATABASE BD_MAQUINARIAS
REMOVE FILE BD_MAQUINARIAS2
GO
```

Nota: El siguiente caso desarrollan el Docente con los estudiantes.

CASO PROPUESTO 01: INSTITUTO

Cree la base de datos **BD_INSTITUTO** en la carpeta C:\INSTITUTO\ con la siguiente configuración:

Archivo de datos: Ubicado en C:\INSTITUTO\DATOS\, un tamaño inicial de 10MB, máximo de 200MB y un factor de crecimiento de 3MB.

Archivo de transacciones: Ubicado en C:\INSTITUTO\TRANSACCIONES\, un tamaño inicial de 2MB, máximo de 80MB y un factor de crecimiento de 20%.

El administrador de la base de datos realiza las siguientes modificaciones:

Agregar archivos:

Agregar dos archivos secundarios llamados BD_INSTITUTO_SEC1 y BD_INSTITUTO_SEC2 de tamaño inicial 2MB y máximo tamaño permitido por el sistema para ambos archivos, los cuales estarán ubicados en C:\INSTITUTO\SECUNDARIO.

Al archivo de transacciones aplique los siguientes cambios:

Ampliar el tamaño inicial del archivo a 20MB.

Ampliar el máximo tamaño a 150MB.

Modificar la tasa de crecimiento en un 15%.

C. Eliminar archivos:

Eliminar el archivo BD_INSTITUTO_SEC2.

Nota: Desarrollan solo los estudiantes para comprobar el logro de su aprendizaje.

CASO PROPUESTO 02: HOSPITAL

Cree la base de datos **BD_HOSPITAL** en la carpeta C:\HOSPITAL con la siguiente configuración:

Archivo de datos: Un tamaño inicial de 15MB, máximo de 150MB y un factor de crecimiento de 10%.

Archivo de Transacciones: Un tamaño inicial de 10MB, máximo de 100MB y un factor de crecimiento de 10%.

El administrador de la base de datos realiza las siguientes modificaciones:

Al archivo de transacciones:

Ampliar el tamaño inicial del archivo de transacciones a 15MB.

Modificar la tasa de crecimiento en un 15%.

Agregar archivos:

Agregar tres archivos secundarios llamados BD_HOSPITAL_SEC1, BD_HOSPITAL_SEC2 y BD_HOSPITAL_SEC3 de tamaño inicial 10MB y máximo tamaño 100MB, los cuales estarán ubicados en C:\HOSPITAL\SECUNDARIOS.

Eliminar archivos:

Eliminar el archivo BD_HOSPITAL_SEC1.

CASO PROPUESTO 03: FERRETERIA

Cree la base de datos **BD_FERRETERIA** en la carpeta C:\FERRETERIA con la siguiente configuración:

Archivo de datos: Un tamaño inicial de 10MB, máximo de 200MB y un factor de crecimiento de 15%. Debe estar ubicado en C:\FERRETERIA\DATOS

Archivo de datos secundario: Un tamaño inicial de 7MB, máximo de 70MB y un factor de crecimiento de 7%. Debe estar ubicado en C:\FERRETERIA\SECUNDARIOS

Archivo de Transacciones: Un tamaño inicial de 8MB, máximo de 80MB y un factor de crecimiento de 8%. Debe estar ubicado en C:\FERRETERIA\TRANS

El administrador de la base de datos realiza las siguientes modificaciones:

Al archivo de transacciones:

Ampliar el tamaño inicial del archivo de transacciones a 25MB.

Modificar la tasa de crecimiento en un 17%.

Agregar archivos:

Agregar 4 archivos secundarios llamados BD_FERRETERIA_SEC2, BD_FERRETERIA_SEC3, BD_FERRETERIA_SEC4 y BD_FERRETERIA_SEC5 de tamaño inicial 10MB y máximo tamaño

100MB, los cuales estarán ubicados en C:\FERRETERIA\SECUNDARIOS.

Eliminar archivos:

Eliminar el archivo BD_FERRETERIA_SEC4 y BD_FERRETERIA_SEC5.

Autoevaluación

Usando TRANSACT/SQL, cree las siguientes bases de datos:

- 1.1. **VENTAS1**, en la carpeta C:\DATABASE\, considerando que el archivo principal tiene tamaño de 20 megabytes, un tamaño máximo de 80 megabytes y un incremento de 10 megabytes.
- 1.2. **VENTAS2**, en la carpeta C:\MSSQL\DATA\, considerando que el archivo principal tiene tamaño de 20 megabytes, un tamaño máximo de 80 megabytes y un incremento de 10 megabytes. Por otro lado, considere que el archivo de transacciones tiene tamaño de 3 megabytes, un tamaño máximo de 13 megabytes y un incremento del 15%.
- 1.3. **VENTAS3**, en la carpeta C:\CIBERMARANATA\DATOS\, con la siguiente configuración:
 - Archivo de datos:** Un tamaño inicial de 20 megabytes, máximo de 120 megabytes y un factor de crecimiento de 5%,
 - Archivo secundario:** Un tamaño inicial de 10 megabytes, máximo de 50 megabytes y un factor de crecimiento de 2 megabytes,
 - Archivo de transacciones:** un tamaño inicial de 8 megabytes, máximo de 75 megabytes y un factor de crecimiento de 2%.
- 1.4. Cree la base de datos **BD_COLEGIO** en la carpeta D:\COLEGIO\DATA\ con la siguiente configuración:
 - Archivo de datos:** Un tamaño inicial de 80MB, máximo de 120MB y un factor de crecimiento de 10%.
 - Archivo secundario:** Un tamaño inicial de 10MB, máximo de 100MB y un factor de crecimiento de 5MB.
- 1.5. La “**AGENCIAPERU**” tiene por misión tener un control de las estadías de los turistas que visitan nuestro país para esto tiene agencias en todo el Perú, por lo tanto, necesita automatizar sus procesos actuales. Cree la base de datos “AGENCIAPERU”, en la carpeta D:\AGENCIAPERU\DATA\, considerando que:
 - Archivo de datos:** Un tamaño inicial de 50 MegaBytes y máximo de 200 MegaBytes. Habilite la opción de incremento en 12%.
 - Archivo de transacciones:** Un tamaño inicial de 10 MegaBytes y máximo de 25 MegaBytes. Habilite la opción de incremento en 12%.

1.3 Tipos de datos

En **SQL Server**, a una columna de una tabla, una variable local o un parámetro tiene un tipo de datos asociado. Un tipo de datos es una característica que especifica el tipo de valor que el objeto puede contener por ejemplo un dato numérico, cadena de caracteres, valores monetarios, fechas u horas, etc.

1.3.1 Identificación de los tipos de datos empleados en SQL Server 2014.

Categorías de tipos de datos

Los tipos de datos en **SQL** se agrupan en las siguientes categorías:

Categorías	Tipos de datos	Breve definición
Numéricos exactos	bigint	-2^63 (-9.223.372.036.854.775.808) a 2^63-1
	numeric	(9.223.372.036.854.775.807)
	bit	-10^38 +1 y 10^38 - 1
	smallint	1,0
	decimal	-2^15 (-32.768) a 2^15-1 (32.767)
	smallmoney	-10^38 +1 y 10^38 - 1
	int	-214.748.3648 a 214.748.3647
	tinyint	-2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647)
	money	0 a 255
Cadenas de caracteres Unicode	nchar	Datos de cadena Unicode de longitud fija, valor 1 y 4.000.
	nvarchar	Datos de cadena Unicode de longitud variable, valor 1 y 4.000.
	ntext	2^30 - 1 (1.073.741.823) bytes.
Numéricos aproximados	float	1,79E+308 a -2,23E-308, 0 y de 2,23E-308 a 1,79E+308
	real	- 3,40E + 38 a -1,18E - 38, 0 y de 1,18E - 38 a 3,40E + 38
Cadenas binarias	binary	Datos binarios de longitud fija, su valor oscila entre 1 y 8.000.
	varbinary	Datos binarios de longitud variable, su valor es de 1 a 8.000.
	image	0 hasta 2^31-1 (2.147.483.647) bytes.
Fecha y hora	date	0001-01-01 a 9999-12-31
	datetimeoffset	0001-01-01 a 9999-12-31
	datetime2	0001-01-01 a 9999-12-31 -De 00:00:00 a 23:59:59.999999
	smalldatetime	1900-01-01 a 2079-06-06 - De 00:00:00 a 23:59:59
	datetime	01/01/1753 hasta 31/12/9999 00:00:00 a 23:59:59.997
	time	00:00:00.000000 a 23:59:59.999999
Cadenas de caracteres	Char	Datos de cadena no Unicode de longitud fija su valor se encuentra entre 1 y 8.000.
	varchar	Datos de cadena no Unicode de longitud variable su valor se encuentra entre 1 y 8.000.
	text	2.147.483.647 bytes.

1.4. Tabla de datos

Una **tabla** es una colección de datos sobre una **entidad** (Persona, Lugar, Cosa) específica, que tiene un número discreto de atributos designados (por ejemplo, cantidad o tipo). Las tablas son los objetos principales de SQL Server y del modelo relacional en general. Las tablas son fáciles de entender, ya que son prácticamente iguales a las listas que utiliza de manera cotidiana.

En SQL Server una tabla suele denominarse tabla de base, para hacer énfasis sobre dónde se almacenan los datos. La utilización de <>Tabla de base>>, también distingue la tabla de una vista (View), (una tabla virtual que es una consulta interna de una tabla

base.) Conforme se utiliza la base de datos con frecuencia se encontrará conveniente definir tablas propias para almacenar datos personales o datos extraídos de otras tablas. Los atributos de los datos de una tabla (tamaño, color, cantidad, fecha, etc.) toman la forma de columnas con nombre en la tabla.

```
CREATE TABLE PRODUCTOS
(
    CODIGO           CHAR (5) PRIMARY KEY NOT NULL,
    DESCRIPCION      VARCHAR (50)          NOT NULL,
    PRECIO_VENTA     MONEY                NOT NULL,
    STOCK_ACTUAL     INT                  NOT NULL,
    FECHA_VENCI     DATE                NOT NULL
)
GO
```

Figura 8: Tabla de datos
Fuente. - Tomado de SQL Server 2014

1.4.1. Creación de una tabla de datos CREATE

La sentencia **CREATE** dentro de SQL Server permite crear todo tipo de objeto desde una tabla, procedimiento almacenado, funciones, triggers, etc. Veamos el formato BÁSICO de la sentencia Create para una tabla o entidad:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1   TIPO,
    CAMPO1   TIPO,
    CAMPO1   TIPO
)
GO
```

Donde:

NOMBRE_TABLA: Es el nombre de la tabla a crear, debemos considerar que dicho nombre cuenta con las mismas reglas que se aplican a las variables de un lenguaje de programación. **Por ejemplo:**

EMPLEADO, TB_EMPLEADO

CAMPO1: Es la especificación de las columnas que cuenta la tabla, así mismo son llamados columnas o atributos. Así, por ejemplo:

CÓDIGO, CODIGO_EMP, COD_EMP

TIPO: Es la especificación del tipo de datos según el contenido del campo de una tabla; estas podrían ser:

CHAR, VARCHAR, INT, MONEY, DATE, ETC.

Al formato básico podemos agregarle cláusulas que permiten optimizar de la mejor manera la creación de la tabla, podemos mencionar:

Asignación de valores nulos y no nulos a las columnas:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NULL | NOT NULL,
    CAMPO2      TIPO NULL | NOT NULL,
    CAMPO3      TIPO NULL | NOT NULL
)
GO
```

- ✓ Asignación de llave primaria:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NOT NULL PRIMARY KEY,
    CAMPO2      TIPO NULL | NOT NULL,
    CAMPO3      TIPO NULL | NOT NULL
)
GO
```

- ✓ Asignación de llave compuesta:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NOT NULL,
    CAMPO2      TIPO NOT NULL,
    CAMPO3      TIPO NULL | NOT NULL,
    PRIMARY KEY (CAMPO1, CAMPO2)
)
GO
```

- ✓ Asignación de llave FORÁNEA:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NOT NULL PRIMARY KEY,
    CAMPO2      TIPO NULL | NOT NULL,
    CAMPO3      TIPO NULL REFERENCES NOMBRE_TABLA2
)
GO
```

- ✓ Asignación de valores por defecto:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NULL | NOT NULL,
    CAMPO2      TIPO DEFAULT VALOR_POR_DEFECTO,
    CAMPO3      TIPO NULL | NOT NULL
)
GO
```

- ✓ Asignación de restricciones:

```
CREATE TABLE NOMBRE_TABLA (
    CAMPO1      TIPO NULL | NOT NULL,
    CAMPO2      TIPO CHECK (CAMPO=VALOR),
    CAMPO3      TIPO NULL | NOT NULL
)
GO
```

Y finalmente, mencionaremos la validación de una tabla; se creará la tabla validando su existencia. Si la tabla ya existe la elimina y la crea, caso contrario solo la crea. Para eso utilizamos la sentencia If.

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

CREATE TABLE TABLA (
    CAMPO1      CHAR      (3)      NOT NULL      PRIMARY KEY,
    CAMPO2      VARCHAR   (30)     NOT NULL
)
GO
```

Todo lo que se puede crear dentro de una base de datos es considerado como objeto; es por eso que la sentencia If lo reconoce con la función **OBJECT_ID**, esta función identifica el código de objeto que le fue asignado al momento de su creación.

1.4.2. Modificación de una tabla de datos ALTER

La sentencia **ALTER** dentro de SQL Server permite modificar el contenido de un determinad objeto de base de datos, esta función puede modificar la estructura de una tabla, procedimiento almacenado, funciones, triggers, etc. Su formato para la modificación de una tabla de datos es:

```
ALTER TABLE NOMBRE_TABLA
    FUNCION ESPECIFICACIÓN
GO
```

Veamos las posibles acciones que podemos tomar con la sentencia Alter Table:

Agregar una columna a la tabla:

```
ALTER TABLE NOMBRE_TABLA
    ADD COLUMNANUEVA TIPO NULL | NOT NULL
)
GO
```

Agregar varias columnas a la tabla:

```
ALTER TABLE NOMBRE_TABLA
    ADD COLUMNANUEVA1    TIPO      NULL | NOT NULL,
    COLUMNANUEVA2    TIPO      NULL | NOT NULL
GO
```

Agregar una llave primaria a la tabla:

```
ALTER TABLE NOMBRE_TABLA
    ADD PRIMARY KEY (COLUMNAS)
GO
```

Agregar llave compuesta a la tabla:

```
ALTER TABLE NOMBRE_TABLA  
    ADD PRIMARY KEY (COLUMNAS1, COLUMNAS2)  
GO
```

Agregar una llave foránea a la tabla:

```
ALTER TABLE NOMBRE_TABLA1  
    ADD FOREIGN KEY (COLUMNAS) REFERENCES NOMBRE_TABLA2  
GO
```

Eliminar una columna de la tabla:

```
ALTER TABLE NOMBRE_TABLA  
    DROP COLUMN COLUMNAS  
)  
GO
```

Modificar una columna a la tabla:

```
ALTER TABLE NOMBRE_TABLA  
    ALTER COLUMN COLUMNAS TIPO  
)  
GO
```

1.4.3. Eliminación de una tabla de datos DROP

La sentencia DROP dentro de SQL Server permite eliminar cualquier tipo de objeto de una base de datos, tales como las tablas, procedimientos almacenados, funciones, triggers, etc. Su formato para la eliminación de una tabla de datos es:

```
DROP TABLE NOMBRE_TABLA  
GO
```

1.5. Restricciones parte I

El principio fundamental del modelo relacional, es que cada fila de una tabla es en cierta medida exclusiva y puede distinguirse de alguna forma de cualquier otra fila de la tabla. La combinación de todas las columnas de una tabla puede utilizarse como un identificador exclusivo, pero en la práctica el identificador suele ser mucho como la combinación de unas pocas columnas y, a menudo, es simplemente una columna, a la cual se le denomina **Primary Key** o **Clave Primaria**.

Así mismo, una **clave Foránea** o **Foreign Key** es una o varias columnas de una tabla cuyos valores deben ser iguales a los de una restricción Primary Key en otra tabla. SQL Server impone de manera automática la **integridad referencial** mediante la utilización de Foreign Key y a esta característica se le denomina integridad referencial declarativa.

a) Definición de relaciones

El término "**relaciones**" usualmente se refiere a las relaciones entre claves foráneas y primarias, y entre tablas. Estas relaciones deben ser definidas porque determinan qué columnas son o no claves primarias o claves foráneas. A continuación, veamos los tipos de relación que pueden existir entre las tablas:

b) Relación Uno-a-Varios:

La relación uno a varios (uno a muchos), es el tipo de relación más común. Se podría decir que:

En este tipo de relación, una fila de la **tabla A** puede tener varias columnas coincidentes en la **tabla B**.

Pero una fila de la **tabla B** sólo puede tener una fila coincidente en la tabla A.

Por ejemplo, las tablas **Editor** y **Libro** tienen una relación uno a varios, por la siguiente razón “**Cada editor produce muchos libros, pero cada Libro procede de un único editor**”.

Una relación de uno a varios sólo se crea si una de las columnas relacionadas es una clave principal o tiene una restricción única (una restricción única impide que el campo tenga valores repetidos). El lado de la clave principal de una relación de uno a varios se indica con un símbolo de llave, mientras que el lado de la clave externa de una relación se indica con un símbolo de infinito en SQL Server.

Veamos el caso más común entre dos tablas o entidades implementada en SQL Server:

“Una misma categoría podría estar asignada a muchos empleados, pero un empleado tiene únicamente una categoría”.

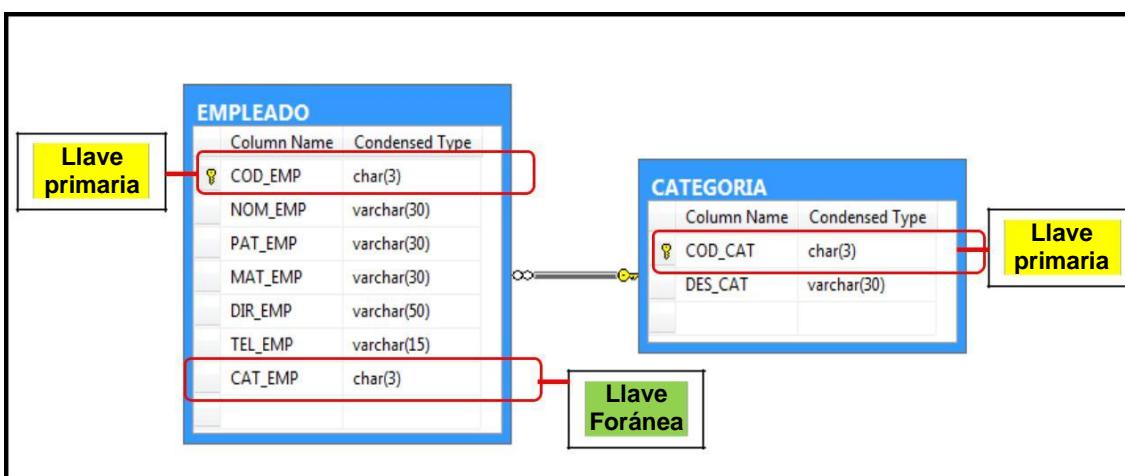


Figura 9: tipo de relación de uno a varios

Fuente. - Tomado desde SQL Server 2014

También podemos observar una relación de uno a muchos en una recursividad:

“Un proveedor puede ser la extensión de otro proveedor y un proveedor puede tener muchas extensiones”.

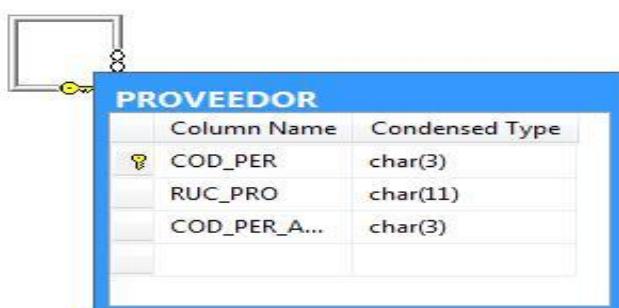


Figura 10: Representación de la recursividad de la tabla Proveedor en SQL Server

Fuente. - Tomado desde SQL Server 2014

c) Relaciones de varios a varios

En las relaciones de varios a varios (muchos a muchos), una fila de la tabla A puede tener varias filas coincidentes en la tabla B, y viceversa.

Para crear una relación de este tipo, debemos tener en cuenta los siguientes aspectos:
 Definir una tercera tabla denominada tabla de unión
 Asigne las claves principales las cuales están formadas por las claves externas de las tablas A y B.

Por ejemplo, la tabla **Autor** y la tabla **Libro** tienen una relación de varios a varios definida por una relación de uno a varios entre cada de estas tablas y la tabla **Autor_Libro**. La clave principal de la tabla **Autor_Libro** es la combinación de la columna **cod_aut** (la clave principal de la tabla **Autor**) y la columna **cod_lib** (la clave principal de la tabla **Libro**).

Un empleado puede pertenecer a muchos departamentos y en un departamento pueden estar registrados muchos empleados. Para poder implementar esta relación compleja debemos adicionar la tabla contrato.

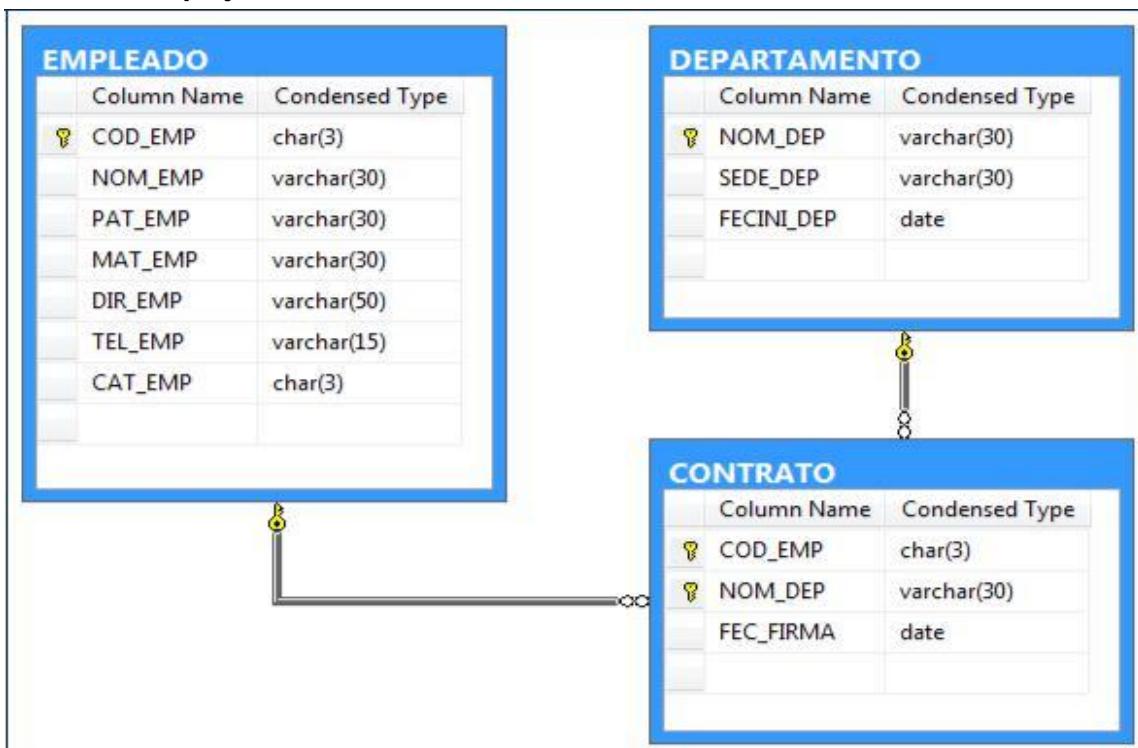


Figura 11: Representación de muchos a muchos en SQL

Fuente. - Tomado desde SQL Server 2014

d) Relaciones de uno a uno

En una relación de uno a uno, implica que una fila de la **tabla A** no puede tener más de una fila coincidente en la **tabla B** y viceversa. Una relación de uno a uno se crea si las dos columnas relacionadas son claves principales o tienen restricciones únicas.

Así mismo, podemos decir que este tipo de relación no es común; porque la mayor parte de la información relacionada de esta manera estaría en una tabla. Se puede utilizar una relación de uno a uno para:

Dividir una tabla con muchas columnas.

Aislarse parte de una tabla por razones de seguridad.

Almacenar datos que no se deseen conservar y se puedan eliminar fácilmente con tan sólo suprimir la tabla.

Almacenar información aplicable únicamente a un subconjunto de la tabla principal.

Implementar entidades del tipo **Generalización** con sus especializaciones.

El lado de la clave principal de una relación de uno a uno se indica con un símbolo de **llave**. El lado de la clave externa también se indica con un símbolo de **llave**.

El ejemplo, a continuación, muestra a la tabla PERSONA (generalización) relacionándose con la tabla CLIENTE (especialización 1) y la tabla PROVEEDOR (especialización 2), de uno a uno.

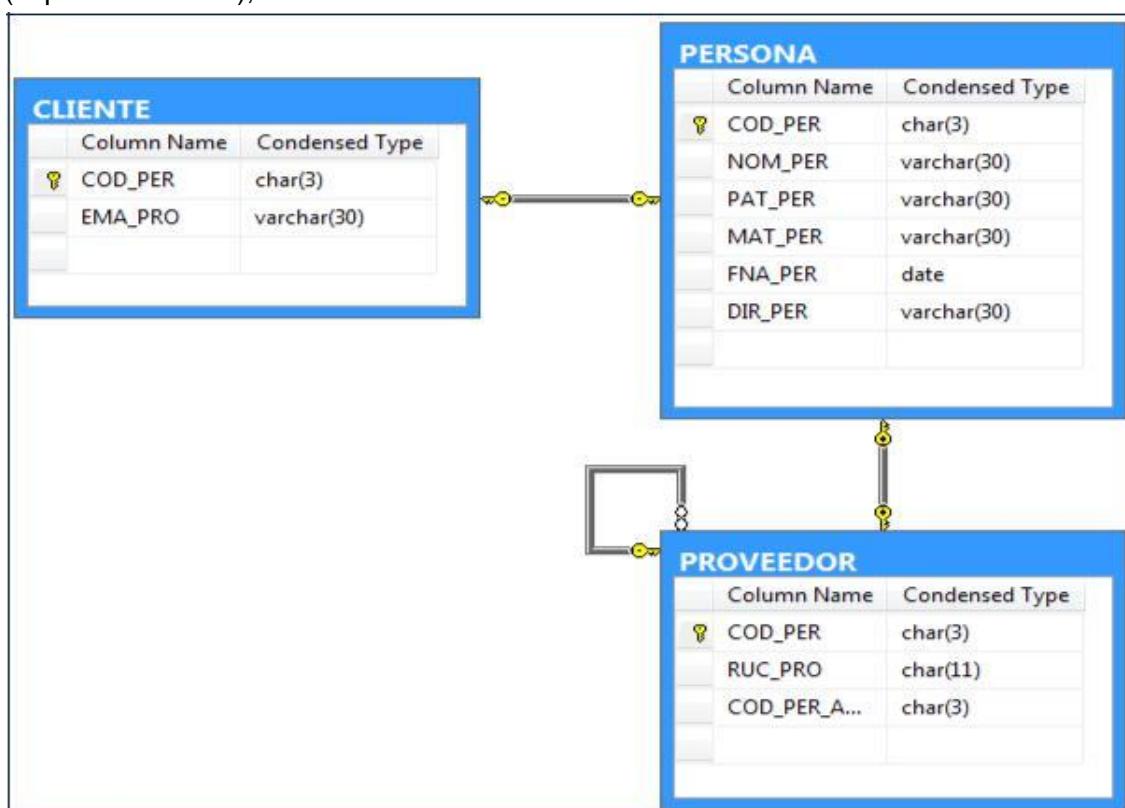


Figura 12: Representación uno a uno en SQL Server

Fuente. - Tomado desde SQL Server 2014

1.5.1. Asignación de PRIMARY KEY a la tabla de datos



Una tabla o entidad suele tener un campo o una combinación de campos cuyos valores identifican de forma única cada fila de la tabla. A estas columnas se les denomina llave o simplemente clave principal de la tabla.

Debemos tener en cuenta:

Toda tabla de una base de datos debe tener por lo menos una asignación **PRIMARY KEY**.

Ningún campo a la cual se aplique **PRIMARY KEY** debe aceptar valores nulos, eso quiere decir, que deben estar asignados como **NOT NULL**.

Toda asignación **PRIMARY KEY** garantiza datos únicos; es decir no se podrá ingresar dos valores similares, estos son usados mayormente para códigos.

Su formato es:

Asignar Primary Key al crear la tabla

```
CREATE TABLE NOMBRE_TABLA (
    COLUMNA1  TIPO NOT NULL PRIMARY KEY,
    COLUMNA2  TIPO NULL,
    COLUMNA3  TIPO NOT NULL
)
GO
```

Asignar varias Primary Key al crear la tabla

```
CREATE TABLE NOMBRE_TABLA (
    COLUMNA1  TIPO NOT NULL,
    COLUMNA2  TIPO NULL,
    COLUMNA3  TIPO NOT NULL,
    PRIMARY KEY (COLUMNA1, COLUMNA2)
)
GO
```

Asignar Primary Key modificando la tabla

```
ALTER TABLE NOMBRE_TABLA
    ADD PRIMARY KEY (COLUMNA1)
GO
```

Asignar varias Primary Key modificando la tabla

```
ALTER TABLE NOMBRE_TABLA
    ADD PRIMARY KEY (COLUMNA1, COLUMNA2)
GO
```

1.5.2. Asignación de FOREIGN KEY a la tabla de datos

También es llamada clave externa o llave foránea, es una columna o combinación de columnas que se utiliza para establecer y exigir un vínculo entre dos tablas.

En una referencia de clave externa, se crea un vínculo entre dos tablas cuando las columnas de una de ellas hacen referencia a las columnas de la otra que contienen el valor de clave principal. Esta columna se convierte en una clave externa para la segunda tabla.

Debemos tener en cuenta:

Una asignación **Foreign Key** desarrolla la relación uno a muchos entre las entidades.

Una tabla puede tener **N** enlaces foráneos, siempre y cuando la otra tabla tenga un campo clave que lo asocie.

Para poder enlazar dos tablas se necesita que una de las columnas sea clave principal y la otra sea una columna simple, pero que tengan el mismo tipo de datos y principalmente la misma capacidad.

Toda asignación **FOREING KEY** garantiza que los datos ingresados en una tabla se encuentren asociada a otra; así se genera una dependencia entre los datos de ambas tablas.

Su formato es:

Asignar Foreign Key a las columnas al crear la tabla

```
CREATE TABLE NOMBRE_TABLA1(
    COLUMNNA1  TIPO NOT NULL PRIMARY KEY,
    COLUMNNA2  TIPO NULL,
    COLUMNNA3  TIPO NOT NULL FOREIGN KEY REFERENCES
NOMBRE_TABLA2
)
GO
```

Asignar Foreign Key al finalizar la especificación de las columnas

```
CREATE TABLE NOMBRE_TABLA1(
    COLUMNNA1  TIPO NOT NULL PRIMARY KEY,
    COLUMNNA2  TIPO NULL,
    COLUMNNA3  TIPO NOT NULL,
    FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA2(COLUMNNA),
    FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA3(COLUMNNA)
)
GO
```

Asignar Foreign Key modificando la tabla

```
ALTER TABLE NOMBRE_TABLA1
    ADD FOREIGN KEY (COLUMNNA1) REFERENCES NOMBRE_TABLA2
GO
```

Asignar varios Foreign Key modificando la tabla

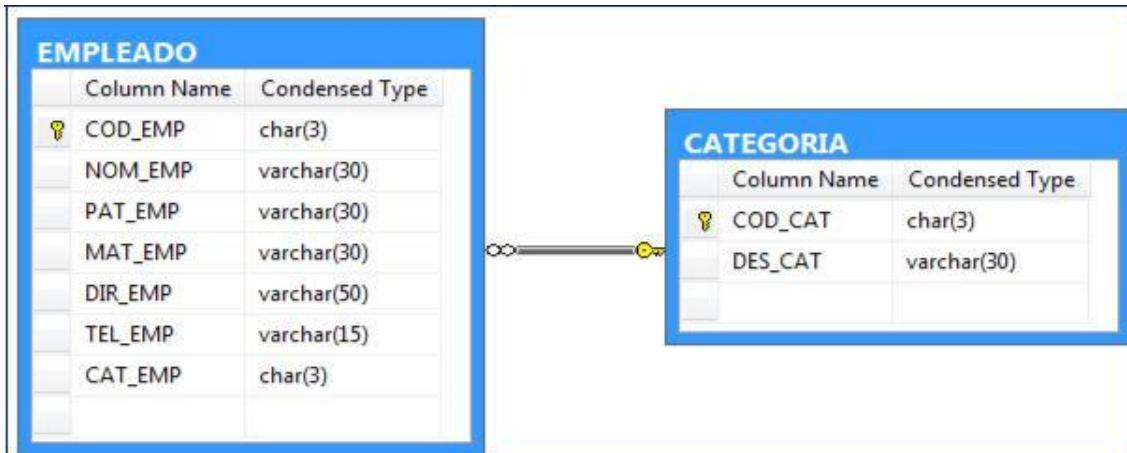
```
ALTER TABLE NOMBRE_TABLA1
    ADD FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA2,
    FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA3,
    FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA4
GO
```

Asignar Foreign Key para una recursividad

```
ALTER TABLE NOMBRE_TABLA
    ADD FOREIGN KEY (COLUMNNA) REFERENCES NOMBRE_TABLA
GO
```

Ejemplo:

Si tenemos el siguiente diagrama, mostraremos las diferentes formas de asignar llaves foráneas entre dos tablas:



Debemos asumir que la tabla **Categoría** ya se encuentra creada, tenemos:

Primera forma: Asignando las llaves foráneas en cada columna de la tabla.

```
CREATE TABLE EMPLEADO (
    COD_EMP      CHAR      (3) NOT NULL PRIMARY KEY,
    NOM_EMP      VARCHAR   (30) NOT NULL,
    PAT_EMP      VARCHAR   (30) NOT NULL,
    MAT_EMP      VARCHAR   (30) NOT NULL,
    DIR_EMP      VARCHAR   (50) NOT NULL,
    TEL_EMP      VARCHAR   (15) NULL,
    CAT_EMP      CHAR      (3) NOT NULL FOREIGN KEY REFERENCES
    CATEGORIA
)
GO
```

Segunda forma: Asignando las llaves foráneas al final de la definición de las columnas de la tabla.

```
CREATE TABLE EMPLEADO (
    COD_EMP      CHAR      (3) NOT NULL PRIMARY KEY,
    NOM_EMP      VARCHAR   (30) NOT NULL,
    PAT_EMP      VARCHAR   (30) NOT NULL,
    MAT_EMP      VARCHAR   (30) NOT NULL,
    DIR_EMP      VARCHAR   (50) NOT NULL,
    TEL_EMP      VARCHAR   (15) NULL,
    CAT_EMP      CHAR      (3) NOT NULL,
    FOREIGN KEY (CAT_EMP) REFERENCES CATEGORIA(COD_CAT)
)
GO
```

Tercera forma: Asumimos que la tabla **Empleado** ya se encuentra creada sin especificar las llaves foráneas.

```
ALTER TABLE EMPLEADO
ADD FOREIGN KEY (CAT_EMP) REFERENCES CATEGORIA
GO
```

Cuarta forma: Especificando el campo clave en la segunda tabla.

```
ALTER TABLE EMPLEADO
ADD FOREIGN KEY (CAT_EMP) REFERENCES CATEGORIA(COD_CAT)
GO
```

1.5.3 Implementación de una base de datos en SQL Server 2014

1. Caso desarrollado: MINIMARKET

Implementaremos un script de SQL Server que permita crear la base de datos **BD_MINIMARKET**. Para ello se cuenta con el siguiente diagrama:

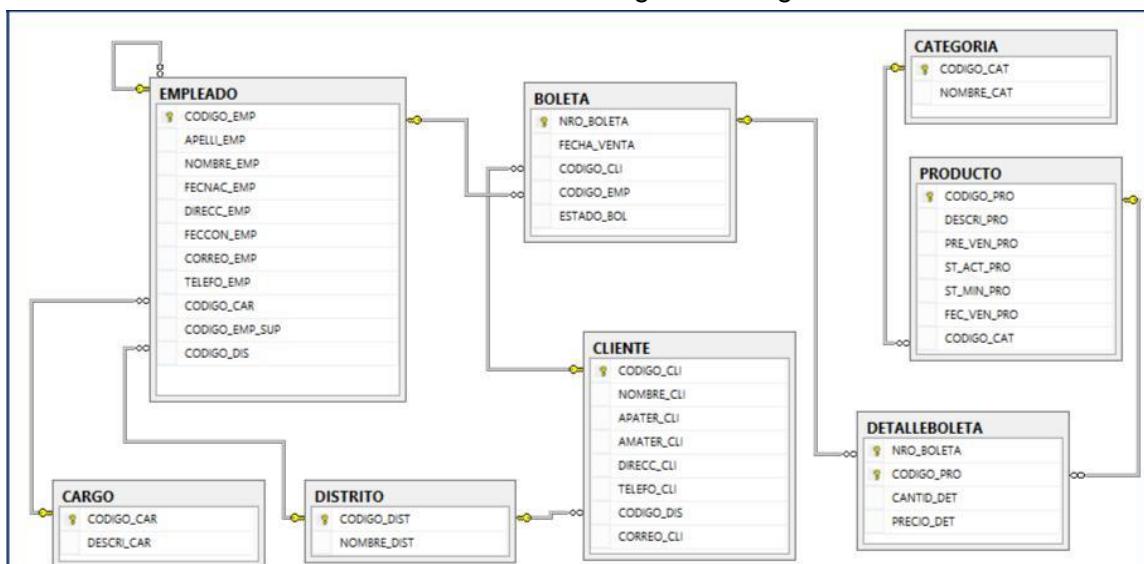


Figura 13: Diagrama de base de datos "Minimarket"

Fuente. - Tomado desde SQL Server 2014

Usando **TRANSACT/SQL**, realice lo siguiente:

Cree la base de datos **BD_MINIMARKET** de manera estándar

Active la base de datos **BD_MINIMARKET**

Cree las tablas mostradas

Agregue las **llaves Primarias** (ADD PRIMARY KEY)

Agregue las **llaves Foráneas, Relaciones** (ADD FOREIGN KEY – REFERENCES)

Solución:

```
USE MASTER
GO

--DEFINIENDO EL FORMATO DE LA FECHA DÍA, MES AÑO
SET DATEFORMAT DMY
GO
```

```
--VERIFICANDO LA EXISTENCIA DE LA BASE
IF DB_ID('BD_MINIMARKET') IS NOT NULL
BEGIN
DROP DATABASE BD_MINIMARKET
END
GO

--CREANDO LA BASE DE DATOS
CREATE DATABASE BD_MINIMARKET
GO

--ACTIVANDO LA BASE DE DATOS
USE BD_MINIMARKET
GO

--CREANDO LAS TABLAS
--TABLA DISTRITO

CREATE TABLE DISTRITO (
CODIGO_DIST      CHAR      (3) NOT NULL      PRIMARY KEY,
NOMBRE_DIST      VARCHAR   (50)
)
GO

--TABLA CATEGORIA
CREATE TABLE CATEGORIA (
CODIGO_CAT      CHAR      (3) NOT NULL      PRIMARY KEY,
NOMBRE_CAT      VARCHAR   (40)
)
GO

--TABLA PRODUCTO
CREATE TABLE PRODUCTO (
CODIGO_PRO      CHAR      (6) NOT NULL      PRIMARY KEY,
DESCRIP_PRO     VARCHAR   (50) NOT NULL,
PRE_VEN_PRO     MONEY      NOT NULL,
ST_ACT_PRO      INT       NOT NULL,
ST_MIN_PRO      INT       NOT NULL,
FEC_VEN_PRO    DATE      NOT NULL,
CODIGO_CAT      CHAR      (3) NOT NULL      REFERENCES CATEGORIA
)
GO

--TABLA CLIENTE
CREATE TABLE CLIENTE (
CODIGO_CLI      CHAR      (6)      NOT NULL PRIMARY KEY,
NOMBRE_CLI      VARCHAR   (30)     NOT NULL,
APATER_CLI     VARCHAR   (30)     NOT NULL,
```

```

AMATER_CLI      VARCHAR (30)    NOT NULL,
DIRECC_CLI     VARCHAR (50)    NULL,
TELEFO_CLI      VARCHAR (12)    NULL,
CODIGO_DIS      CHAR (3)      NOT NULL REFERENCES DISTRITO,
CORREO_CLI      VARCHAR (30)    NULL
)
GO
--TABLA CARGO
CREATE TABLE CARGO (
CODIGO_CAR      INT           NOT NULL PRIMARY KEY,
DESCRI_CAR      VARCHAR (30)   NOT NULL
)
GO
--TABLA EMPLEADO
CREATE TABLE EMPLEADO (
CODIGO_EMP      INT           NOT NULL PRIMARY KEY,
APELLI_EMP      VARCHAR (30)   NOT NULL,
NOMBRE_EMP      VARCHAR (30)   NOT NULL,
FECNAC_EMP      DATE          NOT NULL,
DIRECC_EMP      VARCHAR (60)   NOT NULL,
FECCON_EMP      DATE          NOT NULL,
CORREO_EMP      VARCHAR (35)   NULL,
TELEFO_EMP      VARCHAR (15)   NULL,
CODIGO_CAR      INT           REFERENCES CARGO,
CODIGO_EMP_SUP  INT           REFERENCES EMPLEADO,
CODIGO_DIS      CHAR (3)      REFERENCES DISTRITO
)
GO
--TABLA BOLETA
CREATE TABLE BOLETA (
NRO_BOLETA      CHAR (6)      NOT NULL PRIMARY KEY,
FECHA_VENTA    DATE          NOT NULL,
CODIGO_CLI      CHAR (6)      NOT NULL REFERENCES CLIENTE,
CODIGO_EMP      INT           NOT NULL REFERENCES EMPLEADO,
ESTADO_BOL      CHAR (2)
)
GO
--TABLA DETALLEBOLETA
CREATE TABLE DETALLEBOLETA (
NRO_BOLETA      CHAR (6)      NOT NULL REFERENCES BOLETA,
CODIGO_PRO      CHAR (6)      NOT NULL REFERENCES PRODUCTO,
CANTID_DET     INT           NOT NULL,
PRECIO_DET      MONEY
PRIMARY KEY (NRO_BOLETA, CODIGO_PRO)
)
GO

```

Diagrama Entidad Relación del caso: MINIMARKET

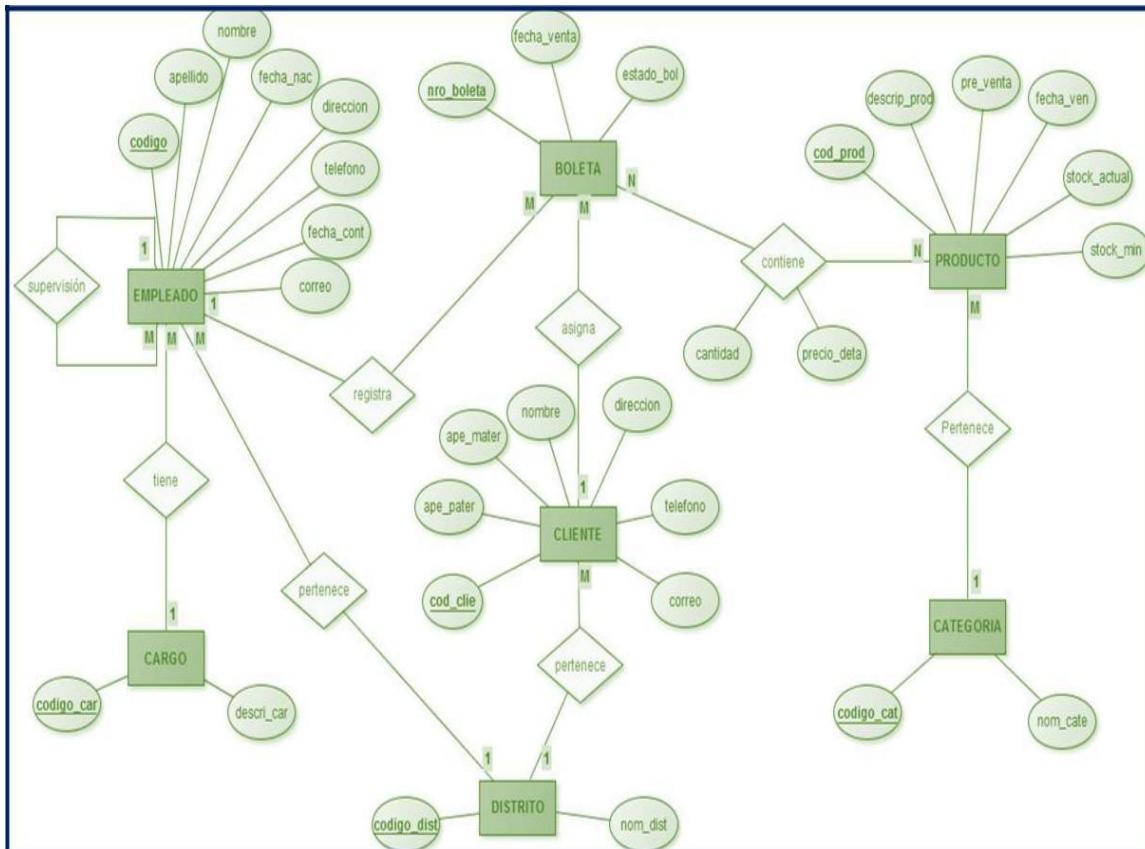


Figura 14: Diagrama entidad relacióndel caso “Minimarket”

Fuente. - Tomado desde yED

2. Caso propuesto: NEGOCIOS

Implementaremos un script de SQL Server que permita crear la base de datos **BD_NEGOCIOS**. Para ello se cuenta con el siguiente diagrama:

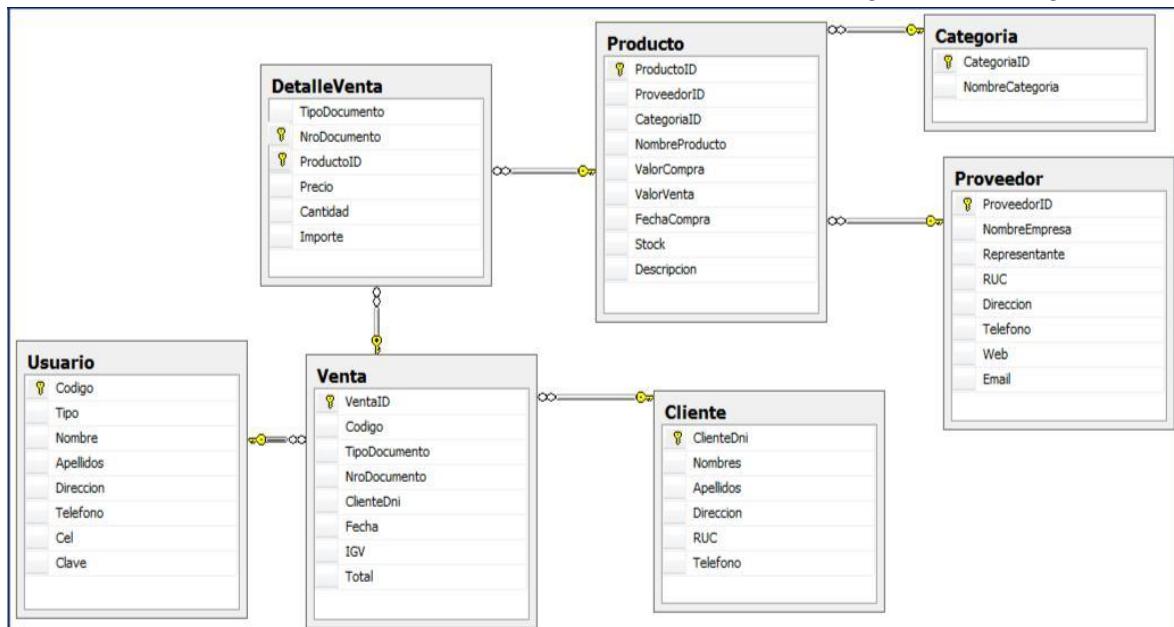


Figura 15: Diagrama de base de datos “Negocios”

Fuente. - Tomado desde SQL Server 2014

Usando **TRANSACT/SQL**, realice lo siguiente:

Cree la base de datos **BD_NEGOCIOS** de manera estándar

Active la base de datos **BD_NEGOCIOS**

Cree las tablas mostradas

Agregue las **llaves Primarias** (ADD PRIMARY KEY)

Agregue las **llaves Foráneas, Relaciones** (ADD FOREIGN KEY – REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

Asigne de manera correcta los valores nulos y no nulos según corresponda.

Visualice el diagrama implementado en SQL Server.

Caso propuesto: AUTO RENTA EIRL

Implementaremos un script de SQL Server que permita crear la base de datos **BD_RENTA**. Para ello se cuenta con el siguiente diagrama:

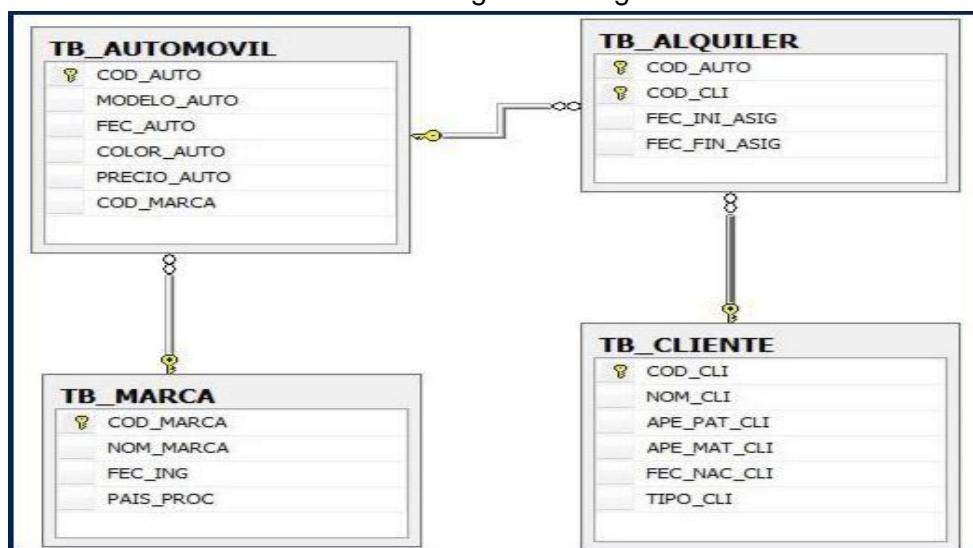


Figura 16: Diagrama de base de datos "Renta"

Fuente. - Tomado desde SQL Server 2014

Usando **TRANSACT/SQL**, realice lo siguiente:

Cree la base de datos **BD_RENTA** de manera estándar

Active la base de datos **BD_RENTA**

Cree las tablas mostradas

Agregue las **llaves Primarias** (ADD PRIMARY KEY)

Agregue las **llaves Foráneas, Relaciones** (ADD FOREIGN KEY REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

Asigne de manera correcta los valores nulos y no nulos según corresponda.
Visualice el diagrama implementado en SQL Server.

Caso propuesto: COLEGIO

Implementaremos un script de SQL Server que permita crear la base de datos **BD_COLEGIO**. Para ello se cuenta con el siguiente diagrama:

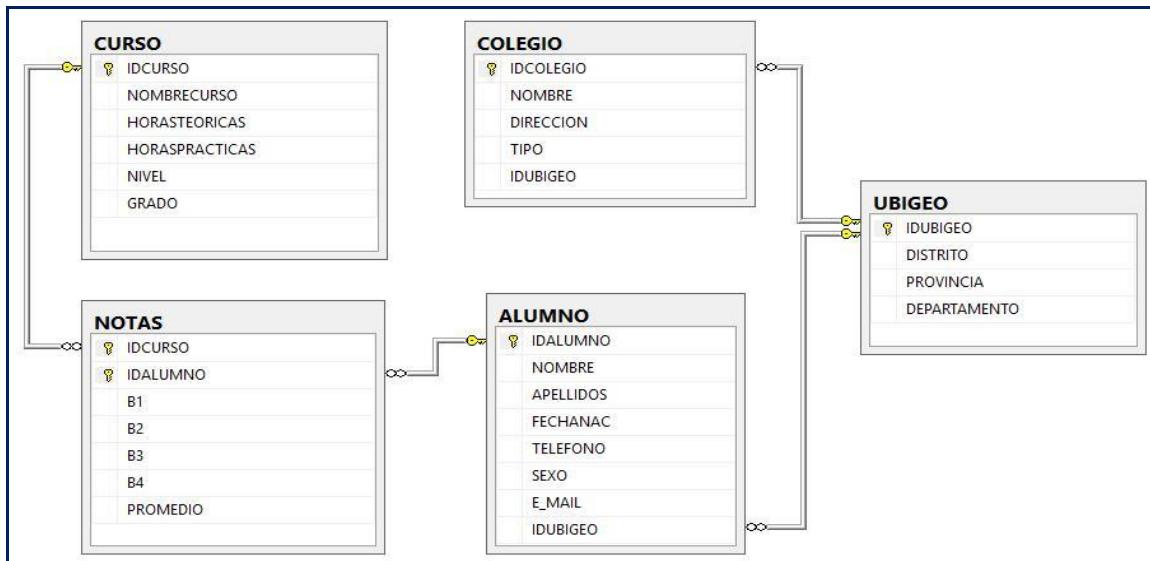


Figura 17: Diagrama de base de datos “Colegio”

Fuente. - Tomado desde SQL Server 2014

Usando **TRANSACT/SQl**, realice lo siguiente:

Cree la base de datos **BD_COLEGIO** de manera estándar

Active la base de datos **BD_COLEGIO**

Cree las tablas mostradas

Agregue las **llaves Primarias** (ADD PRIMARY KEY)

Agregue las **llaves Foráneas, Relaciones** (ADD FOREIGN KEY REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

Asigne de manera correcta los valores nulos y no nulos según corresponda.
Visualice el diagrama implementado en SQL Server.

Caso propuesto: RESERVA DE VUELOS

Implementaremos un script de SQL Server que permita crear la base de datos **BD_RESERVAVUELOS**. Para ello se cuenta con el siguiente diagrama:



Figura 18: Diagrama de base de datos “Reserva de vuelos”

Fuente. - Tomado desde SQL Server 2014

Usando **TRANSACT/SQL**, realice lo siguiente:

Cree la base de datos **BD_RESERVAVUELOS** de manera estándar

Active la base de datos **BD_RESERVAVUELOS**

Cree las tablas mostradas

Agregue las **llaves Primarias** (ADD PRIMARY KEY)

Agregue las **llaves Foráneas, Relaciones** (ADD FOREIGN KEY REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

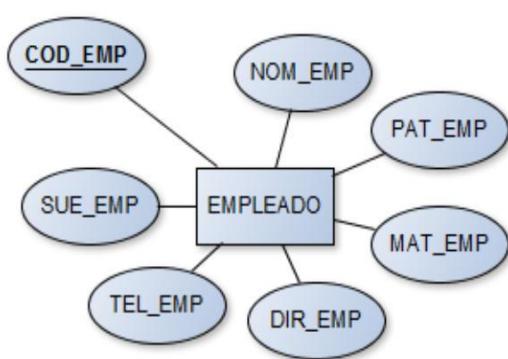
Asigne de manera correcta los valores nulos y no nulos según corresponda.

Visualice el diagrama implementado en SQL Server.

Caso desarrollado: VENTAS

Manejo de la sentencia CREATE

Cree la tabla **empleado** tal como se muestra en la siguiente imagen:



Modelo en yEd

EMPLEADO		
Column Name	Condensed Type	Nullable
COD_EMP	char(3)	No
NOM_EMP	varchar(30)	No
PAT_EMP	varchar(30)	No
MAT_EMP	varchar(30)	No
DIR_EMP	varchar(50)	No
TEL_EMP	varchar(15)	Yes
SUE_EMP	money	No

Modelo en SQL Server

Tener en cuenta:

Diseñe la tabla y sus atributos usando yEd o Studio Case.

Cree la base de datos **BD_VENTAS** de forma estándar y actívela.

Cree la tabla **Empleado**, validando su creación.

Asigne de manera correcta los tipos de datos a cada uno de los campos de tabla.

Asigne la llave primaria al código del empleado.

Asigne de manera correcta los valores nulos y no nulos según corresponda.

Visualice el diagrama implementado en SQL Server.

Primera solución:

```

IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

CREATE TABLE EMPLEADO (
    COD_EMP     CHAR      (3)      NOT NULL  PRIMARY KEY,
    NOM_EMP    VARCHAR   (30)      NOT NULL,
    PAT_EMP    VARCHAR   (30)      NOT NULL,
    MAT_EMP    VARCHAR   (30)      NOT NULL,
    DIR_EMP    VARCHAR   (50)      NOT NULL,
    TEL_EMP    VARCHAR   (15)      NOT NULL,
    SUE_EMP    MONEY      NOT NULL
)

```

```

    MAT_EMP      VARCHAR (30)      NOT NULL,
    DIR_EMP      VARCHAR (50)      NOT NULL,
    TEL_EMP      VARCHAR (15)      NULL,
    SUE_EMP      MONEY            NOT NULL
)
GO

```

Segunda solución:

```

IF OBJECT_ID ('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

CREATE TABLE EMPLEADO (
    COD_EMP      CHAR      (3)      NOT NULL,
    NOM_EMP      VARCHAR   (30)     NOT NULL,
    PAT_EMP      VARCHAR   (30)     NOT NULL,
    MAT_EMP      VARCHAR   (30)     NOT NULL,
    DIR_EMP      VARCHAR   (50)     NOT NULL,
    TEL_EMP      VARCHAR   (15)     NULL,
    SUE_EMP      MONEY            NOT NULL,
    PRIMARY KEY (COD_EMP)
)
GO

```

Nota: Para comprobar la existencia de la tabla use la siguiente sentencia:

```
SELECT * FROM SYS.TABLES
```

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date
1	EMPLEADO	245575913	NULL	1	0	U	USER_TABLE	2015-06-29 10:59:05.460	2015-06-29 11:00:42.513

Figura 19: Listando las tablas de la base de datos activa

Fuente. - Tomado desde SQL Server 2014

Manejo de la sentencia ALTER

Suponga que contamos con la tabla **empleado** el cual tiene la siguiente estructura:

EMPLEADO		
Column Name	Condensed Type	Nullable
COD_EMP	char(3)	No
NOM_EMP	varchar(30)	No
PAT_EMP	varchar(30)	No
MAT_EMP	varchar(30)	No
DIR_EMP	varchar(50)	No
TEL_EMP	varchar(15)	Yes
SUE_EMP	money	No

Figura 20: Tabla Empleado mostrando el tipo de datos

Fuente. - Tomado desde SQL Server 2014

Realice lo siguiente:

Agregar la columna **mov_emp** de tipo varchar (15) con valores nulos permitidos.

Agregar las columnas **cat_emp** (categoría del empleado) de tipo char (3) y **ema_emp** (correo electrónico del empleado) de tipo varchar (50) con valores nulos permitidos.
Eliminar la columna **tel_emp**.

Eliminar las columnas **pat_emp** y **mat_emp**.

Modificar el ancho de la columna **nom_emp** a varchar (60)

Solución:

Agregar la columna **mov_emp** (teléfono móvil) de tipo varchar (15) con valores nulos permitidos.

```
ALTER TABLE EMPLEADO
    ADD MOV_EMP      VARCHAR (15)      NULL
GO
```

Agregar las columnas **cat_emp** (categoría del empleado) de tipo char (3) el cual no acepte valores nulos y **ema_emp** (correo electrónico del empleado) de tipo varchar (50) con valores nulos permitidos.

```
ALTER TABLE EMPLEADO
    ADD EMA_EMP      VARCHAR (50)      NULL,
        CAT_EMP      CHAR (3)      NOT NULL
GO
```

Eliminar la columna **tel_emp**.

```
ALTER TABLE EMPLEADO
    DROP COLUMN TEL_EMP
GO
```

Eliminar las columnas **pat_emp** y **mat_emp**.

```
ALTER TABLE EMPLEADO
    DROP COLUMN PAT_EMP, MAT_EMP
GO
```

Modificar el ancho de la columna **nom_emp** a varchar (60) el cual no permita valores nulos.

```
ALTER TABLE EMPLEADO
    ALTER COLUMN NOM_EMP VARCHAR (60) NOT NULL
GO
```

Nota: Para comprobar los cambios efectuados en la tabla Empleado use la siguiente sentencia: **SP_COLUMNS EMPLEADO**

TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE	RADIX	NULLABLE
EMPLEADO	COD_EMP	1	char	3	3	NULL	NULL	0
EMPLEADO	NOM_EMP	12	varchar	60	60	NULL	NULL	0
EMPLEADO	DIR_EMP	12	varchar	50	50	NULL	NULL	0
EMPLEADO	TEL_EMP	12	varchar	15	15	NULL	NULL	1
EMPLEADO	SUE_EMP	3	money	19	21	4	10	0
EMPLEADO	MOV_EMP	12	varchar	15	15	NULL	NULL	1

Figura 21: Listado de columnas de la tabla Empleado

Fuente. - Tomado desde SQL Server 2014

Implementación de la relación uno a muchos

Script que permite implementar la relación de uno a muchos entre la tabla categoría y empleado, tal como muestra en la siguiente imagen:

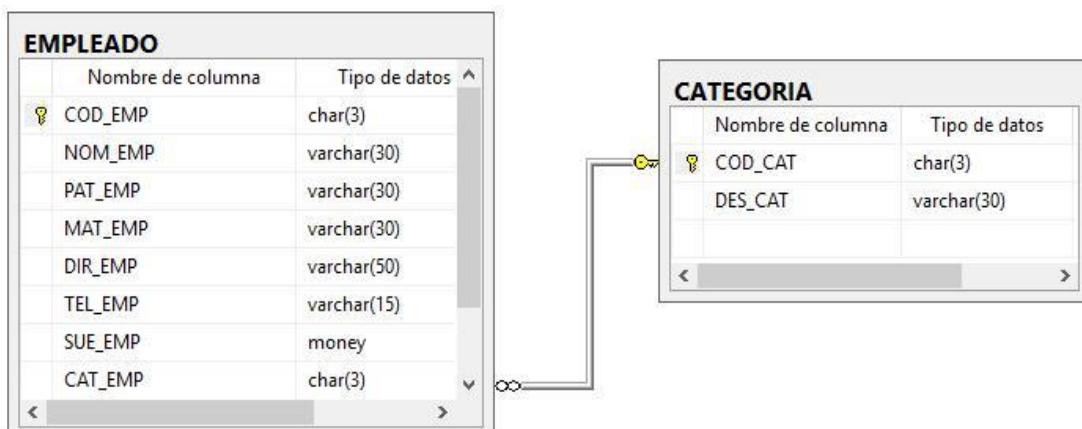


Figura 22: Relación de uno a muchos entre la tabla Empleado y Categoría

Fuente. - Tomado desde SQL Server 2014

Solución:

```

CREATE TABLE CATEGORIA (
    COD_CAT      CHAR (3)          NOT NULL PRIMARY KEY,
    DES_CAT      VARCHAR (30)       NOT NULL
)
GO

CREATE TABLE EMPLEADO (
    COD_EMP      CHAR (3)          NOT NULL PRIMARY KEY,
    NOM_EMP      VARCHAR(30)        NOT NULL,
    PAT_EMP      VARCHAR(30)        NOT NULL,
    MAT_EMP      VARCHAR(30)        NOT NULL,
    DIR_EMP      VARCHAR(50)        NOT NULL,
    TEL_EMP      VARCHAR(15)        NULL,
    SUE_EMP      MONEY             NOT NULL,
    CAT_EMP      CHAR(3)           NOT NULL REFERENCES CATEGORIA
)
GO
  
```

Implementación de la relación recursiva

Script que permite implementar la recursividad a la tabla Proveedor, tal como se muestra en la siguiente imagen:

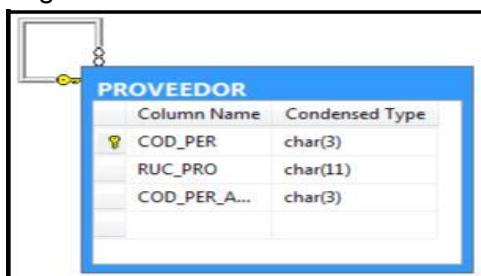


Figura 23: Tabla Proveedor y su recursividad en SQL
Fuente. - Tomado desde SQL Server 2014

Solución:

```
CREATE TABLE PROVEEDOR (
    COD_PER      CHAR (3)      NOT NULL PRIMARY KEY REFERENCES PERSONA,
    RUC_PRO      CHAR (11)     NOT NULL,
    COD_PER_ANT CHAR (3)      NOT NULL REFERENCES PROVEEDOR
)
GO
```

Implementación de la relación muchos a muchos

Script que permite implementar la relación de muchos a muchos entre las entidades empleado y departamento.

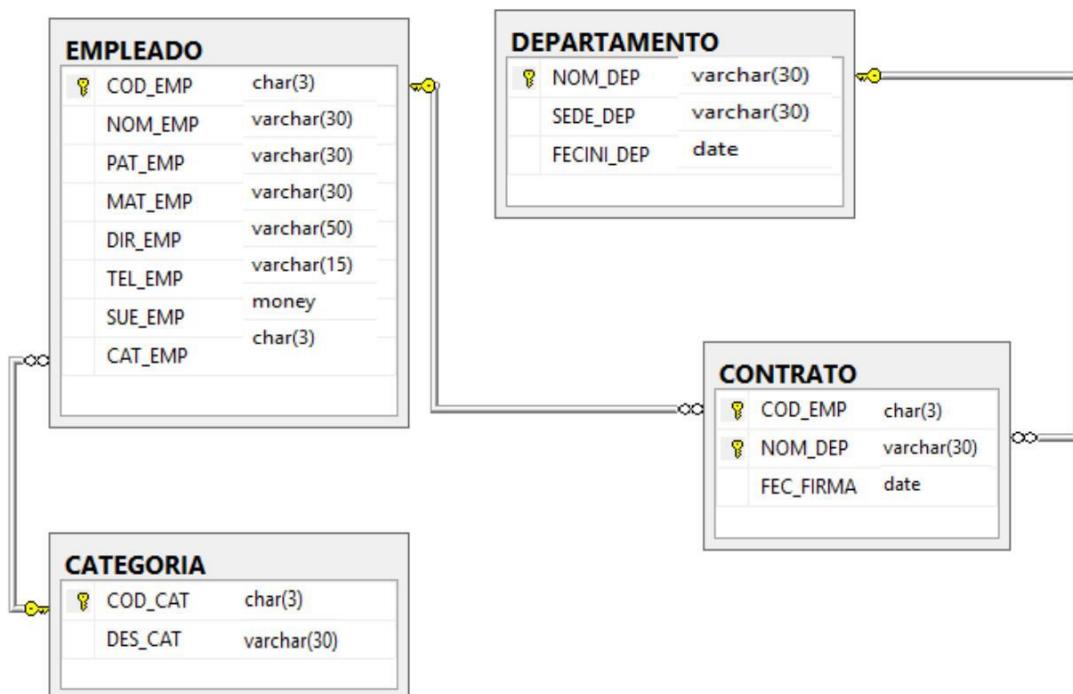


Figura 24: Implementando la relación muchos a muchos en SQL
Fuente. - Tomado desde SQL Server 2014

Solución:

```

CREATE TABLE EMPLEADO (
    COD_EMP      CHAR      (3) NOT NULL PRIMARY KEY,
    NOM_EMP      VARCHAR   (30) NOT NULL,
    PAT_EMP      VARCHAR   (30) NOT NULL,
    MAT_EMP      VARCHAR   (30) NOT NULL,
    DIR_EMP      VARCHAR   (50) NOT NULL,
    TEL_EMP      VARCHAR   (15) NULL,
    SUE_EMP      MONEY     NOT NULL,
    CAT_EMP      CHAR      (3) NOT NULL REFERENCES CATEGORIA,
)
GO

CREATE TABLE DEPARTAMENTO (
    NOM_DEP      VARCHAR   (30) NOT NULL,
    SEDE_DEP     VARCHAR   (30) NOT NULL,
    FECINI_DEP   DATE      NOT NULL,
    PRIMARY KEY (NOM_DEP)
)
GO

CREATE TABLE CONTRATO (
    COD_EMP      CHAR      (3) NOT NULL REFERENCES EMPLEADO,
    NOM_DEP      VARCHAR(30) NOT NULL REFERENCES DEPARTAMENTO,
    FEC_FIRMA    DATE      NOT NULL,
    PRIMARY KEY (COD_EMP, NOM_DEP)
)
GO

```

Implementando la generalización entre entidades

Script que permite implementar la relación uno entre las entidades clientes, persona y proveedor.

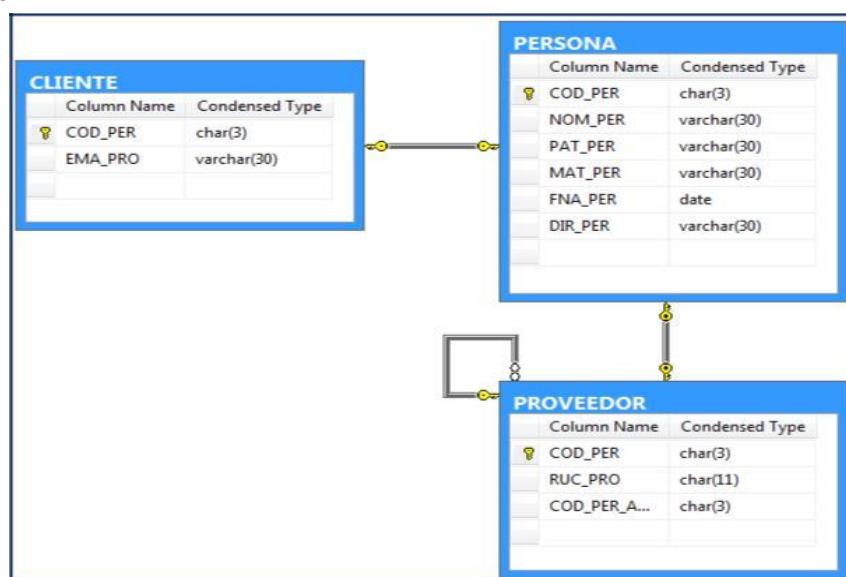


Figura 25: Implementación de la generalización entre tablas en SQL

Fuente. - Tomado desde SQL Server 2014

Solución:

```

CREATE TABLE PERSONA (
    COD_PER      CHAR      (3)      NOT NULL PRIMARY KEY,
    NOM_PER      VARCHAR   (30)     NOT NULL,
    PAT_PER      VARCHAR   (30)     NOT NULL,
    MAT_PER      VARCHAR   (30)     NOT NULL,
    FNA_PER      DATE      NOT NULL,
    DIR_PER      VARCHAR   (30)     NOT NULL,
)
GO

CREATE TABLE PROVEEDOR (
    COD_PER      CHAR      (3) NOT NULL PRIMARY KEY REFERENCES PERSONA,
    RUC_PRO      CHAR      (11)NOT NULL,
    COD_PER_ANT CHAR      (3) NOT NULL REFERENCES PROVEEDOR
)
GO

CREATE TABLE CLIENTE (
    COD_PER      CHAR      (3) NOT NULL PRIMARY KEY REFERENCES PERSONA,
    EMA_PRO      VARCHAR   (30)NOT NULL
)
GO

```

1.6. Restricciones parte II**1.6.1. Restricción DEFAULT**

Un ejemplo de valores por defecto lo podemos encontrar en la asignación **NULL** de un campo, los cuales permiten al usuario no especificar un contenido en dicha columna; asignando el valor “**NULL**” en dicha columna. Así mismo podemos implementar valores por defecto sobre ciertas columnas de una tabla.

También es conocida como cláusula **Default** e indica implícitamente al Motor de base de datos que cargue un valor predeterminado en la columna en la que no se haya especificado ningún valor.

Podemos definir un valor por defecto de las siguientes formas:

Al crear una tabla

```

CREATE TABLE NOMBRE_TABLA (
    COLUMNA1          TIPO      DEFAULT VALOR
    COLUMNA2          TIPO      DEFAULT VALOR
)

```

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	ANCHO	VALOR POR DEFECTO
IDEMPLEADO	INT		
NOMBRE_EMP	VARCHAR	30	
APELLIDOS_EMP	VARCHAR	30	
TELEFONO_EMP	CHAR	15	000-000000
EST_CIVIL_EMP	CHAR	1	S
SUELDO_EMP	MONEY		0
NUM_HIJOS_EMP	INT		0

Solución:

```

Validando la existencia de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

Creando la tabla EMPLEADO
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          NOT NULL PRIMARY KEY,
    NOMBRE_EMP      VARCHAR(30)  NOT NULL,
    APELLIDOS_EMP   VARCHAR(30)  NOT NULL,
    TELEFONO_EMP    CHAR(15)     DEFAULT '000-000000',
    EST_CIVIL_EMP   CHAR(1)      DEFAULT 'S',
    SUELDO_EMP      MONEY        DEFAULT 0,
    NUM_HIJOS_EMP   INT          DEFAULT 0
)
GO

-- Insertando registros para la probar los DEFAULT's --
Probando los valores por defecto asignado a EMPLEADO
INSERT INTO EMPLEADO VALUES (1, 'Juana', 'Zamora', DEFAULT, 'C', 2500, 1)
INSERT INTO EMPLEADO VALUES (2, 'Maria', 'Mejia', '(01)524 5636',
DEFAULT, 1500, 2)
INSERT INTO EMPLEADO VALUES (3, 'Rocio', 'De la Cruz', '982-986985', 'S',
DEFAULT, 2)
INSERT INTO EMPLEADO VALUES (4, 'Marisol', 'Salas', '978-457758', 'C', 1400,
DEFAULT)

-- Usando varios valores por defecto
INSERT INTO EMPLEADO VALUES
(5, 'Heidi', 'Reategui', DEFAULT, DEFAULT, DEFAULT, DEFAULT)

-- Verificando las inserciones
SELECT * FROM EMPLEADO
GO

```

Listando los registros de la tala **EMPLEADO** para comprobar que los registros cumplan con la especificación de los valores por defecto.

IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	NUM_HIJOS_EMP
1	Juana	Zamora	000-000000	C	2500,00	1
2	Maria	Mejia	(01)524-5636	S	1500,00	2
3	Rocio	De la Cruz	982-986985	S	0,00	2
4	Marisol	Zambrano	978-457758	C	1400,00	0
5	Heidi	Reategui	000-000000	S	0,00	0

Figura 26: Listado de empleados desde la tabla EMPLEADO

Fuente. - Tomado de SQL SERVER 2014

Agregando a una tabla ya existente

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

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	VALOR NULO	LLAVE
IDEMPLEADO	INT	NO	PRIMARIA
NOMBRE_EMP	VARCHAR	NO	
APELLIDOS_EMP	VARCHAR	NO	
TELEFONO_EMP	CHAR	SI	
EST_CIVIL_EMP	CHAR	SI	
SUELDO_EMP	MONEY	SI	
NUM_HIJOS_EMP	INT	SI	

Aplicar los siguientes valores por defecto:

Valor por defecto para el teléfono es 000-000000.

Valor por defecto al estado civil es soltero “S”.

Valor por defecto al sueldo es 0

Valor por defecto al numero de hijos es 0

Validando la existencia de la tabla EMPLEADO

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

Creando la tabla

```
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          NOT NULL PRIMARY KEY,
    NOMBRE_EMP      VARCHAR(30)  NOT NULL,
    APELLIDOS_EMP   VARCHAR(30)  NOT NULL,
    TELEFONO_EMP    CHAR(15)     NULL,
    EST_CIVIL_EMP   CHAR(1)      NULL,
    SUELDO_EMP      MONEY        NULL,
    NUM_HIJOS_EMP   INT          NULL
)
GO
```

```

Agregando los valores por defecto
ALTER TABLE EMPLEADO
    ADD DEFAULT '000-000000' FOR TELEFONO_EMP
ALTER TABLE EMPLEADO
    ADD DEFAULT 'S' FOR
EST_CIVIL_EMP ALTER TABLE EMPLEADO
    ADD DEFAULT '0' FOR SUELDO_EMP
ALTER TABLE EMPLEADO
    ADD DEFAULT '0' FOR NUM_HIJOS_EMP
GO

```

```

Insertando registros para la probar los DEFAULT's
Probando los valores por defecto asignado a EMPLEADO
INSERT INTO EMPLEADO
    VALUES (1, 'Juana', 'Zamora', DEFAULT, 'C', 2500, 1)
INSERT INTO EMPLEADO
    VALUES (2, 'Maria', 'Mejia', '(01)524-5636', DEFAULT, 1500, 2)
INSERT INTO EMPLEADO
    VALUES (3, 'Rocio', 'De la Cruz',
        '982-986985', 'S', DEFAULT, 2)
INSERT INTO EMPLEADO
    VALUES (4, 'Marisol', 'Zambrano', '978-
        457758', 'C', 1400, DEFAULT)

```

```

Usando varios valores por defecto
INSERT INTO EMPLEADO
    VALUES (5, 'Heidi', 'Reategui',
        DEFAULT, DEFAULT, DEFAULT, DEFAULT)

```

```

Verificando las inserciones
SELECT * FROM
EMPLEADO GO

```

Podríamos realizar la asignación de valores por defecto en una sola instrucción de alteración de tabla, tal como se muestra en el siguiente script:

```

ALTER TABLE EMPLEADO
    ADD DEFAULT '000-000000' FOR TELEFONO_EMP,
    DEFAULT 'S' FOR EST_CIVIL_EMP,
    DEFAULT '0' FOR SUELDO_EMP,
    DEFAULT '0' FOR NUM_HIJOS_EMP
GO

```

Finalmente, si necesitamos visualizar el tipo de asignación por defecto realizada a la tabla **EMPLEADO** podemos emplear la siguiente sentencia:

```
SP_HELPCONSTRAINT EMPLEADO
```

El resultado sería:

Object Name
1 EMPLEADO
1 DEFAULT on column EST_CIVIL_EMP DF_EMPLEADO_EST_CI_3B75D760 (n/a) (n/a) (n/a) (n/a) ('S')
2 DEFAULT on column NUM_HIJOS_EMP DF_EMPLEADO_NUM_HI_3D5E1FD2 (n/a) (n/a) (n/a) (n/a) ('0')
3 DEFAULT on column SUELDO_EMP DF_EMPLEADO_SUELDO_3C69FB99 (n/a) (n/a) (n/a) (n/a) ('0')
4 DEFAULT on column TELEFONO_EMP DF_EMPLEADO_TELEFO_3A81B327 (n/a) (n/a) (n/a) (n/a) ('000-000000')
5 PRIMARY KEY (clustered) PK_EMPLEADO_E014C316ABC1380 (n/a) (n/a) (n/a) (n/a) IDEMPLEADO

Figura 27: Listado de valores por defecto asignada a la tabla EMPLEADO

Fuente. - Tomado de SQL SERVER 2014

1.6.2. Restricción CHECK

Es importante imponer la integridad de dominio, asegurar que sólo puedan existir entradas de los tipos o rangos esperados para una columna determinada. SQL Server impone la integridad de dominio a través del Check Constraint.

Una columna puede tener cualquier número de restricciones CHECK y la condición puede incluir varias expresiones lógicas combinadas con AND y OR. Por ello, las restricciones CHECK para una columna se validan en el orden en que se crean.

La condición de búsqueda debe dar como resultado una expresión booleana y no puede hacer referencia a otra tabla.

Una restricción CHECK, en el nivel de columna, solo puede hacer referencia a la columna restringida y una restricción CHECK, en el nivel de tabla, solo puede hacer referencia a columnas de la misma tabla.

Las restricciones CHECK y las reglas sirven para la misma función de validación de los datos durante las instrucciones INSERT y DELETE.

Cuando hay una regla y una o más restricciones CHECK para una columna o columnas, se evalúan todas las restricciones.

Podemos definir una restricción CHECK de las siguientes formas:

Al crear una tabla

```
CREATE TABLE NOMBRE_TABLA (
    COLUMNA1 TIPO NULL|NOT NULL      CHECK (CONDICIÓN)
    COLUMNA2 TIPO NULL|NOT NULL      CHECK (CONDICIÓN)
)
```

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	ANCHO	RESTRICCIÓN
IDEMPLEADO	INT		
NOMBRE_EMP	VARCHAR	30	
APELLIDOS_EMP	VARCHAR	30	
TELEFONO_EMP	CHAR	15	
EST_CIVIL_EMP	CHAR	1	S-C-V-D
SUELDO_EMP	MONEY		1000 A 2500
NUM_HIJOS_EMP	INT		>=0

Solución:

```

Validando la existencia de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

1. Creando la tabla EMPLEADO
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          NOT NULL      PRIMARY KEY,
    NOMBRE_EMP       VARCHAR (30)   NOT NULL,
    APELLIDOS_EMP    VARCHAR (30)   NOT NULL,
    TELEFONO_EMP     CHAR (15)    NULL,
    EST_CIVIL_EMP    CHAR (1)     NOT NULL
        CHECK (EST_CIVIL_EMP IN ('S','C','V','D')),

    SUELDO_EMP       MONEY        NOT NULL
        CHECK (SUELDO_EMP>=1000 AND SUELDO_EMP<=2500),

    NUM_HIJOS_EMP    INT          NOT NULL
        CHECK (NUM_HIJOS_EMP >= 0)
)
GO

-- 2. Inserciones validas
INSERT INTO EMPLEADO VALUES (1,'Jose','Blanco',
                            '525-5685','C',1850,1);
INSERT INTO EMPLEADO VALUES (2,'Maria','Rengifo',
                            '985-698569','S',2500,0);
INSERT INTO EMPLEADO VALUES (3,'Milagros','Acosta',
                            '998-562563','D',1100,3);

3. Inserciones que rompen la regla del Constraint
INSERT INTO EMPLEADO VALUES (4,'Janeth','De la Cruz',
                            '999-253625','X',1100,3);
INSERT INTO EMPLEADO VALUES (5,'July','Hijar', '485-
                            5285','S',750,0);
INSERT INTO EMPLEADO VALUES (6,'Carmen','Salirosas',
                            '582-158258','D',2500,-1);

```

Agregar a una tabla ya existente

```
ALTER TABLE NOMBRE_TABLA
ADD CONSTRAINT NOMBRE_RESTRICCIÓN
CHECK (CONDICIÓN)
```

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	VALOR NULO	LLAVE
IDEMPLEADO	INT	NO	PRIMARIA
NOMBRE_EMP	VARCHAR	NO	
APELLIDOS_EMP	VARCHAR	NO	
TELEFONO_EMP	CHAR	SI	
EST_CIVIL_EMP	CHAR	SI	
SUELDO_EMP	MONEY	SI	
NUM_HIJOS_EMP	INT	SI	

Aplicar las siguientes restricciones:

Solo debe permitir el ingreso de valores S (soltero), C (casado), V (viudo) y D (divorciado) al campo estado civil del empleado.

El monto asignado al sueldo del empleado debe encontrarse entre S/. 1000 y S/. 2500.

El ingreso de número de hijos no debe permitir valores negativos.

```
Validando la creación de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

--1. Creando la tabla
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          NOT NULL PRIMARY KEY,
    NOMBRE_EMP      VARCHAR (30) NOT NULL,
    APELLIDOS_EMP   VARCHAR (30) NOT NULL,
    TELEFONO_EMP    CHAR (15)    NULL,
    EST_CIVIL_EMP   CHAR (1)     NULL,
    SUELDO_EMP      MONEY       NULL,
    NUM_HIJOS_EMP   INT         NULL
)

--2. Agregando valores por defecto
ALTER TABLE EMPLEADO
    ADD CONSTRAINT CHK_ESTADO
    CHECK (EST_CIVIL_EMP IN('S', 'C', 'V', 'D'))
```

```

ALTER TABLE EMPLEADO
    ADD CONSTRAINT CHK_SUELDO
    CHECK(SUELDO_EMP>=1000 AND SUELDO_EMP<=2500)

ALTER TABLE EMPLEADO
    ADD CONSTRAINT CHK_HIJOS
    CHECK(NUM_HIJOS_EMP>=0)
GO

--3. Probando las restricciones
-- Inserciones validas
INSERT INTO EMPLEADO VALUES (1,'Jose','Blanco', '525-
5685','C',1850,1);
INSERT INTO EMPLEADO VALUES (2,'Maria','Rengifo', '985-
698569','S',2500,0);
INSERT INTO EMPLEADO VALUES (3,'Milagros','Acosta',
'998-562563','D',1100,3);

```

Veamos algunas inserciones que rompen la regla de las restricciones:

Insertando el valor X en el estado civil del empleado; cuando la restricción solo permite valores S, C, V y D:

```
INSERT INTO EMPLEADO VALUES (4,'Janeth','De la Cruz',
'999-253625','X',1100,3);
```

El mensaje desde el servidor sería:

Figura 28: Rompimiento del constraint estado civil
Fuente. - Tomado de SQL SERVER 2014

Insertando el monto S/. 750.00 en el sueldo del empleado; cuando la restricción solo permite el ingreso de montos mayores o igual a S/. 1000.00 pero menos o iguales a S/. 2500.00:

```
INSERT INTO EMPLEADO VALUES (5,'July','Hijar',
'485-5285','S',750,0);
```

El mensaje desde el servidor sería:

Figura 28: Rompimiento del constraint sueldo del empleado
Fuente. - Tomado de SQL SERVER 2014

Insertando el monto -1 en el número de hijos del empleado; cuando la restricción solo permite la cantidad de cero a más:

```
INSERT INTO EMPLEADO VALUES (6, 'Carmen', 'Salirosas',
                            '582-158258', 'D', 2500, -1);
```

El mensaje desde el servidor sería:

Figura 29: Rompimiento del constraint número de hijos del empleado
Fuente. - Tomado de SQL SERVER 2014

Opciones de las restricciones

Listar todos los Constraint de la tabla **EMPLEADO**

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME='NOMBRE DE LA TABLA'
```

Así, por ejemplo: si necesitamos listar los constraints asignado a la tabla empleado, tenemos:

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

CONSTRAINT_SCHEMA	CONSTRAINT_NAME
dbo	PK_EMPLEADO_E014C3167F4C5262
dbo	CK_EMPLEADO_EST_CI_4AB81AF0
dbo	CK_EMPLEADO_SUELDO_4BAC3F29
dbo	CK_EMPLEADO_NUM_HI_4CA06362

Figura 30: Listando todas las restricciones aplicadas a la tabla empleado
Fuente. - Tomado de SQL SERVER 2014

Eliminar un Constraint CHECK

```
ALTER TABLE NOMBRE_TABLA
DROP CONSTRAINT NOMBRE_RESTRICCIÓN
```

Así, por ejemplo: si necesitamos eliminar el constraint asignado a la columna número de hijos, tenemos:

```
Inhabilitando el constraint
CHK_SUELDO. ALTER TABLE EMPLEADO
DROP CONSTRAINT CK_EMPLEADO_NUM_HI_4CA06362
```

Listando los constraint de la tabla Empleado.

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

Inhabilitar un Constraint CHECK

```
ALTER TABLE NOMBRE_TABLA
NOCHECK CONSTRAINT NOMBRE_RESTRICCIÓN
```

Así, por ejemplo: si necesitamos inhabilitar el constraint asignado a la columna sueldo del empleado, tenemos:

-- Inhabilitando el constraint CHK_SUELDO.

```
ALTER TABLE EMPLEADO NOCHECK CONSTRAINT CHK_SUELDO;
```

Probando la inhabilitando del constraint CHK_SUELDO.

```
INSERT INTO EMPLEADO VALUES (4, 'July', 'Hijar',
'485-5285', 'S', 750, 0);
```

Habilitar un Constraint CHECK

```
ALTER TABLE NOMBRE_TABLA
CHECK CONSTRAINT NOMBRE_RESTRICCIÓN
```

Así, por ejemplo: si necesitamos habilitar el constraint de la columna sueldo del empleado, tenemos:

-- Habilitar el constraint CHK_SUELDO.

```
ALTER TABLE EMPLEADO CHECK CONSTRAINT CHK_SUELDO;
```

Probando la habilitación del constraint CHK_SUELDO.

```
INSERT INTO EMPLEADO VALUES (5, 'July', 'Hijar',
'485-5285', 'S', 750, 0);
```

1.6.3. Restricción UNIQUE

La restricción **UNIQUE** asigna a uno o varios campos de una tabla que sus valores no sean repetidos, es decir que sus valores sean únicos. La forma de implementar la restricción **UNIQUE** es al momento de crear la tabla o agregarla después de crear la tabla; pero si es una tabla existente y contiene registros duplicados, el motor de base de datos envía un mensaje de error al agregar la restricción **UNIQUE**.

Podemos definir una restricción **UNIQUE** de las siguientes formas:

Al crear una tabla

```
CREATE TABLE NOMBRE_TABLA (
    COLUMNA1      TIPO NULL|NOT NULL      UNIQUE
)
```

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	ANCHO	RESTRICCIÓN
IDEMPLEADO	INT		PRIMARY
NOMBRE_EMP	VARCHAR	30	
APELLIDOS_EMP	VARCHAR	30	
TELEFONO_EMP	CHAR	15	UNIQUE
EST_CIVIL_EMP	CHAR	1	
SUELDO_EMP	MONEY		
CORREO_EMP	VARCHAR	40	UNIQUE

Solución:

```
Validando la creación de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

Creando la tabla
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT      NOT NULL      PRIMARY KEY,
    NOMBRE_EMP      VARCHAR(30) NOT NULL,
    APELLIDOS_EMP   VARCHAR(30) NOT NULL,
    TELEFONO_EMP    CHAR(15)    NULL        UNIQUE,
    EST_CIVIL_EMP   CHAR(1)    NULL,
    SUELDO_EMP      MONEY     NULL,
    CORREO_EMP      VARCHAR(40) NOT NULL    UNIQUE
)
GO

Agregando registros validos
INSERT INTO EMPLEADO
    VALUES (1, 'Jose', 'Blanco',
            '525-5685', 'C', 1850, 'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
    VALUES (2, 'Maria', 'Rengifo', '985-
            698569', 'S', 2500, 'mrenfigo@cibertec.edu.pe')
```

```
INSERT INTO EMPLEADO
VALUES (3, 'Milagros', 'Acosta', '998-
562563', 'D', 1100, 'macosta@cibertec.edu.pe')
GO
```

Listando los registros de los Empleados

```
SELECT * FROM
EMPLEADO GO
```

Veamos algunas inserciones que rompen la regla de la restricción **UNIQUE**:

Insertando dos registros, el primero repite el correo electrónico del registro uno, mientras que el segundo repite el teléfono del mismo registro uno:

Agregando registros validos

```
INSERT INTO EMPLEADO
VALUES (4, 'Juan José', 'Blanco',
'(01) 756-5498', 'C', 1850, 'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
VALUES (5, 'Martin', 'Renteria', '525-
5685', 'C', 1200, 'mrenteria@cibertec.edu.pe')
```

El mensaje desde el servidor sería:

```
Msg 2627, Level 14, State 1, Line 1
Violation of UNIQUE KEY constraint 'UQ_EMPLEADO_ABF5D56FAE1C4C77'. Cannot insert duplicate key in object 'dbo.EMPLEADO'.
The duplicate key value is (jblanco@cibertec.edu.pe).
The statement has been terminated.

Msg 2627, Level 14, State 1, Line 3
Violation of UNIQUE KEY constraint 'UQ_EMPLEADO_4B08E75F62C4581A'. Cannot insert duplicate key in object 'dbo.EMPLEADO'.
The duplicate key value is (525-5685).
The statement has been terminated.
```

Figura 31: Rompimiento del constraint UNIQUE
Fuente. - Tomado de SQL SERVER 2014

Agregar a una tabla ya existente

```
ALTER TABLE NOMBRE_TABLA
ADD CONSTRAINT NOMBRE_RESTRICCIÓN
UNIQUE (CAMPO)
```

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura

CAMPO	TIPO	VALOR NULO	LLAVE
IDEMPLEADO	INT	NO	PRIMARIA
NOMBRE_EMP	VARCHAR(30)	NO	
APELLIDOS_EMP	VARCHAR(30)	NO	
TELEFONO_EMP	CHAR(15)	SI	
EST_CIVIL_EMP	CHAR(1)	SI	
SUELDO_EMP	MONEY	SI	
CORREO_EMP	VARCHAR(40)	NO	

Aplicar las siguientes restricciones:

Solo debe permitir el ingreso de valores únicos para las columnas teléfono y correo electrónico del empleado.

Solución:

```

Validando la creación de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO')IS NOT NULL
    DROP TABLE EMPLEADO
GO

Creando la tabla
CREATE TABLE EMPLEADO(
    IDEMPLEADO      INT          NOT NULL  PRIMARY KEY,
    NOMBRE_EMP      VARCHAR(30) NOT NULL,
    APELLIDOS_EMP   VARCHAR(30) NOT NULL,
    TELEFONO_EMP    CHAR(15)     NULL,
    EST_CIVIL_EMP   CHAR(1)      NULL,
    SUELDO_EMP      MONEY       NULL,
    CORREO_EMP      VARCHAR(40) NOT NULL
)
GO

Agregando la restricción UNIQUE
ALTER TABLE EMPLEADO
    ADD CONSTRAINT UNI_TELEFONO
    UNIQUE (TELEFONO_EMP)

ALTER TABLE EMPLEADO
    ADD CONSTRAINT UNI_CORREO
    UNIQUE (CORREO_EMP)
GO

Agregando registros validos
INSERT INTO EMPLEADO
    VALUES (1,'Jose','Blanco',
            '525-5685','C',1850,'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
    VALUES (2,'Maria','Rengifo', '985-
            698569','S',2500,'mrenfigo@cibertec.edu.pe')

```

```
INSERT INTO EMPLEADO
VALUES (3, 'Milagros', 'Acosta', '998-
562563', 'D', 1100, 'macosta@cibertec.edu.pe')
GO
```

Listando los registros de los Empleados

```
SELECT * FROM
EMPLEADO GO
```

Agregando registros validos

```
INSERT INTO EMPLEADO
VALUES (4, 'Juan José', 'Blanco',
'(01) 756-5498', 'C', 1850, 'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
VALUES (5, 'Martin', 'Renteria', '525-
5685', 'C', 1200, 'mreenteria@cibertec.edu.pe')
```

Debemos recordar que para visualizar las restricciones implementadas debemos usar la siguiente sentencia:

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

1.6.4. Restricción IDENTITY

Asigna un valor incremental a una sola columna dentro de una tabla, la asignación Identity lo podrá encontrar normalmente en campos claves de tipo numérico. Identity presenta algunas desventajas que mencionamos a continuación:

No garantiza la unicidad del valor ya que esta es especificada mediante la restricción **Primary Key** o un **Unique**.

No garantiza el registro de números consecutivos al registrar valores consecutivos.

Podemos definir una restricción **IDENTITY** de la siguiente forma:

```
CREATE TABLE NOMBRE_TABLA (
    COLUMNA1      INT      IDENTITY (VALOR_INICIO, VALOR_INCREMENTO)
)
```

El valor de inicio, indica el punto de partida de la secuencia de números; debemos mencionar que no siempre será el número uno. En valor de incremento determina el aumento de la razón numérica según el valor inicial. También debemos mencionar que, al no especificar un valor de inicio y valor de incremento los valores serán (1,1).

Por ejemplo:

Crear la tabla **EMPLEADO** con la siguiente estructura:

CAMPO	TIPO	ANCHO	RESTRICCIÓN
IDEMPLEADO	INT		IDENTITY 100, 1
NOMBRE_EMP	VARCHAR	30	
APELLIDOS_EMP	VARCHAR	30	
TELEFONO_EMP	CHAR	15	
EST_CIVIL_EMP	CHAR	1	
SUELDO_EMP	MONEY		
CORREO_EMP	VARCHAR	40	

Solución:

```

Validando la creación de la tabla EMPLEADO
IF OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

Creando la tabla
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          PRIMARY KEY IDENTITY (100,1),
    NOMBRE_EMP       VARCHAR(30)  NOT NULL,
    APELLIDOS_EMP    VARCHAR(30)  NOT NULL,
    TELEFONO_EMP     CHAR(15)     NULL,
    EST_CIVIL_EMP    CHAR(1)      NULL,
    SUELDO_EMP       MONEY        NULL,
    CORREO_EMP        VARCHAR(40)  NOT NULL
)
GO

Agregando registros validos
INSERT INTO EMPLEADO
    VALUES ('José','Blanco',
            '525-5685','C',1850,'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
    VALUES ('María','Rengifo',
            '985-698569','S',2500,'mrenfigo@cibertec.edu.pe')
INSERT INTO EMPLEADO
    VALUES ('Milagros','Acosta', '998-
            562563','D',1100,'macosta@cibertec.edu.pe')
GO

Listando los registros de los Empleados
SELECT * FROM EMPLEADO

```

El resultado de la consulta a la tabla **EMPLEADO** luego de asignar identity a la columna IdEmpleado es:

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	100	Jose	Blanco	525-5685	C	1850.00	jblanco@cibertec.edu.pe
2	101	Maria	Rengifo	985-698569	S	2500.00	mrenfigo@cibertec.edu.pe
3	102	Milagros	Acosta	998-562563	D	1100.00	macosta@cibertec.edu.pe

Figura 32: Visualizando los registros de la tabla EMPLEADO
Fuente. - Tomado de SQL SERVER 2014

Casos propuestos y desarrollados

Caso desarrollado: COLEGIO

En la base de datos **BD_COLEGIO** implementaremos las restricciones correspondientes a las tablas de datos. En seguida se muestra el diagrama

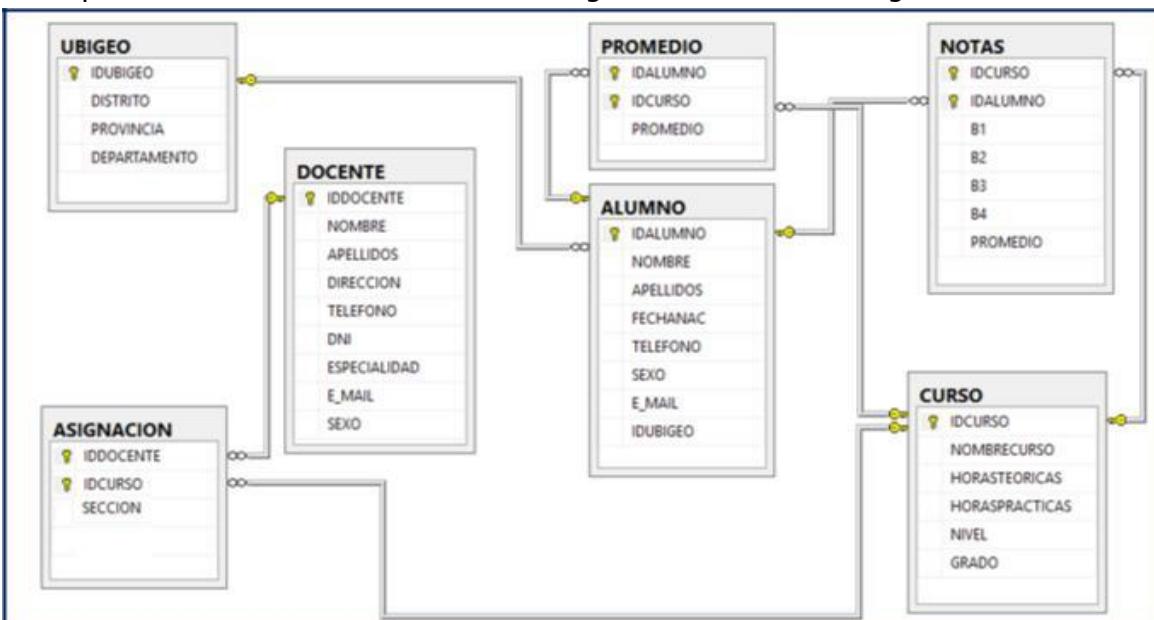


Figura 33: Visualizando el modelo relacional de: "Colegio"
Fuente. - Tomado de SQL SERVER 2014

SCRIPT

USE MASTER

GO

```

IF DB_ID('BD_COLEGIO') IS NOT NULL
DROP DATABASE BD_COLEGIO
GO
    
```

```

CREATE DATABASE BD_COLEGIO
ON PRIMARY
(NAME = 'COLEGIO_DATA',
FILENAME = 'D:\COLEGIO_DATA.MDF',
SIZE = 10MB,
MAXSIZE = 20MB,
FILEGROWTH= 1MB)
LOG ON
(NAME = 'COLEGIO_LOG',
    
```

```

FILENAME = 'D:\COLEGIO_LOG.LDF',
SIZE = 5MB,
MAXSIZE = 10MB,
FILEGROWTH= 1MB)
GO
--ABRIENDO LA BASE DE DATOS BD_COLEGIO
USE BD_COLEGIO
GO

```

--CREACIÓN DE TABLAS:

--TABLA CURSO

```

CREATE TABLE CURSO
(
IDCURSO      CHAR      (5) NOT NULL PRIMARY KEY,
NOMBRECURSO   VARCHAR   (15) NOT NULL,
HORASTEORICAS NUMERIC   NOT NULL,
HORASPRATICAS NUMERIC   NOT NULL,
NIVEL         CHAR      (1) NOT NULL,
GRADO         CHAR      (1) NOT NULL
)

```

--TABLA DOCENTE

```

CREATE TABLE DOCENTE
(
IDDOCENTE     CHAR      (5) NOT NULL PRIMARY KEY,
NOMBRE         VARCHAR   (25) NOT NULL,
APELLIDOS      VARCHAR   (35) NOT NULL,
DIRECCION      VARCHAR   (50) NOT NULL,
TELEFONO        VARCHAR   (12),
DNI            CHAR      (8) NOT NULL,
ESPECIALIDAD   VARCHAR   (25) NOT NULL,
E_MAIL          VARCHAR   (50),
SEXO           CHAR      (1) NOT NULL
)

```

--TABLA UBIGEO

```

CREATE TABLE UBIGEO
(
IDUBIGEO      CHAR      (6) NOT NULL PRIMARY KEY,
DISTRITO      VARCHAR   (35) NOT NULL,
PROVINCIA     VARCHAR   (25) NOT NULL,
DEPARTAMENTO  VARCHAR   (25) NOT NULL
)

```

--TABLA ALUMNO

```
CREATE TABLE ALUMNO
(
  IDALUMNO      CHAR      (5) NOT NULL PRIMARY KEY,
  NOMBRE         VARCHAR   (25) NOT NULL,
  APELLIDOS     VARCHAR   (35) NOT NULL,
  FECHANAC      DATETIME NOT NULL,
  TELEFONO      VARCHAR   (12),
  SEXO          CHAR      (1) NOT NULL,
  E_MAIL         VARCHAR   (50),
  IDUBIGEO      CHAR      (6) NOT NULL REFERENCES UBIGEO
)
```

--TABLA PROMEDIO

```
CREATE TABLE PROMEDIO
(
  IDALUMNO      CHAR      (5) NOT NULL REFERENCES ALUMNO,
  IDCURSO       CHAR      (5) NOT NULL REFERENCES CURSO,
  PROMEDIO      REAL,
  PRIMARY KEY (IDALUMNO, IDCURSO)
)
```

--TABLA ASIGNACIÓN

```
CREATE TABLE ASIGNACION
(
  IDDOCENTE     CHAR      (5) NOT NULL REFERENCES DOCENTE,
  IDCURSO       CHAR      (5) NOT NULL REFERENCES CURSO,
  SECCION        CHAR      (1) NOT NULL,
  PRIMARY KEY (IDDOCENTE, IDCURSO)
)
```

--TABLA NOTAS

```
CREATE TABLE NOTAS
(
  IDCURSO       CHAR      (5) NOT NULL REFERENCES CURSO,
  IDALUMNO      CHAR      (5) NOT NULL REFERENCES ALUMNO,
  B1            FLOAT    NOT NULL,
  B2            FLOAT    NOT NULL,
  B3            FLOAT    NOT NULL,
  B4            FLOAT    NOT NULL,
  PROMEDIO      FLOAT    NOT NULL
  PRIMARY KEY (IDCURSO, IDALUMNO)
)
GO
```

Debe tener en cuenta los siguientes aspectos:

Definir las llaves primarias y foráneas según se muestra en la imagen.
Definir valores por defecto (DEFAULT)
Definir restricciones (CHECK)
Definir restricciones (UNIQUE)

IMPLEMENTACIÓN DE LAS RESTRICCIONES

--ABRIR LA BASE DE DATOS BD_COLEGIO

USE BD_COLEGIO

GO

--APLICANDO RESTRICCIÓN DEFAULT

--ASIGNAR EL VALOR CERO AL B1, B2, B3, B4 DE LA TABLA NOTAS

ALTER TABLE NOTAS

ADD CONSTRAINT DF_NOTASB1 DEFAULT 0
FOR B1

ALTER TABLE NOTAS

ADD CONSTRAINT DF_NOTASB2 DEFAULT 0
FOR B2

ALTER TABLE NOTAS

ADD CONSTRAINT DF_NOTASB3 DEFAULT 0
FOR B3

ALTER TABLE NOTAS

ADD CONSTRAINT DF_NOTASB4 DEFAULT 0
FOR B4

--ASIGNAR EL VALOR CERO AL CAMPO PROMEDIO DE LA TABLA PROMEDIO

ALTER TABLE PROMEDIO

ADD CONSTRAINT DF_NOTASPPROM DEFAULT 0
FOR PROMEDIO

--ASIGNAR EL VALOR CERO AL CAMPO HORASTEORICAS DE LA TABLA CURSO

ALTER TABLE CURSO

ADD CONSTRAINT DF_HT DEFAULT 0
FOR HORASTEORICAS

--ASIGNAR EL VALOR CERO AL CAMPO HORASPRACTICAS DE LA TABLA CURSO

```
ALTER TABLE CURSO
ADD CONSTRAINT DF_HP DEFAULT 0
FOR HORASPRATICAS
GO

--ASIGNAR EL VALOR 'NO REGISTRA' AL CAMPO E_MAIL DE LA TABLA
DOCENTE ALTER TABLE DOCENTE
ADD CONSTRAINT DF_EMAIL DEFAULT 'NO REGISTRA'
FOR E_MAIL
GO

--APLICANDO RESTRICCIÓN CHECK

--EL CAMPO SEXO DE LA TABLA ALUMNO Y DOCENTE DEBE PERMITIR VALOR F y M

ALTER TABLE ALUMNO
ADD CONSTRAINT CHK_SEXO CHECK (SEXO LIKE '[FM]')
GO

ALTER TABLE DOCENTE
ADD CONSTRAINT CHK_SEXODOC CHECK (SEXO LIKE '[FM]')
GO

--EL CODIGO DEL ALUMNO DEBE COMENZAR CON LA LETRA A

ALTER TABLE ALUMNO
ADD CONSTRAINT CHK_IDA CHECK (IDALUMNO LIKE 'A[0-9][0-9][0-9][0-9]')
GO

--EL CODIGO DEL DOCENTE DEBE COMENZAR CON LA LETRA D

ALTER TABLE DOCENTE
ADD CONSTRAINT CHK_IDD CHECK (IDDOCENTE LIKE 'D[0-9][0-9][0-9][0-9]')
GO

--EL CODIGO DEL CURSO DEBE COMENZAR CON LA LETRA C

ALTER TABLE CURSO
ADD CONSTRAINT CHK(IDC) CHECK(IDCURSO LIKE 'C[0-9][0-9][0-9][0-9]')
GO

--EL CAMPO HORASTEORICAS DE LA TABLA CURSO DEBE ACEPTAR
VALORES MAYORES IGUALES A 0

ALTER TABLE CURSO
ADD CONSTRAINT CHK_CHT CHECK(HORASTEORICAS>=0)
GO
```

EL CAMPO HORASPRACTICAS DE LA TABLA CURSO DEBE ACEPTAR VALORES MAYORES IGUALES A 0

```
ALTER TABLE CURSO
ADD CONSTRAINT CHK_CHP CHECK(HORASPRACTICAS>=0)
GO
```

-- EL CAMPO GRADO DEBE ACEPTAR VALORES MAYORES ENTRE 1 Y 6

```
ALTER TABLE CURSO
ADD CONSTRAINT CHK_CGRADO CHECK (GRADO LIKE '[1-6]')
GO
EL CAMPO GRADO DEBE ACEPTAR VALORES MAYORES ENTRE 1 Y 6
ALTER TABLE CURSO
ADD CONSTRAINT CHK_CNIVEL CHECK (NIVEL IN('P','S'))
GO
```

--LOS CAMPOS B1 a B4 Y PROMEDIO DE LA TABLA NOTAS DEBE ACEPTAR VALORES DE 0 A 20

```
ALTER TABLE NOTAS
ADD CONSTRAINT CHK_NB1 CHECK(B1>=0 AND B1<=20),
CONSTRAINT CHK_NB2 CHECK(B2>=0 AND B2<=20),
CONSTRAINT CHK_NB3 CHECK(B3>=0 AND B3<=20),
CONSTRAINT CHK_NB4 CHECK(B4>=0 AND B4<=20),
CONSTRAINT CHK_NPROM CHECK(PROMEDIO>=0 AND PROMEDIO<=20)
```

--APLICANDO RESTRICCIÓN UNIQUE

--EL CAMPO EMAIL DE LA TABLA DOCENTE DEBE SER ÚNICO.

```
ALTER TABLE DOCENTE
ADD CONSTRAINT UQ_EMA
UNIQUE (E_MAIL)
GO
```

--EL CAMPO DIRECCIÓN DE LA TABLA DOCENTE DEBE SER ÚNICO

```
ALTER TABLE DOCENTE
ADD CONSTRAINT UQ_DIRECCION
UNIQUE (DIRECCION)
GO
```

--EL CAMPO DNI DE LA TABLA DOCENTE DEBE SER ÚNICO

```
ALTER TABLE DOCENTE
ADD CONSTRAINT UQ_DNI
UNIQUE (DNI)
GO
```

--EL CAMPO DEPARTAMENTO DE LA TABLA UBIGEO DEBE SER ÚNICO

```
ALTER TABLE UBIGEO
ADD CONSTRAINT UQ_DEPA
UNIQUE (DEPARTAMENTO)
GO
```

Caso propuesto: VENTAS2017

En la base de datos **VENTAS2017** implementaremos las restricciones correspondientes a las tablas de datos. En seguida se muestra el diagrama

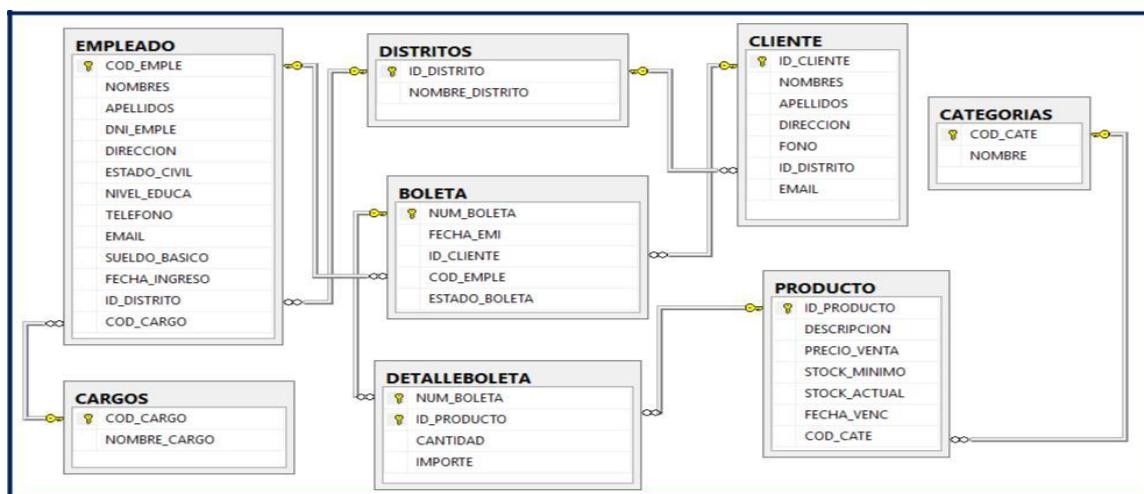


Figura 34: Visualizando el modelo relacional de: "Ventas"

Fuente. - Tomado de SQL SERVER 2014

SCRIPT

--ABRIENDO LA BASE DE DATOS DEL SISTEMA

```
USE MASTER
```

```
GO
```

--VALIDAR LA BASE DE DATOS

```
IF DB_ID('VENTAS2017') IS NOT NULL
    DROP DATABASE VENTAS2017
GO
```

--CREANDO LA BASE DE DATOS

```
CREATE DATABASE VENTAS2017
```

```
GO
```

--ABRIENDO LA BASE DE DATOS

```
USE VENTAS2017
```

```
GO
```

--CAMBIANDO EL FORMATO DE LA FECHA

```
SET DATEFORMAT DMY
```

```
GO
```

--CREANDO TABLAS Y VALIDANDO

--TABLA CATEGORIAS

```
IF OBJECT_ID ('CATEGORIAS') IS NOT NULL
BEGIN
DROP TABLE CATEGORIAS
END
CREATE TABLE CATEGORIAS
(
    COD_CATE      CHAR      (3)  NOT NULL PRIMARY KEY,
    NOMBRE        VARCHAR   (25) NOT NULL
)
GO
```

--TABLA DISTRITOS

```
IF OBJECT_ID('DISTRITOS') IS NOT NULL
BEGIN
DROP TABLE DISTRITOS
END

CREATE TABLE DISTRITOS
(
    ID_DISTRITO    CHAR      (4)  NOT NULL PRIMARY KEY,
    NOMBRE_DISTRITO VARCHAR   (40) NOT NULL
)
GO
```

--TABLA CARGOS

```
IF OBJECT_ID('CARGOS') IS NOT NULL
BEGIN
DROP TABLE CARGOS
END
CREATE TABLE CARGOS
(
    COD_CARGO      CHAR      (3)  NOT NULL PRIMARY KEY,
    NOMBRE_CARGO   VARCHAR   (30) NOT NULL
)
GO
```

--TABLA EMPLEADO

```
IF OBJECT_ID('EMPLEADO') IS NOT NULL
BEGIN
DROP TABLE EMPLEADO
END
```

```
CREATE TABLE EMPLEADO
```

```
(  
    COD_EMPLÉO      CHAR      (5) NOT NULL PRIMARY KEY,  
    NOMBRES         VARCHAR   (25) NOT NULL,  
    APELLIDOS        VARCHAR   (25) NOT NULL,  
    DNI_EMPLÉO       CHAR      (8)  NOT NULL,  
    DIRECCIÓN       VARCHAR   (60) NOT NULL,  
    ESTADO_CIVIL     CHAR      (1)  NOT NULL,  
    NIVEL_EDUCA      VARCHAR   (30) NOT NULL,  
    TELÉFONO         VARCHAR   (12) NOT NULL,  
    EMAIL            VARCHAR   (35) NOT NULL,  
    SUELDO_BASICO    MONEY     NOT NULL,  
    FECHA_INGRESO   DATE     NOT NULL,  
    ID_DISTRITO      CHAR      (4)  NOT NULL REFERENCES DISTRITOS,  
    COD_CARGO         CHAR      (3)  NOT NULL REFERENCES CARGOS  
)  
GO
```

```
--TABLA CLIENTE
```

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

```
CREATE TABLE CLIENTE
```

```
(  
    ID_CLIENTE        CHAR      (6) NOT NULL PRIMARY KEY,  
    NOMBRES          VARCHAR   (25) NOT NULL,  
    APELLIDOS         VARCHAR   (25) NOT NULL,  
    DIRECCIÓN        VARCHAR   (60) NULL,  
    FONO              CHAR      (9)  NULL,  
    ID_DISTRITO       CHAR      (4)  NOT NULL REFERENCES DISTRITOS,  
    EMAIL             VARCHAR   (35) NULL  
)  
GO
```

```
--TABLA PRODUCTO
```

```
IF OBJECT_ID('PRODUCTO') IS NOT NULL  
BEGIN  
    DROP TABLE PRODUCTO  
END
```

```
CREATE TABLE PRODUCTO
```

```
(  
    ID_PRODUCTO       CHAR      (6) NOT NULL PRIMARY KEY,  
    DESCRIPCION        VARCHAR   (45) NOT NULL,  
    PRECIO_VENTA      MONEY     NOT NULL,
```

```

STOCK_MINIMO      INT          NULL,
STOCK_ACTUAL      INT          NULL,
FECHA_VENC        DATE         NULL,
COD_CATE          CHAR(3)     NOT NULL REFERENCES CATEGORIAS
)

```

GO

--TABLA BOLETA

IF OBJECT_ID('BOLETA') IS NOT NULL

BEGIN

DROP TABLE BOLETA

END

CREATE TABLE BOLETA

(

NUM_BOLETA	CHAR(8)	NOT NULL PRIMARY KEY,
FECHA_EMI	DATE	NOT NULL,
ID_CLIENTE	CHAR(6)	NOT NULL REFERENCES CLIENTE,
COD_EMPLA	CHAR(5)	NOT NULL REFERENCES EMPLEADO,
ESTADO_BOLETA	VARCHAR(25)	NOT NULL

)

GO

--TABLA DETALLE_BOLETA

IF OBJECT_ID ('DETALLEBOLETA') IS NOT

NULL BEGIN

DROP TABLE DETALLEBOLETA

END

CREATE TABLE DETALLEBOLETA

(

NUM_BOLETA	CHAR(8)	NOT NULL REFERENCES BOLETA,
ID_PRODUCTO	CHAR(6)	NOT NULL REFERENCES PRODUCTO,
CANTIDAD	INT	NOT NULL,
IMPORTE	MONEY	NOT NULL,

PRIMARY KEY (NUM_BOLETA, ID_PRODUCTO)

)

GO

Tener en cuenta:

Definir las llaves primarias y foráneas según se muestra en la imagen.

Definir los siguientes valores por defecto:

- Asignar el valor cero al stock mínimo y stock actual de la tabla producto.
- Asignar el valor "AC" al estado de la boleta de la tabla boleta.
- Asignar el valor "No registra" al correo electrónico (Email) de la tabla cliente.

Asignar el valor “000000000” al teléfono de la tabla cliente.

Definir las siguientes restricciones:

El precio de venta del producto debe ser mayor a cero.

El estado de la boleta solo debe permitir los valores AC(activo) y AN(anulado).

La fecha de emisión de la boleta debe ser mayor a la fecha actual.

El campo Estado civil de los empleados solo debe permitir los valores C (Casado), S (Soltero), T (Conviviente), D (Divorciado), V (Viudo)

El campo nivel de educación de los empleados debe permitir los valores (Primaria, Secundaria, Superior, Universitario).

El sueldo básico de los empleados debe ser como mínimo 850.00

El campo código de la tabla empleado debe empezar con la letra **E**

El campo código de la tabla categoría debe empezar con la letra **C**

Definir las siguientes restricciones:

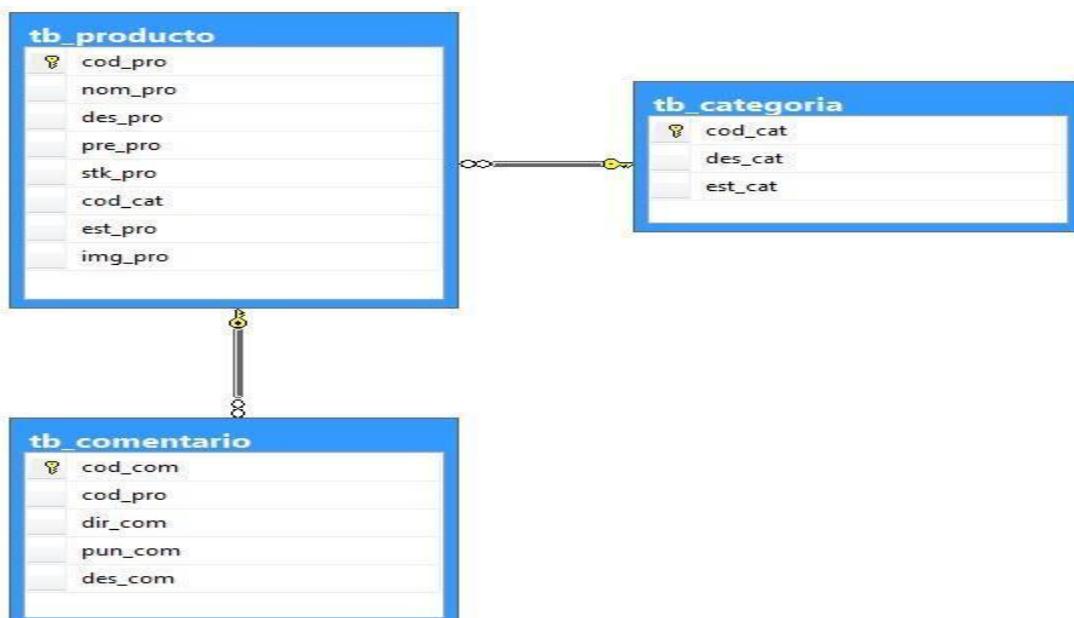
Los nombres de los distritos deben ser valores únicos.

La descripción del producto deben ser valores únicos.

- El DNI de la tabla empleado deben ser valores únicos.
- El campo dirección de la tabla empleado deben ser valores únicos.

Caso propuesto: PORTAL

Implemente la base de datos **Portal2017** a partir del siguiente diagrama de base de datos:



Debe tener en cuenta los siguientes aspectos:

Defina como identity a los campos código de categoría (cod_cat) y código de comentario (cod_com).

Definir las llaves primarias y foráneas según se muestra en la imagen.

Definir los siguientes valores por defecto:

- Asignar el valor '01' al estado de la categoría (est_cat).
- Asignar el valor “NO REGISTRA IMAGEN” a la imagen del producto (img_pro).
- Asignar el valor 0 al stock del producto(stk_pro).

Definir las siguientes restricciones:

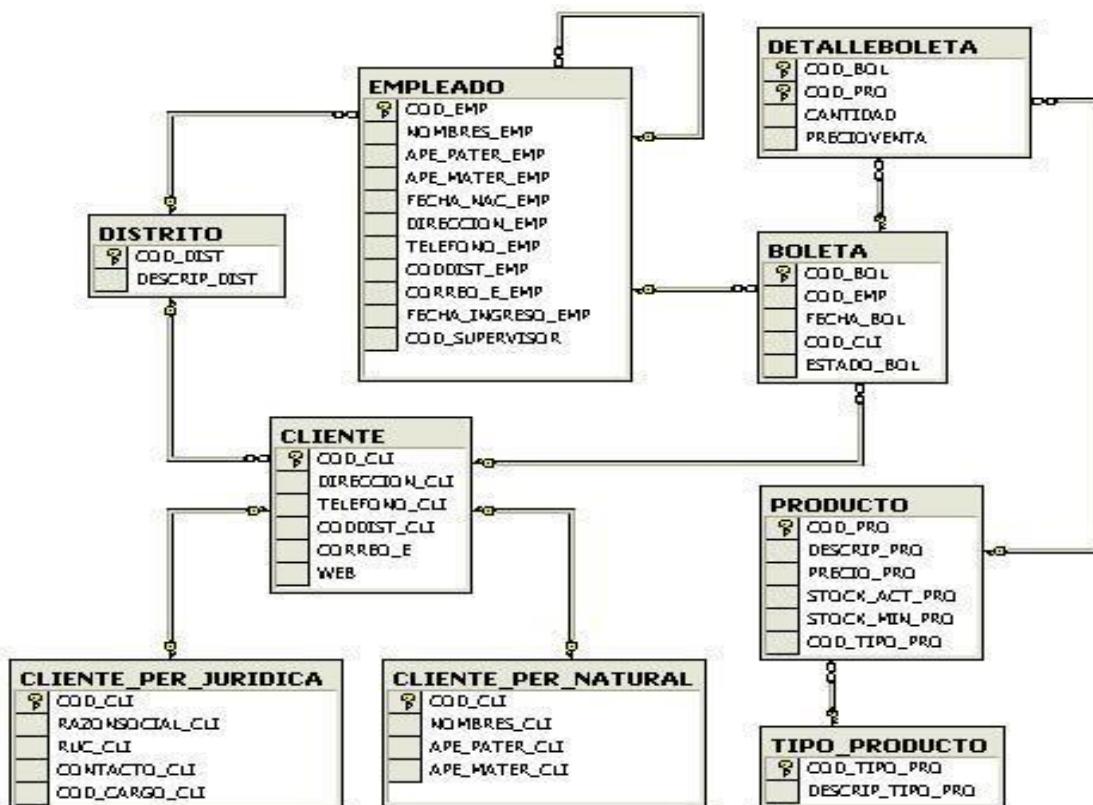
- El precio del producto (pre_pro) debe ser mayor a cero.
- El estado de la categoría solo debe permitir los valores como 01(activo), 02(desactivo) y 03(No categorizado).

Definir las siguientes restricciones:

- El nombre del producto (nom_pro) deben ser valores únicos.

CASO PROPUESTO: COMERCIAL ANGULO SAC

Implemente un script de SQL Server que permita crear la base de datos **BD_COMERCIALANGULO**. Para ello se cuenta con el siguiente diagrama:

**Usando TRANSACT/SQL, realice lo siguiente:**

Cree la base de datos **BD_COMERCIALANGULO**

Active la base de datos **BD_COMERCIALANGULO**

Cree las tablas mostradas

Agregue las llaves Primarias (ADD PRIMARY KEY)

Agregue las llaves Foráneas, Relaciones (ADD FOREIGN KEY – REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

Asigne de manera correcta los valores nulos y no nulos según corresponda.

Visualice el diagrama implementado en SQL Server.

Restricciones:

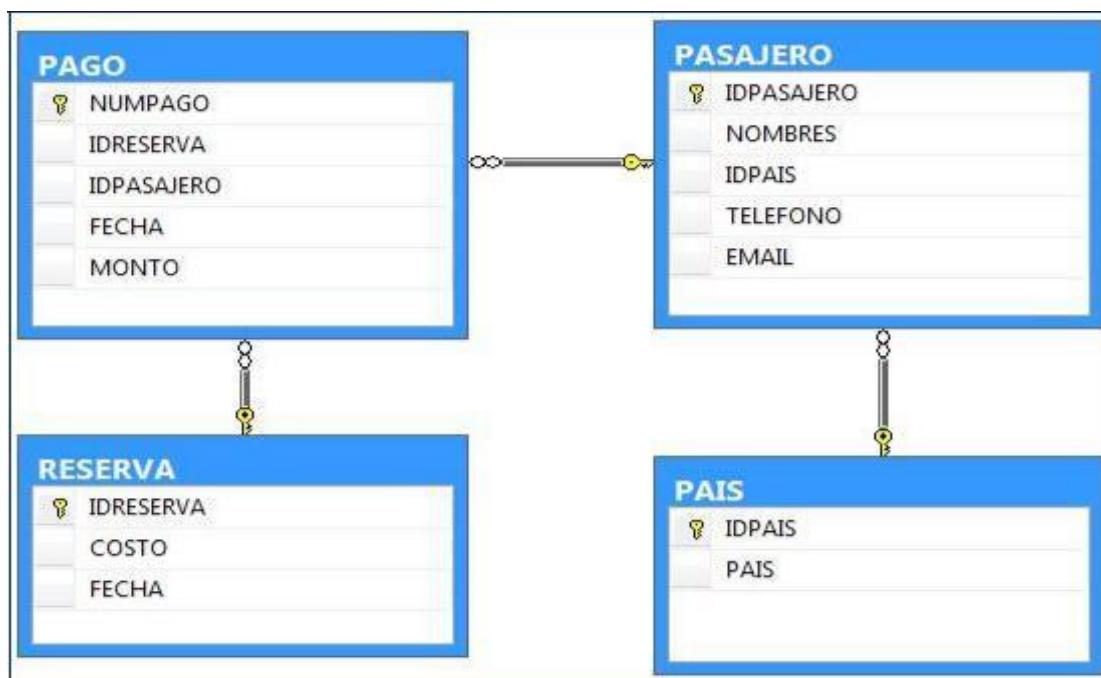
Utilizando su propio criterio deberá aplicar restricciones a los campos de las tablas, considere:

4 restricciones para default

- 6 restricciones para check
- 4 restricciones para unique
- 2 restricciones para identity

5. CASO PROPUESTO: AGENCIA DE VIAJES

Implemente la base de datos **AGENCIA2017** a partir del siguiente diagrama de base de datos:



Debe tener en cuenta los siguientes aspectos:

Defina IDENTITY al campo numPago de la tabla PAGO.

Definir las llaves primarias y foráneas según se muestra en la imagen.

Definir los siguientes constraint:

El nombre del país debe contener valores únicos.

La fecha actual debe ser un valor por defecto de la columna fecha de reserva.

La fecha de pago debe mayor a la fecha actual.

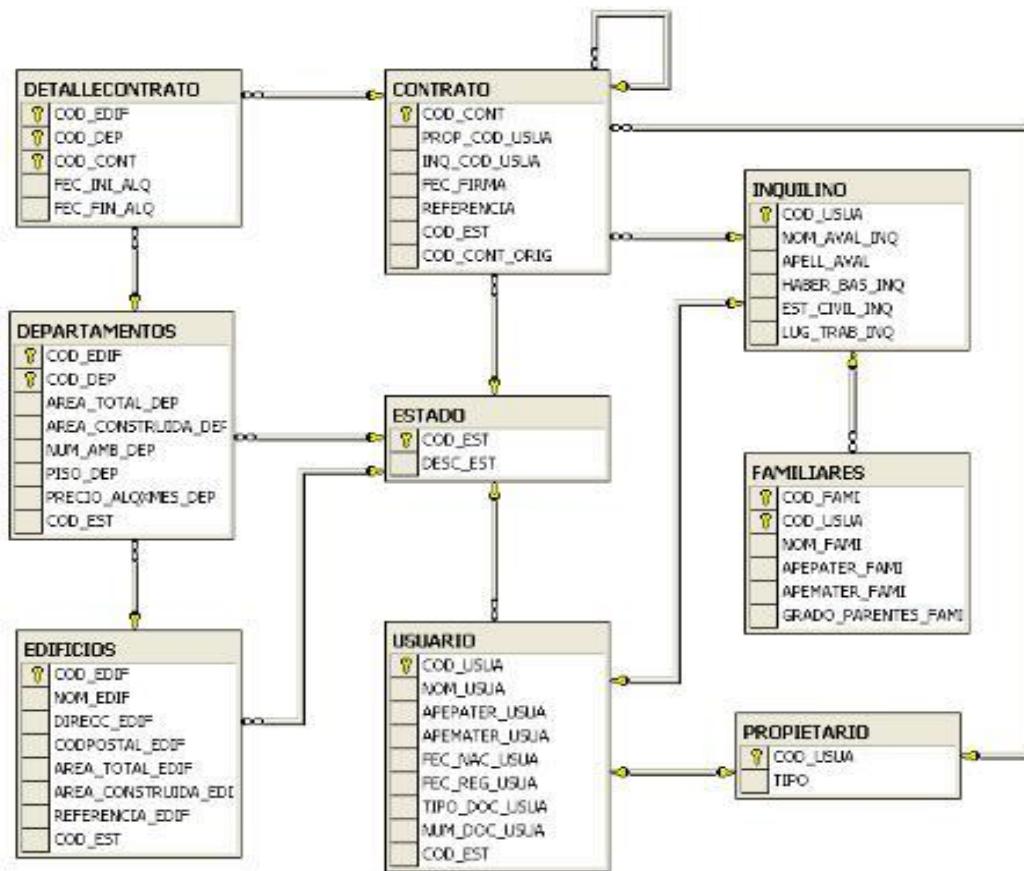
El monto debe permitir registrar valores superiores a cero.

El email del pasajero no debe permitir el registro de valores duplicados.

- El costo de la reserva debe ser mayor a cero.
- El valor por defecto del teléfono del pasajero es 000-0000.

CASO PROPUESTO: CONTROL DE CONTRATOS

Se desea implementar una base de datos para el control de contratos de departamentos entre diferentes edificios. Para ello se cuenta con el siguiente diagrama:



Usando TRANSACT/SQL, realice lo siguiente:

Cree la base de datos **BD_COMERCIALANGULO**

Active la base de datos **BD_COMERCIALANGULO**

Cree las tablas mostradas

Agregue las llaves Primarias (ADD PRIMARY KEY)

Agregue las llaves Foráneas, Relaciones (ADD FOREIGN KEY – REFERENCES)

Asigne de manera correcta los tipos de datos a cada uno de los campos de la tabla.

Asigne de manera correcta los valores nulos y no nulos según corresponda.

Visualice el diagrama implementado en SQL Server.

Restricciones:

Utilizando su propio criterio deberá aplicar restricciones a los campos de las tablas, considere:

4 restricciones para default

6 restricciones para check

4 restricciones para unique

2 restricciones para identity

Resumen

Recuerde que SQL Server es un sistema de administración de bases de datos relacionales (RDBMS: Relational Database Management System) Cliente/Servidor de alto rendimiento y se ha diseñado para admitir un elevado volumen de procesamiento de transacciones (como las de entrada de pedidos en línea, inventario, facturación o contabilidad), además de aplicaciones de almacén de datos y de ayuda en la toma de decisiones (como aplicaciones de análisis de ventas) sobre redes basadas en el sistema operativo Microsoft.

Cuando se crea una nueva base de datos, por defecto se generan dos archivos mdf (datos) y ldf (registro).

Los tipos de datos definen el valor de datos que se permite en cada columna.

SQL Server proporciona varios tipos de datos diferentes.

Recuerde siempre que las tablas son el corazón de las bases de datos relacionales en general y de SQL Server en particular. Las restricciones de integridad aseguran que la clave primaria identifique únicamente a cada entidad representada de la base de datos y además aseguran que las relaciones entre entidades de la base de datos se preserven durante las actualizaciones.

Las restricciones se aplican a reglas de negocio especificadas en el análisis del proceso de negocio.

La cláusula **DEFAULT** permite definir un valor por defecto a una determinada columna de una tabla, es así que se definen valores predeterminados en el conjunto de valores de una tabla.

La cláusula **CHECK** permite definir una condición de restricción al registrar los valores dentro de una tabla; estos deberán cumplir los requisitos especificados en los constraint de tipo **CHECK**.

La cláusula **UNIQUE** permite definir un valor único entre un conjunto de valores contenidos en una tabla.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

<http://msdn.microsoft.com/es-es/library/bb500433.aspx>

Instalación del tutorial de SQL Server 2014

<http://www.casdreams.com/auladeinformatica/cet/mansql2.htm>

Conceptos básicos del lenguaje SQL

http://www.academia.edu/9470781/Base_de_Datos

Tutorial sobre la creación de una base de datos en SQL.

<http://technet.microsoft.com/es-es/library/ms175198.aspx>

creación de una base de datos en SQL.

<http://msdn.microsoft.com/es-es/library/ms176061.aspx> Creación de base de datos (Transact-SQL de SQL Server)

<http://technet.microsoft.com/es-es/library/ms174979.aspx>

Tutorial para investigar el comando *Create Table*.

<http://technet.microsoft.com/es-es/library/ms186775.aspx>

Tutorial para investigar la propiedad *Identidad* (*Identity*)

<http://technet.microsoft.com/es-es/library/ms190273.aspx>

Tutorial para investigar el comando *Alter* para tablas que han sido creadas

<http://www.edu4java.com/es/sql/sql4.html>

Sentencias DDL, DML del SQL Server.

[https://technet.microsoft.com/es-es/library/ms190765\(v=sql.105\).aspx](https://technet.microsoft.com/es-es/library/ms190765(v=sql.105).aspx)

Integridad de los datos

<http://www.incanatoit.com/2014/12/lenguaje-sql-lenguaje-ddl-sql-server-2014.html>

En esta página web hallará algunos conceptos de lenguaje SQL, DDL y la implementación de una base de datos en SQL Server 2014.

https://books.google.com.pe/books?id=LvPWdGufkboC&pg=PA5&lpg=PA5&dq=creacion+modificacion+y+eliminacion+de+usuarios+base+de+datos+en+sql+server2014&source=bl&ots=dysdq-7Tnj&sig=I4au7wPuSJ8vm2tNTf_uC4FFNM&hl=es-419&sa=X&ved=0ahUKEwjay_30_PTRAhXCMYKHZEIDnsQ6AEIJzAD#v=o nепage&q&f=false

Es esta página web encontrará el curso completo de SQL Server 2014.



MANIPULACIÓN DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la segunda unidad, el estudiante manipula los datos de las tablas empleando las tres sentencias DML (INSERT, UPDATE y DELETE).

TEMARIO

2.1 Tema 7: Sentencias DML

- 2.1.1 Inserción de datos INSERT
 - 2.1.2 Actualización de datos: UPDATE
 - 2.1.3 Eliminación de datos: DELETE
 - 2.1.4 Integración de sentencias SQL
- Ejercicios propuestos

ACTIVIDADES PROPUESTAS

Emplean comandos SQL para ingresar, modificar o eliminar datos.

Emplean los comandos CREATE y DROP para el uso de índices.

2.1. Sentencias DML

2.1.1. Inserción de datos: INSERT

La sentencia **INSERT** se utiliza para añadir registros a las tablas de la base de datos. El formato de la sentencia es:

```
INSERT INTO Nombre_tabla (columna1, columna2, ..., columnaN)
VALUES (Valor1, Valor2, ..., ValorN)
```

Donde:

Nombre_tabla: Aquí se especifica el nombre de la tabla a la cual agregaremos registros.

Columna: Es la especificación de las columnas que se ingresarán valores; esta especificación puede ser opcional en la medida que se ingresen valores para cada columna de la tabla en estricto orden. Entonces podríamos decir que el formato para la inserción sería:

```
INSERT INTO Nombre_tabla VALUES (Valor1, Valor2, ..., ValorN)
```

Values: Es la cláusula que permite especificar los valores que contendrá una tabla.

Valor: Es la información que contendrá cada columna de la tabla, hay que tener en cuenta los tipos de datos especificados en la creación de la tabla; además de los valores por defecto o la asignación de identitys.

2.1.1.1 Formas de inserción de registros

Antes que nada, debemos indicar que existen 2 formas de insertar filas a una tabla de datos:

Inserción individual de filas

Inserción múltiple de filas

Para los casos que presentaremos usaremos la tabla **EMPLEADO** el cual cuenta con la siguiente estructura:

EMPLEADO	
Column Name	Condensed Type
IDEMPLEADO	int
NOMBRE_EMP	varchar(30)
APELLIDOS_EMP	varchar(30)
TELEFONO_EMP	char(15)
EST_CIVIL_EMP	char(1)
SUELDO_EMP	money
CORREO_EMP	varchar(40)

Figura 35: Tabla Empleado con sus tipos de datos
Fuente. - Tomado desde SQL Server 2014

a) Insertar un registro a la tabla EMPLEADO

Implemente un script que permita insertar un registro a la tabla Empleado con todos los datos especificados en dicha tabla.

```
INSERT INTO EMPLEADO  
VALUES (1, 'Jose', 'Blanco', '525-5685',  
       'C', 1850, 'jblanco@cibertec.edu.pe')
```

Como observamos los valores especificados en la cláusula VALUES se encuentran en el mismo orden que la especificación de las columnas al crear la tabla.

b) Insertar varios registros a la tabla EMPLEADO

Implemente un script que permita insertar dos registros a la tabla Empleado con todos los datos especificados en dicha tabla.

```
INSERT INTO EMPLEADO  
VALUES (2, 'Maria', 'Rengifo', '985-698569',  
       'S', 2500, 'mrenfigo@cibertec.edu.pe')  
INSERT INTO EMPLEADO  
VALUES (3, 'Milagros', 'Acosta', '998-562563',  
       'D', 1100, 'macosta@cibertec.edu.pe')
```

Como observamos la inserción de varios registros en este formato es totalmente idéntico a la especificada en una fila de registro.

c) Insertar un registro a la tabla EMPLEADO especificando un determinado orden en las columnas

Implemente un script que permita insertar un registro en la tabla Empleado con el siguiente orden de campos nombres, apellidos, sueldo, correo electrónico, código y estado civil del empleado.

```
INSERT INTO EMPLEADO (NOMBRE_EMP, APELLIDOS_EMP, SUELDO_EMP,  
                      CORREO_EMP, IDEMPLEADO, EST_CIVIL_EMP)  
VALUES ('Angela', 'Torres', '1800',  
       'atorres@cibertec.edu.pe', 4, 'S')
```

Cuando se especifican las columnas debemos considerar el orden de las mismas, pues los valores también deberán ser ingresados en dicho orden. Como notara en la especificación de las columnas no se consideró al campo **TELÉFONO_EMP**, pues, siendo un campo nulo este se rellenará con NULL tal como se muestra en la figura.

Finalmente, no se olvide que, para comprobar si las inserciones son correctas debe ejecutar la siguiente sentencia:

```
SELECT * FROM EMPLEADO
```

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	1	Jose	Blanco	525-5685	C	1850.00	jblanco@cibertec.edu.pe
2	2	Maria	Rengifo	985-698569	S	2500.00	mrenfigo@cibertec.edu.pe
3	3	Milagros	Acosta	998-562563	D	1100.00	macosta@cibertec.edu.pe
4	4	Angela	Torres	NULL	S	1800.00	atores@cibertec.edu.pe

Figura 55: Visualizando los registros de la tabla EMPLEADO
Fuente. - Tomado de SQL SERVER 2014

d) Inserción de múltiples filas de registro:

Implemente un script que permita insertar dos registros a la tabla Empleados en una sola sentencia:

```
INSERT INTO EMPLEADO VALUES
(5, 'Janeth', 'Rengifo', '985-698569',
 'S', 2500, 'mrenfigo@cibertec.edu.pe'),
(6, 'Milagros', 'Acosta', '998-562563',
 'D', 1100, 'macosta@cibertec.edu.pe')
```

Como observamos en el script para insertar varios registros en este formato debemos separar cada registro por comas, sin la necesidad de colocar la cláusula VALUES ni repetir la sentencia INSERT INTO. Así como es una gran ayuda para el registro masivo de información, también se debe tener cuidado en los valores especificados, pues, si uno de los valores rompe algún criterio; solo se registrarán los valores correctos, las demás líneas no serán consideradas por el servidor; dicho de otra manera, los datos posteriores no se registrarán y se tendrán que colocar en una nueva sentencia de INSERT INTO.

e) Inserción de registros a partir de variables locales

Implemente un script que permita insertar un registro a la tabla Empleado a partir de variables locales.

```
-- Declarando las variables locales
DECLARE @IDE INT, @NOM VARCHAR (30), @APE VARCHAR (30),
@TEL CHAR (15), @EST CHAR (1), @SUE MONEY,
@COR VARCHAR (40)
```

Asignando los valores a las variables

```
SELECT @IDE=7, @NOM='Lucero', @APE='Erazo',
@TEL='985-966858', @EST='S', @SUE=1450,
@COR='lerazo@cibertec.edu.pe'
```

Enviando los valores a la tabla Empleado desde las variables

```
INSERT INTO EMPLEADO
VALUES (@IDE, @NOM, @APE, @TEL, @EST, @SUE, @COR)
```

En el script debemos tener en cuenta que DECLARE permite declarar una variable local y estas siempre empiezan con una arroba (@), la separación de comas de debe a que se pueden declarar más de dos variables desde una misma sentencia DECLARE.

SELECT, permite asignar un valor a una variable local, para lo cual debemos considerar que el valor no rompa la regla de declaración, así tenemos que cuando se envía un número de cualquier tipo; este no debe estar encerrado entre comillas, mientras que los textos o fechas sí.

Finalmente, la sentencia **INSERT INTO** especifica sus valores mediante la invocación de las variables locales en la cláusula **VALUES** en estricto orden; pues no se ha especificado a que columnas se registraría valor.

f) Inserción de registros a partir de una consulta

Implemente un script que permita insertar todos los registros de la tabla Personal a la tabla Empleado.

```
INSERT INTO EMPLEADO
    SELECT * FROM PERSONAL
```

En el script debemos considerar que la sentencia **SELECT * FROM PERSONAL** devuelve un conjunto de registros y que al ejecutar las sentencias **INSERT INTO** y **SELECT** al mismo tiempo; el conjunto de registros se agregará a la tabla **EMPLEADO**.

Observemos el siguiente script que permite registrar cuatro filas a la tabla **PERSONA**:

```
Validando la creación de la tabla
PERSONA IF OBJECT_ID('PERSONA') IS NOT NULL
    DROP TABLE PERSONA
GO

Creando la tabla
CREATE TABLE PERSONA (
    CODIGO_PER INT             NOT NULL PRIMARY KEY,
    NOMBRE_PER  VARCHAR (30)    NOT NULL,
    APELLI_PER  VARCHAR (30)    NOT NULL,
    TELEFO_PER  CHAR (15)      NULL,
    ECIVIL_PER  CHAR (1)       NULL,
    SUELDO_PER  MONEY          NULL,
    CORREO_PER  VARCHAR (40)   NOT NULL
)
GO

Insertando registros
INSERT INTO PERSONA VALUES
    (100, 'Carlos', 'Ramos', '(01)522-1523',
     'S', 1520, 'cramos@gmail.com'),
    (101, 'David', 'Fernandez', '(01)425-5454',
     'C', 2500, 'dfernandez@gmail.com'),
    (102, 'Selena', 'Susaya', '(01)421-1523',
     'V', 2200, 'ssusaya@gmail.com'),
    (103, 'Jhon', 'Hernandez', '(01)485-9588',
```

```
'C', 1800, 'jhernandez@gmail.com'),
(104, 'Carlos', 'Pacheco', '(01)523-1523',
 'C', 1750, 'cpacheco@gmail.com')

GO
```

Finalmente, el resultado de la inserción de los registros desde la tabla Persona a la tabla Empleado se muestra en la siguiente imagen, gracias a la sentencia:

```
SELECT * FROM EMPLEADO
```

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	1	Jose	Blanco	525-5685	C	1850,00	jblanco@cibertec.edu.pe
2	2	Maria	Rengifo	985-698569	S	2500,00	mrenfigo@cibertec.edu.pe
3	3	Milagros	Acosta	998-562563	D	1100,00	macosta@cibertec.edu.pe
4	5	Janeth	Rengifo	985-698569	S	2500,00	mrenfigo@cibertec.edu.pe
5	6	Milagros	Acosta	998-562563	D	1100,00	macosta@cibertec.edu.pe
6	7	Lucero	Erazo	985-966858	S	1450,00	lerazo@cibertec.edu.pe
7	100	Carlos	Ramos	(01)522-1523	S	1520,00	cramos@gmail.com
8	101	David	Fernandez	(01)425-5454	C	2500,00	dfemandez@gmail.com
9	102	Selena	Susaya	(01)421-1523	V	2200,00	ssusaya@gmail.com
10	103	Jhon	Hemandez	(01)485-9588	C	1800,00	jhemandez@gmail.com
11	104	Carlos	Pacheco	(01)523-1523	C	1750,00	cpacheco@gmail.com

Figura 36: Visualizando los registros de la tabla EMPLEADO

Fuente. - Tomado de SQL SERVER 2014

g) Crear una tabla de registros a partir de una consulta

Implemente un script que permita crear una copia de seguridad de la tabla Empleado.

```
SELECT * INTO BK_EMPLEADO FROM EMPLEADO
```

Debemos tener en cuenta que al especificar un asterisco (*) en la sentencia estamos indicando que todas las columnas de la tabla Empleado serán registradas en la nueva tabla llamada BK_EMPLEADO. A partir de aquí podríamos variar la sentencia y especificar que columnas desea enviar a la nueva tabla. Por ejemplo:

```
SELECT IDEMPLEADO, NOMBRE_EMP, APELLIDOS_EMP INTO
BK_EMPLEADO2 FROM EMPLEADO
```

Estas sentencias crean una tabla llamada **BK_EMPLEADO2**, los cuales tienen por columnas el código, nombres y apellidos del empleado y todos los registros de la tabla Empleado.

2.1.2. Actualización de datos: UPDATE

La sentencia **UPDATE** se utiliza para cambiar el contenido de los registros de una tabla de la base de datos. Su formato es:

```
UPDATE NOMBRE_TABLA
SET COLUMNA = NUEVO_VALOR O EXPRESION
WHERE CONDICION
```

Donde:

Nombre_tabla: Aquí se especifica el nombre de la tabla a la cual modificaremos los valores de sus registros.

SET: Es la cláusula que permite asignar un nuevo valor a una determinada columna(s), en caso se quiera actualizar más de dos columnas se deberá separar por comas:

SET COLUMNA=VALOR

SET COLUMNA1=VALOR1, COLUMNA2=VALOR2

WHERE: Es la condición que deben cumplir los registros para una actualización de los mismos; recuerde que una actualización es análoga a una modificación de valores, pero se debe especificar que registros se actualizarán; ya que de otra manera la actualización ocurrirá a todos los registros.

2.1.3. Eliminación de datos: DELETE

La sentencia **DELETE** se utiliza para eliminar registros de una tabla de la base de datos. Hay que considerar que si la fila a suprimir se encuentra asociado a otra tabla; este no podrá ser eliminado hasta que no se borre en la primera tabla; esto es considerado como un problema de integridad referencial.

Su formato es:

```
DELETE FROM NOMBRE_TABLA
WHERE CONDICION
```

Donde:

NOMBRE_TABLA: Aquí se especifica el nombre de la tabla a la cual eliminaremos sus registros.

WHERE: Es la condición que deben cumplir los registros para una eliminación de registros correcto. Debemos considerar que esta cláusula es opcional y no colocar implica la eliminación de todos los registros.

2.1.4. Integración de sentencias SQL

Actividad desarrollada 01:

Crear la base de datos **COMERCIO** a partir del siguiente diagrama de base de datos:

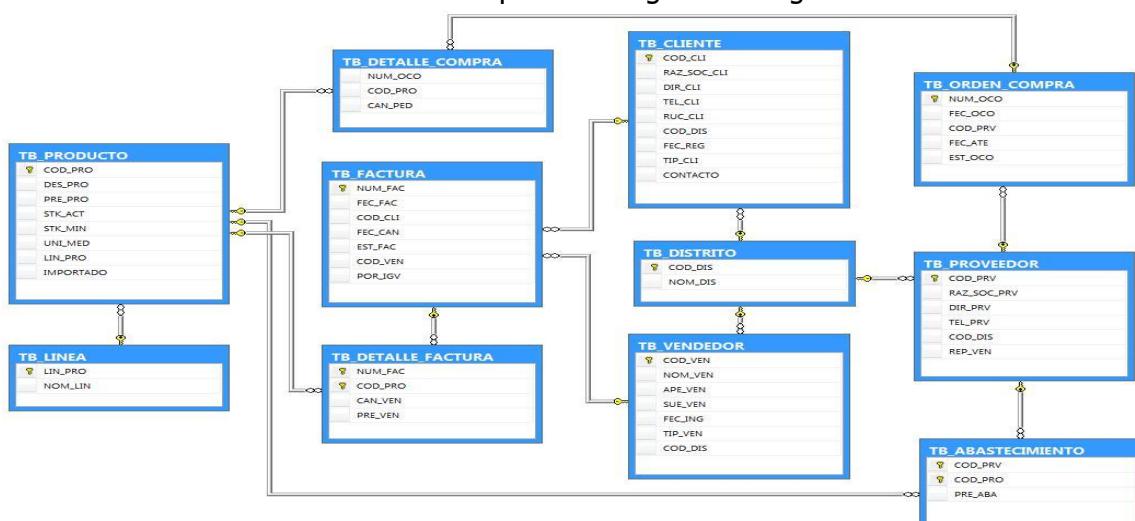


Figura 37: Visualizando el diagrama de base de datos COMERCIO

Fuente. - Tomado de SQL SERVER 2014

Implementación de la base de datos en SQL Server 2014

```
USE MASTER
GO

/* VALIDANDO LA EXISTENCIA DE LA BASE COMERCIO */
IF DB_ID('COMERCIO') IS NOT NULL
    DROP DATABASE COMERCIO
GO

/* CREANDO LA BASE DE DATOS */
CREATE DATABASE COMERCIO
GO

/* ACTIVANDO LA BASE DE DATOS */
USE COMERCIO
GO

/* CREANDO LAS TABLAS */
CREATE TABLE TB_DISTRITO (
    COD_DIS char (3)      NOT NULL,
    NOM_DIS varchar (50)   NULL,
    PRIMARY KEY (COD_DIS)
)
GO

CREATE TABLE TB_LINEA (
    LIN_PRO int      NOT NULL,
    NOM_LIN varchar (30) NOT NULL,
    PRIMARY KEY (LIN_PRO)
)
GO

CREATE TABLE TB_VENDEDOR (
    COD_VEN char (3)      NOT NULL,
    NOM_VEN varchar (20)   NOT NULL,
    APE_VEN varchar (20)   NOT NULL,
    SUE_VEN money          NOT NULL,
    FEC_ING date           NULL,
    TIP_VEN int             NOT NULL,
    COD_DIS char (3)      NOT NULL,
    PRIMARY KEY (COD_VEN)
)
GO
```

```
CREATE TABLE TB_PROVEEDOR (
    COD_PRV char (4)          NOT NULL,
    RAZ_SOC_PRV varchar (80) NOT NULL,
    DIR_PRV varchar (100)      NOT
                               NULL,
    TEL_PRV varchar (15)       NULL,
    COD_DIS char (3)          NOT NULL,
    REP_VEN varchar (80)       NULL,
    PRIMARY KEY (COD_PRV)
)
GO

CREATE TABLE TB_PRODUCTO (
    COD_PRO char (4)          NOT NULL,
    DES_PRO varchar (50)       NOT NULL,
    PRE_PRO decimal (10, 2)    NOT NULL,
    STK_ACT int                NOT NULL,
    STK_MIN int                NOT NULL,
    UNI_MED varchar (30)       NOT NULL,
    LIN_PRO int                NOT NULL,
    IMPORTADO varchar (10)     NOT NULL,
    PRIMARY KEY (COD_PRO)
)
GO

CREATE TABLE TB_CLIENTE (
    COD_CLI char (5)          NOT NULL,
    RAZ_SOC_CLI char (30)      NOT NULL,
    DIR_CLI varchar (100)      NOT NULL,
    TEL_CLI varchar (9)        NOT NULL,
    RUC_CLI varchar (15)       NULL,
    COD_DIS char (3)          NOT NULL,
    FEC_REG date               NULL,
    TIP_CLI int                NULL,
    CONTACTO varchar (30)      NULL,
    PRIMARY KEY (COD_CLI)
)
GO

CREATE TABLE TB_ABASTECIMIENTO (
    COD_PRV char (4)          NOT NULL,
    COD_PRO char (4)          NOT NULL,
    PRE_ABA decimal (7, 2)     NULL
    PRIMARY KEY (COD_PRV, COD_PRO)
)
GO
```

```
CREATE TABLE TB_ORDEN_COMPRA (
    NUM_OCO varchar (5)      NOT NULL,
    FEC_OCO date             NULL,
    COD_PRV char (4)        NOT NULL,
    FEC_ATE date             NULL,
    EST_OCO int              NOT NULL,
    PRIMARY KEY (NUM_OCO)
)
GO

CREATE TABLE TB_FACTURA (
    NUM_FAC int               NOT NULL,
    FEC_FAC date             NULL,
    COD_CLI char (5)        NOT NULL,
    FEC_CAN date             NULL,
    EST_FAC int              NOT NULL,
    COD_VEN char (3)        NOT NULL,
    POR_IGV decimal (4,2)    NULL,
    PRIMARY KEY (NUM_FAC)
)
GO

CREATE TABLE TB_DETALLE_FACTURA (
    NUM_FAC int               NOT NULL,
    COD_PRO char (4)        NOT NULL,
    CAN_VEN int              NOT NULL,
    PRE_VEN money            NOT NULL,
    PRIMARY KEY (NUM_FAC, COD_PRO)
)
GO

CREATE TABLE TB_DETALLE_COMPRA (
    NUM_OCO varchar (5)      NOT NULL,
    COD_PRO char (4)        NOT NULL,
    CAN_PED int              NOT NULL
)
GO

/* LLAVES FORANEAS */
ALTER TABLE TB_ABASTECIMIENTO
    ADD FOREIGN KEY(COD_PRV) REFERENCES TB_PROVEEDOR (COD_PRV)
GO

ALTER TABLE TB_ABASTECIMIENTO
    ADD FOREIGN KEY(COD_PRO) REFERENCES TB_PRODUCTO (COD_PRO)
GO
```

```
ALTER TABLE TB_CLIENTE
    ADD FOREIGN KEY(COD_DIS) REFERENCES TB_DISTRITO (COD_DIS)
GO

ALTER TABLE TB_DETALLE_COMPRA
    ADD FOREIGN KEY(COD_PRO) REFERENCES TB_PRODUCTO (COD_PRO)
GO

ALTER TABLE TB_DETALLE_COMPRA
    ADD FOREIGN KEY(NUM_OCO) REFERENCES TB_ORDEN_COMPRA (NUM_OCO)
GO

ALTER TABLE TB_DETALLE_FACTURA
    ADD FOREIGN KEY(COD_PRO) REFERENCES TB_PRODUCTO (COD_PRO)
GO

ALTER TABLE TB_DETALLE_FACTURA
    ADD FOREIGN KEY(NUM_FAC) REFERENCES TB_FACTURA (NUM_FAC)
GO

ALTER TABLE TB_FACTURA
    ADD FOREIGN KEY(COD_CLI) REFERENCES TB_CLIENTE (COD_CLI)
GO

ALTER TABLE TB_FACTURA
    ADD FOREIGN KEY(COD_VEN) REFERENCES TB_VENDEDOR (COD_VEN)
GO

ALTER TABLE TB_ORDEN_COMPRA
    ADD FOREIGN KEY(COD_PRV) REFERENCES TB_PROVEEDOR (COD_PRV)
GO

ALTER TABLE TB_PRODUCTO
    ADD FOREIGN KEY(LIN_PRO) REFERENCES TB_LINEA (LIN_PRO)
GO

ALTER TABLE TB_PROVEEDOR
    ADD FOREIGN KEY(COD_DIS) REFERENCES TB_DISTRITO (COD_DIS)
GO

ALTER TABLE TB_VENDEDOR
    ADD FOREIGN KEY(COD_DIS) REFERENCES TB_DISTRITO (COD_DIS)
GO
```

2. Aplicando la restricción DEFAULT:

- 2.1 Asigne la fecha actual a la fecha de ingreso al registro (**FEC_ING**) del vendedor; esto tiene por objetivo llenar dicha fecha de registro con la fecha actual obtenido desde el sistema.

```
ALTER TABLE TB_VENDEDOR
    ADD CONSTRAINT DEF_FECHAINF
        DEFAULT GETDATE() FOR FEC_ING
GO

--PRUEBAS DE INSERCIÓN CORRECTA
INSERT TB_VENDEDOR VALUES ('V01', 'JUANA', 'MESES',
                           1000.00, GETDATE(), '1', 'D08')

INSERT TB_VENDEDOR VALUES ('V02', 'JUAN', 'SOTO', 1200.00,
                           '2014-02-05', '2', 'D03')

INSERT TB_VENDEDOR (COD_VEN, NOM_VEN, APE_VEN,
                   SUE_VEN, TIP_VEN, COD_DIS)
VALUES ('V03', 'CARLOS', 'AREVALO', 1500.00, '2', 'D09')
```

- 2.2 Asigne el valor por defecto “**000-000000**” al teléfono del proveedor (**TEL_PRV**); esto permitirá registrar un valor cuando el proveedor aun no registra su teléfono actual.

```
ALTER TABLE TB_PROVEEDOR
    ADD CONSTRAINT DEF_TELEFONO
        DEFAULT '000-000000' FOR TEL_PRV
GO

--PRUEBA DE INSERCIÓN CORRECTA
INSERT TB_PROVEEDOR VALUES ('PR01', 'Faber Castell',
                            'Av. Isabel La Católica 1875',
                            DEFAULT, 'D13', 'Carlos Aguirre')

INSERT TB_PROVEEDOR VALUES ('PR02', 'Atlas',
                            'Av. Lima 471',
                            '5380926', 'D13', 'Cesar Torres')

INSERT TB_PROVEEDOR (COD_PRV, RAZ_SOC_PRV,
                     DIR_PRV, COD_DIS, REP_VEN)
VALUES ('PR03', '3M', 'Av. Venezuela 3018',
       'D16', 'Omar Injoque')
```

- 2.3 Asigne el valor por defecto “**no registra**” al nombre de representante (**REP_VEN**) en la tabla Proveedor; esto permitirá no dejar vacío la columna del nombre

del representante y determinar en un futuro que proveedores aun no registran el nombre de su representante.

```

ALTER TABLE TB_PROVEEDOR
    ADD CONSTRAINT DEF_REPRESENTANTE
        DEFAULT 'NO REGISTRA' FOR REP_VEN
GO

--PRUEBA DE INSERCIÓN CORRECTA
INSERT TB_PROVEEDOR VALUES ('PR01', 'Faber Castell',
                             'Av. Isabel La Católica 1875',
                             '4280112', 'D12', DEFAULT)

INSERT TB_PROVEEDOR VALUES ('PR02', 'Atlas ',
                           'Av. Lima 471',
                           '5380926', 'D13', 'Cesar Torres')

INSERT TB_PROVEEDOR (COD_PRV, RAZ_SOC_PRV,
                     DIR_PRV, TEL_PRV, COD_DIS)
VALUES ('PR03', '3M ', 'Av. Venezuela 3018',
       '4258596', 'D16')

```

2.4 Asigne el valor “**VERDADERO**” al campo importado (**IMPORTADO**) de la tabla Producto.

```

ALTER TABLE TB_PRODUCTO
    ADD CONSTRAINT DEF_IMPORTADO
        DEFAULT 'VERDADERO' FOR IMPORTADO
GO

--PRUEBA DE INSERCIÓN CORRECTA

INSERT TB_PRODUCTO VALUES ('P001', 'Papel Bond A-4',
                           35.00, 200, 1500, 'M11', 2, DEFAULT)

INSERT TB_PRODUCTO VALUES ('P002', 'Papel Bond Oficio', 35.00,
                           50, 1500, 'M11', 2, 'FALSO')

INSERT TB_PRODUCTO (COD_PRO, DES_PRO, PRE_PRO,
                     STK_ACT, STK_MIN, UNI_MED, LIN_PRO)
VALUES ('P003', 'Papel Bulky ', 10.00, 498, 1000, 'M11', 2)

```

2.5 Asigne el valor **0.18** a la columna porcentaje de **IGV (POR_IGV)** de la tabla Factura.

```

ALTER TABLE TB_FACTURA
    ADD CONSTRAINT DEF_IGV
        DEFAULT 0.18 FOR POR_IGV
GO

--PRUEBA DE INSERCIÓN CORRECTA
INSERT TB_FACTURA VALUES (1, '1998-06-07', 'C001', '1998-
05-08', 2, 'V01', DEFAULT)

INSERT TB_FACTURA VALUES (2, '1998-06-09', 'C019', '1998-
05-08', 3, 'V02', '0.18')

INSERT TB_FACTURA (FEC_FAC, COD_CLI,
                   FEC_CAN, EST_FAC, COD_VEN)
VALUES (3, '1998-01-09', 'C003',
       '1998-03-11', 2, 'V04')

```

3. Aplicando la restricción CHECK:

3.1 Asigne una restricción al sueldo del vendedor (**SUE_VEN**) de tal forma que solo permita ingresar valores positivos mayores a cero.

```

ALTER TABLE TB_VENDEDOR
    ADD CONSTRAINT CHK_SUELDO
        CHECK (SUE_VEN>=750)
GO

--INSERCIÓN CORRECTA
INSERT TB_VENDEDOR VALUES ('V01', 'JUANA', 'MESES', 750.00,
                           '2015-01-15', '1', 'D08')

--INSERCIÓN INCORRECTA
INSERT TB_VENDEDOR VALUES ('V02', 'JUAN', 'SOTO', 550,
                           '2014-02-05', '2', 'D03')

```

3.2 Asigne una restricción a la columna unidad de medida (**UNI_MED**) de tal forma que solo permita el registro de “**MII**”, “**Uni**”, “**Cie**” y “**Doc**”.

```

ALTER TABLE TB_PRODUCTO
    ADD CONSTRAINT CHK_UNIDAD
        CHECK (UNI_MED IN('Mll', 'Uni', 'Cie', 'Doc'))
GO
/* INSERCIÓNES CORRECTAS */
INSERT TB_PRODUCTO VALUES
    ('P001', 'Papel Bond A-4', 35.00, 200, 1500,
     'Mll', 2, 'VERDADERO')
INSERT TB_PRODUCTO VALUES
    ('P002', 'Papel Bond Oficio', 35.00, 50,
     1500, 'Uni', 2, 'FALSO')

```

```

INSERT TB_PRODUCTO VALUES
    ('P003','Papel Bulky ', 10.00, 498, 1000,
     'Cie', 2, 'VERDADERO')
INSERT TB_PRODUCTO VALUES
    ('P004','Papel Periódico', 9.00, 4285,
     1000, 'Doc', 2, 'FALSO')

/* INSERCIÓN INCORRECTA */
INSERT TB_PRODUCTO VALUES
    ('P005','Cartucho Tinta Negra', 40.00, 50,
     30, 'Cen', '1', 'FALSO')

```

3.3 Asigne una restricción a la columna tipo de cliente (**TIP_CLI**) de tal forma que solo permita el registro de “1” y “0”.

```

ALTER TABLE TB_CLIENTE
    ADD CONSTRAINT CHK_TIPO
    CHECK (TIP_CLI IN (1,2))
GO

INSERT TB_CLIENTE VALUES
    ('C001', 'Finseth', 'Av. Los Viñedos 150', '4342318',
     '48632081', 'D05', '1991-12-10', 1, 'Alicia Barreto')

INSERT TB_CLIENTE VALUES
    ('C002', 'Orbi', 'Av. Emilio Cavenecia 225', '4406335',
     '57031642', 'D04', '1990-02-01', 2, 'Alfonso Beltran')

INSERT TB_CLIENTE VALUES
    ('C003', 'Serviems', 'Jr. Collagate 522', '75012403',
     NULL, 'D05', '1995-06-03', 3, 'Christian Laguna')

```

4. Aplicando la restricción UNIQUE:

4.1 Asigne una restricción a la columna descripción del producto de tal forma que no permite registrar valores repetidos en la descripción.

```

ALTER TABLE TB_PRODUCTO
    ADD CONSTRAINT UNI_DESCRIPCION UNIQUE (DES_PRO)
GO

/* INSERCIÓN CORRECTA */
INSERT TB_PRODUCTO VALUES
    ('P001','Papel Bond A-4',35.00,200,1500,
     'M11',2,'VERDADERO')

/* INSERCIÓN INCORRECTA */
INSERT TB_PRODUCTO VALUES
    ('P002','Papel Bond A-4', 35.00, 50, 1500,
     'M11',2, 'FALSO')

```

4.2 Asigne una restricción a la columna razón social del proveedor de tal forma que no permite valores repetidos en la razón social del proveedor.

```
ALTER TABLE TB_PROVEEDOR
    ADD CONSTRAINT UNI_RAZON UNIQUE (RAZ_SOC_PRV)
GO
/* INserCIÓN CORRECTA */
INSERT TB_PROVEEDOR VALUES ('PR01', 'Faber Castell',
                            'Av. Isabel La Católica 1875',
                            '4330895', 'D13', 'Carlos Aguirre')
/* INserCIÓN INCORRECTA */
INSERT TB_PROVEEDOR VALUES ('PR02', 'Faber Castell',
                            'Av. Lima 471',
                            '5380926', 'D13', 'Cesar Torres')
```

5. Aplicando la restricción IDENTITY:

5.1 Asigne una restricción a la columna número de factura de tal forma que se registre valores numéricos enteros consecutivos desde el número cien.

```
CREATE TABLE TB_FACTURA (
    NUM_FAC int          NOT NULL IDENTITY (100,1),
    FEC_FAC date         NULL,
    COD_CLI char (5)    NOT NULL,
    FEC_CAN date         NULL,
    EST_FAC int          NOT NULL,
    COD_VEN char (3)    NOT NULL,
    POR_IGV decimal (4,2) NULL,
    PRIMARY KEY (NUM_FAC)
)
GO
```

Usando la sentencia INSERT

```
/* INSERTANDO REGISTROS */
INSERT TB_DISTRITO VALUES ('D01', 'Surco')
INSERT TB_DISTRITO VALUES ('D02', 'Jesús María')
INSERT TB_DISTRITO VALUES ('D03', 'San Isidro')
INSERT TB_DISTRITO VALUES ('D04', 'La Molina')
INSERT TB_DISTRITO VALUES ('D05', 'San Miguel')
INSERT TB_DISTRITO VALUES ('D06', 'Miraflores')
INSERT TB_DISTRITO VALUES ('D07', 'Barranco')
INSERT TB_DISTRITO VALUES ('D08', 'Chorrillos')
INSERT TB_DISTRITO VALUES ('D09', 'San Borja')
INSERT TB_DISTRITO VALUES ('D10', 'Lince')
INSERT TB_DISTRITO VALUES ('D11', 'Breña')
```

```

INSERT TB_DISTRITO VALUES ('D12', 'Magdalena')
INSERT TB_DISTRITO VALUES ('D13', 'Rimac')
INSERT TB_DISTRITO VALUES ('D14', 'Surquillo')
INSERT TB_DISTRITO VALUES ('D15', 'Pueblo Libre')
INSERT TB_DISTRITO VALUES ('D16', 'Bellavista')
INSERT TB_DISTRITO VALUES ('D17', 'Callao')
INSERT TB_DISTRITO VALUES ('D18', 'San Martin de Porres')
INSERT TB_DISTRITO VALUES ('D19', 'Santa Anita')
INSERT TB_DISTRITO VALUES ('D20', 'Los Olivos')
INSERT TB_DISTRITO VALUES ('D21', 'Independencia')
INSERT TB_DISTRITO VALUES ('D22', 'Lima - Cercado')
INSERT TB_DISTRITO VALUES ('D24', 'San Luis')
INSERT TB_DISTRITO VALUES ('D25', 'El Agustino')
INSERT TB_DISTRITO VALUES ('D26', 'San Juan de Lurigancho')
INSERT TB_DISTRITO VALUES ('D27', 'Ate - Vitarte')
INSERT TB_DISTRITO VALUES ('D28', 'San Juan de Miraflores')
INSERT TB_DISTRITO VALUES ('D29', 'Carmen de la Legua')
INSERT TB_DISTRITO VALUES ('D30', 'Comas')
INSERT TB_DISTRITO VALUES ('D31', 'Villa María del Triunfo')
INSERT TB_DISTRITO VALUES ('D32', 'Villa el Salvador')
INSERT TB_DISTRITO VALUES ('D33', 'La Perla')
INSERT TB_DISTRITO VALUES ('D34', 'Ventanilla')
INSERT TB_DISTRITO VALUES ('D35', 'Puente Piedra')
INSERT TB_DISTRITO VALUES ('D36', 'Carabayllo')
INSERT TB_DISTRITO VALUES ('D37', 'Santa María')
INSERT TB_DISTRITO VALUES ('D38', 'San Guchito')
INSERT TB_DISTRITO VALUES ('D45', 'La Punta')

INSERT TB_LINEA VALUES (1,'ACCESORIOS PC')
INSERT TB_LINEA VALUES (2,'PAPELES')
INSERT TB_LINEA VALUES (3,'UTILES DE OFICINA')

INSERT TB_VENDEDOR VALUES ('V01', 'JUANA', 'MESES',
                           1000.00, '2015-01-15', 1, 'D08')
INSERT TB_VENDEDOR VALUES ('V02', 'JUAN', 'SOTO', 1200.00,
                           '2014-02-05', 2, 'D03')
INSERT TB_VENDEDOR VALUES ('V03', 'CARLOS', 'AREVALO', 1500.00,
                           '2013-03-25', 2, 'D09')
INSERT TB_VENDEDOR VALUES ('V04', 'CESAR', 'OJEDA',
                           1450.00, '2014-05-05', 1, 'D01')
INSERT TB_VENDEDOR VALUES ('V05', 'JULIO', 'VEGA', 1500.00,
                           '2014-01-10', 1, 'D01')
INSERT TB_VENDEDOR VALUES ('V06', 'ANA', 'ORTEGA', 1200,
                           '2015-02-20', 1, 'D05')
INSERT TB_VENDEDOR VALUES ('V07', 'JOSE', 'PALACIOS',
                           1500.00, '2013-03-02', 1, 'D02')
INSERT TB_VENDEDOR VALUES ('V08', 'RUBEN', 'SALAS', 1450.00,
                           '2014-04-15', 1, 'D04')

```

```
'2014-05-07', 2, 'D04'))  
INSERT TB_VENDEDOR VALUES ('V09', 'PATRICIA', 'ARCE',  
    1800.00, '2013-06-28', 2, 'D04')  
INSERT TB_VENDEDOR VALUES ('V10', 'RENATO', 'IRIARTE', 1550.00,  
    '2013-04-16', 2, 'D01')  
  
INSERT TB_PROVEEDOR VALUES ('PR01', 'Faber Castell',  
    'Av. Isabel La Católica 1875',  
    '4330895', 'D13', 'Carlos Vega')  
INSERT TB_PROVEEDOR VALUES ('PR02', 'Atlas',  
    'Av. Lima 471',  
    '5380926', 'D13', 'Cesar Torres')  
INSERT TB_PROVEEDOR VALUES ('PR03', '3M',  
    'Av. Venezuela 3018',  
    '2908165', 'D16', 'Omar Injoque')  
INSERT TB_PROVEEDOR VALUES ('PR04', 'Dito',  
    'Av. Metropolitana 376',  
    NULL, 'D19', 'Ramón Flores')  
INSERT TB_PROVEEDOR VALUES ('PR05', 'Acker',  
    'Calle Las Dunas 245',  
    '4780143', 'D27', 'Julio Acuña')  
INSERT TB_PROVEEDOR VALUES ('PR06', 'Deditec',  
    'Calle Pichincha 644',  
    '5662848', 'D11', 'Javier Luna')  
INSERT TB_PROVEEDOR VALUES ('PR07', 'Officetec',  
    'Calle Las Perdices 225 Of. 204',  
    '4216184', 'D03', 'Carlos Robles')  
INSERT TB_PROVEEDOR VALUES ('PR08', 'Invicta',  
    'Av. Los Frutales 564',  
    '4364247', 'D27', 'Alberto Rojas')  
INSERT TB_PROVEEDOR VALUES ('PR09', 'Dipropor',  
    'Av. Del Aire 901',  
    '4742046', 'D24', 'Roberto Roca')  
INSERT TB_PROVEEDOR VALUES ('PR10', 'Miura',  
    'Av. La Paz 257',  
    '4459710', 'D06', 'Jorge Vásquez')  
INSERT TB_PROVEEDOR VALUES ('PR11', 'Praxis',  
    'Av. José Gálvez 1820 - 1844',  
    '4703944', 'D10', 'Ericka Zela')  
INSERT TB_PROVEEDOR VALUES ('PR12', 'Sumigraf',  
    'Av. Manco Cápac 754',  
    '3320343', 'D13', 'Karina Rios')  
INSERT TB_PROVEEDOR VALUES ('PR13', 'Limmsa',  
    'Prolg. Huaylas 670',  
    '2546995', 'D08', 'Laura Ortega')  
INSERT TB_PROVEEDOR VALUES ('PR14', 'Veninsa',  
    'Av. Tejada 338',
```

```
'2473832', 'D07', 'Melisa Ramos'))  
INSERT TB_PROVEEDOR VALUES ('PR15', 'Crosland',  
    'Av. Argentina 3206 - 3250',  
    '4515272', 'D17', 'Juan Ramirez')  
INSERT TB_PROVEEDOR VALUES ('PR16', 'Petramas',  
    'Calle Joaquín Madrid 141 2do P',  
    NULL, 'D09', 'Rocío Guerrero')  
  
INSERT TB_PROVEEDOR  
VALUES  
( 'PR17', 'Reawse', 'Av. Santa Rosa 480',  
    NULL, 'D19', 'María Pérez'),  
( 'PR18', 'Edusa', 'Av. Morales Duarez 1260',  
    '4525536', 'D29', 'Pablo Martel'),  
( 'PR19', 'Ottmer', 'Urb.Pro Mz B6 Lt 16',  
    '5369893', 'D18', 'Angela Rendilla'),  
( 'PR20', 'Bari', 'Av. Arnaldo Marquez 1219',  
    NULL, 'D02', 'Vanesa Quintana')  
  
INSERT TB_PRODUCTO VALUES  
( 'P001', 'Papel Bond A-4', 35.00, 200, 1500,  
    'M11', 2, 'VERDADERO'),  
( 'P002', 'Papel Bond Oficio', 35.00, 50,  
    1500, 'M11', 2, 'FALSO'),  
( 'P003', 'Papel Bulky ', 10.00, 498,  
    1000, 'M11', 2, 'VERDADERO'),  
( 'P004', 'Papel Periódico', 9.00, 4285,  
    1000, 'M11', 2, 'FALSO'),  
( 'P005', 'Cartucho Tinta Negra', 40.00, 50,  
    30, 'Uni', 1, 'FALSO'),  
( 'P006', 'Cartucho Tinta Color', 45.00, 58,  
    35, 'Uni', 1, 'FALSO'),  
( 'P007', 'Porta Diskettes', 3.50, 300,  
    100, 'Uni', 1, 'VERDADERO'),  
( 'P008', 'Caja de Diskettes * 10 ', 30.00, 125,  
    180, 'Uni', 1, 'FALSO'),  
( 'P009', 'Borrador de Tinta', 10.00, 100,  
    500, 'Doc', 3, 'FALSO'),  
( 'P010', 'Borrador Blanco', 8.00, 2000,  
    400, 'Doc', 3, 'FALSO'),  
( 'P011', 'Tajador Metal', 20.00, 1120,  
    300, 'Doc', 3, 'FALSO'),  
( 'P012', 'Tajador Plástico', 12.00, 608,  
    300, 'Doc', 3, 'FALSO'),  
( 'P013', 'Folder Manila Oficio', 20.00, 200,  
    150, 'Cie', 3, 'FALSO'),  
( 'P014', 'Folder Manila A-4', 20.00, 150, 150,
```

```

('Cie',3, 'VERDADERO'),
('P015','Sobre Manila Oficio', 15.00, 300,
130, 'Cie',3, 'FALSO'),
('P016','Sobre Manila A-4', 18.00, 200,
100, 'Cie',3, 'FALSO'),
('P017','Lapicero Negro', 10.00, 3000,
1000, 'Doc',3, 'FALSO'),
('P018','Lapicero Azul ', 10.00, 2010,
1500, 'Doc',3, 'FALSO'),
('P019','Lapicero Rojo', 8.00, 1900,
1000, 'Doc',3, 'VERDADERO'),
('P020','Folder Plástico A-4', 50.00, 3080,
1100, 'Cie',3, 'FALSO'),
('P021','Protector de Pantalla', 50.00, 20,
5, 'Uni',1, 'FALSO')

```

INSERT TB_CLIENTE VALUES

```

('C001', 'Finseth', 'Av. Los Viñedos 150', '4342318',
'48632081', 'D05', '1991-12-10', 1, 'Alicia Barreto'),
('C002', 'Orbi', 'Av. Emilio Cavenecia 225', '4406335',
'57031642', 'D04', '1990-02-01', 2, 'Alfonso Beltran'),
('C003', 'Serviemsa', 'Jr. Collagate 522', '75012403',
NULL, 'D05', '1995-06-03', 2, 'Christian Laguna'),
('C004', 'Issa', 'Calle Los Aviadores 263', '3725910',
'46720159', 'D01', '1992-09-12', 1, 'Luis Apumayta'),
('C005', 'Mass', 'Av. Tomas Marsano 880', '4446177',
'83175942', 'D14', '1992-10-01', 1, 'Katia Armejo'),
('C006', 'Berker', 'Av. Los Proceres 521', '3810322',
'54890124', 'D05', '1989-07-05', 1, 'Judith Aste'),
('C007', 'Fidenza', 'Jr. El Niquel 282', '5289034',
'16204790', 'D20', '1991-10-02', 2, 'Héctor Vivanco'),
('C008', 'Intech', 'Av. San Luis 2619 5to P', '2249493',
'34021824', 'D09', '1997-01-07', 2, 'Carlos Villanueva'),
('C009', 'Prominent', 'Jr. Iquique 132', '43233519',
NULL, 'D11', '1993-06-11', 1, 'Jorge Valdivia'),
('C010', 'Landu', 'Av. Nicolás de Ayllon 1453', '3267840',
'30405261', 'D05', '1989-11-08', 2, 'Raquel Espinoza'),
('C011', 'Filasur', 'Av. El Santuario 1189', '4598175',
'70345201', 'D26', '1990-03-09', 1, 'Angélica Vivas'),
('C012', 'Sucerte', 'Jr. Grito de Huaura 114', '4206434',
'62014503', 'D05', '1990-10-05', 1, 'Karina Vega'),
('C013', 'Hayashi', 'Jr. Ayacucho 359', '42847990',
NULL, 'D22', '1993-06-11', 2, 'Ernesto Uehara'),
('C014', 'Kadia', 'Av. Santa Cruz 1332 Of.201', '4412418',
'22202915', 'D06', '1995-05-04', 1, 'Miguel Arce'),
('C015', 'Meba', 'Av. Elmer Faucett 1638', '4641234',
'50319542', 'D16', '1993-05-12', 2, 'Ricardo Gomez')

```

```
('C016', 'Cardeli', 'Jr. Bartolome Herrera 451', '2658853',
 '26403158', 'D10', '1991-12-03', 2, 'Giancarlo Bonifaz'),
('C017', 'Payet', 'Calle Juan Fanning 327', ' 4779834',
 '70594032', 'D07', '1993-05-12', 1, 'Paola Uribe'),
('C018', 'Dasin', 'Av. Saenz Peña 338 - B', ' 4657574',
 '35016752', 'D17', '1991-12-03', 1, 'Ángela Barreto'),
('C019', 'Corefo', 'Av. Canadá 3894 - 3898', '4377499',
 '57201691', 'D24', '1998-01-03', 2, 'Rosalyn Cortez'),
('C020', 'Cramer', 'Jr. Mariscal Miller 1131', '4719061',
 '46031783', 'D02', '1996-08-11', 1, 'Christopher Ramos')

INSERT TB_ABASTECIMIENTO VALUES ('PR01', 'P003', 8.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR01', 'P005', 35.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR01', 'P007', 3.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR01', 'P009', 8.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR01', 'P011', 18.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR02', 'P002', 30.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR02', 'P007', 3.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR03', 'P002', 32.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR03', 'P004', 7.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR04', 'P001', 28.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR04', 'P006', 40.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR05', 'P018', 9.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR06', 'P009', 7.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR06', 'P017', 8.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR07', 'P016', 15.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR07', 'P019', 7.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR08', 'P006', 42.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR08', 'P010', 6.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR09', 'P002', 30.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR09', 'P014', 17.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR11', 'P001', 27.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR11', 'P006', 44.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR12', 'P002', 33.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR12', 'P010', 7.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR13', 'P005', 35.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR14', 'P016', 15.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR15', 'P020', 45.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR16', 'P008', 25.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR16', 'P012', 9.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR16', 'P013', 15.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR19', 'P008', 28.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR19', 'P016', 16.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR20', 'P012', 10.00)
INSERT TB_ABASTECIMIENTO VALUES ('PR20', 'P020', 43.00)
```

INSERT TB_ORDEN_COMPRA VALUES

```
('OC001', '1998-05-03', 'PR08', '1998-12-03', 1),
('OC002', '1998-08-04', 'PR16', '1998-10-04', 1),
('OC003', '1998-02-08', 'PR10', '1998-02-08', 3),
('OC004', '1998-05-04', 'PR01', '1998-05-04', 3),
('OC005', '1998-06-03', 'PR07', '1998-10-03', 1),
('OC006', '1998-02-01', 'PR19', '1998-02-01', 1),
('OC007', '1998-06-02', 'PR20', '1998-05-04', 3),
('OC008', '1998-02-06', 'PR04', '1998-01-07', 1),
('OC009', '1998-03-08', 'PR11', '1998-10-09', 1),
('OC010', '1998-05-09', 'PR01', '1998-05-09', 1),
('OC011', '1998-02-10', 'PR03', '1998-03-10', 1),
('OC012', '1998-04-10', 'PR14', '1998-05-10', 3),
('OC013', '1998-02-11', 'PR05', '1998-06-11', 1),
('OC014', '1998-03-11', 'PR19', '1998-05-12', 1),
('OC015', '1998-03-11', 'PR18', '1998-10-12', 1),
('OC016', '1998-06-12', 'PR06', '1998-06-12', 3),
('OC017', '1999-08-01', 'PR09', '1999-08-01', 1),
('OC018', '1999-01-02', 'PR20', '1999-08-02', 1),
('OC019', '1999-03-03', 'PR11', '1999-06-03', 1),
('OC020', '1999-07-10', 'PR12', '1999-08-13', 1),
('OC021', '1999-08-30', 'PR14', '1999-09-13', 1),
('OC022', '1999-06-14', 'PR05', '1999-08-14', 2)
```

INSERT TB_FACTURA VALUES

```
('1998-06-07','C001', '1998-05-08', 2, 'V01', '0.19'),
('1998-06-09','C019', '1998-05-08', 3, 'V02', '0.19'),
('1998-01-09','C003', '1998-03-11', 2, 'V04', '0.19'),
('1998-06-09','C016', '1998-05-11', 2, 'V07', '0.19'),
('1998-01-10','C015', '1998-12-10', 2, 'V08', '0.19'),
('1998-10-10','C009', '1998-05-08', 3, 'V05', '0.19'),
('1998-05-10','C019', '1998-05-08', 1, 'V09', '0.19'),
('1998-09-10','C012', '1998-06-11', 2, 'V10', '0.19'),
('1998-03-10','C008', '1998-11-11', 2, 'V09', '0.19'),
('1998-10-01','C017', '1998-06-11', 2, 'V02', '0.19'),
('1998-10-11','C019', '1998-01-12', 2, 'V05', '0.19'),
('1998-01-12','C014', '1998-01-12', 1, 'V04', '0.19'),
('1998-01-12','C011', '1998-01-12', 3, 'V08', '0.19'),
('1998-03-12','C020', '1998-01-12', 2, 'V09', '0.19'),
('1998-08-12','C015', '1999-06-01', 2, 'V07', '0.19'),
('1999-06-01','C016', '1999-09-01', 2, 'V05', '0.19'),
('1999-06-01','C015', '1999-06-01', 1, 'V06', '0.19'),
('1999-05-02','C016', '1999-04-02', 3, 'V10', '0.19'),
('1999-07-02','C008', '1999-01-03', 3, 'V03', '0.19'),
('1999-06-02','C013', '1999-10-03', 2, 'V02', '0.19'),
('1999-02-07','C011', '1999-02-23', 1, 'V01', '0.19')
```

```
INSERT TB_DETALLE_FACTURA VALUES (100, 'P007', 6, 5)
INSERT TB_DETALLE_FACTURA VALUES (100, 'P011', 25, 25)
INSERT TB_DETALLE_FACTURA VALUES (100, 'P013', 11, 20)
INSERT TB_DETALLE_FACTURA VALUES (102, 'P004', 8, 10)
INSERT TB_DETALLE_FACTURA VALUES (103, 'P002', 10, 40)
INSERT TB_DETALLE_FACTURA VALUES (103, 'P011', 6, 20)
INSERT TB_DETALLE_FACTURA VALUES (103, 'P017', 21, 12)
INSERT TB_DETALLE_FACTURA VALUES (103, 'P019', 12, 10)
INSERT TB_DETALLE_FACTURA VALUES (104, 'P004', 3, 10)
INSERT TB_DETALLE_FACTURA VALUES (104, 'P009', 50, 5)
INSERT TB_DETALLE_FACTURA VALUES (105, 'P003', 20, 10)
INSERT TB_DETALLE_FACTURA VALUES (105, 'P006', 50, 50)
INSERT TB_DETALLE_FACTURA VALUES (105, 'P020', 5, 60)
INSERT TB_DETALLE_FACTURA VALUES (106, 'P002', 20, 35)
INSERT TB_DETALLE_FACTURA VALUES (106, 'P003', 15, 10)
INSERT TB_DETALLE_FACTURA VALUES (106, 'P009', 12, 5)
INSERT TB_DETALLE_FACTURA VALUES (107, 'P003', 20, 12)
INSERT TB_DETALLE_FACTURA VALUES (107, 'P012', 4, 12)
INSERT TB_DETALLE_FACTURA VALUES (108, 'P004', 15, 9)
INSERT TB_DETALLE_FACTURA VALUES (108, 'P008', 15, 30)
INSERT TB_DETALLE_FACTURA VALUES (108, 'P020', 50, 50)
INSERT TB_DETALLE_FACTURA VALUES (109, 'P001', 5, 30)
INSERT TB_DETALLE_FACTURA VALUES (109, 'P002', 15, 35)
INSERT TB_DETALLE_FACTURA VALUES (109, 'P006', 2, 50)
INSERT TB_DETALLE_FACTURA VALUES (109, 'P019', 1, 8)
INSERT TB_DETALLE_FACTURA VALUES (110, 'P002', 3, 35)
INSERT TB_DETALLE_FACTURA VALUES (111, 'P002', 20, 35)
INSERT TB_DETALLE_FACTURA VALUES (112, 'P002', 3, 35)
INSERT TB_DETALLE_FACTURA VALUES (112, 'P006', 1, 50)
INSERT TB_DETALLE_FACTURA VALUES (113, 'P002', 130, 35)
INSERT TB_DETALLE_FACTURA VALUES (113, 'P003', 5, 12)
INSERT TB_DETALLE_FACTURA VALUES (113, 'P015', 4, 12)
INSERT TB_DETALLE_FACTURA VALUES (114, 'P009', 10, 8)
INSERT TB_DETALLE_FACTURA VALUES (115, 'P008', 5, 30)
INSERT TB_DETALLE_FACTURA VALUES (115, 'P016', 3, 18)
INSERT TB_DETALLE_FACTURA VALUES (116, 'P006', 15, 50)
INSERT TB_DETALLE_FACTURA VALUES (116, 'P008', 2, 30)
INSERT TB_DETALLE_FACTURA VALUES (117, 'P002', 120, 40)
INSERT TB_DETALLE_FACTURA VALUES (117, 'P005', 120, 40)
INSERT TB_DETALLE_FACTURA VALUES (118, 'P003', 4, 12)
INSERT TB_DETALLE_FACTURA VALUES (118, 'P005', 6, 40)
INSERT TB_DETALLE_FACTURA VALUES (119, 'P002', 150, 40)
INSERT TB_DETALLE_FACTURA VALUES (119, 'P003', 6, 10)
INSERT TB_DETALLE_FACTURA VALUES (119, 'P006', 2, 45)
INSERT TB_DETALLE_FACTURA VALUES (119, 'P008', 10, 30)
INSERT TB_DETALLE_FACTURA VALUES (120, 'P009', 120, 10)
INSERT TB_DETALLE_FACTURA VALUES (120, 'P015', 5, 15)
```

```
INSERT TB_DETALLE_COMPRA VALUES ('OC001', 'P006', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC001', 'P016', 20)
INSERT TB_DETALLE_COMPRA VALUES ('OC002', 'P003', 200)
INSERT TB_DETALLE_COMPRA VALUES ('OC002', 'P005', 500)
INSERT TB_DETALLE_COMPRA VALUES ('OC003', 'P005', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC004', 'P009', 10)
INSERT TB_DETALLE_COMPRA VALUES ('OC004', 'P013', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC005', 'P007', 150)
INSERT TB_DETALLE_COMPRA VALUES ('OC005', 'P008', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC008', 'P002', 10)
INSERT TB_DETALLE_COMPRA VALUES ('OC008', 'P012', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC009', 'P009', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC009', 'P011', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC010', 'P001', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC011', 'P008', 5)
INSERT TB_DETALLE_COMPRA VALUES ('OC011', 'P016', 10)
INSERT TB_DETALLE_COMPRA VALUES ('OC012', 'P007', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC012', 'P011', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC013', 'P013', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC014', 'P004', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC014', 'P008', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC014', 'P020', 50)
INSERT TB_DETALLE_COMPRA VALUES ('OC016', 'P015', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC017', 'P012', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC017', 'P014', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC019', 'P006', 100)
INSERT TB_DETALLE_COMPRA VALUES ('OC020', 'P005', 500)
INSERT TB_DETALLE_COMPRA VALUES ('OC020', 'P011', 100)

/* LISTANDO LOS REGISTROS */
SELECT * FROM TB_CLIENTE
SELECT * FROM TB_ABASTECIMIENTO
SELECT * FROM TB_DETALLE_COMPRA
SELECT * FROM TB_DETALLE_FACTURA
SELECT * FROM TB_DISTRITO
SELECT * FROM TB_FACTURA
SELECT * FROM TB_LINEA
SELECT * FROM TB_ORDEN_COMPRA
SELECT * FROM TB_PRODUCTO
SELECT * FROM TB_PROVEEDOR
SELECT * FROM TB_VENDEDOR
```

7. Actualización de datos UPDATE

- 7.1 Actualizar el sueldo de todos los vendedores de tal forma que se **aumente en S/. 100 a todos los vendedores.**

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1000.00	2015-01-15	1	D08
2	V02	JUAN	SOTO	1200.00	2014-02-05	2	D03
3	V03	CARLOS	AREVALO	1500.00	2013-03-25	2	D09
4	V04	CESAR	OJEDA	1450.00	2014-05-05	1	D01
5	V05	JULIO	VEGA	1500.00	2014-01-10	1	D01
6	V06	ANA	ORTEGA	1200.00	2015-02-20	1	D05
7	V07	JOSE	PALACIOS	1500.00	2013-03-02	1	D02
8	V08	RUBEN	SALAS	1450.00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1800.00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1550.00	2013-04-16	2	D01

Figura 38: Listado de vendedores inicial
Fuente. - Tomado desde SQL Server 2014

```
UPDATE TB_VENDEDOR
SET SUE_VEN+=100
GO
```

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1100.00	2015-01-15	1	D08
2	V02	JUAN	SOTO	1300.00	2014-02-05	2	D03
3	V03	CARLOS	AREVALO	1600.00	2013-03-25	2	D09
4	V04	CESAR	OJEDA	1550.00	2014-05-05	1	D01
5	V05	JULIO	VEGA	1600.00	2014-01-10	1	D01
6	V06	ANA	ORTEGA	1300.00	2015-02-20	1	D05
7	V07	JOSE	PALACIOS	1600.00	2013-03-02	1	D02
8	V08	RUBEN	SALAS	1550.00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1900.00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1650.00	2013-04-16	2	D01

Figura 39: Listado de vendedores después de la actualización de su sueldo
Fuente. - Tomado desde SQL Server 2014

- 7.2 El estado peruano decide reducir la asignación del **IGV a 18%**, por tal motivo debe actualizar el porcentaje de **IGV** de la tabla Factura a **0.18**.

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	100	1998-06-07	C001	1998-05-08	2	V01	0.19
2	101	1998-06-09	C019	1998-05-08	3	V02	0.19
3	102	1998-01-09	C003	1998-03-11	2	V04	0.19
4	103	1998-06-09	C016	1998-05-11	2	V07	0.19
5	104	1998-01-10	C015	1998-12-10	2	V08	0.19
6	105	1998-10-10	C009	1998-05-08	3	V05	0.19
7	106	1998-05-10	C019	1998-05-08	1	V09	0.19
8	107	1998-09-10	C012	1998-06-11	2	V10	0.19
9	108	1998-03-10	C008	1998-11-11	2	V09	0.19
10	109	1998-10-01	C017	1998-06-11	2	V02	0.19
11	110	1998-10-11	C019	1998-01-12	2	V05	0.19
12	111	1998-01-12	C014	1998-01-12	1	V04	0.19
13	112	1998-01-12	C011	1998-01-12	3	V08	0.19
14	113	1998-03-12	C020	1998-01-12	2	V09	0.19
15	114	1998-08-12	C015	1999-06-01	2	V07	0.19
16	115	1999-06-01	C016	1999-09-01	2	V05	0.19
17	116	1999-06-01	C015	1999-06-01	1	V06	0.19
18	117	1999-05-02	C016	1999-04-02	3	V10	0.19
19	118	1999-07-02	C008	1999-01-03	3	V03	0.19
20	119	1999-06-02	C013	1999-10-03	2	V02	0.19
21	120	1999-02-07	C011	1999-02-23	1	V01	0.19

Figura 40: Listado de facturas inicial
Fuente. - Tomado desde SQL Server 2014

```
UPDATE TB_FACTURA
SET POR_IGV=0.18
GO
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	100	1998-06-07	C001	1998-05-08	2	V01	0.18
2	101	1998-06-09	C019	1998-05-08	3	V02	0.18
3	102	1998-01-09	C003	1998-03-11	2	V04	0.18
4	103	1998-06-09	C016	1998-05-11	2	V07	0.18
5	104	1998-01-10	C015	1998-12-10	2	V08	0.18
6	105	1998-10-10	C009	1998-05-08	3	V05	0.18
7	106	1998-05-10	C019	1998-05-08	1	V09	0.18
8	107	1998-09-10	C012	1998-06-11	2	V10	0.18
9	108	1998-03-10	C008	1998-11-11	2	V09	0.18
10	109	1998-10-01	C017	1998-06-11	2	V02	0.18
11	110	1998-10-11	C019	1998-01-12	2	V05	0.18
12	111	1998-01-12	C014	1998-01-12	1	V04	0.18
13	112	1998-01-12	C011	1998-01-12	3	V08	0.18
14	113	1998-03-12	C020	1998-01-12	2	V09	0.18
15	114	1998-08-12	C015	1999-06-01	2	V07	0.18
16	115	1999-06-01	C016	1999-09-01	2	V05	0.18
17	116	1999-06-01	C015	1999-06-01	1	V06	0.18
18	117	1999-05-02	C016	1999-04-02	3	V10	0.18
19	118	1999-07-02	C008	1999-01-03	3	V03	0.18
20	119	1999-06-02	C013	1999-10-03	2	V02	0.18
21	120	1999-02-07	C011	1999-02-23	1	V01	0.18

Figura 41: Listado de facturas después de actualizar el IGV

Fuente. - Tomado desde SQL Server 2014

7.3 Actualizar el tipo de vendedor a 3 solo a los vendedores cuyo tipo sea 2.

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1000.00	2015-01-15	1	D08
2	V02	JUAN	SOTO	1200.00	2014-02-05	2	D03
3	V03	CARLOS	AREVALO	1500.00	2013-03-25	2	D09
4	V04	CESAR	OJEDA	1450.00	2014-05-05	1	D01
5	V05	JULIO	VEGA	1500.00	2014-01-10	1	D01
6	V06	ANA	ORTEGA	1200.00	2015-02-20	1	D05
7	V07	JOSE	PALACIOS	1500.00	2013-03-02	1	D02
8	V08	RUBEN	SALAS	1450.00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1800.00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1550.00	2013-04-16	2	D01

Figura 42: Listado inicial de la tabla Vendedor

Fuente. - Tomado desde SQL Server 2014

```
UPDATE TB_VENDEDOR
SET TIP_VEN=3
WHERE TIP_VEN=1
GO
```

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1000.00	2015-01-15	3	D08
2	V02	JUAN	SOTO	1200.00	2014-02-05	2	D03
3	V03	CARLOS	AREVALO	1500.00	2013-03-25	2	D09
4	V04	CESAR	OJEDA	1450.00	2014-05-05	3	D01
5	V05	JULIO	VEGA	1500.00	2014-01-10	3	D01
6	V06	ANA	ORTEGA	1200.00	2015-02-20	3	D05
7	V07	JOSE	PALACIOS	1500.00	2013-03-02	3	D02
8	V08	RUBEN	SALAS	1450.00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1800.00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1550.00	2013-04-16	2	D01

Figura 43: Listado de vendedores después de la actualización del tipo de vendedor

Fuente. - Tomado desde SQL Server 2014

7.4 Actualizar el estado de las órdenes de compra a 2 solo para las órdenes cuyo código de proveedor sea “PR01”.

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC001	1998-05-03	PR08	1998-12-03	1
2	OC002	1998-08-04	PR16	1998-10-04	1
3	OC003	1998-02-08	PR10	1998-02-08	3
4	OC004	1998-05-04	PR01	1998-05-04	3
5	OC005	1998-06-03	PR07	1998-10-03	1
6	OC006	1998-02-01	PR19	1998-02-01	1
7	OC007	1998-06-02	PR20	1998-05-04	3
8	OC008	1998-02-06	PR04	1998-01-07	1
9	OC009	1998-03-08	PR11	1998-10-09	1
10	OC010	1998-05-09	PR01	1998-05-09	1
11	OC011	1998-02-10	PR03	1998-03-10	1
12	OC012	1998-04-10	PR14	1998-05-10	3
13	OC013	1998-02-11	PR05	1998-06-11	1
14	OC014	1998-03-11	PR19	1998-05-12	1
15	OC015	1998-03-11	PR18	1998-10-12	1
16	OC016	1998-06-12	PR06	1998-06-12	3
17	OC017	1999-08-01	PR09	1999-08-01	1
18	OC018	1999-01-02	PR20	1999-08-02	1
19	OC019	1999-03-03	PR11	1999-06-03	1
20	OC020	1999-07-10	PR12	1999-08-13	1
21	OC021	1999-08-30	PR14	1999-09-13	1
22	OC022	1999-06-14	PR05	1999-08-14	2

Figura 44: Listado inicial de las órdenes de compra

Fuente. - Tomado desde SQL Server 2014

```
UPDATE TB_ORDEN_COMPRA
SET EST_OCO=2
WHERE COD_PRV='PR01'
GO
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC001	1998-05-03	PR08	1998-12-03	1
2	OC002	1998-08-04	PR16	1998-10-04	1
3	OC003	1998-02-08	PR10	1998-02-08	3
4	OC004	1998-05-04	PR01	1998-05-04	2
5	OC005	1998-06-03	PR07	1998-10-03	1
6	OC006	1998-02-01	PR19	1998-02-01	1
7	OC007	1998-06-02	PR20	1998-05-04	3
8	OC008	1998-02-06	PR04	1998-01-07	1
9	OC009	1998-03-08	PR11	1998-10-09	1
10	OC010	1998-05-09	PR01	1998-05-09	2
11	OC011	1998-02-10	PR03	1998-03-10	1
12	OC012	1998-04-10	PR14	1998-05-10	3
13	OC013	1998-02-11	PR05	1998-06-11	1
14	OC014	1998-03-11	PR19	1998-05-12	1
15	OC015	1998-03-11	PR18	1998-10-12	1
16	OC016	1998-06-12	PR06	1998-06-12	3
17	OC017	1999-08-01	PR09	1999-08-01	1
18	OC018	1999-01-02	PR20	1999-08-02	1
19	OC019	1999-03-03	PR11	1999-06-03	1
20	OC020	1999-07-10	PR12	1999-08-13	1
21	OC021	1999-08-30	PR14	1999-09-13	1
22	OC022	1999-06-14	PR05	1999-08-14	2

Figura 45: Listado de órdenes de compra después de la actualización del estado
Fuente. - Tomado desde SQL Server 2014

7.5 Se decide realizar una **reducción del 20% a los precios** de los productos siempre y cuando sean productos no importados.

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P001	Papel Bond A-4	35.00	200	1500	Mil	2	VERDADERO
2	P002	Papel Bond Oficio	35.00	50	1500	Mil	2	FALSO
3	P003	Papel Bulky	10.00	498	1000	Mil	2	VERDADERO
4	P004	Papel Periódico	9.00	4285	1000	Mil	2	FALSO
5	P005	Cartucho Tinta Negra	40.00	50	30	Uni	1	FALSO
6	P006	Cartucho Tinta Color	45.00	58	35	Uni	1	FALSO
7	P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO
8	P008	Caja de Diskettes * 10	30.00	125	180	Uni	1	FALSO
9	P009	Borrador de Tinta	10.00	100	500	Doc	3	FALSO
10	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
11	P011	Tajador Metal	20.00	1120	300	Doc	3	FALSO
12	P012	Tajador Plástico	12.00	608	300	Doc	3	FALSO
13	P013	Folder Manila Oficio	20.00	200	150	Cie	3	FALSO
14	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
15	P015	Sobre Manila Oficio	15.00	300	130	Cie	3	FALSO
16	P016	Sobre Manila A-4	18.00	200	100	Cie	3	FALSO
17	P017	Lapicero Negro	10.00	3000	1000	Doc	3	FALSO
18	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
19	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO
20	P020	Folder Plástico A-4	50.00	3080	1100	Cie	3	FALSO
21	P021	Protector de Pantalla	50.00	20	5	Uni	1	FALSO

Figura 46: Listado inicial de productos
Fuente. - Tomado desde SQL Server 2014

```
UPDATE TB_PRODUCTO
SET PRE_PRO=PRE_PRO*0.8
WHERE IMPORTADO='FALSO'
GO
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P001	Papel Bond A-4	35.00	200	1500	Mil	2	VERDADERO
2	P002	Papel Bond Oficio	28.00	50	1500	Mil	2	FALSO
3	P003	Papel Bulky	10.00	498	1000	Mil	2	VERDADERO
4	P004	Papel Periódico	7.20	4285	1000	Mil	2	FALSO
5	P005	Cartucho Tinta Negra	32.00	50	30	Uni	1	FALSO
6	P006	Cartucho Tinta Color	36.00	58	35	Uni	1	FALSO
7	P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO
8	P008	Caja de Diskettes * 10	24.00	125	180	Uni	1	FALSO
9	P009	Borador de Tinta	8.00	100	500	Doc	3	FALSO
10	P010	Borador Blanco	6.40	2000	400	Doc	3	FALSO
11	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
12	P012	Tajador Plástico	9.60	608	300	Doc	3	FALSO
13	P013	Folder Manila Oficio	16.00	200	150	Cie	3	FALSO
14	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
15	P015	Sobre Manila Oficio	12.00	300	130	Cie	3	FALSO
16	P016	Sobre Manila A-4	14.40	200	100	Cie	3	FALSO
17	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
18	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO
19	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO
20	P020	Folder Plástico A-4	40.00	3080	1100	Cie	3	FALSO
21	P021	Protector de Pantalla	40.00	20	5	Uni	1	FALSO

Figura 47: Listado de productos después de la actualización de sus precios

Fuente. - Tomado desde SQL Server 2014

8. Eliminación de registros DELETE:

8.1 Eliminar el detalle especificado al número de **factura número 113**.

```
DELETE TB_DETALLE_FACTURA
WHERE NUM_FAC=113
GO
```

Actividad desarrollada 02:

Para los casos que presentaremos, usaremos la tabla **EMPLEADO**, el cual cuenta con la siguiente estructura:

EMPLEADO	
Column Name	Condensed Type
IDEMPLEADO	int
NOMBRE_EMP	varchar(30)
APELLIDOS_EMP	varchar(30)
TELEFONO_EMP	char(15)
EST_CIVIL_EMP	char(1)
SUELDO_EMP	money
CORREO_EMP	varchar(40)

Figura 48: Tabla Empleado
Fuente. - Tomado desde SQL Server 2014

Inicialmente los registros de los empleados se presentan de la siguiente manera:

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	100	Jose	Blanco	525-5685	C	1850,00	jblanco@cibertec.edu.pe
2	101	Maria	Rengifo	985-698569	S	2500,00	mrenfigo@cibertec.edu.pe
3	102	Milagros	Acosta	998-562563	D	1100,00	macosta@cibertec.edu.pe

Figura 49: Visualizando los registros de la tabla EMPLEADO

Fuente. - Tomado de SQL SERVER 2014

El script en SQL para la implementación de la tabla Empleado es:

```

USE MASTER
GO

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

CREATE DATABASE EMPRESA
GO

USE EMPRESA
GO

Validando la creación de la tabla EMPLEADO IF
OBJECT_ID('EMPLEADO') IS NOT NULL
    DROP TABLE EMPLEADO
GO

Creando la tabla
CREATE TABLE EMPLEADO (
    IDEMPLEADO      INT          PRIMARY KEY,
    NOMBRE_EMP      VARCHAR (30)  NOT NULL,
    APELLIDOS_EMP   VARCHAR (30)  NOT NULL,
    TELEFONO_EMP    CHAR (15)     NULL,
    EST_CIVIL_EMP   CHAR (1)      NULL,
    SUELDO_EMP      MONEY        NULL,
    CORREO_EMP      VARCHAR (40)  NOT NULL
)
GO

Agregando registros válidos
INSERT INTO EMPLEADO
    VALUES (100, 'Jose', 'Blanco',
            '525-5685', 'C', 1850, 'jblanco@cibertec.edu.pe')
INSERT INTO EMPLEADO
    VALUES (102, 'Maria', 'Rengifo',
            '985-698569', 'S', 2500, 'mrenfigo@cibertec.edu.pe')
INSERT INTO EMPLEADO

```

```

VALUES (103, 'Milagros', 'Acosta',
       '998-562563', 'D', 1100, 'macosta@cibertec.edu.pe')
GO

```

Listando los registros de los Empleados

```

SELECT * FROM
EMPLEADO GO

```

Actualice el estado civil de todos los empleados a Soltero.

```

--Actualizando al estado civil Soltero
UPDATE EMPLEADO
    SET EST_CIVIL_EMP='S'

--Listando los registros de los empleados
SELECT * FROM EMPLEADO

```

Para comprobar la actualización use la sentencia SELECT * FROM EMPLEADO

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	100	Jose	Blanco	525-5685	S	1850,00	jblanco@cibertec.edu.pe
2	101	Maria	Rengifo	985-698569	S	2500,00	mrenfigo@cibertec.edu.pe
3	102	Milagros	Acosta	998-562563	S	1100,00	macosta@cibertec.edu.pe

Figura 50: Visualizando los registros de la tabla EMPLEADO después de la actualización del estado civil **Fuente**. - Tomado de SQL SERVER 2014

Actualice el sueldo de todos los empleados aumentando en 20%, solo aquellos empleados cuyo sueldo sea menor a S/. 1250.00.

```

--Actualizando el aumento del 20%
UPDATE EMPLEADO
    SET SUELDO_EMP=SUELDO_EMP*1.2
    WHERE SUELDO_EMP<=1250
GO

```

--Listando los registros de los empleados

```

SELECT * FROM EMPLEADO

```

Si observamos los registros originales notamos que en la tercera fila el sueldo del empleado es S/. 1100.00; y cumple con el criterio de actualización, por tanto, después de ejecutar la sentencia UPDATE el resultado sería:

	IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1	100	Jose	Blanco	525-5685	C	1850,00	jblanco@cibertec.edu.pe
2	101	Maria	Rengifo	985-698569	S	2500,00	mrenfigo@cibertec.edu.pe
3	102	Milagros	Acosta	998-562563	D	1320,00	macosta@cibertec.edu.pe

Figura 51: Visualizando los registros de la tabla EMPLEADO después de la actualización **Fuente**. - Tomado de SQL SERVER 2014

El resultado sería S/. 1320.00 que es el resultado de aumentar el 20% al primer valor S/. 1100.00.

Actualice el correo electrónico de todos los empleados aumentando el código al inicio del correo más un guion bajo, quedando el primer correo de la siguiente forma:

100_jblanco@cibertec.edu.pe

```
--Actualizando el correo electrónico
UPDATE EMPLEADO
    SET CORREO_EMP=CAST (IDEMPLEADO AS CHAR(3))+'_'+CORREO_EMP
GO
--Listando los registros de los empleados
SELECT * FROM EMPLEADO
```

Solo debemos tener en cuenta que el código del empleado es un valor numero y que para unirlo con el guion bajo y el correo original, este debe ser convertido a char; es por esa razón que convertimos dicho valor con la función **CAST**. Finalmente el resultado sería:

Resultados						
IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1 100	Jose	Blanco	525-5685	C	1850.00	100_jblanco@cibertec.edu.pe
2 101	Maria	Rengifo	985-698569	S	2500.00	101_mrenfigo@cibertec.edu.pe
3 102	Milagros	Acosta	998-562563	D	1320.00	102_macosta@cibertec.edu.pe

Figura 52: Visualizando los registros de la tabla EMPLEADO después de la actualización en el correo electrónico **Fuente**. - Tomado de SQL SERVER 2014

Eliminar todos los registros de los empleados cuyo estado civil sea soltero.

```
--Eliminando a los empleados solteros
DELETE FROM EMPLEADO
    WHERE EST_CIVIL_EMP='S'
GO

--Listando los registros de los empleados
SELECT * FROM EMPLEADO
```

Para comprobar la actualización use la sentencia SELECT * FROM EMPLEADO

Resultados						
IDEMPLEADO	NOMBRE_EMP	APELLIDOS_EMP	TELEFONO_EMP	EST_CIVIL_EMP	SUELDO_EMP	CORREO_EMP
1 100	Jose	Blanco	525-5685	C	1850.00	100_jblanco@cibertec.edu.pe
2 102	Milagros	Acosta	998-562563	D	1320.00	102_macosta@cibertec.edu.pe

Figura 53: Visualizando los registros de la tabla EMPLEADO después de la eliminación **Fuente**. - Tomado de SQL SERVER 2014

Actividades Propuestas

ACTIVIDAD 01: VENTAS

Con la misma Base de datos trabajado en la sesión anterior **Ventas2017**, realizaremos las siguientes operaciones:

Inserción de registros INSERT:

Insertar los siguientes registros a cada tabla de la base de datos **VENTAS2017** usando registros únicos.

Tabla: Categorías

	COD_CATE	NOMBRE
1	C01	ABARROTES
2	C02	LACTEOS
3	C03	FRUTAS
4	C04	BEBIDAS
5	C05	CONDIMENTOS
6	C06	VERDURAS
7	C07	MENESTRAS
8	C08	CEREALES
9	C09	DETERGENTES
10	C10	GOLOSINAS
11	C11	CONSERVAS
12	C12	HELADOS

Tabla: Distritos

	ID_DISTRITO	NOMBRE_DISTRITO
1	D001	COMAS
2	D002	LINCE
3	D003	CERCADO
4	D004	MIRAFLORES
5	D005	LOS OLIVOS
6	D006	S.J.L.
7	D007	S.J.M.
8	D008	INDEPENDENCIA
9	D009	PUENTE PIEDRA
10	D010	ANCÓN

Tabla: Cargos

	COD_CARGO	NOMBRE_CARGO
1	C01	GERENTE
2	C02	ADMINISTRADOR(A)
3	C03	VENDEDOR(A)
4	C04	CONTADOR(A)
5	C05	SECRETARIA

Tabla: Empleados

COD_EMPLE	NOMBRES	APELLIDOS	DNI_EMPLE	DIRECCION	ESTADO_CIVIL	NIVEL_EDUCA	TELEFONO	EMAIL
1 E0001	BRENDA	QUISPE RAMOS	40138140	JR. PACHITEA # 140	C	SUPERIOR	990387487	BRENDDITA@GMAIL.COM
2 E0002	ANGELICA	ANDRADE CERNA	40138356	JR. MANUEL PRADO # 250	C	SECUNDARIA	970315487	ANDRADECERNA@GMAIL.COM
3 E0003	ROSAURA	TARAZONA RIVAS	40457140	JR. LOS ROSALES # 350	S	SECUNDARIA	993315487	TARANIZARIVAS@GMAIL.COM
4 E0004	RUTH	TARDIO HUAMAN	40198740	AV. JOSE BALTA # 1700	S	UNIVERSITARIO	990115487	HUAMANTAR@GMAIL.COM
5 E0005	GLORIA	ZELADA VELA	78538140	AV. CARLOS MARIATEGUI # 5000	C	SUPERIOR	900315487	ZELADAVELA@HOTMAIL.COM
6 E0006	SOLEDAD	ZARATE LOPEZ	35638140	AV. ALFONSO UGARTE # 3420	S	SECUNDARIA	990316687	LOPEZZA@GMAIL.COM
7 E0007	BETSABE	ZAMUDIO SALAS	40134520	JR. URUGUAY # 145	T	SUPERIOR	99038797	ZAMUDIOZALAS@GMAIL.COM
8 E0008	ANETH	ROSALES TINEO	40138222	AV. BOLIVIA # 1500	T	SUPERIOR	990377487	ROSALESTINEO@GMAIL.COM
9 E0009	RAUL	BECERRA VILLA	40133647	AV. CARLOS IZAGUIRRE # 3245	S	SECUNDARIA	990005487	VILLABECE@GMAIL.COM
10 E0010	VICTOR	CANALES HUAMANI	40124140	JR. ANDAHUAYLAS # 354	C	UNIVERSITARIO	997815487	CANALESHUAMANI@GMAIL.COM
11 E0011	ANGEL	PORRAS LINARES	02338140	AV. WIESSE # 2650	T	UNIVERSITARIO	990253878	LINARESPOR@GMAIL.COM
12 E0012	CARLA	URIARTE REGINALDO	40008140	AV. LAS FLORES # 4000	D	SUPERIOR	991254787	REGINALDOURI@GMAIL.COM
13 E0013	CELIA	ESPINAL CHUCO	40130004	AV. MONTENEGRO # 3564	T	SUPERIOR	988025947	CELIAESPINAL@GMAIL.COM
14 E0014	THALIA	ROMANI VELIZ	40133365	AV. MANCO CAPAC # 2560	S	UNIVERSITARIO	988315487	THALIAROMA@HOTMAIL.COM
15 E0015	JUANA	PAREDES PAREDES	40888784	PUE. LOS HUERTOS # 250	C	UNIVERSITARIO	955315487	JUANITAPAE@HOTMAIL.COM

SUELDO_BASICO	FECHA_INGRESO	ID_DISTRITO	COD_CARGO
6000,00	2014-03-02	D005	C02
1500,00	2014-06-05	D005	C03
1500,00	2014-08-05	D002	C03
2500,00	2015-05-03	D003	C04
1500,00	2015-10-10	D005	C03
1500,00	2015-05-04	D001	C03
1500,00	2015-03-03	D006	C03
1500,00	2014-05-04	D006	C03
1500,00	2013-12-12	D003	C03
1500,00	2014-07-03	D002	C03
1500,00	2013-11-03	D004	C03
1500,00	2014-05-12	D004	C03
1000,00	2013-06-03	D008	C03
1500,00	2015-03-03	D007	C03
1500,00	2014-06-06	D002	C03

Tabla: Clientes

ID_CLIENTE	NOMBRES	APELLIDOS	DIRECCION	FONO	ID_DISTRITO	EMAIL
1 CLI001	ANETH LUANA	TINEO URIBE	AV. LOS GIRASOLES # 1800	990990230	D006	LUANITAHERMOSA@GMAIL.COM
2 CLI002	JOSE LUIS	TARAZONA ZELA	AV. LAS FLORES # 1800	NULL	D006	JOSESITO@GMAIL.COM
3 CLI003	ANA MARIA	VILLAVICENCIO CASTRO	AV. LAS FLORES # 2560	989434228	D005	ANITAMARIA@GMAIL.COM
4 CLI004	JOSE ANTONIO	ENCISO NOLASCO	AV. PROCERES DE LA INDEPENDENCIA # 5000	987845874	D006	JOSANTONIO@GMAIL.COM
5 CLI005	ALEJANDRA	CHUCO HUERTA	AV. GRAN CHIMU # 3500	963245874	D005	ALEJANDRITA5245@HOTMAIL.COM
6 CLI006	MARIO	CERPA CARRASCO	AV. JOSE CARLOS MARIATEGUI # 3600	NULL	D003	MARIO56345@GMAIL.COM
7 CLI007	ANA TERESA	ANTON PALOMINO	JR. BOLOGNESI # 850	987546321	D007	ANITATERESITA@GMAIL.COM
8 CLI008	MANUEL	VASQUEZ LICAS	PJE. LOS HUEROS # 450	987564789	D007	MANUEL9878@YAHOO.COM
9 CLI009	CARLOS	VEGA REYES	AV. BRASIL # 3562	965487545	D008	CARLITOSVEGUITA@HOTMAIL.COM
10 CLI010	LUANA	VILLA RAMOS	AV. LA VILLA DEL MAR # 2563	965884578	D010	LUANITAVILLA@GMAIL.COM

Tabla: Producto

ID_PRODUCTO	DESCRIPCION	PRECIO_VENTA	STOCK_MINIMO	STOCK_ACTUAL	FECHA_VENC	COD_CATE
1 PRO001	ARROZ COSTENO X 50 - SACO	160,00	5	200	2018-03-04	C01
2 PRO002	AZUCAR RUBIA X 50 - SACO	100,00	2	80	2018-05-05	C01
3 PRO003	FIDEOS ANITA X 12 - BOLSA	30,00	3	150	2018-10-08	C01
4 PRO004	FIDEOS MOLITALIA X 12 - BOLSA	35,00	5	120	2018-08-08	C01
5 PRO005	YOGURT GLORIA X 6 - PAQUETE	26,00	5	230	2019-08-25	C02
6 PRO006	LECHE GLORIA X 48 - CAJA	98,00	5	240	2018-08-03	C02
7 PRO007	ACEITE PRIMOR X 12 - CAJA	75,00	10	180	2020-10-10	C01
8 PRO008	LENTEJAS X 50 - SACO	230,00	5	50	2018-02-02	C07
9 PRO009	PALLARES X 50 - SACO	290,00	2	30	2018-01-02	C07
10 PRO010	ATUN A1 X 48 - CAJA	145,00	2	80	2020-12-12	C01
11 PRO011	GALLETA CHARADA X 100 - BOLSA	72,00	5	160	2018-11-30	C10
12 PRO012	GASEOSA COCACOLA X 24 - CAJA	24,00	2	50	2018-10-06	C04
13 PRO013	DETERGENTE ARIEL X KILO - BOLSA	15,00	2	40	2019-08-07	C09
14 PRO014	COMINO X 100 - CAJA	10,00	5	300	2018-03-02	C05
15 PRO015	GASEOSA PEPSI X 12 MEDIANA- PAQUETE	20,00	5	50	2018-03-02	C04
16 PRO016	DOS CABALLOS X 6 - CAJA	28,00	5	45	2019-03-02	C11
17 PRO017	CEREALES ANGEL X KILO - BOLSA	18,00	5	70	2019-03-02	C05
18 PRO018	QUAKER X KILO - BOLSA	7,00	5	300	2018-04-02	C01
19 PRO019	PEZIDURI X LITRO - POTE	8,50	5	300	2017-08-08	C12
20 PRO020	ARROZ PAISANA SUPERIOR x 5kg- Bolsa	17,00	5	80	2018-05-07	C01

Tabla: Boleta

	NUM_BOLETA	FECHA_EMI	ID_CLIENTE	COD_EMPLE	ESTADO_BOLETA
1	10000001	2016-05-11	CLI002	E0002	AC
2	10000002	2016-05-03	CLI002	E0005	AC
3	10000003	2016-05-03	CLI001	E0002	AC
4	10000004	2016-05-02	CLI003	E0005	AC
5	10000005	2016-04-03	CLI001	E0005	AC
6	10000006	2014-04-30	CLI010	E0007	AC
7	10000007	2015-03-04	CLI009	E0007	AC

Tabla: Detalleboleta

	NUM_BOLETA	ID_PRODUCTO	CANTIDAD	IMPORTE
1	10000001	PRO002	3	300,00
2	10000001	PRO010	2	290,00
3	10000002	PRO002	2	200,00
4	10000002	PRO006	3	294,00
5	10000002	PRO008	1	230,00
6	10000003	PRO008	2	460,00
7	10000003	PRO010	3	435,00
8	10000003	PRO012	6	120,00
9	10000003	PRO014	10	100,00
10	10000004	PRO004	6	210,00
11	10000004	PRO011	11	792,00
12	10000004	PRO013	3	45,00
13	10000005	PRO011	7	504,00
14	10000005	PRO012	4	96,00
15	10000005	PRO013	5	75,00
16	10000005	PRO014	8	80,00
17	10000006	PRO007	2	150,00
18	10000006	PRO009	1	290,00
19	10000007	PRO001	10	1600,00
20	10000007	PRO003	5	150,00
21	10000007	PRO011	2	144,00
22	10000007	PRO013	6	90,00

Crear la tabla PRODUCTO_BK de la tabla producto.

```
--Validar la existencia de la tabla
IF OBJECT_ID('PRODUCTO_BK') IS NOT NULL
BEGIN
    DROP TABLE PRODUCTO_BK
END
GO

--Creando la copia de la tabla
SELECT * INTO PRODUCTO_BK      FROM PRODUCTO
GO

--Listando los registros de la tabla Backup
SELECT * FROM
PRODUCTO_BK GO
```

Crear la tabla **CLIENTE_BK** de la tabla Cliente con las siguientes columnas Id_cliente, nombres+apellidos, dirección, fono, correo electrónico (Email).

Actualización de datos UPDATE:

- Actualizar el precio de venta del producto en un 50% más sobre el precio de venta actual de todos los productos.

```
UPDATE PRODUCTO SET PRECIO_VENTA=PRECIO_VENTA*1.5
```

```
GO
```

Actualizar su dirección de la cliente ANETH LUANA con “AV. TUPAC INCA YUPANQUI # 1250”.

Actualizar su teléfono y email del cliente JOSE LUIS por 990990388, josesito555@gmail.com

Actualizar el teléfono de todos los clientes a “000000000”

Actualizar el sueldo básico de todos los empleados en un 20% más sobre el sueldo básico actual.

Actualizar con la fecha actual todas las boletas registradas en el año 2014.

Actualizar el stock actual de la tabla producto con 150 solo de la categoría abarrotes (utilice su código)

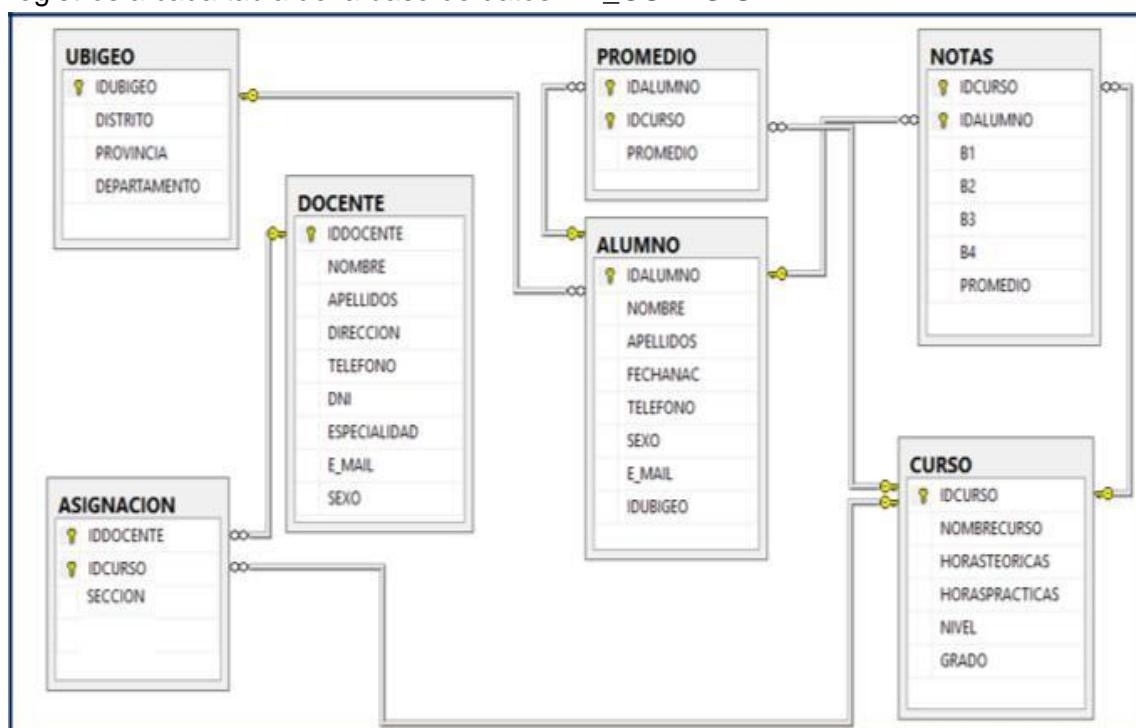
Eliminación de registros DELETE:

Eliminar el distrito “Ancón”.

Eliminar al cliente “Mario Cerpa Carrasco”

ACTIVIDAD 02: COLEGIO

Para comprobar las restricciones aplicadas en la sesión anterior, debe insertar registros a cada tabla de la base de datos **BD_COLEGIO**



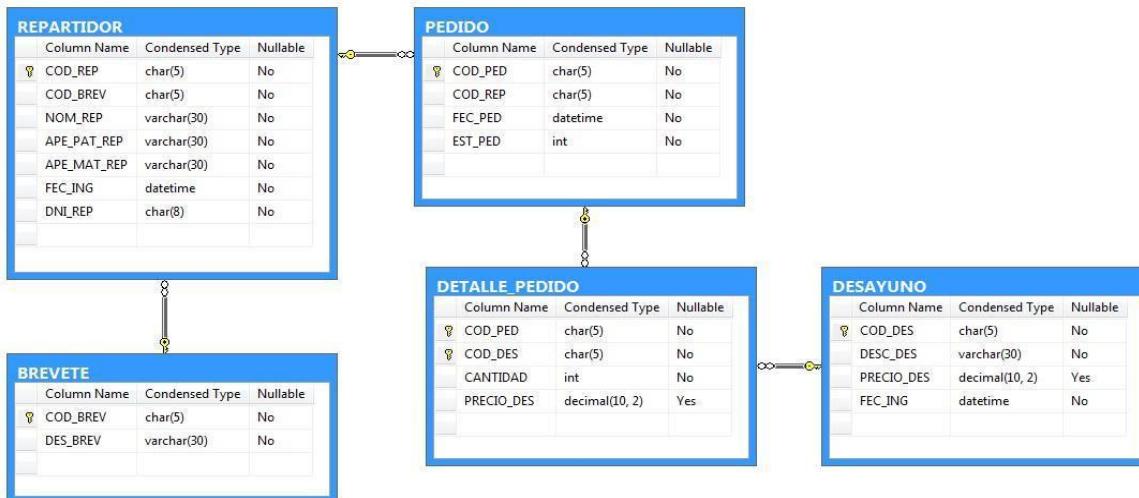
Inserción de registros INSERT:

Insertar 3 registros a cada tabla de la base de datos **BD_COLEGIO** usando registros únicos.

Insertar 3 registros a cada tabla de la base de datos **BD_COLEGIO** usando múltiples registros.

ACTIVIDAD 03: DELIVERI

Crear la base de datos **DELIVERI** a partir del siguiente diagrama de base de datos:



1. Inserción de registros INSERT:

Insertar 2 registros a todas las tablas de la base de datos **DELIVERI** usando registros únicos.

Insertar 2 registros a todas las tablas usando múltiples registros.

Crear la tabla **DESAYUNO_BK** de la tabla **DESAYUNO**.

Crear la tabla **REPARTIDOR_BK** de la tabla **REPARTIDOR** con las siguientes columnas cod_rep, cod_brev, nom_rep+ape_pat_rep+ape_mat_rep.

2. Actualizacion de datos UPDATE:

Actualizar el precio del desayuno aumentando S/. 5.00 al precio actual de todos los productos.

Actualizar la fecha del pedido con la fecha actual a todos los pedidos.

3. Eliminacion de registros DELETE:

Eliminar un registro de la tabla Desayuno.

Eliminar un registro de la tabla Repartidor.

Resumen

El Lenguaje de Manipulación de Datos o DML es un lenguaje que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos se clasifica en:

INSERT INTO: consiste en añadir a una tabla una o más filas y en cada fila todos o parte de sus campos.

UPDATE: Permite la actualización de uno o varios registros de una única tabla.

DELETE: consiste en borrar datos de una tabla, para borrar todos los datos de una tabla ejecutamos TRUNCATE.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

<https://msdn.microsoft.com/es-es/library/ff848766.aspx>

página web referente al tema de manipulación de datos en SQL Server

[https://technet.microsoft.com/es-es/library/ms177591\(v=sql.90\).aspx](https://technet.microsoft.com/es-es/library/ms177591(v=sql.90).aspx)

Instrucciones de lenguaje de manipulación de datos (DML) (Transact-SQL)

<http://www.incanatoit.com/2014/12/lenguaje-de-manipulacion-datos-dml-sql-server-2014.html>

Lenguaje de Manipulación de datos SQL Server - Bases de Datos en Microsoft SQL Server 2014

https://es.wikipedia.org/wiki/Lenguaje_de_manipulaci%C3%B3n_de_datos

Lenguaje de manipulación de datos (DML)



IMPLEMENTACIÓN DE CONSULTAS

LOGROS DE LA UNIDAD DE APRENDIZAJE

Al término de la tercera unidad, el estudiante recupera información de una base de datos utilizando SQL Server 2014 y aplicando múltiples condiciones de comparación o funciones para el manejo de campos tipo fecha.

TEMARIO

3.1 Tema 8 : Recuperación de datos

- 3.1.1 : Introducción al lenguaje de consultas SQL
- 3.1.2 : Uso de la sentencia: SELECT, FROM, WHERE, ORDER BY
- 3.1.3 : Manipulación de consultas condicionales: Operadores lógicos (AND, OR y NOT). Operadores de comparación <, <, =, >, >=, <=. Operador para el manejo de cadenas LIKE. Otros operadores IN, BETWEEN.
- 3.1.4 : Funciones para el manejo de fechas (DAY (), MONTH (), YEAR (), DATEPART (), DATEDIFF())
- 3.1.5 : Ejercicios de aplicación
- 3.1.6 : Ejercicios de reforzamiento

ACTIVIDADES PROPUESTAS

Generan consultas sencillas a la base de datos con el comando SELECT, INNER JOIN y el uso de funciones para el manejo de fechas.

Emplean operadores lógicos y condicionales en las consultas SQL a la base de datos.

Determinan la necesidad de agrupar la salida de datos de una selección.

Emplean funciones para calcular la suma, promedio, mínimo, máximo y cantidad de un rango de datos.

Emplean combinaciones externas y cruzadas entre tablas para determinar ocurrencias o no.

3.1. Recuperación de datos

3.1.1. Introducción al lenguaje de consultas SQL.

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft. SQL se utiliza para crear objetos QueryDef, como el argumento de origen del método OpenRecordSet y como la propiedad RecordSource del control de datos. También se puede utilizar con el método Execute para crear y manipular directamente las bases de datos DEPARTAMENTOS y crear consultas SQL de paso, para manipular bases de datos remotas cliente - servidor.

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos. Esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

3.1.2. Uso de la sentencia SELECT, FROM, WHERE, ORDER BY

3.1.2.1 Sentencia SELECT

Es un comando que permite componer una consulta; este es interpretado por el servidor, el cual recupera los datos especificados de las tablas. Su formato es:

```
SELECT  
    [ALL | * | DISTINCT]  
    [TOP VALOR [PERCENT]]  
    [ALIAS.] COLUMN [AS] CABECERA  
  
FROM NOMBRE_TABLA [ALIAS]  
  
[WHERE CONDICION]  
[ORDER BY COLUMN [ASC | DESC]]
```

3.1.2.2 Consultas usando el operador *

El operador asterisco cumple dos funciones dentro de la implementación de una consulta, la primera es mostrar todos los registros de la tabla y la segunda es mostrar todas las columnas que la componen con el orden especificado en su creación. Veamos algunas consultas:

Script que permite mostrar los todos los registros de la tabla Cliente.

```
SELECT *  
FROM TB_CLIENTE
```

Script que permite mostrar los todos los registros de la tabla Producto usando alias en la tabla.

```
SELECT P.*  
FROM TB_PRODUCTO P
```

En caso, no requiera usar alias podríamos optar por la siguiente sentencia:

```
SELECT TB_PRODUCTO.*
FROM TB_PRODUCTO
```

3.1.2.3 Consultas especificando las columnas

La especificación de las columnas permite determinar el orden de los campos en el resultado de la consulta, su objetivo es meramente visual; pues al final el resultado es el mismo, es decir mostrar todos los registros administrando las columnas de la tabla. Veamos algunas consultas:

Script que permita mostrar los registros de la tabla Cliente, tal como se muestra en el siguiente formato:

COD_CLI	RAZ_SOC_CLI	DIR_CLI	FEC_REG	TIP_CLI
XXXX	XXXXXXXXXX	XXXXXX	99/99/9999	9
XXXX	XXXXXXXXXX	XXXXXX	99/99/9999	9

```
SELECT COD_CLI, RAZ_SOC_CLI, DIR_CLI, FEC_REG,
TIP_CLI FROM TB_CLIENTE
```

O también podría definirse usando alias:

```
SELECT C.COD_CLI, C.RAZ_SOC_CLI, C.DIR_CLI, C.FEC_REG,
C.TIP_CLI FROM TB_CLIENTE C
```

Script que permita mostrar los registros de la tabla Factura, tal como se muestra en el siguiente formato:

NUM_FAC	FEC_FAC	POR_IGV	EST_FAC	COD_CLI
XXXX	99/99/9999	9.99	9	XXXX
XXXX	99/99/9999	9.99	9	XXXX

```
SELECT F.NUM_FAC, F.FEC_FAC, F.POR_IGV, F.EST_FAC,
F.COD_CLI FROM TB_FACTURA F
```

3.1.2.4 Consultas especificando cabeceras de columnas

La especificación de cabeceras permite asignar títulos a las columnas mostradas en el resultado de una consulta SELECT. Veamos algunas consultas:

Script que permita mostrar los registros de la tabla Cliente, tal como se muestra en el siguiente formato:

CODIGO	RAZON SOCIAL	DIRECCIÓN	FECHA DE REGISTRO	TIPO
XXXX	XXXXXXXXXX	XXXXXX	99/99/9999	9
XXXX	XXXXXXXXXX	XXXXXX	99/99/9999	9

```
SELECT C.COD_CLI AS CODIGO, C.RAZ_SOC_CLI AS [RAZON SOCIAL],
C.DIR_CLI AS DIRECCION, C.FEC_REG AS [FECHA DE REGISTRO],
C.TIP_CLI AS TIPO
FROM TB_CLIENTE C
```

Una segunda versión podría ser de la siguiente manera:

```
SELECT      C.COD_CLI CODIGO,
              C.RAZ_SOC_CLI [RAZON SOCIAL],
              C.DIR_CLI DIRECCION,
              C.FEC_REG [FECHA DE REGISTRO],
              C.TIP_CLI TIPO
FROM TB_CLIENTE C
```

Y finalmente, podríamos optar por la siguiente sentencia:

```
SELECT      CODIGO=C.COD_CLI,
              [RAZON SOCIAL]=C.RAZ_SOC_CLI,
              DIRECCION=C.DIR_CLI,
              [FECHA DE REGISTRO]=C.FEC_REG,
              TIPO=C.TIP_CLI
FROM TB_CLIENTE C
```

Script que permita mostrar los registros de la tabla Vendedor, tal como se muestra en el siguiente formato:

CODIGO	VENDEDOR	FECHA DE INGRESO
XXXX	XXXXXXX XXXXXXXXXXXX	99/99/9999
XXXX	XXXXXXX XXXXXXXXXXXX	99/99/9999

```
SELECT V.COD_VEN AS CODIGO,
        V.NOM_VEN+SPACE (1)+V.APE_VEN AS
        VENDEDOR, V.FEC_ING AS [FECHA DE INGRESO]
FROM TB_VENDEDOR V
```

	CODIGO	VENDEDOR	FECHA DE INGRESO
1	V01	JUANA MESES	2015-01-15
2	V02	JUAN SOTO	2014-02-05
3	V03	CARLOS AREVALO	2013-03-25
4	V04	CESAR OJEDA	2014-05-05
5	V05	JULIO VEGA	2014-01-10
6	V06	ANA ORTEGA	2015-02-20
7	V07	JOSE PALACIOS	2013-03-02
8	V08	RUBEN SALAS	2014-05-07
9	V09	PATRICIA ARCE	2013-06-28
10	V10	RENATO IRIARTE	2013-04-16

Figura 54: Listado de vendedores
Fuente. - Tomado desde SQL Server 2014

3.1.2.5 Consultas distinguida

Este tipo de consulta permita mostrar solo una ocurrencia de un conjunto de valores repetidos a partir de una columna de la tabla. Veamos algunas consultas:

Script que permita mostrar las fechas de registro de factura sin repetirse; obtenidos desde la tabla Factura:

```
SELECT DISTINCT FEC_FAC
FROM TB_FACTURA
```

Script que permita mostrar los códigos de los distritos en la cual se encuentran registrados los vendedores, debe tener en cuenta que dichos códigos no deben mostrarse de manera repetida:

```
SELECT DISTINCT COD_DIS
FROM TB_VENDEDOR
```

3.1.2.6 Consultas ordenadas

Este tipo de consulta permita mostrar la información de los registros ordenados de tal forma que podemos especificar que columnas deben ordenarse tanto de forma ascendente como descendente. Veamos algunas consultas ordenadas:

Script que permite mostrar los registros de la tabla Cliente ordenadas por la razón social de forma ascendente:

```
SELECT C. *
FROM TB_CLIENTE C
ORDER BY C.RAZ_SOC_CLI
```

Recordemos que el operador asterisco hace referencia a todas las columnas de la tabla específica, es así, que en la cláusula Order By se puede especificar a cualquier columna de la tabla Cliente; así mismo especificar que cuando no se especifica el orden el servidor aplica ascendente de forma predeterminada. Con la misma finalidad también podríamos usar la siguiente sentencia:

```
SELECT C. *
FROM TB_CLIENTE C
ORDER BY 2 ASC
```

El número dos hace referencia a la segunda columna de la lista y es, justamente la columna que queremos ordenar, esto nos podría ayudar a ordenar columnas que fueron compuestas por una concatenación, función o campo calculado.

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI
1	C006	Berker	Av. Los Proceres 521	3810322
2	C016	Cardeli	Jr. Bartolome Herrera 451	2658853
3	C019	Corefo	Av. Canada 3894 - 3898	4377499
4	C020	Cramer	Jr. Mariscal Miller 1131	4719061
5	C018	Dasin	Av. Saenz Peña 338 - B	4657574
6	C007	Fidenza	Jr. El Niquel 282	5289034
7	C011	Filasur	Av. El Santuario 1189	4598175
8	C001	Finseth	Av. Los Viñedos 150	4342318
9	C013	Hayashi	Jr. Ayacucho 359	42847990

Figura 55: Listado de clientes
Fuente. - Tomado desde SQL Server 2014

Script que permite mostrar el código, tipo, nombre completo (nombres y apellidos) y el sueldo de los vendedores ordenador por el tipo de vendedor de forma ascendente y ante la coincidencia por el nombre completo del vendedor de forma descendente.

```
SELECT V.COD_VEN AS CODIGO,
       V.TIP_VEN AS TIPO,
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       V.SUE_VEN AS SUELDO
  FROM TB_VENDEDOR V
 ORDER BY 2 ASC, 3 DESC
```

Los números dos y tres hacen referencia a las columnas tipo y vendedor especificada en la consulta, así mismo, podríamos usar la siguiente sentencia:

```
SELECT V.COD_VEN AS CODIGO,
       V.TIP_VEN AS TIPO,
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR,
       V.SUE_VEN AS SUELDO
  FROM TB_VENDEDOR V
 ORDER BY TIPO ASC, VENDEDOR DESC
```

En esta última sentencia, estamos haciendo referencia a cabeceras que fueron especificadas en la consulta, y el resultado es el mismo como se observa en la siguiente imagen:

	CODIGO	TIPO	VENDEDOR	SUELDO
1	V05	1	JULIO VEGA	1600.00
2	V01	1	JUANA MESES	1100.00
3	V07	1	JOSE PALACIOS	1600.00
4	V04	1	CESAR OJEDA	1550.00
5	V06	1	ANA ORTEGA	1300.00
6	V08	2	RUBEN SALAS	1550.00
7	V10	2	RENATO IRIARTE	1650.00
8	V09	2	PATRICIA ARCE	1900.00
9	V02	2	JUAN SOTO	1300.00
10	V03	2	CARLOS AREVALO	1600.00

Figura 56: Listado de vendedores
Fuente. - Tomado desde SQL Server 2014

3.1.3. Manipulación de consultas condicionales. Uso de condicionales: operadores lógicos AND, OR. Operadores de comparación >, <, =, <>, <=, >=. Operador para manejo de cadenas LIKE. Otros operadores: IN, BETWEEN.

3.1.3.1 Cláusula Where de la sentencia SELECT

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones que deben cumplir dichos resultados para su muestra. Así mismo debemos tener en cuenta que si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla.

WHERE es opcional, pero cuando aparece debe ir a continuación de FROM o INNER JOIN. Veamos algunas consultas de condición simple:

Script que permite mostrar los detalles registrados para la factura numero 100 obtenidos desde la tabla Detalle de factura.

```
SELECT DT.*
FROM TB_DETALLE_FACTURA DT
WHERE DT.NUM_FAC=100
```

	NUM_FAC	COD_PRO	CAN_VEN	PRE_VEN
1	100	P007	6	5.00
2	100	P011	25	25.00
3	100	P013	11	20.00

Figura 57: Listado de detalle de la factura 100
Fuente. - Tomado desde SQL Server 2014

Script que permite mostrar solo los clientes asignados como tipo de cliente 2.

```
SELECT C.*
FROM TB_CLIENTE C
WHERE C.TIP_CLI=2
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI
1	C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	D04	1990-02-01	2
2	C003	Serviemsa	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2
3	C007	Fidenza	Jr. El Niquel 282	5289034	16204790	D20	1991-10-02	2
4	C008	Intech	Av. San Luis 2619 5to P	2249493	34021824	D09	1997-01-07	2
5	C010	Landu	Av. Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2
6	C013	Hayashi	Jr. Ayacucho 359	42847990	NULL	D22	1993-06-11	2

Figura 58: Listado de clientes con tipo 2
Fuente. - Tomado desde SQL Server 2014

Script que permite toda la información registrada del producto “Papel Periódico” obtenido desde la tabla Producto.

```
SELECT P.*
FROM TB_PRODUCTO P
WHERE P.DES_PRO='PAPEL PERIÓDICO'
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P004	Papel Periódico	7.20	4285	1000	MII	2	FALSO

Figura 59: Listado de productos con descripción Papel periódico
Fuente. - Tomado desde SQL Server 2014

3.1.3.2 Operadores de comparación

Operador	Descripción
=	Determina la igualdad entre dos valores.
>	Determina si el primer valor es mayor que el segundo.
>=	Determina si el primer valor es mayor o igual que el segundo.
<	Determina si el primer valor es menor que el segundo.
<=	Determina si el primer valor es menor o igual que el segundo.

SQL usa los operadores de comparación de la misma manera que los lenguajes de programación; es decir se comparan dos expresión u valores y se emiten un Verdadero y Falso resultando de la comparación, es así mismo, como se aplica en SQL. Veamos algunas consultas:

Script que permita listar los productos cuyo precio sea menor a S/. 10.00.

```
SELECT P.*
  FROM TB_PRODUCTO P
 WHERE PRE_PRO < 10
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P004	Papel Periódico	7.20	4285	1000	Mil	2	FALSO
2	P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO
3	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
4	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
5	P012	Tajador Plástico	9.60	608	300	Doc	3	FALSO
6	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
7	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO
8	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO

Figura 60: Listado de productos con precio inferior a S/. 10

Fuente. - Tomado desde SQL Server 2014

Script que permita listar las facturas cuyo registro en el número de factura sea mayor a 116.

```
SELECT F.*
  FROM TB_FACTURA F
 WHERE F.NUM_FAC > 116
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	117	1999-05-02	C016	1999-04-02	3	V10	0.18
2	118	1999-07-02	C008	1999-01-03	3	V03	0.18
3	119	1999-06-02	C013	1999-10-03	2	V02	0.18
4	120	1999-02-07	C011	1999-02-23	1	V01	0.18

Figura 61: Listado de facturas con numero de factura superior a 116

Fuente. - Tomado desde SQL Server 2014

3.1.3.3 Operadores lógicos:

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not. A excepción de los dos últimos todos poseen la siguiente sintaxis:

```
<expresión1> OperadorLogico <expresión2>
```

En donde **expresión1** y **expresión2** son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

Tabla AND

Expresión1	Expresión2	Resultado
V	V	V
V	F	F
F	V	F
F	F	F

Tabla OR

Expresión1	Expresión2	Resultado
V	V	V
V	F	V
F	V	V
F	F	F

Tabla XOR

Expresión1	Expresión2	Resultado
V	V	F
V	F	V
F	V	V
F	F	F

Tabla Eqv

Expresión1	Expresión2	Resultado
V	V	V
V	F	F
F	V	F
F	F	V

Tabla Imp

Expresión1	Expresión2	Resultado
V	V	V
V	F	F
F	V	V
F	F	V
V	NULL	NULL
F	NULL	V
NULL	NULL	NULL
NULL	V	V
NULL	F	NULL

Veamos algunas consultas que usen operadores lógicos:

Script que permita listar los **vendedores de tipo 2** y además que sean del **distríto cuyo código es D04**.

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1100.00	2015-01-15	1	D08
2	V02	JUAN	SOTO	1300.00	2014-02-05	2	D03
3	V03	CARLOS	AREVALO	1600.00	2013-03-25	2	D09
4	V04	CESAR	OJEDA	1550.00	2014-05-05	1	D01
5	V05	JULIO	VEGA	1600.00	2014-01-10	1	D01
6	V06	ANA	ORTEGA	1300.00	2015-02-20	1	D05
7	V07	JOSE	PALACIOS	1600.00	2013-03-02	1	D02
8	V08	RUBEN	SALAS	1550.00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1900.00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1650.00	2013-04-16	2	D01

```
SELECT V.*  
FROM TB_VENDEDOR V  
WHERE TIP_VEN=2 AND COD_DIS='D04'
```

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V08	RUBEN	SALAS	1550.00	2014-05-07	2	D04
2	V09	PATRICIA	ARCE	1900.00	2013-06-28	2	D04

Figura 62: Listado de vendedores de tipo 2 y código de distrito D04
Fuente. - Tomado desde SQL Server 2014

Script que permita listar los **productos de tipo nacional** y además tenga como **unidad de medida la docena (doc)**.

```
SELECT P.*  
FROM TB_PRODUCTO P  
WHERE IMPORTADO='FALSO' AND UNI_MED='DOC'
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
2	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
3	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
4	P012	Tajador Plastico	9.60	608	300	Doc	3	FALSO
5	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
6	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO

Figura 63: Listado de productos de importación falso y unidad de medida “Doc”
Fuente. - Tomado desde SQL Server 2014

Script que permita listar las facturas registradas (FEC_FAC) desde el primero de enero hasta 31 de diciembre del año 1998.

```
SELECT F.*  
FROM TB_FACTURA F  
WHERE F.FEC_FAC>='1998/01/01' AND  
F.FEC_FAC<='1998/12/31'
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	100	1998-06-07	C001	1998-05-08	2	V01	0.18
2	101	1998-06-09	C019	1998-05-08	3	V02	0.18
3	102	1998-01-09	C003	1998-03-11	2	V04	0.18
4	103	1998-06-09	C016	1998-05-11	2	V07	0.18
5	104	1998-01-10	C015	1998-12-10	2	V08	0.18
6	105	1998-10-10	C009	1998-05-08	3	V05	0.18
7	106	1998-05-10	C019	1998-05-08	1	V09	0.18
8	107	1998-09-10	C012	1998-06-11	2	V10	0.18
9	108	1998-03-10	C008	1998-11-11	2	V09	0.18
10	109	1998-10-01	C017	1998-06-11	2	V02	0.18
11	110	1998-10-11	C019	1998-01-12	2	V05	0.18
12	111	1998-01-12	C014	1998-01-12	1	V04	0.18
13	112	1998-01-12	C011	1998-01-12	3	V08	0.18
14	113	1998-03-12	C020	1998-01-12	2	V09	0.18
15	114	1998-08-12	C015	1999-06-01	2	V07	0.18

Figura 64: Listado de facturas del año 1998
 Fuente. - Tomado desde SQL Server 2014

Script que permita listar las **facturas registradas desde el número 100 hasta el 104.**

```
SELECT D.*  

    FROM TB_DETALLE_FACTURA D  

    WHERE NUM_FAC>=100 AND NUM_FAC<=104
```

	NUM_FAC	COD_PRO	CAN_VEN	PRE_VEN
1	100	P007	6	5.00
2	100	P011	25	25.00
3	100	P013	11	20.00
4	102	P004	8	10.00
5	103	P002	10	40.00
6	103	P011	6	20.00
7	103	P017	21	12.00
8	103	P019	12	10.00
9	104	P004	3	10.00
10	104	P009	50	5.00

Figura 65: Listado de facturas con numero entre 100 y 104
 Fuente..- Tomado desde SQL Server 2014

Script que permita **listar los clientes** cuyo **código de distrito** se encuentre registrado como **D01**; así mismo como los de código **D22**.

```
SELECT C.*  

    FROM TB_CLIENTE C  

    WHERE COD_DIS='D01' OR COD_DIS='D22'
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C004	Issa	Calle Los Aviadores 263	3725910	46720159	D01	1992-09-12	1	Luis Apumaya
2	C013	Hayashi	Jr. Ayacucho 359	42847990	NULL	D22	1993-06-11	2	Ernesto Uehara

Figura 66: Listado de clientes de los distritos D01 y D22
 Fuente.- Tomado desde SQL Server 2014

Script que permita **listar las facturas** registradas desde el número 100 hasta el 102 y además del 117 al 120.

```
SELECT D.*  
      FROM TB_DETALLE_FACTURA D  
     WHERE (NUM_FAC>=100 AND NUM_FAC<=102) OR  
           (NUM_FAC>=117 AND NUM_FAC<=120)
```

	NUM_FAC	COD_PRO	CAN_VEN	PRE_VEN
1	100	P007	6	5.00
2	100	P011	25	25.00
3	100	P013	11	20.00
4	102	P004	8	10.00
5	117	P002	120	40.00
6	117	P005	120	40.00
7	118	P003	4	12.00
8	118	P005	6	40.00
9	119	P002	150	40.00
10	119	P003	6	10.00
11	119	P006	2	45.00
12	119	P008	10	30.00
13	120	P009	120	10.00
14	120	P015	5	15.00

Figura 67: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

Script que permite **listar solo los clientes** que aun **no registran un número de RUC (RUC_CLI)**.

```
SELECT C.*  
      FROM TB_CLIENTE C  
     WHERE RUC_CLI IS NULL
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C003	Serviems	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
2	C009	Prominent	Jr. Iquique 132	43233519	NULL	D11	1993-06-11	1	Jorge Valdivia
3	C013	Hayashi	Jr. Ayacucho 359	42847990	NULL	D22	1993-06-11	2	Ernesto Uehara

Figura 68: Listado de clientes con RUC nulos
Fuente.- Tomado desde SQL Server 2014

Script que permite **listar todos los productos** registrados a **excepción de la unidad de medida UNI**.

```
SELECT P.*  
      FROM TB_PRODUCTO P  
     WHERE NOT P.UNI_MED='UNI'
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P001	Papel Bond A-4	35.00	200	1500	MII	2	VERDADERO
2	P002	Papel Bond Oficio	28.00	50	1500	MII	2	FALSO
3	P003	Papel Bulky	10.00	498	1000	MII	2	VERDADERO
4	P004	Papel Periódico	7.20	4285	1000	MII	2	FALSO
5	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
6	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
7	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
8	P012	Tajador Plástico	9.60	608	300	Doc	3	FALSO
9	P013	Folder Manila Oficio	16.00	200	150	Cie	3	FALSO
10	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
11	P015	Sobre Manila Oficio	12.00	300	130	Cie	3	FALSO
12	P016	Sobre Manila A-4	14.40	200	100	Cie	3	FALSO
13	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
14	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO
15	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO
16	P020	Folder Plástico A-4	40.00	3080	1100	Cie	3	FALSO

Figura 69: Listado de productos

Fuente.- Tomado desde SQL Server 2014

Script que permite **listar todas las órdenes de compra registradas en el año 1999 con asignación de estado 1.**

```
SELECT O.*  
      FROM TB_ORDEN_COMPRA O  
     WHERE NOT(FEC_OCO>='1998/01/01' AND FEC_OCO<='1998/12/31'  
              AND EST_OCO=1)
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC003	1998-02-08	PR10	1998-02-08	3
2	OC004	1998-05-04	PR01	1998-05-04	2
3	OC007	1998-06-02	PR20	1998-05-04	3
4	OC010	1998-05-09	PR01	1998-05-09	2
5	OC012	1998-04-10	PR14	1998-05-10	3
6	OC016	1998-06-12	PR06	1998-06-12	3
7	OC017	1999-08-01	PR09	1999-08-01	1
8	OC018	1999-01-02	PR20	1999-08-02	1
9	OC019	1999-03-03	PR11	1999-06-03	1
10	OC020	1999-07-10	PR12	1999-08-13	1
11	OC021	1999-08-30	PR14	1999-09-13	1
12	OC022	1999-06-14	PR05	1999-08-14	2

Figura 70: Listado de órdenes de compra del año 1999 y estado 1

Fuente.- Tomado desde SQL Server 2014

3.1.3.4 Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es de la siguiente manera:

EXPRESIÓN LIKE MODELO

En donde, expresión es el campo de donde se comparan sus valores textuales. Se puede utilizar el operador LIKE para encontrar valores en los campos que coincidan con el modelo especificado. De acuerdo con el modelo empleado, se puede especificar un

valor completo (Ana María) o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (LIKE 'AN%'). El operador LIKE se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like 'C%' en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

Veamos la siguiente consulta el cual devuelve los registros de los clientes cuya razón social comienzan con la letra M seguido de cualquier letra entre A y I y de solo tres dígitos:

```
SELECT *
  FROM TB_CLIENTE
 WHERE RAZ_SOC_CLI Like 'M[A-I]__'
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C005	Mass	Av. Tomas Mansano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo
2	C015	Meba	Av. Elmer Faucett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez

Figura 71: Listado de clientes
Fuente..- Tomado desde SQL Server 2014

Si, por el contrario, necesitamos una consulta que muestre los registros de los clientes cuya razón social comienzan con la letra M seguido de cualquier letra entre A y I y de cualquier cantidad de dígitos, el script sería como sigue:

```
SELECT *
  FROM TB_CLIENTE
 WHERE RAZ_SOC_CLI Like 'M[A-I]%'
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C005	Mass	Av. Tomas Mansano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo
2	C015	Meba	Av. Elmer Faucett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez

Figura 72: Listado de clientes
Fuente..- Tomado desde SQL Server 2014

3.1.3.5 Operador de intervalo de valor BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo, emplearemos el operador Between cuya sintaxis es de la siguiente manera:

```
CAMPO BETWEEN VALOR1 AND VALOR2
```

En este caso la consulta devolvería los registros que contengan un "campo" con el valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo. Veamos algunas consultas que involucren el uso de la cláusula BETWEEN.

Script que permita listar todas las órdenes de compra registradas en el primer semestre del año 1999.

```
SELECT O.*  
      FROM TB_ORDEN_COMPRA O  
     WHERE FEC_OCO BETWEEN '1999/01/01' AND '1999/06/30'
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC018	1999-01-02	PR20	1999-08-02	1
2	OC019	1999-03-03	PR11	1999-06-03	1
3	OC022	1999-06-14	PR05	1999-08-14	2

Figura 73: Listado de órdenes de compra del primer semestre del año 1999

Fuente.- Tomado desde SQL Server 2014

Script que permita listar todas las facturas cuyo número de registro se encuentre entre los números 111 y 117.

```
SELECT F.*  
      FROM TB_FACTURA F  
     WHERE F.NUM_FAC BETWEEN 111 AND 117
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	111	1998-01-12	C014	1998-01-12	1	V04	0.18
2	112	1998-01-12	C011	1998-01-12	3	V08	0.18
3	113	1998-03-12	C020	1998-01-12	2	V09	0.18
4	114	1998-08-12	C015	1999-06-01	2	V07	0.18
5	115	1999-06-01	C016	1999-09-01	2	V05	0.18
6	116	1999-06-01	C015	1999-09-01	1	V06	0.18

Figura 74: Listado de facturas con numero de factura entre 111 y 117

Fuente.- Tomado desde SQL Server 2014

Script que permita listar todos los productos cuya letra inicial de su descripción se encuentre entre las letras A y F.

```
SELECT P.*  
      FROM TB_PRODUCTO P  
     WHERE DES_PRO BETWEEN 'A%' AND 'F%'
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P005	Cartucho Tinta Negra	32.00	50	30	Uni	1	FALSO
2	P006	Cartucho Tinta Color	36.00	58	35	Uni	1	FALSO
3	P008	Caja de Diskettes * 10	24.00	125	180	Uni	1	FALSO
4	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
5	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO

Figura 75: Listado de productos con inicial A y F en la descripción

Fuente.- Tomado desde SQL Server 2014

3.1.3.6 Operador IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los valores en una lista. Su sintaxis es la siguiente:

```
EXPRESIÓN IN(VALOR1, VALOR2,...)
```

Script que permita listar todas las facturas registradas con los números del 111 al 117.

```
SELECT F.*
FROM TB_FACTURA F
WHERE F.NUM_FAC IN (111, 112, 113, 114, 115, 117)
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	111	1998-01-12	C014	1998-01-12	1	V04	0.18
2	112	1998-01-12	C011	1998-01-12	3	V08	0.18
3	113	1998-03-12	C020	1998-01-12	2	V09	0.18
4	114	1998-08-12	C015	1999-06-01	2	V07	0.18
5	115	1999-06-01	C016	1999-09-01	2	V05	0.18
6	117	1999-05-02	C016	1999-04-02	3	V10	0.18

Figura 76: Listado de facturas con número entre 111 y 117

Fuente.- Tomado desde SQL Server 2014

Script que permita listar todos los registros de los vendedores cuyo código de distrito sea D08 o D04.

```
SELECT V.*
FROM TB_VENDEDOR V
WHERE COD_DIS IN ('D08', 'D04')
```

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1100.00	2015-01-15	1	D08
2	V08	RUBEN	SALAS	1550.00	2014-05-07	2	D04
3	V09	PATRICIA	ARCE	1900.00	2013-06-28	2	D04

Figura 77: Listado de vendedores del distrito D08 y D04

Fuente.- Tomado desde SQL Server 2014

3.1.4. Funciones para el manejo de fechas: DAY (), MONTH (), YEAR () , DATEPART (), DATEDIFF()

3.1.4.1 Función Day

Devuelve un valor numérico entero que representa el día según el mes de una determinada fecha. Veamos algunas consultas:

Script que permite mostrar el día correspondiente a la fecha actual.

```
SELECT DAY(GETDATE()) AS DIA
```

DIA	
1	13

Figura 78: Mostrando el día desde la fecha actual

Fuente.- Tomado desde SQL Server 2014

Debemos considerar que para obtener la fecha actual se usa la función GETDATE().

Script que permita mostrar el día correspondiente a una determinada fecha obtenida a partir de una variable local.

```
DECLARE @FECHA DATE = '2017/02/07'
SELECT DAY(@FECHA) AS DIA
```

DIA	
1	7

Figura 79: Mostrando el día desde una fecha
Fuente.- Tomado desde SQL Server 2014

Debemos recordar que para declarar una variable local se debe usar la instrucción **DECLARE** y que cada nombre de variable se inicia anteponiendo el símbolo arroba (@).

Script que permite mostrar los datos de la factura con el siguiente formato:

NUMERO	DIA DE REGISTRO	FECHA DE REG.	CODIGO DE CLIENTE
99	99	99/99/9999	XXXX
99	99	99/99/9999	XXXX

```
SELECT F.NUM_FAC AS NUMERO,
       DAY(F.FEC_FAC) AS [DIA DE REGISTRO],
       F.FEC_FAC AS FECHA,
       F.COD_CLI AS [CODIGO DE CLIENTE]
  FROM TB_FACTURA F
     GO
```

NUMERO	DIA DE REGISTRO	FECHA	CODIGO DE CLIENTE
1	100	7	1998-06-07 C001
2	101	9	1998-06-09 C019
3	102	9	1998-01-09 C003
4	103	9	1998-06-09 C016
5	104	10	1998-01-10 C015
6	105	10	1998-10-10 C009
7	106	10	1998-05-10 C019
8	107	10	1998-09-10 C012
9	108	10	1998-03-10 C008

Figura 80: Listado de facturas
Fuente. - Tomado desde SQL Server 2014

3.1.4.2 Función Month

Devuelve un valor numérico entero que representa al mes de una determinada fecha.

Veamos algunas consultas:

Script que permite mostrar el mes correspondiente a la fecha actual.

```
SELECT MONTH(GETDATE()) AS MES
```

MES	
1	2

Figura 81: Mostrando el mes desde la fecha actual
Fuente.- Tomado desde SQL Server 2014

Script que permita mostrar el mes correspondiente a una determinada fecha obtenida a partir de una variable local.

```
DECLARE @FECHA DATE = '2017/02/07'
SELECT MONTH(@FECHA) AS MES
GO
```

MES	
1	2

Figura 82: Mostrando el mes desde una fecha
Fuente.- Tomado desde SQL Server 2014

Script que permita mostrar los datos de la factura con el siguiente formato:

NUMERO	DIA DE REG.	MES DE REG.	FECHA DE REG.	CLIENTE
99	99	99	99/99/9999	XXXX
99	99	99	99/99/9999	XXXX

```
SELECT F.NUM_FAC AS NUMERO,
       DAY(F.FEC_FAC) AS [DIA DE REG.],
       MONTH(F.FEC_FAC) AS [MES DE REG.],
       F.FEC_FAC AS [FECHA DE REG.],
       F.COD_CLI AS [CODIGO DE CLIENTE]
  FROM TB_FACTURA F
GO
```

NUMERO	DIA DE REG.	MES DE REG.	FECHA DE REG.	CODIGO DE CLIENTE
1 100	7	6	1998-06-07	C001
2 101	9	6	1998-06-09	C019
3 102	9	1	1998-01-09	C003
4 103	9	6	1998-06-09	C016
5 104	10	1	1998-01-10	C015
6 105	10	10	1998-10-10	C009
7 106	10	5	1998-05-10	C019
8 107	10	9	1998-09-10	C012
9 108	10	3	1998-03-10	C008
10 109	1	10	1998-10-01	C017
11 110	11	10	1998-10-11	C019

Figura 83: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

3.1.4.3 Función Year

Devuelve un valor numérico entero que representa al año de una determinada fecha.
Veamos algunas consultas:

Script que permite mostrar el año actual.

```
SELECT YEAR(GETDATE()) AS AÑO
```

AÑO	
1	2017

Figura 84: Mostrando el año actual
Fuente.- Tomado desde SQL Server 2014

Script que permite mostrar el año correspondiente a una determinada fecha obtenida a partir de una variable local.

```
DECLARE @FECHA DATE = '2017/02/07'
SELECT YEAR(@FECHA) AS AÑO
GO
```

AÑO	
1	2017

Figura 85: Mostrando el año desde una fecha
Fuente.- Tomado desde SQL Server 2014

Script que permite mostrar los datos de la factura con el siguiente formato:

NUMERO	DIA DE REG.	MES DE REG.	FECHA DE REG.	CLIENTE
99	99	99	99/99/9999	XXXX
99	99	99	99/99/9999	XXXX

```
SELECT F.NUM_FAC AS NUMERO,
      DAY(F.FEC_FAC) AS [DIA DE REG.],
      MONTH(F.FEC_FAC) AS [MES DE REG.],
      YEAR(F.FEC_FAC) AS [AÑO DE REG.],
      F.FEC_FAC AS [FECHA DE REG.]
   FROM TB_FACTURA F
GO
```

NUMERO	DIA DE REG.	MES DE REG.	AÑO DE REG.	FECHA DE REG.
1	100	7	1998	1998-06-07
2	101	9	1998	1998-06-09
3	102	9	1998	1998-01-09
4	103	9	1998	1998-06-09
5	104	10	1998	1998-01-10
6	105	10	1998	1998-10-10
7	106	10	1998	1998-05-10

Figura 86: Listado de facturas
Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar los datos de la factura con el siguiente formato:

NUMERO	FECHA DE REG.(DMY)	FECHA DE REG.
99	99/99/9999	9999/99/99
99	99/99/9999	9999/99/99

```
SELECT F.NUM_FAC AS NUMERO,
       CAST(DAY(F.FEC_FAC) AS VARCHAR(2)) + '/' +
       CAST(MONTH(F.FEC_FAC) AS VARCHAR(2)) + '/' +
       CAST(YEAR(F.FEC_FAC) AS CHAR(4)) AS [FECHA DE
       REG.], F.FEC_FAC AS [FECHA DE REG.]
FROM TB_FACTURA F
GO
```

	NUMERO	FECHA DE REG.	FECHA DE REG.
1	100	7/6/1998	1998-06-07
2	101	9/6/1998	1998-06-09
3	102	9/1/1998	1998-01-09
4	103	9/6/1998	1998-06-09
5	104	10/1/1998	1998-01-10
6	105	10/10/1998	1998-10-10
7	106	10/5/1998	1998-05-10
8	107	10/9/1998	1998-09-10
9	108	10/3/1998	1998-03-10

Figura 87: Listado de facturas
Fuente. - Tomado desde SQL Server 2014

3.1.4.4 Función DatePart

Devuelve un valor numérico entero que representa una parte específica de una determinada fecha. Su formato es:

```
DATEPART ( ARGUMENTO , FECHA_A_EVALUAR )
```

Si asumimos que hoy es 2015-07-13 17:03:55.300 entonces podemos mencionar los siguientes argumentos de la función:

ARGUMENTO	VALOR DEVUELTO
year, yyyy, yy	2015
month, mm, m	07
dayofyear, dy, y	194
day, dd, d	13
week, wk, ww	29
weekday, dw	1
hour, hh	17
minute, n	3
second, ss, s	55
millisecond, ms	300

Veamos algunas consultas:

Script que permita mostrar todos los valores correspondientes al argumento que presenta la función DATEPART.

```
SELECT 'AÑO ACTUAL' AS FUNCION,DATEPART(YY,GETDATE()) AS VALOR
UNION
SELECT 'MES ACTUAL',DATEPART(MM,GETDATE())
UNION
SELECT 'NUMERO DE DIA DEL AÑO',DATEPART(DY,GETDATE())
UNION
SELECT 'DIA ACTUAL',DATEPART(DD,GETDATE())
UNION
SELECT 'NUMERO DE SEMANA',DATEPART(WK,GETDATE())
UNION
SELECT 'NUMERO DE DIA EN LA SEMANA',DATEPART(DW,GETDATE())
UNION
SELECT 'HORA ACTUAL',DATEPART(HH,GETDATE())
UNION
SELECT 'MINUTO ACTUAL',DATEPART(N,GETDATE())
UNION
SELECT 'SEGUNDO ACTUAL',DATEPART(SS,GETDATE())
UNION
SELECT 'MILISEGUNDOS ACTUAL',DATEPART(MS,GETDATE())
```

Debemos mencionar que la sentencia UNION permite unir dos o más sentencias; siempre y cuando cuenten con la misma cantidad de columnas resultantes en la consulta.

	FUNCION	VALOR
1	AÑO ACTUAL	2015
2	DIA ACTUAL	13
3	HORA ACTUAL	17
4	MES ACTUAL	7
5	MILISEGUNDOS ACTUAL	300
6	MINUTO ACTUAL	3
7	NUMERO DE DIA DEL AÑO	194
8	NUMERO DE DIA EN LA SEMANA	1
9	NUMERO DE SEMANA	29
10	SEGUNDO ACTUAL	55

Figura 88: Listado de fechas y horas desde la fecha actual
Fuente.- Tomado desde SQL Server 2014

Script que permita mostrar los datos de la factura con el siguiente formato:

NUMERO	FECHA DE REG.(DMY)	FECHA DE REG.
99	99/99/9999	9999/99/99
99	99/99/9999	9999/99/99

```

SELECT F.NUM_FAC AS NUMERO,
       CAST(DATEPART(DD,F.FEC_FAC) AS VARCHAR(2))+ '/' +
       CAST(DATEPART(MM,F.FEC_FAC) AS VARCHAR(2))+ '/' +
       CAST(DATEPART(YYYY,F.FEC_FAC) AS CHAR(4)) AS [FECHA DE REG.],
       F.FEC_FAC AS [FECHA DE REG.]
FROM TB_FACTURA F
GO

```

	NUMERO	FECHA DE REG.	FECHA DE REG.
1	100	7/6/1998	1998-06-07
2	101	9/6/1998	1998-06-09
3	102	9/1/1998	1998-01-09
4	103	9/6/1998	1998-06-09
5	104	10/1/1998	1998-01-10
6	105	10/10/1998	1998-10-10
7	106	10/5/1998	1998-05-10
8	107	10/9/1998	1998-09-10
9	108	10/3/1998	1998-03-10

Figura 89: Listado de fechas desde la tabla factura
Fuente.- Tomado desde SQL Server 2014

3.1.4.5 Función DateDiff

Devuelve un valor numérico entero que representa el recuento entre dos fechas especificadas. Su formato es:

```
DATEDIFF ( ARGUMENTO , FECHAINICIO , FECHAFINAL )
```

Script que permite mostrar la diferencia de valores entre la fecha de cancelación y la fecha de facturación obtenidas desde la tabla factura con el siguiente formato:

NUMERO	DIAS	MESES	AÑOS	SEMANAS
999	99	99	99	99
999	99	99	99	99

```

SELECT F.NUM_FAC,
       DATEDIFF(DD,F.FEC_CAN,F.FEC_FAC) AS [DIAS],
       DATEDIFF(MM,F.FEC_CAN,F.FEC_FAC) AS [MESES],
       DATEDIFF(YY,F.FEC_CAN,F.FEC_FAC) AS [AÑOS],
       DATEDIFF(WW,F.FEC_CAN,F.FEC_FAC) AS [SEMANAS]
FROM TB_FACTURA F
GO

```

	NUM_FAC	DIAS	MESES	AÑOS	SEMANAS
1	100	30	1	0	5
2	101	32	1	0	5

Figura 90: Listado de fechas desde la factura
Fuente.- Tomado desde SQL Server 2014

Script que permita mostrar la diferencia de valores entre la fecha de atención y la fecha actual obtenidas desde la tabla Orden de Compra con el siguiente formato:

NUM_OCO	DIAS	MESES	AÑOS	SEMANAS
999	99	99	99	99
999	99	99	99	99

```
SELECT O.NUM_OCO,
       DATEDIFF(DD,O.FEC_ATE,GETDATE()) AS [DIAS],
       DATEDIFF(MM,O.FEC_ATE,GETDATE()) AS [MESES],
       DATEDIFF(YY,O.FEC_ATE,GETDATE()) AS [AÑOS],
       DATEDIFF(WW,O.FEC_ATE,GETDATE()) AS [SEMANAS]
  FROM TB_ORDEN_COMPRA O
GO
```

	NUM_OCO	DIAS	MESES	AÑOS	SEMANAS
1	OC001	6066	199	17	867
2	OC002	6126	201	17	875
3	OC003	6364	209	17	909

Figura 91: Listado de fechas desde las órdenes de compra
Fuente.- Tomado desde SQL Server 2014

3.1.5 Ejercicios de aplicación

Actividad 01: VENTAS2017

Utilizaremos la base de datos **VENTAS2017** para realizar las consultas. El script de la base de datos se debe descargar de Moodle:

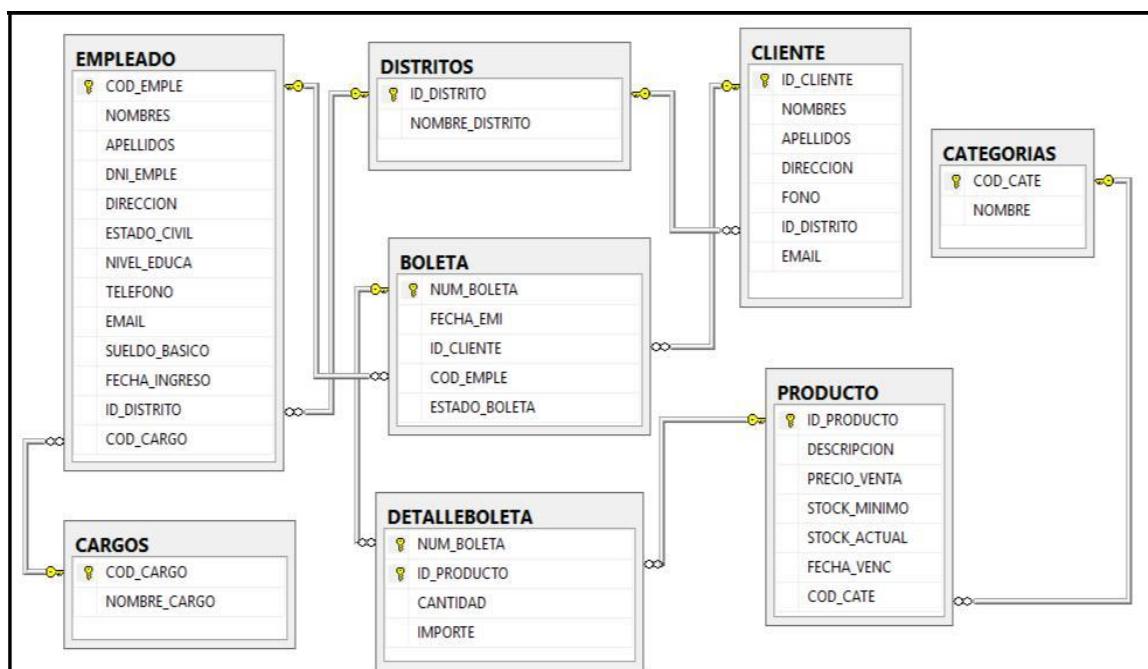


Figura 92: Base de datos "Ventas2017"
Fuente. - Tomado desde SQL Server 2014

CONSULTAS BÁSICAS

- Seleccionar todos los registros de la tabla cliente.
- Seleccionar los campos id_cliente, nombres, apellidos, dirección de la tabla cliente.
- Seleccionar los campos id_producto, descripción y precio_venta de la tabla producto utilizando Alias P.
- Seleccionar los campos id_cliente, nombres y apellidos concatenados, fono, email de la tabla cliente. Utilizar Alias y cambiar el nombre de las columnas de respuesta por Código, Cliente, Teléfono, Correo.
- Seleccione todos distintos tipos de distrito (id_distrito) que aparecen en la tabla cliente. Utilizar distinct.
- Seleccionar los campos id_producto, descripción, stock_actual y stock_minimo de la tabla producto. utilizar alias, además cambiando los nombres de las columnas a Código, Descripción, Stock actual, Stock mínimo y ordenar los registros de forma ascendente por descripción de producto.
- Mostrar todos los registros de la tabla detalle boleta, pero cuya cantidad sea mayor a 5. Utilizar alias y sentencia Where.
- Mostrar todos los campos de la tabla producto, pero de aquellos cuyo código del producto sea PRO002 y PRO004. Utilizar Alias, sentencia Where y operadores de comparación y lógicos.
- Seleccionar todos los campos de la tabla cliente, pero de aquellos cuyo código de distrito sea D006. Utilizar Alias y operador de comparación.
- Seleccionar id_producto, descripción y precio_venta de la tabla producto, pero de aquellos cuyo precio de venta esté entre 30 y 100. Utilizar alias, operadores de comparación y lógicos.
- Seleccionar todos los campos de la tabla cliente a excepción de los clientes de distrito D003 y D010. Utilizar alias, operadores de comparación y operador lógico NOT
- Seleccionar todos los campos de la tabla Empleado cuyo nombre comienza con R y apellidos con T. Utilizar alias, operadores lógicos y sentencia LIKE
- Seleccionar todos los campos de la tabla Empleado cuya segunda letra de apellidos es A y estado civil soltero (S). Utilizar alias, operadores lógicos y sentencia LIKE
- Mostrar todos los campos de la tabla producto, pero de aquellos que su stock actual esté entre 60 y 120. Utilizar alias y sentencia BETWEEN
- Mostrar todos los campos de la tabla Empleado, pero de aquellos que sean de los distritos D005, D007 y D009. Utilizar Alias y sentencia IN
- Mostrar los campos Num_Boleta, Id_cliente y el día en que se realizó la venta. Mostrar como nombre de columnas: Número Boleta, Código_cliente y Día de Venta. Utilizar Alias y función DAY.
- Mostrar todos los campos de la tabla empleado, pero sólo aquellos que hayan sido contratados (Fecha_ingreso) en el año 2015.
- Mostrar todos los campos de la tabla empleado, pero sólo aquellos que hayan sido contratados (Fecha_ingreso) en el año 2014 y de estado civil casado y nivel de educación superior.
- Mostrar todos los campos de la tabla producto, pero de aquellos que su categoría sea C10 o C05

Mostrar los campos Num_Boleta, Id_cliente, el día, el mes y el año en que se realizó la venta por separado. Mostrar como nombre de columnas: Número Boleta, Código Cliente, Día de Venta, mes de venta y Años de venta, sólo aquellas ventas realizadas a partir del año 2014 y ordenados de forma ascendente por código de cliente. Utilizar Alias y funciones Day, Month y Year.

3.1.6 E ejercicios de reforzamiento

Actividad 01: BD_STAR

PARTE 1: DDL Lenguaje de Definición de Datos

Cree la base de datos **BD_STAR** en la carpeta C:\PELICULAS\DATOS\ con la siguiente configuración:

Archivo de datos: Un tamaño inicial de 10MB, máximo de 40MB y un factor de crecimiento de 5MB.

Archivo secundario: Un tamaño inicial de 5MB, máximo de 30MB y un factor de crecimiento de 10%.

Archivo de transacciones: Un tamaño inicial de 4MB, máximo de 20MB y un factor de crecimiento de 10MB.

Active la base de datos BD_STAR.

Implemente el diagrama de base de datos en SQL Server 2014.

Actualice la base de datos y cree sus tablas. Deberá identificar adecuadamente el tipo de dato para sus atributos y si son nulos o no nulos.

- Cree sus llaves primarias y foráneas.
- Cree una restricción que permita únicamente valores mayores a 0 en el campo DUR_PEL (duración de la película) de la tabla PELÍCULAS.
- Cree una restricción que no permita ingresar fechas de nacimiento mayores a la fecha del día.
- Por defecto registrar el color de ojos NEGROS.

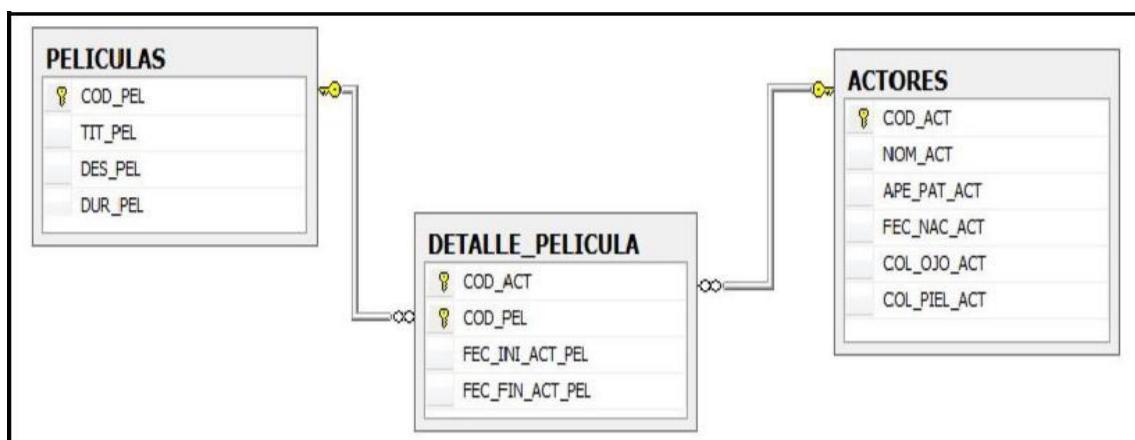


Figura 93: Base de datos “BD_STAR”
Fuente. - Tomado desde SQL Server 2014

PARTE 2: DML Lenguaje de Manipulación de Datos

Inserte 4 registros en cada tabla.

Seleccione los actores cuyo color de piel sea cobrizo.

Muestre los actores con código menor a 5.

Muestre las películas cuya duración se encuentre entre 80 y 120 minutos.

Elimine los detalles de película que registran películas cuyo código de película se encuentre entre 1 y 3.

Actualice el color de ojos a 'MARRÓN OSCURO' de los actores con código 1 o 4.

Actividad 02: LIGA DE SURCO

DER FINAL

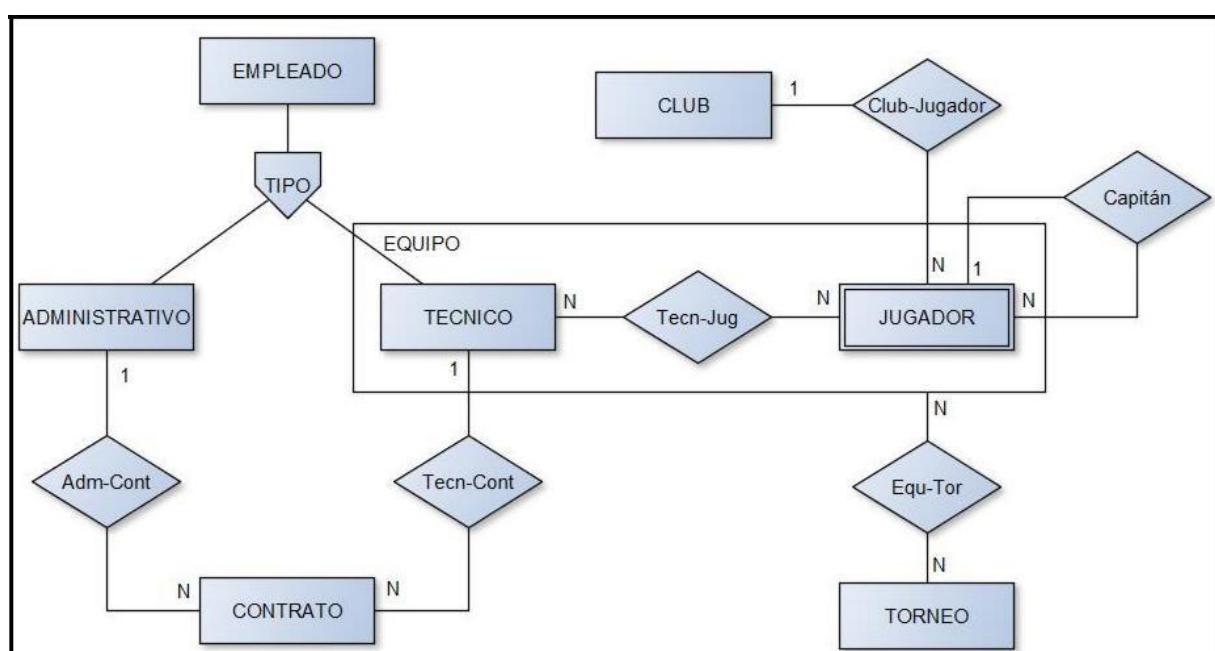


Figura 94: DER final de la "Liga de surco"

Fuente. - Tomado desde SQL Server 2014

b) Implementación en SQL Server 2014:

```

USE MASTER
GO

--VALIDANDO LA BASE DE DATOS LIGASURCO

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

--CREANDO LA BASE DE DATOS LIGASURCO
CREATE DATABASE LIGASURCO
GO
  
```

```
--ABRIENDO LA BASE DE DATOS LIGASURCO
```

```
USE LIGASURCO
```

```
GO
```

```
--CREANDO LA TABLA CLUB
```

```
CREATE TABLE CLUB (
    COD_CLU      CHAR      (4) NOT NULL,
    NOM_CLU      VARCHAR   (40) NOT NULL,
    FEC_CLU      DATE      NOT NULL,
    DIR_CLU      VARCHAR   (40) NOT NULL,
    LOC_CLU      INT       NOT NULL
)
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA CLUB
```

```
ALTER TABLE CLUB
    ADD PRIMARY KEY (COD_CLU)
GO
```

```
--CREANDO LA TABLA JUGADOR
```

```
CREATE TABLE JUGADOR (
    COD_JUG      CHAR      (4) NOT NULL,
    COD_CLU      CHAR      (4) NOT NULL,
    NOM_JUG      VARCHAR   (20) NOT NULL,
    PAT_JUG      VARCHAR   (20) NOT NULL,
    MAT_JUG      VARCHAR   (20) NOT NULL,
    DIR_JUG      VARCHAR   (40) NOT NULL,
    SEX_JUG      CHAR      (1) NOT NULL,
    FEC_JUG      DATE      NOT NULL,
    COD_JUG_CAP CHAR      (4) NOT NULL
)
GO
```

```
--CREANDO CLAVES COMPUESTAS EN LA TABLA JUGADOR
```

```
ALTER TABLE JUGADOR
    ADD PRIMARY KEY (COD_JUG, COD_CLU)
GO
```

```
--REALIZANDO UNA RECURSIVIDAD A LA TABLA JUGADOR
```

```
ALTER TABLE JUGADOR
    ADD FOREIGN KEY (COD_JUG, COD_CLU) REFERENCES JUGADOR
GO
```

```
--RELACIONANDO LA TABLA JUGADOR CON LA TABLA CLUB
```

```
ALTER TABLE JUGADOR  
ADD FOREIGN KEY (COD_CLU) REFERENCES CLUB  
GO
```

```
--CREANDO LA TABLA EMPLEADO
```

```
CREATE TABLE EMPLEADO (  
COD_EMP      CHAR      (4)  NOT NULL,  
NOM_EMP      VARCHAR   (20) NOT NULL,  
PAT_EMP      VARCHAR   (20) NOT NULL,  
MAT_EMP      VARCHAR   (20) NOT NULL,  
DIR_EMP      VARCHAR   (40) NOT NULL,  
SEX_EMP      CHAR      (1)  NOT NULL,  
FEC_EMP      DATE      NOT NULL,  
TEL_EMP      CHAR      (15) NOT NULL,  
CEL_EMP      CHAR      (15) NOT NULL  
)  
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA EMPLEADO
```

```
ALTER TABLE EMPLEADO  
ADD PRIMARY KEY (COD_EMP)  
GO
```

```
--CREANDO LA TABLA TECNICO
```

```
CREATE TABLE TECNICO (  
COD_EMP      CHAR      (4)  NOT NULL,  
ESP_TEC      VARCHAR   (20) NOT NULL  
)  
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA TECNICO
```

```
ALTER TABLE TECNICO  
ADD PRIMARY KEY (COD_EMP)  
GO
```

```
--RELACIONANDO LA TABLA TECNICO CON LA TABLA EMPLEADO
```

```
ALTER TABLE TECNICO  
ADD FOREIGN KEY (COD_EMP) REFERENCES EMPLEADO  
GO
```

```
--CREANDO LA TABLA ADMINISTRATIVO

CREATE TABLE ADMINISTRATIVO (
COD_EMP      CHAR      (4) NOT NULL,
NIV_ADM      VARCHAR   (20) NOT NULL
)
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA ADMINISTRATIVO
```

```
ALTER TABLE ADMINISTRATIVO
ADD PRIMARY KEY (COD_EMP)
GO
```

```
--RELACIONANDO LA TABLA ADMINISTRATIVO CON LA TABLA EMPLEADO
```

```
ALTER TABLE ADMINISTRATIVO
ADD FOREIGN KEY (COD_EMP) REFERENCES
EMPLEADO GO
```

```
--CREANDO LA TABLA EQUIPO
```

```
CREATE TABLE EQUIPO (
COD_CLU      CHAR      (4) NOT NULL,
COD_JUG      CHAR      (4) NOT NULL,
COD_EMP      CHAR      (4) NOT NULL,
CAT_EQU      VARCHAR   (40) NOT NULL,
DIS_EQU      VARCHAR   (40) NOT NULL
)
GO
```

```
--CREANDO CLAVES COMPUESTAS EN LA TABLA EQUIPO
```

```
ALTER TABLE EQUIPO
ADD PRIMARY KEY (COD_CLU, COD_JUG, COD_EMP)
GO
```

```
--RELACIONANDO LA TABLA EQUIPO CON LA TABLA JUGADOR
```

```
ALTER TABLE EQUIPO
ADD FOREIGN KEY (COD_CLU, COD_JUG) REFERENCES JUGADOR
GO
```

```
--RELACIONANDO LA TABLA EQUIPO CON LA TABLA TECNICO
```

```
ALTER TABLE EQUIPO  
ADD FOREIGN KEY (COD_EMP) REFERENCES TECNICO  
GO
```

```
--CREANDO LA TABLA CONTRATO
```

```
CREATE TABLE CONTRATO (  
NUM_CON      INT          NOT NULL,  
FIN_CON      DATE         NOT NULL,  
FTE_CON      DATE         NOT NULL,  
COD_EMP      CHAR(4)     NOT NULL  
)  
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA CONTRATO
```

```
ALTER TABLE CONTRATO  
ADD PRIMARY KEY (NUM_CON)  
GO
```

```
--RELACIONANDO LA TABLA CONTRATO CON LA TABLA EMPLEADO
```

```
ALTER TABLE CONTRATO  
ADD FOREIGN KEY (COD_EMP) REFERENCES EMPLEADO  
GO
```

```
--CREANDO LA TABLA TORNEO
```

```
CREATE TABLE TORNEO (  
COD_TOR      CHAR(4)    NOT NULL,  
NOM_TOR      VARCHAR(40) NOT NULL,  
FIN_TOR      DATE       NOT NULL,  
FTE_TOR      DATE       NOT NULL,  
DIS_TOR      VARCHAR(40) NOT NULL  
)  
GO
```

```
--ASIGNANDO LA CLAVE PRINCIPAL A LA TABLA TORNEO
```

```
ALTER TABLE TORNEO  
ADD PRIMARY KEY (COD_TOR)  
GO
```

```
--CREANDO LA TABLA EQUIPO_TORNEO
CREATE TABLE EQUIPO_TORNEO (
    COD_TOR      CHAR(4) NOT NULL,
    COD_CLU      CHAR(4) NOT NULL,
    COD_JUG      CHAR(4) NOT NULL,
    COD_EMP      CHAR(4) NOT NULL
)
GO

--ASIGNANDO LA CLAVES COMPUSTAS A LA TABLA EQUIPO_TORNEO
ALTER TABLE EQUIPO_TORNEO
ADD PRIMARY KEY (COD_TOR, COD_CLU, COD_JUG, COD_EMP)
GO

--RELACIONANDO LA TABLA EQUIPO_TORNEO CON LA TABLA TORNEO
ALTER TABLE EQUIPO_TORNEO
ADD FOREIGN KEY (COD_TOR) REFERENCES TORNEO
GO

--RELACIONANDO LA TABLA EQUIPO_TORNEO CON LA TABLA EQUIPO
ALTER TABLE EQUIPO_TORNEO
ADD FOREIGN KEY (COD_JUG, COD_CLU, COD_EMP) REFERENCES EQUIPO
GO
```

Diagrama final desde SQL Server 2014:

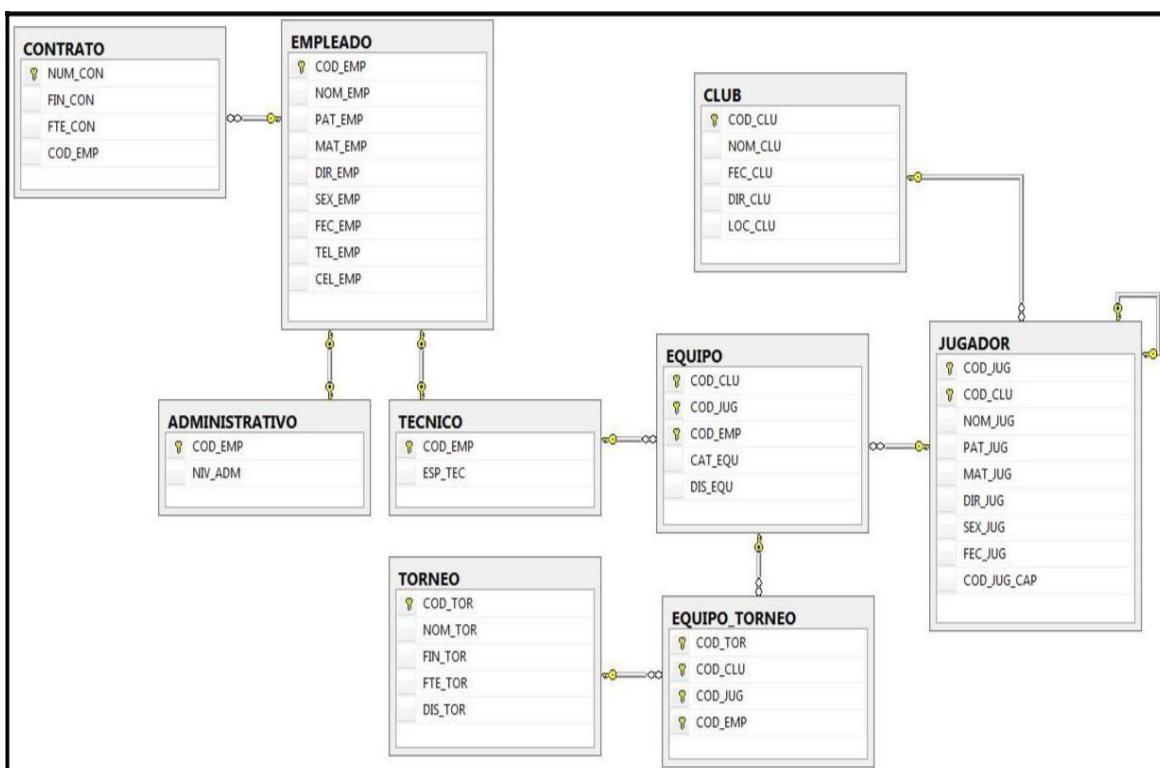


Figura 95: Modelo relacional “Liga de surco”
Fuente. - Tomado desde SQL Server 2014

Debe tener en cuenta:

Active la base de datos “**LIGASURCO**”

Restricciones

- El nombre del club debe ser único
- Su dirección del jugador debe ser único
- En el campo sexo de los jugadores, solo debe permitir “Masculino”
- El teléfono del empleado por defecto debe tener el siguiente valor “0000000000”
Código del empleado debe empezar con las letras **E**, seguido de 2 ceros y un número correlativo: Ejemplo: **E001**
- Código del jugador debe empezar con la letra **J**, seguido de 2 ceros y un número correlativo: Ejemplo: **J001**
- Código del torneo debe empezar con la letra **T**, seguido de 2 ceros y un número correlativo: Ejemplo: **T001**
- Código del club debe empezar con la letra **C**, seguido de 2 ceros y un número correlativo: Ejemplo: **C001**

Inserte 3 registros en cada tabla.

Actualice su dirección del jugador con “AV. BRASIL # 1160”, para el jugador que tiene el código **J003**.

Actualice su nuevo club del jugador que tiene código **C003**, para el jugador que tiene código **J002**

Muestre los empleados que haya sido contratados en el año 2017

Muestre los jugadores cuyo nombre comienza con la letra “**M**”.

Elimine a los equipos que se encuentran en segunda división.

Resumen

SQL es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft SQL.

La sentencia SELECT permite componer una consulta que puede ser interpretado por el servidor de base de datos; el cual emitirá un resultado expresado en registros.

La consulta condicionada permite obtener registros desde una base de datos de acuerdo al criterio especificado por el usuario usando la cláusula Where dentro de la sentencia SELECT.

Cada vez que haga una consulta haciendo referencia a un valor numérico, la comparación será escrita de la siguiente manera:

WHERE CAMPO (=, >, <, etc) VALOR

La cláusula Order by especifica que los resultados de la consulta deben ser ordenados en sentido ascendente o descendente, basándose en los valores de una o más columnas.

Las consultas condicionadas soportan operadores lógicos como AND, OR, XOR, IS y NOT.

Las funciones para el manejo de fechas son DAY, MONTH, YEAR, GETDATE y DATEPART los cuales se pueden implementar dentro de una consulta SELECT.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

<http://www.tutorialspoint.com/sql/sql-select-query.htm>

Sintaxis y casos desarrollados de la sentencia SELECT.

<http://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Como-realizar-consultas-simples-con-SELECT.aspx>

Fundamentos del uso de la sentencia SELECT.

[https://msdn.microsoft.com/es-es/library/ms187731\(v=sql.120\).aspx](https://msdn.microsoft.com/es-es/library/ms187731(v=sql.120).aspx) Casos desarrollados de la sentencia SELECT explicados por Microsoft.

<http://technet.microsoft.com/es-es/library/ms174420.aspx>

Tutorial para el uso de la función datepart ()

<http://technet.microsoft.com/es-es/library/ms189794.aspx> Tutorial para el uso de la función datediff()

<http://technet.microsoft.com/es-es/library/ms187922.aspx> Tutorial para el uso del operador BETWEEN

<http://technet.microsoft.com/es-es/library/ms179859.aspx> Tutorial para el uso del operador LIKE



INTRODUCCIÓN A LA PROGRAMACIÓN EN TRANSACT / SQL

LOGROS DE LA UNIDAD DE APRENDIZAJE

Al término de la cuarta unidad, el estudiante recupera información eficazmente de una tabla de la base de datos implementando procedimientos almacenados sin y con parámetros de entrada.

TEMARIO

4.1. Tema 9: Introducción a la programación en SQL SERVER 2014

- 4.1.1. Declaración de variables locales
- 4.1.2. Procedimientos almacenados con una tabla.
- 4.1.3. Procedimientos almacenados con uno y dos parámetros de entrada.
- 4.1.4. Procedimientos almacenados de mantenimiento de datos (INSERT, UPDATE y DELETE)

ACTIVIDADES PROPUESTAS

Los estudiantes entienden la importancia de la implementación de procedimientos almacenados en una base de datos.

Los estudiantes implementan procedimientos almacenados usando uno o más parámetros de entrada.

Los alumnos integran las consultas dentro de los procedimientos almacenados.

4.1 Introducción a la programación en SQL SERVER 2014

Como sabemos SQL es un lenguaje de consulta estructurada, el cual nos permite realizar consultas sobre la información almacenada en la base de datos, pero hasta ahora no hemos usado la real potencia que presenta cuando se le incorpora funcionalidad que solo lo poseen los lenguajes de programación.

Transact SQL es el lenguaje de programación que proporciona SQL para ampliar la forma de obtener la información, es así que en esta unidad estudiaremos las variables locales, las sentencias de control de flujo como if y case y el control de repeticiones con while.

Finalmente, podemos decir que Transact SQL proporciona palabras clave especiales llamadas lenguaje de control de flujo que permiten controlar el flujo de ejecución de las instrucciones. El lenguaje de control de flujo se puede utilizar en instrucciones sencillas, lotes, procedimientos almacenados y disparadores.

4.1.1 Declaración de variables locales

Una variable es un espacio de memoria a la que se asigna un determinado valor. Este valor puede cambiar durante el lote o el procedimiento almacenado donde se utiliza la variable. SQL Server presenta dos tipos de variables: locales y globales.

Las **variables locales** están definidas por el usuario y es aquí donde nos enfocaremos en esta unidad,

Mientras que las **variables globales** las suministra el sistema y están predefinidas; es decir, no podrán el usuario solo podrá invocarlas, pero no manipularlas.

Por otra parte, las variables locales se declaran, nombran y escriben mediante la palabra clave **DECLARE**, y reciben un valor inicial mediante una instrucción **SELECT** o **SET**. Dichas variables deben declararse, recibir un valor y utilizarse en su totalidad dentro del mismo lote o procedimiento; por ningún motivo debemos ejecutar el conjunto de instrucciones por separado.

Así mismo, los nombres de las variables locales deben empezar con el símbolo “@”. A cada variable local se le debe asignar un tipo de dato definido por el usuario.

Formato de declaración de variable:

```
DECLARE @VARIABLE TIPO_DATOS
```

Formato de asignación de valor a la variable:

```
SET @VARIABLE = VALOR O SENTENCIA  
SELECT @VARIABLE = VALOR O SENTENCIA
```

Veamos algunos casos del uso de variables:

Script que permita declarar una variable de tipo entero, se le asigne un valor y lo muestre mediante la sentencia SELECT:

```
DECLARE @NUMERO INT
SET @NUMERO = 10
SELECT @NUMERO
```

Iniciamos declarando la variable local número de tipo INT, luego asignamos el valor 10 a la variable mediante la instrucción SET y finalmente se muestra el valor mediante la sentencia SELECT. Otra forma de expresar la sentencia, podría ser de la siguiente manera:

```
DECLARE @NUMERO INT = 10
SELECT @NUMERO
```

Como sucede en los lenguajes de programación se puede declarar y asignar en una misma línea; si necesita realizar más declaraciones y asignaciones será cuestión de separarlos por medios de comas. Finalmente, podríamos optar por la siguiente script:

```
DECLARE @NUMERO INT = 10
PRINT @NUMERO
```

La instrucción PRINT permite imprimir un determinado valor, si esta impresión esta acompañada de un texto representativo del valor se debe tener en cuenta que los valores sean de tipo Char o Varchar, de otra manera tendríamos que aplicar la sentencia CAST o CONVERT.

```
CAST(VARIABLE O VALOR AS TIPO_DATOS)
CONVERT(TIPO_DATOS, VARIABLE O VALOR)
```

Finalmente podríamos presentar la siguiente alternativa de solución:

```
DECLARE @NUMERO INT = 10
PRINT 'EL NUMERO ES: '+CAST(@NUMERO AS VARCHAR(10))
```



Figura 96: Mostrando el valor asignado a una variable local
Fuente. - Tomado desde SQL Server 2014

Script que permita calcular el promedio de un determinado alumno, el cual cuenta con cuatro notas y estas son de tipo entero:

```
DECLARE @NOMBRE VARCHAR (30) = 'JUAN PEREZ', @N1 INT = 10,
        @N2 INT = 16, @N3 INT = 15,
        @N4 INT = 20, @PROM DECIMAL (5,2)

SET @PROM = (@N1 + @N2 + @N3 + @N4)/4.0

SELECT @NOMBRE AS ALUMNO, @PROM AS PROMEDIO
```

	ALUMNO	PROMEDIO
1	JUAN PEREZ	15.25

Figura 97: Promedio de notas de un alumno usando variables locales

Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar la razón social de un determinado proveedor a partir del código del mismo:

```
DECLARE @COD CHAR (4) = 'PR01', @RAZ VARCHAR (30)

SELECT @RAZ = RAZ_SOC_PRV FROM TB_PROVEEDOR WHERE COD_PRV=@COD
SELECT @COD AS CODIGO, @RAZ AS PROVEEDOR
```

	CODIGO	PROVEEDOR
1	PR01	Faber Castell

Figura 98: Listado de proveedor con código PR01

Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar los datos de un determinado vendedor añadiendo a la lista de columnas mostradas el nombre del distrito con el siguiente formato:

CODIGO	NOMBRES	DISTRITO
XXX	XXXXX XXXXXXXXX	XXXXXXXXXX

```
DECLARE @COD CHAR(3)='V01', @CODDIS CHAR(3), @DIST VARCHAR(30)

--Obteniendo el código del distrito del vendedor
SELECT @CODDIS = COD_DIS FROM TB_VENDEDOR WHERE COD_VEN = @COD

--Obteniendo el nombre del distrito a partir del código
SELECT @DIST = NOM_DIS FROM TB_DISTRITO WHERE COD_DIS = @CDDIS

--Mostrando las columnas solicitadas
SELECT V.COD_VEN AS CODIGO,
       V.NOM_VEN+SPACE(1)+V.APE_VEN AS NOMBRES,
       @DIST AS DISTRITO
  FROM TB_VENDEDOR V
 WHERE V.COD_VEN = @COD
```

	CODIGO	NOMBRES	DISTRITO
1	V01	JUANA MESES	Chomillos

Figura 99: Listado del vendedor con código V01

Fuente. - Tomado desde SQL Server 2014

4.1.2 Procedimientos almacenados con una tabla

Los procedimientos almacenados son grupos formados por instrucciones SQL y el lenguaje de control de flujo. Cuando se ejecuta un procedimiento, se prepara un plan de ejecución para que la subsiguiente ejecución sea muy rápida. Los procedimientos almacenados pueden:

Incluir parámetros

Llamar a otros procedimientos

Devolver un valor de estado a un procedimiento de llamada o lote para indicar el éxito o el fracaso del mismo y la razón de dicho fallo.

Devolver valores de parámetros a un procedimiento de llamada o lote

Ejecutarse en SQL Server remotos

Así mismo, la posibilidad de escribir procedimientos almacenados mejora notablemente la potencia, eficacia y flexibilidad de SQL. Los procedimientos compilados mejoran la ejecución de las instrucciones y lotes. Además, los procedimientos almacenados pueden ejecutarse en otro SQL Server si el servidor del usuario y el remoto están configurados para permitir logins remotos.

Por otra parte, los procedimientos almacenados se diferencian de las instrucciones SQL ordinarias y de lotes de instrucciones SQL en que están precompilados. La primera vez que se ejecuta un procedimiento, el procesador de consultas SQL Server lo analiza y prepara un plan de ejecución que se almacena en forma definitiva en una tabla de sistema. Posteriormente, el procedimiento se ejecuta según el plan almacenado. Puesto que ya se ha realizado la mayor parte del trabajo de procesamiento de consultas, los procedimientos almacenados se ejecutan casi de forma instantánea.

Finalmente, los procedimientos almacenados se crean con **CREATE PROCEDURE**. Para ejecutar un procedimiento almacenado, ya sea un procedimiento del sistema o uno definido por el usuario, use el comando **EXECUTE**.

4.1.2.1 Creando un procedimiento almacenado

Cuando se crea un procedimiento almacenado el servidor de base de datos lo agrega como un objeto dentro de la base de datos actual. Es así que, podemos recuperar las sentencias implementadas en un determinado procedimiento almacenado en cualquier momento.

Formato de creación del procedimiento almacenado sin parámetros:

```
CREATE PROCEDURE NOMBRE_PROCEDIMIENTO  
AS  
    SENTENCIAS  
GO
```

O también podríamos usar el siguiente formato:

```
CREATE PROC NOMBRE_PROCEDIMIENTO  
AS  
BEGIN  
    SENTENCIAS  
END  
GO
```

En el formato debemos considerar que es análogo colocar PROCEDURE y PROC, la instrucción BEGIN...END, solo marca el inicio y fin del procedimiento almacenado (opcional) y las sentencias pueden ser consultas o control de flujo ya que un procedimiento no siempre devuelve un valor resultante. Seguidamente veamos como validar la existencia de un procedimiento almacenado:

```
IF OBJECT_ID (' NOMBRE_PROCEDIMIENTO ') IS NOT NULL  
    DROP PROC NOMBRE_PROCEDIMIENTO  
GO  
CREATE PROCEDURE NOMBRE_PROCEDIMIENTO  
AS  
    SENTENCIAS  
GO
```

Finalmente, veremos la sentencia que permite ejecutar los procedimientos almacenados:

```
EXECUTE NOMBRE_PROCEDIMIENTO
```

O también podríamos usar la siguiente sentencia: EXEC NOMBRE

Si necesitamos visualizar el bloque de sentencias que compone el procedimiento almacenado puede usar la siguiente sentencia:

```
EXEC SP_HELPTEXT 'NOMBRE DEL PROCEDIMIENTO'
```

Ejemplo:

Veamos un caso de implementación de procedimiento almacenado en la cual permita listar todos los registros de la tabla proveedor. Además, deberá probar el procedimiento de tal forma que visualice el resultado esperado.

```
--Validando la existencia del procedimiento almacenado  
IF OBJECT_ID('SP_LISTAPROVEEDOR') IS NOT  
    NULL DROP PROC SP_LISTAPROVEEDOR  
GO  
  
--Creando el procedimiento almacenado  
CREATE PROC SP_LISTAPROVEEDOR  
AS  
    SELECT * FROM TB_PROVEEDOR  
GO  
  
--Ejecutando el procedimiento  
EXECUTE SP_LISTAPROVEEDOR  
GO  
  
--Mostrando las sentencias del procedimiento almacenado  
EXEC SP_HELPTEXT 'SP_LISTAPROVEEDOR'
```

Text	
1	
2	-Creando el procedimiento almacenado
3	CREATE PROC SP_LISTAPROVEEDOR
4	AS
5	SELECT * FROM TB_PROVEEDOR

Figura 100: Sentencias de un procedimiento almacenado
Fuente. - Tomado desde SQL Server 2014

4.1.2.2 Modificar un procedimiento almacenado

La modificación de un procedimiento almacenado permite realizar alguna modificación dentro del bloque de sentencias que la compone. Pero tenga en cuenta que, si valida el procedimiento almacenado ya no será necesario modificarlo solo ejecutar la creación del procedimiento con su respectiva sentencia de validación, vista en el punto anterior. Su formato es:

```
ALTER PROC NOMBRE_PROCEDIMIENTO
AS
    SENTENCIAS
GO
```

Para modificar el procedimiento almacenado deberá contar con el bloque de sentencias que la componen, eso quiere decir que no se pueden hacer modificaciones a un sector específico del procedimiento almacenado.

Ejemplo:

Veamos la modificación del procedimiento almacenado **SP_LISTAPROVEEDOR** en la cual se listarán campos específicos de la tabla proveedor. Además, deberá probar el procedimiento para visualizar el resultado esperado.

```
--Modificando el procedimiento almacenado
ALTER PROC SP_LISTAPROVEEDOR
AS
    SELECT P.COD_PRV AS CODIGO,
           P.RAZ_SOC_PRV AS [RAZON SOCIAL],
           P.TEL_PRV AS TELEFONO,
           P.COD_DIS AS [CODIGO DE DISTRITO]
    FROM TB_PROVEEDOR P
GO

--Ejecutando el procedimiento
EXECUTE SP_LISTAPROVEEDOR
GO
```

4.1.2.3 Eliminar un procedimiento almacenado

La eliminación del procedimiento almacenado es considerada como la inhabilitación y eliminación del objeto de tipo procedimiento almacenado. Su formato es:

```
DROP PROC NOMBRE_PROCEDIMIENTO
```

Ejemplo:

Veamos como eliminar el procedimiento almacenado SP_LISTAPROVEEDOR, no se olvide de visualizar si el procedimiento ya no se encuentre entre los objetos de la base de datos.

```
DROP PROC SP_LISTAPROVEEDOR  
GO
```

Para visualizar los procedimientos almacenados registrados en la base de datos:

```
SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES  
WHERE ROUTINE_TYPE = 'PROCEDURE'  
ORDER BY ROUTINE_NAME  
GO
```

4.1.3. Procedimientos almacenados con uno y dos parámetros de entrada

Los parámetros dentro de un procedimiento almacenado cumplen una función importante ya que por medio de ellas enviaremos valores al procedimiento, es así que las sentencias implementadas dentro del procedimiento usaran a dichos parámetros como una variable local.

El formato del procedimiento con parámetros es:

```
CREATE PROCEDURE NOMBRE (@PARAMETRO1 TIPO, @PARAMETRO2 TIPO)  
AS  
SENTENCIAS  
GO
```

Otra forma de implementar el procedimiento almacenado es:

```
CREATE PROCEDURE NOMBRE  
@PARAMETRO1 TIPO,  
@PARAMETRO2 TIPO  
AS  
SENTENCIAS  
GO
```

Su formato de ejecución varía dependiendo de la cantidad de parámetros declarados, estos deberán ser enviados en el mismo orden que fueron implementadas, esto es debido al tipo de datos que presenta cada parámetro del procedimiento almacenado:

```
EXEC NOMBRE 'VALOR1', 'VALOR2'
```

Ejemplo:

Veamos un caso de implementación de procedimiento almacenado con parámetros en la cual permita listar todos los registros de la tabla proveedor según las iniciales del representante vendedor (REP_VEN).

```
--Validando la existencia del procedimiento almacenado
IF OBJECT_ID('SP_LISTAPROVEEDORxINICIAL') IS NOT NULL
    DROP PROC SP_LISTAPROVEEDORxINICIAL
GO

--Creando el procedimiento almacenado
CREATE PROC SP_LISTAPROVEEDORxINICIAL (@INICIAL CHAR(1))
AS
    SELECT P. *
    FROM TB_PROVEEDOR P
    WHERE REP_VEN LIKE @INICIAL+'%'
GO

--Ejecutando el procedimiento
EXECUTE SP_LISTAPROVEEDORxINICIAL 'V'
GO
```

	COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
1	PR20	Bari	Av. Amaldo Marquez 1219	NULL	D02	Vanesa Quintana

Figura 101: Listado de proveedores cuyo nombre de su representante inicie con “V”

Fuente. - Tomado desde SQL Server 2014

4.1.3.1 Estructura condicional IF...ELSE

La instrucción IF permite condicionar la ejecución de una o mas sentencias. Las sentencias solo se ejecutarán si la condición es verdadera, es decir, si devuelve TRUE, caso contrario, no ejecutará ninguna sentencia. Mientras que, la cláusula ELSE es una alternativa a la instrucción IF que solo permite ejecutar las sentencias especificadas dentro del bloque ELSE.

Finalmente, debemos considerar que cada bloque de la estructura necesitará más de dos sentencias por lo tanto será necesario usar el bloque BEGIN END para marcar un punto de inicio y final del bloque.

El formato de la estructura condicional IF es:

```
IF ( CONDICION )
BEGIN
    SENTENCIAS_VERDADERAS
END
ELSE
BEGIN
    SENTENCIAS_FALSAS
END
```

Ejemplo:

Veamos un caso del uso de la estructura IF. Procedimiento Almacenado que permita listar los datos de los proveedores según el código de distrito, en caso no existan registros de vendedores mostrar el mensaje “El distrito no cuenta con proveedores registrados”.

```
--Validando la existencia del procedimiento almacenado
IF OBJECT_ID (' SP_LISTAPROVEEDORxDISTRITO ') IS NOT
    NULL DROP PROC SP_LISTAPROVEEDORxDISTRITO
GO

--Creando el procedimiento almacenado
CREATE PROC SP_LISTAPROVEEDORxDISTRITO (@DIS CHAR(3))
AS
    IF EXISTS (SELECT * FROM TB_PROVEEDOR WHERE COD_DIS = @DIS)
        BEGIN
            SELECT * FROM TB_PROVEEDOR WHERE COD_DIS = @DIS
        END
    ELSE
        BEGIN
            PRINT 'EL DISTRITO NO CUENTA CON PROVEEDORES'
        END
    GO

--Ejecutando el procedimiento
EXECUTE SP_LISTAPROVEEDORxDISTRITO 'D06'
GO
```

	COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
1	PR10	Miura	Av. La Paz 257	4459710	D06	Jorge Vasquez

Figura 102: Listado de vendedores del distrito D06
Fuente. - Tomado desde SQL Server 2014

4.1.3.2 Estructura condicional múltiple CASE

Evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles. La expresión CASE tiene dos formatos:

CASE simple, el cual compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.

CASE buscada, evalúa un conjunto de expresiones booleanas para determinar el resultado.

El formato de la estructura condicional múltiple CASE es:

```
--SIMPLE
CASE COLUMN
    WHEN 'ALTERNATIVA1' THEN 'VALOR RESULTANTE 1'
    WHEN 'ALTERNATIVA2' THEN 'VALOR RESULTANTE 2'
    ELSE 'VALOR RESULTANTE POR DEFECTO'
END
```

```
--BUSCADA
CASE
    WHEN CONDICION1 THEN 'VALOR RESULTANTE 1'
    WHEN CONDICION2 THEN 'VALOR RESULTANTE 2'
    ELSE 'VALOR RESULTANTE POR DEFECTO'
END
```

Ejemplo:

Veamos el uso de la estructura CASE. Procedimiento Almacenado que permita listar los datos de los clientes además añadir una columna a todos los registros que indique la descripción del tipo de cliente de forma que si el valor es 1 asignar “Corporativo” y si el valor es 2 asigne “Personal”.

```
--Validando la existencia del procedimiento almacenado
```

```
IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO
```

```
--Creando el procedimiento almacenado
```

```
CREATE PROC SP_LISTACLIENTES
AS
    SELECT C.COD_CLI AS CODIGO,
           C.RAZ_SOC_CLI AS CLIENTE,
           C.RUC_CLI AS RUC,
           C.TIP_CLI AS [CODIGO DE TIPO],
           CASE C.TIP_CLI
               WHEN 1 THEN 'CORPORATIVO'
               WHEN 2 THEN 'PERSONAL'
           END AS TIPO
    FROM TB_CLIENTE C
GO
```

```
--Ejecutando el procedimiento
```

```
EXEC SP_LISTACLIENTES
GO
```

	CODIGO	CLIENTE	RUC	CODIGO DE TIPO	TIPO
1	C001	Finseth	48632081	1	CORPORATIVO
2	C002	Orbi	57031642	2	PERSONAL
3	C003	Serviems	NULL	2	PERSONAL
4	C004	Issa	46720159	1	CORPORATIVO
5	C005	Mass	83175942	1	CORPORATIVO
6	C006	Berker	54890124	1	CORPORATIVO
7	C007	Fidenza	16204790	2	PERSONAL
8	C008	Intech	34021824	2	PERSONAL
9	C009	Prominent	NULL	1	CORPORATIVO
10	C010	Landu	30405261	2	PERSONAL
11	C011	Filasur	70345201	1	CORPORATIVO

Figura 103: Listado de clientes y su tipo
Fuente. - Tomado desde SQL Server 2014

4.1.3.3 Estructura repetitiva 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**.

Finalmente, debemos considerar que cada bloque de la estructura repetitiva necesitará más de dos sentencias por lo tanto será necesario usar el bloque **BEGIN END** para marcar un punto de inicio y final del bloque.

El formato de la estructura condicional WHILE es:

```
WHILE CONDICION
BEGIN
    SENTENCIAS | BREAK | CONTINUE
END
GO
```

Ejemplo:

Veamos un caso del uso de la estructura repetitiva WHILE, donde se necesita mostrar los 100 primeros números naturales e imprimir un mensaje de “Número Par” o “Número Impar” dependiendo de su valor.

```
DECLARE @CONTADOR INT
SET @CONTADOR = 0

WHILE (@CONTADOR < 100)
BEGIN
    SET @CONTADOR = @CONTADOR + 1
    IF (@CONTADOR % 2 = 0)
        PRINT CAST(@CONTADOR AS VARCHAR) + ' NUMERO PAR'
    ELSE
        PRINT CAST(@CONTADOR AS VARCHAR) + ' NUMERO IMPAR'
END
GO
```

Messages	
1	NUMERO IMPAR
2	NUMERO PAR
3	NUMERO IMPAR
4	NUMERO PAR
5	NUMERO IMPAR
6	NUMERO PAR
7	NUMERO IMPAR
8	NUMERO PAR
9	NUMERO IMPAR
10	NUMERO PAR
11	NUMERO IMPAR
12	NUMERO PAR
13	NUMERO IMPAR
14	NUMERO PAR
15	NUMERO IMPAR
16	NUMERO PAR

Figura 104: Listado de números pares e impares usando While
Fuente. - Tomado desde SQL Server 2014

4.1.4. Procedimientos almacenados de mantenimiento de datos (INSERT, UPDATE y DELETE)

Usando la tabla TB_PROVEEDOR que se encuentra dentro de la base de datos COMERCIO empezaremos a implementar los procedimientos de mantenimiento de inserción, actualización y eliminación.

TB_PROVEEDOR		
Nombre de columna	Tipo de datos	Permitir val...
COD_PRV	char(4)	<input type="checkbox"/>
RAZ_SOC_PRV	varchar(80)	<input type="checkbox"/>
DIR_PRV	varchar(100)	<input type="checkbox"/>
TEL_PRV	varchar(15)	<input checked="" type="checkbox"/>
COD_DIS	char(3)	<input type="checkbox"/>
REP_VEN	varchar(80)	<input checked="" type="checkbox"/>

Figura 105: Tabla TB_PROVEEDOR de la base de datos COMERCIO
Fuente. - Tomado desde SQL Server 2014

4.1.4.1. INSERT

Ejemplo 01:

Crear un procedimiento almacenado que permita registrar datos a la tabla proveedores. Asignar de nombre **sp_Registrar_Proveedor**. Además, debe validar la existencia del procedimiento almacenado.

```
IF OBJECT_ID ('SP_REGISTRAR_PROVEEDOR') IS NOT NULL
BEGIN
DROP PROCEDURE SP_REGISTRAR_PROVEEDOR
END
GO
CREATE PROCEDURE SP_REGISTRAR_PROVEEDOR
@CODP      CHAR      (4),
@RSOCIAL   VARCHAR   (80),
@DIREC    VARCHAR   (100),
@TELEF    VARCHAR   (15),
@CDDIS    CHAR      (3),
```

```

@REPVEN      VARCHAR      (80)
AS
BEGIN
INSERT INTO TB_PROVEEDOR VALUES (@CODP, @RSOCIAL, @DIREC,
@TELEF, @CODDIS, @REPVEN)
END
GO

```

Ejecutando el procedimiento almacenado

Vamos a comprobar ingresando los datos respectivos a cada parámetro de entrada.

```

EXECUTE SP_REGISTRAR_PROVEEDOR 'PR21', 'GRUPO ABR S.A.C.', 'AV.
GRAN CHIMÚ # 4521', '989898568', 'D26', 'JULIO LOPEZ MORENO'
GO

```

Resultado:

Ahora vamos a mostrar el dato insertado mediante un procedimiento almacenado a la tabla **TB_PROVEEDOR**

```

SELECT * FROM TB_PROVEEDOR
GO

```

COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
PR21	GRUPO ABR S.A.C.	AV. GRAN CHIMÚ # 4521	989898568	D26	JULIO LOPEZ MORENO

Figura 106: Registro insertado mediante un procedimiento almacenados a la tabla TB_PROVEEDOR

Fuente. - Tomado desde SQL Server 2014

4.1.4.2. UPDATE

Ejemplo 02:

Crear un procedimiento almacenado que permita actualizar todos los datos de la tabla proveedores. Asignar de nombre **sp_actualizar_proveedor**. Además, debe validar la existencia del procedimiento almacenado. La actualización se realizará mediante su código.

```

IF OBJECT_ID ('SP_ACTUALIZAR_PROVEEDOR') IS NOT NULL
BEGIN
DROP PROCEDURE SP_ACTUALIZAR_PROVEEDOR
END
GO
CREATE PROCEDURE SP_ACTUALIZAR_PROVEEDOR
@CODP          CHAR      (4),
@RSOCIAL       VARCHAR   (80),
@DIREC         VARCHAR   (100),
@TELEF         VARCHAR   (15),
@CODDIS        CHAR      (3),
@REPVEN        VARCHAR   (80)
AS
BEGIN

```

```

UPDATE TB_PROVEEDOR SET RAZ_SOC_PRV=@RSOCIAL, DIR_PRV=@DIREC,
TEL_PRV=@TELEF, COD_DIS=@CODDIS, REP_VEN=@REPVEN
WHERE COD_PRV=@CODP
END
GO

```

Ejecutando el procedimiento almacenado

Vamos a comprobar modificando su dirección y teléfono del proveedor **PR21**. Los demás datos se mantienen igual.

```

EXECUTE SP_ACTUALIZAR_PROVEEDOR 'PR21', 'GRUPO ABR S.A.C.', 'AV. CAMPOY
3250', '999999999', 'D26', 'JULIO LOPEZ MORENO'
GO

```

Resultado:

Ahora vamos a mostrar los datos de la tabla **TB_PROVEEDOR** con los cambios realizados.

```

SELECT * FROM TB_PROVEEDOR
GO

```

COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
PR21	GRUPO ABR S.A.C.	AV. CAMPOY # 3250	999999999	D26	JULIO LOPEZ MORENO

Figura 107: Registro insertado mediante un procedimiento almacenado a la tabla TB_PROVEEDOR

Fuente. - Tomado desde SQL Server 2014

4.1.4.3. DELETE

Ejemplo 03:

Crear un procedimiento almacenado que permita eliminar un proveedor mediante su código. Asignar de nombre **sp_eliminar_proveedor**. Además, debe validar la existencia del procedimiento almacenado.

```

IF OBJECT_ID ('SP_ELIMINAR_PROVEEDOR') IS NOT NULL
BEGIN
DROP PROCEDURE SP_ELIMINAR_PROVEEDOR
END
GO
CREATE PROCEDURE SP_ELIMINAR_PROVEEDOR
@CODP CHAR (4)
AS
BEGIN
DELETE FROM TB_PROVEEDOR WHERE COD_PRV=@CODP
END
GO

```

Ejecutando el procedimiento almacenado

Vamos a comprobar eliminando al proveedor PR21.

```
EXECUTE SP_ELIMINAR_PROVEEDOR 'PR21'
GO
```

Resultado:

Para comprobar ejecute el siguiente código

```
SELECT * FROM TB_PROVEEDOR
GO
```

Actividad

Usando la base de datos **COMERCIO** implementaremos los siguientes procedimientos almacenados:

Procedimientos almacenados básicos

Procedimiento almacenado que permita mostrar la fecha actual.

```
IF OBJECT_ID('SP_FECHAACTUAL') IS NOT
    NULL DROP PROC SP_FECHAACTUAL
GO

CREATE PROC SP_FECHAACTUAL
AS
    SELECT GETDATE()
GO

--Probando el procedimiento
EXEC SP_FECHAACTUAL
GO
```

(No column name)	
1	2015-07-14 01:01:50.433

Figura 108: Listando la fecha y hora actual
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla Producto donde se visualice el código, descripción, precio, stock actual y stock mínimo y la unidad de medida de los productos. Defina de manera adecuada la cabecera del listado.

```
IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS
AS
    SELECT P.COD_PRO AS CODIGO,
```

```

P.DES_PRO AS DESCRIPCION,
P.PRE_PRO AS PRECIO,
P.STK_ACT AS [STOCK ACTUAL],
P.STK_MIN AS [STOCK MINIMO],
P.UNI_MED AS [UNIDAD DE MEDIDA]
FROM TB_PRODUCTO P
GO

--Probando el procedimiento
EXEC SP_LISTAPRODUCTOS
GO

```

	CODIGO	DESCRIPCION	PRECIO	STOCK ACTUAL	STOCK MINIMO	UNIDAD DE MEDIDA
1	P001	Papel Bond A-4	35.00	200	1500	Mil
2	P002	Papel Bond Oficio	28.00	50	1500	Mil
3	P003	Papel Bulky	10.00	498	1000	Mil
4	P004	Papel Periódico	7.20	4285	1000	Mil
5	P005	Cartucho Tinta Negra	32.00	50	30	Uni
6	P006	Cartucho Tinta Color	36.00	58	35	Uni
7	P007	Porta Diskettes	3.50	300	100	Uni

Figura 109: Listado de productos
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla cliente donde se visualice el código, razón social, dirección, teléfono, ruc y fecha de registro del cliente. Defina de manera adecuada la cabecera del listado.

```

IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO

CREATE PROC SP_LISTACLIENTES
AS
    SELECT C.COD_CLI AS CODIGO,
           C.RAZ_SOC_CLI AS [RAZON SOCIAL],
           C.DIR_CLI AS DIRECCION,
           C.TEL_CLI AS TELEFONO,
           C.RUC_CLI AS RUC,
           C.FEC_REG AS [FECHA DE REGISTRO]
    FROM TB_CLIENTE C
GO

--Probando el procedimiento
EXEC SP_LISTACLIENTES
GO

```

	CODIGO	RAZON SOCIAL	DIRECCION	TELEFONO	RUC	FECHA DE REGISTRO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	1991-12-10
2	C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	1990-02-01
3	C003	Serviemsa	Jr. Collagate 522	75012403	NULL	1995-06-03
4	C004	Issa	Calle Los Aviadores 263	3725910	46720159	1992-09-12
5	C005	Mass	Av. Tomas Marsano 880	4446177	83175942	1992-10-01
6	C006	Berker	Av. Los Proceres 521	3810322	54890124	1989-07-05
7	C007	Fidenza	Jr. El Niquel 282	5289034	16204790	1991-10-02
8	C008	Intech	Av. San Luis 2619 5to P	2249493	34021824	1997-01-07

Figura 110: Listado de clientes
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla Vendedor donde se visualice el código, nombre completo, sueldo y fecha de ingreso del vendedor. Defina de manera adecuada la cabecera del listado.

```
IF OBJECT_ID('SP_LISTAVENDEDORES') IS NOT NULL
    DROP PROC SP_LISTAVENDEDORES
GO

CREATE PROC SP_LISTAVENDEDORES
AS
    SELECT V.COD_VEN AS CODIGO,
           V.NOM_VEN+SPACE(1)+V.APE_VEN AS NOMBRES,
           V.SUE_VEN AS SUELDO,
           V.FEC_ING AS [FECHA DE INGRESO]
      FROM TB_VENDEDOR V
GO

--Probando el procedimiento
EXEC SP_LISTAVENDEDORES
GO
```

	CODIGO	NOMBRES	SUELDO	FECHA DE INGRESO
1	V01	JUANA MESES	1100.00	2015-01-15
2	V02	JUAN SOTO	1300.00	2014-02-05
3	V03	CARLOS AREVALO	1600.00	2013-03-25
4	V04	CESAR OJEDA	1550.00	2014-05-05
5	V05	JULIO VEGA	1600.00	2014-01-10
6	V06	ANA ORTEGA	1300.00	2015-02-20
7	V07	JOSE PALACIOS	1600.00	2013-03-02
8	V08	RUBEN SALAS	1550.00	2014-05-07
9	V09	PATRICIA ARCE	1900.00	2013-06-28
10	V10	RENATO IRIARTE	1650.00	2013-04-16

Figura 111: Listado de vendedores
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla Producto cuya unidad de medida sea “DOC”.

```

IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS
AS
    SELECT P.*
        FROM TB_PRODUCTO P
        WHERE P.UNI_MED='DOC'
GO

--Probando el procedimiento
EXEC SP_LISTAPRODUCTOS
GO

```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
2	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
3	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
4	P012	Tajador Plastico	9.60	608	300	Doc	3	FALSO
5	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
6	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO
7	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO

Figura 112: Listado de productos de unidad de medida “Doc”

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla Cliente cuyo código de distrito sea “D05”.

```

IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO

CREATE PROC SP_LISTACLIENTES
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.COD_DIS='D05'
GO

--Probando el procedimiento
EXEC SP_LISTACLIENTES
GO

```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto
2	C003	Serviems	Jr. Colligate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
3	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste
4	C010	Landu	Av.Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza
5	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega

Figura 113: Listado de clientes del distrito con código D05

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita mostrar los registros de la tabla Orden de Compra cuyo año de registro sea de 1999.

```
IF OBJECT_ID('SP_LISTAORDENES') IS NOT NULL
    DROP PROC SP_LISTAORDENES
GO

CREATE PROC SP_LISTAORDENES
AS
    SELECT O.*
        FROM TB_ORDEN_COMPRA O
        WHERE YEAR(O.FEC_OCO)=1999
GO

--Probando el procedimiento
EXEC SP_LISTAORDENES
GO
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC017	1999-08-01	PR09	1999-08-01	1
2	OC018	1999-01-02	PR20	1999-08-02	1
3	OC019	1999-03-03	PR11	1999-06-03	1
4	OC020	1999-07-10	PR12	1999-08-13	1
5	OC021	1999-08-30	PR14	1999-09-13	1
6	OC022	1999-06-14	PR05	1999-08-14	2

Figura 114: Listado de órdenes de compra registradas en el año 1999

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste el detalle de las facturas cuyo código sea “103” y además se refiera al producto cuyo código sea “P002”.

```
IF OBJECT_ID('SP_DETALLEFAC') IS NOT NULL
    DROP PROC SP_DETALLEFAC
GO

CREATE PROC SP_DETALLEFAC
AS
    SELECT DF.*
        FROM TB_DETALLE_FACTURA DF
        WHERE DF.NUM_FAC=103 AND COD_PRO='P002'
GO

--Probando el procedimiento
EXEC SP_DETALLEFAC
GO
```

	NUM_FAC	COD_PRO	CAN_VEN	PRE_VEN
1	103	P002	10	40.00

Figura 115: Listando el detalle de la factura 103 y código de producto P002

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste las facturas registradas en el año 1998 pertenecientes al segundo semestre; es decir, de Julio a Diciembre.

```
IF OBJECT_ID('SP_LISTAFACUTURAS') IS NOT NULL
    DROP PROC SP_LISTAFACUTURAS
GO

CREATE PROC SP_LISTAFACUTURAS
AS
    SELECT F.*
        FROM TB_FACTURA F
        WHERE YEAR(F.FEC_FAC) =1998 AND
              MONTH(F.FEC_FAC) BETWEEN 07 AND 12
GO

--Probando el procedimiento
EXEC SP_LISTAFACUTURAS
GO
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	105	1998-10-10	C009	1998-05-08	3	V05	0.18
2	107	1998-09-10	C012	1998-06-11	2	V10	0.18
3	109	1998-10-01	C017	1998-06-11	2	V02	0.18
4	110	1998-10-11	C019	1998-01-12	2	V05	0.18
5	114	1998-08-12	C015	1999-06-01	2	V07	0.18

Figura 116: Listado de facturas registradas en el primer semestre del año 1998

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los productos cuyo precio se encuentren entre S/. 5.00 y S/. 10.00 y además sean de origen nacional, es decir, no productos importados.

```
IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS
AS
    SELECT P./*
        FROM TB_PRODUCTO P
        WHERE P.PRE_PRO BETWEEN 5 AND 10 AND
              P.IMPORTADO = 'FALSO'
GO

--Probando el procedimiento
EXEC SP_LISTAPRODUCTOS
GO
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P004	Papel Periódico	7.20	4285	1000	Mil	2	FALSO
2	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
3	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
4	P012	Tajador Plástico	9.60	608	300	Doc	3	FALSO
5	P017	Lapicero Negro	8.00	3000	1000	Doc	3	FALSO
6	P018	Lapicero Azul	8.00	2010	1500	Doc	3	FALSO

Figura 117: Listado de productos con precio entre S/. 5 y S/. 10 y de importación falsa (nacional)

Fuente. - Tomado desde SQL Server 2014

Procedimientos Almacenados condicionados

Procedimiento Almacenado que liste todos los productos según sea importado o no.

```
--Validando la existencia del procedimiento
IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO
--Creando el procedimiento almacenado
CREATE PROC SP_LISTAPRODUCTOS (@IMP VARCHAR(30)) AS
    SELECT P.*
        FROM TB_PRODUCTO P
        WHERE P.IMPORTADO=@IMP
GO
--Probando el procedimiento
EXEC SP_LISTAPRODUCTOS
'FALSO' GO
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P002	Papel Bond Oficio	28.00	50	1500	Mil	2	FALSO
2	P004	Papel Periódico	7.20	4285	1000	Mil	2	FALSO
3	P005	Cartucho Tinta Negra	32.00	50	30	Uni	1	FALSO
4	P006	Cartucho Tinta Color	36.00	58	35	Uni	1	FALSO
5	P008	Caja de Diskettes * 10	24.00	125	180	Uni	1	FALSO
6	P009	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
7	P010	Borrador Blanco	6.40	2000	400	Doc	3	FALSO
8	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
9	P012	Tajador Plástico	9.60	608	300	Doc	3	FALSO
10	P013	Folder Manila Oficio	16.00	200	150	Cie	3	FALSO
11	P015	Sobre Manila Oficio	12.00	300	130	Cie	3	FALSO
12	P016	Sobre Manila A-4	14.40	200	100	Cie	3	FALSO

Figura 118: Listado de productos de tipo Falso en la columna importado (producto nacional)

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste las facturas registradas en un determinado año (FEC_FAC).

```

IF OBJECT_ID('SP_LISTAFACTURAS') IS NOT NULL
    DROP PROC SP_LISTAFACTURAS
GO

CREATE PROC SP_LISTAFACTURAS (@AÑO INT)
AS
    SELECT F.*
        FROM TB_FACTURA F
        WHERE YEAR(F.FEC_FAC) = @AÑO
GO

--Probando el procedimiento
EXEC SP_LISTAFACTURAS 1998
GO

```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IVG
1	100	1998-06-07	C001	1998-05-08	2	V01	0.18
2	101	1998-06-09	C019	1998-05-08	3	V02	0.18
3	102	1998-01-09	C003	1998-03-11	2	V04	0.18
4	103	1998-06-09	C016	1998-05-11	2	V07	0.18
5	104	1998-01-10	C015	1998-12-10	2	V08	0.18

Figura 119: Listado de facturas registradas en el año 1998

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de los clientes según el código del distrito.

```

IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO

CREATE PROC SP_LISTACLIENTES(@DIS CHAR(3))
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.COD_DIS=@DIS
GO

--Probando el procedimiento
EXEC SP_LISTACLIENTES 'D16'
GO

```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C015	Meba	Av. Elmer Faucett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez

Figura 120: Listado de clientes del distrito cuyo código es D16

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de los clientes según la letra inicial en su razón social.

```
IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO

CREATE PROC SP_LISTACLIENTES (@INICIAL CHAR(1))
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.RAZ_SOC_CLI LIKE @INICIAL + '%'
GO

--Probando el procedimiento
EXEC SP_LISTACLIENTES 'M'
GO
```

Procedimiento Almacenado que liste todos los datos de los vendedores cuyo sueldo sea inferior a un determinado monto.

```
IF OBJECT_ID('SP_LISTAVENDEDOR') IS NOT NULL
    DROP PROC SP_LISTAVENDEDOR
GO

CREATE PROC SP_LISTAVENDEDOR (@MONTO MONEY)
AS
    SELECT V.*
        FROM TB_VENDEDOR V
        WHERE V.SUE_VEN < @MONTO
GO

--Probando el procedimiento
EXEC SP_LISTAVENDEDOR 1250
GO
```

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V01	JUANA	MESES	1100.00	2015-01-15	1	D08

Figura 121: Listado de vendedores con sueldo inferior a S/. 1250

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de las órdenes de compra según el código del proveedor.

```

IF OBJECT_ID('SP_LISTAORDENES') IS NOT NULL
    DROP PROC SP_LISTAORDENES
GO

CREATE PROC SP_LISTAORDENES (@PRO CHAR(4))
AS
    SELECT O.*
        FROM TB_ORDEN_COMPRA O
        WHERE O.COD_PRV=@PRO
GO

--Probando el procedimiento
EXEC SP_LISTAORDENES 'PR01'
GO

```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC004	1998-05-04	PR01	1998-05-04	2
2	OC010	1998-05-09	PR01	1998-05-09	2

Figura 122: Listado de órdenes de compra del proveedor PR01
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de los productos entre un rango de montos que representan el precio del producto.

```

IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS (@R1 MONEY, @R2 MONEY)
AS
    SELECT P.*
        FROM TB_PRODUCTO P
        WHERE P.PRE_PRO BETWEEN @R1 AND @R2
GO

--Probando el procedimiento
EXEC SP_LISTAPRODUCTOS 15,20
GO

```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P011	Tajador Metal	16.00	1120	300	Doc	3	FALSO
2	P013	Folder Manila Oficio	16.00	200	150	Cie	3	FALSO
3	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO

Figura 123: Listado de productos cuyo rango de precio es 15 y 20
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de los clientes según el código del distrito y tipo de cliente.

```

IF OBJECT_ID('SP_LISTACLIENTES') IS NOT NULL
    DROP PROC SP_LISTACLIENTES
GO

CREATE PROC SP_LISTACLIENTES (@DIS CHAR(3), @TIP INT)
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.COD_DIS=@DIS AND C.TIP_CLI=@TIP
GO

--Probando el procedimiento
EXEC SP_LISTACLIENTES 'D14', 1
GO

```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C005	Mass	Av. Tomas Marsano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo

Figura 124: Listando los clientes del distrito con código D14 y tipo de cliente 1

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste todos los datos de las órdenes de compra según el año de registro (FEC_OCO) y un determinado estado (EST_OCO).

```

IF OBJECT_ID('SP_LISTAORDENES') IS NOT NULL
    DROP PROC SP_LISTAORDENES
GO

CREATE PROC SP_LISTAORDENES (@AÑO INT, @EST INT)
AS
    SELECT O.*
        FROM TB_ORDEN_COMPRA O
        WHERE YEAR(O.FEC_OCO)=@AÑO AND EST_OCO=@EST
GO

--Probando el procedimiento
EXEC SP_LISTAORDENES 1999, 1
GO

```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC017	1999-08-01	PR09	1999-08-01	1
2	OC018	1999-01-02	PR20	1999-08-02	1
3	OC019	1999-03-03	PR11	1999-06-03	1
4	OC020	1999-07-10	PR12	1999-08-13	1
5	OC021	1999-08-30	PR14	1999-09-13	1

Figura 125: Listado de órdenes de compra del año 1999 y estado 1

Fuente. - Tomado desde SQL Server 2014

Integrando los procedimientos almacenados con la estructura IF

Procedimiento Almacenado que liste todas las facturas registradas en un determinado año, en caso dicho año no presente registros mostrar el mensaje “No registra facturas en el año ...”

```

IF OBJECT_ID('SP_LISTAFACTURAS') IS NOT NULL
    DROP PROC SP_LISTAFACTURAS
GO

CREATE PROC SP_LISTAFACTURAS (@AÑO INT)
AS
    IF EXISTS (SELECT * FROM TB_FACTURA WHERE
        YEAR(FEC_FAC) = @AÑO) SELECT * FROM TB_FACTURA WHERE
        YEAR(FEC_FAC) = @AÑO
    ELSE
        PRINT 'NO REGISTRA FACTURAS EN EL AÑO '+CAST(@AÑO AS
CHAR(4))
GO

--PRUEBA
EXEC SP_LISTAFACTURAS 2015
GO

```

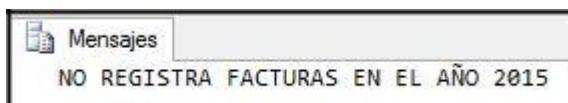


Figura 126: Mensaje desde la validación del año
Fuente. - Tomado desde SQL Server 2014

```
EXEC SP_LISTAFACTURAS 1998
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	100	1998-06-07	C001	1998-05-08	2	V01	0.19
2	101	1998-06-09	C019	1998-05-08	3	V02	0.19
3	102	1998-01-09	C003	1998-03-11	2	V04	0.19
4	103	1998-06-09	C016	1998-05-11	2	V07	0.19
5	104	1998-01-10	C015	1998-12-10	2	V08	0.19

Figura 127: Listado de facturas registradas en el año 1998
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste las órdenes de compra registradas por un determinado proveedor (COD_PRV).

```

IF OBJECT_ID('SP_LISTAORDENES') IS NOT NULL
    DROP PROC SP_LISTAORDENES
GO

CREATE PROC SP_LISTAORDENES (@PRO CHAR(4))
AS
    IF EXISTS (SELECT * FROM TB_ORDEN_COMPRA WHERE COD_PRV=@PRO)
        SELECT OC.*
        FROM TB_ORDEN_COMPRA OC
        WHERE OC.COD_PRV=@PRO
    ELSE
        PRINT 'PROVEEDOR NO REGISTRA ORDENES DE COMPRA'
GO

```

--PRUEBA:

```
EXEC SP_LISTAORDENES 'PR07'
GO
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC005	1998-06-03	PR07	1998-10-03	1

Figura 128: Listado de registro de orden de compra del proveedor PR07
Fuente. - Tomado desde SQL Server 2014



Figura 129: Mensaje desde la validación del proveedor
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste los productos entre un rango de valores pertenecientes al precio del producto; de tal forma que el primer valor debe ser inferior al segundo, caso contrario, mostrar el mensaje “EL RANGO DE VALORES ES INCORRECTO”.

```
IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS (@MONTO1 MONEY, @MONTO2 MONEY)
AS
    IF (@MONTO1 < @MONTO2)
        SELECT P.*
            FROM TB_PRODUCTO P
            WHERE P.PRE_PRO BETWEEN @MONTO1 AND @MONTO2
    ELSE
        PRINT 'EL RANGO DE VALORES ES INCORRECTO'
GO

--PRUEBA:
EXEC SP_LISTAPRODUCTOS 30,20
GO
```

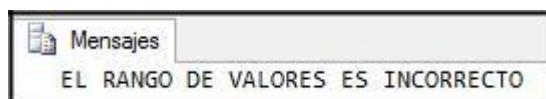


Figura 130: Mensaje de validación obtenida desde el rango de valores incorrectos
Fuente. - Tomado desde SQL Server 2014

```
EXEC SP_LISTAPRODUCTOS 20,30
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P008	Caja de Diskettes * 10	30.00	125	180	Uni	1	FALSO
2	P011	Tajador Metal	20.00	1120	300	Doc	3	FALSO
3	P013	Folder Manila Oficio	20.00	200	150	Cie	3	FALSO
4	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO

Figura 131: Listado de productos con precios entre S/. 20 y S/. 30

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita registrar los datos de un producto para lo cual se debe validar la existencia única del código del producto, en caso registre duplicidad de código mostrar el mensaje “CÓDIGO DE PRODUCTO YA SE ENCUENTRA REGISTRADO”.

```

IF OBJECT_ID('SP_REGISTRAPRODUCTO') IS NOT NULL
    DROP PROC SP_REGISTRAPRODUCTO
GO

CREATE PROC SP_REGISTRAPRODUCTO
    (@COD CHAR(4),
     @DES VARCHAR(50),
     @PRE MONEY,
     @SAC INT,
     @SMI INT,
     @UNI VARCHAR(30),
     @LIN INT,
     @IMP VARCHAR(10))
AS
IF EXISTS (SELECT * FROM TB_PRODUCTO WHERE COD_PRO=@COD)
PRINT 'CÓDIGO DE PRODUCTO YA SE ENCUENTRA REGISTRADO'
ELSE
INSERT INTO TB_PRODUCTO VALUES (@COD, @DES, @PRE, @SAC, @SMI, @UNI,
@LIN, @IMP)
GO

--PRUEBA
EXEC SP_REGISTRAPRODUCTO 'P021', 'Cartulina Duplex',
                           0.5, 1000, 100, 'Doc', 2, 'FALSO'
GO

```

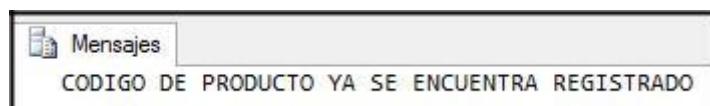


Figura 132: Mensaje desde la validación del producto

Fuente. - Tomado desde SQL Server 2014

```

EXEC SP_REGISTRAPRODUCTO 'P022', 'Cartulina Duplex',
                           0.5, 1000, 100, 'Doc', 2, 'FALSO'
GO

```



Figura 133: Mensaje de inserción correcta del producto
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita registrar los datos de un vendedor para lo cual se debe validar la existencia del código del vendedor, así como el código del distrito. En cualquiera de los casos mostrar el mensaje “ERROR AL INTENTAR REGISTRAR AL VENDEDOR”.

```

IF OBJECT_ID('SP_REGISTRAVENDEDOR') IS NOT NULL
    DROP PROC SP_REGISTRAVENDEDOR
GO
CREATE PROC SP_REGISTRAVENDEDOR
    (@COD CHAR(3),
     @NOM VARCHAR(20),
     @APE VARCHAR(20),
     @SUE MONEY,
     @FEC DATE,
     @TIP INT,
     @DIS CHAR(3))
AS
IF EXISTS (SELECT * FROM TB_VENDEDOR WHERE COD_VEN=@COD) OR
    NOT EXISTS (SELECT * FROM TB_DISTRITO WHERE COD_DIS=@DIS)
        PRINT 'ERROR AL INTENTAR REGISTRAR AL VENDEDOR'
ELSE
    INSERT INTO TB_VENDEDOR VALUES (@COD, @NOM, @APE, @SUE, @FEC,
                                     @TIP, @DIS)
    GO
--PRUEBA
EXEC SP_REGISTRAVENDEDOR 'V11', 'MARIA', 'ZAMORA',
                           1950, '2015/02/05', 1, 'D07'
GO

```



Figura 134: Mensaje de inserción correcta de vendedor
Fuente. - Tomado desde SQL Server 2014

```

EXEC SP_REGISTRAVENDEDOR 'V01', 'MARIA', 'ZAMORA',
                           1950, '2015/02/05', 1, 'D07'

```

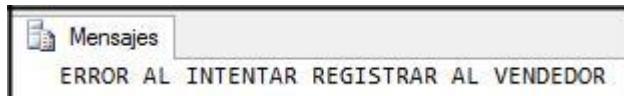


Figura 135: Mensaje desde la validación del vendedor no valido
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que permita registrar una factura para lo cual debe validar que el número de factura sea única, caso contrario, mostrará el mensaje “NÚMERO DE FACTURA YA SE ENCUENTRA REGISTRADA”. Además de validar la existencia del código del vendedor, caso contrario, se mostrará el mensaje “CODIGO DE VENDEDOR NO VALIDO”. Finalmente validar la existencia del código del cliente, en caso no exista mostrar el mensaje “CODIGO DE CLIENTE NO VALIDO”.

```

IF OBJECT_ID('SP_REGISTRAFACTURA') IS NOT NULL
    DROP PROC SP_REGISTRAFACTURA
GO
CREATE PROC SP_REGISTRAFACTURA
    (@FFA DATE,
    @CLI CHAR (4),
    @FCA DATE,
    @EST INT,
    @VEN CHAR (3))
AS
IF NOT EXISTS (SELECT * FROM TB_VENDEDOR WHERE COD_VEN=@VEN)
    PRINT 'CODIGO DE VENDEDOR NO VALIDO'
ELSE IF NOT EXISTS (SELECT * FROM TB_CLIENTE WHERE COD_CLI=@CLI)
    PRINT 'CODIGO DE CLIENTE NO VALIDO'
ELSE
    BEGIN
        INSERT INTO TB_FACTURA (FEC_FAC, COD_CLI, FEC_CAN, EST_FAC, COD_VEN)
        VALUES (@FFA, @CLI, @FCA, @EST, @VEN)
        PRINT 'REGISTRO DE FACTURA CORRECTA'
    END
GO

--PRUEBA
EXEC SP_REGISTRAFACTURA '2016/06/05', 'C003', '2016/07/03', 1, 'V01'
GO

```

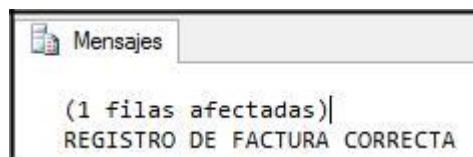


Figura 136: Mensaje de registro correcto
Fuente. - Tomado desde SQL Server 2014

```
EXEC SP_REGISTRAFACTURA '2016/06/05', 'C003', '2016/07/03', 1, 'V31'
```



Figura 137: Mensaje desde la validación de vendedor no valido
Fuente. - Tomado desde SQL Server 2014

```
EXEC SP_REGISTRAFACTURA '2016/06/05', 'C033', '2016/07/03', 1, 'V01'
```

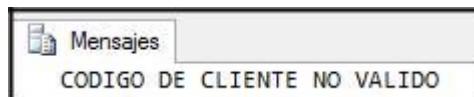


Figura 138: Mensaje desde la validación de cliente no valido

Fuente. - Tomado desde SQL Server 2014

Integrando los procedimientos almacenados con la sentencia CASE

Procedimiento Almacenado que liste todos los productos adicionando la columna “ESTADO DE STOCK” que muestre el mensaje “STOKEAR” solo si el stock actual del producto es menor o igual a 20; caso contrario mostrará el mensaje “EN OBSERVACION”.

```
IF OBJECT_ID('SP_LISTAPRODUCTOS') IS NOT NULL
    DROP PROC SP_LISTAPRODUCTOS
GO

CREATE PROC SP_LISTAPRODUCTOS
AS
    SELECT P.*,
    CASE
        WHEN P.STK_ACT<=100 THEN 'STOKEAR'
        ELSE 'EN OBSERVACION'
    END AS [ESTADO DE STOCK]
    FROM TB_PRODUCTO P
GO

--PRUEBA
EXEC SP_LISTAPRODUCTOS
GO
```

COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO	ESTADO DE STOCK
1 P001	Papel Bond A-4	35.00	200	1500	Mil	2	VERDADERO	EN OBSERVACION
2 P002	Papel Bond Oficio	35.00	50	1500	Mil	2	FALSO	STOKEAR
3 P003	Papel Bulky	10.00	498	1000	Mil	2	VERDADERO	EN OBSERVACION
4 P004	Papel Periódico	9.00	4285	1000	Mil	2	FALSO	EN OBSERVACION
5 P005	Cartucho Tinta Negra	40.00	50	30	Uni	1	FALSO	STOKEAR
6 P006	Cartucho Tinta Color	45.00	58	35	Uni	1	FALSO	STOKEAR
7 P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO	EN OBSERVACION

Figura 139: Listando los productos mostrando su estado según el stock

Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste toda la información de los vendedores adicionando la columna CATEGORIA en la cual se muestre las categorías según la siguiente tabla:

RANGO DEL SUELDO	MENSAJE DE LA COLUMNA CATEGORIA
[1751]	A
[651 ; 1750]	B
[0 ; 650]	C

```

IF OBJECT_ID('SP_LISTAVENDEDORES') IS NOT NULL
    DROP PROC SP_LISTAVENDEDORES
GO

CREATE PROC SP_LISTAVENDEDORES
AS
    SELECT V.*,
        CASE
            WHEN V.SUE_VEN<=650 THEN 'C'
            WHEN V.SUE_VEN>=651 AND V.SUE_VEN<=1750 THEN
                'B' WHEN V.SUE_VEN>=1751 THEN 'A'
        END AS [CATEGORIA]
    FROM TB_VENDEDOR V
GO

--PRUEBA
EXEC SP_LISTAVENDEDORES
GO

```

COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS	CATEGORIA
7	V07	JOSE	PALACI...	1500,00	2013-03-02	1	D02
8	V08	RUBEN	SALAS	1450,00	2014-05-07	2	D04
9	V09	PATRICIA	ARCE	1800,00	2013-06-28	2	D04
10	V10	RENATO	IRIARTE	1550,00	2013-04-16	2	D01
11	V11	MARIA	ZAMORA	1950,00	2015-02-05	1	D07
12	V12	MARIA	ZAMORA	1950,00	2015-02-05	1	D07

Figura 140: Listado de vendedores mostrando su categoría
Fuente. - Tomado desde SQL Server 2014

Procedimiento Almacenado que liste toda la información de los vendedores adicionando la columna CORREO CORPORATIVO en la cual se muestre la generación de su email en base a la primera de su nombre, cuatro letras de su apellido y la asignación de la siguiente letra:

FECHA DE INGRESO	LETRA FINAL
Entre Enero y Abril	A
Entre Mayo y Agosto	B
Entre Setiembre a Diciembre	C

```

IF OBJECT_ID('SP_LISTAVENDEDORES') IS NOT NULL
    DROP PROC SP_LISTAVENDEDORES
GO

CREATE PROC SP_LISTAVENDEDORES
AS
    SELECT V.* , LOWER (LEFT(V.NOM_VEN,1)+LEFT(V.APE_VEN,4)+
        CASE MONTH(V.FEC_ING)
            WHEN 1 THEN 'A'
            WHEN 2 THEN 'A'
            WHEN 3 THEN 'A'
        END) AS CORREO_Corporativo
    FROM TB_VENDEDOR V
GO

```

```

        WHEN 4 THEN 'A'
        WHEN 5 THEN 'B'
        WHEN 6 THEN 'B'
        WHEN 7 THEN 'B'
        WHEN 8 THEN 'B'
        WHEN 9 THEN 'C'
        WHEN 10 THEN 'C'
        WHEN 11 THEN 'C'
        WHEN 12 THEN 'C'
    END+ '@CIBERTEC.EDU.PE') AS [CORREO CORPORATIVO]
FROM TB_VENDEDOR V
GO

--PRUEBA
EXEC SP_LISTAVENDEDORES
GO

```

COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS	CORREO CORPORATIVO
1 V01	JUANA	MESES	1000.00	2015-01-15	1	D08	jmesea@cibertec.edu.pe
2 V02	JUAN	SOTO	1200.00	2014-02-05	2	D03	jsotaa@cibertec.edu.pe
3 V03	CARLOS	AREVALO	1500.00	2013-03-25	2	D09	careva@cibertec.edu.pe
4 V04	CESAR	OJEDA	1450.00	2014-05-05	1	D01	cojedb@cibertec.edu.pe
5 V05	JULIO	VEGA	1500.00	2014-01-10	1	D01	jvegaa@cibertec.edu.pe
6 V06	ANA	ORTEGA	1200.00	2015-02-20	1	D05	aortea@cibertec.edu.pe
7 V07	JOSE	PALACIOS	1500.00	2013-03-02	1	D02	jpalaa@cibertec.edu.pe
8 V08	RUBEN	SALAS	1450.00	2014-05-07	2	D04	rsalab@cibertec.edu.pe
9 V09	PATRICIA	ARICE	1800.00	2013-06-28	2	D04	parceb@cibertec.edu.pe
10 V10	RENATO	IRIARTE	1550.00	2013-04-16	2	D01	rriias@cibertec.edu.pe

Figura 141: Listado de vendedores con sus correos autogenerados

Fuente. - Tomado desde SQL Server 2014

Manejo de la sentencia While

Script que permite mostrar los N últimos registros de la tabla FACTURA condicionando; que las fechas de registro de la factura sean las más actuales posibles.

```

--1. Declarando las variables
DECLARE @N INT=1, @TOPE INT=3

--2. Implementando la sentencia repetitiva
WHILE @N<=100
BEGIN
--3. Evaluando el valor solicitado
    IF @N=@TOPE
    BEGIN
        SELECT TOP(@N) * FROM TB_FACTURA ORDER BY FEC_FAC
        DESC BREAK
    END
--4. Aumentando en valor de búsqueda
    ELSE
    BEGIN
        SET @N+=1
        CONTINUE
    END
END

```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	118	1999-07-02	C008	1999-01-03	3	V03	0.19
2	119	1999-06-02	C013	1999-10-03	2	V02	0.19
3	115	1999-06-01	C016	1999-09-01	2	V05	0.19

Figura 142: Listando las N últimas facturas controladas por variables locales

Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar los registros de la tabla DISTRITO de un rango específico, dichos registros serán mostrados según el orden especificado en el código del distrito (COD_DIS).

```
--1. Declarando las variables
DECLARE @VI INT=3, @VF INT=5

--2. Listando los registros del rango solicitado
SELECT *
    FROM (SELECT ROW_NUMBER ()
          OVER (ORDER BY D.COD_DIS) AS [NUMERO],*
        FROM TB_DISTRITO D) X
   WHERE NUMERO BETWEEN @VI AND @VF
GO
```

	NUMERO	COD_DIS	NOM_DIS
1	3	D03	San Isidro
2	4	D04	La Molina
3	5	D05	San Miguel

Figura 143: Listado de distritos entre un determinado rango de registros

Fuente. - Tomado desde SQL Server 2014

Resumen

En esta sección se han explicado las extensiones de programación que van más allá de las implementaciones típicas de SQL y que hacen de Transact SQL un lenguaje de programación especializado.

Las instrucciones de Transact/SQL pueden agruparse en procesos o lotes, permanecer en la base de datos, ejecutarse repetidamente en forma de procedimientos almacenados.

Los procedimientos almacenados de Transact/SQL pueden ser bastante complejos y pueden llegar a ser una parte importante del código fuente de su aplicación.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

<http://technet.microsoft.com/es-es/library/ms187926.aspx>

Tutorial para la creación de procedimientos almacenados

<http://www.devjoker.com/contenidos/Tutorial-de-Transact-SQL/238/Procedimientos-almacenados-en-Transact-SQL.aspx>

Tutorial para la creación de procedimientos almacenados

[https://msdn.microsoft.com/es-es/library/ms174290\(v=sql.120\).aspx](https://msdn.microsoft.com/es-es/library/ms174290(v=sql.120).aspx)

Lenguaje de control de flujo (Transact-SQL)

<https://msdn.microsoft.com/es-es/library/ms345415.aspx>

Página web sobre la creación de procedimientos almacenados

<http://www.ingenieriasystems.com/2014/01/Manual-de-Microsoft-SQL-Server-Full-Transact-SQL.html>

Manual de Microsoft SQL Server - Full Transact SQL

<https://es.scribd.com/doc/103132742/Ejercicios-Resueltos-de-Sql-Server>

Ejercicios Resueltos de SQL Server



CONSULTAS MULTITABLAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la quinta unidad, el estudiante crea procedimientos almacenados para el mantenimiento de datos; además recupera información de dos o más tablas empleando combinaciones internas y procedimientos almacenados.

TEMARIO

5.1 Tema 10 : Uniones Internas (INNER JOIN)

- 5.1.1 : Combinaciones internas con Inner Join
- 5.1.2 : Procedimientos almacenados con 2 o más tablas
- 5.1.3 : Procedimientos almacenados con 2 o más parámetros
- 5.1.4 : Ejercicios de aplicación

ACTIVIDADES PROPUESTAS

Los estudiantes implementan consultas usando dos o más tablas e integradas a los procedimientos almacenados.

Los estudiantes implementan consultas que permiten hacer uso de las uniones internas integradas a un procedimiento almacenado con parámetros.

5.1. Uniones internas (INNER JOIN)

Las condiciones de combinación se pueden especificar en las cláusulas FROM o WHERE, aunque se recomienda que se especifiquen en la cláusula FROM. Las cláusulas WHERE y HAVING pueden contener también condiciones de búsqueda para filtrar aún más las filas seleccionadas por las condiciones de combinación.

Las combinaciones se pueden clasificar en:

COMBINACIONES INTERNAS: Es aquella en la que los valores de las columnas de una tabla que se están combinando a otra se comparan mediante el operador de igualdad. Es decir, devuelve todas las columnas de ambas tablas y sólo devuelve las filas en las que haya un valor igual en la columna de la combinación.

En el estándar ISO, las combinaciones internas se pueden especificar en la cláusula FROM o en la cláusula WHERE. Este es el único tipo de combinación que ISO admite en la cláusula WHERE. Las combinaciones internas especificadas en la cláusula WHERE se conocen como combinaciones internas al estilo antiguo.

Finalmente, a este tipo de combinación también se le conoce como una combinación equivalente.

COMBINACIONES EXTERNAS. Puede ser una combinación externa izquierda, derecha o completa. Las combinaciones externas se especifican en la cláusula FROM con uno de los siguientes conjuntos de palabras clave:

LEFT JOIN o LEFT OUTER JOIN

El conjunto de resultados de una combinación externa izquierda incluye todas las filas de la tabla de la izquierda especificada en la cláusula LEFT OUTER, y no solo aquellas en las que coincidan las columnas combinadas. Cuando una fila de la tabla de la izquierda no tiene filas coincidentes en la tabla de la derecha, la fila asociada del conjunto de resultados contiene valores NULL en todas las columnas de la lista de selección que procedan de la tabla de la derecha.

RIGHT JOIN o RIGHT OUTER JOIN

Una combinación externa derecha es el inverso de una combinación externa izquierda. Se devuelven todas las filas de la tabla de la derecha. Cada vez que una fila de la tabla de la derecha no tenga correspondencia en la tabla de la izquierda, se devuelven valores NULL para la tabla de la izquierda.

FULL JOIN o FULL OUTER JOIN

Una combinación externa completa devuelve todas las filas de las tablas de la izquierda y la derecha. Cada vez que una fila no tenga coincidencia en la otra tabla, las columnas de la lista de selección de la otra tabla contendrán valores NULL. Cuando haya una coincidencia entre las tablas, la fila completa del conjunto de resultados contendrá los valores de datos de las tablas base.

CROSS JOIN

Las combinaciones cruzadas devuelven todas las filas de la tabla izquierda y, cada fila de la tabla izquierda se combina con todas las filas de la tabla de la derecha.

Las combinaciones cruzadas se llaman también productos cartesianos.

Las tablas o vistas de la cláusula FROM se pueden especificar en cualquier orden en las combinaciones internas o externas completas; sin embargo, el orden especificado de las tablas o vistas sí es importante si utiliza una combinación externa izquierda o derecha.

5.1.1. Combinaciones internas con Inner Join

Para entender el tema de combinaciones internas, haremos uso de una consulta que permita combinar las columnas de las tablas Cliente y Distrito respectivamente.

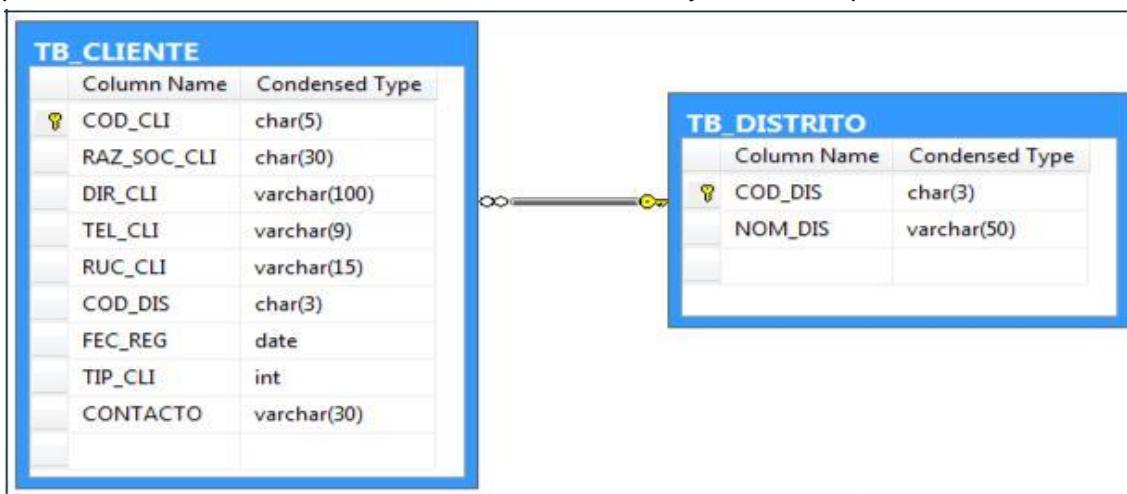


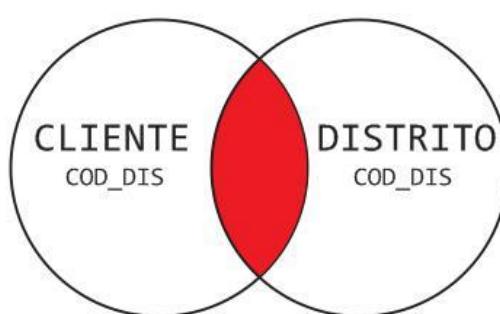
Figura 144: Diagrama de la tabla cliente y distrito
Fuente. - Tomado desde SQL Server 2014

Debemos considerar que una combinación solo será efectiva si cumplimos con las siguientes especificaciones:

Ambas tablas deben tener una columna en común y es esta justamente la que permitirá unir a las tablas.

Dichas columnas deben tener el mismo tipo de datos y la misma capacidad.

No necesariamente las columnas de unión deben llamarse igual en ambas tablas solo debe cumplir “mismo tipo mismo capacidad”.



Si todo es correcto en la especificación de la combinación entre las tablas Cliente y Distrito, podremos tener el control de todas las columnas de Cliente y todas las columnas de Distrito; es a partir de aquí, que podemos administrarlo de la mejor manera, **por ejemplo, mostrar los datos del cliente además del nombre del distrito**. Veamos una primera implementación de combinación entre las tablas Cliente y Distrito en la cual muestre las columnas de ambas tablas en un mismo resultado:

```
SELECT *
  FROM TB_CLIENTE C
INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
GO
```

COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO	COD_DIS	NOM_DIS	
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D05	San Miguel
2	C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	D04	1990-02-01	2	Alfonso Beltran	D04	La Molina
3	C003	Serviemsa	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna	D05	San Miguel
4	C004	Issa	Calle Los Aviadores 263	3725910	46720159	D01	1992-09-12	1	Luis Apumayta	D01	Surco
5	C005	Mass	Av. Tomas Marsano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo	D14	Surquillo
6	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste	D05	San Miguel
7	C007	Fidenza	Jr. El Niquel 282	5289034	16204790	D20	1991-10-02	2	Hector Vivanco	D20	Los Olivos

Figura 145: Listado de clientes con Inner Join

Fuente. - Tomado desde SQL Server 2014

Como vemos, la tabla Cliente muestra sus columnas código, razón social, dirección, teléfono, ruc, código de distrito, fecha de registro, tipo de cliente y nombre del contacto; mientras que, la tabla Distrito muestra el código del distrito y el nombre del mismo.

Solo debemos considerar que existe la posibilidad de que en muchos distritos no hallan clientes registrados; eso quiere decir que se mostrarán todos los registros de los clientes con sus respectivos registros y que los distritos que no registraron NO se mostrarán en el resultado de la consulta.

Así mismo, podemos mencionar que el siguiente script tiene el mismo resultado:

```
SELECT *
FROM TB_CLIENTE C, TB_DISTRITO D
WHERE C.COD_DIS=D.COD_DIS
GO
```

Con el objeto de demostrar el manejo de todas las columnas combinadas mostraremos un listado con el siguiente formato:

Código	Razón Social	Dirección	Teléfono	Distrito
XXXX	XXXXXXXXXX	XXXXXXXXXX	99999999	XXXXXXXXXX
XXXX	XXXXXXXXXX	XXXXXXXXXX	99999999	XXXXXXXXXX

```
SELECT C.COD_CLI AS CODIGO,
C.RAZ_SOC_CLI AS [RAZON SOCIAL],
C.DIR_CLI AS DIRECCION,
C.TEL_CLI AS TELEFONO,
D.NOM_DIS AS DISTRITO
FROM TB_CLIENTE C
INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
GO
```

CODIGO	RAZON SOCIAL	DIRECCION	TELEFONO	DISTRITO
1	C001	Finseth	Av. Los Viñedos 150	San Miguel
2	C002	Orbi	Av. Emilio Cavenecia 225	La Molina
3	C003	Serviemsa	Jr. Collagate 522	San Miguel
4	C004	Issa	Calle Los Aviadores 263	Surco
5	C005	Mass	Av. Tomas Marsano 880	Surquillo

Figura 146: Listado de clientes
Fuente. - Tomado desde SQL Server 2014

Como verá en la combinación podemos tener el control de todas las columnas es por eso que administraremos de manera conveniente, pero solo podrán ser invocadas aquellas columnas que participan de la combinación.

Si a pesar de tener la combinación correcta entre dos tablas, podemos controlar que columnas deseamos visualizar de la siguiente manera:

```
SELECT C.*
  FROM TB_CLIENTE C
    INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
GO
```

En el script anterior se muestran solo las columnas de la tabla Cliente a pesar de que se encuentran combinadas a la tabla Distrito. También podríamos verlo de la siguiente manera:

```
SELECT D.*
  FROM TB_CLIENTE C
    INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
GO
```

En la cual solo se muestran las columnas de la tabla Distrito dentro de la combinación. Hasta aquí no parece ser tan necesario entender este concepto, pero veremos un caso de aplicación. **Por ejemplo, si necesitamos mostrar los registros de los clientes que viven en el distrito de San Miguel** el script sería el siguiente:

```
SELECT C.*
  FROM TB_CLIENTE C
    INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
      WHERE D.NOM_DIS='SAN MIGUEL'
GO
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto
2	C003	Serviems	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
3	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste
4	C010	Landu	Av. Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza
5	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega

Figura 147: Listado de clientes del distrito de San Miguel con Inner Join

Fuente. - Tomado desde SQL Server 2014

Como vemos en el resultado de la consulta, solo se visualizan los registros de los clientes cuyo distrito es **San Miguel**, es decir, de código **D05**.

5.1.2. Procedimientos almacenados con 2 o más tablas

Ejemplo 01:

Veamos un caso, donde permita mostrar los datos más importantes del producto, así como la línea a la que pertenece. Para esto debe utilizar las tablas TB_PRODUCTO y TB_LINEA que se encuentra en la base de datos **COMERCIO**. Validar la existencia del procedimiento almacenado.

```

IF OBJECT_ID('SP_LISTAR_PRODUCTOS_LINEA') IS NOT NULL
DROP PROCEDURE SP_LISTAR_PRODUCTOS_LINEA
GO
CREATE PROCEDURE SP_LISTAR_PRODUCTOS_LINEA
AS
SELECT
P.COD_PRO AS CÓDIGO,
P.DES_PRO AS DESCRIPCIÓN,
P.PRE_PRO AS PRECIO,
P.STK_ACT AS STOCK,
P.UNI_MED AS UNIDAD_MEDIDA,
P.IMPORTADO AS IMPORTADO,
L.NOM_LIN AS NOMBRE_LINEA
FROM TB_PRODUCTO AS P INNER JOIN TB_LINEA
L ON P.LIN_PRO = L.LIN_PRO GO

```

Ejecutando

```

EXECUTE SP_LISTAR_PRODUCTOS_LINEA
GO

```

Resultado

	CÓDIGO	DESCRIPCIÓN	PRECIO	STOCK	UNIDAD_MEDIDA	IMPORTADO	NOMBRE_LINEA
1	P001	Papel Bond A-4	35.00	200	Mil	VERDADERO	PAPELES
2	P002	Papel Bond Oficio	35.00	50	Mil	FALSO	PAPELES
3	P003	Papel Bulky	10.00	498	Mil	VERDADERO	PAPELES
4	P004	Papel Periódico	9.00	4285	Mil	FALSO	PAPELES
5	P005	Cartucho Tinta Negra	40.00	50	Uni	FALSO	ACCESORIOS PC
6	P006	Cartucho Tinta Color	45.00	58	Uni	FALSO	ACCESORIOS PC
7	P007	Porta Diskettes	3.50	300	Uni	VERDADERO	ACCESORIOS PC
8	P008	Caja de Diskettes * 10	30.00	125	Uni	FALSO	ACCESORIOS PC
9	P009	Borrador de Tinta	10.00	100	Doc	FALSO	UTILES DE OFICINA
10	P010	Borrador Blanco	8.00	2000	Doc	FALSO	UTILES DE OFICINA
11	P011	Tajador Metal	20.00	1120	Doc	FALSO	UTILES DE OFICINA
12	P012	Tajador Plástico	12.00	608	Doc	FALSO	UTILES DE OFICINA
13	P013	Folder Manila Oficio	20.00	200	Cie	FALSO	UTILES DE OFICINA
14	P014	Folder Manila A-4	20.00	150	Cie	VERDADERO	UTILES DE OFICINA
15	P015	Sobre Manila Oficio	15.00	300	Cie	FALSO	UTILES DE OFICINA
16	P016	Sobre Manila A-4	18.00	200	Cie	FALSO	UTILES DE OFICINA
17	P017	Lapicero Negro	10.00	3000	Doc	FALSO	UTILES DE OFICINA
18	P018	Lapicero Azul	10.00	2010	Doc	FALSO	UTILES DE OFICINA
19	P019	Lapicero Rojo	8.00	1900	Doc	VERDADERO	UTILES DE OFICINA
20	P020	Folder Plástico A-4	50.00	3080	Cie	FALSO	UTILES DE OFICINA
21	P021	Protector de Pantalla	50.00	20	Uni	FALSO	ACCESORIOS PC

Figura 148: Listado de productos y su línea a la que pertenece

Fuente. - Tomado desde SQL Server 2014

Ejemplo 02:

Veamos otro caso, donde permita mostrar los datos de la factura, apellidos y nombres del empleado y razón social del cliente. Para esto debe utilizar las tablas TB_FACTURA, TB_CLIENTE, TB_VENDEDOR que se encuentra en la base de datos **COMERCIO**. Validar la existencia del procedimiento almacenado.

```

IF OBJECT_ID('SP_LISTAR_FACTURA_X_CLIENTE_X_VENDEDOR') IS NOT NULL
DROP PROCEDURE SP_LISTAR_FACTURA_X_CLIENTE_X_VENDEDOR GO

CREATE PROCEDURE SP_LISTAR_FACTURA_X_CLIENTE_X_VENDEDOR
AS
SELECT
F.NUM_FAC,
F.FEC_FAC,
F.EST_FAC,
V.APE_VEN+' '+NOM_VEN AS EMPLEADO,
UPPER(C.RAZ_SOC_CLI) AS CLIENTE
FROM TB_FACTURA AS F INNER JOIN TB_CLIENTE AS C
ON F.COD_CLI = C.COD_CLI INNER JOIN TB_VENDEDOR AS V
ON F.COD_VEN = V.COD_VEN
GO

```

Ejecutando

```

EXECUTE
SP_LISTAR_FACTURA_X_CLIENTE_X_VENDEDOR GO

```

Resultado

	NUM_FAC	FEC_FAC	EST_FAC	EMPLEADO	CLIENTE
1	100	1998-06-07	2	MESES JUANA	FINSETH
2	101	1998-06-09	3	SOTO JUAN	COREFO
3	102	1998-01-09	2	OJEDA CESAR	SERVIEMSA
4	103	1998-06-09	2	PALACIOS JOSE	CARDELI
5	104	1998-01-10	2	SALAS RUBEN	MEBA
6	105	1998-10-10	3	VEGA JULIO	PROMINENT
7	106	1998-05-10	1	ARCE PATRICIA	COREFO
8	107	1998-09-10	2	IRIARTE RENATO	SUCERTE
9	108	1998-03-10	2	ARCE PATRICIA	INTECH
10	109	1998-10-01	2	SOTO JUAN	PAYET
11	110	1998-10-11	2	VEGA JULIO	COREFO
12	111	1998-01-12	1	OJEDA CESAR	KADIA
13	112	1998-01-12	3	SALAS RUBEN	FILASUR
14	113	1998-03-12	2	ARCE PATRICIA	CRAMER
15	114	1998-08-12	2	PALACIOS JOSE	MEBA
16	115	1999-06-01	2	VEGA JULIO	CARDELI
17	116	1999-06-01	1	ORTEGA ANA	MEBA
18	117	1999-05-02	3	IRIARTE RENATO	CARDELI
19	118	1999-07-02	3	AREVALO CARL...	INTECH
20	119	1999-06-02	2	SOTO JUAN	HAYASHI
21	120	1999-02-07	1	MESES JUANA	FILASUR

Figura 149: Listado de facturas emitidas
Fuente. - Tomado desde SQL Server 2014

5.1.3. Procedimientos almacenados con 2 o más parámetros

Ejemplo 01:

Veamos un caso, donde permita listar las órdenes mediante un procedimiento almacenado. Pasar 2 parámetros de entrada para el número de orden. Validar la existencia del procedimiento almacenado.

```

IF OBJECT_ID('SP_LISTAR_ORDENES_X_NUMERO') IS NOT NULL
DROP PROCEDURE SP_LISTAR_ORDENES_X_NUMERO GO

CREATE PROCEDURE SP_LISTAR_ORDENES_X_NUMERO
@NUM1 VARCHAR (5),
@NUM2 VARCHAR (5)
AS
SELECT
O.NUM_OCO AS NRO_ORDEN,
O.FEC_OCO AS FECHA_ORDEN,
O.EST_OCO AS ESTADO_ORDEN,
DETA.CAN_PED AS CANTIDAD,
P.DES_PRO AS DESCRIPCION
FROM TB_ORDEN_COMPRA AS O INNER JOIN TB_DETALLE_COMPRA AS DETA
ON O.NUM_OCO = DETA.NUM_OCO INNER JOIN TB_PRODUCTO AS P
ON DETA.COD_PRO = P.COD_PRO WHERE O.NUM_OCO=@NUM1 OR
O.NUM_OCO=@NUM2 GO

```

Ejecutando

```

EXECUTE SP_LISTAR_ORDENES_X_NUMERO 'OC002', 'OC003'
GO

```

Resultado

	NRO_ORDEN	FECHA_ORDEN	ESTADO_ORDEN	CANTIDAD	DESCRIPCION
1	OC002	1998-08-04	1	200	Papel Bulky
2	OC002	1998-08-04	1	500	Cartucho Tinta Negra
3	OC003	1998-02-08	3	50	Cartucho Tinta Negra

Figura 150: Listado de órdenes emitidas
Fuente. - Tomado desde SQL Server 2014

Ejemplo 02:

Veamos un caso, donde permita listar las facturas emitidas en un año específico y que sean productos que pertenecen exclusivamente a una línea. Pasar 2 parámetros de entrada; el primero para fecha factura (considere sólo el año) y el segundo para línea del producto. Validar la existencia del procedimiento almacenado.

```

IF OBJECT_ID('SP_LISTAR_FACTURAS_EMITIDAS') IS NOT NULL
DROP PROCEDURE SP_LISTAR_FACTURAS_EMITIDAS GO

CREATE PROCEDURE SP_LISTAR_FACTURAS_EMITIDAS @AÑO INT,
@LINEA VARCHAR (40)
AS
SELECT
F.NUM_FAC AS NRO_FACTURA,
F.FEC_FAC AS FECHA_FACTURA,
F.EST_FAC AS ESTADO_FACTURA,
P.DES_PRO AS DESCRIPCION,

```

```

L.NOM_LIN AS LINEA_PRODUCTO
FROM TB_FACTURA AS F INNER JOIN TB_DETALLE_FACTURA AS DETA
ON F.NUM_FAC = DETA.NUM_FAC INNER JOIN TB_LINEA AS L INNER
JOIN TB_PRODUCTO AS P
ON L.LIN_PRO = P.LIN_PRO
ON DETA.COD_PRO = P.COD_PRO
WHERE YEAR(F.FEC_FAC) =@AÑO AND
L.NOM_LIN=@LINEA GO

```

Ejecutando

```

EXECUTE SP_LISTAR_FACTURAS_EMITIDAS 1998, 'PAPELES'
GO

```

Resultado

	NRO_FACTURA	FECHA_FACTURA	ESTADO_FACTURA	DESCRIPCION	LINEA_PRODUCTO
1	102	1998-01-09	2	Papel Periódico	PAPELES
2	103	1998-06-09	2	Papel Bond Oficio	PAPELES
3	104	1998-01-10	2	Papel Periódico	PAPELES
4	105	1998-10-10	3	Papel Bulky	PAPELES
5	106	1998-05-10	1	Papel Bond Oficio	PAPELES
6	106	1998-05-10	1	Papel Bulky	PAPELES
7	107	1998-09-10	2	Papel Bulky	PAPELES
8	108	1998-03-10	2	Papel Periódico	PAPELES
9	109	1998-10-01	2	Papel Bond A-4	PAPELES
10	109	1998-10-01	2	Papel Bond Oficio	PAPELES
11	110	1998-10-11	2	Papel Bond Oficio	PAPELES
12	111	1998-01-12	1	Papel Bond Oficio	PAPELES
13	112	1998-01-12	3	Papel Bond Oficio	PAPELES
14	113	1998-03-12	2	Papel Bond Oficio	PAPELES
15	113	1998-03-12	2	Papel Bulky	PAPELES

Figura 151: Listado de facturas emitidas por año y línea

Fuente. - Tomado desde SQL Server 2014

Ejemplo 03:

Veamos un caso, donde permita listar los productos que corresponden a un proveedor. Además, considerar que el stock actual del producto debe ser mayor o igual ingresado en el parámetro de entrada y el precio de venta menor o igual ingresado en el parámetro. Valide la existencia del procedimiento almacenado.

```

IF OBJECT_ID('SP_PRODUCTOS_X_PROVEEDOR') IS NOT
NULL DROP PROCEDURE SP_PRODUCTOS_X_PROVEEDOR GO

CREATE PROCEDURE
SP_PRODUCTOS_X_PROVEEDOR @PROVEEDOR
VARCHAR (80), @STOCK_AC INT,
@PRE_PRO DECIMAL (10,2)
AS
BEGIN

```

```

SELECT
P.COD_PRO AS CÓDIGO,
P.DES_PRO AS DESCRIPCIÓN,
P.PRE_PRO AS PRECIO_PRODUCTO,
P.STK_ACT AS STOCK_ACTUAL,
PR.RAZ_SOC_PRV AS RAZÓN_SOCIAL,
AB.PRE_ABA AS PRECIO_ABASTECIMIENTO
FROM TB_PROVEEDOR AS PR INNER JOIN TB_ABASTECIMIENTO AS
AB ON PR.COD_PRV = AB.COD_PRV INNER JOIN TB_PRODUCTO AS P
ON AB.COD_PRO = P.COD_PRO
WHERE PR.RAZ_SOC_PRV=@PROVEEDOR AND P.STK_ACT>=@STOCK_AC AND
P.PRE_PRO<=@PRE_PRO ORDER BY P.COD_PRO
END
GO

```

Ejecutando

```

EXECUTE SP_PRODUCTOS_X_PROVEEDOR 'FABER CASTELL', 20, 100
GO

```

Resultado

	CÓDIGO	DESCRIPCIÓN	PRECIO_PRODUCTO	STOCK_ACTUAL	RAZÓN_SOCIAL	PRECIO_ABASTECIMIENTO
1	P003	Papel Bulky	10.00	498	Faber Castell	8.00
2	P005	Cartucho Tinta Negra	40.00	50	Faber Castell	35.00
3	P007	Porta Diskettes	3.50	300	Faber Castell	3.00
4	P009	Borrador de Tinta	10.00	100	Faber Castell	8.00
5	P011	Tajador Metal	20.00	1120	Faber Castell	18.00

Figura 152: Listado de productos por proveedor
Fuente. - Tomado desde SQL Server 2014

5.1.4. Ejercicios de aplicación

Actividad 01: COMERCIO

Usando la base de datos **COMERCIO** y por medio de procedimientos almacenados, implemente las siguientes consultas:

Inner Join simple

Script que permita mostrar el número de factura, fecha de facturación, nombre completo del cliente, fecha de cancelación, estado de la factura, nombre completo del vendedor y el porcentaje de IGV aplicado.

```

IF OBJECT_ID('SP_LISTADO1') IS NOT NULL
    DROP PROC SP_LISTADO1
GO

CREATE PROC SP_LISTADO1
AS
    SELECT F.NUM_FAC AS [NUMERO DE FACTURA],
           F.FEC_FAC AS [FECHA DE FACTURACION],
           C.RAZ_SOC_CLI AS [CLIENTE],

```

```

F.FEC_CAN AS [FECHA DE CANCELACION],
F.EST_FAC AS [ESTADO],
V.NOM_VEN+SPACE(1)+V.APE_VEN AS [VENDEDOR],
F.POR_IVG AS [IGV]
FROM TB_FACTURA F
INNER JOIN TB_CLIENTE C ON F.COD_CLI=C.COD_CLI
INNER JOIN TB_VENDEDOR V ON F.COD_VEN=V.COD_VEN
GO

--PRUEBA
EXEC SP_LISTADO1
GO

```

	NUMERO DE FACTURA	FECHA DE FACTURACION	CLIENTE	FECHA DE CANCELACION	ESTADO	VENDEDOR	IGV
1	100	1998-06-07	Finseth	1998-05-08	2	JUANA MESES	0.19
2	101	1998-06-09	Corefo	1998-05-08	3	JUAN SOTO	0.19
3	102	1998-01-09	Serviemsma	1998-03-11	2	CESAR OJEDA	0.19
4	103	1998-06-09	Cardelli	1998-05-11	2	JOSE PALACIOS	0.19
5	104	1998-01-10	Meba	1998-12-10	2	RUBEN SALAS	0.19
6	105	1998-10-10	Prominent	1998-05-08	3	JULIO VEGA	0.19

Figura 153: Listado de registro de facturas
Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar el número de orden de compra, fecha de registro de la orden de compra, nombre completo del proveedor, fecha de atención y el estado de la orden de compra.

```

IF OBJECT_ID('SP_LISTADO2') IS NOT NULL
    DROP PROC SP_LISTADO2
GO

CREATE PROC SP_LISTADO2
AS
    SELECT OC.NUM_OCO AS [NUMERO ORDEN],
           OC.FEC_OCO AS [FECHA DE REGISTRO],
           P.RAZ_SOC_PRV AS [PROVEEDOR],
           OC.FEC_ATE AS [FECHA DE ATENCION],
           OC.EST_OCO AS [ESTADO]
      FROM TB_ORDEN_COMPRA OC
     INNER JOIN TB_PROVEEDOR P ON OC.COD_PRV=P.COD_PRV
GO

--PRUEBA
EXEC SP_LISTADO2
GO

```

	NUMERO ORDEN	FECHA DE REGISTRO	PROVEEDOR	FECHA DE ATENCION	ESTADO
1	OC001	1998-05-03	Invicta	1998-12-03	1
2	OC002	1998-08-04	Petramas	1998-10-04	1
3	OC003	1998-02-08	Miura	1998-02-08	3
4	OC004	1998-05-04	Faber Castell	1998-05-04	3
5	OC005	1998-06-03	Officetec	1998-10-03	1

Figura 154: Listado de órdenes de compra
Fuente.- Tomado desde SQL Server 2014

Inner Join condicionado

Script que permita mostrar todos los productos según el tipo de línea de producto; este deberá ser ingresado como la descripción de la línea de producto.

```

IF OBJECT_ID('SP_LISTADO3') IS NOT NULL
    DROP PROC SP_LISTADO3
GO

CREATE PROC SP_LISTADO3 (@LINEA VARCHAR (30))
AS
    SELECT P.* 
        FROM TB_PRODUCTO P
        INNER JOIN TB_LINEA L ON
            P.LIN_PRO=L.LIN_PRO WHERE L.NOM_LIN=@LINEA
GO

--PRUEBA
EXEC SP_LISTADO3 'UTILES DE OFICINA'
GO

```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P009	Borrador de Tinta	10.00	100	500	Doc	3	FALSO
2	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
3	P011	Tajador Metal	20.00	1120	300	Doc	3	FALSO
4	P012	Tajador Plástico	12.00	608	300	Doc	3	FALSO
5	P013	Folder Manila Oficio	20.00	200	150	Cie	3	FALSO
6	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
7	P015	Sobre Manila Oficio	15.00	300	130	Cie	3	FALSO
8	P016	Sobre Manila A-4	18.00	200	100	Cie	3	FALSO
9	P017	Lapicero Negro	10.00	3000	1000	Doc	3	FALSO
10	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
11	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO
12	P020	Folder Plástico A-4	50.00	3080	1100	Cie	3	FALSO

Figura 155: Listado de productos de tipo útiles de oficina
Fuente. - Tomado desde SQL Server 2014

Script que permite mostrar todas las órdenes de compra registradas según la razón social del proveedor.

```

IF OBJECT_ID('SP_LISTADO4') IS NOT NULL
    DROP PROC SP_LISTADO4
GO

CREATE PROC SP_LISTADO4 (@PROVEEDOR VARCHAR (80))
AS
    SELECT OC.*
        FROM TB_ORDEN_COMPRA OC
        INNER JOIN TB_PROVEEDOR P ON OC.COD_PRV=P.COD_PRV
        WHERE P.RAZ_SOC_PRV=@PROVEEDOR
GO

--PRUEBA
EXEC SP_LISTADO4 'PRAXIS'
GO

```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC009	1998-03-08	PR11	1998-10-09	1
2	OC019	1999-03-03	PR11	1999-06-03	1

Figura 156: Listando las órdenes de compra de un determinado proveedor
 Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar todos los clientes registrados en un determinado distrito.

```
IF OBJECT_ID('SP_LISTADO5') IS NOT
NULL DROP PROC SP_LISTADO5
GO

CREATE PROC SP_LISTADO5 (@DISTRITO VARCHAR(50))
AS
SELECT C.*
FROM TB_CLIENTE C
INNER JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
WHERE D.NOM_DIS=@DISTRITO
GO

--PRUEBA
EXEC SP_LISTADO5 'SAN MIGUEL'
GO
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto
2	C003	Serviems	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
3	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste
4	C010	Landu	Av. Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza
5	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega

Figura 157: Listado de clientes del distrito de San Miguel
 Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar el número de factura, fecha de registro de la factura, descripción del producto, subtotal (CAN_VEN x PRE_VEN) y el nombre completo del vendedor; según el año registrado en la fecha de facturación (FEC_FAC).

```
IF OBJECT_ID('SP_LISTADO6') IS NOT NULL
DROP PROC SP_LISTADO6
GO

CREATE PROC SP_LISTADO6 (@AÑO INT)
AS
SELECT F.NUM_FAC AS [NUMERO DE FACTURA],
F.FEC_FAC AS [FECHA DE FAC.],
P.DES_PRO AS [PRODUCTO],
D.CAN_VEN*D.PRE_VEN AS SUBTOTAL,
V.NOM_VEN+SPACE(1)+V.APE_VEN AS VENDEDOR
FROM TB_DETALLE_FACTURA D
```

```

    INNER JOIN TB_FACTURA F ON F.NUM_FAC=D.NUM_FAC
    INNER JOIN TB_PRODUCTO P ON D.COD_PRO=P.COD_PRO
    INNER JOIN TB_VENDEDOR V ON V.COD_VEN=F.COD_VEN
    WHERE YEAR(F.FEC_FAC) =@AÑO
GO

--PRUEBA
EXEC SP_LISTADO6 1998
GO

```

	NUMERO DE FACTURA	FECHA DE FAC.	PRODUCTO	SUBTOTAL	VENDEDOR
1	100	1998-06-07	Porta Diskettes	30.00	JUANA MESES
2	100	1998-06-07	Tajador Metal	625.00	JUANA MESES
3	100	1998-06-07	Folder Manila Oficio	220.00	JUANA MESES
4	102	1998-01-09	Papel Periódico	80.00	CESAR OJEDA
5	103	1998-06-09	Papel Bond Oficio	400.00	JOSE PALACIOS
6	103	1998-06-09	Tajador Metal	120.00	JOSE PALACIOS
7	103	1998-06-09	Lapicero Negro	252.00	JOSE PALACIOS
8	103	1998-06-09	Lapicero Rojo	120.00	JOSE PALACIOS
9	104	1998-01-10	Papel Periódico	30.00	RUBEN SALAS

Figura 158: Listado de facturas del año 1998

Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar el número de factura, fecha de registro de la factura, nombre del cliente y el estado de la factura solo para las facturas registradas por un determinado vendedor, se debe considerar que la búsqueda se realizará por el nombre y apellido del vendedor.

```

IF OBJECT_ID('SP_LISTADO7') IS NOT NULL
    DROP PROC SP_LISTADO7
GO

CREATE PROC SP_LISTADO7 (@VENDEDOR VARCHAR(40))
AS
    SELECT F.NUM_FAC AS [NUMERO DE FACT.],
           F.FEC_FAC AS [FECHA DE REG.],
           C.RAZ_SOC_CLI AS CLIENTE,
           V.NOM_VEN+SPACE(1)+V.APE_VEN AS [VENDEDOR],
           F.EST_FAC AS ESTADO
    FROM TB_FACTURA F
    INNER JOIN TB_CLIENTE C ON F.COD_CLI=C.COD_CLI
    INNER JOIN TB_VENDEDOR V ON F.COD_VEN=V.COD_VEN
    WHERE V.NOM_VEN+SPACE(1)+V.APE_VEN = @VENDEDOR
GO

--PRUEBA
EXEC SP_LISTADO7 'CESAR OJEDA'
GO

```

	NUMERO DE FACT.	FECHA DE REG.	CLIENTE	VENDEDOR	ESTADO
1	102	1998-01-09	Serviems	CESAR OJEDA	2
2	111	1998-01-12	Kadia	CESAR OJEDA	1

Figura 159: Listado de facturas de un determinado vendedor

Fuente. - Tomado desde SQL Server 2014

Script que permita mostrar todas las facturas registradas por un determinado cliente y un vendedor en ambos casos el factor de búsqueda será los nombres completos tanto del cliente como del vendedor.

```

IF OBJECT_ID('SP_LISTADO8') IS NOT
    NULL DROP PROC SP_LISTADO8
GO
CREATE PROC SP_LISTADO8 (@CLIENTE VARCHAR (30),@VENDEDOR
VARCHAR(40))
AS
    SELECT F. *
        FROM TB_FACTURA F
            INNER JOIN TB_CLIENTE C ON F.COD_CLI=C.COD_CLI
            INNER JOIN TB_VENDEDOR V ON F.COD_VEN=V.COD_VEN
            WHERE C.RAZ_SOC_CLI=@CLIENTE AND
                V.NOM_VEN+SPACE(1)+V.APE_VEN=@VENDEDOR
GO
--PRUEBA
EXEC SP_LISTADO8 'SERVIEMSA', 'CESAR
OJEDA' GO

```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IGV
1	102	1998-01-09	C003	1998-03-11	2	V04	0.19

Figura 159: Listado de facturas de un determinado cliente y vendedor

Fuente. - Tomado desde SQL Server 2014

Resumen

En una consulta multitabla, las tablas que contienen los datos son designadas en la cláusula FROM.

Cada fila de resultados es una combinación de datos procedentes de una única fila en cada una de las tablas, y es la única fila que extrae sus datos de esa combinación particular.

Las consultas multitablas más habituales utilizan la relación padre / hijo creadas por las claves primarias y claves foráneas.

Una tabla puede componerse consigo misma; las auto composiciones requieren el uso de alias.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

[https://technet.microsoft.com/es-es/library/ms190014\(v=sql.105\).aspx](https://technet.microsoft.com/es-es/library/ms190014(v=sql.105).aspx)
Usar combinaciones internas

<http://www.devjoker.com/contenidos/articulos/17/Consultas-combinadas-JOINS.aspx>

Consultas combindadas JOINS
http://www.aulaclic.es/sql/t_3_4.htm

Tutorial para el uso de inner join
<https://es.wikipedia.org/wiki/Join>

Uso de Joins



AGRUPAMIENTOS, SUBCONSULTAS Y VISTAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la sexta unidad, el estudiante crea y emplea agrupamientos, subconsultas y vistas complejas en una base de datos de un proceso de negocio real.

TEMARIO

6.1 Tema 11 : Agrupamiento de datos

- 6.1.1 : Empleo de funciones agregadas
- 6.1.2 : Empleo de GROUP BY y HAVING
- 6.1.3 : Ejercicios con procedimiento almacenado y combinaciones internas

6.2 Tema 12 : Subconsultas

- 6.2.1 : Subconsultas
- 6.2.2 : Subconsultas usando procedimientos almacenados con un parámetro
- 6.2.3 : Subconsultas usando procedimientos almacenados con dos o más parámetros
- 6.2.4 : Ejercicios de aplicación

6.3 Tema 13 : Vistas

- 6.3.1 : Vistas multitable
- 6.3.2 : Clasificación de las vistas

Ejercicios integradores

ACTIVIDADES PROPUESTAS

Los estudiantes integran las subconsultas y agrupamiento de datos en procedimientos almacenados.

Los estudiantes usan los resultados de una consulta como parte de otra en un determinado proceso.

Los estudiantes implementan vistas simples y complejas haciendo uso de varias tablas.

6.1 Agrupamiento de datos

Agrupa un conjunto de filas seleccionado en un conjunto de filas de resumen de acuerdo con los valores de una o más columnas o expresiones en SQL Server 2014. Se devuelve una fila para cada grupo. Las funciones de agregado de la lista <select> de la cláusula SELECT proporcionan información de cada grupo en lugar de filas individuales.

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P001	Papel Bond A-4	35.00	200	1500	Mil	2	VERDADERO
2	P002	Papel Bond Oficio	35.00	50	1500	Mil	2	FALSO
3	P003	Papel Bulky	10.00	498	1000	Mil	2	VERDADERO
4	P004	Papel Periódico	9.00	4285	1000	Mil	2	FALSO
5	P005	Cartucho Tinta Negra	40.00	50	30	Uni	1	FALSO
6	P006	Cartucho Tinta Color	45.00	58	35	Uni	1	FALSO
7	P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO
8	P008	Caja de Diskettes * 10	30.00	125	180	Uni	1	FALSO
9	P009	Borrador de Tinta	10.00	100	500	Doc	3	FALSO
10	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
11	P011	Tajador Metal	20.00	1120	300	Doc	3	FALSO
12	P012	Tajador Plástico	12.00	608	300	Doc	3	FALSO
13	P013	Folder Manila Oficio	20.00	200	150	Cie	3	FALSO
14	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
15	P015	Sobre Manila Oficio	15.00	300	130	Cie	3	FALSO
16	P016	Sobre Manila A-4	18.00	200	100	Cie	3	FALSO
17	P017	Lapicero Negro	10.00	3000	1000	Doc	3	FALSO
18	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
19	P019	Lapicero Rojo	8.00	1900	1000	Doc	3	VERDADERO
20	P020	Folder Plástico A-4	50.00	3080	1100	Cie	3	FALSO
21	P021	Protector de Pantalla	50.00	20	5	Uni	1	FALSO

Figura 160: Listado de productos

Fuente. - Tomado desde SQL Server 2014

De la lista anterior podemos realizar las siguientes consultas:

Listar los productos agrupados por el tipo de importación.

```
SELECT P.IMPORTADO,COUNT(*) AS [TOTAL DE
PRODUCTOS] FROM TB_PRODUCTO P
GROUP BY P.IMPORTADO
GO
```

	IMPORTADO	TOTAL DE PRODUCTOS
1	FALSO	16
2	VERDADERO	5

Figura 161: Listando el total de productos según el tipo de producto

Fuente. - Tomado desde SQL Server 2014

Listar los productos agrupados por la unidad de medida.

```
SELECT P.UNI_MED, COUNT(*) AS [TOTAL DE
UNIDADES] FROM TB_PRODUCTO P
GROUP BY P.UNI_MED
GO
```

	UNI_MED	TOTAL DE UNIDADES
1	Cie	5
2	Doc	7
3	Mil	4
4	Uni	5

Figura 162: Listando el total de unidades por unidad de medida
 Fuente. - Tomado desde SQL Server 2014

Listar los productos agrupados por el número de línea de producto.

```
SELECT P.LIN_PRO, COUNT(*) AS [TOTAL POR LINEA]
FROM TB_PRODUCTO P
GROUP BY P.LIN_PRO
GO
```

	LIN_PRO	TOTAL POR LINEA
1	1	5
2	2	4
3	3	12

Figura 163: Listando el total de productos por línea de producto
 Fuente. - Tomado desde SQL Server 2014

6.1.1. Empleo de funciones agregadas

Las funciones agregadas permiten realizar un cálculo específico sobre un conjunto de valores y al final devuelve un solo valor. Estas funciones deben estar definidas en la especificación de las columnas en una consulta que tenga la cláusula GROUP BY.

Las funciones agregadas se emplean en:

- Una lista de selección de una instrucción SELECT.
- En la implementación de una subconsulta.
- En la implementación de una consulta Externa.
- En la cláusula HAVING del GROUP BY.

Las principales funciones agregadas son:

- Función de conteo (Count)
- Función suma (Sum)
- Función promedio (Avg)
- Función máximo (max)
- Función mínimo (min)

6.1.1.1 Función de conteo (count)

Función agregada que permite calcular el número de registros que puede devolver una consulta de acuerdo a un criterio específico.

Su formato es:

COUNT (EXPRESION)

Debemos considerar que la expresión puede tomar valores como * para hacer referencia a cualquier columna de una tabla o específicamente una columna de tabla.

Veamos un script que permite mostrar el total de vendedores registrados en la empresa:

```
SELECT COUNT(*) AS 'TOTAL DE VENDEDORES'  
FROM TB_VENDEDOR  
GO
```

También podemos usar el siguiente script para representar el total de vendedores registrados:

```
SELECT COUNT(COD_VEN) AS 'TOTAL DE VENDEDORES'  
FROM TB_VENDEDOR  
GO
```

6.1.1.2 Función suma (sum)

Devuelve la suma de todos los valores de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
SUM(EXPRESION)
```

Debemos considerar que la expresión puede tomar valores numéricos de cualquier tipo o también una columna numérica de una tabla.

Veamos un script que permite determinar el monto acumulado de los precios registrados en la tabla producto:

```
SELECT SUM(P.PRE_PRO) AS [ACUMULADO DE PRECIOS]  
FROM TB_PRODUCTO P  
GO
```

6.1.1.3 Función promedio (avg)

Devuelve el promedio de los valores de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
AVG(EXPRESION)
```

Debemos considerar que la expresión puede tomar valores numéricos de cualquier tipo o también una columna numérica de una tabla.

Veamos un script que permite determinar el promedio de los precios registrados en la tabla producto:

```
SELECT AVG(P.PRE_PRO) AS [PROMEDIO DE PRECIOS]  
FROM TB_PRODUCTO P  
GO
```

6.1.1.4 Función máximo (max)

Devuelve el máximo valor de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
MAX(EXPRESION)
```

Debemos considerar que la expresión puede tomar un valor numérico de cualquier tipo o también una columna numérica de una tabla.

Veamos un script que permite determinar el máximo precio registrado en la tabla producto:

```
SELECT MAX(P.PRE_PRO) AS 'MAXIMO PRECIO'  
FROM TB_PRODUCTO P  
GO
```

6.1.1.5 Función mínimo (min)

Devuelve el mínimo valor de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
MIN(EXPRESION)
```

Debemos considerar que la expresión puede tomar un valor numérico de cualquier tipo o también una columna numérica de una tabla.

Veamos un script que permite determinar el mínimo precio registrado en la tabla producto:

```
SELECT MIN(P.PRE_PRO) AS 'MINIMO PRECIO'  
FROM TB_PRODUCTO P  
GO
```

6.1.2. Empleo de GROUP BY, HAVING

Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores NULL en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores NULL no se evalúan en ninguna de las funciones SQL agregadas. Debemos tener en cuenta los siguientes aspectos:

Si usamos la cláusula **WHERE** esta excluirá aquellas filas que no desea agrupar.

Si usamos la cláusula **HAVING** esta permitirá filtrar los registros una vez agrupados.

Por último, debemos mencionar que todos los campos de la lista de la sentencia SELECT deben incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada. Veamos un script que permita listar los valores que pertenecen a la columna importado de la tabla producto.

```
SELECT P.IMPORTADO
FROM TB_PRODUCTO P
GROUP BY P.IMPORTADO
GO
```

IMPORTADO	
1	FALSO
2	VERDADERO

Figura 164: Listando los valores únicos de la columna importado
Fuente. - Tomado desde SQL Server 2014

Hay que tener en cuenta que la columna IMPORTADO contiene un conjunto de valores repetidos por cada producto; y que al colocar la cláusula GROUP BY especificando el nombre de la columna IMPORTADO se buscará los valores distinguidos de dicha columna, es decir mostrará los datos no repetidos. Ahora lo entregaremos con una función agregada para una funcionalidad específica, **por ejemplo**, mostrar el total de productos importados y nacionales.

```
SELECT      P.IMPORTADO,
            COUNT(*) AS [TOTAL DE PRODUCTOS]
FROM        TB_PRODUCTO P
GROUP BY    P.IMPORTADO
GO
```

	IMPORTADO	TOTAL DE PRODUCTOS
1	FALSO	16
2	VERDADERO	5

Figura 165: Total de productos importados y nacionales
Fuente. - Tomado desde SQL Server 2014

La imagen muestra que existen 16 productos nacionales y 5 importados.

6.1.3. Ejercicios con procedimiento almacenado y combinaciones internas

La implementación de procedimientos almacenados usando la cláusula GROUP BY nos permitirá evolucionar la forma básica que hasta ahora se está tratando a los datos agrupados,

Ejemplo 01.

Implemente un procedimiento almacenado que permita listar el total de facturas registradas por año.

```
IF OBJECT_ID('SP_TOTALFACTURASXAÑO') IS NOT NULL
    DROP PROC SP_TOTALFACTURASXAÑO
GO
CREATE PROC SP_TOTALFACTURASXAÑO (@AÑO INT)
AS
```

```

SELECT      YEAR(F.FEC_FAC) AS AÑO,
            COUNT(*) AS [TOTAL DE FACTURAS]
FROM TB_FACTURA F
GROUP BY YEAR(F.FEC_FAC)
HAVING YEAR(F.FEC_FAC) =@AÑO
GO
--PRUEBA
EXEC SP_TOTALFACTURASXAÑO 1998
GO

```

	AÑO	TOTAL DE FACTURAS
1	1998	15

Figura 166: Listando el total de facturas registradas en el año 1998
Fuente. - Tomado desde SQL Server 2014

La cláusula HAVING permite condicionar al resultado de la agrupación. Si realizamos la prueba para el año 1999 el resultado mostraría:

```

EXEC SP_TOTALFACTURASXAÑO 1999
GO

```

	AÑO	TOTAL DE FACTURAS
1	1999	6

Figura 167: Listando el total de facturas registradas en el año 1999
Fuente. - Tomado desde SQL Server 2014

Ejemplo 02.

Implemente un procedimiento almacenado que permita listar la cantidad y la sumatoria acumulado de precios de los productos agrupado por tipo de línea.

```

IF OBJECT_ID('SP_LISTADO_DE_PRODUCTOS_X_LINEA') IS NOT
NULL DROP PROCEDURE SP_LISTADO_DE_PRODUCTOS_X_LINEA GO

CREATE PROCEDURE SP_LISTADO_DE_PRODUCTOS_X_LINEA
@LINEA VARCHAR (30)
AS
BEGIN
SELECT
L.NOM_LIN AS LINEA_PRODUCTO,
COUNT(P.COD_PRO) AS CANTIDAD,
SUM(P.PRE_PRO) AS TOTAL_ACUMULADO
FROM TB_PRODUCTO AS P INNER JOIN TB_LINEA AS
L ON P.LIN_PRO = L.LIN_PRO GROUP BY L.NOM_LIN

HAVING (L.NOM_LIN) =@LINEA
END
GO

```

Ejecutando

Vamos a ejecutar indicando el tipo de línea **UTILES DE OFICINA**

```
EXECUTE SP_LISTADO_DE_PRODUCTOS_X_LINEA 'UTILES DE OFICINA'
GO
```

Resultado

	LINEA_PRODUCTO	CANTIDAD	TOTAL_ACUMULADO
1	UTILES DE OFICINA	12	201.00

Figura 168: Resumen por tipo de línea
Fuente. - Tomado desde SQL Server 2014

6.2 Subconsultas

6.2.1. Subconsultas

Una subconsulta es una consulta SELECT que devuelve un valor único y que puede estar anidada en una instrucción SELECT, INSERT, UPDATE o DELETE, o dentro de otra subconsulta. Una subconsulta se puede utilizar en cualquier parte en la que se permita una expresión. Veamos el primer uso de una subconsulta en donde se listarán todos los clientes de un determinado distrito:

Paso 1: Determinar el código del distrito

```
DECLARE @DISTRITO VARCHAR (50) = 'SAN MIGUEL'
SELECT D.COD_DIS
    FROM TB_DISTRITO D
    WHERE D.NOM_DIS=@DISTRITO
GO
```

El resultado de la ejecución es **D05**, esto nos indica el código que tiene el distrito de San Miguel en la tabla Distrito.

Paso 2: Probarlo en la consulta

```
SELECT C. *
    FROM TB_CLIENTE C
    WHERE C.COD_DIS='D05'
GO
```

Como ya conocemos el código del distrito será cuestión de probarlo en la consulta sobre los datos del cliente; así mismo podremos darnos cuenta que, si el código “D05” proviene del SELECT expuesto en el paso 1, entonces, para poder implementar la subconsulta será cuestión de integrar ambos códigos, tal como se muestra en el paso 3.

Paso 3: Integrando ambos pasos

```

DECLARE @DISTRITO VARCHAR (50) = 'SAN
MIGUEL' SELECT C./*
    FROM TB_CLIENTE C
    WHERE C.COD_DIS = (SELECT D.COD_DIS
                        FROM TB_DISTRITO D
                        WHERE D.NOM_DIS=@DISTRITO)
GO

```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto
2	C003	Serviemsma	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
3	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste
4	C010	Landu	Av.Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza
5	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega

Figura 169: Listado de clientes a partir de una subconsulta
Fuente. - Tomado desde SQL Server 2014

6.2.2. Subconsultas usando procedimiento almacenado con un parámetro

Ejemplo 01:

Procedimiento almacenado que liste los vendedores según el distrito, debe tener en cuenta que el parámetro a ingresar es el nombre del distrito.

```

IF OBJECT_ID('SP_LISTAR_VENDEDORES_POR_DISTrito') IS NOT
NULL DROP PROCEDURE SP_LISTAR_VENDEDORES_POR_DISTrito GO

CREATE PROCEDURE SP_LISTAR_VENDEDORES_POR_DISTrito
(@DIS VARCHAR (30))
AS SELECT V.* FROM TB_VENDEDOR AS V
WHERE V.COD_DIS= (SELECT D.COD_DIS FROM TB_DISTrito AS D
WHERE D.NOM_DIS=@DIS)
GO

```

Ejecutando

Vamos a ejecutar indicando el distrito “SURCO”

```

EXECUTE SP_LISTAR_VENDEDORES_POR_DISTrito
'SURCO' GO

```

Resultado

	COD_VEN	NOM_VEN	APE_VEN	SUE_VEN	FEC_ING	TIP_VEN	COD_DIS
1	V04	CESAR	OJEDA	1450,00	2014-05-05	1	D01
2	V05	JULIO	VEGA	1500,00	2014-01-10	1	D01
3	V10	RENATO	IRIARTE	1550,00	2013-04-16	2	D01

Figura 170: Vendedores por distrito
Fuente. - Tomado desde SQL Server 2014

Ejemplo 02:

Procedimiento almacenado que liste el detalle de ventas que se ha emitido en una factura, debe tener en cuenta que el parámetro a ingresar es número de factura.

```
IF OBJECT_ID('SP_LISTAR_DETALLE_POR_NUMERO') IS NOT
NULL DROP PROCEDURE SP_LISTAR_DETALLE_POR_NUMERO GO

CREATE PROCEDURE SP_LISTAR_DETALLE_POR_NUMERO
@NFAC INT
AS
SELECT
P.DES_PRO AS DESCRIPCIÓN,
DT.CAN_VEN AS CANTIDAD,
DT.PRE_VEN AS PRECIO_VENTA
FROM TB_PRODUCTO AS P INNER JOIN TB_DETALLE_FACTURA
DT ON P.COD_PRO=DT.COD_PRO
WHERE DT.NUM_FAC = (SELECT F.NUM_FAC FROM TB_FACTURA AS F
WHERE F.NUM_FAC= @NFAC)
GO
```

Ejecutando

Vamos a ejecutar indicando el número **100**

```
EXECUTE SP_LISTAR_DETALLE_POR_NUMERO 100
GO
```

Resultado

	DESCRIPCIÓN	CANTIDAD	PRECIO_VENTA
1	Porta Diskettes	6	5,00
2	Tajador Metal	25	25,00
3	Folder Manila Oficio	11	20,00

Figura 171: Detalle de ventas por factura
Fuente. - Tomado desde SQL Server 2014

6.2.3. Subconsultas usando procedimientos almacenados con dos o más parámetros

Ejemplo 01:

Procedimiento almacenado que liste número de orden, razón social, fecha de orden, estado de orden en base al código del proveedor y el año en que se emitió. Debe tener en cuenta que el parámetro a ingresar es código del proveedor y el año.

```
IF OBJECT_ID('SP_LISTAR_ORDENES_POR_PROVEEDOR_y_AÑO') IS NOT
NULL DROP PROCEDURE SP_LISTAR_ORDENES_POR_PROVEEDOR_y_AÑO GO

CREATE PROCEDURE SP_LISTAR_ORDENES_POR_PROVEEDOR_y_AÑO
@CODPROV CHAR (4),
@AÑOORDEN INT
AS
```

```
SELECT O.NUM_OCO, PR.RAZ_SOC_PRV, O.FEC_OCO, O.EST_OCO
FROM TB_ORDEN_COMPRA O JOIN TB_PROVEEDOR PR ON
O.COD_PRV=PR.COD_PRV
WHERE PR.COD_PRV IN (SELECT ABA.COD_PRV FROM TB_ABASTECIMIENTO
ABA WHERE ABA.COD_PRV=@CODPROV) AND YEAR(O.FEC_OCO) =@AÑOORDEN GO
```

Ejecutando

Vamos a ejecutar indicando el código del proveedor PR01 y el año 1998

```
EXECUTE SP_LISTAR_ORDENES_POR_PROVEEDOR_AÑO 'PR01', 1998
GO
```

Resultado

	NUM_OCO	RAZ_SOC_PRV	FEC_OCO	EST_OCO
1	OC004	Faber Castell	1998-05-04	3
2	OC010	Faber Castell	1998-05-09	1

Figura 172: Listado de órdenes por proveedor y año.

Fuente. - Tomado desde SQL Server 2014

6.2.4. Ejercicios de aplicación

Los ejercicios de aplicación estarán en un archivo PDF para descargar de Moodle; tanto para desarrollar en clase; así como para experimenten los estudiantes.

6.3. Vistas

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.

Podríamos determinar algunas ventajas en el uso de las vistas. Como, por ejemplo: 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 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 estas 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**. En el siguiente esquema vemos que la vista alumno (**VALUMNO**) está compuesta por información proveniente de las tablas Alumno, Matricula y Curso.

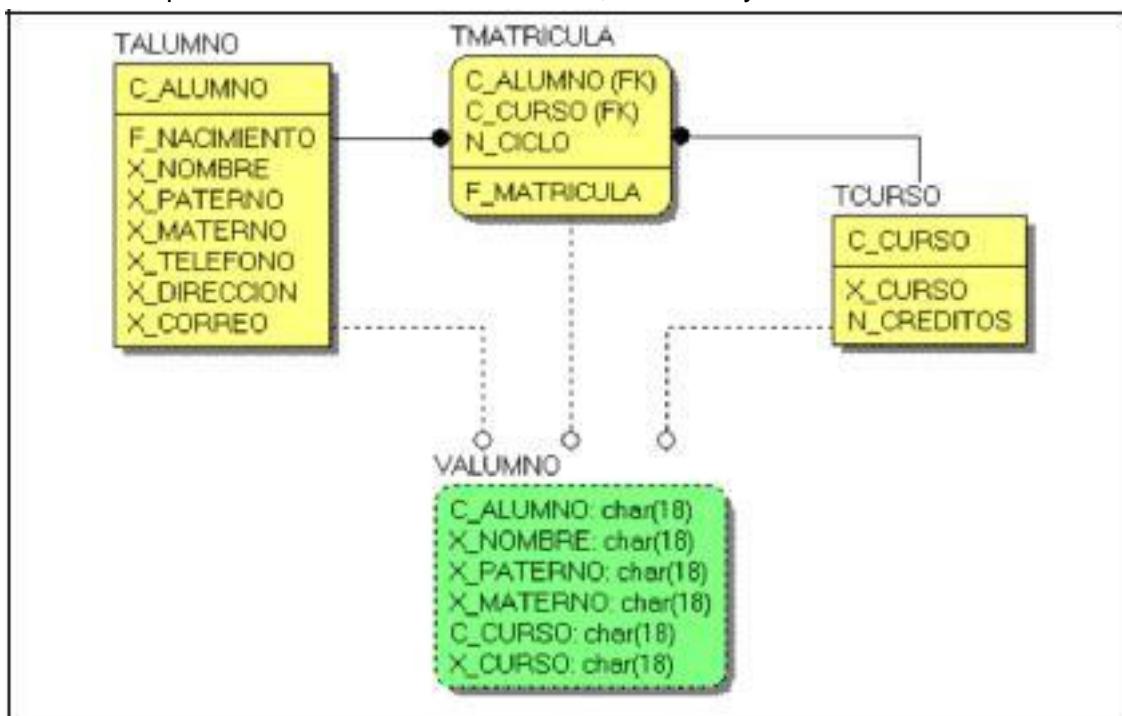


Figura 173: Implementación de la vista Alumno a partir de otras tablas
Fuente. - Tomado desde SQL Server 2014

6.3.1. Vistas multitabla.

Una vista se puede considerar como una tabla virtual o una consulta almacenada. Los datos accesibles a través de una vista no están almacenados en un objeto distinto de la base de datos. Lo que está almacenado en la base de datos es una instrucción SELECT. El resultado de la instrucción SELECT forma la tabla virtual que la vista devuelve. El usuario puede utilizar dicha tabla virtual haciendo referencia al nombre de la vista en instrucciones Transact SQL, de la misma forma en que se hace referencia a las tablas. Una vista no puede contener la cláusula ORDER BY, pero sí lo puede incluir al usar la vista.

Formato de creación de vista

```
CREATE VIEW NOMBRERVISTA
AS
  SENTENCIAS
GO
```

Formato de ejecución de una vista

```
SELECT * FROM NOMBRERVISTA
```

Ejemplo:

Veamos un caso de implementación de vistas en la base de datos COMERCIO, de tal forma que la vista contenga información del código, razón social, teléfono y nombre del distrito de los clientes.

```
IF OBJECT_ID('VCLIENTES') IS NOT NULL
    DROP VIEW VCLIENTES
GO
CREATE VIEW VCLIENTES
AS
    SELECT C.COD_CLI AS CODIGO,
           C.RAZ_SOC_CLI AS RSOCIAL,
           C.TEL_CLI AS TELEFONO,
           D.NOM_DIS AS DISTRITO
      FROM TB_CLIENTE C
     JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
GO

--PROBAR
SELECT * FROM VCLIENTES
```

	CODIGO	RSOCIAL	TELEFONO	DISTRITO
1	C001	Finseth	4342318	San Miguel
2	C002	Orbi	4406335	La Molina
3	C003	Serviemsa	75012403	San Miguel
4	C004	Issa	3725910	Surco
5	C005	Mass	4446177	Surquillo
6	C006	Berker	3810322	San Miguel
7	C007	Fidenza	5289034	Los Olivos
8	C008	Intech	2249493	San Borja
9	C009	Prominent	43233519	Breña
10	C010	Landu	3267840	San Miguel
11	C011	Filasur	4598175	San Juan de Lurigancho

Figura 174: Listando los registros de clientes y el distrito

Fuente. - Tomado desde SQL Server 2014

A partir de la ejecución de la vista, podemos realizar diferentes consultas, **por ejemplo:**

Listar los clientes del distrito de San Miguel

```
SELECT * FROM VCLIENTES V
WHERE V.DISTRITO='SAN MIGUEL'
GO
```

	CODIGO	RSOCIAL	TELEFONO	DISTRITO
1	C002	Orbi	4406335	La Molina
2	C005	Mass	4446177	Surquillo
3	C014	Kadia	4412418	Miraflores
4	C017	Payet	4779834	Barranco
5	C018	Dasin	4657574	Callao

Figura 175: Listando los clientes del distrito de San Miguel

Fuente. - Tomado desde SQL Server 2014

Listar los clientes cuya letra inicial en su razón social empiece con la letra F.

```
SELECT * FROM VCLIENTES V
WHERE V.RSOCIAL LIKE 'F%'
GO
```

	CODIGO	RSOCIAL	TELEFONO	DISTRITO
1	C001	Finseth	4342318	San Miguel
2	C007	Fidenza	5289034	Los Olivos
3	C011	Filasur	4598175	San Juan de Lurigancho

Figura 176: Listando los clientes con letra inicial F
Fuente. - Tomado desde SQL Server 2014

Listar los clientes cuyo número telefónico tenga en su segundo carácter el número 4.

```
SELECT * FROM VCLIENTES V
WHERE V.TELEFONO LIKE '_4%'
GO
```

	CODIGO	RSOCIAL	TELEFONO	DISTRITO
1	C002	Orbi	4406335	La Molina
2	C005	Mass	4446177	Surquillo
3	C014	Kadia	4412418	Miraflores
4	C017	Payet	4779834	Barranco
5	C018	Dasin	4657574	Callao

Figura 177: Listando los clientes cuyo segundo número telefónico sea 4 desde una vista
Fuente. - Tomado desde SQL Server 2014

6.3.2. Clasificación de las vistas.

6.3.2.1 Vistas horizontales

Un uso común de las vistas es restringir el acceso de un usuario a únicamente filas seleccionadas de una tabla.

Ejemplo

Veamos, como implementar dos vistas horizontales que permitan separar los clientes de tipo 1 y 2.

```
IF OBJECT_ID ('VCLIENTES1') IS NOT NULL
    DROP VIEW VCLIENTES1
GO

CREATE VIEW VCLIENTES1
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.TIP_CLI=1
GO
```

```

IF OBJECT_ID ('VCLIENTES2') IS NOT NULL
    DROP VIEW VCLIENTES2
GO

CREATE VIEW VCLIENTES2
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.TIP_CLI=2
GO

--PRUEBA
SELECT * FROM VCLIENTES1
SELECT * FROM VCLIENTES2

```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto
2	C004	Issa	Calle Los Aviadores 263	3725910	46720159	D01	1992-09-12	1	Luis Apumayta
3	C005	Mass	Av. Tomas Mansano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo
4	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste
5	C009	Prominent	Jr. Iquique 132	43233519	NULL	D11	1993-06-11	1	Jorge Valdivia
6	C011	Filasur	Av. El Santuario 1189	4598175	70345201	D26	1990-03-09	1	Angelica Vivas
7	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega
8	C014	Kadia	Av. Santa Cruz 1332 O...	4412418	22202915	D06	1995-05-04	1	Miguel Arce
	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	D04	1990-02-01	2	Alfonso Beltran
2	C003	Serviemsia	Jr. Collagte 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna
3	C007	Fidenza	Jr. El Niquel 282	5289034	16204790	D20	1991-10-02	2	Hector Vivanco
4	C008	Intech	Av. San Luis 2619 5to P	2249493	34021824	D09	1997-01-07	2	Carlos Villanueva
5	C010	Landu	Av. Nicolas de Ayllon 14...	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza
6	C013	Hayashi	Jr. Ayacucho 359	42847990	NULL	D22	1993-06-11	2	Ernesto Uehara
7	C015	Meba	Av. Elmer Fauchett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez
8	C016	Cardeli	Jr. Bartolome Herrera 451	2658853	26403158	D10	1991-12-03	2	Giancarlo Bonifaz
9	C019	Corefo	Av. Canada 3894 - 3898	4377499	57201691	D24	1998-01-03	2	Rosalyn Cortez

Figura 178: Listado de clientes con vistas horizontales

Fuente. - Tomado desde SQL Server 2014

6.3.2.2 Vistas verticales

Una vista vertical se caracteriza por mostrar ciertas columnas que el diseñador desea que visualice el usuario.

Ejemplo 01:

visualicemos los registros de la tabla Producto

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P001	Papel Bond A-4	35.00	200	1500	MII	2	VERDADERO
2	P002	Papel Bond Oficio	35.00	50	1500	MII	2	FALSO
3	P003	Papel Bulky	10.00	498	1000	MII	2	VERDADERO
4	P004	Papel Periódico	9.00	4285	1000	MII	2	FALSO
5	P005	Cartucho Tinta Negra	40.00	50	30	Uni	1	FALSO
6	P006	Cartucho Tinta Color	45.00	58	35	Uni	1	FALSO
7	P007	Porta Diskettes	3.50	300	100	Uni	1	VERDADERO
8	P008	Caja de Diskettes * 10	30.00	125	180	Uni	1	FALSO
9	P009	Borrador de Tinta	10.00	100	500	Dec	3	FALSO

Figura 179: Listado de registros de productos

Fuente. - Tomado desde SQL Server 2014

Ejemplo 02:

Creando una vista de los productos, el cual permitirá a los usuarios visualizar el nombre del producto, importado y unidad de medida.

```

IF OBJECT_ID ('VPRODUCTOS') IS NOT NULL
    DROP VIEW VPRODUCTOS
GO
CREATE VIEW VPRODUCTOS
AS
    SELECT P.DES_PRO AS DESCRIPCION,
           P.IMPORTADO AS IMPORTADO,
           P.UNI_MED AS [UNIDAD DE MEDIDA]
      FROM TB_PRODUCTO P
GO

--PRUEBA
SELECT * FROM VPRODUCTOS
GO

```

EJERCICIOS INTEGRADORES

Actividad

Usando la base de datos **COMERCIO** implemente los siguientes casos:

Agrupamiento de datos

Procedimiento almacenado que permita mostrar el total de productos agrupados según la unidad de medida.

```

IF OBJECT_ID('SP_TOTALPRODUCTOS') IS NOT NULL
    DROP PROC SP_TOTALPRODUCTOS
GO

CREATE PROC SP_TOTALPRODUCTOS
AS
    SELECT P.UNI_MED AS [UNIDAD DE MEDIDA],
           COUNT(*) AS [TOTAL]
      FROM TB_PRODUCTO P
     GROUP BY P.UNI_MED
GO

EXEC SP_TOTALPRODUCTOS
GO

```

Procedimiento almacenado que permita mostrar el total de clientes agrupados por distrito.

```

IF OBJECT_ID('SP_TOTALCLIENTES') IS NOT NULL
    DROP PROC SP_TOTALCLIENTES
GO

```

```

CREATE PROC SP_TOTALCLIENTES
AS
    SELECT D.NOM_DIS AS [DISTRITO],
           COUNT(*) AS [TOTAL]
      FROM TB_CLIENTE C
     JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
    GROUP BY D.NOM_DIS
GO

EXEC SP_TOTALCLIENTES
GO

```

	DISTRITO	TOTAL
1	Barranco	1
2	Bellavista	1
3	Breña	1
4	Callao	1
5	Jesús María	1
6	La Molina	1
7	Lima - Cercado	1
8	Lince	1
9	Los Olivos	1
10	Miraflores	1

Figura 180: Listando la cantidad de clientes por distrito
Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permita mostrar el total de facturas registradas por un vendedor, hay que tener en cuenta que se debe mostrar el nombre completo del vendedor.

```

IF OBJECT_ID('SP_TOTALFACTURAS') IS NOT NULL
    DROP PROC SP_TOTALFACTURAS
GO

CREATE PROC SP_TOTALFACTURAS
AS
    SELECT V.NOM_VEN+SPACE(1)+V.APE_VEN AS [VENDEDOR],
           COUNT(*) AS [TOTAL]
      FROM TB_FACTURA F
     JOIN TB_VENDEDOR V ON F.COD_VEN=V.COD_VEN
    GROUP BY V.NOM_VEN+SPACE(1)+V.APE_VEN
GO

EXEC SP_TOTALFACTURAS
GO

```

	VENDEDOR	TOTAL
1	ANA ORTEGA	1
2	CARLOS AREVALO	1
3	CESAR OJEDA	2
4	JOSE PALACIOS	2
5	JUAN SOTO	3
6	JUANA MESES	2
7	JULIO VEGA	3
8	PATRICIA ARCE	3
9	RENATO IRIARTE	2
10	RUBEN SALAS	2

Figura 181: Listando el total de facturas por vendedor

Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permite mostrar el año de registro de la factura, la razón social del cliente y el total facturas registradas por el cliente en un determinado año.

```

IF OBJECT_ID('SP_FACTURASXCLIENTE') IS NOT NULL
    DROP PROC SP_FACTURASXCLIENTE
GO

CREATE PROC SP_FACTURASXCLIENTE
AS
    SELECT      YEAR(F.FEC_FAC) AS AÑO,
                C.RAZ_SOC_CLI AS [CLIENTE],
                COUNT(*) AS [TOTAL]
    FROM TB_FACTURA F
    JOIN TB_CLIENTE C ON F.COD_CLI=C.COD_CLI
    GROUP BY YEAR(F.FEC_FAC),C.RAZ_SOC_CLI
GO

EXEC SP_FACTURASXCLIENTE
GO

```

	AÑO	CLIENTE	TOTAL
1	1998	Cardeli	1
2	1999	Cardeli	2
3	1998	Corefo	3
4	1998	Cramer	1
5	1998	Filasur	1
6	1999	Filasur	1
7	1998	Finseth	1
8	1999	Hayashi	1

Figura 182: Listando el total de facturas por año y por cliente

Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permita mostrar el máximo monto acumulado agrupados por el número de la factura.

```
IF OBJECT_ID('SP_MONTOSXFACTURA') IS NOT NULL
    DROP PROC SP_MONTOSXFACTURA
GO

CREATE PROC SP_MONTOSXFACTURA
AS
    SELECT      D.NUM_FAC AS [NUMERO DE FACT],
                MAX(D.PRE_VEN*D.CAN_VEN) AS [MAXIMO SUBTOTAL]
    FROM TB_DETALLE_FACTURA D
    GROUP BY D.NUM_FAC
GO

EXEC SP_MONTOSXFACTURA
GO
```

	NUMERO DE FACT	MAXIMO SUBTOTAL
1	100	625.00
2	102	80.00
3	103	400.00
4	104	250.00
5	105	2500.00
6	106	700.00
7	107	240.00
8	108	2500.00
9	109	525.00
10	110	105.00

Figura 183: Listando los máximos montos de cada factura
Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permite mostrar el monto total acumulado por año según la fecha de facturación.

```
IF OBJECT_ID('SP_MONTOxAÑO') IS NOT NULL
    DROP PROC SP_MONTOxAÑO
GO

CREATE PROC SP_MONTOxAÑO
AS
    SELECT      YEAR(F.FEC_FAC) AS AÑO,
                SUM(D.PRE_VEN*D.CAN_VEN) AS [MONTO TOTAL]
    FROM TB_FACTURA F
    JOIN TB_DETALLE_FACTURA D ON D.NUM_FAC=F.NUM_FAC
    GROUP BY YEAR(F.FEC_FAC)
GO

EXEC SP_MONTOxAÑO
GO
```

	AÑO	MONTO TOTAL
1	1998	15891.00
2	1999	18627.00

Figura 184: Listando el monto total acumulado de ventas por años
 Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permita mostrar el total de facturas agrupadas por año y mes.

```
IF OBJECT_ID('SP_FACTURASXAÑOXMES') IS NOT NULL
    DROP PROC SP_FACTURASXAÑOXMES
GO

CREATE PROC SP_FACTURASXAÑOXMES
AS
    SELECT      YEAR(F.FEC_FAC) AS [AÑO],
                MONTH(F.FEC_FAC) AS [MES],
                COUNT(*) AS [TOTAL DE FACTURAS]
        FROM TB_FACTURA F
       GROUP BY YEAR(F.FEC_FAC), MONTH(F.FEC_FAC)
GO

EXEC SP_FACTURASXAÑOXMES
GO
```

	AÑO	MES	TOTAL DE FACTURAS
1	1998	1	4
2	1999	2	1
3	1998	3	2
4	1998	5	1
5	1999	5	1
6	1998	6	3
7	1999	6	3
8	1999	7	1
9	1998	8	1
10	1998	9	1
11	1998	10	3

Figura 185: Listando el total de facturas por año y mes
 Fuente. - Tomado desde SQL Server 2014

SubConsultas

Procedimiento almacenado que permita listar las órdenes de compra de un determinado proveedor buscado por la razón social del proveedor, especifique las cabeceras de la mejor manera posible.

```
IF OBJECT_ID('SP_SUBCONSULTA1') IS NOT NULL
    DROP PROC SP_SUBCONSULTA1
```

```
GO
```

```
CREATE PROC SP_SUBCONSULTA1(@RSOCIAL VARCHAR (80))
AS
    SELECT O.NUM_OCO AS [NUM. DE ORDEN],
           O.FEC_OCO AS [FECHA DE REG.],
           (SELECT P.RAZ_SOC_PRV FROM TB_PROVEEDOR P
            WHERE P.COD_PRV=O.COD_PRV) AS [PROVEEDOR],
           O.FEC_ATE AS [FECHA DE ATEN.],
           O.EST_OCO AS [ESTADO DE ORD.]
      FROM TB_ORDEN_COMPRA O
     WHERE O.COD_PRV=(SELECT P.COD_PRV FROM TB_PROVEEDOR P
                      WHERE P.RAZ_SOC_PRV=@RSOCIAL)
```

```
GO
```

```
EXEC SP_SUBCONSULTA1 'FABER CASTELL'
```

```
GO
```

	NUM. DE ORDEN	FECHA DE REG.	PROVEEDOR	FECHA DE ATEN.	ESTADO DE ORD.
1	OC004	1998-05-04	Faber Castell	1998-05-04	3
2	OC010	1998-05-09	Faber Castell	1998-05-09	1

Figura 186: Listando órdenes de compra del proveedor Faber Castell
Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permita listar las facturas de un determinado cliente y vendedor.

```
IF OBJECT_ID('SP_SUBCONSULTA3') IS NOT NULL
    DROP PROC SP_SUBCONSULTA3
GO

CREATE PROC SP_SUBCONSULTA3(@CLIENTE VARCHAR(80), @VENDEDOR
VARIABLE(80))
AS
    SELECT F.*
        FROM TB_FACTURA F
        WHERE F.COD_CLI=(SELECT C.COD_CLI
                          FROM TB_CLIENTE C
                          WHERE C.RAZ_SOC_CLI=@CLIENTE) AND
              F.COD_VEN=(SELECT V.COD_VEN
                          FROM TB_VENDEDOR V
                          WHERE V.NOM_VEN+SPACE(1)+V.APE_VEN=@VENDEDOR)
GO

EXEC SP_SUBCONSULTA3 'SERVIEMSA', 'CESAR OJEDA'
GO
```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IVG
1	102	1998-01-09	C003	1998-03-11	2	V04	0.19

Figura 187: Listando las facturas de un determinado cliente y vendedor

Fuente. - Tomado desde SQL Server 2014

Procedimiento almacenado que permita mostrar las órdenes de compra de un determinado proveedor en un determinado año.

```

IF OBJECT_ID('SP_SUBCONSULTA4') IS NOT NULL
    DROP PROC SP_SUBCONSULTA4
GO
CREATE PROC SP_SUBCONSULTA4(@PROVEEDOR VARCHAR(80),@AÑO INT)
AS
    SELECT *
        FROM TB_ORDEN_COMPRA O
        WHERE O.COD_PRV=(SELECT P.COD_PRV
                            FROM TB_PROVEEDOR P
                            WHERE P.RAZ_SOC_PRV=@PROVEEDOR) AND
YEAR(O.FEC_OCO)=@AÑO
GO

EXEC SP_SUBCONSULTA4 'FABER CASTELL',1998
GO

```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC004	1998-05-04	PR01	1998-05-04	3
2	OC010	1998-05-09	PR01	1998-05-09	1

Figura 187: Listando las órdenes del proveedor Faber Castell en el año 1998

Fuente. - Tomado desde SQL Server 2014

Vistas

Vista que permita mostrar los datos de los proveedores. Seguidamente cree un procedimiento almacenado que, usando la vista, filtre a los vendedores por el nombre del distrito.

```

IF OBJECT_ID('VISTA1') IS NOT NULL
    DROP PROC VISTA1
GO

CREATE VIEW VISTA1
AS
    SELECT P.COD_PRV AS CODIGO,
           P.RAZ_SOC_PRV AS PROVEEDOR,
           P.DIR_PRV AS DIRECCION,
           P.TEL_PRV AS TELEFONO,
           D.NOM_DIS AS DISTRITO,
           P.REP_VEN AS REPRESENTANTE
      FROM TB_PROVEEDOR P
     INNER JOIN TB_DISTRITO D ON P.COD_DIS=D.COD_DIS
GO

```

```

IF OBJECT_ID('SP_VISTA1') IS NOT NULL
    DROP PROC SP_VISTA1
GO

CREATE PROC SP_VISTA1
AS
    SELECT * FROM VISTA1
GO

EXEC SP_VISTA1
GO

```

	CODIGO	PROVEEDOR	DIRECCION	TELEFONO	DISTRITO	REPRESENTANTE
1	PR01	Faber Castell	Av. Isabel La Catolica 1875	4330895	Rimac	Carlos Aguirre
2	PR02	Atlas	Av. Lima 471	5380926	Rimac	Cesar Torres
3	PR03	3M	Av. Venezuela 3018	2908165	Bellavista	Omar Injoque

Figura 188: Listando los datos del proveedor desde una vista
Fuente. - Tomado desde SQL Server 2014

Vista que permita mostrar el número de factura, fecha de factura, nombre completo del cliente y nombre completo del vendedor. Luego cree un procedimiento almacenado que usando la vista, filtre las facturas por año.

```

IF OBJECT_ID('VISTA2') IS NOT NULL
    DROP PROC VISTA2
GO

CREATE VIEW VISTA2
AS
    SELECT      F.NUM_FAC AS [NUMERO],
                F.FEC_FAC AS [FECHA],
                C.RAZ_SOC_CLI AS [RAZONSOCIAL],
                V.NOM_VEN+SPACE(1)+V.APE_VEN AS [VENDEDOR]
    FROM TB_FACTURA F
    JOIN TB_CLIENTE C ON F.COD_CLI=C.COD_CLI
    JOIN TB_VENDEDOR V ON V.COD_VEN=F.COD_VEN
GO

IF OBJECT_ID('SP_VISTA2') IS NOT NULL
    DROP PROC SP_VISTA2
GO

CREATE PROC SP_VISTA2 (@AÑO INT)
AS
    SELECT V.* FROM VISTA2 V WHERE YEAR(V.FECHA)=@AÑO
GO

EXEC SP_VISTA2 1998
GO

```

	NUMERO	FECHA	RAZONSOCIAL	VENDEDOR
1	100	1998-06-07	Finseth	JUANA MESES
2	101	1998-06-09	Corefo	JUAN SOTO
3	102	1998-01-09	Serviems	CESAR OJEDA
4	103	1998-06-09	Cardeli	JOSE PALACIOS
5	104	1998-01-10	Meba	RUBEN SALAS
6	105	1998-10-10	Prominent	JULIO VEGA
7	106	1998-05-10	Corefo	PATRICIA ARCE
8	107	1998-09-10	Sucerte	RENATO IRIA...
9	108	1998-03-10	Intech	PATRICIA ARCE
10	109	1998-10-01	Payet	JUAN SOTO
11	110	1998-10-11	Corefo	JULIO VEGA
12	111	1998-01-12	Kadia	CESAR OJEDA
13	112	1998-01-12	Filasur	RUBEN SALAS
14	113	1998-03-12	Cramer	PATRICIA ARCE
15	114	1998-08-12	Meba	JOSE PALACIOS

Figura 189: Listando las facturas mediante una vista del año 1998

Fuente. - Tomado desde SQL Server 2014

Implemente vistas para separar las órdenes de compra por el estado de las órdenes de compra (1, 2 o 3). Seguidamente cree un procedimiento almacenado que ingresando como parámetro el estado de la orden (1, 2 o 3) muestra los datos de la vista correspondiente.

--CREANDO LAS VISTAS

```

IF OBJECT_ID('VISTA_ORDENES1') IS NOT
    NULL DROP PROC VISTA_ORDENES1
GO
CREATE VIEW VISTA_ORDENES1
AS
    SELECT      O.*
        FROM TB_ORDEN_COMPRA O
        WHERE O.EST_OCO=1
GO
IF OBJECT_ID('VISTA_ORDENES2') IS NOT
    NULL DROP PROC VISTA_ORDENES2
GO

CREATE VIEW VISTA_ORDENES2
AS
    SELECT      O.*
        FROM TB_ORDEN_COMPRA O
        WHERE O.EST_OCO=2
GO
IF OBJECT_ID('VISTA_ORDENES3') IS NOT NULL
    DROP PROC VISTA_ORDENES3
GO
CREATE VIEW VISTA_ORDENES3
AS
    SELECT      O.*
        FROM TB_ORDEN_COMPRA O
        WHERE O.EST_OCO=3
GO

```

```
--CREANDO EL PROCEDIMIENTO ALMACENADO
IF OBJECT_ID('SP_SELECCIONAVISTA') IS NOT NULL
    DROP PROC SP_SELECCIONAVISTA
GO

CREATE PROC SP_SELECCIONAVISTA(@ESTADO INT)
AS
    IF (@ESTADO=1)
        SELECT * FROM VISTA_ORDENES1
    ELSE IF (@ESTADO=2)
        SELECT * FROM VISTA_ORDENES2
    ELSE
        SELECT * FROM VISTA_ORDENES3
GO

--PROBANDO EL PROCEDIMIENTO ALMACENADO
EXEC SP_SELECCIONAVISTA 1 GO
```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC001	1998-05-03	PR08	1998-12-03	1
2	OC002	1998-08-04	PR16	1998-10-04	1
3	OC005	1998-06-03	PR07	1998-10-03	1
4	OC006	1998-02-01	PR19	1998-02-01	1
5	OC008	1998-02-06	PR04	1998-01-07	1
6	OC009	1998-03-08	PR11	1998-10-09	1
7	OC010	1998-05-09	PR01	1998-05-09	1
8	OC011	1998-02-10	PR03	1998-03-10	1
9	OC013	1998-02-11	PR05	1998-06-11	1
10	OC014	1998-03-11	PR19	1998-05-12	1
11	OC015	1998-03-11	PR18	1998-10-12	1
12	OC017	1999-08-01	PR09	1999-08-01	1
13	OC018	1999-01-02	PR20	1999-08-02	1

Figura 190: Listando las órdenes de compra con estado 1

Fuente. - Tomado desde SQL Server 2014

Implemente vistas que permitan separar las facturas registradas en el año 1998 y 1999 respectivamente. Seguidamente cree un procedimiento almacenado que muestre dichas vistas según el año ingresado como parámetro, en caso ingreso un año no registrado mostrar el mensaje “Año no registra ventas”

```
--CREANDO LAS VISTAS
IF OBJECT_ID('VISTA_FACT1998') IS NOT
    NULL DROP PROC VISTA_FACT1998
GO

CREATE VIEW VISTA_FACT1998
AS
    SELECT      F. *
    FROM TB_FACTURA F
    WHERE YEAR(F.FEC_FAC) = 1998
GO
```

```

IF OBJECT_ID('VISTA_FACT1999') IS NOT NULL
    DROP PROC VISTA_FACT1999
GO

CREATE VIEW VISTA_FACT1999
AS
    SELECT      F.*
        FROM TB_FACTURA F
        WHERE YEAR(F.FEC_FAC) =1999
GO

--CREANDO EL PROCEDIMIENTO ALMACENADO
IF OBJECT_ID('SP_SELECCIONAVISTA') IS NOT NULL
    DROP PROC SP_SELECCIONAVISTA
GO

CREATE PROC SP_SELECCIONAVISTA (@AÑO INT)
AS
    IF (@AÑO=1998)
        SELECT * FROM VISTA_FACT1998
    ELSE IF (@AÑO=1999)
        SELECT * FROM VISTA_FACT1999
    ELSE
        PRINT 'AÑO NO REGISTRA VENTAS'
GO

--PROBANDO EL PROCEDIMIENTO ALMACENADO
EXEC SP_SELECCIONAVISTA 1999 GO

```

	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IVG
1	115	1999-06-01	C016	1999-09-01	2	V05	0.19
2	116	1999-06-01	C015	1999-06-01	1	V06	0.19
3	117	1999-05-02	C016	1999-04-02	3	V10	0.19
4	118	1999-07-02	C008	1999-01-03	3	V03	0.19
5	119	1999-06-02	C013	1999-10-03	2	V02	0.19
6	120	1999-02-07	C011	1999-02-23	1	V01	0.19

Figura 191: Listando las facturas registradas en el año 1999

Fuente. - Tomado desde SQL Server 2014

Resumen

Las funciones de agrupamiento sirven para mostrar resúmenes de datos.

Una subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT. Se puede asignar en la cláusula WHERE una subconsulta al buscar un determinado valor.

Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis.

La subconsulta no puede contener la cláusula ORDER BY, ni puede ser la UNION de varias sentencias SELECT, además tiene algunas restricciones en cuanto a número de columnas según el lugar donde aparece en la consulta principal.

Cuando se ejecuta una consulta que contiene una subconsulta, la subconsulta se ejecuta por cada fila de la consulta principal.

Una vista es una consulta que es accesible como una tabla virtual en una base de datos relacional.

Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que solo se almacena de ellas la definición, no los datos. Los datos que se recuperan mediante una consulta a una vista se presentarán igual que los de una tabla.

Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

[https://es.wikipedia.org/wiki/Vista_\(base_de_datos\)](https://es.wikipedia.org/wiki/Vista_(base_de_datos))
Vistas en SQL Server.

<http://deletesql.com/viewtopic.php?f=5&t=13>
Agrupamiento de datos.

<http://basededatos.umh.es/sql/sql04.htm>
Funciones de agrupamiento.

<https://desarrolloweb.com/articulos/2337.php>
Subconsultas en SQL.



DIAGRAMADOR DE BASE DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la séptima unidad, el estudiante empleará correctamente el diagramador de SQL Server 2014 y la aplicará en su proyecto.

TEMARIO

7.1 Tema 14 : Diagramador de SQL Server 2014

7.1.1 : Uso del diagramador de base de datos SQL Server 2014

ACTIVIDADES PROPUESTAS

Los estudiantes emplean el diagramador de base de datos en la base de datos de su proyecto.

7.1. Diagramador de SQL Server 2014

Puede utilizar el Explorador de objetos para crear nuevos diagramas de base de datos. Estos diagramas muestran gráficamente la estructura de la base de datos. Mediante el uso de diagramas de base de datos puede crear y modificar tablas, columnas, relaciones y claves. Además, puede modificar índices y restricciones.

7.1.1 Uso del diagramador de base de datos de SQL Server 2014

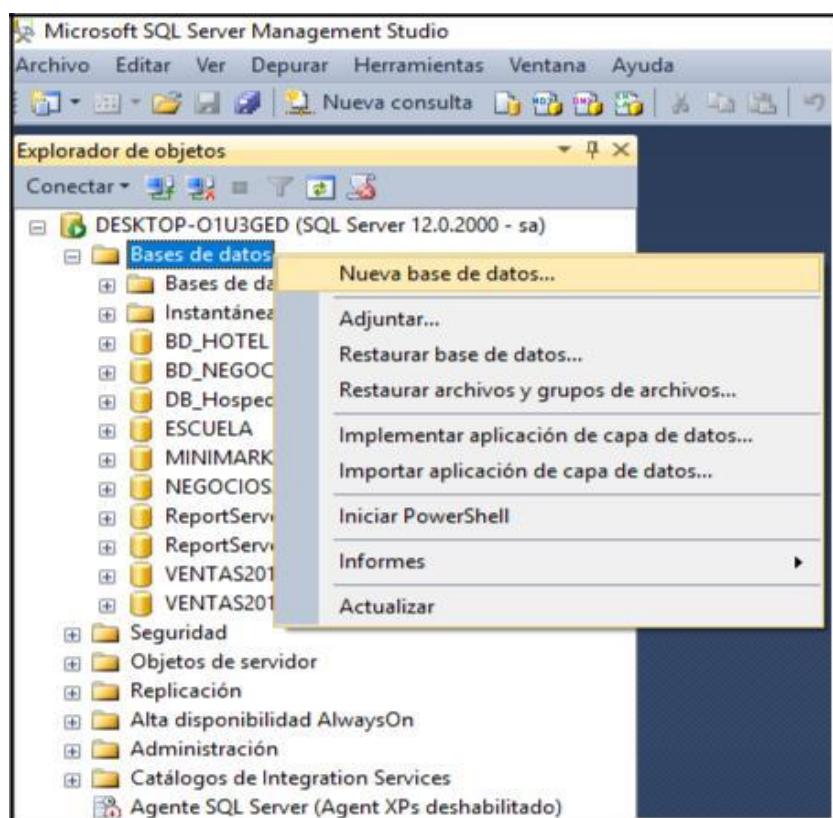
EJEMPLO:

Para esto tenemos que tener instalado SQL Server 2014.

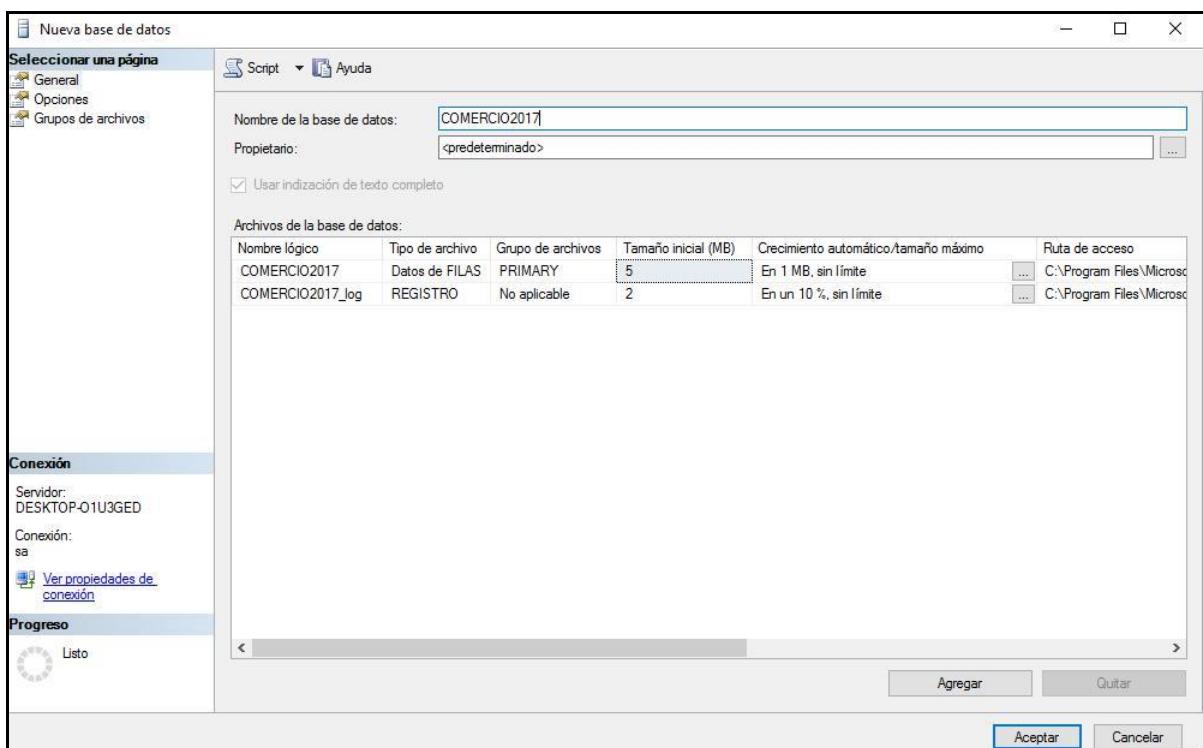
7.1.1.1 Crear una base de datos

En el Explorador de objetos, conéctese a una instancia del SQL Server Database Engine (Motor de base de datos de SQL Server) y expándala.

Haga clic con el botón secundario en Bases de datos y, a continuación, en Nueva base de datos.



- c) En Nueva base de datos, especifique un nombre a la base de datos.



Si desea crear la base de datos aceptando todos los valores predeterminados, haga clic en **Aceptar**; de lo contrario, continúe con los siguientes pasos (opcionales).

Para cambiar el propietario, haga clic en (...), con el fin de seleccionar otro.

Para cambiar los valores predeterminados de los archivos de datos y de registro de transacciones principales, en la cuadricula Archivos de la base de datos, haga clic en la celda correspondiente y especifique el nuevo valor.

Para cambiar opciones de base de datos, seleccione la página Opciones y modifique las opciones de la base de datos.

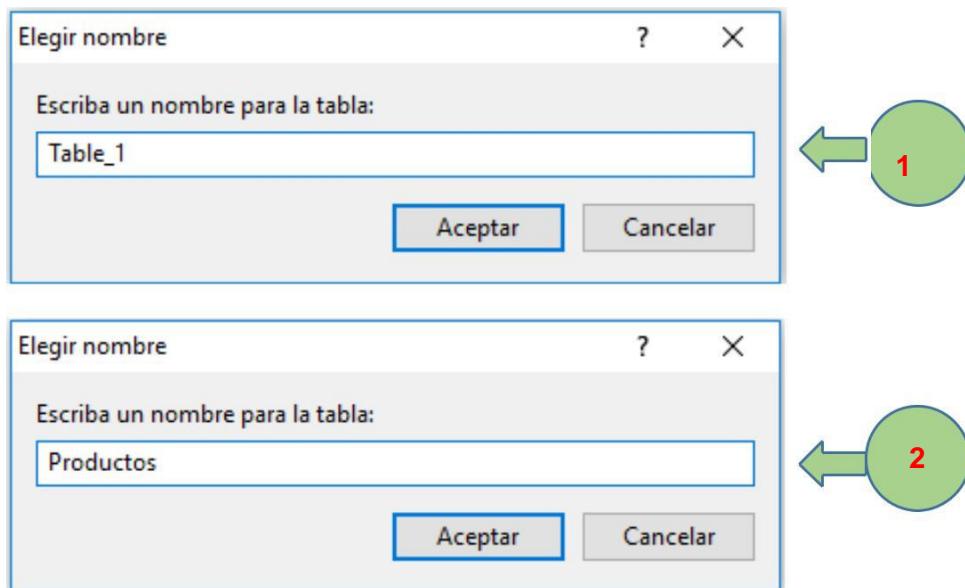
Para crear la base de datos, haga clic en **Aceptar**.

7.1.1.2 Crear tablas

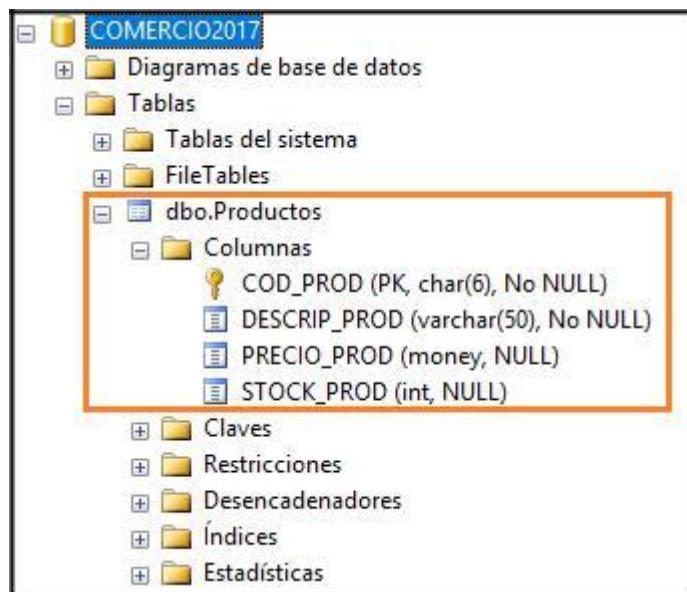
Seleccione la base de datos **COMERCIO2017**, carpeta tablas, menú contextual y agregue campo por campo especificando el tipo de dato, longitud y definir si el campo será null o not null. Además, asignar una clave principal que campo correspondiente.

	Nombre de columna	Tipo de datos	Permitir val...
PK	COD_PROD	char(6)	<input type="checkbox"/>
	DESCRIP_PROD	varchar(50)	<input type="checkbox"/>
	PRECIO_PROD	money	<input checked="" type="checkbox"/>
	STOCK_PROD	int	<input checked="" type="checkbox"/>

Guarde la tabla e ingrésale un nombre. Para el caso asigne, **Productos**



La tabla aparecerá en el explorador de objetos:



7.1.1.3 Propiedades de un campo/atributo

DESKTOP-O1U3GED....7 - dbo.Productos X

Nombre de columna	Tipo de datos	Permitir val...
COD_PROD	char(6)	<input type="checkbox"/>
DESCRIP_PROD	varchar(50)	<input type="checkbox"/>
PRECIO_PROD	money	<input checked="" type="checkbox"/>
STOCK_PROD	int	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Propiedades de columna

(General)

(Nombre)	PRECIO_PROD
Permitir valores NULL	Sí
Tipo de datos	money
Valor o enlace predeterminado	

Diseñador de tablas

Conjunto de columnas	No
Descripción	
Determinístico	Sí
Disperso	No
Especificación de columna calculada	
Especificación de identidad	No
Especificación de texto completo	No
Indizable	Sí

7.1.1.4 Asignando restricciones a los campos

DEFAULT: Establece un valor por defecto.

Puede definir un valor por defecto usando las propiedades de los atributos:

Ejemplo: Asignar al campo **PRECIO_PROD** por defecto el valor 0.

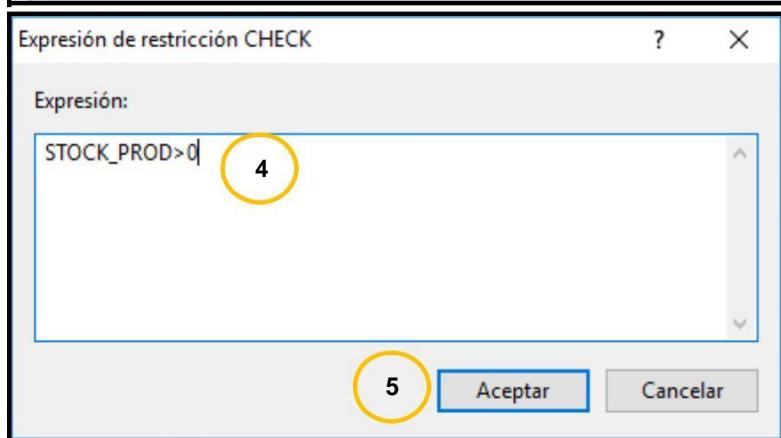
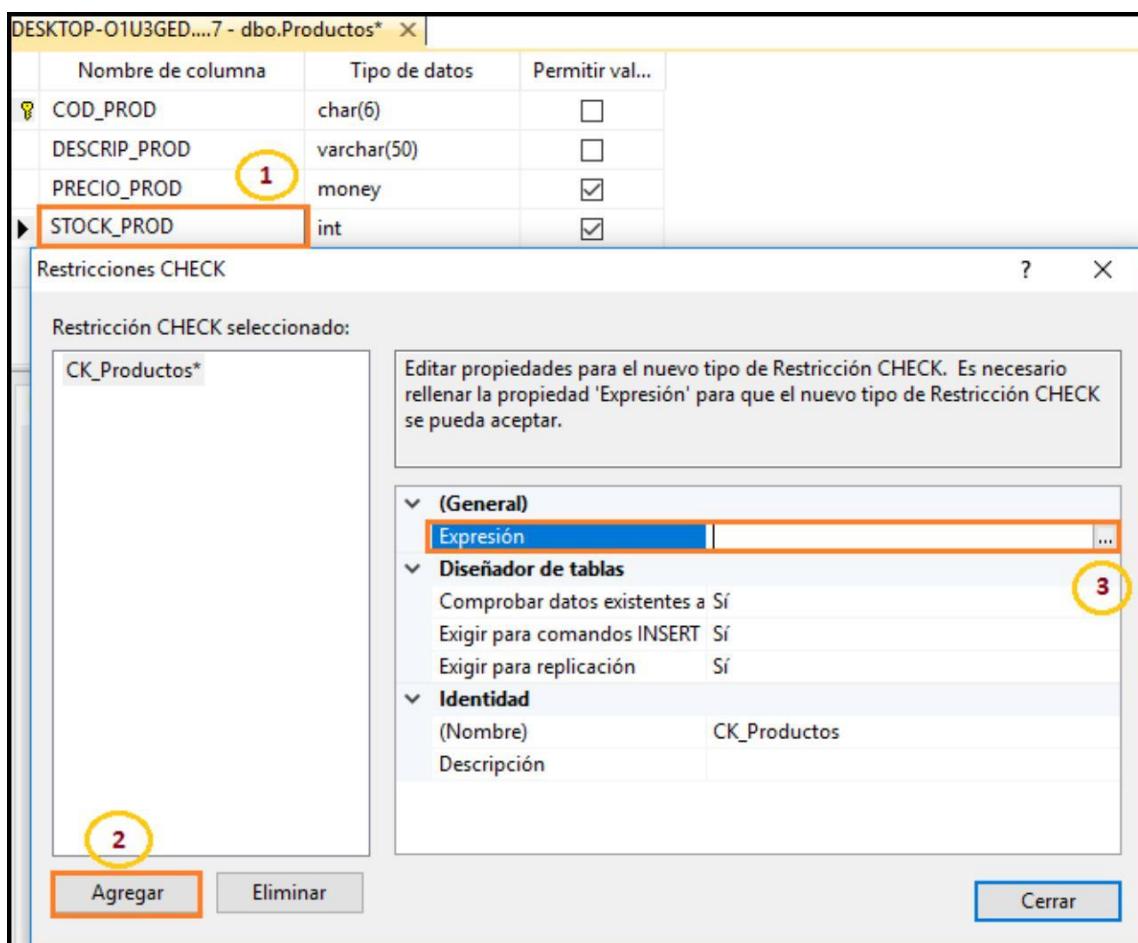
Propiedades de columna

(General)

(Nombre)	PRECIO_PROD
Permitir valores NULL	Sí
Tipo de datos	money
Valor o enlace predeterminado	0

CHECK: Garantiza el ingreso de datos válidos.

Ejemplo: Asignar al campo **STOCK_PROD** que valide mayores que 0.



Las restricciones aparecerán en la siguiente carpeta:



Ahora creamos la tabla **Proveedores**, tal como se muestra a continuación:

Nombre de columna	Tipo de datos	Permitir val...
Ruc_Prove	char(11)	<input type="checkbox"/>
Razon_social	varchar(50)	<input type="checkbox"/>
Direccion	varchar(50)	<input checked="" type="checkbox"/>
Telefono	varchar(12)	<input type="checkbox"/>
PagWeb	varchar(30)	<input checked="" type="checkbox"/>

7.1.1.5 Creando una integridad referencial (Clave foránea)

Para aplicar una integridad referencial entre 2 tablas, primero debemos tener en cuenta lo siguiente:

Las tablas a relacionarse deben, contar con un campo en común.

Una de las tablas debe tener una clave principal (Primary Key) y la otra, una clave foránea (Foreign key)

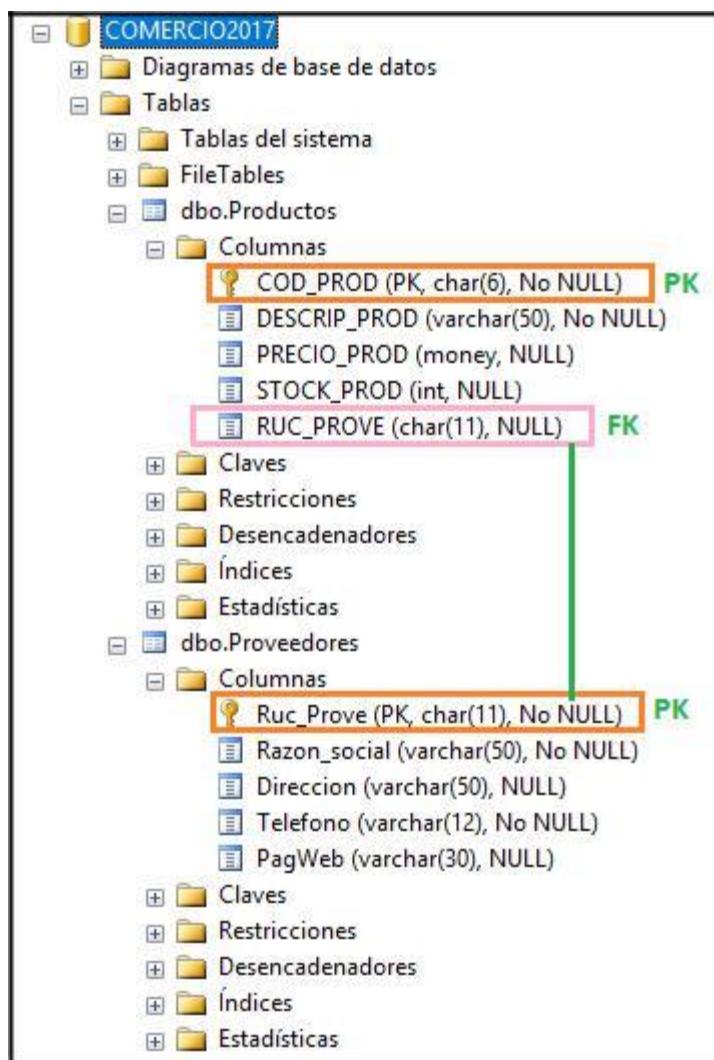
En nuestro caso, relacionaremos como ejemplo la tabla **Productos** con **Proveedores**.

Pasos:

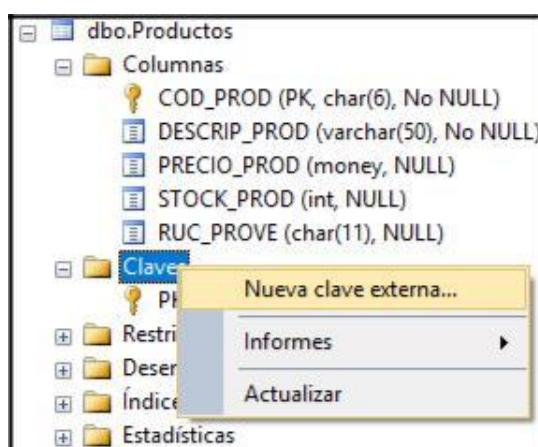
Agregamos un campo foráneo a la tabla Productos **Ruc_Prove**, de tipo char y longitud 11. Esto con la finalidad de relacionar ambas tablas.

Nombre de columna	Tipo de datos	Permitir val...
COD_PROD	char(6)	<input type="checkbox"/>
DESCRIP_PROD	varchar(50)	<input type="checkbox"/>
PRECIO_PROD	money	<input checked="" type="checkbox"/>
STOCK_PROD	int	<input checked="" type="checkbox"/>
RUC_PROVE	char(11)	<input checked="" type="checkbox"/>

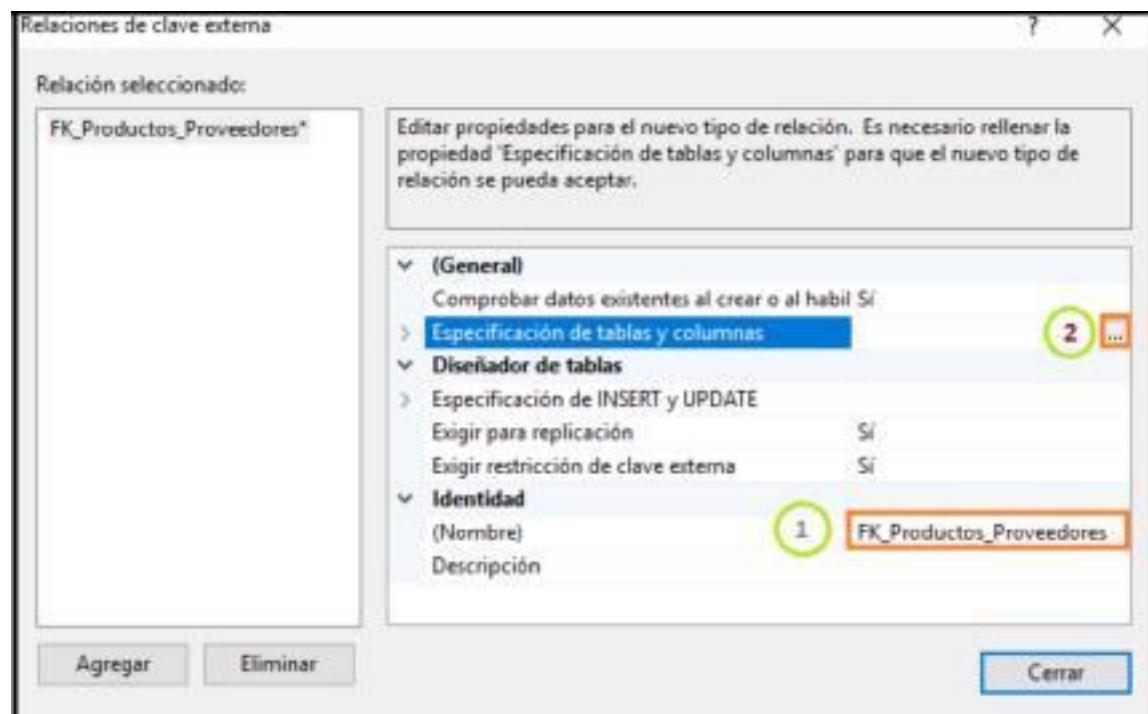
Ahora las tablas se muestran con la siguiente apariencia:



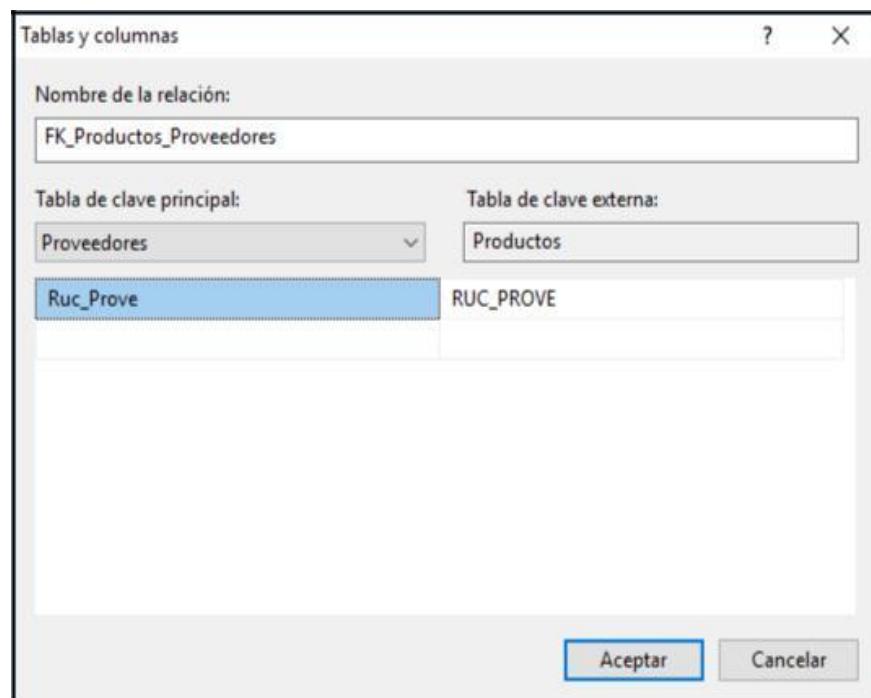
Las tablas, no están relacionadas aún. Para ello, debe hacer clic secundario sobre la carpeta **claves** de la tabla **productos** y elegir la opción **nueva clave externa**.



Al hacer click en la opción **Nueva clave externa** muestra la siguiente ventana, donde debe cambiar el nombre de la **Identidad** y dar clic en le botón de tres puntos (...)

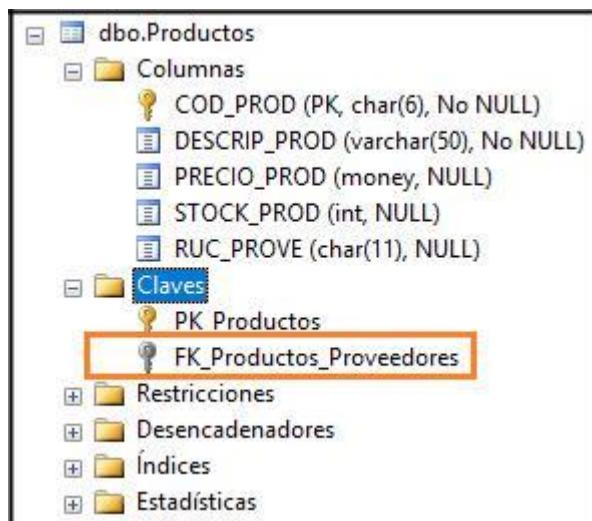


En seguida aparece otra ventana, donde debe especificar la relación de las 2 tablas. Quedará así:

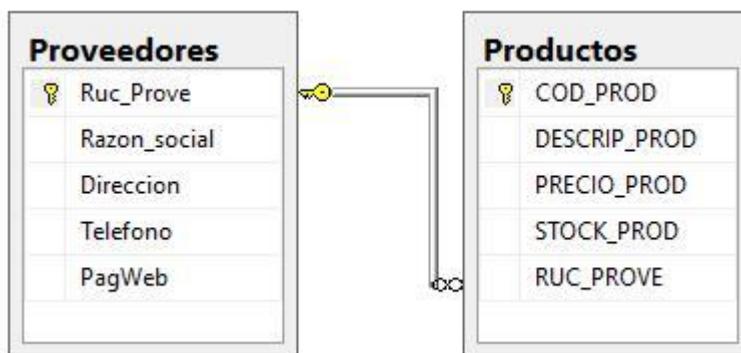


Finalmente, pulse el botón **Aceptar** y **Cerrar**.

Actualice la carpeta **Claves** y se mostrará la llave foránea.



El resultado de la relación es la siguiente:



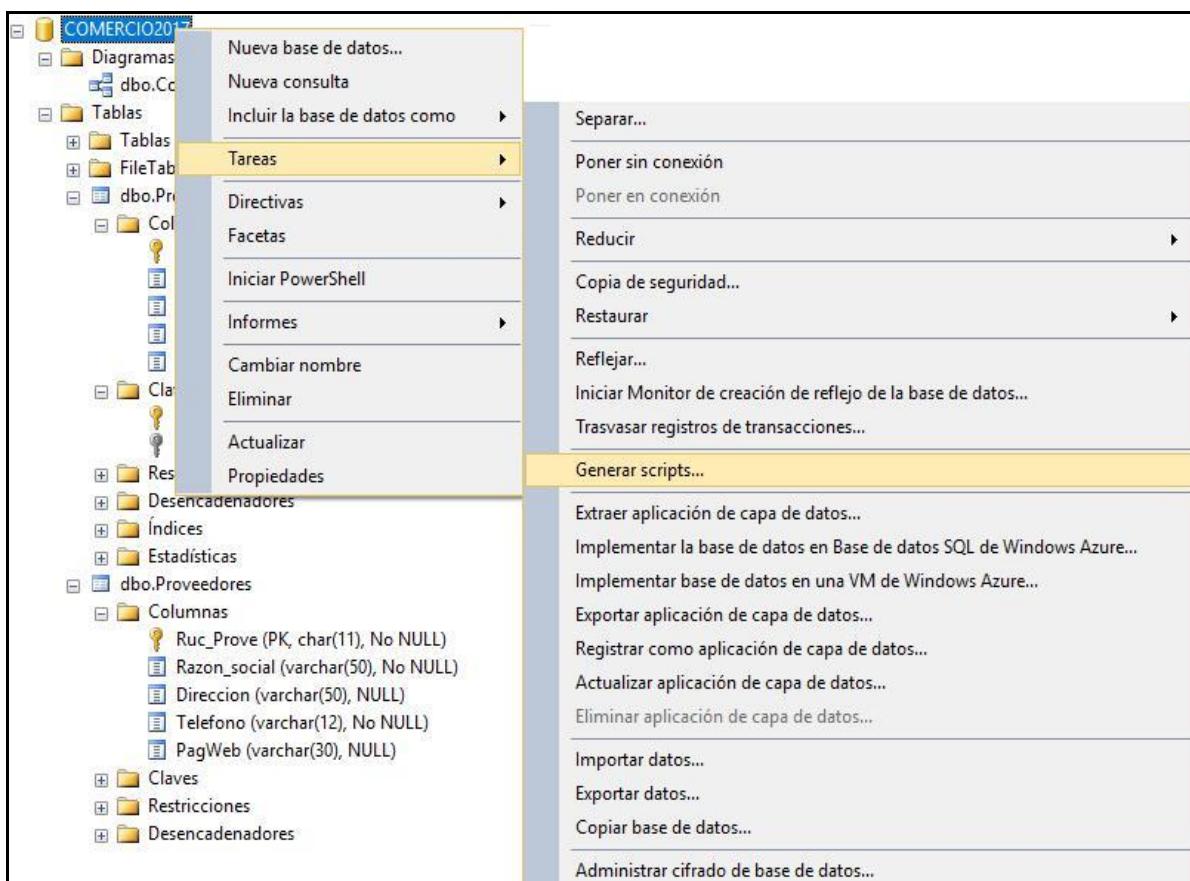
7.1.1.6 Generar el Script Transact/SQL

Para crear scripts de Transact-SQL puede utilizar el **Asistente para generar scripts de SQL Server** o el Explorador de objetos.

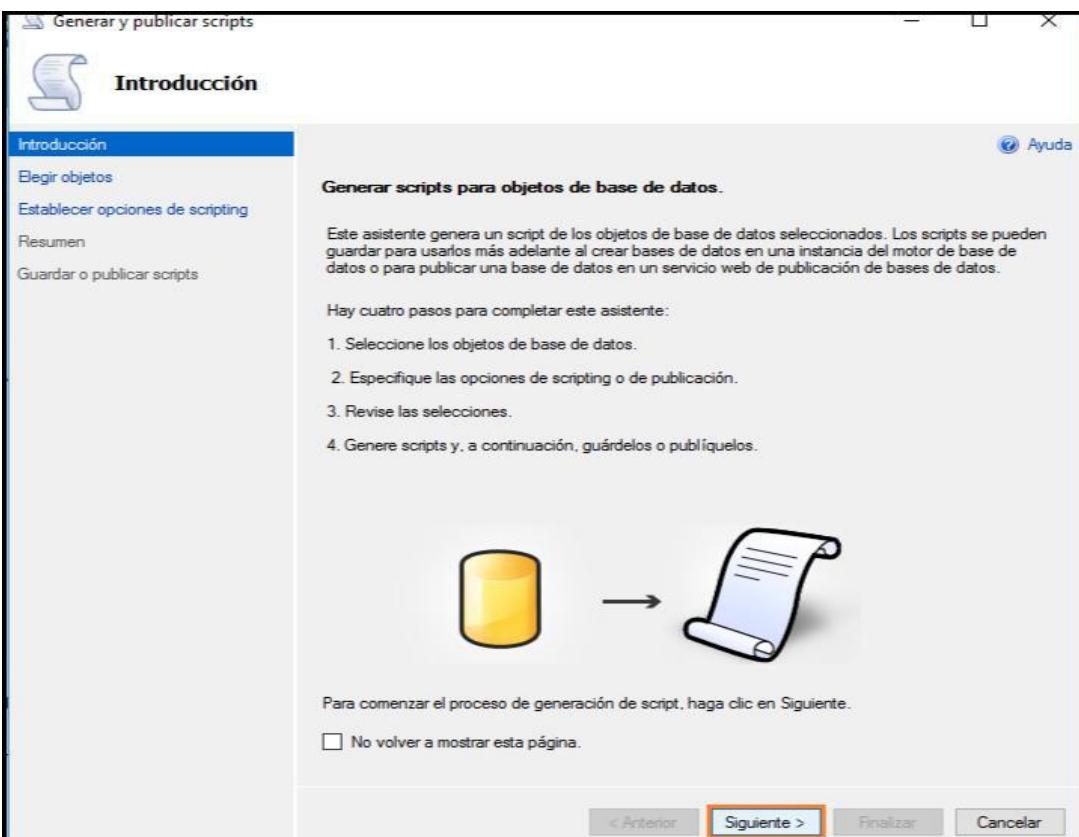
Para abrir el Asistente para generar scripts, previamente asegúrese de guardar todos los cambios en la creación de tablas:

En el Explorador de objetos, expanda **Bases de datos**, haga clic con el botón secundario en la base de datos **COMERCIO2017**, seleccione **Tareas** y, a continuación, haga clic en **Generar scripts**.

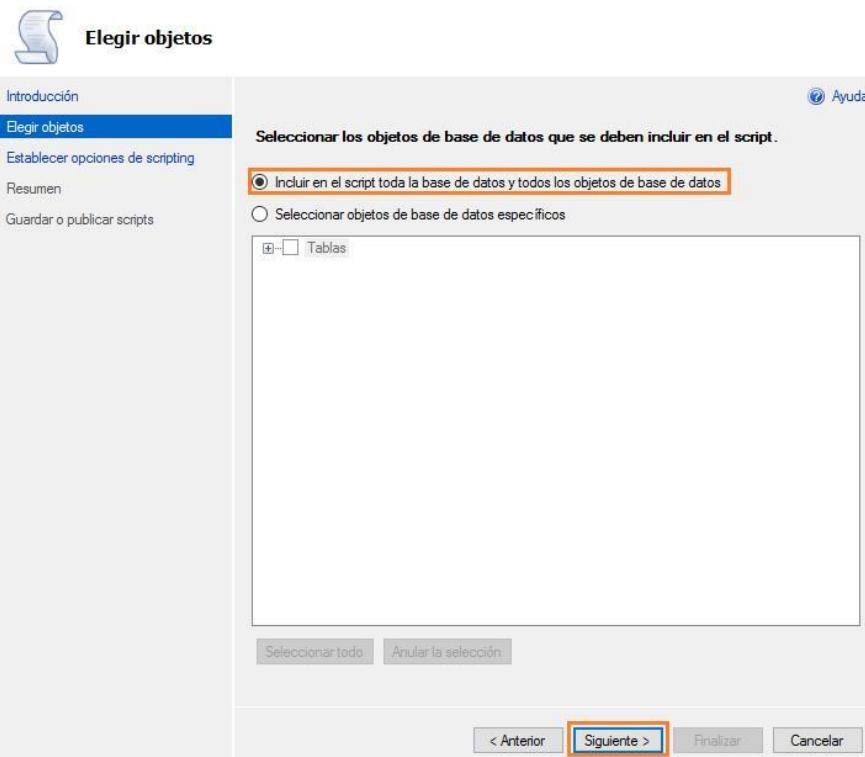
Siga las instrucciones del asistente para incluir los objetos de la base de datos en la script.



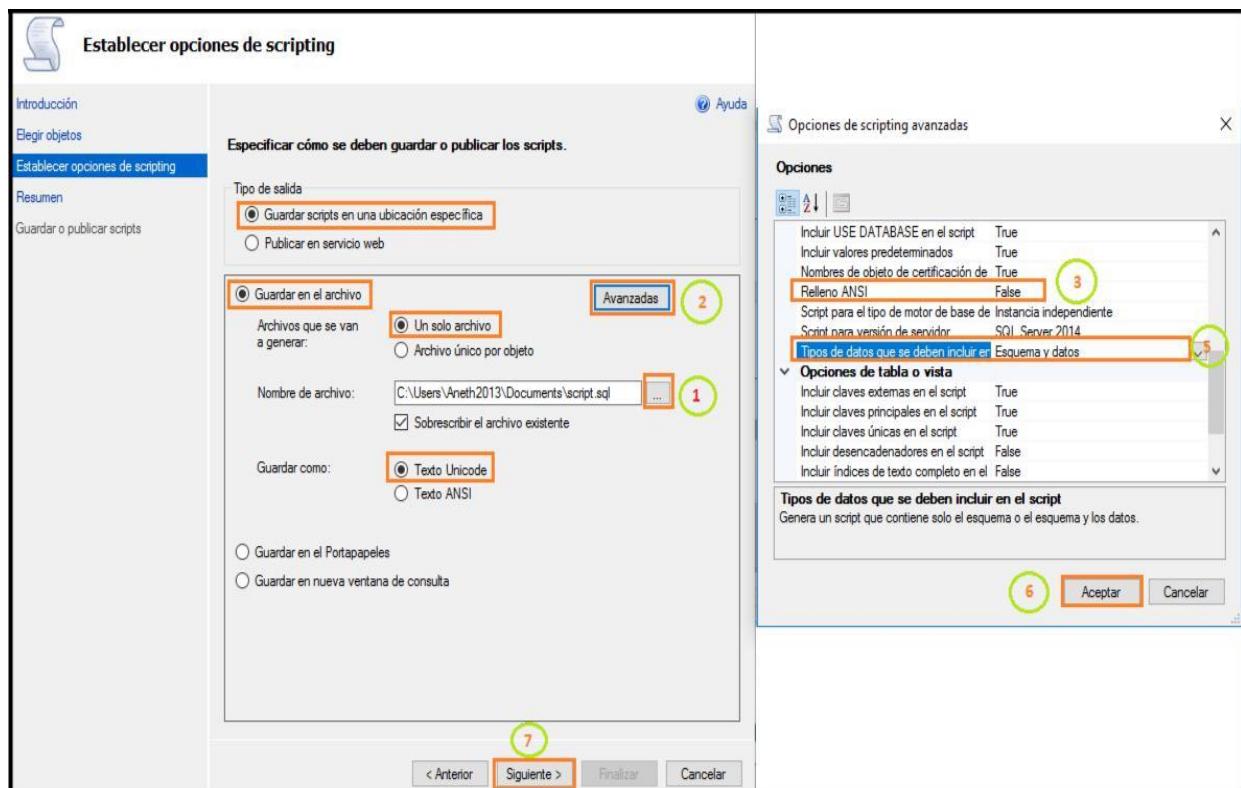
c. En la siguiente ventana haga clic en el botón **Siguiente**



En la siguiente ventana es para seleccionar todos o algunos objetos de la cual se generará el Script. Para nuestro caso dejaremos, tal como se muestra a continuación.

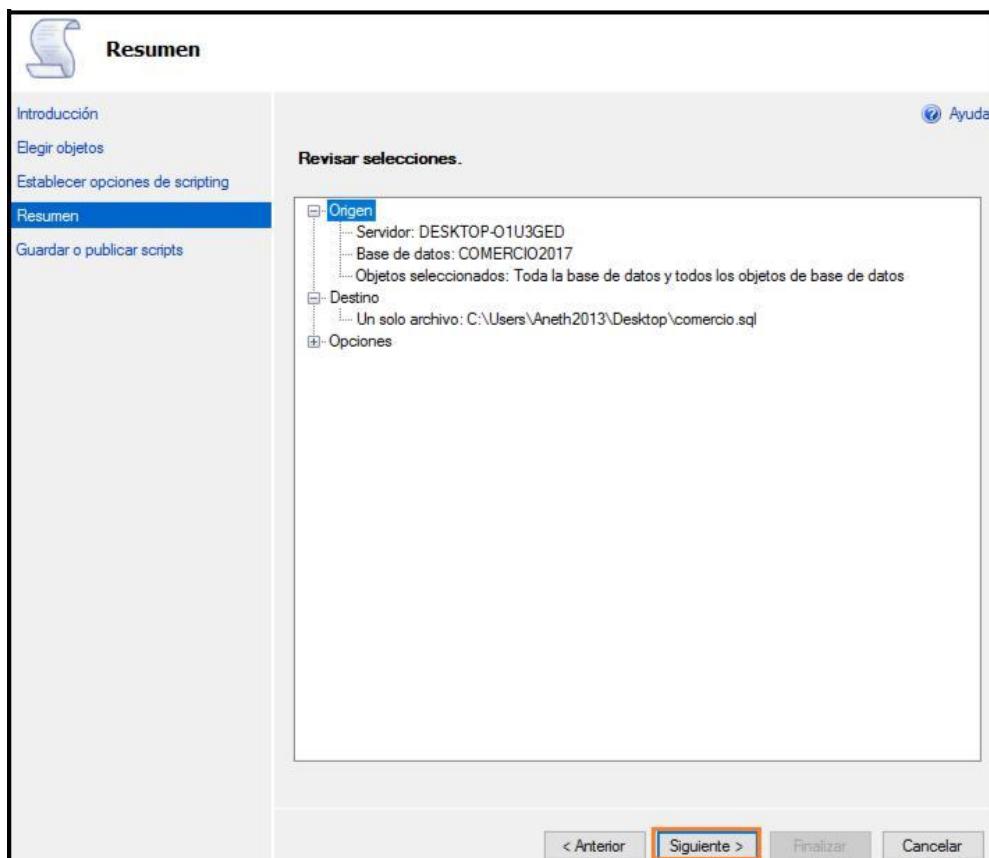


Se mostrará la ventana **Establecer opciones de scripting**, en la cual debe realizar lo siguiente:

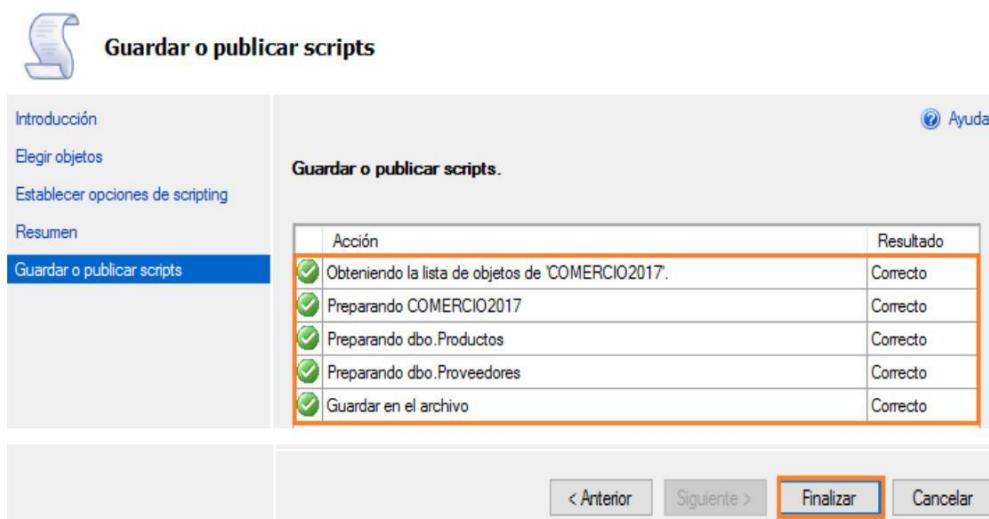


Nota: En la ventana anterior en número 1 (**Nombre de archivo**), debe especificar la ubicación y el nombre del archivo a generar. En el número 5, se está seleccionando la opción Esquema y datos con la finalidad de generar la estructura de los objetos; así como los datos. Para este caso el nombre del archivo será **BD_COMERCIO** y la ubicación **Escritorio**.

En seguida se muestra la ventana **Resumen**, donde nos detalle de cómo se va a generar el Script.



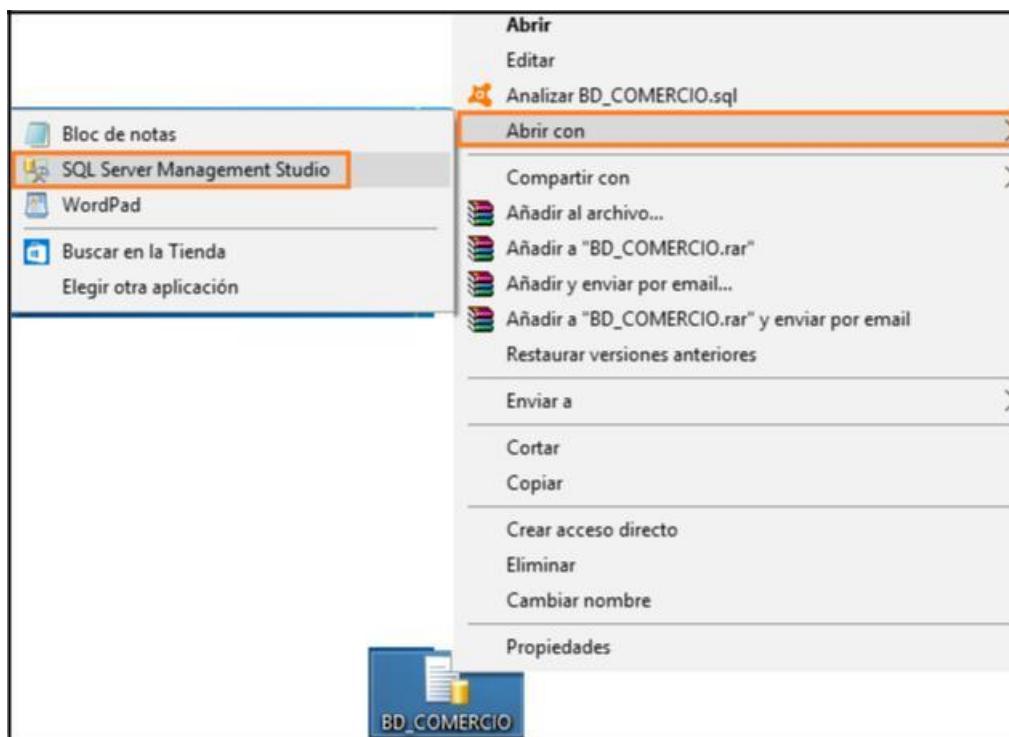
g. En seguida comienza a generarse el Script, tal como muestra:



Al Finalizar, vamos a verificar el script que se encuentra en el **Escritorio**.



Vamos a proceder abrir.



Finalmente se muestra el Script en SQL Server 2014

```

USE [master]
GO
/******** Object: Database [COMERCIO2017] Script Date: 13/04/2017 17:54:45 *****/
CREATE DATABASE [COMERCIO2017]
CONTAINMENT = NONE
ON PRIMARY
( NAME = N'COMERCIO2017', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\COMERCIO2017.mdf' ,
SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
( NAME = N'COMERCIO2017_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\COMERCIO2017_log.ldf' ,
SIZE = 2048KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
CREATE TABLE [dbo].[Productos](
    [COD_PROD] [char](6) NOT NULL,
    [DESCRIP_PROD] [varchar](50) NOT NULL,
    [PRECIO_PROD] [money] NULL,
    [STOCK_PROD] [int] NULL,
    [RUC_PROVE] [char](11) NULL,
    CONSTRAINT [PK_Productos] PRIMARY KEY CLUSTERED
    (
        [COD_PROD] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
/******** Object: Table [dbo].[Proveedores] Script Date: 13/04/2017 17:54:45 *****/
CREATE TABLE [dbo].[Proveedores](
    [RUC_Prove] [char](11) NOT NULL,
    [Razon_social] [varchar](50) NOT NULL,
    [Direccion] [varchar](50) NULL,
    [Telefono] [varchar](12) NOT NULL,
    [PagWeb] [varchar](30) NULL,
    CONSTRAINT [PK_Proveedores] PRIMARY KEY CLUSTERED
    (
        [RUC_Prove] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

```

Hacer notar al estudiante que, al código generado por SQLServer hay que ponerle la validación a la base de datos IF **DB_ID** y para las tablas IF **EXISTS**, porque de correr nuevamente el script generado puede provocar error. Por ejemplo:

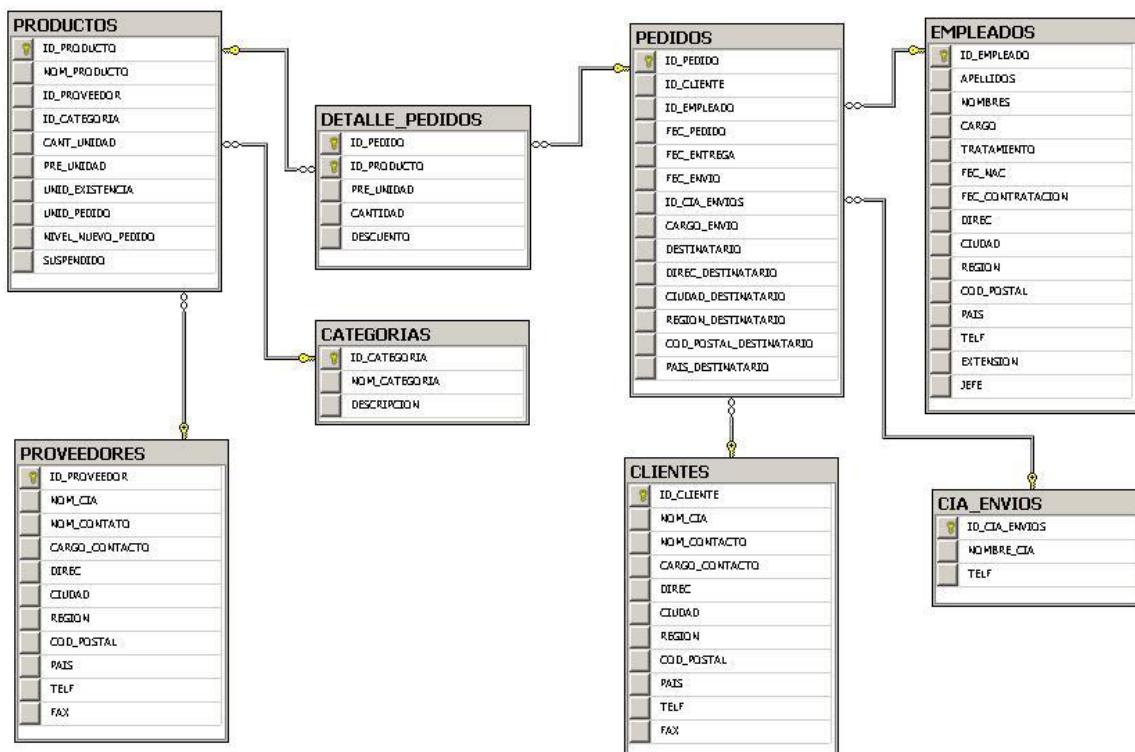
```
IF DB_ID('COMERCIO2017') IS NOT NULL
DROP DATABASE [COMERCIO2017]
GO

IF EXISTS (SELECT * FROM
sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Productos]') AND type in (N'U'))
DROP TABLE [dbo].[Productos]
GO
```

Nota: Puede proceder a comprobar dicho código, ejecutándolo.

Propuesto:

A continuación, se muestra la base de datos **NegociosANE** completa, genere este diagrama usando el diagramador de bases de datos de SQL Server 2014.



Resumen

El diagrama de base de datos de SQL Server 2014, nos permite hacer una simulación de las tablas que se tiene en una base de datos. Además, podemos ver las relaciones que existen entre las tablas.

Si desea conocer otras herramientas diagramador para dicha tarea, puede consultar las siguientes páginas:

https://es.wikipedia.org/wiki/Herramienta_CASE
Herramienta Case

https://en.wikipedia.org/wiki/Erwin_Data_Modeler
Erwin Data Modeler

“Donde hay educación no hay distinción de clases”

CONFUCIO