



Base de Datos

Índice

Presentación	7
Red de contenidos	9
Unidad de Aprendizaje 1	
SISTEMA DE BASE DE DATOS	
1.1 Tema 1 : Introducción y conceptos básicos	11
1.1.1 : Introducción al curso	13
1.1.2 : Surgimiento histórico de las bases de datos	13
1.1.3 : Definición de base de datos	15
1.1.4 : Objetivos de las bases de datos	16
1.2 Tema 2 : Representación de la información	19
1.2.1 : Niveles de abstracción referidos a la información	19
1.2.2 : Reglas de negocio	22
1.2.3 : Relaciones de correspondencia	22
1.3 Tema 3 : Arquitectura de un sistema de base de datos	27
1.3.1 : Arquitectura de un sistema de base de datos	27
1.3.2 : Organizaciones de archivos y el nivel interno de la arquitectura	30
1.4 Tema 4 : Introducción a las bases de datos en SQL Server 2014	35
1.4.1 : Lenguaje estructurado de consultas	35
1.4.2 : Historia del lenguaje estructurado	35
1.4.3 : Importancia de la base de datos	36
1.5 Tema 5 : Creación de base de datos en SQL Server 2014	41
1.5.1 : Componentes de una base de datos	41
1.5.2 : Crear, modificar y eliminar una base de datos	42
1.5.3 : Propiedades del archivo de una base de datos	44
Unidad de Aprendizaje 2	
MODELO CONCEPTUAL	55
2.1 Tema 6 : Modelo conceptual	55
2.1.1 : Modelo conceptual de datos	57
2.2 Tema 7 : Diagrama entidad relacional	59
2.2.1 : Modelo entidad relacional	59
2.2.2 : Diagrama entidad relacional	61
2.3 Tema 8 : Tipos de datos	73
2.4 Tema 9 : Sentencias DDL para la tabla de datos	75
2.4.1 : Creación de una tabla de datos CREATE	75
2.4.2 : Modificación de una tabla de datos ALTER	77
2.4.3 : Eliminación de una tabla de datos DROP	78
2.5 Tema 10 : Integridad referencial	79
2.5.1 : Asignación de PRIMARY KEY a la tabla de datos	82
2.5.2 : Asignación de FOREIGN KEY a la tabla de datos	83

2.6 Tema 11 : Implementación de una base de datos en SQL Server 2014	87
2.6.1 : Implementación de una base de datos en SQL Server 2014	87
Unidad de Aprendizaje 3	
MODELO RELACIONAL Y NORMALIZACION	103
3.1 Tema 12 : Normalización parte 1	105
3.1.1 : Primera forma normal	109
3.2 Tema 13 : Normalización parte 2	117
3.2.1 : Segunda forma normal	117
3.2.2 : Tercera forma normal	118
3.2.3 : Ejercicios de normalización	120
3.3 Tema 14 : Normalización parte 3	125
3.3.1 : Detalle del detalle / ítems	125
3.4 Tema 15 : Implementación de detalle de detalle en SQL Server 2014	131
3.4.1 : Obtención del modelo lógico-global de los datos del DER	131
Unidad de Aprendizaje 4	
MANIPULACION DE DATOS	139
4.1 Tema 16 : Restricciones	141
4.1.1 : Restricción DEFAULT	141
4.1.2 : Restricción CHECK	144
4.1.3 : Restricción UNIQUE	150
4.1.4 : Restricción IDENTITY	153
4.2 Tema 17 : Sentencias DML	155
4.2.1 : Inserción de datos INSERT	155
4.2.2 : Inserción de datos UPDATE	160
4.2.3 : Inserción de datos DELETE	160
4.3 Tema 18 : Integración de sentencias SQL	161
Unidad de Aprendizaje 5	
IMPLEMENTACION DE CONSULTAS	187
5.1 Tema 19 : Recuperación de datos	189
5.1.1 : Introducción al lenguaje de consultas SQL.	189
5.1.2 : Uso de la sentencia SELECT, FROM, WHERE, ORDER BY	189
5.1.3 : Manipulación de consultas condicionales. Uso de condicionales: operadores lógicos AND, OR. Operadores de comparación >, <, =, <>, <=, >=. Operador para manejo de cadenas LIKE. Otros operadores: IN, BETWEEN.	193
5.1.4 : Funciones para el manejo de fechas: DAY(), MONTH(), YEAR(), DATEPART().	203
Unidad de Aprendizaje 6	
INTRODUCCION A LA PROGRAMACION TRANSACT-SQL	213
6.1 Tema 20 : Introducción a la programación en SQL Server 2014	215
6.1.1 : Declaración de variables locales	215
6.1.2 : Procedimientos almacenados con una tabla	218
6.1.3 : Aplicación usando procedimientos almacenados con uno y dos Parámetros	221

Unidad de Aprendizaje 7	
CONSULTAS MULTITABLAS	247
7.1 Tema 21 : Uniones internas y externas (INNER JOIN)	249
7.1.1 : Combinaciones internas con Inner Join	250
7.1.2 Combinaciones externas con Left, Right, Full, Cross Join	253
Unidad de Aprendizaje 8	
SUBCONSULTAS, VISTAS Y AGRUPAMIENTO	271
8.1 Tema 22 : Subconsultas	273
8.1.1 : Subconsultas.	273
8.2 Tema 23 : Vistas	275
8.2.1 : Vistas multitabla.	276
8.2.2 Clasificación de las vistas.	278
8.3 Tema 24 : Agrupamiento de datos	281
8.3.1 Empleo de funciones agregadas	282
8.3.2 Empleo de GROUP BY, HAVING	284
8.3.3 Aplicación con procedimiento almacenado	285

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.

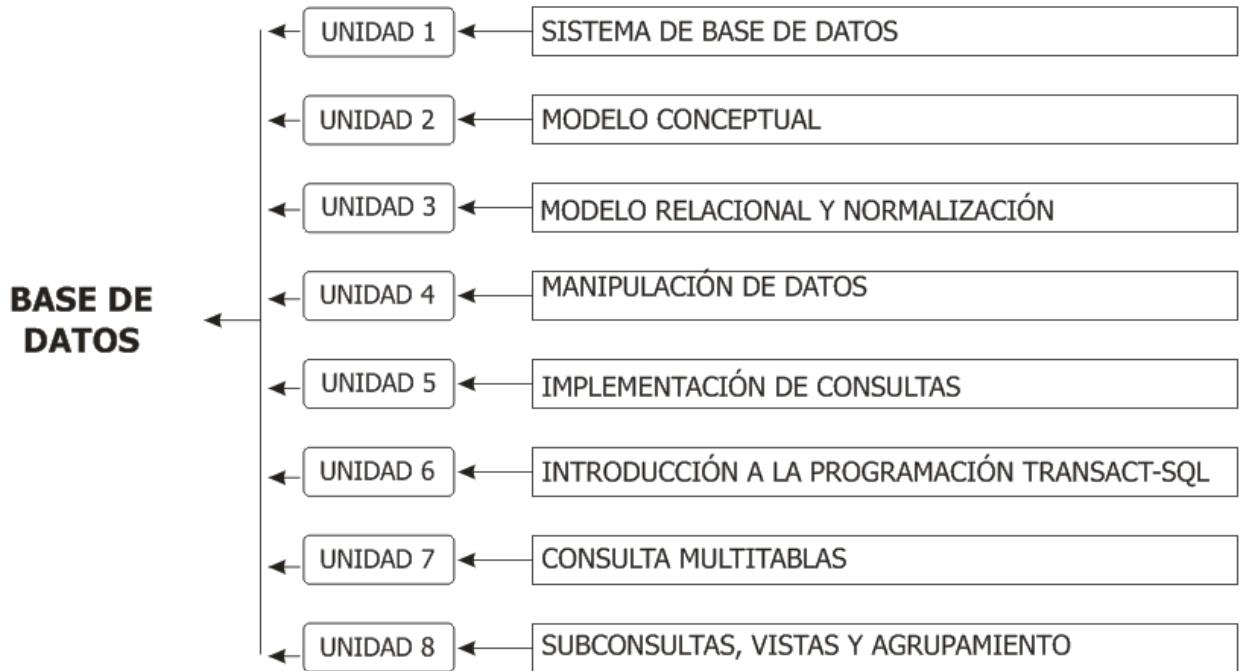
El curso es práctico y consiste en la exposición de técnicas para diseño de base de datos que se complementan con casos reales empleados en las empresas del país. En primer lugar, se inicia con el reconocimiento del proceso de negocio. Luego, se determinan los actores (**entidades**) y la forma cómo 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 se desarrolla sobre la base de las siguientes Unidades de Aprendizaje: Sistema de base de datos; modelo conceptual; modelo relacional y normalización; manipulación de datos, implementación de consultas; introducción a la programación **Transact-SQL**; consulta multitableas; subconsultas; vistas y agrupamiento.

Al final de cada sesión, se añade una Autoevaluación, 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





SISTEMA DE BASE DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno describe los componentes de una base de datos a partir de casos propuestos por el profesor, tomados de situaciones cotidianas y de ejemplos producidos individualmente. Asimismo, determina los elementos de una base de datos y sus interrelaciones.

TEMARIO

1.1 Tema 1 : Introducción y conceptos básicos

- 1.1.1 : Introducción al curso
- 1.1.2 : Surgimiento histórico de las bases de datos
- 1.1.3 : Definición de base de datos
- 1.1.4 : Objetivos de las bases de datos

1.2 Tema 2 : Representación de la información

- 1.2.1 : Niveles de abstracción referidos a la información
- 1.2.2 : Reglas de negocio
- 1.2.3 : Relaciones de correspondencia

1.3 Tema 3 : Arquitectura de un sistema de base de datos

- 1.3.1 : Arquitectura de un sistema de base de datos
- 1.3.2 : Organizaciones de archivos y el nivel interno de la arquitectura

1.4 Tema 4 : Introducción a las bases de datos en SQL Server 2014

- 1.4.1 : Lenguaje estructurado de consultas
- 1.4.2 : Historia del lenguaje estructurado
- 1.4.3 : Importancia de la base de datos

1.5 Tema 5 : Creación de base de datos en SQL Server 2014

- 1.5.1 : Componentes de una base de datos
- 1.5.2 : Crear, modificar y eliminar una base de datos
- 1.5.3 : Propiedades del archivo de una base de datos

ACTIVIDADES PROPUESTAS

- Los alumnos reconocen los principales conceptos de base de datos.
- Los alumnos representan el mundo real en entidades.
- Los alumnos implementan una base de datos en SQL Server 2014.

1.1. INTRODUCCIÓN Y CONCEPTOS BÁSICOS

1.1.1. Introducción al curso

La base de datos hoy en día es considerada como una de las herramientas más ampliamente difundidas en la actual sociedad de la información, es así que son utilizadas como fuentes de recuperación y almacenamiento de información en todos los campos a nivel científico, social, económico, político y cultural.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos registrados en lugares de almacenamiento permanente que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos muchas veces llamados sistemas de gestión de base de datos.

Así mismo, este sistema de Gestión de Bases de datos es un tipo de software muy específico dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la usan; o lo que es lo mismo, una agrupación de programas que sirven para definir, construir y manipular una base de datos, permitiendo así almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

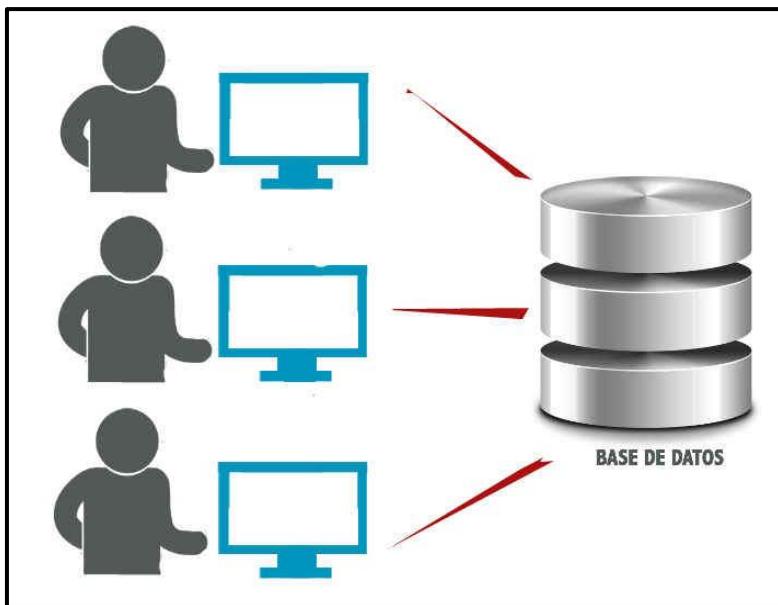


Figura 1: Usuario, Interfaz y Base de Datos
Fuente.- Tomado de <http://itechnode.com/wp-content/uploads/base-de-datos-4.jpg>

1.1.2. Surgimiento histórico de las bases de datos

Al estudiar el desarrollo del procesamiento automatizado de datos, en lo que se refiere al aseguramiento técnico, se habla de diferentes generaciones.

Desde el punto de vista del aseguramiento matemático y, en particular, del aseguramiento de programas, algunos autores reconocen tres generaciones:

- Solución de tareas aisladas
- Integración de tareas aisladas en sistemas particulares
- Integración de sistemas particulares en sistemas automatizados de dirección

Este proceso de integración ocurre paralelamente, aunque no simultáneamente, en dos esferas:

a) Integración de los programas

Los lenguajes de programación integran a las bases de datos con un propósito específico, el cual se manifiesta en sus aplicaciones **Cliente-Servidor**. Es así que en la actualidad los lenguajes de programación presentan un entorno sofisticado de desarrollo que permiten trabajar de manera integrada con las bases no necesariamente de un determinado proveedor de datos.

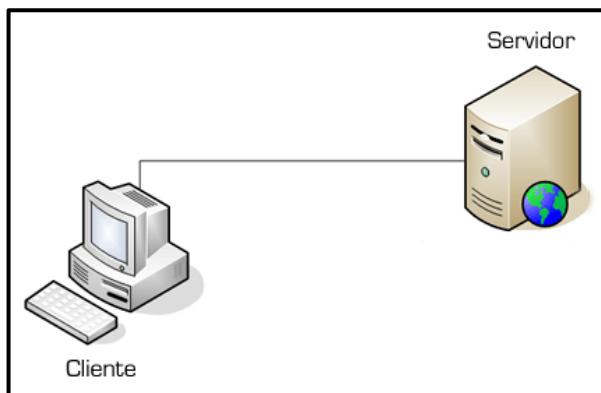


Figura 2: Cliente-Servidor

Fuente.- Tomado de <http://www.coders.me/wp-content/uploads/2008/03/cliente-servidor.gif>

b) Integración de los datos

Se han producido tres categorías de técnicas para su manipulación:

- **Sistemas orientados a los dispositivos**

Programas y archivos que son diseñados y empleados de acuerdo con las características físicas de la unidad central y los periféricos. Cada programa está altamente interconectado con sus archivos, por lo que la integración de datos de diferentes sistemas es imposible prácticamente.

- **Sistemas orientados a los archivos**

La lógica de los programas depende de las técnicas de organización de los archivos (secuencial, directo, etc.). Cada usuario organiza su archivo de acuerdo con sus necesidades y las relaciones entre los elementos se establecen a través de los programas de aplicación.

Esta forma de trabajo implica redundancia de datos, que trae aparejada mayor gasto de memoria y complica las operaciones de actualización (modificar un dato donde quiera que aparezca). Esto aumenta el tiempo de tratamiento y atenta contra la integridad de la información.

Cuando se habla de integridad, se está haciendo referencia a que, en todo momento, los datos almacenados estén correctos en correspondencia con la realidad.

Además, en la vida real, se establecen relaciones entre los objetos que son muy difíciles de representar u obtener a partir de sistemas tradicionales de archivos. Por ejemplo, si se tiene información sobre trabajadores y estudiantes de una facultad, las aplicaciones requeridas van a definir la manera de organizar y estructurar los archivos. Si se desean obtener datos como promedio de las calificaciones de cada alumno, listado de estudiantes por grupo, categoría científica y docente de cada grupo, y salario de cada uno, resulta adecuado establecer dos archivos: uno de profesores y otro de estudiantes.

¿Qué ocurre si se quieren establecer vínculos entre los profesores y estudiantes? Por ejemplo, si se desea obtener lo siguiente:

- Los estudiantes de un profesor
- Los profesores de un estudiante

De esta manera, se estructuraría un archivo de profesores y estudiantes que resolvería algunas demandas, pero sería ineficiente para otras.

Entonces, ¿es posible representar de manera eficiente, utilizando los medios de cómputo, los casos o procesos de la realidad objetiva (procesos de negocio), aunque sea, por supuesto, de forma esquemática, pero en la que se establezcan determinados vínculos entre los elementos u objetos que forman parte de esos procesos o casos?

Se observa que es posible hacerlo a través de la utilización de bases de datos (**BD**) y de los sistemas de gestión de bases de datos (**SGBD**) que dirigen su manipulación.

- **Sistemas orientados a base de datos**

Un Sistema de Gestión de Bases de Datos consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a los mismos.

1.1.3. Definición de base de datos

Veamos algunas definiciones:

“Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular”.

“Se le llama base de datos a los bancos de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto”.

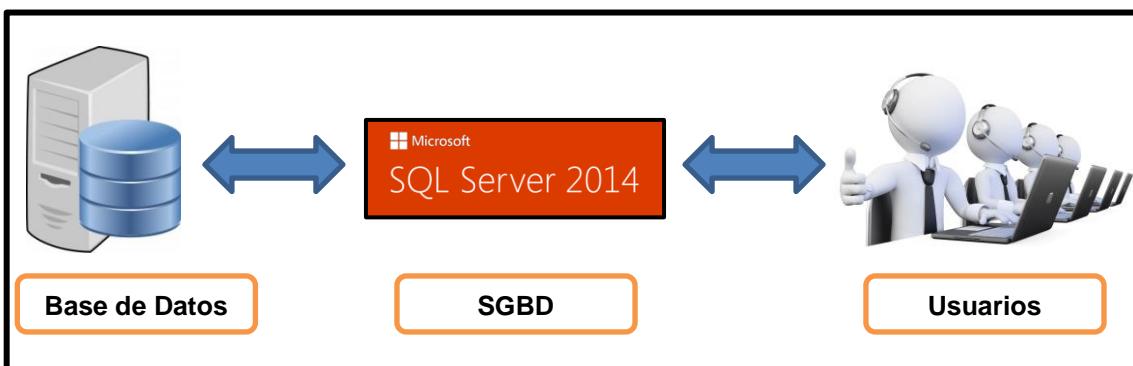
“Una base de datos es un “almacén” que nos permite guardar grandes cantidades de información de forma organizada con el objeto de encontrar y utilizar fácilmente dicha información”.

“Una base de datos correctamente diseñada permite obtener acceso a información exacta y actualizada. Puesto que un diseño correcto es esencial para lograr los objetivos fijados para la base de datos, parece lógico emplear el tiempo que sea necesario en aprender los principios de un buen diseño ya que, en ese caso, es mucho más probable que la base de datos termine adaptándose a sus necesidades y pueda modificarse fácilmente”. Microsoft.

El software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista y a la vez se denomina sistema de gestión de bases de datos (**SGBD**).

Es importante diferenciar los términos base de datos y **SGBD**. El objetivo fundamental de un **SGBD** consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, de tal forma que no le sea necesario conocer el modo de almacenamiento de los datos en la computadora ni el método de acceso empleado.

Los programas de aplicación operan sobre los datos almacenados en la base, utilizando las facilidades que brindan los **SGBD**, los que, en la mayoría de los casos, poseen lenguajes especiales de manipulación de la información que facilitan el trabajo de los usuarios.



1.1.4. Objetivos de las base de datos

Existen muchas formas de organizar las bases de datos, pero hay un conjunto de objetivos generales que deben cumplir todos los **SGBD**, de modo que faciliten el proceso de diseño de aplicaciones y que los tratamientos sean más eficientes y rápidos, dando la mayor flexibilidad posible a los usuarios.

Los objetivos fundamentales de los **SGBD** son los siguientes:

1.1.4.1. Independencia de los datos y los programas de aplicación

Se ha observado que, con archivos tradicionales, la lógica de la aplicación contempla la organización de los archivos y el método de acceso. Por ejemplo, si por razones de eficiencia se utiliza un archivo secuencial indexado, el programa de aplicación debe considerar la existencia de los índices y la secuencia del archivo. Entonces, es imposible modificar la estructura de almacenamiento o la estrategia de acceso sin afectar el programa de aplicación (naturalmente, lo que se afecta en el programa son las partes de éste que tratan los archivos, lo que es ajeno al problema real que el programa de aplicación necesita resolver). En un **SGBD**, sería indeseable la existencia de aplicaciones y datos dependientes entre sí, por dos razones fundamentales:

- Diferentes aplicaciones necesitarán diferentes aspectos de los mismos datos (por ejemplo, puede requerirse la representación decimal o binaria).
- Se debe modificar la estructura de almacenamiento o el método de acceso, según los cambios en el caso o proceso de la realidad sin necesidad de modificar los programas de aplicación (también para buscar mayor eficiencia).

La independencia de los datos se define como la inmunidad de las aplicaciones a los cambios en la estructura de almacenamiento y en la estrategia de acceso. Todo esto constituye el objetivo fundamental de los **SGBD**.

1.1.4.2. Minimización de la redundancia

Se ha comprobado cómo, con los archivos tradicionales, se produce redundancia de la información. Uno de los objetivos de los **SGBD** es minimizar la redundancia de los datos. Se dice disminuirla, no eliminarla, pues, aunque se definen las bases de datos

como no redundantes, en realidad sí existe, pero en un grado no significativo que servirá para disminuir el tiempo de acceso a los datos o para simplificar el método de direccionamiento. Lo que se trata de lograr es la eliminación de la redundancia superflua.

1.1.4.3. Integración y sincronización de las bases de datos

La integración consiste en garantizar una respuesta a los requerimientos de diferentes aspectos de los mismos datos por diferentes usuarios, de forma que, aunque el sistema almacene la información con cierta estructura y cierto tipo de representación, debe garantizar entregar al programa de aplicación los datos que solicita y en la forma en que lo solicita.

Está vinculada a la sincronización, que consiste en la necesidad de garantizar el acceso múltiple y simultáneo a la base de datos, de modo que puedan ser compartidos por diferentes usuarios a la vez. Están relacionadas, ya que lo usual es que diferentes usuarios trabajen con diferentes enfoques y requieran los mismos datos, pero desde diferentes puntos de vista.

1.1.4.4. Integridad de los datos

Consiste en garantizar la no contradicción entre los datos almacenados, de modo que, en cualquier momento del tiempo, sean correctos, es decir, que no se detecte inconsistencia entre los mismos. Está relacionada con la minimización de la redundancia, ya que es más fácil garantizar la integridad si se elimina ésta.

1.1.4.5. Seguridad y recuperación

Seguridad (también llamada protección): garantiza el acceso autorizado a los datos, la forma de interrumpir cualquier intento de acceso no autorizado, ya sea por error del usuario o por mala intención.

Recuperación: permite que el sistema de bases de datos disponga de métodos que garanticen la restauración de las bases de datos al producirse alguna falla técnica, interrupción de la energía eléctrica, etc.

1.1.4.6. Facilidad de manipulación de la información

Los usuarios de una base de datos pueden acceder a ella con solicitudes para resolver muchos problemas diferentes. El **SGBD** debe contar con la capacidad de una búsqueda rápida por diferentes criterios, debe permitir que los usuarios planteen sus demandas de una forma simple, aislando de las complejidades del tratamiento de los archivos y del direccionamiento de los datos. Los **SGBD** actuales brindan lenguajes de alto nivel, con diferentes grados de facilidad para el usuario no programador, que garantizan este objetivo, los llamados sublenguajes de datos.

1.1.4.7. Control centralizado

Uno de los objetivos más importantes de los **SGBD** es garantizar el control centralizado de la información. Permite comprobar, de manera sistemática y única, los datos que se almacenan en la base de datos, así como el acceso a ella.

Lo anterior implica que debe existir una persona o un conjunto, que tenga la responsabilidad de los datos operacionales: el administrador de la base de datos puede considerarse parte integrante del **SGBD**.

1.2. Representación de la información

En Base de Datos existen muchos términos que seguro lo tenemos como saber previo tal es así que debemos distinguir entre los términos dato e información para poder entender los demás que usaremos en este curso.

Tomemos en cuenta que un **dato** es cualquier valor que puede representar algo, así como un número, imagen, sonido, símbolo, nombre, etc.; su característica principal es que no permite tomar ninguna decisión, ya que son simplemente datos que no tienen un significado concreto. Estos datos deben procesarse para dotarlos de significado y convertirlos en algo que nos pueda servir como **información**.

En cualquier proceso y construcción de un sistema informático, el diseño de la base de datos ocupa un lugar importante, a tal punto que esta puede verse como un proceso relativamente independiente dentro del diseño del sistema y compuesto por una serie de etapas. Es por ello que resulta de interés el estudio de los problemas relacionados con el diseño de las bases de datos y el modelamiento de la información.

1.2.1. Niveles de abstracción referidos a la información

Cuando se habla de información, se hace referencia, de forma general, a tres niveles diferentes de abstracción, tendiéndose a saltar de uno a otro sin establecer una advertencia previa.

1° NIVEL: MUNDO REAL

En este nivel encontramos a las entidades u objetos, que son elementos que componen nuestro universo y son considerados como elementos que existen y además están bien diferenciados entre sí, estos poseen propiedades o características que son fácilmente detectables. Por ejemplo:

	Número de placa
	Color del auto
	Número de puertas
	Marca del auto
	Modelo del auto

Así mismo, podríamos nombrar a un teléfono móvil, alumno, empleado, producto, etc, pero no siempre será algo tangible, también podríamos nombrar a elementos que no son fácilmente palpables como un suceso o una transacción. Por ejemplo:

	Hora de inicio del siniestro
	Número de personas afectadas
	Distrito donde se occasionó el siniestro
	Hora final del siniestro
	Número de bomberos

La determinación de cierta entidad u objeto correspondiente a un caso o proceso está muy relacionada con el nivel de abstracción en el que se esté realizando el análisis. Así, por ejemplo, si se estudia el comportamiento de un insecto específico en determinadas condiciones climáticas, las propiedades y

relaciones que interesan son de un cierto tipo; sin embargo, si se estuviera realizando un estudio sobre las diferentes especies de insectos, serían otros los objetos por definir, así como las propiedades que los caracterizarían y las relaciones que se establecerían. Si se estuviera analizando todo el reino animal, serían también otros los objetos a definir, con sus características y propiedades.

2º NIVEL: DOMINIO DE LAS IDEAS

En este nivel se decide la información que debe existir en la base de datos sobre un caso o proceso del mundo real, es decir, qué debe almacenarse.

Aquí, es donde realmente se define el contenido informativo que representará al caso, proceso o ente de la realidad objetiva que se está analizando. De modo que, en este nivel, se definen qué objetos y propiedades son representativas, ya que sobre estos es necesario almacenar información.

En este nivel, se trabaja con los conceptos más importantes del modelo de datos, que establecen la relación entre el mundo real y la información almacenada físicamente en la base de datos, como un dato o registro que definiremos a continuación:

- Campo o Atributo:** Es la unidad menor de información de un determinado objeto que se almacena en una base de datos y representa una propiedad de un objeto; por ejemplo, el color, tamaño o algo particular del objeto.

Sin embargo, hay que distinguir entre el nombre y el valor del atributo, ya que un nombre de atributo puede tomar diferentes valores sobre un cierto conjunto que se denomina dominio. A un valor de un atributo se le denomina ocurrencia del atributo. Por ejemplo:

	Atributo	Dominio	Ocurrencia
	Color	{Azul, Rojo, Gris}	Gris
	Marca	{Toyota, Kia, Chery}	Kia
	Modelo	{Rav4, Sportage, Tiggo}	Sportage

Ahora bien, una colección identificable de atributos es un registro y representa un objeto con sus propiedades. Una vez más, es imprescindible distinguir entre nombre y ocurrencia de artículo.

Una ocurrencia de artículo o tupla consiste en un grupo de ocurrencias de campos relacionados, representando una asociación entre ellos. Por ejemplo, tenemos un artículo correspondiente al objeto profesor, en un caso o proceso de la realidad que pretenda representar el comportamiento de una facultad. El nombre o tipo de artículo puede ser **PROFESOR**, que esté formado por los siguientes tipos de campos o atributos:

	Atributo	Descripción
	COD_PROF	Código único del profesor.
	NOM_PROF	Nombre completo del profesor.
	CAT_PROF	Categoría asignada al profesor.
	SUE_PROF	Sueldo asignado al profesor.

Una ocurrencia de este artículo puede ser:

PJPEREZ	JUAN PEREZ ROJAS	PARCIAL	S/. 2500.00
---------	------------------	---------	-------------

- **Archivo o archivos:** Son un conjunto de ocurrencias de un mismo tipo de artículo. En la práctica, llama la atención las colecciones o conjuntos de objetos similares. Además, es necesario almacenar la información de las mismas propiedades para cada uno de ellos; por ejemplo, el conjunto de profesores de la facultad.

Entonces, una base de datos contendrá muchas ocurrencias de cada uno de los tipos de artículos, lo que implica que la base de datos, por supuesto, también contendrá muchas ocurrencias de los distintos tipos de atributos.

Uno de los momentos cruciales en el diseño de un caso de la realidad objetiva que se concreta en una base de datos es, precisamente, la selección de los conjuntos de objetos y sus propiedades.

Además, existe otro concepto muy importante en este nivel que es el concepto de **llave o clave**. Se denomina a este último como un atributo o conjunto de atributos de un artículo que define que cada ocurrencia de artículo de la base de datos sea única. En principio, cada artículo tiene una llave, ya que se tiene como hipótesis que cada elemento u ocurrencia del artículo es diferente de las demás. Por ejemplo, el **código del profesor** puede constituir la llave del artículo **profesor**.

3º NIVEL: DATOS

El tercer nivel corresponde a los datos propiamente dichos, los cuales son representados mediante cadenas de caracteres o de bits.

En este nivel es necesario tener en cuenta la diferencia entre tipo de dato y valor del dato. El tipo de dato corresponde a un atributo o tipo de atributo, que está asociado a un tipo de artículo correspondiente, mientras que, el valor corresponde a una ocurrencia del atributo.

Sin embargo, una colección de bits o caracteres que representa un único valor de datos y que puede existir independientemente de cualquier información que se almacena adquiere significado Solo cuando se le asocia a un tipo de atributo. Se puede, por ejemplo, almacenar permanentemente los valores ROJO, AZUL, VERDE, etc. y asociarlos en un momento determinado a un tipo de atributo a través de los valores que toma, representando una ocurrencia en una tupla. Veamos la integración de todos los niveles expresados en la entidad Auto:



1.2.2. Reglas de negocio

Toda organización funciona siguiendo múltiples reglas o también llamada políticas, estas pueden ser explícitas o tácitas las cuales se encuentran integradas en los procesos, aplicaciones informáticas, documentos, etc.

Así mismo, se podría decir que describe políticas, normas, operaciones, definiciones y restricciones presentes en una organización y que son de vital importancia para alcanzar los objetivos. Mencionaremos algunos ejemplos:

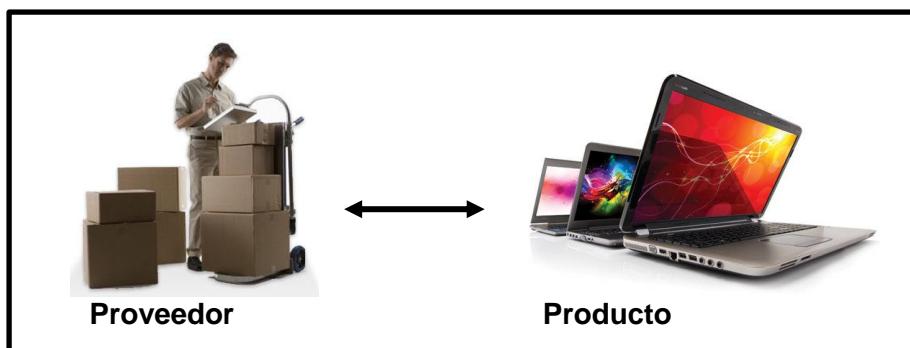
- Un docente dictará únicamente en una sede.
- Un cliente solo puede adquirir 3 promociones.
- Después de 15' se considera tardanza.

Por otra parte, una aplicación informática permite reflejar parte del funcionamiento del mundo real, haciendo que un proceso sea manejado por un usuario de manera correcta; para que esto ocurra debemos aplicar restricciones, de modo que se prevea acciones de manera correcta. Para un sistema de base de datos podríamos nombrar las siguientes reglas de negocio:

- Controlar el saldo negativo de un cliente.
- Controlar el stock negativo de un producto.
- Crear boletas de ventas a clientes que no se encuentran registrados.
- Controlar el número de intentos al ingresar un usuario y una clave.

1.2.3. Relaciones de correspondencia

Es importante notar que, en general, habrá asociaciones o relaciones enlazando las entidades básicas. Estos enlaces se pueden establecer entre diferentes objetos o tipos de artículos o entre un mismo tipo de artículo. Por ejemplo:



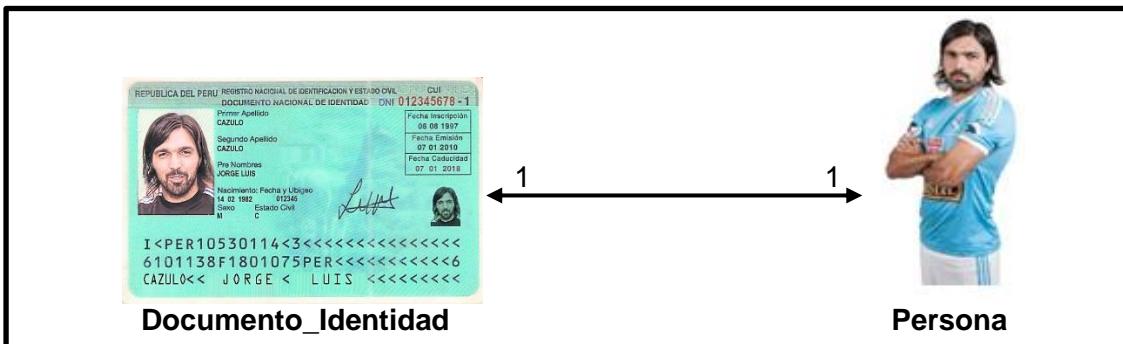
Se puede tener una relación entre dos tipos de objetos: **PROVEEDOR** y **PRODUCTO**, de modo que un proveedor puede suministrar muchos productos y que un producto puede ser suministrado por muchos proveedores y se conoce, además, la **CANTIDAD** de cada producto que suministra un proveedor dado.

Es necesario profundizar acerca de los diferentes tipos de relaciones que pueden ocurrir en la práctica y establecer la correspondencia que existe entre los datos. Esta relación puede ser simple o compleja.

1.2.3.1 Relación Simple

Se le llama así, a la relación cuya correspondencia es catalogada como biunívoca también llamada relación de **UNO A UNO**; entre las ocurrencias de las entidades.

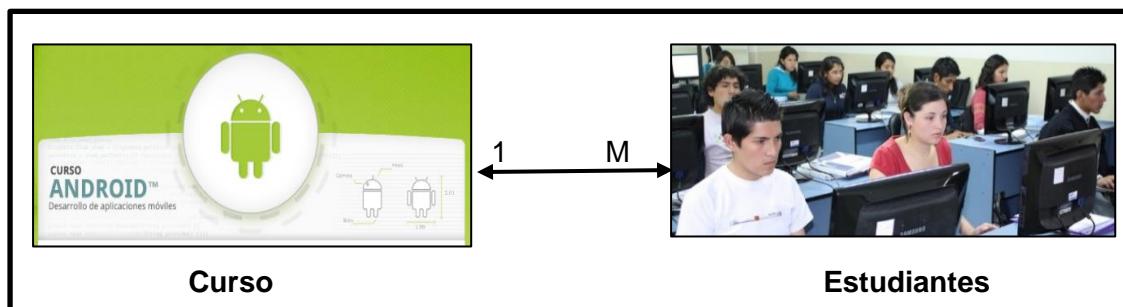
Matemáticamente podríamos decir que se asocia cada uno de los elementos de un conjunto con uno, y solo uno, de los elementos de otro conjunto. Así por ejemplo:



Las entidades **Documento_Identidad** y **Persona** se encuentran asociados, la correspondencia entre ellos es **simple**, por lo tanto podríamos decir que: “A cada persona le corresponde un documento de identidad y viceversa”.

1.2.3.2 Relación Compleja

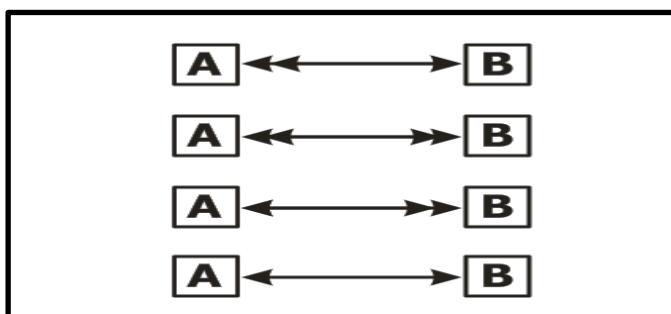
Se le llama así, a la relación cuya correspondencia es catalogada como unívoca y reciproca. Así por ejemplo:



Si las entidades son **Curso** y **Estudiante**, la relación es más complicada, porque en cada curso, están inscritos varios alumnos. La terminología usual expresa que la correspondencia de **estudiante a curso** es simple, ya que cada estudiante es miembro de un único curso, mientras que la correspondencia de **curso a estudiante** es compleja, pues cada curso tiene, por lo general, muchos estudiantes.

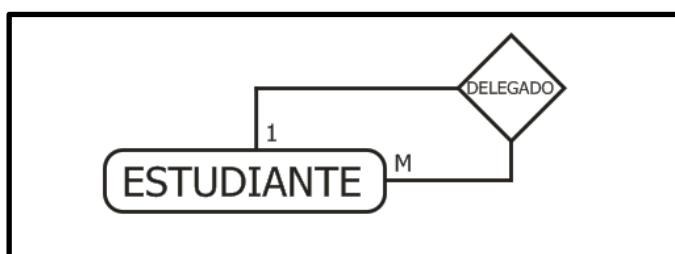
Finalmente, hay cuatro tipos de relaciones posibles entre dos tipos de artículos A y B:

- La correspondencia de A a B puede ser simple y la recíproca compleja.
- La correspondencia de A a B puede ser compleja y la recíproca simple.
- Ambas correspondencias pueden ser complejas o ambas pueden ser simples.



1.2.3.3 Características con respecto a las correspondencias

- Aunque la mayoría de las relaciones asocian dos tipos de entidades, éste no es siempre el caso. Por ejemplo, CURSO_HORARIO_ESTUDIANTE. Esto podría representar el hecho de que un curso se dicta a una cierta hora a un cierto estudiante. Esto no es lo mismo que la combinación CURSO_HORARIO y CURSO_ESTUDIANTE, ya que la información de que "el curso C1 se dicta en el horario H1 al estudiante E1" dice más que la combinación "el curso C1 se dicta en el horario H1" y "el estudiante E1 recibe clases en el horario H1".
- Las relaciones pueden establecerse entre un mismo tipo de entidad. Por ejemplo, una asociación entre un estudiante y otro puede venir dada por el hecho de que un estudiante sea el delegado de otros estudiantes. A este tipo de relación se le llama relación recursiva o simplemente recursividad.



- Es importante señalar que una asociación entre entidades puede ser considerada en sí como una entidad, ya que una relación es concebida como un objeto bien diferenciado sobre el cual se desea almacenar información.
- Entonces, un modelo de datos no es más que la representación de un caso de la realidad objetiva a través de los objetos, sus propiedades y las relaciones que se establecen entre ellos.

1.2.3.4 Ejemplo Integrador

Caso: BIBLIOTECA

En una biblioteca, se desea diseñar la base de datos para el control de los préstamos de libros. De cada libro se conoce el código que lo identifica, su título y la cantidad de páginas que tiene.

Así mismo, un **libro** se clasifica por una materia y por ésta se clasifican muchos libros. De cada **materia** se conoce el código que la identifica y su nombre. Los libros tienen muchos **ejemplares**, pero un ejemplar lo es de un solo libro. De cada ejemplar se sabe su código y su estado de conservación.

Finalmente, un libro se le puede prestar a muchos **usuarios** y a un usuario se le puede prestar muchos ejemplares; del usuario se conoce su DNI, nombre y apellido paterno, su dirección, y su ocupación.

Del caso expuesto, debe tener en cuenta:

- Determine las entidades y lístelos.
- Determine los atributos por cada entidad encontrada.
- Especifique el nivel de datos por cada entidad.
- Especifique las relaciones de correspondencia entre las entidades.

Solución:

- Determinando las entidades del caso



- Determinando los atributos por cada entidad

Listado de entidades	Listado de atributos
LIBRO	Código que lo identifica, título y la cantidad de páginas que tiene.
MATERIA	Código que lo identifica y nombre de la materia.
EJEMPLAR	Código que lo identifica y estado de conservación.
USUARIO	Número de DNI, nombres, apellido paterno, dirección y ocupación.

- Especificando el nivel de datos del caso

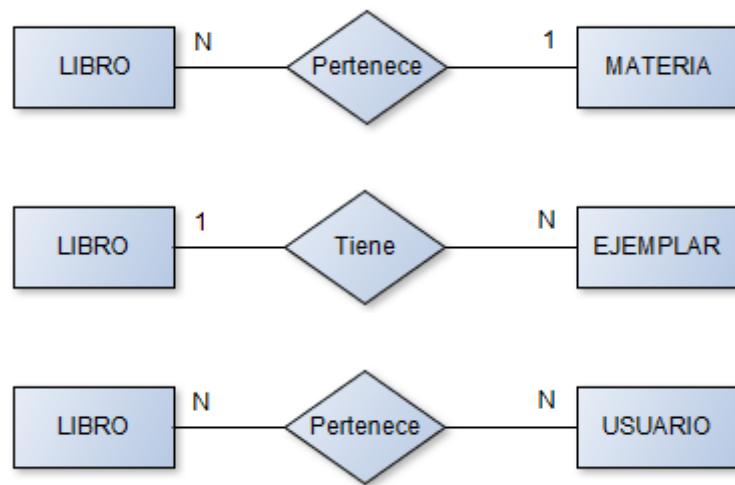
Entidad: LIBRO		
Código	Titulo	Cantidad de Paginas
L001	BASE DE DATOS RELACIONALES	500
L002	VISUAL C# 2014	600

Entidad: MATERIA	
Código	Nombre
M001	BASE DE DATOS
M002	LENGUAJE DE PROGRAMACIÓN

Entidad: EJEMPLAR	
Código	Estado de conservación
E001	BUENO
E002	REGULAR

Entidad: USUARIO				
Codigo	Nombres	Apellido Paterno	Dirección	Ocupación
U001	MARIA	ACOSTA	AV. EL SOL 657	DOCTOR
U002	GUADALUPE	ZAMORA	AV. LAS PALMERAS 3458	ABOGADO

- Especificando las relaciones de correspondencia



1.3. Arquitectura de un sistema de bases de datos

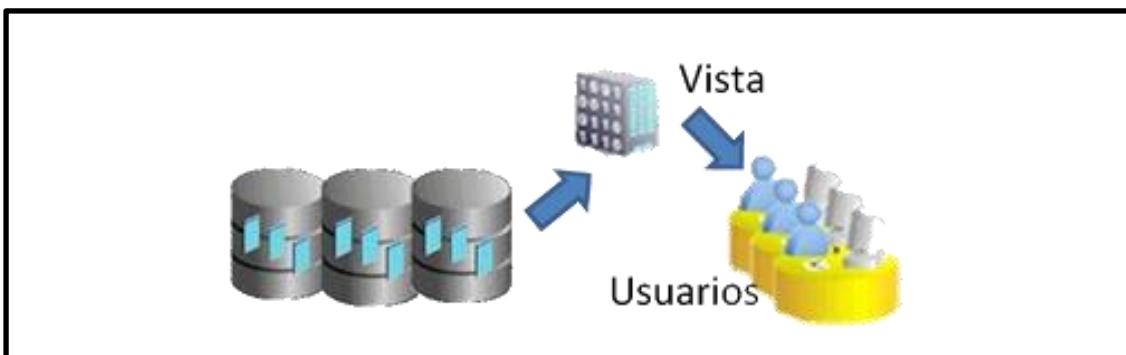


Figura 3: Vistas de una base de datos

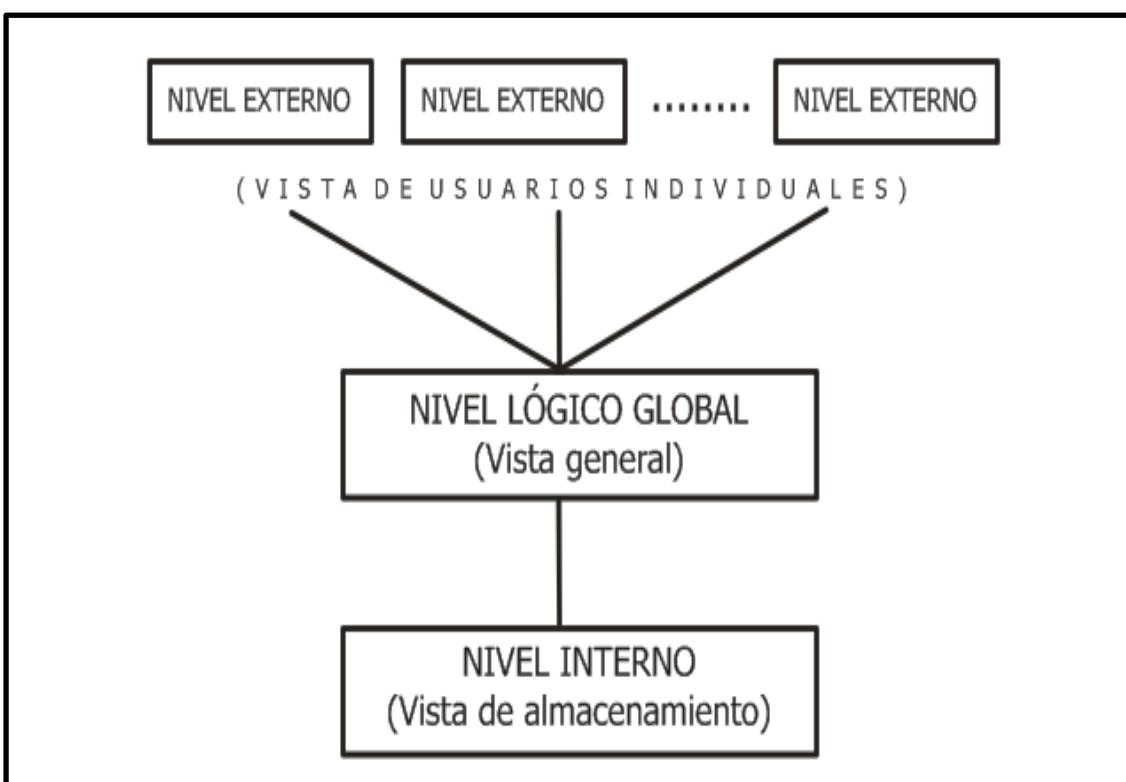
Fuente.- Tomado de http://4.bp.blogspot.com/_IlyXjNjht4/TQpziVPyon/AAAAAAAABU/legf33kEanE/s320/vistas.png

1.3.1. Arquitectura de un sistema de base de datos

A continuación, presentaremos la arquitectura de un **SGBD**, aunque no podemos asegurar que cualquier **SGBD** se corresponda exactamente con ella. Sin embargo, esta arquitectura se corresponde suficientemente bien con un gran número de sistemas. Además, está de acuerdo con la arquitectura propuesta por el grupo **ANSI/SPARC** (American National Standards Institute, Standards Planning And Requirements Committee).

La arquitectura se divide en tres niveles generales:

- El **nivel interno** es el más cercano al almacenamiento físico, o sea, es el relacionado con la forma en que los datos están realmente almacenados.
- El **nivel externo** es el más cercano a los usuarios, o sea, es el relacionado con la forma en que los datos son vistos por cada usuario individualmente.
- El **nivel lógico global** es un nivel intermedio entre los dos anteriores.



Existirán varias "vistas externas" diferentes. Cada vista es una representación más o menos abstracta de alguna porción de la base de datos total y existirá únicamente una "vista general", consistente en una representación también abstracta de la base de datos en su totalidad. Igualmente, existirá una única "vista interna" que representa a la base de datos completa, tal y como está realmente almacenada.

1.3.1.1 Nivel Externo

Es el nivel del usuario individual, donde un usuario puede ser un programador de aplicación o un usuario final con cualquier grado de sofisticación. Cada usuario tiene un lenguaje a su disposición:

- Para el programador, ese lenguaje puede ser un lenguaje de programación convencional, tal como Java, o un lenguaje de programación específico de un sistema, tal como el Visual C#.
- Para el usuario final, el lenguaje puede ser un lenguaje de consulta (interrogaciones, query) o un lenguaje de propósito especial, quizás basado en sistemas de menús y ventanas, y construido para satisfacer los requerimientos de un usuario, y se encuentra soportado por algún programa de aplicación en línea.

Es importante señalar que todo lenguaje debe incluir un sublenguaje de datos, o sea, un subconjunto del lenguaje que trata específicamente con los objetos de la base de datos y sus operaciones.

En principio, cualquier sublenguaje de datos es realmente una combinación de, al menos, dos lenguajes subordinados: un **lenguaje de definición de datos (DDL)**, el cual garantiza la definición o descripción de los objetos de la base de datos, y un **lenguaje de manipulación de datos (DML)**, el que garantiza la manipulación o procesamiento de esos objetos.

Ya se ha indicado que un usuario individual estará, generalmente, interesado solo en cierta porción de la base de datos. Aún más, la vista de esa porción será generalmente abstracta cuando se compara con la forma en que los datos están físicamente almacenados. El término definido por el comité **ANSI/SPARC** para una vista de un usuario es **vista externa**, la cual es el contenido de la base de datos tal y como es vista por un usuario en particular. Dicho de otra manera, para ese usuario, la vista externa es la base de datos.

El sublenguaje de datos del usuario se define en términos de artículos externos; por ejemplo, una operación del DML que permite recuperar artículos, generará una ocurrencia de artículos externos y no una ocurrencia de artículos almacenados.

Cada vista externa se define mediante un esquema externo consistente, básicamente, en definiciones de cada uno de los diferentes tipos de artículos externos en esa vista.

El esquema externo se escribe usando la porción del **DDL** del sublenguaje de datos del usuario; además, tiene que existir una definición de la correspondencia entre el esquema externo y el esquema lógico global.

1.3.1.2 Nivel Lógico Global

La vista lógica es una representación del contenido informativo total de la base de datos. En comparación con la forma en que los datos están almacenados físicamente, es una forma **abstracta**.

Esta vista puede ser muy diferente de la forma en la que los datos son vistos por un usuario en particular. Pretende ser una vista de los datos tal como son en lugar de cómo los usuarios están forzados a verlos por las restricciones, digamos, de un lenguaje particular o de un determinado hardware que utilicen.

Consiste en múltiples ocurrencias de múltiples tipos de artículos lógicos. Por ejemplo, puede ser una colección de ocurrencias de artículos de departamentos, más una colección de ocurrencia de artículos de empleados, etc. Un artículo lógico no es necesariamente igual a un artículo externo ni a un artículo almacenado.

La vista lógica se define mediante el esquema lógico que incluye las definiciones de cada uno de los diferentes tipos de artículos lógicos. El esquema lógico se describe usando otro lenguaje de definición de datos: el **DDL** lógico. Si se desea lograr la independencia de los datos, entonces las definiciones del **DDL** lógico no deben comprender ninguna consideración sobre la estructura de almacenamiento ni la estrategia de acceso. Ellas tienen que ser definiciones sólo referentes al contenido informativo.

Entonces, es una vista del contenido total de la base de datos y el esquema lógico es una definición de esa vista. Sin embargo, el esquema lógico no es, simplemente, un conjunto de definiciones como las que se encuentran, por ejemplo, en un programa Java. Las definiciones en el esquema lógico deben incluir una gran cantidad de aspectos adicionales, tales como los chequeos de protección y los chequeos de integridad.

En la mayoría de los sistemas actuales, el esquema lógico es realmente sólo un poco más que la simple unión de todos los esquemas externos individuales; posiblemente con la adición de algunos chequeos simples de protección e integridad. Sin embargo, está claro que los sistemas del futuro soportarán un nivel lógico mucho más sofisticado, que permita también describir la forma en que se usan los datos, cómo fluyen de un punto a otro, para qué se usan en cada punto, a qué controles son sometidos, etc.

1.3.1.3 Nivel Interno

La vista interna es una representación de bajo nivel de la base de datos completa, que consiste en múltiples ocurrencias de múltiples tipos de artículos internos.

"Artículo interno" es el término definido por **ANSI/SPARC** para la construcción que hasta ahora hemos llamado artículo almacenado. La vista interna está entonces aún a un paso del nivel físico, ya que ella no opera en términos de artículos físicos (también llamados páginas o bloques) ni con consideraciones específicas de los equipos, tales como tamaños de sectores o pistas.

Básicamente, la vista interna asume un espacio de dirección lineal infinita. Los detalles de cómo se hace corresponder ese espacio con el almacenamiento físico son muy específicos de un sistema y deliberadamente se omitieron de la arquitectura.

La vista interna se describe mediante el esquema interno, el cual no sólo define los diferentes tipos de artículos almacenados, sino que también especifica los índices que existen, la representación de los campos almacenados, la secuencia física en que están los artículos almacenados, etc. El esquema interno se describe usando otro lenguaje de definición de datos: el **DDL** interno.

1.3.1.4 Correspondencias entre los niveles de arquitectura

En el esquema presentado de la arquitectura de un **SGBD**, se observan los niveles de correspondencias, uno entre los niveles externo y lógico global y otro entre los niveles lógico global e interno.

- La **correspondencia lógica/interna** especifica la forma en que los artículos y campos lógicos se representan en el nivel interno. Además, si se cambia la estructura de la vista interna, mejor dicho, si se hace un cambio en el esquema interno, entonces, la correspondencia lógica/interna tiene también que cambiar, de modo que el esquema lógico permanezca invariable. En otras palabras, los efectos de estos cambios deben ser aislados por debajo del nivel lógico para que se mantenga la independencia de los datos.
- Existe también una **correspondencia externo/lógica** entre cada vista externa particular y la vista lógica. Las diferencias que pueden existir entre estos dos niveles son similares a las que pueden existir entre las vistas lógica e interna. Por ejemplo, los campos pueden tener diferente tipos de datos, se pueden cambiar los nombres de artículos y campos, múltiples campos lógicos pueden ser combinados en un único campo externo, etc. Puede existir al mismo tiempo cualquier cantidad de vistas externas; cualquier cantidad de usuarios puede compartir una vista externa dada; las diferentes vistas externas se pueden solapar. Algunos sistemas permiten la definición de una vista externa a partir de otra (mediante una correspondencia externa/externa); esta característica es útil cuando varias vistas externas están estrechamente relacionadas entre sí.

1.3.2. Organizaciones de archivos y el nivel interno de la arquitectura

Físicamente, se puede decir que las bases de datos se almacenan siguiendo diferentes organizaciones de archivos. Cada una de estas tiene distintas características de desempeño; ninguna se puede decir que sea óptima para todas las aplicaciones, sino que se decide emplear una u otra en dependencia de la aplicación.

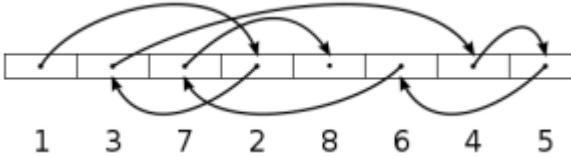
A continuación, presentaremos las principales características de las más usadas organizaciones de archivos, de modo que puedan tener ideas de cómo se pueden organizar los datos en estos archivos para lograr determinados objetivos. Pero, antes de ello, abordaremos algunas definiciones y características relativas a los archivos. Empezaremos diciendo que la utilización de los archivos se debe a dos causas fundamentales:

- Manejo de mucha información
- Almacenamiento de información permanente

Ya que el trabajo en memoria interna es muy rápido, pero es un recurso relativamente escaso y caro, y la memoria periférica es más barata, aunque más lenta. Además, influye en esto el hecho de que el contenido de la memoria central se pierde al cesar el fluido eléctrico, lo que hace aconsejable almacenar en soportes externos los grandes volúmenes de información.

1.3.2.1 Operaciones fundamentales sobre archivos

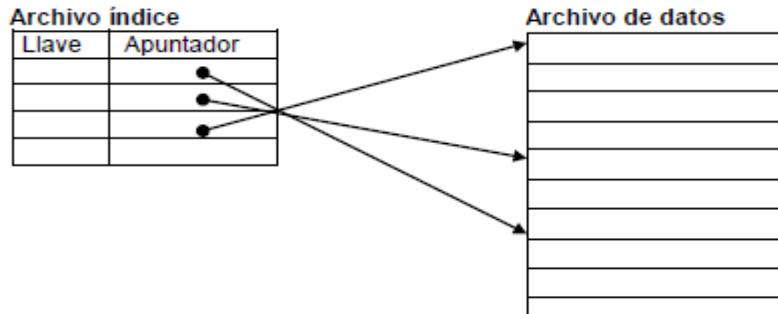
Las operaciones fundamentales que se realizan sobre archivos pueden colocarse en dos grandes grupos:

Acceso	<p>Se entiende el acceso a un archivo como la forma por la cual es posible tener conocimiento de la información contenida en los respectivos registros, mejor dicho, el modo como se pueden leer los registros del archivo, las ocurrencias de estos registros almacenadas en él.</p> <p>Las diferentes formas de accesar un archivo pueden agruparse en dos tipos:</p> <ol style="list-style-type: none"> En secuencia o secuencial. Se caracteriza por el hecho de que, en la lectura del archivo, es obligatorio que, a continuación del tratamiento del registro de orden n, se traten los de orden $n+1$, $n+2$, ... hasta un límite determinado. Puede realizarse desde el inicio del archivo o a partir de un registro n.  <ol style="list-style-type: none"> Aleatorio o directo. Se caracteriza porque se lee, y consecuentemente trata cualquier registro en cualquier orden (a través de la llave o una transformación de ésta). No puede realizarse en periféricos de acceso secuencial. 
Actualización	<p>La mayoría de los archivos deben ser actualizados con el transcurso del tiempo para que se ajusten a la realidad. La actualización de un archivo se realiza mediante tres operaciones fundamentales:</p> <ul style="list-style-type: none"> Alta.- Se le llama así a la creación de un nuevo registro en el archivo. Baja.- Se le llama así cuando se elimina un registro. Esta puede ser lógica o física. Modificación.- Se le llama así cuando se cambia alguna información en el registro. <p>Para realizar las actualizaciones es importante validar los datos que se van a actualizar.</p>

1.3.2.2 Organización de los archivos

Es necesario indicar que la bibliografía existente al respecto asume diferentes posiciones respecto a la terminología de emplear. En muchos casos, las definiciones son ambiguas y hay poco acuerdo entre los textos que tratan este tema.

A pesar de esto, en la actualidad, las organizaciones generalmente aceptadas son:

Organización Secuencial	<p>Es la forma más sencilla para almacenar los registros de un archivo, uno después de otro, a continuación del otro. Todos los registros se almacenan por su posición: uno es el primero, el siguiente es el segundo y así sucesivamente.</p> <p>Es la más vieja forma de organizar un archivo y fue empleada para las cintas magnéticas desde los inicios de la computación.</p> <table border="1" data-bbox="684 496 1124 570"> <tr> <td>1</td><td>2</td><td>...</td><td>n</td><td>n+1</td><td>...</td></tr> </table> <p>Así por ejemplo, el archivo POSTULANTE, en el que están las ocurrencias de postulantes una a continuación de otra; sería de la siguiente manera:</p> <table border="1" data-bbox="470 743 1335 1073"> <thead> <tr> <th>Nº</th><th>DNI</th><th>NOMBRES</th><th>FECHA</th><th>HORA</th></tr> </thead> <tbody> <tr><td>1</td><td>43462698</td><td>ABANTO JIMES ANGELA MILAGROS</td><td>07/05/2012</td><td>09:00</td></tr> <tr><td>2</td><td>45017141</td><td>AVALOS BARAHONA LESDY TANIA</td><td>07/05/2012</td><td>09:00</td></tr> <tr><td>3</td><td>41194541</td><td>AVELLANEDA BAUTISTA SUHGEY MAVILA</td><td>07/05/2012</td><td>09:00</td></tr> <tr><td>4</td><td>10798934</td><td>BOCANEGRA ARIAS SONIA KAREN</td><td>07/05/2012</td><td>09:20</td></tr> <tr><td>5</td><td>43524781</td><td>CALDERON VALERA DORIS HAYDEE</td><td>07/05/2012</td><td>09:20</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>12</td><td>43129691</td><td>GARCIA CRISTOBAL CARLOS ALOR</td><td>07/05/2012</td><td>10:00</td></tr> <tr><td>13</td><td>42626923</td><td>HUANACUNI TICONA SOLEDAD</td><td>07/05/2012</td><td>10:20</td></tr> <tr><td>14</td><td>31037402</td><td>HUAYHUAS ROJAS PITHER</td><td>07/05/2012</td><td>10:20</td></tr> <tr><td>15</td><td>04080376</td><td>ILLANES BUSTAMANTE JOSE LUIS</td><td>07/05/2012</td><td>10:20</td></tr> <tr><td>16</td><td>09019417</td><td>LA MATTA CASTRO LUIS ALBERTO</td><td>07/05/2012</td><td>10:40</td></tr> </tbody> </table>	1	2	...	n	n+1	...	Nº	DNI	NOMBRES	FECHA	HORA	1	43462698	ABANTO JIMES ANGELA MILAGROS	07/05/2012	09:00	2	45017141	AVALOS BARAHONA LESDY TANIA	07/05/2012	09:00	3	41194541	AVELLANEDA BAUTISTA SUHGEY MAVILA	07/05/2012	09:00	4	10798934	BOCANEGRA ARIAS SONIA KAREN	07/05/2012	09:20	5	43524781	CALDERON VALERA DORIS HAYDEE	07/05/2012	09:20	12	43129691	GARCIA CRISTOBAL CARLOS ALOR	07/05/2012	10:00	13	42626923	HUANACUNI TICONA SOLEDAD	07/05/2012	10:20	14	31037402	HUAYHUAS ROJAS PITHER	07/05/2012	10:20	15	04080376	ILLANES BUSTAMANTE JOSE LUIS	07/05/2012	10:20	16	09019417	LA MATTA CASTRO LUIS ALBERTO	07/05/2012	10:40
1	2	...	n	n+1	...																																																														
Nº	DNI	NOMBRES	FECHA	HORA																																																															
1	43462698	ABANTO JIMES ANGELA MILAGROS	07/05/2012	09:00																																																															
2	45017141	AVALOS BARAHONA LESDY TANIA	07/05/2012	09:00																																																															
3	41194541	AVELLANEDA BAUTISTA SUHGEY MAVILA	07/05/2012	09:00																																																															
4	10798934	BOCANEGRA ARIAS SONIA KAREN	07/05/2012	09:20																																																															
5	43524781	CALDERON VALERA DORIS HAYDEE	07/05/2012	09:20																																																															
...																																																															
12	43129691	GARCIA CRISTOBAL CARLOS ALOR	07/05/2012	10:00																																																															
13	42626923	HUANACUNI TICONA SOLEDAD	07/05/2012	10:20																																																															
14	31037402	HUAYHUAS ROJAS PITHER	07/05/2012	10:20																																																															
15	04080376	ILLANES BUSTAMANTE JOSE LUIS	07/05/2012	10:20																																																															
16	09019417	LA MATTA CASTRO LUIS ALBERTO	07/05/2012	10:40																																																															
Organización Indizada	<p>Los accesos a los registros almacenados se realizan a través de un índice. La forma básica de un índice incluye una llave de registro y la dirección de almacenamiento para éste. Para encontrar un registro específico, se rastrea primero el índice y, al encontrar la dirección, se accede directamente el registro.</p>  <p>Entonces, un índice es un archivo auxiliar que se utiliza para accesar los registros de otro archivo, que llamaremos archivo principal o de datos, por el valor de un dato o conjunto de datos, que es la llave o clave de indización.</p> <p>A los registros del índice se les llama entradas. Cada entrada corresponde a un valor o intervalo de valores de la llave y es el padre cuyos hijos son los registros del archivo principal, en los cuales la llave toma el valor, o un valor del intervalo de valores que corresponde a la entrada en cuestión.</p>																																																																		

	<p>En la siguiente imagen se muestra un archivo índice muy simple en el que se tiene la llave y un puntero al dato en el archivo principal.</p> <p>Llave apuntador</p> <table border="1"><tr><td>Alvarez, Luis</td><td>●</td><td></td></tr><tr><td>García, Berta</td><td>●</td><td></td></tr><tr><td>Lazo, Juan</td><td>●</td><td></td></tr><tr><td>López, María</td><td>●</td><td></td></tr><tr><td>Martínez, José</td><td>●</td><td></td></tr></table> <p>Archivo Índice</p> <p>The diagram illustrates a simple index structure. On the left, labeled 'Archivo Índice', is a table with five rows, each containing a name ('Llave') and a small black dot ('apuntador'). On the right, labeled 'Archivo de datos', is a table with six rows, each containing a name and some descriptive text. Arrows connect the dots in the index table to specific rows in the data table. Specifically, the first row in the index points to the fourth row in the data table; the second row points to the third; the third row points to the second; the fourth row points to the fifth; and the fifth row points to the sixth. The data table has columns for names and other information, with ellipses indicating more entries.</p> <table border="1"><tr><td>López, María</td><td>Técn A</td><td>...</td></tr><tr><td>Lazo, Juan</td><td>Oper B</td><td>...</td></tr><tr><td>García, Berta</td><td>Oper C</td><td>...</td></tr><tr><td>Martínez, José</td><td>Técn A</td><td>...</td></tr><tr><td>Alvarez, Luis</td><td>Técn B</td><td>...</td></tr></table> <p>Archivo de datos</p>	Alvarez, Luis	●		García, Berta	●		Lazo, Juan	●		López, María	●		Martínez, José	●		López, María	Técn A	...	Lazo, Juan	Oper B	...	García, Berta	Oper C	...	Martínez, José	Técn A	...	Alvarez, Luis	Técn B	...
Alvarez, Luis	●																														
García, Berta	●																														
Lazo, Juan	●																														
López, María	●																														
Martínez, José	●																														
López, María	Técn A	...																													
Lazo, Juan	Oper B	...																													
García, Berta	Oper C	...																													
Martínez, José	Técn A	...																													
Alvarez, Luis	Técn B	...																													

1.4. Introducción a las bases de datos en SQL Server 2014



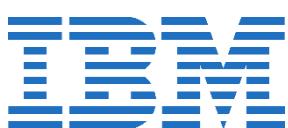
1.4.1. Lenguaje estructurado de consultas

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.

1.4.2. 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.

Historia de versiones			
Versión	Año	Nombre de la versión	Nombre clave
1.0 (OS/2)	1989	SQL Server 1-0	SQL
4.21 (WinNT)	1993	SQL Server 4.21	SEQUEL
6.0	1995	SQL Server 6.0	SQL95
6.5	1996	SQL Server 6.5	Hydra
7.0	1998	SQL Server 7.0 ¹	Sphinx
-	1999	SQL Server 7.0 OLAP Tools	Plato
8.0	2000	SQL Server 2000 ²	
8.0	2003	SQL Server 2000 64-bit Edition	Liberty
9.0	2005	SQL Server 2005 ³	Yukon
10.0	2008	SQL Server 2008 ⁴	Katmai
10.25	2010	SQL Azure DB	CloudDatabase
10.50	2010	SQL Server 2008 R2 ⁵	Kilimanjaro
11.0	2012	SQL Server 2012 ⁶	Denali
12.0	2014	SQL Server 2014 ⁷	SQL14 (antes Hekaton)

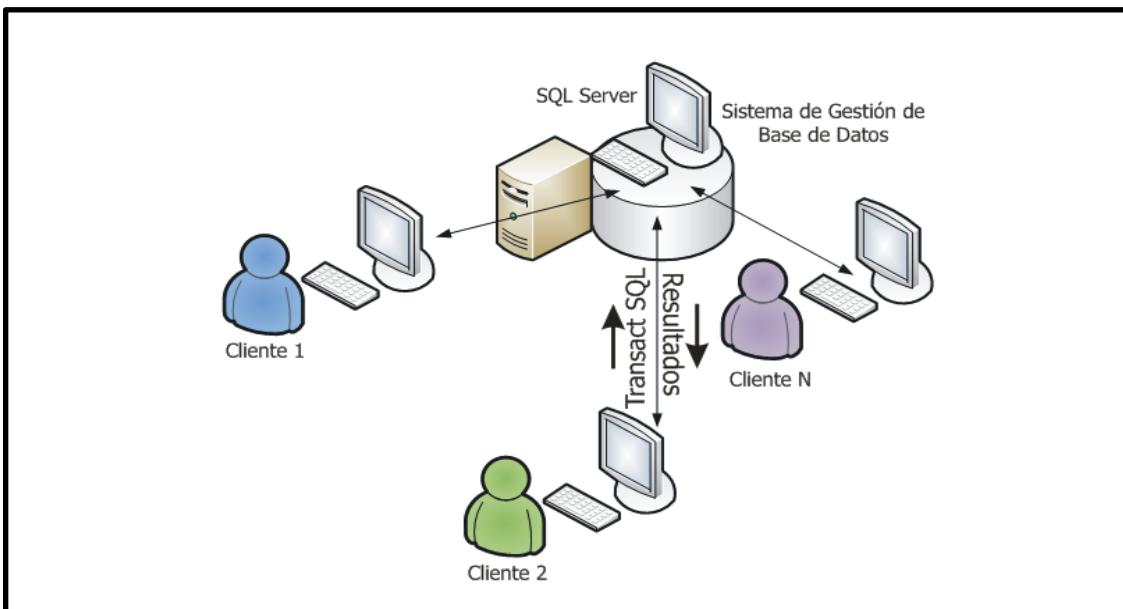
Figura 4: Historia de versiones de SQL
Fuente.- Tomado de https://es.wikipedia.org/wiki/Microsoft_SQL_Server

1.4.3. 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.

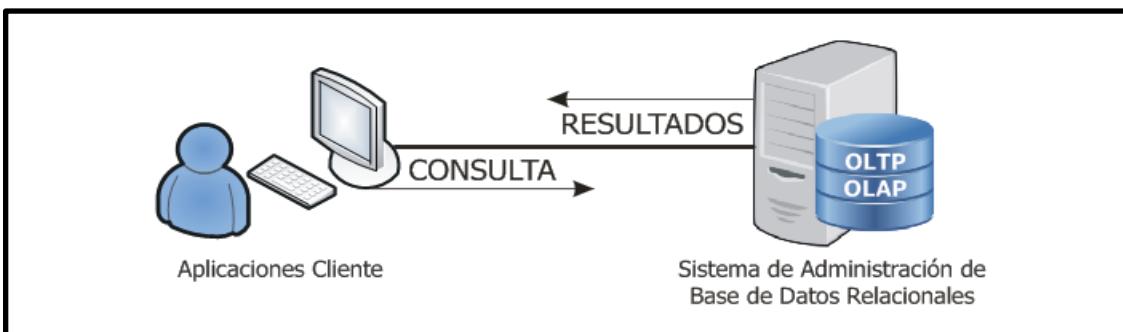
1.4.3.1 Implementación de las base de datos con SQL SERVER

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**.



1.4.3.2 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.

1.4.3.3 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.

1.4.3.4 SQL SERVER 2014

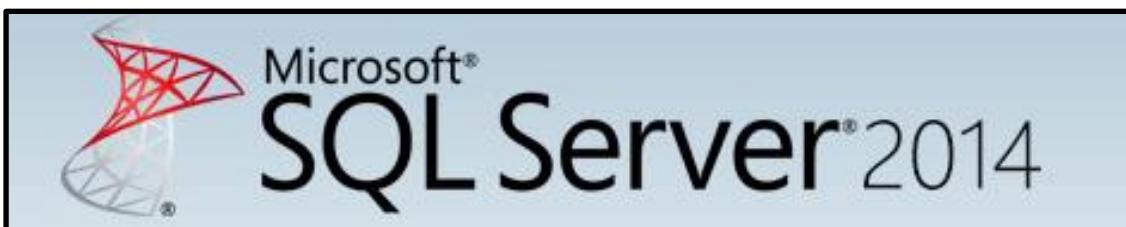


Figura 5: Logo SQL Server 2014

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.

1.4.3.5 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 6: Logo SQL Server 2014

Fuente.- Tomado de http://s14.postimg.org/v3q5hljh/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 7: 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 8: Logo SQL Server 2014

Fuente.- Tomado de <http://zdneta4.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.

1.5. Creación de base de datos en SQL Server 2014

1.5.1. Componentes de una base de datos

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:

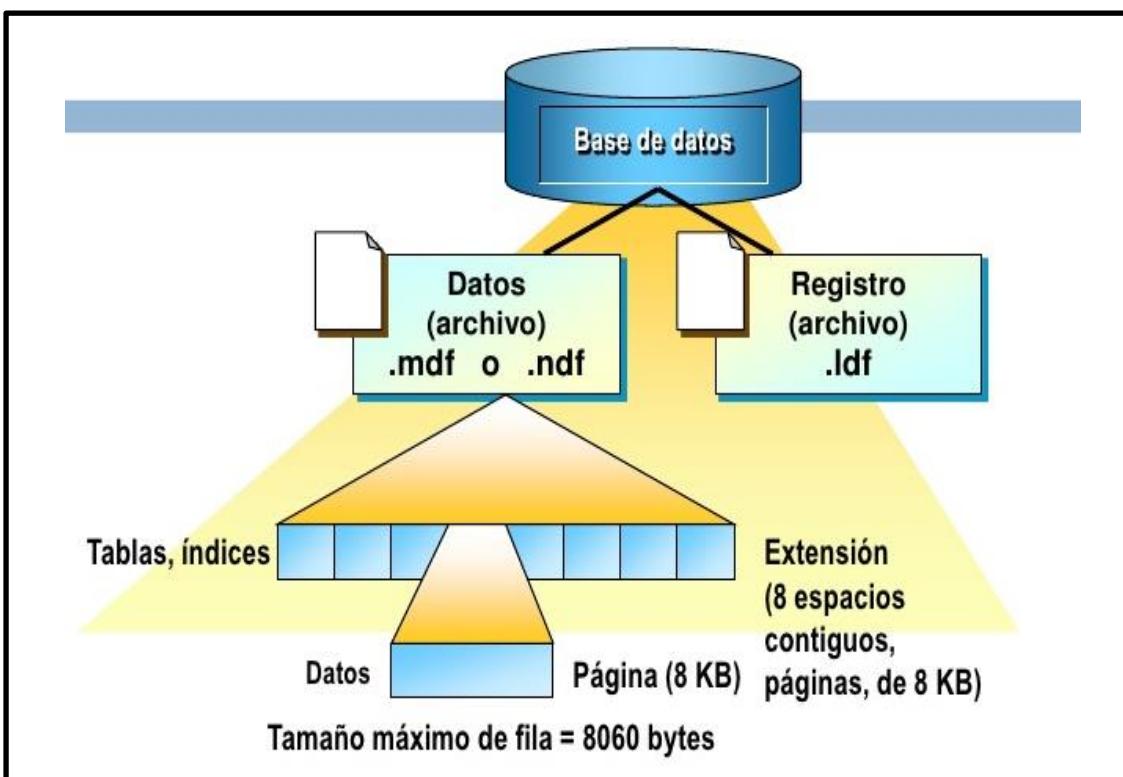


Figura 9: Componentes de una base de datos en SQL Server

Fuente.- Tomado de <http://image.slidesharecdn.com/transactionlog-091106221712-phpapp02/95/sql-server-como-se-almacenar-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

	<p>tengan archivos de datos secundarios si el archivo principal es lo suficientemente grande como para contener todos los datos.</p> <p>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.</p>
Archivo de Transacciones	<p>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.</p>

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

1.5.2. Crear, modificar y eliminar una base de datos

1.5.2.1 Creación de una Base de Datos estándar

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:

<p>¿Dónde se guarda la base de datos?</p>	<p>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:</p> <p>C:\Program Files\Microsoft SQL Server\ MSSQL12.SQLEXPRESS\ MSSQL\DATA</p>
<p>¿Cuál es el tamaño asignado a los archivos de la base de datos?</p>	<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%

Sentencia 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.

1.5.2.2 Creación de una Base de Datos 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.

Sentencia 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
```

1.5.3. 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	<p>Es la especificación de la ruta y el nombre del archivo físico; estos nombres serán visibles desde el explorador de archivos de windows.</p> <p style="text-align: center;">FILENAME='C:\base\ventas.mdf'</p> <p>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.</p>
SIZE	<p>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:</p> <p style="text-align: center;">SIZE=2048KB SIZE=2MB SIZE=2</p> <p>Todas las representaciones son similares; lo que lo diferencia es la forma de especificar la unidad de medida.</p>
MAXSIZE	<p>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:</p> <p style="text-align: center;">MAXSIZE=300 (300 Megabyte) MAXSIZE=UNLIMITED (Límite permitido por el sistema)</p>
FILEGROWTH	<p>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.</p> <p style="text-align: center;">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).</p>

1.5.3.1 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 COMERCIO:

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

Podemos interpretar de la siguiente manera, si la base de datos Comercio existe, entonces se eliminará dicha base, caso contrario se omite la eliminación.

Actividad

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
<hr/>							
1	BD_EJEMPLO	1	C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA\BD_EJEMPLO.mdf		PRIMARY	4288 KB	Unlimited
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

Caso 2: BD_COMERCIAL

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

- Valide la existencia de la base de datos.
- Asigne el nombre “**BD_COMERCIAL_PRI**” al archivo principal con un tamaño inicial de **50MB**, un tamaño máximo aceptado por el sistema, una tasa de crecimiento del **10%** y debe ser guardado en la carpeta **C:\COMERCIAL**.
- 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: BD_COMERCIAL

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

- Valide la existencia de la base de datos.
- Asigne el nombre “**BD_COMERCIAL_PRI**” al archivo principal 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 el nombre “**BD_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('BD_COMERCIAL') IS NOT NULL
BEGIN
    DROP DATABASE BD_COMERCIAL
END
GO

CREATE DATABASE BD_COMERCIAL
ON
(
    NAME=BD_COMERCIAL_PRI,
    FILENAME='C:\COMERCIAL\DATA\BD_COMERCIAL_PRI.MDF',
    SIZE=40,
    MAXSIZE=250,
    FILEGROWTH=5MB
)
LOG ON (
    NAME=BD_COMERCIAL_LOG,
    FILENAME='C:\COMERCIAL\LOG\BD_COMERCIAL_TRA.LDF',
    SIZE=10,
    MAXSIZE=50,
    FILEGROWTH=5%
)
GO

SP_HELPDB BD_COMERCIAL
GO
```

Caso 4: BD_COMERCIAL

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

- Valide la existencia de la base de datos.
- Asigne el nombre “**BD_COMERCIAL_PRI**” al archivo principal 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:\COMERCIAL\DATA**.
- Asigne el nombre “**BD_COMERCIAL_SEC**” al archivo secundario 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:\COMERCIAL\SEC.
- Asigne el nombre “BD_COMERCIAL_TRA” al archivo de transacciones 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:\COMERCIAL\LOG.
 - 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\DATA\BD_COMERCIAL_PRI.MDF',
    SIZE=100,
    MAXSIZE=UNLIMITED,
    FILEGROWTH=15%
),
(
    NAME=BD_COMERCIAL_SEC,
    FILENAME='C:\COMERCIAL\SEC\BD_COMERCIAL_SEC.NDF',
    SIZE=50,
    MAXSIZE=250,
    FILEGROWTH=10%
)
LOG ON (
    NAME=BD_COMERCIAL_LOG,
    FILENAME='C:\COMERCIAL\LOG\BD_COMERCIAL_TRA.LDF',
    SIZE=5,
    MAXSIZE=50,
    FILEGROWTH=5
)
GO

SP_HELPDB BD_COMERCIAL
GO
```

Autoevaluación

De los siguientes casos, realice lo siguiente:

- Determine las entidades y lístelos.
- Determine los atributos por cada entidad.
- Especifique el nivel de datos por cada entidad.
- Especifique las relaciones de correspondencia entre las entidades.
- Diseñe las relaciones de correspondencia con sus respectivos atributos usando Studio Case-YeD o Diaw.

1. CASO: Control de escuelas

Para el control de las escuelas primarias en la ciudad de Lima se tiene la siguiente información:

De cada escuela, un número que la identifica, su nombre y su dirección. De cada aula, el número que la identifica, la cantidad de asientos que tiene y el piso en que se encuentra situada. De cada grupo de clases, un identificador del grupo, el grado escolar del grupo y la cantidad de alumnos que tiene. De cada maestro, su DNI, su nombre, su sexo y el año en que se graduó. De cada alumno, su número de expediente, su nombre, su sexo y su fecha de nacimiento.

Una escuela tiene muchas aulas y muchos grupos de clases, pero cada aula pertenece a una escuela y lo mismo sucede con cada grupo.

Un grupo siempre recibe clases en la misma aula y un aula pertenece a un solo grupo. En un grupo imparte clases un maestro y éste solo imparte clases en un grupo. En cada grupo de clases hay muchos alumnos, pero un alumno forma parte de un solo grupo.

2. CASO: Biblioteca

En un centro de información científica (biblioteca), se desea controlar el uso de la bibliografía que existe. En el centro existen varias salas. De cada sala se conoce el número que la identifica, especialidad y cantidad de empleados.

En cada sala están disponibles revistas y libros. De cada revista se conoce el código que la identifica, nombre, fecha de publicación y país de procedencia. De cada libro se conoce su código, título, editorial y país de procedencia. Cada libro y revista existente sólo se encuentra en una sala. Las revistas sólo pueden ser consultadas en las salas; sin embargo, los libros pueden ser solicitados en préstamo por los usuarios, llevándose el control de dichos préstamos. Un libro puede ser prestado a varios usuarios (durante la existencia del libro) y un usuario puede solicitar varios libros. De cada usuario se sabe su DNI, nombre, distrito en que reside y departamento en que trabaja. Para cada libro se conoce la fecha de inicio de un préstamo realizado a determinado usuario.

3. CASO: Hospital

En un hospital se desea controlar la actividad asistencial que se brinda en las consultas de la Sala de Emergencias.

Allí brindan sus servicios los médicos organizados en equipos. A un equipo pertenecen varios médicos y un médico pertenece a un equipo. De cada equipo se conoce el código que lo identifica, el nombre del jefe del equipo y la periodicidad con la que le corresponde hacer guardia al equipo. De cada médico se conoce su DNI, nombre, especialidad y categoría. Los pacientes que llegan al Cuerpo de Guardia pueden ser atendidos por varios médicos (si sus síntomas indican la necesidad de que varios especialistas lo asistan) y un médico atiende a muchos pacientes. De cada paciente se sabe su DNI, nombre, edad, sexo y ocupación. Se sabe el tiempo dedicado por el médico a la atención de un determinado paciente, así como el diagnóstico que le hizo y el tratamiento que le indicó.

Los médicos en la Sala de Emergencias pueden utilizar en su labor asistencial diferentes medios de diagnóstico (Rayos X, análisis, etc.) y un medio de diagnóstico puede ser empleado por muchos médicos. De cada medio de diagnóstico se conoce el código que lo identifica, su descripción y el costo por unidad. Para cada médico se conoce la cantidad de veces que ha ordenado la aplicación de un medio de diagnóstico dado.

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

4.1. **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.

4.2. **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%.

4.3. **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 4 *megabytes*, máximo de 75 *megabytes* y un factor de crecimiento de 2%.

5. Caso: VENTAS

Implemente la base de datos **Ventas** en SQL Server 2014, tal como se muestra en el siguiente diagrama de base de datos:

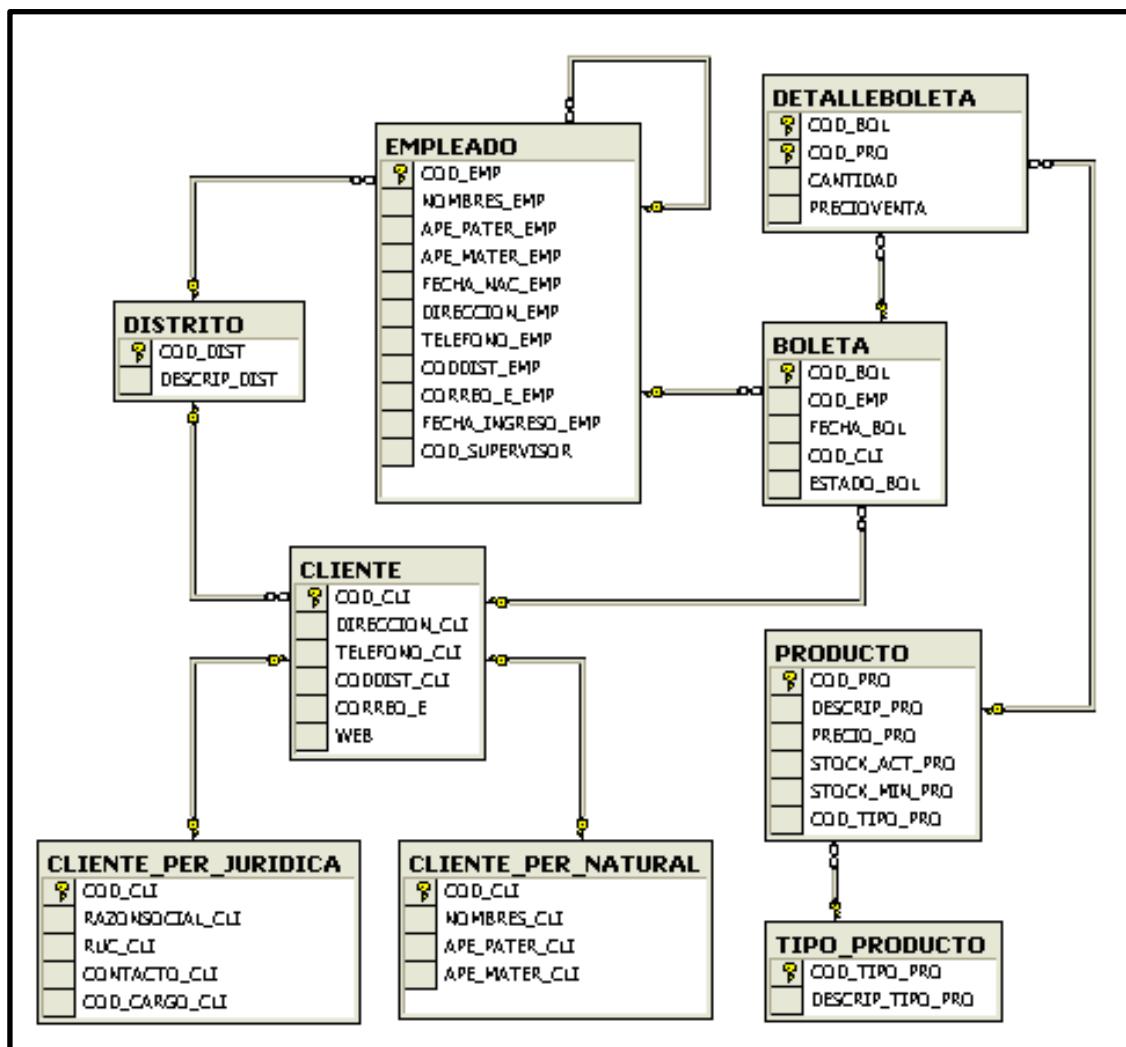


Figura 10: Diagrama de base de datos Ventas

Fuente.- Tomado de SQL Server 2014

Resumen

1. El procesamiento automatizado de datos ha pasado por diferentes etapas en su desarrollo hasta llegar a la actual, en la que se emplean bases de datos para el almacenamiento de la información.
2. El principal **objetivo de los SGBD** es garantizar la independencia de los datos respecto a los programas de aplicación.
3. Otro objetivo muy importante de los **SGBD** es la minimización de la redundancia.
4. **Entidad:** objeto del cual se describen ciertas características.
5. **Campo o atributo:** es la unidad menor de información sobre un objeto (almacenada en la base de datos) y representa una propiedad de un objeto.
6. Un **atributo** puede tomar diferentes valores sobre un cierto conjunto que se denomina dominio.
7. **Dominio:** rango de valores posibles de un atributo.
8. A un valor de un atributo definido en el dominio dado, en un cierto momento del tiempo, se le denomina **ocurrencia del atributo**.
9. Un **artículo o registro** es una colección identificable de campos asociados y representa un objeto con sus propiedades.
10. Una **ocurrencia de artículo o tupla** consiste en un grupo de ocurrencias de campos relacionados, representando una asociación entre ellos.
11. Un **archivo o archivos** es un conjunto de ocurrencias de un mismo tipo de artículo.
12. Una **base de datos** está formada por múltiples **archivos**.
13. Existen asociaciones o relaciones enlazando las entidades, que pueden tener o no atributos. Pueden establecerse sobre la misma entidad o sobre entidades diferentes. En una relación puede participar cualquier cantidad de entidades.
14. Las relaciones pueden ser de **uno a uno** (1 : 1), de **uno a muchos** (1 : m) y de **muchos a muchos** (m : m).
15. La **arquitectura de un SGBD** está compuesta por tres niveles: **externo, lógico global e interno**. El nivel lógico global de los datos representa el contenido informativo total de la base de datos y, desde el punto de vista del diseñador, es el más importante.
16. Existe una **correspondencia entre el nivel externo y el lógico global**; y entre el lógico global y el interno. Para garantizar la independencia de los datos respecto a los programas de aplicación, cualquier cambio en el nivel interno debe reflejarse adecuadamente en la correspondencia interna/lógico global, para no afectar el nivel lógico global.

17. Las principales **operaciones** que se realizan sobre los archivos son los siguientes: **acceso**, que puede ser secuencial o directo, y **actualización**, que se refiere a las altas, bajas y modificaciones de la información.

18. El **nivel interno** de la arquitectura de un SBD puede considerarse estructurada por archivos con distintos modos de organización.

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

- <http://www3.uji.es/~mmarques/f47/apun/node33.html>
En esta página web hallará algunos conceptos complementarios a los mostrados en el manual sobre la arquitectura de una base de datos.
- <http://www.mitecnologico.com/Main/EstructuraArchivosOrganizacionSecuencialIndexado>
En esta página web encontrará definiciones complementarias al almacenamiento secuencial indexado.
- <http://sistemas.itlp.edu.mx/tutoriales/basedat1/>
En esta página web hallará algunos conceptos complementarios a los mostrados en el manual sobre la introducción a la base de datos.
- <http://sistemas.itlp.edu.mx/tutoriales/basedat1/>
En esta página web encontrará definiciones complementarias sobre los objetivos de las base de datos.
- <http://sistemas.itlp.edu.mx/tutoriales/basedat1/>
En esta página web encontrará ejercicios sobre la representación de la información.



MODELO CONCEPTUAL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno diseña el diagrama entidad relación (**DER**) de un proceso de negocio a partir de casos planteados por el profesor, relacionando las características del modelo conceptual con el diseño de una base de datos en **SQL Server 2014**.

TEMARIO

2.1 Tema 6 : Modelo conceptual

2.1.1 : Modelo conceptual de datos

2.2 Tema 7 : Diagrama entidad relacional

2.2.1 : Modelo entidad relacional

2.2.2 : Diagrama entidad relacional

2.3 Tema 8 : Tipos de datos

2.4 Tema 9 : Sentencias DDL para la tabla de datos

2.4.1 : Creación de una tabla de datos CREATE

2.4.2 : Modificación de una tabla de datos ALTER

2.4.3 : Eliminación de una tabla de datos DROP

2.5 Tema 10 : Integridad referencial

2.5.1 : Asignación de PRIMARY KEY a la tabla de datos

2.5.2 : Asignación de FOREIGN KEY a la tabla de datos

2.6 Tema 11 : Implementación de una base de datos en SQL Server 2014

2.6.1 : Implementación de una base de datos en SQL Server 2014

2.6.2 : Asignación de FOREIGN KEY a la tabla de datos

ACTIVIDADES PROPUESTAS

- Los alumnos implementan un modelo conceptual a partir de un caso.
- Los alumnos implementan una base de datos y sus entidades en SQL Server 2014.

2.1 Modelo conceptual

2.1.1 Modelo conceptual de datos

Características del modelo conceptual

El proceso de diseño de la base de datos transita a través de una serie de pasos en los cuales se va avanzando de un nivel de abstracción menor a otro más profundo, mediante la elaboración de una sucesión de modelos. En los últimos años, se ha generalizado la concepción del diseño de las base de datos propuestas por el grupo **ANSI/SPARC**, la cual constituye, al mismo tiempo, una arquitectura para los **SBD**, tal y como la acabamos de estudiar.

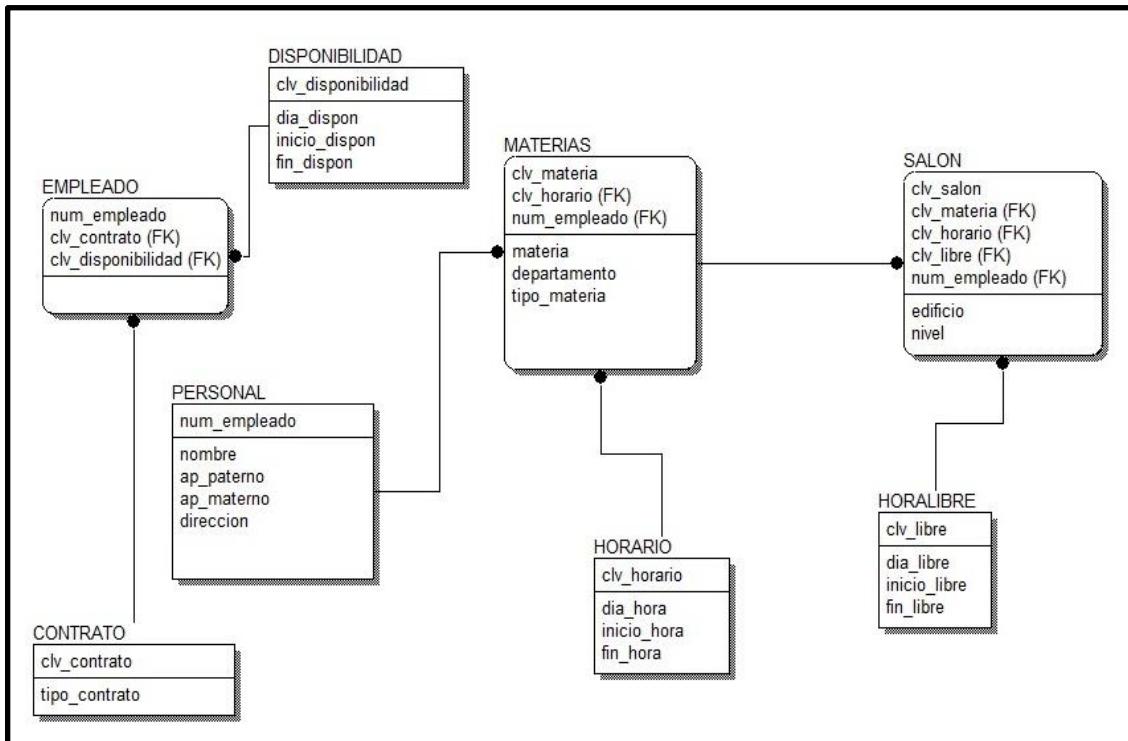


Figura 11: Modelo conceptual
Fuente.- Tomado de la aplicación Erwin

Hemos visto en esta arquitectura que cada nivel de la misma es una cierta forma de representación abstracta de la información y una de las funciones más importantes del **SGBD** consiste precisamente en permitirle al usuario la interacción con los datos en estos términos abstractos, en lugar de tenerlo que hacer directamente con la forma en que esos datos están físicamente almacenados. Es por ello que, al acometerse la tarea de diseño de una base de datos, la atención se debe centrar en el aspecto lógico de la información, ya que los detalles relacionados con el almacenamiento físico son parte de todo **SGBD** comercial que se utilice, y por tanto, no pueden ser modificados.

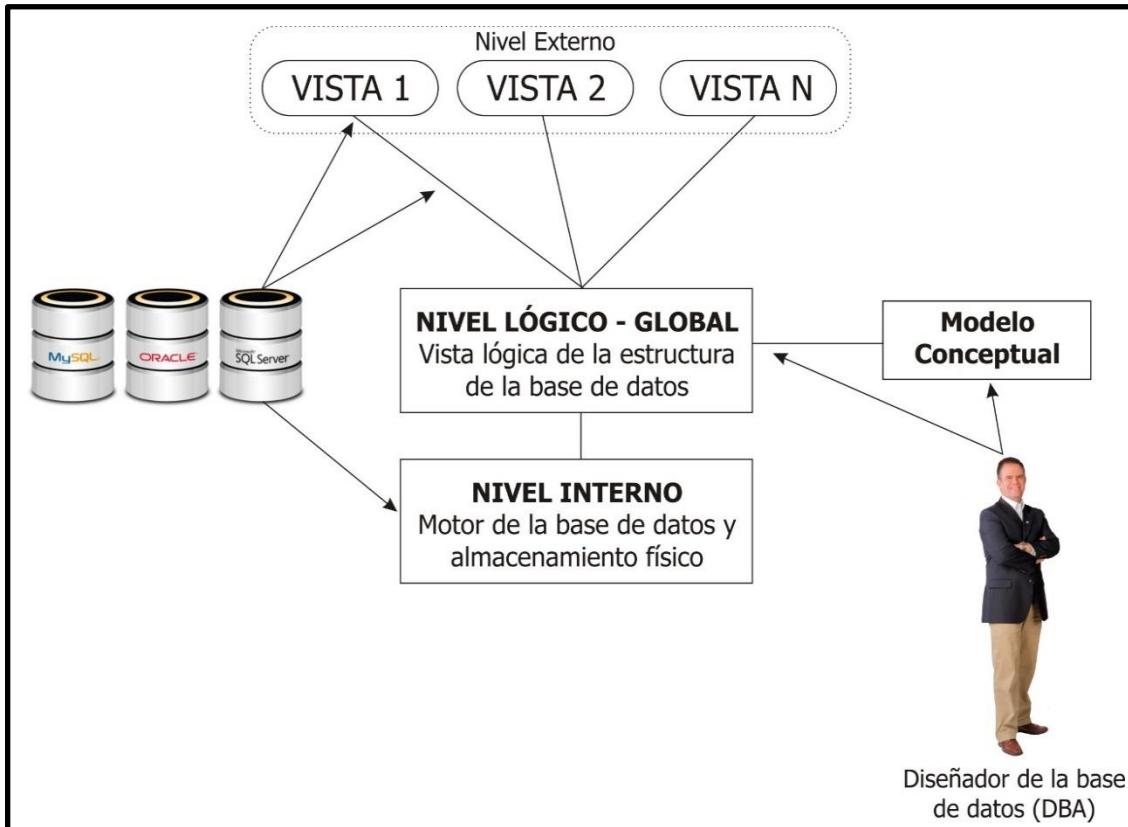
Los **SGBD** existentes utilizan diferentes modelos de datos para la representación en el nivel lógico global. Son comunes a todos ellos las siguientes características:

- La representación de la información se basa en el uso de determinadas estructuras de datos que poseen una capacidad descriptiva limitada; sólo diferencian un rasgo semántico: el tipo de proyección (1:1, 1:n, n:m).
- Utilizan una terminología que no es familiar al usuario del sistema, por lo que dificultan la comunicación usuario - diseñador.

Además, cada uno de estos modelos está vinculado con un tipo particular de **SGBD**.

Por todo ello, es necesario tratar con otro tipo de modelo cuando se aborda el problema del diseño de la base de datos, el cual debe superar los problemas anteriores y constituye un nivel de abstracción intermedio entre la realidad informativa y el nivel lógico global de la arquitectura. A este nuevo tipo de modelo se le denomina modelo conceptual.

A continuación, mostraremos los respectivos niveles:



El proceso de modelamiento conceptual es denominado también modelamiento **semántico**, ya que con estos modelos se pretende reflejar en mayor medida la semántica o significado de los datos y sus interrelaciones.

2.2 Diagrama Entidad-Relación

2.2.1. Modelo Entidad-Relación

Este modelo (presentado mediante un diagrama y denominado **DER**) fue propuesto en 1976 y ha encontrado una amplia aceptación como instrumento para modelar el mundo real en el proceso de diseño de las bases de datos. El modelo opera con los conceptos de **entidad** y **relación** que estudiamos anteriormente.

Las **ocurrencias** de **entidades** se clasifican en distintas entidades, tales como "**empleado**", "**departamento**", etc. Existirá un predicado asociado con cada entidad que permitirá comparar si una ocurrencia arbitraria pertenece a una entidad dada. Las ocurrencias pueden pertenecer a más de una entidad, o sea, las entidades no son mutuamente disjuntas. Por ejemplo, una ocurrencia de la entidad "mujeres" también pertenece a la entidad "persona".

Se le llama relación a una asociación matemática entre n entidades.

$$\{ (e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n \}$$

Cada elemento de esa relación es una ocurrencia de relación (e_1, e_2, \dots, e_n) , donde las E_i y e_i no tienen que ser necesariamente diferentes. El rol de una entidad en una relación expresa la función que desempeña dicha entidad en la relación.

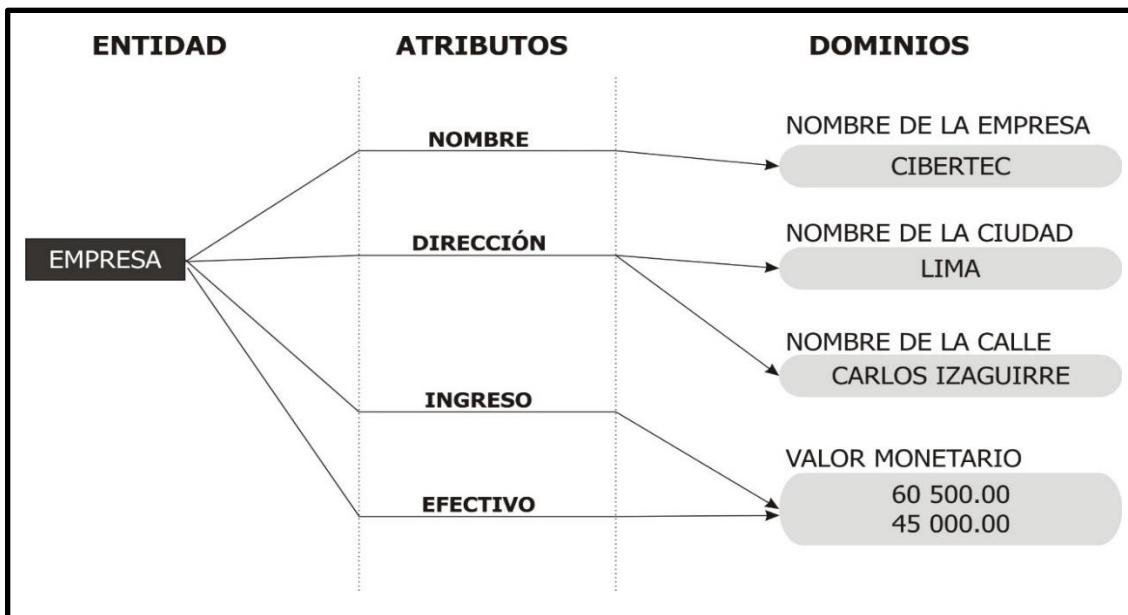
En la relación "matrimonio" definida entre ocurrencias de la entidad "persona", mejor dicho, "**matrimonio** = $\{(e_1, e_2) \mid e_1 \in \text{"persona"}, e_2 \in \text{"persona"}\}$ ", el primer elemento en el tuplo puede aparecer en el rol de "esposo" y el segundo, en el rol de "esposa".

Información adicional sobre una entidad (además de los predicados y las relaciones) se obtiene mediante los atributos asociados con la entidad. Ejemplos de valores que pueden tomar los atributos son: "rojo", "3", "Juan", etc. y ellos se clasifican en dominios mutuamente disjuntos, tales como "color", "edad", "nombre", etc. Un valor de un dominio puede ser equivalente a otro valor en un dominio diferente. Por ejemplo, "100" en el dominio "centímetros" es equivalente a "1" en el dominio "metros".

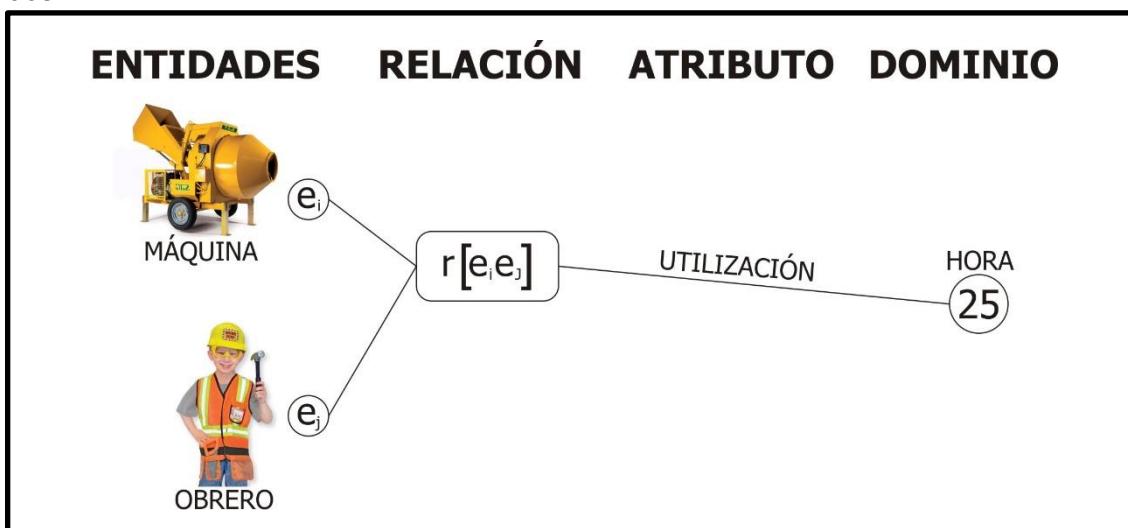
Un atributo se define en el modelo entidad – relación, como una función matemática que establece una correspondencia desde una entidad o relación hacia un dominio o un producto cartesiano de dominios:

$$\begin{aligned} \text{atrib}_1: E_i &\rightarrow D_{i1} \times D_{i2} \times \dots \times D_{in} \\ \text{atrib}_2: R_i &\rightarrow D_{i1} \times D_{i2} \times \dots \times D_{in} \end{aligned}$$

En la figura siguiente se muestran los atributos definidos para la entidad EMPRESA. El atributo NOMBRE hace corresponder a las ocurrencias de empresa con elementos del dominio NOMBRE DE EMPRESA. El atributo DIRECCIÓN establece una correspondencia desde la entidad EMPRESA hacia el par de dominios NOMBRE DE CIUDAD, NOMBRE DE CALLE. INGRESO Y EFECTIVO establecen ambos una correspondencia desde la entidad EMPRESA hacia el dominio VALOR MONETARIO. Nótese que un atributo se define siempre como una función, por lo que siempre hace corresponder a una ocurrencia dada con un único valor de una tupla, pues se define un producto cartesiano de dominios.



Las relaciones pueden también tener atributos. En la figura siguiente, el atributo **UTILIZACIÓN** define el número de horas que un obrero específico e_j usa una máquina e_i y constituye un atributo de la relación correspondiente. Él no es ni un atributo del **OBRERO** ni de la **MÁQUINA**, ya que su significado depende de la relación entre ellos dos.



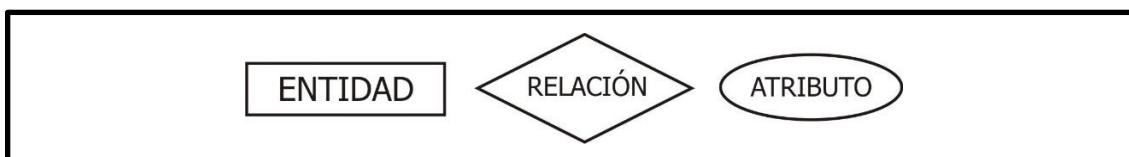
Es importante destacar las siguientes características de los atributos en este modelo:

- Los atributos sólo son correspondencias funcionales. Así, por ejemplo, si tenemos la entidad **AUTOMÓVIL** y el atributo **COLOR**, el hecho de que un auto pueda tener más de un color no se puede representar como un atributo en este modelo.
- El único hecho que puede ser registrado sobre los valores en este modelo es su pertenencia a un dominio. Si se desea representar otra propiedad, el atributo asociado tiene que ser convertido en una entidad. Por ejemplo, si queremos registrar la longitud de onda de cada color no podemos hacerlo en el modelo entidad - relación, sino convirtiendo el atributo **COLOR** en una entidad.

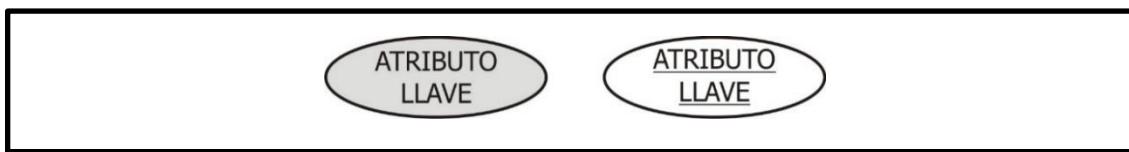
2.2.2. Diagrama Entidad-Relación

El modelo entidad-relación tiene asociada una representación gráfica denominada diagrama entidad-relación (**DER**).

En un **DER**, cada entidad se representa mediante un **rectángulo**, cada relación mediante un **rombo** y cada dominio mediante un **ovalo**. Mediante líneas se conectan las entidades con las relaciones, y de la misma manera igual que las entidades con los dominios, representando a los atributos.



“Los **atributos llaves** de las entidades se representan mediante un **ovalo relleno** o con el **nombre subrayado**”.



En ocasiones, una entidad no puede ser identificada únicamente por el valor de sus propios atributos. En estos casos, se utilizan conjuntamente las relaciones con los atributos para lograr la requerida identificación única. Estas entidades reciben el nombre de **entidades débiles** y se representan en el **DER** con un **doble rectángulo**.



El modelo entidad-relación restringe las relaciones a usar para identificar las entidades débiles a relaciones binarias de, a lo sumo, **1:n**. Así, por ejemplo, una ocurrencia de trabajador puede tener n ocurrencias persona-dependiente asociadas, donde, además, la existencia de una ocurrencia en la segunda entidad depende de la existencia de una ocurrencia que le corresponda en la primera entidad.

Por ejemplo, en el modelo que se representa en el **DER** de la figura, habrá personas dependientes de un trabajador sólo si ese trabajador existe. Para indicar esa dependencia en la existencia, se usa una flecha en el **DER**. La llave de una entidad débil se forma combinando la llave de la entidad regular que la determina con algún otro atributo o conjunto de atributos de la débil, que definan únicamente cada entidad débil asociada a una entidad regular dada (una entidad se denomina regular si no es débil).

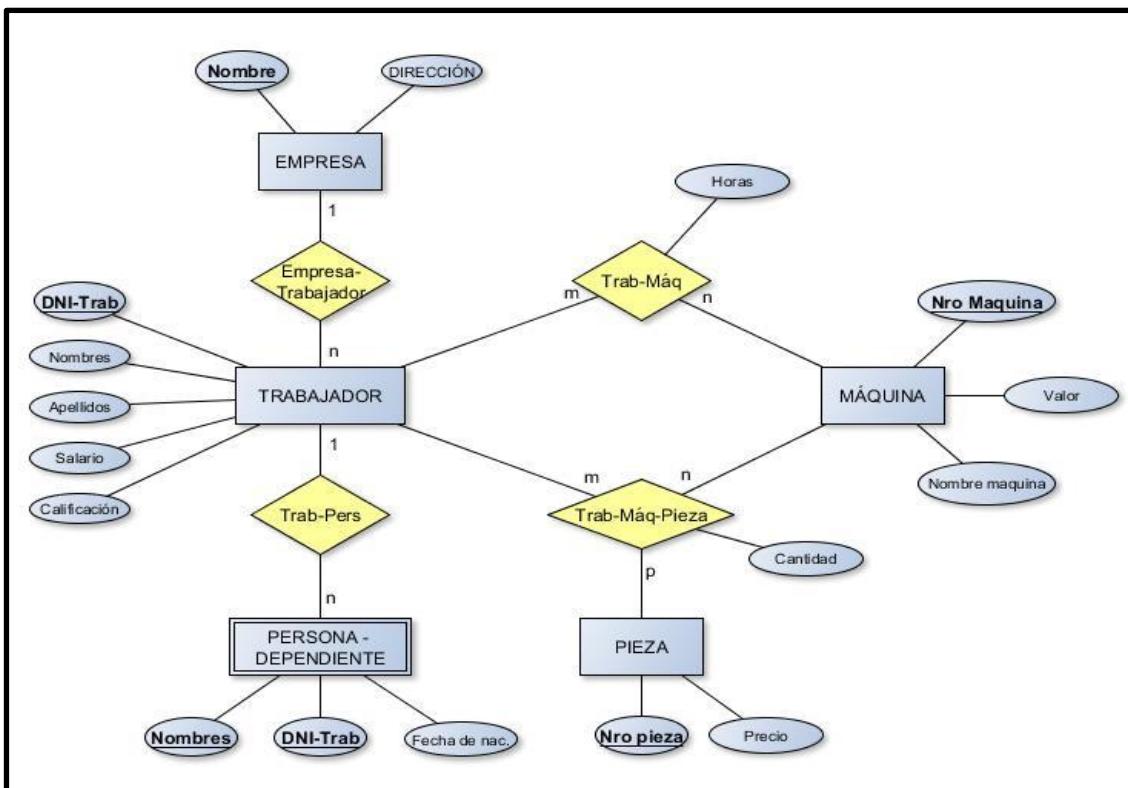


Figura 12: Diagrama Entidad Relación diseñado en yEd 3.14.2
Fuente.- Tomado desde la aplicación yEd

Para cada relación se determina su tipo (simple o complejo) y en el **DER** se escribe el tipo de correspondencia, por ejemplo:

- Una empresa puede tener varios (**n**) trabajadores asociados y un trabajador pertenece a una sola (**1**) empresa.
- En la relación **trab-máq-pieza**, un trabajador puede trabajar en **n** máquinas, produciendo **p** piezas, una pieza puede ser producida por **m** trabajadores en **n** máquinas y en una máquina pueden trabajar **m** trabajadores produciendo **p** piezas.
- **m**, **n** y **p** no identifican un número específico, sino solamente el tipo de correspondencia que se establece en la relación (muchos).

Aunque en el **modelo entidad-relación** se define que “**La llave de una relación es la combinación de las llaves de todas las entidades asociadas**”, es conveniente, desde ahora, analizar profundamente esto.

En una relación de **MUCHOS A MUCHOS (n:m)**, efectivamente, “**La llave de la relación está formada por las llaves de las entidades que participan en la relación**”, pues, como a cada ocurrencia de una de las entidades le corresponden varias ocurrencias de la otra entidad y viceversa, es preciso utilizar la identificación de cada una de las entidades que participan en la relación para referirse a una sola ocurrencia de cada una de ellas y, así, referirse a una ocurrencia de la relación.

Por ejemplo, en la relación **trab-máq** la llave será **DNI-Trab, Nro-Máquina**. Pero en una relación de muchos a uno (**m:1**), la llave de la relación es la llave de la entidad del extremo muchos (**m**), pues a cada ocurrencia de esa entidad le corresponde sólo una

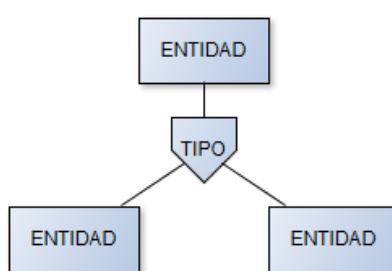
ocurrencia de la entidad del otro extremo, por lo que, con la llave de la entidad del extremo muchos está perfectamente determinada también una única ocurrencia de la entidad del extremo 1. Por ejemplo, en la relación Empresa-trabajador la llave será DNI-trab.

De modo similar, en una relación de **UNO A UNO (1:1)** la llave de la relación está formada por la llave de cualquiera de las dos entidades que participan, pues “**A una ocurrencia de una de ellas le corresponde sólo una ocurrencia de la otra y viceversa**”, por lo que, con la llave de una de las entidades está perfectamente determinada también una única ocurrencia de la otra entidad. Por ejemplo, si en un centro de trabajo un trabajador, que se identifica con DNI-Trab, es jefe de un piso, que se identifica con numpiso, y un piso tiene un jefe, es decir, que la relación es de 1:1, entonces, la llave de la relación puede ser o DNI-trab o numpiso.

Una entidad se puede relacionar consigo misma. A estas relaciones se les llama, usualmente, **recursivas o cílicas**. Es posible extender la capacidad semántica del modelo entidad-relación aplicando sobre sus objetos básicos (entidad y relación) diferentes operaciones, tales como:

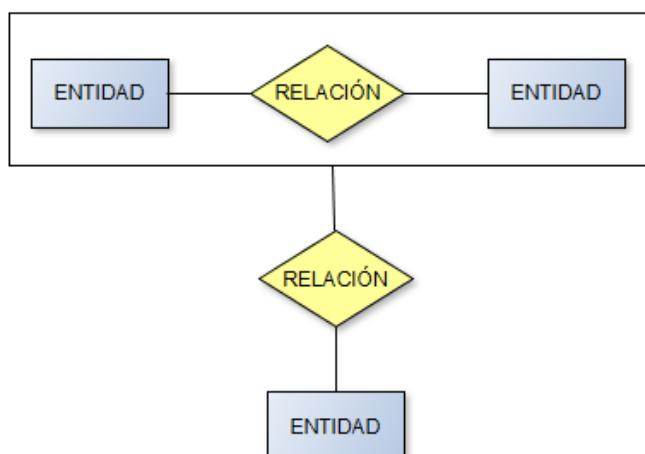
- **GENERALIZACIÓN**

Permite formar una nueva entidad, mediante la unión de otras entidades. El proceso inverso se denomina especialización y divide una entidad en cierto número de otras entidades.



- **AGREGACIÓN**

Construye una nueva entidad sobre la base de una relación. A las entidades, relaciones y conjuntos definidos hasta ahora les llamaremos tipos básicos para distinguirlos de los nuevos tipos de datos que se obtendrán con las operaciones anteriores.



Veamos cada una de las operaciones:

2.2.2.1 Generalización / Especialización

Si T_1, T_2, \dots, T_n son entidades (que pueden a su vez ser resultado de una generalización), la generalización define una nueva entidad T con el siguiente significado:

$$T = \{ t \mid t \in T_i, 1 \leq i \leq n \}$$

Mejor dicho, para cada ocurrencia t en T existe, al menos, un conjunto T_i que contiene a esa ocurrencia. Por ejemplo, en el **DER** anterior, puede ser necesario distinguir los trabajadores de una empresa de acuerdo a su ocupación como obreros, dirigentes y administrativos. Esto no puede ser representado en el modelo que está representado en el **DER** de la figura anterior y sólo a través del hecho de que la entidad "obrero", por ejemplo, es siempre (mejor dicho, en todo momento) un subconjunto de la entidad "trabajador", podemos deducir cierta clase de dependencia entre los dos tipos

Usando la generalización podemos obtener un nuevo diagrama como se muestra parcialmente en la figura siguiente:

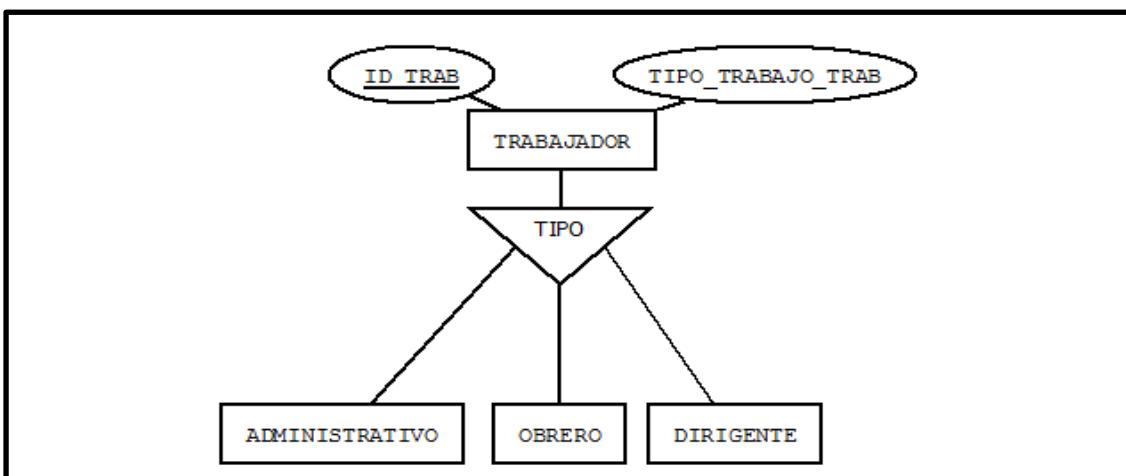


Figura 13: Generalización de trabajador
Fuente.- Tomado desde la aplicación diaW

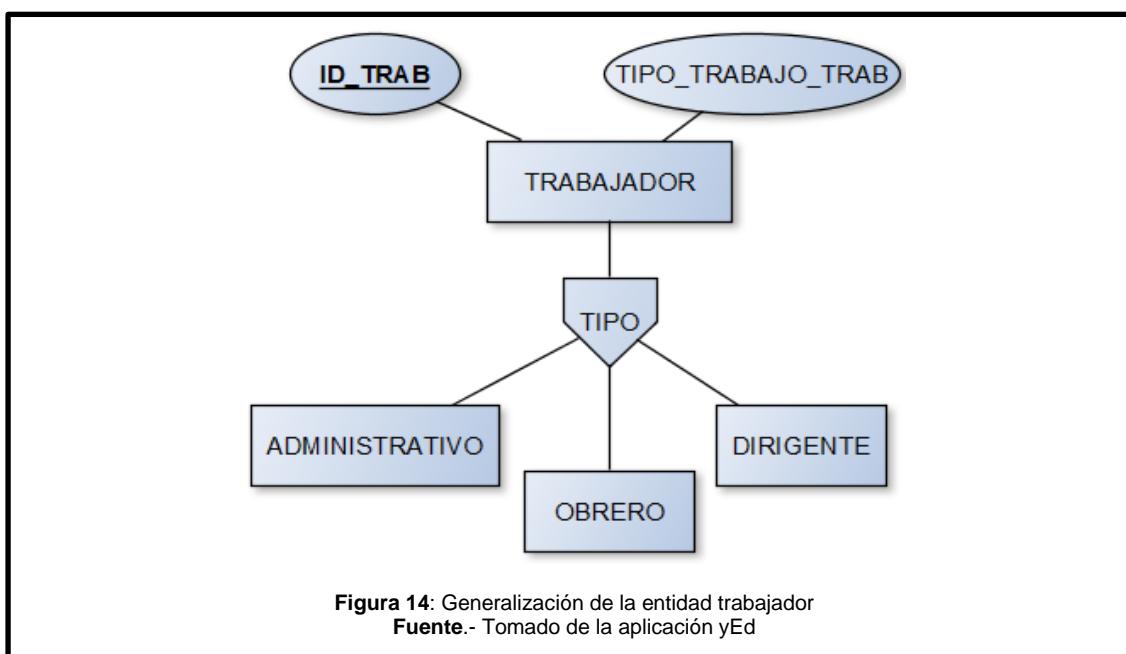


Figura 14: Generalización de la entidad trabajador
Fuente.- Tomado de la aplicación yEd

Nótese que hemos introducido un nuevo atributo para la entidad trabajador. Este atributo nos permite distinguir entre los miembros de diferentes clases de trabajadores.

Si tenemos una entidad **TRABAJADOR** y queremos usar la operación de Especialización como inversa a la generalización, tenemos que especificar "roles" en el modelo, dicho de otra forma, reglas que definan cuándo una ocurrencia de **TRABAJADOR** pertenece a uno u otro componente de la entidad. Entonces, la representación de esta operación en el **DER** se generaliza como se muestra en la siguiente figura:

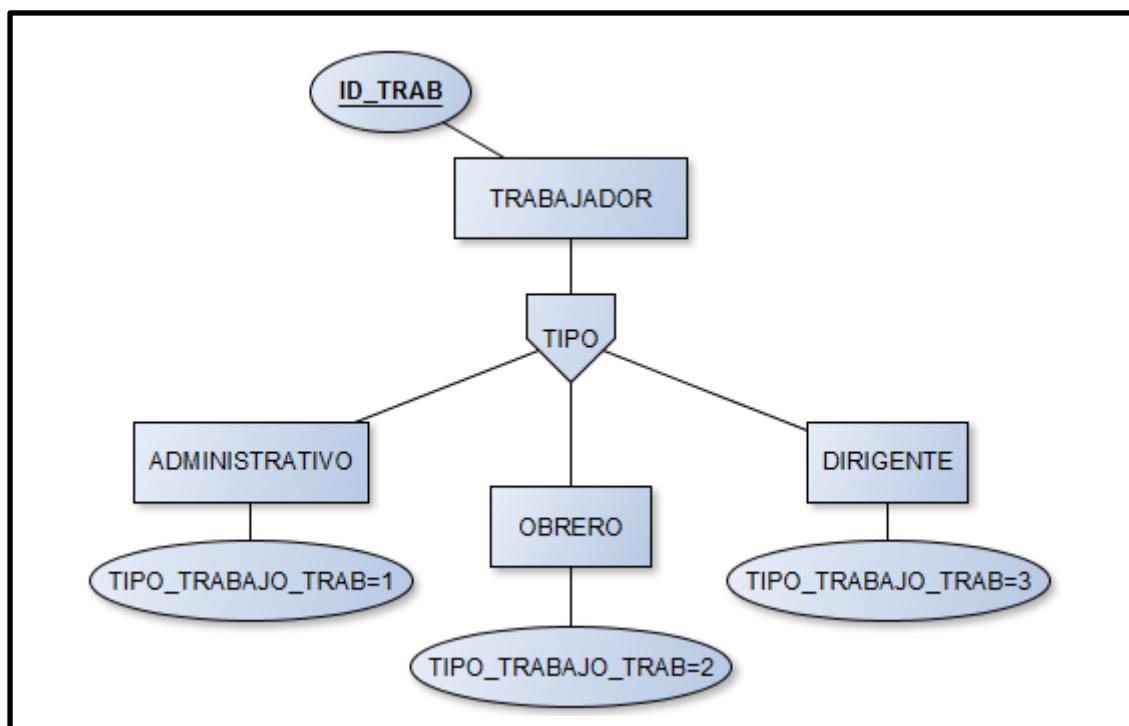


Figura 15: Especialización de la entidad Trabajador
Fuente.- Tomado desde la aplicación yEd

Si para cada ocurrencia de la entidad **Trabajador** nosotros podemos siempre deducir a cuál entidad componente pertenece usando alguna propiedad ya representada, entonces, no es necesario introducir un nuevo atributo **Tipo de Trabajo**.

Las reglas que definen la especialización de una entidad se denominan "caracterizaciones". Por ejemplo, **Tipo de Trabajo = 1** es la caracterización de la entidad Administrativo dentro de la entidad Trabajador.

En una Generalización / Especialización, los atributos y relaciones de la entidad "generalizada" son heredados por las entidades componentes (entidades especializadas). La llave de una especialización es la llave de la generalización.

Además, se pueden definir nuevos atributos y relaciones para cada entidad especializada. Por ejemplo, la relación Obrero-Máquina se define ahora sólo para la entidad especializada Obrero, componente de la entidad generalizada Trabajador:

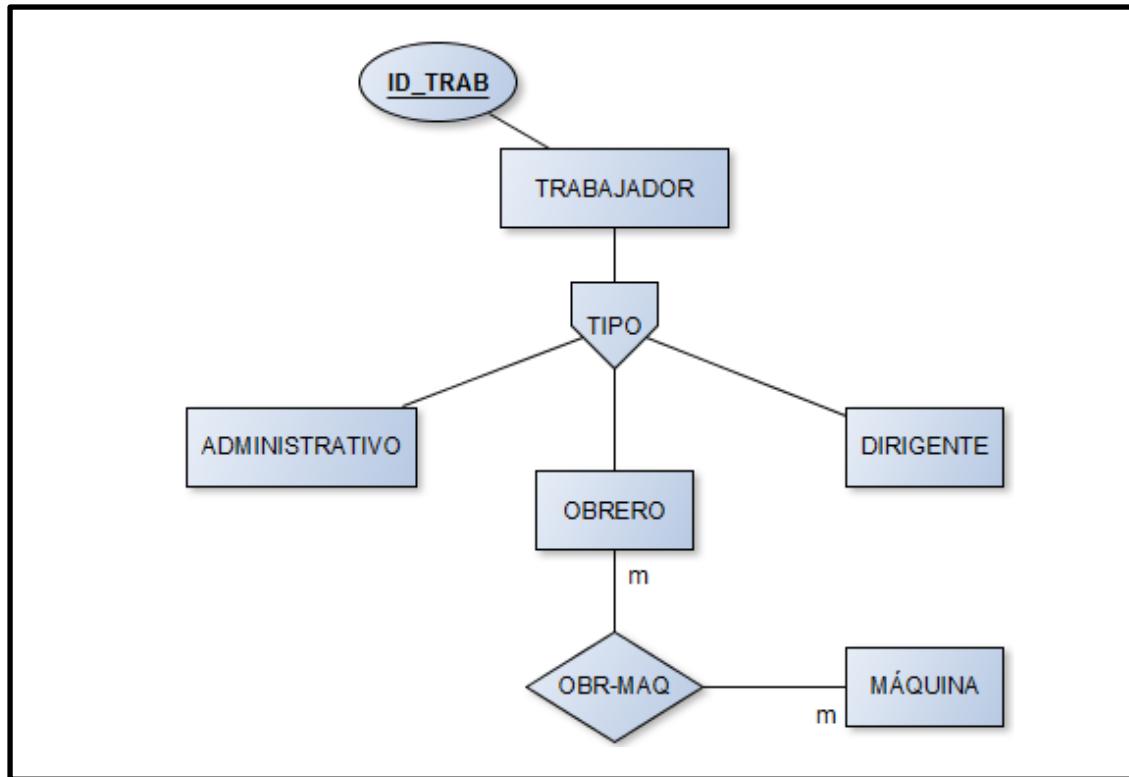


Figura 16: Generalización de la entidad Trabajador
Fuente.- Tomado desde la aplicación yEd

Si bien es cierto que, según lo visto anteriormente, las operaciones de Generalización y Especialización pueden denotarse de modo diferente, no es menos cierto que, con la notación empleada para la generalización, pueden expresarse las entidades generalizadas y especializadas perfectamente y es ésta la empleada normalmente.

Es importante agregar algo más a lo visto hasta ahora para poder expresar las siguientes situaciones que se presentan:

- Las ocurrencias de las especializaciones pueden abarcar o no el universo de las ocurrencias de la generalización; es decir, la totalidad de las ocurrencias de la generalización puede o no estar contenidas en alguna o algunas de las especializaciones. Por lo tanto, las especializaciones pueden ser totales (T) o parciales (P).
- Una ocurrencia de la generalizada puede o no estar en más de un conjunto T_i , o lo que es lo mismo, la intersección entre algunos de los conjuntos T_i puede o no ser vacía. Es decir, las especializaciones pueden ser solapadas (S) o disjuntas (D).

Es por ello que, en el **DER**, se añade en cada generalización, entre paréntesis, la especificación:

- (T, S): indicando que la especialización realizada es total y solapada
- (T, D): indicando que la especialización realizada es total y disjunta
- (P, S): indicando que la especialización realizada es parcial y solapada
- (P, D): indicando que la especialización realizada es parcial y disjunta

Entonces, el ejemplo visto anteriormente quedaría de la siguiente manera:

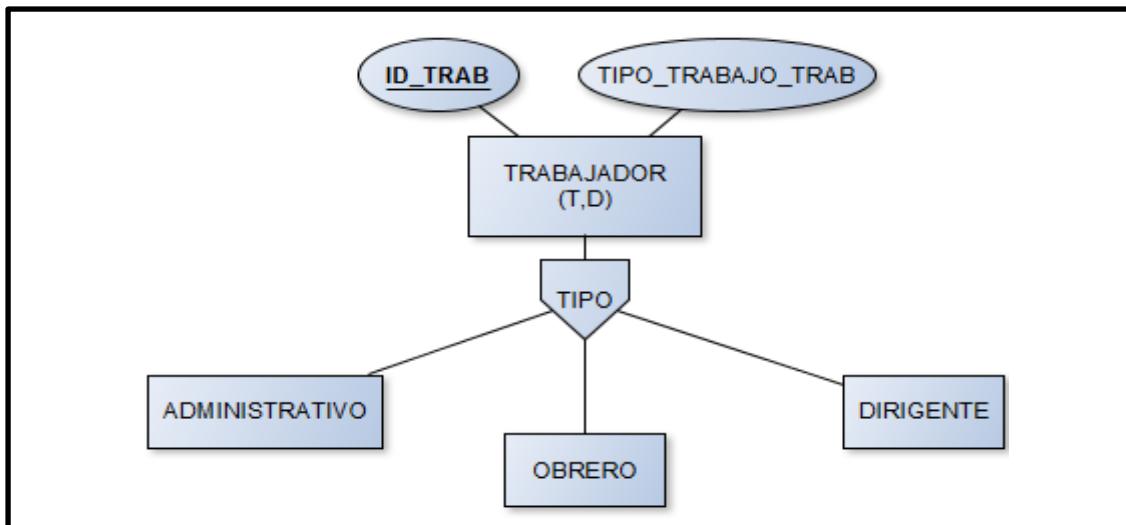


Figura 17: Generalización de la entidad Trabajador
Fuente.- Tomado desde la aplicación yEd

Donde:

- **T(total)**: ya que todo trabajador en el ejemplo es administrativo o dirigente u obrero.
- **D(disjunto)**: Un trabajador pertenece solo a una de las especializaciones.

Otro ejemplo de Generalización/Especialización podría ser el caso de **ESTUDIANTE**, **PRACTICANTE** y **BECADO**. Un **PRACTICANTE** es un caso especial de **ESTUDIANTE**. Lo mismo ocurre con **BECADO**. Pero un **PRACTICANTE** también puede ser **BECADO**. Hay muchos Estudiantes que no son **PRACTICANTES** ni **BECADOS**. Obviando los atributos en el **DER**, esta situación se representaría del modo siguiente:

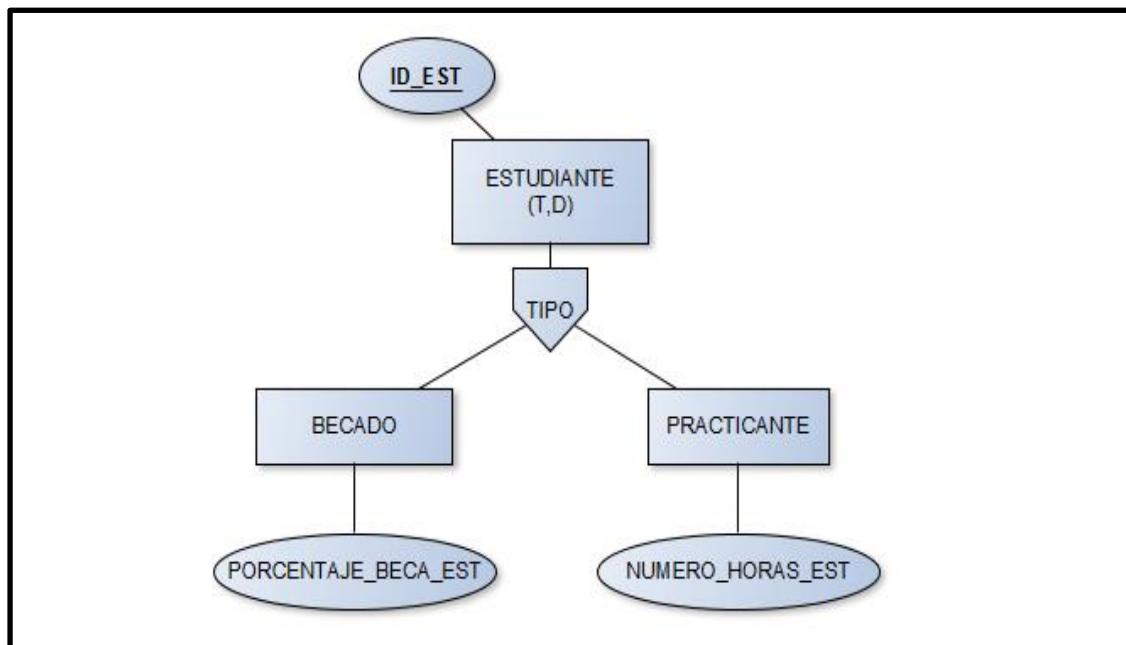


Figura 18: Generalización de la entidad Estudiante
Fuente.- Tomado desde la aplicación yEd

2.2.2.2 Agregación

Obsérvese en el ejemplo que representa la situación de la producción en las empresas, que la relación **Trab-Maq-Pieza** representa la idea de que una actividad en la empresa se describe en términos de "**un obrero en alguna máquina produce una pieza dada en alguna cantidad específica**".

Sin embargo, la misma situación puede ser vista de forma algo diferente. En la empresa, las máquinas pueden estar asignadas a los obreros y estos "**equipos**" producir piezas en cierta cantidad.

En el modelo entidad-relación original esta situación no hubiera podido ser modelada correctamente, ya que una relación no puede relacionarse con otra relación o entidad.

Con la operación de Agregación esta situación se resuelve fácilmente, tal y como se muestra en la figura siguiente:

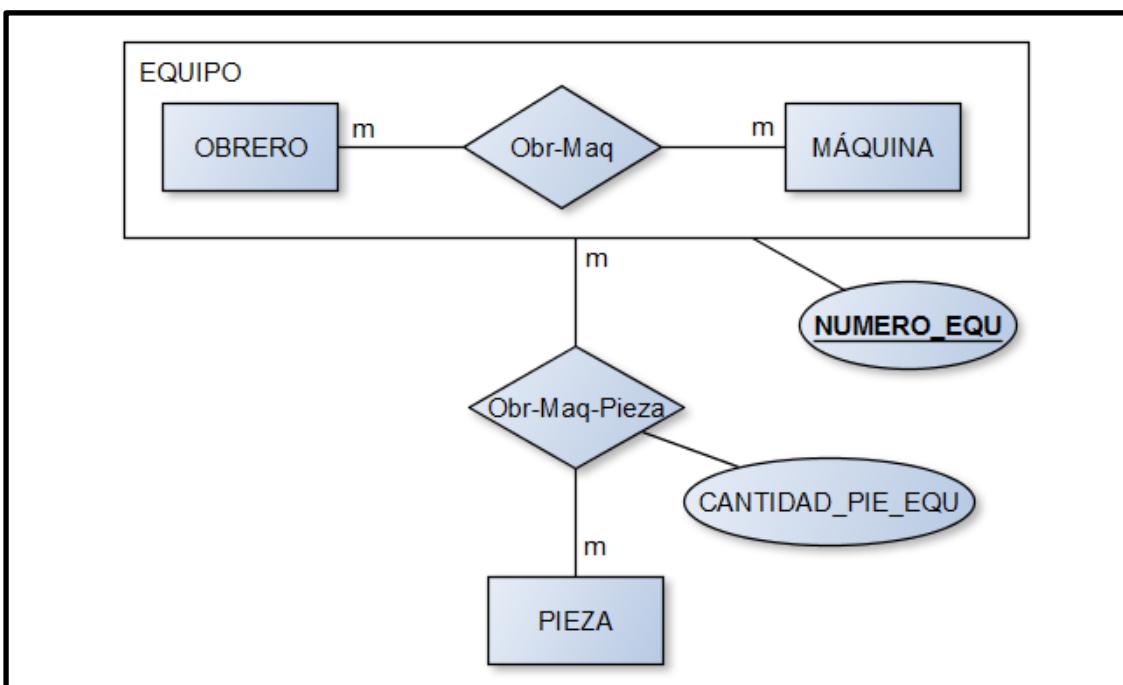


Figura 19: Implementación de la Agregación Equipo
Fuente.- Tomado desde la aplicación yEd

La agregación se define de la siguiente forma:

Si T_1, T_2, \dots, T_n son entidades, la operación define una nueva entidad T con el significado siguiente:

$$T = \{t \mid \exists t_1, t_2, \dots, t_n (t_1 \in T_1 \wedge t_2 \in T_2 \dots \wedge t_n \in T_n \wedge (t_1, t_2, \dots, t_n) = t)\}$$

Mejor dicho, las nuevas ocurrencias se forman como tuplas de ocurrencias de las entidades componentes. Para que la operación tenga sentido, las entidades T_1, T_2, \dots, T_n tienen que formar parte en alguna relación común y esa relación siempre será incluida en la representación de la entidad generada (entidad agregada).

A la nueva entidad se le pueden asignar atributos. También puede tomar parte en cualquier relación. Otro ejemplo de Agregación se muestra a continuación:

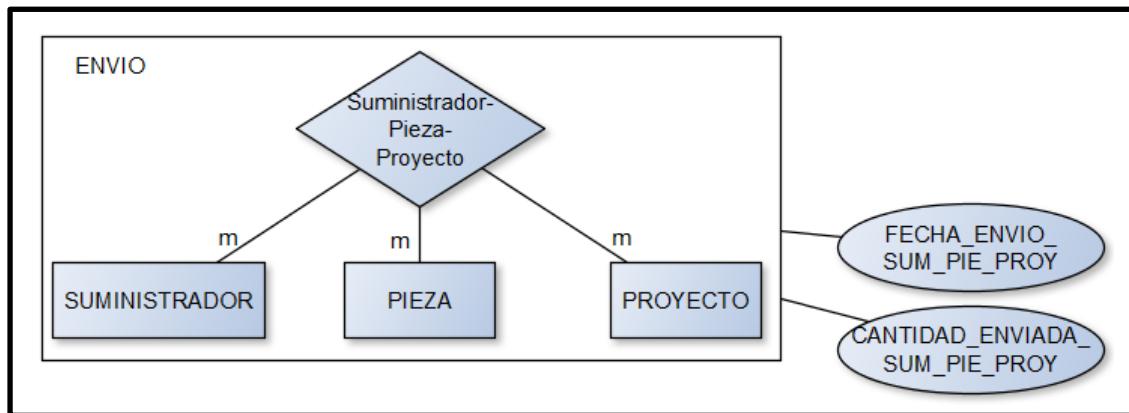


Figura 20: Implementación de la agregación Envío
Fuente.- Tomado desde la aplicación yEd

La nueva entidad **ENVÍO** se define como una agregación de tres entidades: **Suministrador**, **Pieza** y **Proyecto** con los nuevos atributos Fecha del Envío y Cantidad Enviada.

Hay una diferencia importante entre estos dos atributos: está claro que la **Fecha del Envío** no puede pertenecer a ninguna de las entidades componentes; sin embargo, la Cantidad Enviada se refiere claramente a las piezas. Diremos entonces, que la Cantidad Enviada es una "caracterización" de la entidad **PIEZA** con respecto al **ENVÍO**.

La llave de una entidad agregada es la llave de la relación que la origina; excepto en el caso en que se defina un identificador (llave) para ella. En el ejemplo anterior del **ENVÍO**, la llave de la entidad agregada **ENVÍO** está formada por la llave del suministrador, más la llave de la pieza, más la llave del proyecto.

Pero la situación del ejemplo podría tener la variante que se muestra en la siguiente figura, donde se ha definido un número de envío (**NUM_ENV**) que identifica al envío y que, por lo tanto, pasa a ser la llave de la entidad agregada:

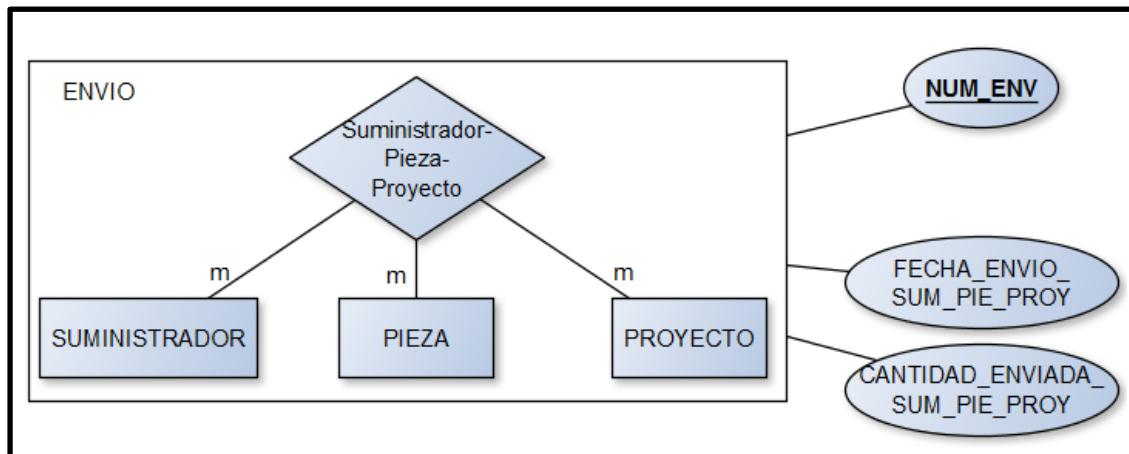


Figura 21: Implementación de la agregación Envío con sus atributos
Fuente.- Tomado desde la aplicación yEd

Para el modelo entidad - relación, incluyendo las dos operaciones estudiadas, pueden plantearse una serie de restricciones de integridad:

- Al aplicar la generalización/especialización, una entidad puede pertenecer a una jerarquía de diferentes entidades. Por ejemplo, las entidades **PERSONA**, **TRABAJADOR**, **OBRERO** forman una jerarquía de entidades, sucesivamente más especializadas. Entonces, una entidad existente en un nivel dado, tiene que existir en todos los niveles superiores. De forma inversa, si una entidad se elimina de un conjunto en un nivel dado, debe ser eliminada también en los niveles más bajos.
- La agregación constituye una entidad agregada sobre la base de una relación, por lo que dicha entidad se comportará de forma similar a como se comporta la relación. Entonces, para que una ocurrencia de la agregación exista, deben existir las ocurrencias de todas las entidades que toman parte en la relación. Lo inverso no tiene que ocurrir necesariamente, ya que, por ejemplo, en el caso visto del **ENVÍO** pueden existir suministradores que no abastecen a ningún proyecto, sino que se registran como tales, porque en determinado momento pudieran estar activos. Desde luego, si la política de la organización es tal que un suministrador se considera como tal solo si realmente suministra piezas a algún proyecto, entonces, la existencia de, al menos, una ocurrencia de la entidad agregada **ENVÍO** para un suministrador es indispensable para la existencia de la ocurrencia de ese suministrador en la entidad **SUMINISTRADOR**.

Finalmente, es importante señalar que en un **DER** pueden combinarse todos los elementos y operaciones explicadas anteriormente para representar adecuadamente un caso de la realidad, de modo que, por ejemplo, en una agregación puede participar una entidad débil o una generalizada; una especialización puede ser, a su vez, la generalización de otras especializaciones; una entidad débil puede ser, a su vez, una generalizada o ser débil de una generalizada, etc.

2.2.2.3 Entidad débil

Una entidad débil es aquella cuya existencia depende de otra entidad llamada fuerte. A manera de ejemplo, se muestra a continuación un **DER** que representa al alumno de Medicina como entidad débil de grupo, ya que el alumno se identifica por un número consecutivo dentro del grupo al que pertenece y que, por lo tanto, puede repetirse en distintos grupos, por lo que, para identificar a un alumno, es preciso decir “**el alumno número n del grupo código g**”. A la vez, la entidad alumnomedicina es una generalización que tiene como especializaciones a los alumnos de 3er. ciclo o menos y a los alumnos de 4to. ciclo o más.

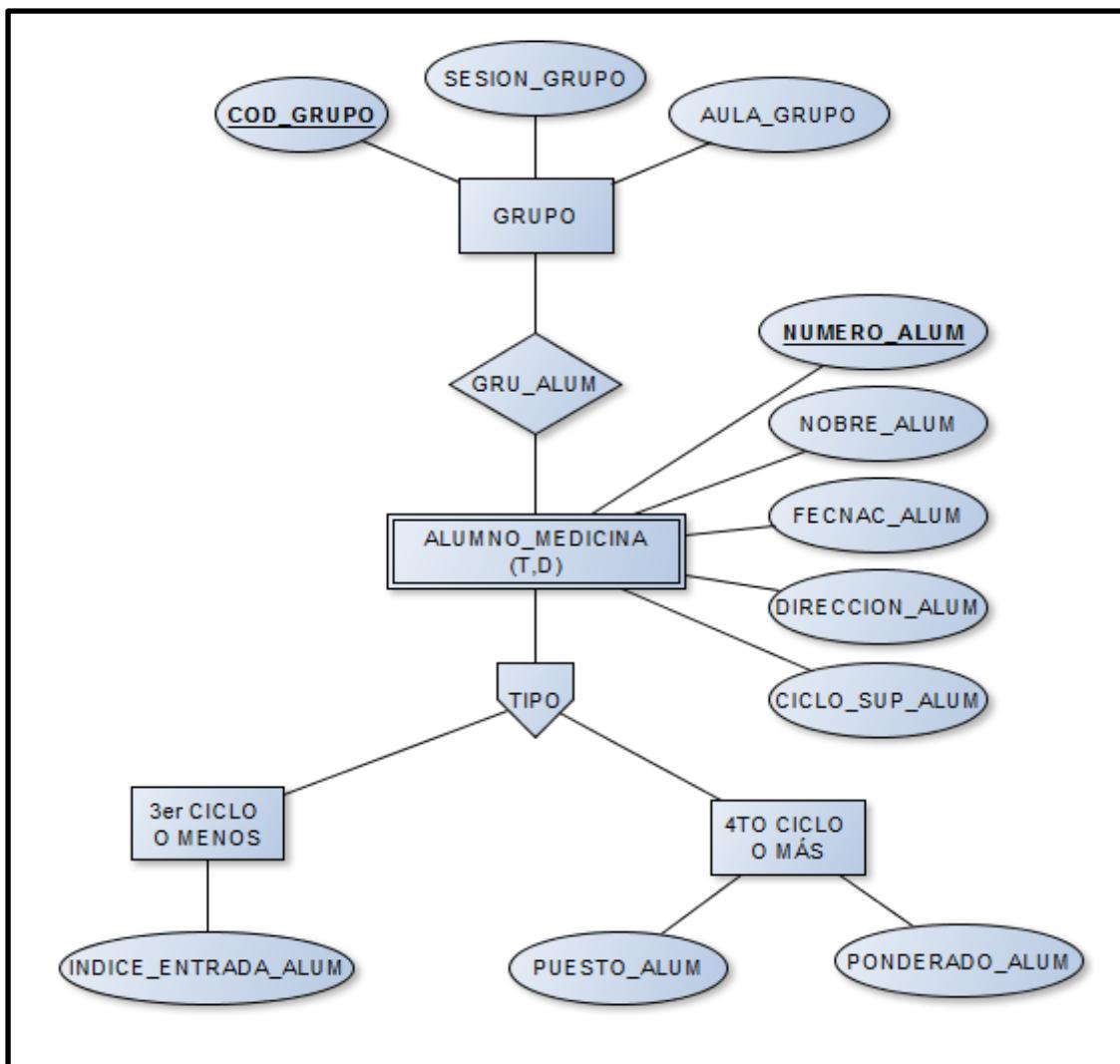


Figura 22: Representación de la entidad débil
Fuente.- Tomado desde la aplicación yEd

2.2.2.4 Relación Recursiva

Relación donde la entidad se relaciona consigo mismo. Las multiplicidades se colocan respecto a los nombres de rol.

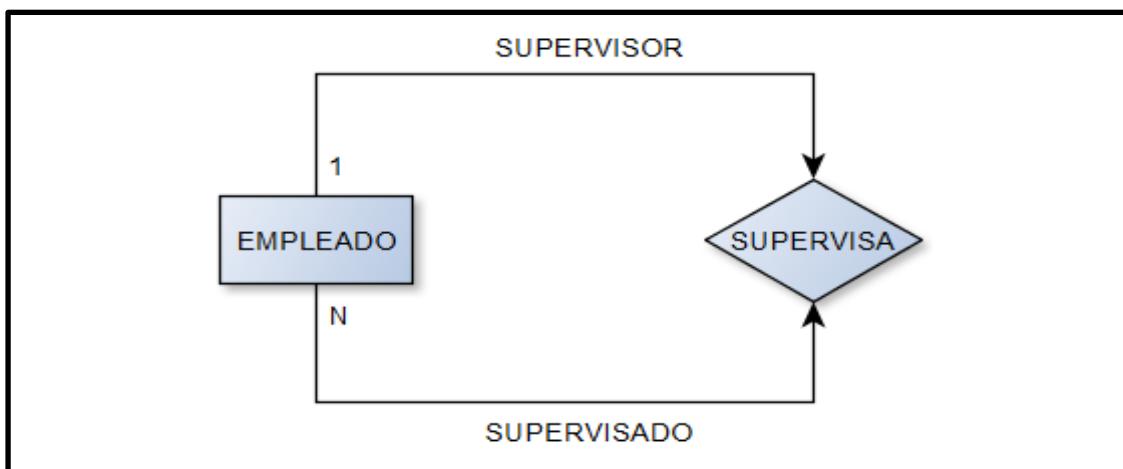


Figura 23: Representación de la recursividad supervisor
Fuente.- Tomado desde la aplicación yEd

2.3. 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.

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
		-922.337.203.685.477.5808 a 922.337.203.685.477.5807
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 23:59:59.9999999
	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	00:00:00.0000000 a 23:59:59.9999999
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.

2.4. Sentencias DDL para la 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.

2.4.1. 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.

2.4.2. 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 (COLUMNA1,COLUMNA2)
GO
```

- Agregar una llave foránea a la tabla:

```
ALTER TABLE NOMBRE_TABLA1
ADD FOREING KEY (COLUMN) REFERENCES NOMBRE_TABLA2
GO
```

- Eliminar una columna de la tabla:

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

- Modificar una columna a la tabla:

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

2.4.3. 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
```

2.5. Integridad Referencial

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:

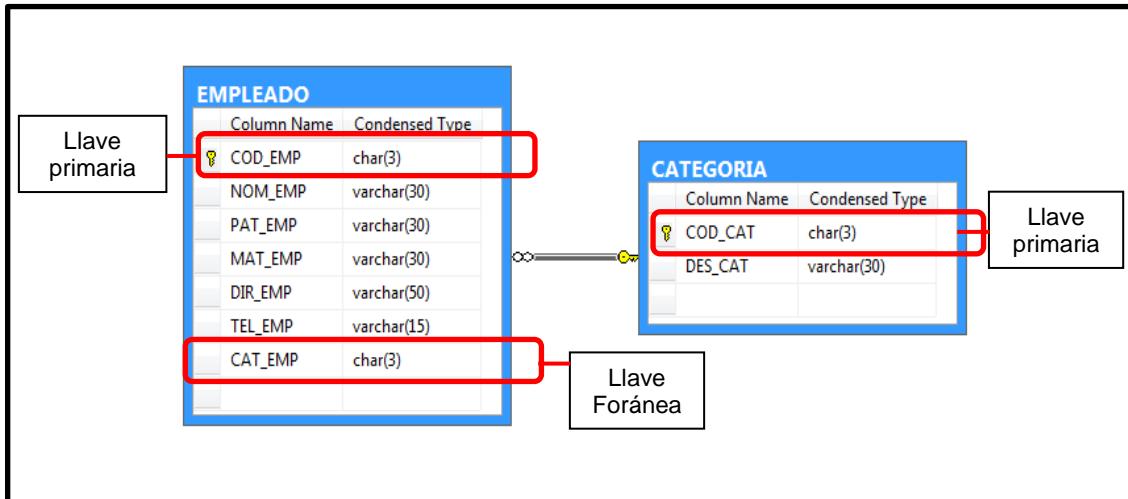
- En este tipo de relación, una fila de la tabla A puede tener varias columnas coincidentes en la tabla B.
- Pero una fila de la tabla B sólo puede tener una fila coincidente en la tabla A.

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”.



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”.

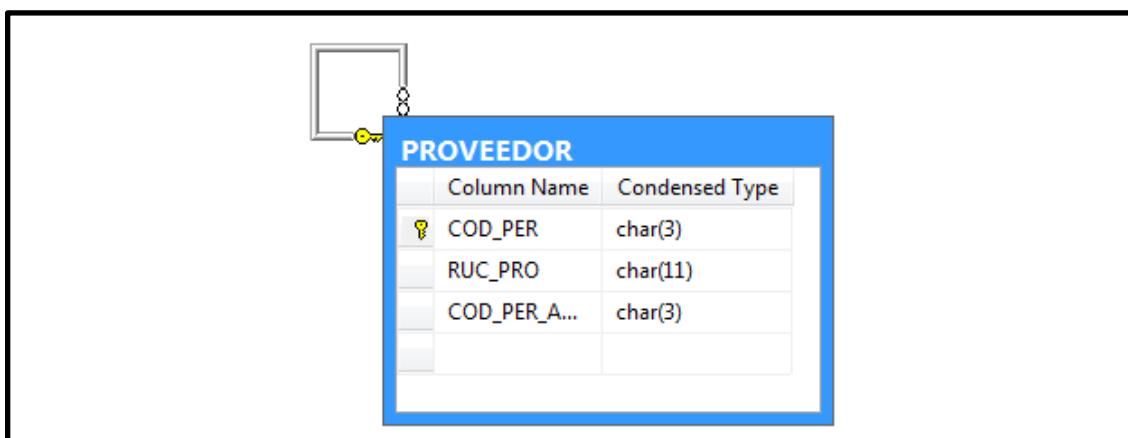


Figura 24: Representación de la recursividad de la tabla Proveedor en SQL Server
Fuente.- Tomado desde SQL Server 2014

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.

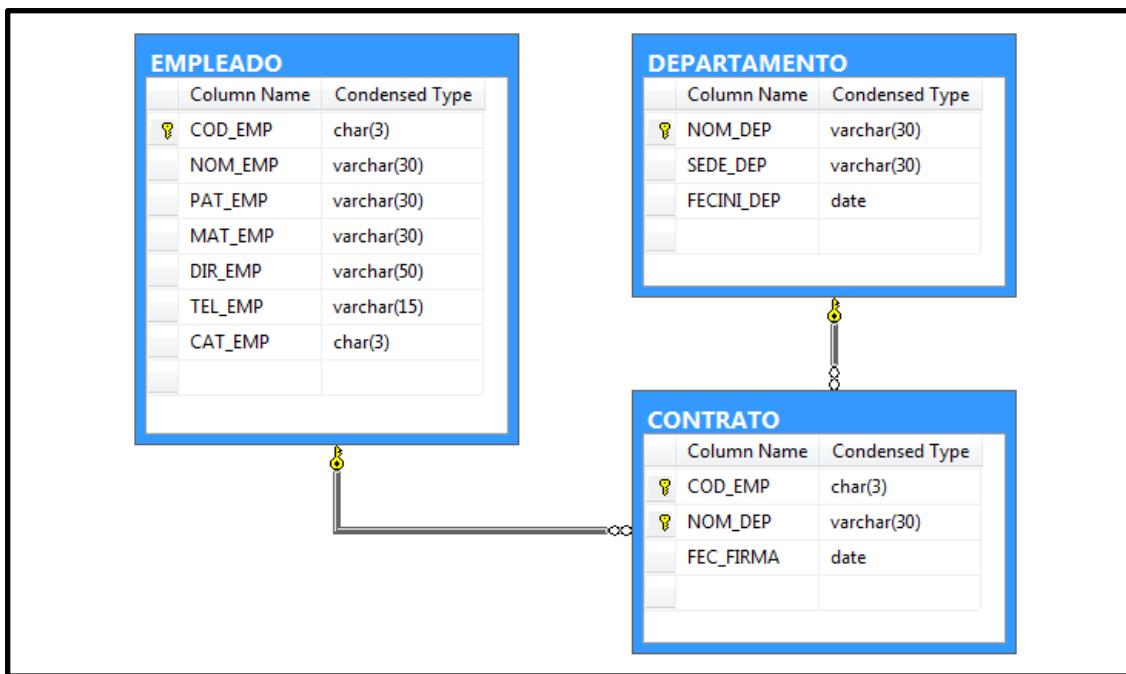


Figura 25: Representación de muchos a muchos en SQL

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.
- Aisljar 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.

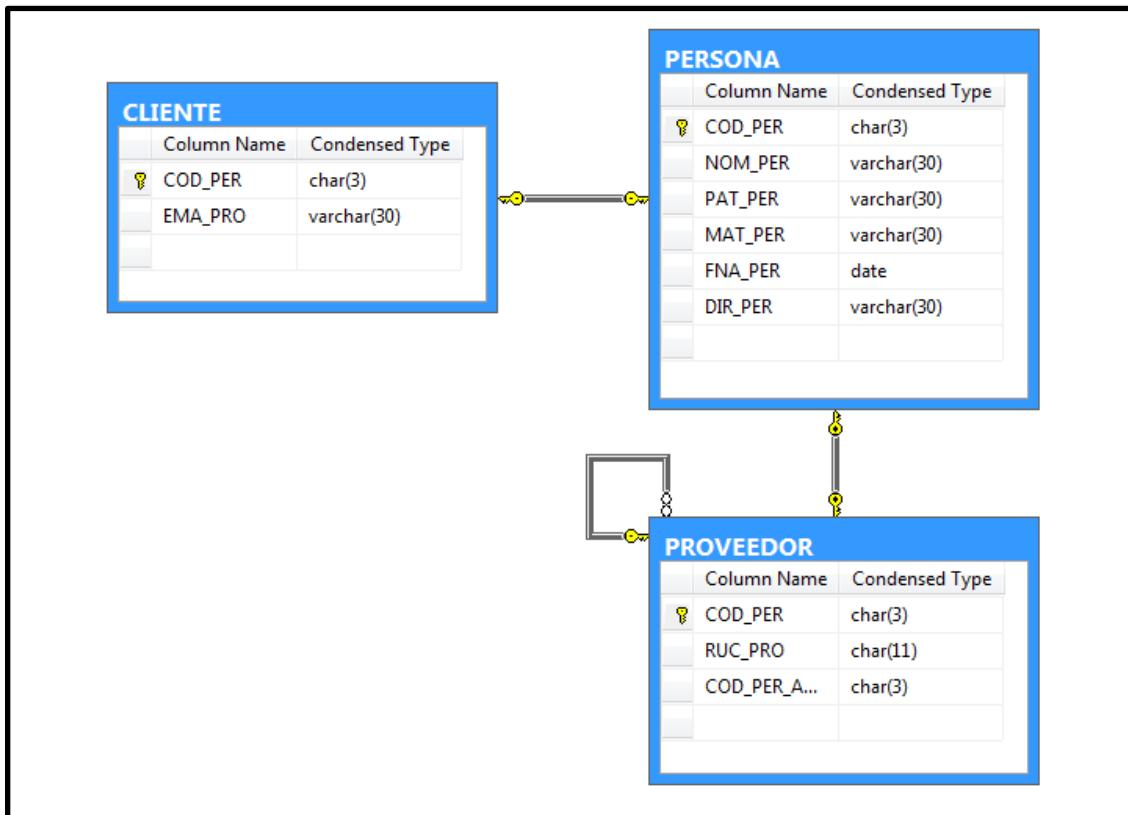


Figura 26: Representación uno a uno en SQL Server
Fuente.- Tomado desde SQL Server 2014

2.5.1. 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
```

2.5.2. 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 **FOREIGN KEY** garantiza que los datos ingresados en una tabla se encuentre 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(
    COLUMNA1  TIPO NOT NULL PRIMARY KEY,
    COLUMNA2  TIPO NULL,
    COLUMNA3  TIPO NOT NULL FOREIGN KEY REFERENCES NOMBRE_TABLA2
)
GO
```

Asignar Foreign Key al finalizar la especificación de las columnas

```
CREATE TABLE NOMBRE_TABLA1(
    COLUMNA1 TIPO NOT NULL PRIMARY KEY,
    COLUMNA2 TIPO NULL,
    COLUMNA3 TIPO NOT NULL,
    FOREIGN KEY (COLUMNA) REFERENCES NOMBRE_TABLA2(COLUMNA),
    FOREIGN KEY (COLUMNA) REFERENCES NOMBRE_TABLA3(COLUMNA)
)
GO
```

Asignar Foreign Key modificando la tabla

```
ALTER TABLE NOMBRE_TABLA1
    ADD FOREIGN KEY (COLUMNA1) REFERENCES NOMBRE_TABLA2
GO
```

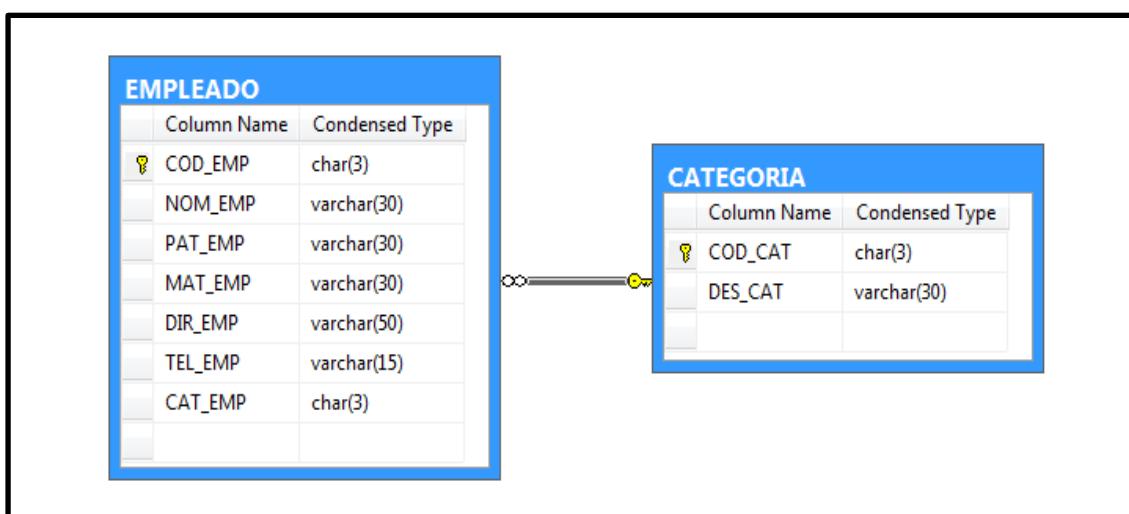
Asignar varios Foreign Key modificando la tabla

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

Asignar Foreign Key para una recursividad

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

Si tenemos el siguiente diagrama, mostraremos las diferentes formas de asignar foráneos 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

```


2.6. Implementación de una base de datos en SQL Server 2014

2.6.1. Implementación de una base de datos en SQL Server 2014

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

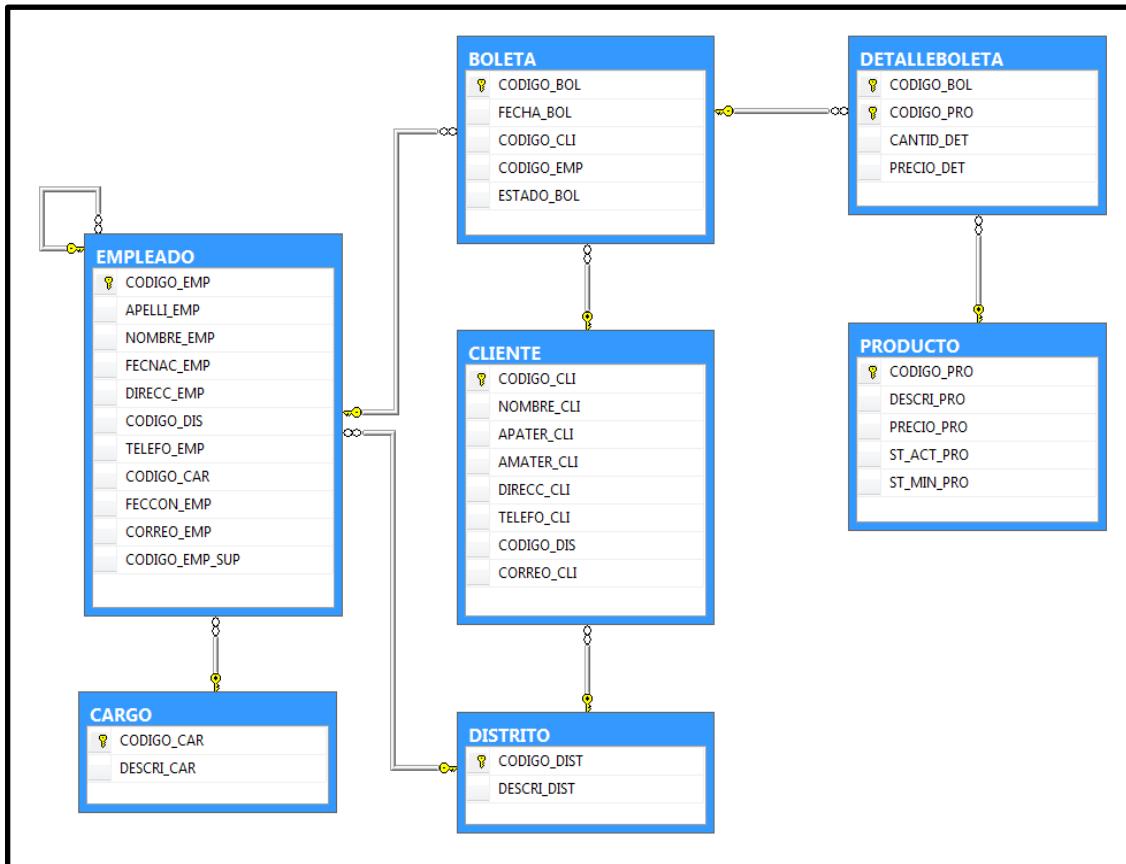


Figura 27: Diagrama de base de datos “Tienda”
Fuente.- Tomado desde SQL Server 2014

Usando **TRANSACT/SQl**, cree las siguientes bases de datos:

1. Cree la base de datos **BD_TIENDA**
2. Active la base de datos **BD_TIENDA**
3. Cree las tablas mostradas
4. Agregue las **llaves Primarias** (ADD PRIMARY KEY)
5. 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_TIENDA') IS NOT NULL
BEGIN
    DROP DATABASE BD_TIENDA
END
GO

--CREANDO LA BASE DE DATOS
CREATE DATABASE BD_TIENDA
GO

--ACTIVANDO LA BASE DE DATOS
USE BD_TIENDA
GO

--CREANDO LAS TABLAS
CREATE TABLE DISTRITO(
    CODIGO_DIST      CHAR(3)      NOT NULL  PRIMARY KEY,
    DESCRIBE_DIST   VARCHAR(50)
)
GO

CREATE TABLE PRODUCTO(
    CODIGO_PRO      CHAR(4)      NOT NULL  PRIMARY KEY,
    DESCRIBE_PRO    VARCHAR(40) NOT NULL,
    PRECIO_PRO      MONEY       NOT NULL,
    ST_ACT_PRO     INT         NOT NULL,
    ST_MIN_PRO     INT         NOT NULL
)
GO

CREATE TABLE CLIENTE(
    CODIGO_CLI      CHAR(6)      NOT NULL  PRIMARY KEY,
    NOMBRE_CLI      VARCHAR(50) NOT NULL,
    APATER_CLI     VARCHAR(50) NOT NULL,
    AMATER_CLI     VARCHAR(50) NOT NULL,
    DIRECC_CLI     VARCHAR(50) NULL,
    TELEFO_CLI     VARCHAR(9)  NULL,
    CODIGO_DIS     CHAR(3)      NOT NULL REFERENCES DISTRITO,
    CORREO_CLI     VARCHAR(50) NULL
)
GO

CREATE TABLE CARGO(
    CODIGO_CAR      INT         NOT NULL  PRIMARY KEY,
    DESCRIBE_CAR   VARCHAR(30) NOT NULL
)
GO

CREATE TABLE EMPLEADO (
    CODIGO_EMP      INT         NOT NULL  PRIMARY KEY,
    APELLI_EMP     VARCHAR(50) NOT NULL,
    NOMBRE_EMP     VARCHAR(50) NOT NULL,
    FECNAC_EMP    DATE        NOT NULL,
    DIRECC_EMP    VARCHAR(60) NOT NULL,
```

```
CODIGO_DIS      CHAR(3)      REFERENCES DISTRITO,  
TELEFO_EMP     VARCHAR(15)   NULL,  
CODIGO_CAR     INT          REFERENCES CARGO,  
FECCON_EMP    DATE         NOT NULL,  
CORREO_EMP    VARCHAR(35)   NULL,  
CODIGO_EMP_SUP INT          REFERENCES EMPLEADO  
)  
GO  
  
CREATE TABLE BOLETA(  
    CODIGO_BOL      CHAR(6)      NOT NULL PRIMARY KEY,  
    FECHA_BOL      DATE        NOT NULL,  
    CODIGO_CLI      CHAR(6)      NOT NULL REFERENCES CLIENTE,  
    CODIGO_EMP      INT          NOT NULL REFERENCES EMPLEADO,  
    ESTADO_BOL      CHAR(2)  
)  
GO  
  
CREATE TABLE DETALLEBOLETA(  
    CODIGO_BOL      CHAR(6)      NOT NULL REFERENCES BOLETA,  
    CODIGO_PRO      CHAR(4)      NOT NULL REFERENCES PRODUCTO,  
    CANTID_DET      INT          NOT NULL,  
    PRECIO_DET      MONEY  
    PRIMARY KEY (CODIGO_BOL,CODIGO_PRO)  
)  
GO
```

Actividad

Caso 1: LIGA DE SURCO

La Liga de Surco requiere controlar la constitución de los diferentes equipos deportivos del distrito y de esta manera programar torneos que les permitan mejorar su calidad deportiva. Para ello, ha decidido crear una base de datos.

La liga cuenta con diferentes **clubes** de los cuales se tiene su nombre, fecha de creación, dirección y número de locales. Los clubes tienen distintos tipos de **jugadores** contratados. De los jugadores se conoce su código, el cual se puede repetir para diferentes clubes, los nombres y apellidos, dirección, sexo y fecha de nacimiento, entre otros datos. Cabe mencionar que un jugador es **capitán** de otros jugadores. Ello implicará que deba ser capacitado en cursos de liderazgo y coaching deportivo.

Asimismo, la liga tiene **empleados** de dos tipos: **administrativos** y **técnicos**. De los empleados se almacena un código, los nombres y apellidos, dirección, sexo, fecha de nacimiento y teléfono fijo y celular. Es importante mencionar que para los empleados de tipo Administrativos se almacena su nivel (pregrado o postgrado) y en el caso de los Técnicos, la especialidad deportiva (fútbol, voleibol, natación, etc.)

La liga asigna un **Técnico** un grupo de jugadores y estos pueden tener diferentes Técnicos durante la etapa de jugadores, lo cual constituye un **Equipo**; de este se almacena la categoría (de acuerdo a la fecha de nacimiento del jugador, como Sub-15, etc.) y la disciplina.

Los empleados administrativos elaboran varios contratos de los cuales se guarda el número, la fecha de inicio y fin, entre otros datos. Los contratos son confeccionados para los técnicos.

Finalmente, la liga programa a los equipos en diferentes torneos para que eleven su nivel deportivo controlando la cantidad de participaciones que tiene un determinado equipo. Del torneo se registra el nombre del torneo, las fechas de inicio y fin, así como la disciplina correspondiente.

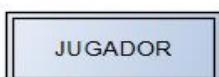
Implemente el diagrama de entidad relación (DER):

Solución:

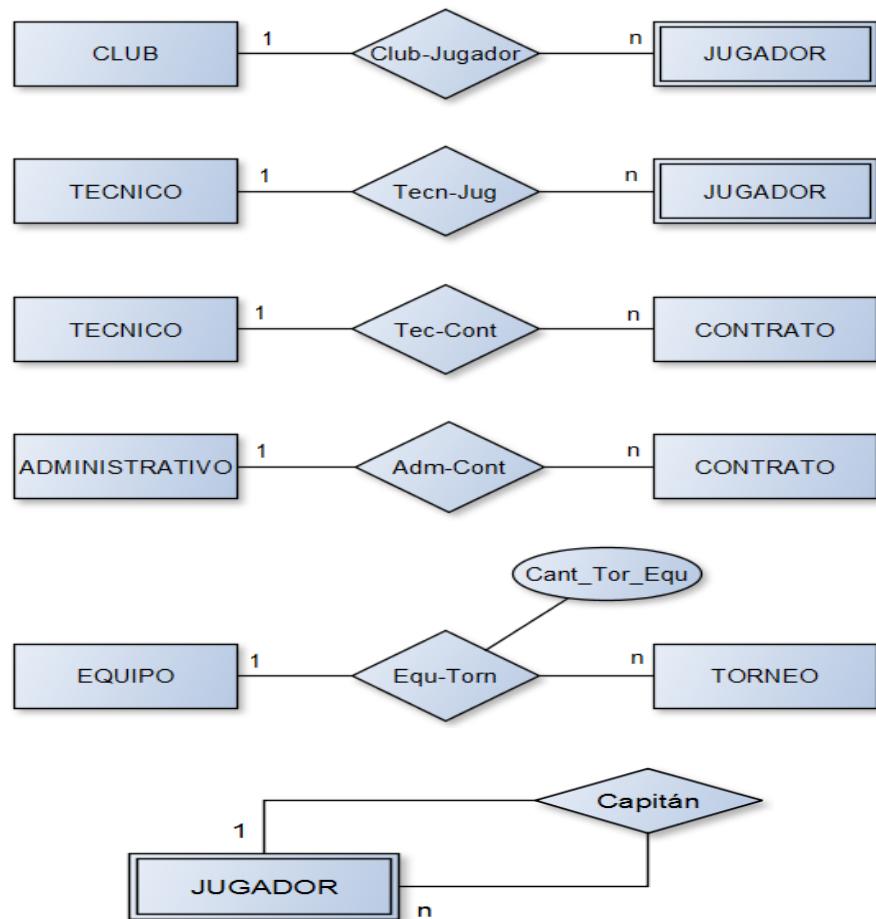
a. Determinando las entidades fuertes:



b. Determinando las entidades débiles



c. Identificando las relaciones



d. Identificando los atributos

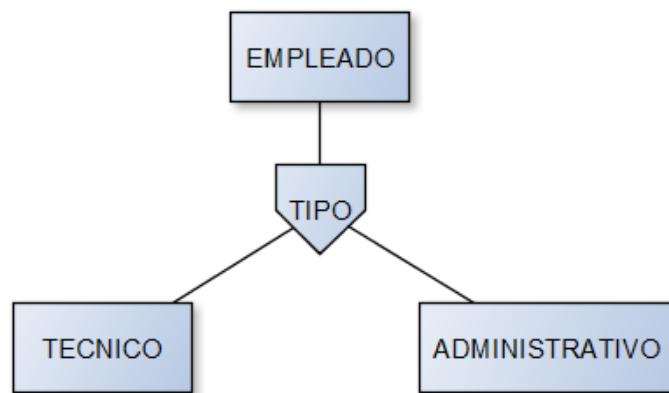
ENTIDAD	ATRIBUTOS
CLUB	cod_club, nom_club, fec_cre_club, dir_club, nro_local_club
JUGADOR	cod_jug, nom_jug, ape_pat_jug, ape_mat_jug, dir_jug, sex_jug, fecnac_jug
EMPLEADO	cod_emp, nom_emp, ape_pat_emp, ape_mat_emp, dir_emp, sex_emp, fecnac_emp, tel_fijo_emp, tel_cel_emp
TECNICO	esp_tec_emp
ADMINISTRATIVO	niv_adm_emp
EQUIPO	cat_equ, disc_equ
CONTRATO	num_cont, fec_ini_con, fec_ter_con
TORNEO	cod_tor, nom_tor, fec_ini_tor, fec_ter_tor, disc_tor

e. Determinando los identificadores o claves (simples o complejas) de cada entidad:

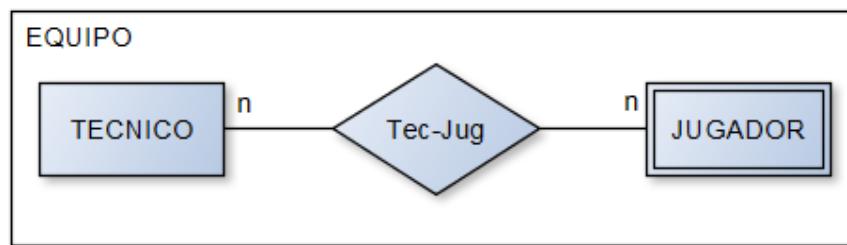
ENTIDAD	ATRIBUTO IDENTIFICADOR
CLUB	cod_club
JUGADOR	cod_club, cod_jug
EMPLEADO	cod_emp
EQUIPO	cod_club, cod_jug, cod_emp
CONTRATO	num_cont
TORNEO	cod_tor

f. Determinando la generalización y agregación

Generalización



Agregación



g. Diagrama de entidad relación (DER)

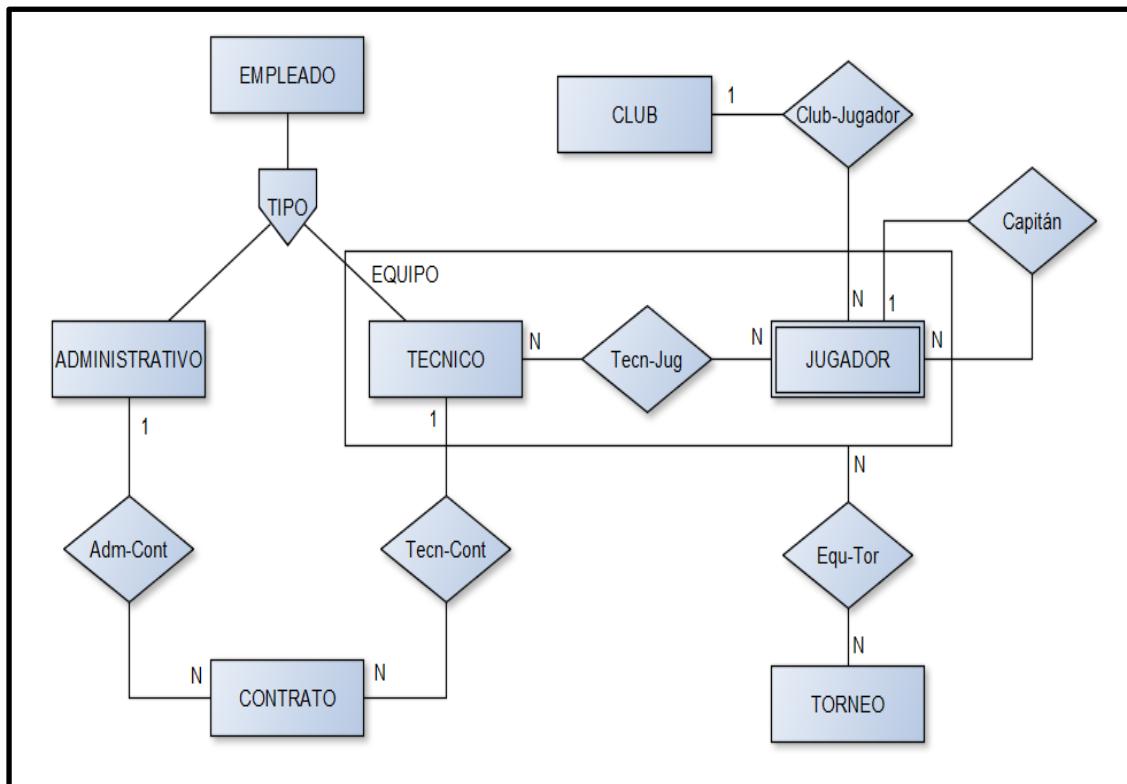


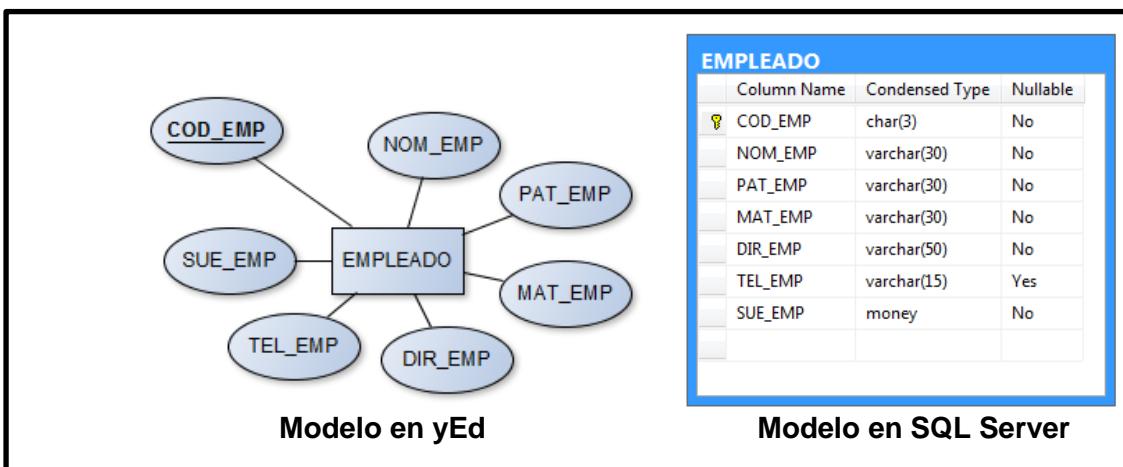
Figura 28: DER final de la liga de Surco
Fuente.- Tomado desde SQL Server 2014

Debe tener en cuenta:

En el DER se deberá agregar los atributos e identificadores de cada entidad, así como en la relación de EQUIPO – TORNEO (E - T) el atributo cantidad de torneos en que participa un equipo (CANT_TOR_EQU), como se vio anteriormente.

Caso 2: Manejo de la sentencia CREATE

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



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)       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 29: Listando las tablas de la base de datos activa
Fuente.- Tomado desde SQL Server 2014

Caso 3: 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 30: 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 31: Listado de columnas de la tabla Empleado
Fuente.- Tomado desde SQL Server 2014

Caso 4: 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:

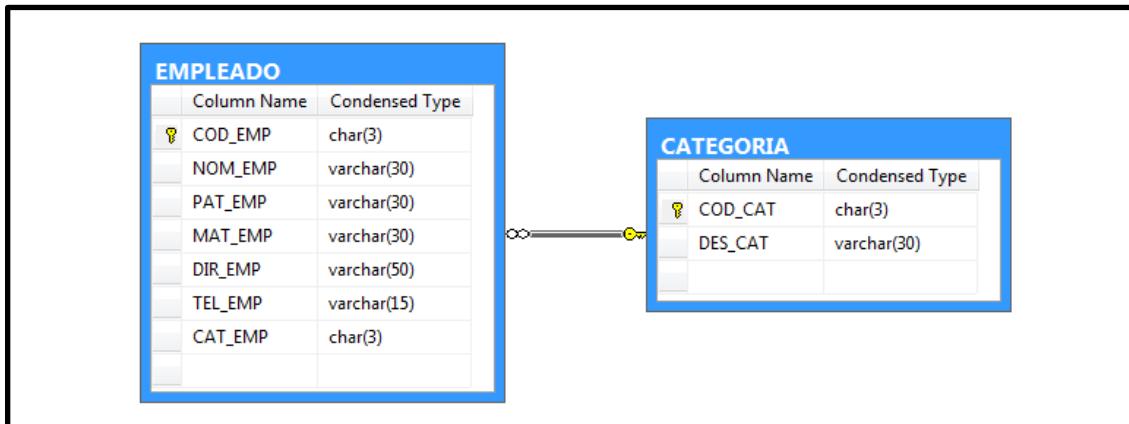


Figura 32: Relación de uno a muchos entre la tabla Empleado y Categoría
Fuente.- Tomado desde SQL Server 2014

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,
    CAT_EMP      CHAR(3)          NOT NULL REFERENCES CATEGORIA
)
GO
  
```

Caso 5: Implementación de la relación recursiva

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

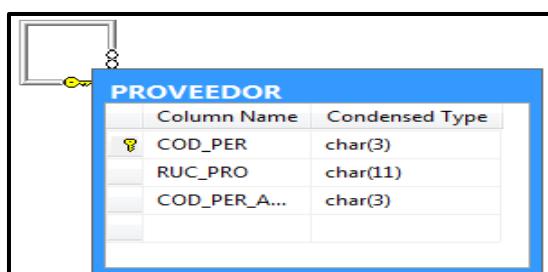


Figura 33: Tabla Proveedor y su recursividad en SQL
Fuente.- Tomado desde SQL Server 2014

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
```

Caso 6: Implementación de la relación muchos a muchos

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

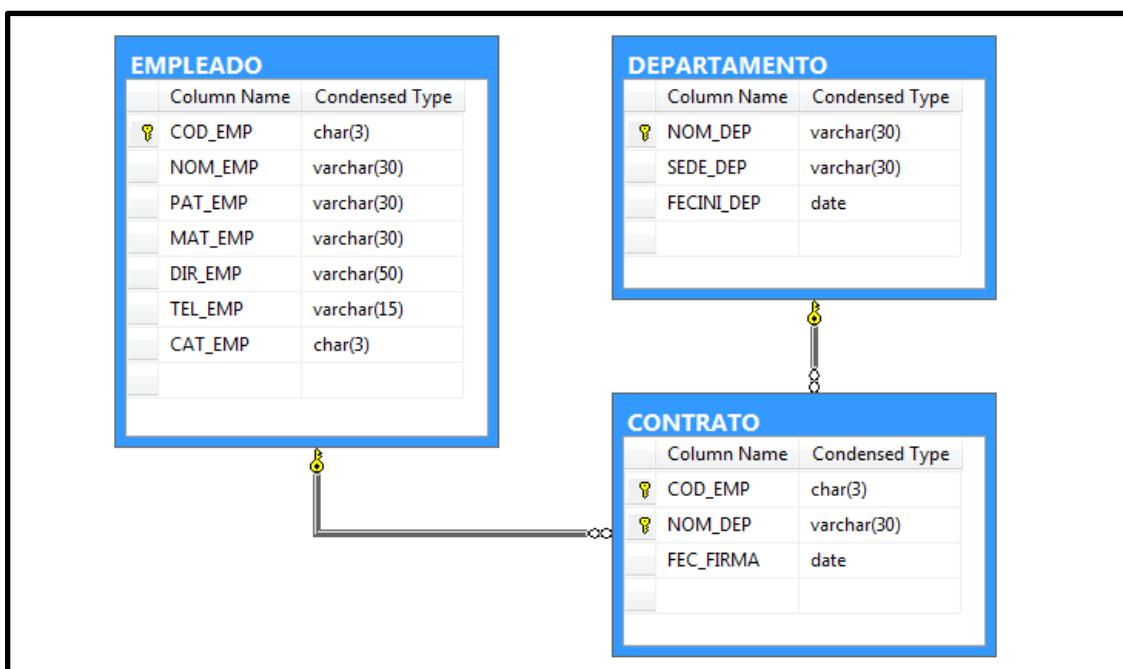


Figura 34: Implementando la relación muchos a muchos en SQL
Fuente.- Tomado desde SQL Server 2014

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,
    CAT_EMP     CHAR(3)      NOT NULL REFERENCES CATEGORIA,
)
GO

CREATE TABLE DEPARTAMENTO(
    NOM_DEPENDENCIA VARCHAR(30) NOT NULL,
    SEDE_DEPENDENCIA VARCHAR(30) NOT NULL,
    FECINI_DEPENDENCIA 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

```

Caso 7: Implementando la generalización entre entidades

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

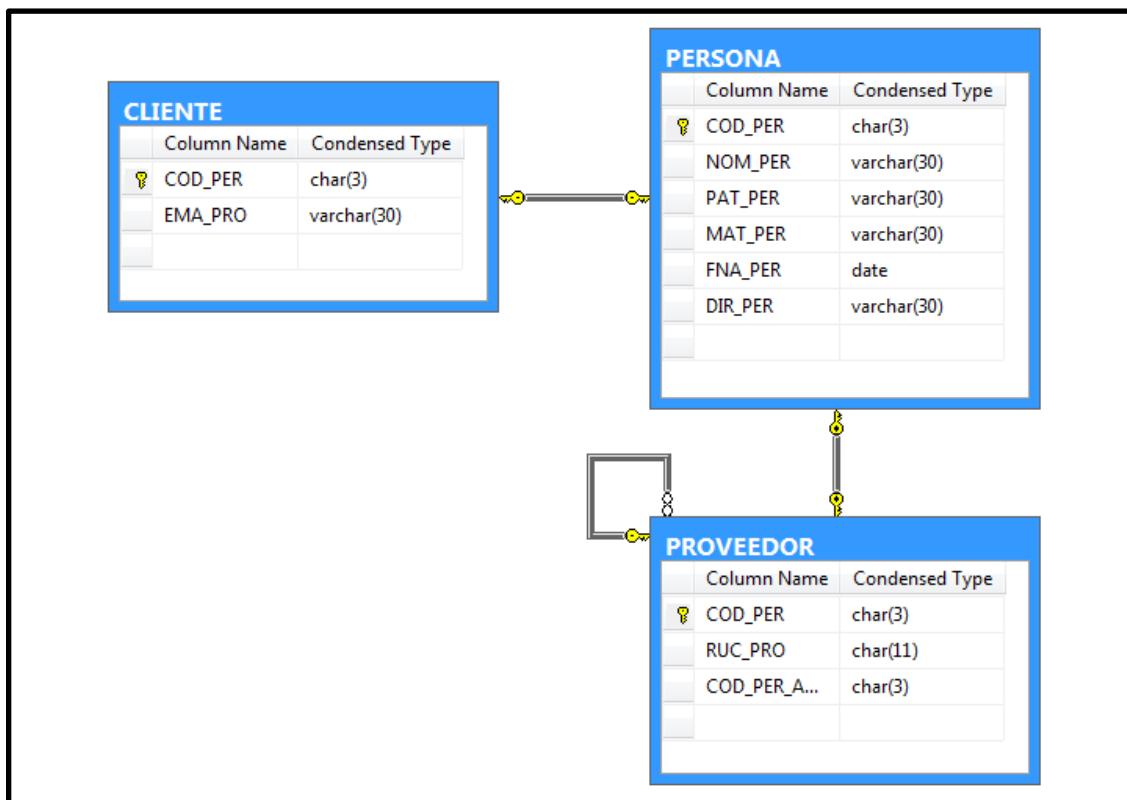


Figura 35: Implementación de la generalización entre tablas en SQL
Fuente..- Tomado desde SQL Server 2014

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
```

Autoevaluación

De cada uno de los siguientes casos realice:

- Identifique las entidades fuertes
- Identifique las entidades débiles
- Identifique las relaciones entre las entidades
- Liste los atributos por cada entidad
- Identifique las llaves simples y complejas de todas las entidades
- Determine la generalización y agregación del caso
- Diseñe el DER final usando yEd, DiaW o Studio Case.
- Implemente el diagrama de base de datos en SQL Server 2014.

Caso 01: Rapid Car

La empresa de taxis “Rapid Car” cuenta con un conjunto de taxis que brindan servicio a los hospitales de dicha ciudad. En la empresa trabajan choferes, cada uno de los cuales se caracteriza por su DNI, nombre y años de servicio.

Así mismo, en la empresa hay numerosos taxis, de los que se conoce la placa, el número del motor, la marca y el modelo. Un taxi puede ser conducido por diferentes choferes (en distintos momentos), pero un chofer siempre conduce el mismo taxi. Se conoce la cantidad de kilómetros totales recorridos por un chofer en su correspondiente taxi. La empresa brinda servicios a varios hospitales.

Por otra parte, de cada hospital se conoce su nombre, tipo y dirección. Un chofer le brinda servicios (realiza viajes) a distintos hospitales y a un hospital le brindan servicios distintos choferes. Se sabe la cantidad de viajes realizados por cada chofer a cada hospital.

Caso 02: Biblioteca Nacional

La Biblioteca Nacional del Perú desea efectuar el control de préstamos de los libros y cubículos a sus diferentes usuarios, para lo cual lo ha contratado a fin de que diseñe y cree una base de datos.

Con respecto a los libros y cubículos son considerados “Recursos” de la biblioteca de los cuales se tiene un código que los identifica. Los libros son prestados a través de una forma denominada Solicitud de Préstamo, de tal manera que un tipo de libro puede ser pedido en varias solicitudes de préstamo, dado que se controla la cantidad de libros existentes (stock). De los libros se almacena su nombre, edición, fecha de edición, cantidad, autor y un breve resumen del libro y de los cubículos se registra su capacidad y cantidad de equipos con que cuenta. Para llevar el control de los libros prestados se registra la fecha de inicio y fin del préstamo, así como el correspondiente control de la devolución.

Así mismo, la biblioteca cuenta con dos (2) tipos de empleados: contratados y practicantes. De los empleados se guarda el código, nombre, apellidos, sexo, dirección, fecha de nacimiento, documento de identidad y el tipo del trabajador. Una solicitud de préstamo es atendida únicamente por un empleado del tipo contratado, dado que debe dar su autorización mediante la firma de dicho documento. De la solicitud de préstamo se almacena el número de la misma, la fecha de solicitud y datos de los usuarios, libros y empleados.

Sin embargo, los pedidos de los cubículos se efectúan a través de la Internet generándose un número único para su identificación, siendo tramitado exclusivamente por empleados del tipo practicantes (por no requerir una firma física) para aquellos usuarios que lo requieran. De los pedidos de cubículos se registra la fecha del préstamo, el turno solicitado y su correspondiente aprobación. Es importante mencionar que de los empleados contratados se almacena la fecha de inicio del contrato y de los practicantes las fechas de inicio y fin de las prácticas.

Asimismo, los usuarios pueden ser de dos tipos: alumnos y profesores, por quienes son de diferentes institutos o colegios, y pueden generar varias solicitudes de préstamo para diferentes libros o pedidos de cubículos. De los usuarios se almacena su código, nombre, apellidos, dirección, teléfono de casa y documento de identidad.

Finalmente, la biblioteca aplica sanciones basadas en el tiempo que excedió la entrega de uno o varios libros. Las sanciones son de tres tipos:

- Definitiva, por haber perdido uno o más libros.
- Parcial, cuya duración es de un mes de suspensión, por haber excedido la fecha máxima del préstamo en una semana.
- Inicial, cuya duración es de una semana, por haber excedido la fecha máxima del préstamo.

De las sanciones se guarda el tipo de la sanción, fecha inicio, fecha término.

Caso 03: Movimiento Mercantil

Se desea diseñar una base de datos sobre el movimiento mercantil de un organismo en un año. En el organismo existen mercancías de las que se conoce su código, nombre y unidad de medida. Las mercancías proceden de diferentes países de los que se sabe nombre y tipo de moneda. Para la transportación de las mercancías existen diversas formas, cada una de las cuales se caracteriza por su tipo (barco, avión, tren, etc.) y tarifa.

Así mismo, para cada mercancía de diferentes países existen diferentes formas de transportación; para cada país existen diferentes mercancías que son transportadas en diferentes formas de transportación; y una forma de transportación puede serlo de diferentes mercancías de diferentes países. Una mercancía procedente de un país transportada de una forma dada constituye un embarque y para éste se conoce su fecha de arribo y cantidad.

Por otra parte, un embarque se distribuye entre diferentes almacenes y en un almacén se tienen diferentes embarques, cada uno en cierta cantidad. De cada almacén se tiene su código y dirección. Un almacén distribuye los productos entre diferentes empresas y cada empresa recibe productos de diferentes almacenes. Una empresa se caracteriza por su número, nombre y rama económica; a su vez, las empresas establecen relaciones contractuales entre sí. Entre dos empresas dadas sólo se puede establecer un contrato anual. De cada contrato se conoce su número, valor y fecha de vencimiento.

Resumen

1. El Modelo entidad-relación permite representar lógicamente un caso.
2. En el diagrama entidad-relación, una entidad se representa mediante un rectángulo, una relación mediante un rombo, un atributo mediante un círculo. Todos estos elementos deben aparecer debidamente identificados por medio de un nombre. Los atributos que constituyen la llave de una entidad, deben tener el círculo relleno.
3. Una entidad cuyos atributos no sean suficientes para identificarla se denomina débil y su llave está formada por algún o algunos de sus atributos más la llave de la entidad que le da origen. Se representa con un doble rectángulo y con la relación entre ella y la entidad que le da origen. Esta relación es de, a lo sumo, muchos (por el extremo de la débil) a uno (por el extremo de la entidad que la origina).
4. La llave de una relación de m:n está formada por la llave de las entidades que participan en la relación. La llave de una relación de 1:n está formada por la llave de la entidad del extremo muchos. La llave de una relación de 1:1 está formada por la llave de cualquiera de las entidades que participan.
5. En una Generalización/Especialización, la entidad generalizada describe las características generales o comunes que son aplicables a todas las especializaciones. Las especializaciones, como casos especiales de la generalización, sólo contemplan sus propiedades particulares. La llave de cada Especialización es la misma de la generalización.
6. Una agregación es el resultado de considerar una relación como una entidad. Los atributos de la relación pasan a ser atributos de la entidad agregada. La llave de la entidad agregada es la llave de la relación que la originó, excepto en el caso en que se defina especialmente un identificador para la agregación, pasando entonces a ser la llave el identificador.
7. Todos estos elementos pueden combinarse en un DER. Por ejemplo, una entidad débil puede ser, a su vez, una generalización que tenga sus especializaciones; una generalización puede tener especializaciones que, a su vez, puedan ser generalizaciones de otras especializaciones; entre las entidades que participan en una agregación puede haber una entidad débil, etc.

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

- <http://www.ongei.gob.pe/publica/metodologias/Lib5011/cap2-3.htm>
En esta página web hallará algunos conceptos complementarios a los mostrados en el manual sobre modelo conceptual de datos.
- <http://elies.rediris.es/elies9/5-2.htm>
En esta página web encontrará definiciones complementarias al modelo conceptual.
- http://www.ecured.cu/index.php/Diagrama_Entidad_Relaci%C3%B3n
En esta página web hallará algunos conceptos complementarios al diagrama entidad relación



MODELO RELACIONAL Y NORMALIZACIÓN

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno organiza datos no estructurados y los integra al diseño de la base de datos a partir de documentos comerciales utilizando la técnica de Normalización (1FN, 2FN, 3FN), además lo implementa en SQL Server 2014.

TEMARIO

3.1 Tema 12 : Normalización parte 1

3.1.1 : Primera forma normal

3.2 Tema 13 : Normalización parte 2

3.2.1 : Segunda forma normal

3.2.2 : Tercera forma normal

3.2.3 : Ejercicios de normalización

3.3 Tema 14 : Normalización parte 3

3.3.1 : Detalle del detalle / ítems

3.4 Tema 15 : Implementación de detalle de detalle en SQL Server 2014

3.4.1 : Obtención del modelo lógico-global de los datos del DER

ACTIVIDADES PROPUESTAS

- Los alumnos reconocen los principales conceptos de la normalización de documentos.
- Los alumnos aplican la normalización a formatos.
- Los alumnos implementan el resultado de la normalización en SQL Server 2014.

3.1. Normalización parte 1

Modelo relacional:

Uno de los modelos matemáticos más importantes y actuales para la representación de las bases de datos, es el enfoque relacional. Este se basa en la teoría matemática de las relaciones lo que le suministra una fundamentación teórica que permite aplicar todos los resultados de dicha teoría a problemas tales como el diseño de sublenguajes de datos y otros.

El término relación se puede definir matemáticamente como sigue:

- Dados los conjuntos **D₁, D₂, ..., D_n** (no necesariamente distintos), **R** es una relación sobre esos **n** conjuntos si está constituida por un conjunto de n-tuplos ordenados **d₁,d₂,...d_n** tales que **d₁ ∈ D₁ , d₂ ∈ D₂, ..., d_n ∈ .**
- Los conjuntos **D₁, D₂, ..., D_n** se llaman dominios de **R** y **n** constituye el grado de la relación. La cantidad de tuplas constituye la cardinalidad (tipo de relación de correspondencia **1-1, 1-n, n-1 y n-n**)

Es conveniente representar una relación como una tabla bidimensional donde cada fila representa un n-tuplo, tal como se muestra en la siguiente imagen:

COLUMNA1	COLUMNA2	COLUMNA3	COLUMNA4	
CODIGO DE FACTURA	FECHA FACTURADA	CODIGO DE VENDEDOR	CODIGO DE CLIENTE	
FAC001	2015-04-05 10:13:39.497	VEN001	CLI001	FILA 1
FAC002	2015-04-05 10:13:39.497	VEN001	CLI002	FILA 2

Las columnas de la tabla también son consideradas como **atributos** de una entidad y las filas como **ocurrencias** o tuplas.

En el modelo relacional, tanto los objetos o entidades, como las relaciones que se establecen entre ellos, se representan a través de "tablas", que en la terminología relacional se denominan relaciones.

Cada relación está compuesta de filas (las ocurrencias de los objetos) y se les denomina, en la terminología relacional, como tuplos, tuplas o uplos, uplas (en realidad, n-tuplos, pero en muchos casos se suprime la n cuando no existe posibilidad de confusión).

También la relación está compuesta por columnas (los atributos o campos que toman valores en sus respectivos dominios).

Debemos tener en cuenta los siguientes aspectos:

1. No pueden haber dos filas o tuplas iguales en una misma tabla o entidad.
2. El orden asignado a las filas no es significativo.
3. El orden asignado a las columnas no es significativo.
4. Cada valor dentro de la relación (cada valor de un atributo) es un dato atómico (o elemental), es decir, no descomponible; por ejemplo, un número, una cadena de caracteres. En otras palabras, en cada posición fila-columna existe un solo valor, nunca un conjunto de valores.

Una relación que satisface este último punto se denomina "**normalizada**", aunque veremos más adelante que, en realidad, lo que ocurre es que está en Primera Forma Normal.

La teoría de la **normalización** se basa en la necesidad de encontrar una representación del conjunto de relaciones que en el proceso de actualización sea más adecuada. Llevar una relación no normalizada a normalizada es muy simple. Existen diferentes niveles de normalización que se llaman formas normales que veremos más adelante.

Veamos un ejemplo donde un suministrador y producto se puede representar fácil y claramente mediante el modelo relacional.

Los atributos de estas dos entidades son los siguientes:

- **Suministrador:** Número, que lo identifica, Nombre, Tipo y Distrito donde radica.
- **Producto:** Número, que lo identifica, Nombre, Precio unitario y Peso.

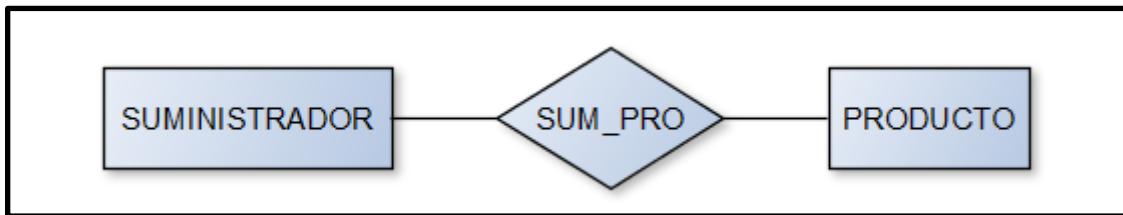


Figura 36: Modelo Entidad Relación
Fuente.- Tomado desde la aplicación yEd

Además, un **suministrador** puede suministrar muchos **productos** y un **producto** puede ser suministrado por varios **suministradores**. Se conoce la cantidad de un determinado producto que suministra un suministrador dado.

TABLA: **SUMINISTRADOR**

CÓDIGO_SUM	NOMBRE_SUM	TIPO_SUM	DISTRITO_SUM
S001	Maria Rojas Zamora	30	San Isidro
S002	Carla Segura Ramos	10	Miraflores
S003	Jenifer Lopez Arias	20	Lince
S004	Guadalupe Salvatierra Gonzales	20	Lince
S005	Janeth Cruz Cruz	15	Surco

TABLA: **PRODUCTO**

CÓDIGO_PRO	DESCRIP_PRO	PRECIO_PRO	PESO_PRO
P001	Clavo 1 pulgada.	S/. 0.20	12
P002	Tuerca 3/8	S/. 0.50	17
P003	Martillo	S/. 15.50	80
P004	Tornillo 1 1/2	S/. 2.00	10
P005	Alicate de presión	S/. 14.00	50

TABLA: SUM-PRO

CÓDIGO_SUM	CÓDIGO_PRO	CANTIDAD
S001	P003	100
S002	P001	150
S002	P003	120
S005	P002	50
S004	P002	30
S001	P005	50
S003	P003	20
S001	P001	10
S002	P003	15
S005	P005	2
S001	P003	50
S002	P002	60
S001	P001	10
S004	P003	14

La representación en el modelo relacional es más simple que con el modelo jerárquico y el modelo reticular, ya que con tres (3) tablas se tiene todo el modelo representado.

En el **modelo relacional**, el resultado de una demanda es también una relación y las demandas simétricas (en el sentido de ser una la inversa de la otra; por ejemplo:

- Si queremos recuperar los números de los suministradores que suministran el producto “P4”.
- Y recuperar los números de los productos que suministra el suministrador “S2” requieren operaciones simétricas.

Las diversas formas de expresar las recuperaciones dan lugar a los lenguajes relacionales cuyas formas más representativas son:

- **Álgebra relacional** (basado en las operaciones del álgebra de relaciones)
- **Cálculo relacional** (basado en el cálculo de predicados)

Ventajas del modelo relacional:

- Una de las principales ventajas es su simplicidad, pues el usuario formula sus demandas en términos del contenido informativo de la base de datos sin tener que atender a las complejidades de la realización del sistema, lo que implica gran independencia de los datos.
- La información se maneja en forma de tablas, lo que constituye una manera familiar de representarla.
- Al igual que en el modelo reticular, si se tienen relaciones normalizadas, no surgen dificultades grandes en la actualización.

Veamos en el modelo del **SUMINISTRADOR-PRODUCTO** presentado anteriormente, un ejemplo de cada tipo de operación de actualización:

- **Creación.**- Añadir un producto P7. Se agrega la nueva ocurrencia en la tabla PRODUCTO. Es posible hacerlo aunque ningún suministrador lo suministre.
- **Eliminación.**- Se puede eliminar el suministrador S1 sin perder el producto P6, a pesar de que es el único suministrador que lo suministra.
- **Modificación.**- Se puede cambiar el precio del producto P2 sin necesidad de búsquedas adicionales ni posibilidad de inconsistencias.

No obstante, veremos que el proceso de normalización no es suficiente hasta el punto aquí visto.

Desventajas

Se dice que la fundamental desventaja consiste en la dificultad de lograr productividad adecuada de los sistemas, ya que no se emplean los medios técnicos idóneos, tales como las memorias asociativas, por ello, es siendo necesario simular este proceso, pero, en realidad, la eficiencia y productividad de los sistemas actuales resultan realmente muy satisfactorias.

Normalización

La teoría de la normalización se ha desarrollado para obtener estructuras de datos eficientes que eviten las anomalías de actualización. El concepto de normalización fue introducido por E.D. Codd y fue pensado para aplicarse a sistemas relacionales. Sin embargo, tiene aplicaciones más amplias.

La normalización es la expresión formal del modo de realizar un buen diseño. Provee los medios necesarios para describir la estructura lógica de los datos en un sistema de información.

Ventajas de la normalización

- Evita anomalías en la actualización.
- Mejora la independencia de los datos, porque permite realizar extensiones de la base de datos, afectando muy poco, o nada, a los programas de aplicación existentes que acceden la base de datos.

La normalización involucra varias fases que se realizan en orden. La realización de la 2º fase supone que se ha concluido la 1ra. Tras completar cada fase se dice que la relación está en:

- Primera Forma Normal (1FN)
- Segunda Forma Normal (2FN)
- Tercera Forma Normal (3FN)
- Forma Normal de Boyce-Codd (FNBC)

Existen, además, la Cuarta (4FN) y la Quinta (5FN) Formas Normales.

Las relaciones en 1FN son un subconjunto del universo de todas las relaciones posibles. Las relaciones en 2FN son un subconjunto de las que están en 1FN y así sucesivamente, como se muestra en la siguiente figura:

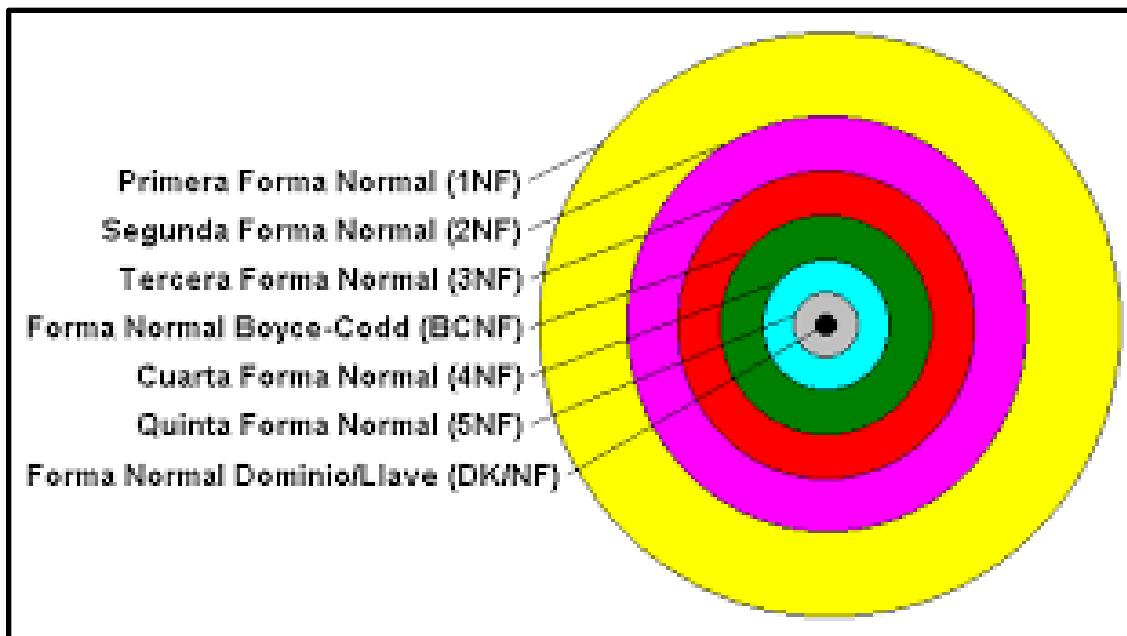


Figura 37: Formas Normales
 Fuente.- Tomado de <http://cvitae.tripod.com/loida/normaliz.html>

3.1.1. Primera Forma Normal

Para las explicaciones de los contenidos correspondientes a las formas normales de la 1FN hasta la 3FN, desarrollaremos un ejemplo que consiste en el diseño de la base de datos para la automatización del control de los pedidos de productos y que se presenta a continuación. Supongamos que los modelos para pedir los productos sean como se muestra en la siguiente figura:

FECHA 30/06/2015	Nº PEDIDO : 02368-52 Nº CLIENTE : 1542 NOMBRE DEL CLIENTE : JUAN JOSE PEREZ LOZA DIRECCION : AV. EL SOL 574 - CHORRILLOS			
DETALLE DEL PEDIDO				
CODIGO DE PRODUCTO	DESCRIPCION	PRECIO UNITARIO	CANTIDAD	TOTAL
15-5562	Televisor 42 pulg. LG	S/. 2,500.00	1	S/. 2,500.00
15-2154	Blue Ray LG	S/. 250.00	1	S/. 250.00
15-0125	Play Station 4 - 500GB	S/. 1,850.00	1	S/. 1,850.00
15-1545	Audifonos Phillips	S/. 50.00	1	S/. 50.00
IMPORTE TOTAL		S/. 4,650.00		

Figura 38: Hoja de Pedido
 Fuente.- Tomado desde la aplicación Excel

PASO 1

Listando todos los atributos y se determinar la llave de toda relación.

- a) El análisis de este modelo de pedido de productos muestra que los atributos que se listan a continuación son de interés:

Atributo	Descripción
NUM_PED	Número de pedido.
FEC_PED	Fecha en que se realizó el pedido.
NOM_CLI	Nombre del cliente solicitante.
DIR_CLI	Dirección del cliente.
NUM_PRO	Número o código del producto.
DES_PRO	Descripción del producto solicitado en pedido.
PUNI_PRO	Precio unitario del producto.
CANT_PED	Cantidad de productos solicitados en el pedido.
TOT_PED	Importe por cada producto solicitado.
IMP_PED	Importe total a pagar por todos los productos pedidos.

- b) El número de pedido es único y se utiliza para referirse a un pedido, por tanto, se usará como clave (llave).

Nota.- esta es una suposición inicial, que luego se discutirá en la siguiente sesión.

- c) Se determina los subconjuntos posibles, es decir los valores cuyos datos son repetidos. Aquí debemos mencionar que estos atributos se determinando observando la boleta de pedido.

Atributo	Descripción
NUM_PRO	Número o código del producto.
DES_PRO	Descripción del producto solicitado en pedido.
PUNI_PRO	Precio unitario del producto.
CANT_PED	Cantidad de productos solicitados en el pedido.
TOT_PED	Importe por cada producto solicitado.

- d) Se determina el atributo del cual dependen los demás atributos total o parcialmente, en este caso es **NUM_PRO**, para crear la clave compuesta.

- e) Luego, se determina la relación resultante:

PEDIDO (NUM_PED, FEC_PED, NUM_CLI, NOM_CLI, DIR_CLI,
NUM_PRO, DES_PRO, PUNI_PRO,
CANT_PED, TOT_PED, IMP_PED)

3.1.1.1 Definición de la Primera Forma Normal (1FN)

Definición Principal

“Una relación está en 1FN si cumple la propiedad de que sus dominios no tienen elementos que, a su vez, sean conjuntos”.

Los valores que puede tomar un atributo de una relación son los elementos contenidos en su correspondiente dominio.

Si se permitiera que un elemento del dominio de un atributo fuera un conjunto, entonces, dicho atributo pudiera tomar como valor ese conjunto de valores. Eso implicaría que en una posición fila, columna habría un conjunto de valores y no un único valor, como hemos visto que debe ocurrir en el modelo relacional.

Segunda definición

“Toda relación normalizada, o sea, con valores atómicos de los atributos, está en 1FN”.

Tercera definición

“Una relación está en 1FN si no incluye ningún grupo repetitivo”.

Del caso se puede observar que la relación **PEDIDO-PRODUCTOS** tiene 5 atributos repetitivos:

NUM_PRO
DES_PRO
PUNI_PRO
CANT_PED

Ya que un pedido puede contener más de una línea de pedido y, por lo tanto, puede contener varios números de producto (NUM_PRO), varias descripciones de producto (DES_PROD), varios precios unitarios (PUNI_PRO) y varias cantidades (CANT_PED).

Esta situación acarrea problemas de actualización, pues habría que decidir la cantidad máxima de líneas de pedido de productos que se permitiría en un pedido, dado que los campos de la tabla PEDIDO deben tener un tamaño dado y, entonces, serían capaces de almacenar sólo una determinada cantidad máxima de valores cada uno de ellos.

Esto sería agregar una limitación en el modelo, pues no tiene ese comportamiento en la realidad. Además, en general, se desaprovecharía memoria, dado que si se decide, por ejemplo, que se va a permitir hasta 20 líneas de pedido en cada pedido, habría que definir tamaños de campos para los grupos repetitivos que permitieran almacenar esa cantidad de valores. Entonces, si en un pedido se solicitan menos de 20 productos, lo cual puede ser muy frecuente, no se utilizaría parte del espacio reservado para cada campo.

PASO 2

Determinar las relaciones de grupos repetidos de los que no lo son.

Hay que eliminar esos grupos repetitivos para que la relación esté en 1FN. Para ello, se crea:

- 1) Una relación para los campos que sean únicos, es decir, se dejan en la relación original sólo los atributos que no son repetitivos:

PEDIDO (NUM_PED, FEC_PED, NUM_CLI, NOM_CLI, DIR_CLI, PR_PED)

- 2) Se crea una relación para los grupos repetitivos. Además, se crea una llave compuesta formada por la llave primaria de la relación original (NUM_PED) y el atributo del cual dependen los demás atributos repetidos total o parcialmente, en este caso es NUM_PROD:

PED-PROD (NUM_PED, NUM_PROD, DES_PRO, PUNI_PRO, CANT_PED, TOT_PED)

PASO 3

Determinación de la llave de cada relación.

Ambas tienen como llave, o parte de la llave, a **NUM_PED**. Pero en **PED-PROD** es necesario la llave compuesta para identificar los productos individuales pues, por ejemplo, **CANT_PED** se refiere a la cantidad en que se pide un determinado producto en un pedido dado (el producto puede solicitarse en otros pedidos en diferentes cantidades y en el pedido se pueden estar pidiendo otros productos en diferentes cantidades) y, por lo tanto, para identificar una cantidad dada es necesario referirse al pedido y al producto correspondientes.

Ahora, estas nuevas dos relaciones en 1FN modelan el que nos ocupa. Los problemas de actualización mencionados anteriormente quedan resueltos con este nuevo modelo.

En lugar de tener varios valores en cada campo de acuerdo a la cantidad de líneas de pedido, tal y como ocurría en la tabla **PEDIDO** original, se tienen varias ocurrencias en la tabla **PED-PROD**, una por cada producto que se solicita en el pedido. Esto permite que se soliciten tantos productos como se desee en cada pedido, pues sólo significa agregar una nueva ocurrencia en la relación **PED-PROD** por cada producto solicitado. Sin embargo, este modelo en 1FN tiene aún problemas de actualización, como se muestra en las siguientes operaciones:

- **Creación.**- La información sobre un nuevo producto no se puede insertar si no hay un pedido que lo incluya.
- **Eliminación.**- Eliminar una línea de pedido que sea la única que pida un producto implica perder la información del producto.
- **Modificación.**- Por cada línea de pedido en la que se solicite determinado producto se tiene una ocurrencia en PED-PROD, que repite la información sobre éste. Si cambia algún atributo del producto, entonces, es necesario hacer muchas actualizaciones.

Entonces, será necesario aplicar formas normales más fuertes a este modelo para eliminar los problemas de actualización que presenta, como veremos en los próximos capítulos.

3.1.1.2 Dependencia Funcional (DF)

Dada una relación R, se dice que el atributo Y de R es funcionalmente dependiente del atributo X de R, si y sólo si, cada valor X en R tiene asociado a él, precisamente, un valor de Y en R en cualquier momento del tiempo.

Y depende funcionalmente de X se denota como:

$$X \rightarrow Y$$

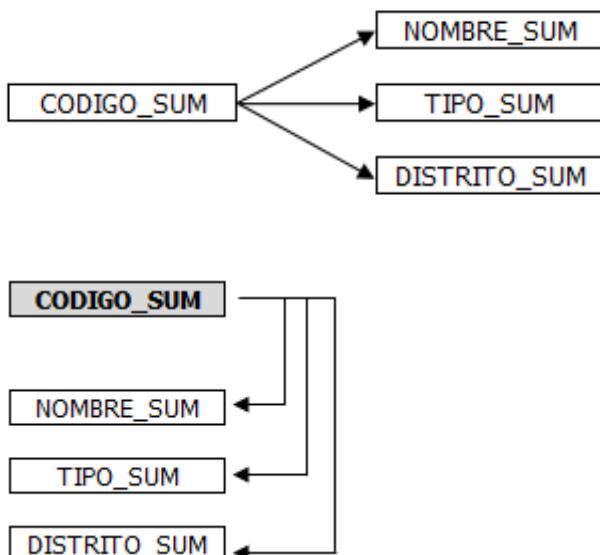
Observemos la siguiente tabla:

TABLA: **SUMINISTRADOR**

CÓDIGO_SUM	NOMBRE_SUM	TIPO_SUM	DISTRITO_SUM
S001	Maria Rojas Zamora	30	San Isidro
S002	Carla Segura Ramos	10	Miraflores
S003	Jenifer Lopez Arias	20	Lince
S004	Guadalupe Salvatierra Gonzales	20	Lince
S005	Janeth Cruz Cruz	15	Surco

Las columnas **NOMBRE_SUM**, **TIPO_SUM** y **DISTRITO_SUM** son funcionalmente dependientes cada uno de **CÓDIGO_SUM**, ya que para un valor de **CÓDIGO_SUM** existe un valor correspondiente de **NOMBRE_SUM**, **TIPO_SUM** y **DISTRITO_SUM**.

Estas cuatro (4) representaciones son formas de mostrar las dependencias funcionales.



$CÓDIGO_SUM \rightarrow NOMBRE_SUM$
 $CÓDIGO_SUM \rightarrow TIPO_SUM$
 $CÓDIGO_SUM \rightarrow DISTRITO_SUM$

CODIGO_SUM → **NOMBRE_SUM** **TIPO_SUM** **DISTRITO_SUM**

El reconocimiento de las dependencias funcionales es una parte esencial de la comprensión de la semántica, del significado del dato. El hecho de que **DISTRITO_SUM** dependa funcionalmente de **CODIGO_SUM** significa que cada suministrador está situado en un distrito.

La noción de dependencia funcional puede ser extendida hasta cubrir el caso en que X, Y o ambos atributos sean compuestos.

3.1.1.3 Dependencia Funcional Completa (DFC)

El atributo Y es funcionalmente y completamente dependiente del atributo X, si es funcionalmente dependiente de X y no es funcionalmente dependiente de algún subconjunto de X.

En algunos textos lo representan como: $X \Rightarrow Y$

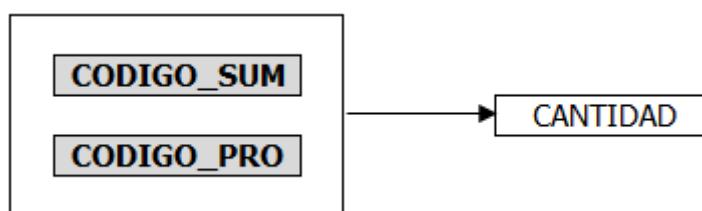
El atributo **CANTIDAD** es funcionalmente dependiente del par de atributos (**CODIGO_SUM** = Código del suministrador y **CODIGO_PRO** = Código de producto).

TABLA: **SUM-PRO**

CÓDIGO_SUM	CÓDIGO_PRO	CANTIDAD
5001	P003	100
5002	P001	150
5002	P003	120
5005	P002	50
5004	P002	30
5001	P005	50
5003	P003	20
5001	P001	10
5002	P003	15
5005	P005	2
5001	P003	50
5002	P002	60
5001	P001	10
5004	P003	14

Por ello, se podría denotar **CANTIDAD** como **CANTIDAD_PRO_SUM**, que en el diccionario de datos se detallaría como la **cantidad de productos provista por un suministrador**.

Esta última explicación muestra que la cantidad tiene dependencia funcional completa tanto del suministrador como del producto (Entidades Fuertes).



3.1.1.4 Llave

Al hablar de entidad en el modelo entidad - relación, se asumió que existe una llave para cada entidad, formada por un conjunto de atributos que definen de forma única la entidad. Un concepto análogo se define para las relaciones o tablas en el modelo relacional.

Sean:

- R una relación con atributos A₁, A₂, ..., A_n y
- X un subconjunto de A₁, A₂, ..., A_n

Se dice que X es una llave de R si y solo si:

- 1) X → A₁, A₂, ..., A_n
Todos los atributos de la relación dependen funcionalmente de X
- 2) $\exists Y \subset X \mid Y \rightarrow A_1, A_2, \dots, A_n$

Lo planteado en el punto 2 es una condición de **minimalidad** que no se requería para el concepto de llave en el modelo entidad - relación.

Una relación puede tener varias llaves. Una de ellas se nombra "llave primaria" (la que se escoge para trabajar) y las restantes se nombran "**llaves candidatas**". Una **superllave** será cualquier superconjunto de una llave. Entonces, una llave es un caso especial de superllave.

3.1.1.5 Procedimiento para hallar las llaves candidatas de una relación

Supongamos que se quiere encontrar las llaves candidatas de una relación R(A, B, C, D, E) con las siguientes dependencias funcionales:

$$\begin{aligned} A &\rightarrow B \\ BC &\rightarrow D \\ AB &\rightarrow E \end{aligned}$$

Para comenzar, se parte de que no existen más llaves que dependencias funcionales, pues el concepto de llave incluye la existencia de dependencia funcional. Se analiza, por tanto, cada una de las **DF** presentes en la relación, añadiendo los atributos que sean imprescindibles en la parte izquierda para lograr determinar a todos los atributos de la relación. El conjunto de atributos así formado debe ser mínimo.

Luego se analiza cada uno de esos conjuntos mínimos, de forma que, si alguno es un superconjunto de otro, ya no es llave, sino superllave. Pueden resultar varias llaves candidatas.

En el ejemplo:

- | | | |
|---------|-------------------|---------------|
| 1. A→B | AC→A B E C D AC | es llave |
| 2. BC→D | BCA→B C D A E BCA | es superllave |
| 3. AB→E | ABC→A B E C D ABC | es superllave |

La única llave es AC. No hay ninguna otra llave candidata, pues en las otras DF se obtiene el mismo conjunto ABC y este conjunto contiene a AC.

Ejemplo:

Sea la relación R (ciudad, calle, código postal). Para abbreviar, R(C, A, P) donde se tiene que:

- Una calle en una ciudad tiene un código postal: CA→P.
- El código postal tiene una estructura tal que su valor determina la ciudad: P→C.
- Pero en una ciudad, varias calles pueden tener el mismo código, por
- lo que no se cumple P→A.

Entonces, analizando cada DF se tiene:

- CA→P, pero CA se determina a sí mismo también: CA→CA, por lo tanto, CA→CAP, es decir, que CA determina a todos los atributos de R y, como no existe ningún subconjunto de CA que, a su vez, determine a todos los atributos de la relación, se puede concluir que AC es llave.
- P → C y no determina a, pero agregando A en el lado izquierdo: PA→CA y es obvio que PA→PA también, entonces PA→CAP, y no existe ningún subconjunto de PA que, a su vez, determine a todos los atributos de la relación, por lo tanto, PA también es llave.

Atributo llave

Un atributo Ai de R es un atributo llave si él es (o es miembro de) una llave (candidata o primaria).

Atributo no llave

Un atributo Aj de R es un atributo no llave si él no es miembro de ninguna llave candidata.

Por ejemplo, en el caso de la relación R(C, A, P) vista anteriormente, C, A y P son todos atributos llaves.

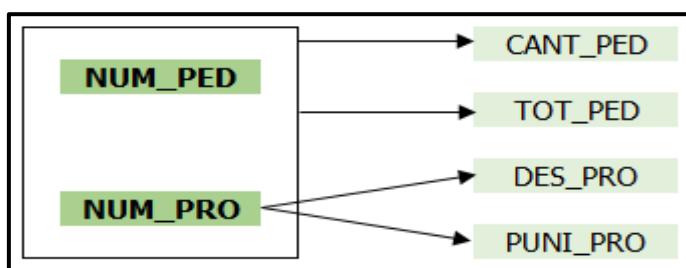
3.2. Normalización parte 2

3.2.1. Segunda Forma Normal

Una relación R se dice que está en **2FN** si está en **1FN** y si, y solo si, los atributos no llaves (ni primarias, ni candidatas) de R, si los hubiese, son funcional y completamente dependientes de la llave primaria de R.

Entonces, se aplica solo a relaciones con llaves compuestas, pues no es posible que en una relación cuya llave primaria sea simple (compuesta por un solo atributo) haya atributos que dependan de parte de la llave primaria. Una relación que esté en 1FN y que tenga una llave primaria simple está en 2FN.

En el ejemplo de los pedidos de productos, habíamos visto que en la relación **PED-PRO** subsistían problemas de actualización. Analicemos las dependencias funcionales que existen en dicha relación:



Esta relación no está en 2FN, pues **DES_PRO** y **PUNI_PRO** no dependen funcional y completamente de la llave (**NUM_PED**, **NUM_PRO**).

Paso único.- Se determina si existen relaciones con clave compuesta. Si **NO** las hay, las relaciones obtenidas en la Primera Forma Normal se encuentran en Segunda Forma Normal. De lo contrario, se efectúa lo siguiente:

1. Se crea una relación para todos los atributos que dependen funcional y completamente de la llave (y los atributos que no se analizan por ser atributos llaves, pertenecientes a claves candidatas).

PED-PRO (NUM_PED, NUM_PRO, CANT_PED, TOT_PED)

2. Se crea una relación para los atributos que dependan de cada parte (subconjunto) de la llave. La llave de la relación así formada será la parte (subconjunto) de la llave primaria de la cual dependen los atributos.

PRODUCTO (NUM_PRO, DES_PRO, PUNI_PROD)

Los problemas planteados en la 1FN se resuelven con la 2FN. Veamos:

- **Creación.** Se puede insertar la información sobre un producto aunque no haya un pedido que lo solicite.
- **Eliminación.** Se puede eliminar una línea de pedido y no se pierde la información sobre el producto, aunque sea el único pedido que pide ese producto.
- **Modificación.** Si cambia un atributo del producto, sólo hay que cambiarlo en un lugar. Se elimina redundancia.

Pero aún tenemos problemas en este caso que son similares a los vistos, pero con la relación PEDIDO y, específicamente, cuando se trata de insertar, eliminar o modificar la información de proveedores:

3.2.2. Tercera Forma Normal

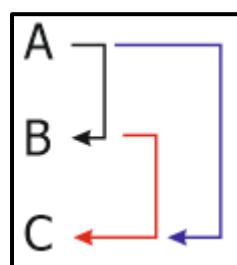
La tercera forma normal (3NF) es una forma normal usada en la normalización de bases de datos. La 3NF fue definida originalmente por E.F. Codd en 1971. La definición de Codd indica que una tabla está en 3NF si y solo si las dos condiciones siguientes se mantienen:

- La tabla está en la segunda forma normal (2NF)
- Ningún atributo no-primario de la tabla es dependiente transitivamente de una clave primaria

Esto es lo mismo que decir que se deben eliminar las dependencias transitivas de atributos no llaves respecto a la llave primaria, estando ya la relación en 2FN.

3.2.2.1 Dependencia transitiva

Sean A, B y C conjuntos de atributos de una relación R. Si B es dependiente funcionalmente de A y C lo es de B, entonces C depende transitivamente de A.



Este paso se ejecuta examinando todas las relaciones para ver si hay atributos no llaves que dependan unos de otros. Si se encuentran, se forma una nueva relación para ellos.

Veamos el siguiente caso; desde una lista donde se visualiza el tipo de sorteo, el año en que se sorteó, el nombre del ganador y la fecha de nacimiento del ganador ya que solo pueden entrar al sorteo personas mayores de edad.

SORTEO	AÑO	GANADOR	FECHA DE NACIMIENTO
Dia de la Madre	2010	Maria Rosales Neyra	25/02/1980
Navidad	2010	Carlos Rojas Silva	05/08/1975
Dia de la Madre	2011	Guadalupe Pardo Ferrer	20/12/1982
Navidad	2011	Carlos Garcia Fernandez	30/04/1984
Dia de la Madre	2012	Fernanda Jimenez Gutierrez	04/05/1978
Navidad	2012	Gustavo Lucio Deza	14/11/1980

Si observamos la tabla el atributo ganador depende funcionalmente de sorteo y el año en que se realizó el sorteo, mientras que la fecha de nacimiento depende funcionalmente de ganador y transitivamente de sorteo y año.

Para solucionar esta anomalía separaremos las tablas en **Sorteo** y **Participantes** de la siguiente forma:

Tabla: Sorteo

SORTEO	AÑO	GANADOR
Dia de la Madre	2010	Maria Rosales Neyra
Navidad	2010	Carlos Rojas Silva
Dia de la Madre	2011	Guadalupe Pardo Ferrer
Navidad	2011	Carlos Garcia Fernandez
Dia de la Madre	2012	Fernanda Jimenez Gutierrez
Navidad	2012	Gustavo Lucio Deza

Tabla: Participante

PARTICIPANTE	FECHA DE NACIMIENTO
Maria Rosales Neyra	25/02/1980
Carlos Rojas Silva	05/08/1975
Guadalupe Pardo Ferrer	20/12/1982
Carlos Garcia Fernandez	30/04/1984
Fernanda Jimenez Gutierrez	04/05/1978
Gustavo Lucio Deza	14/11/1980

Ahora las tablas se encuentran en 3FN separando el atributo transitivo a otra tabla, en donde este si dependa de la PK.

Veamos otro caso de 3FN; desde una lista donde se visualiza el número de DNI, nombre completo de un empleado, la especificación de su cargo y el monto de asignación salarial correspondiente según su cargo.

DNI	Empleado	Cargo	Salario
10584852	Jose Perez Roca	Jefe	S/. 3,500.00
20145175	Linda Torres Huaman	Contador	S/. 3,000.00
44578787	Carla Zegarra de la Cruz	Contador	S/. 3,000.00
36336955	Ivonne Sotomayor Rios	Administrativo	S/. 1,500.00

Como vemos el atributo “salario” no depende funcionalmente del numero de DNI del empleado; entonces procedemos a separar el salario en una nueva entidad.

Cargo	Salario
Jefe	S/. 3,500.00
Contador	S/. 3,000.00
Administrativo	S/. 1,500.00

Finalmente, la entidad Empleado quedaría de la siguiente manera:

DNI	Empleado	Cargo
10584852	Jose Perez Roca	Jefe
20145175	Linda Torres Huaman	Contador
44578787	Carla Zegarra de la Cruz	Contador
36336955	Ivonne Sotomayor Rios	Administrativo

La regla de la **Tercera Forma Normal** establece que todas las dependencias parciales se deben eliminar y separar dentro de sus propias tablas. Una dependencia parcial es un término que describe a aquellos datos que no dependen de la llave primaria de la tabla para identificarlos.

Finalmente, podemos decir que:

“Una tabla está normalizada en 3FN forma si todas las columnas que no son llave son funcionalmente dependientes por completo de la llave primaria y no hay dependencias transitivas. Una dependencia transitiva es aquella en la cual existen columnas que no son llave que dependen de otras columnas que tampoco son llave”.

3.2.3. Ejercicios de normalización

Para los siguientes casos determine los siguientes aspectos:

- Determine las dependencias funcionales suponiendo que inicialmente todos los datos se encuentran agrupados en una relación.
- Determine las llaves candidatas y señale la llave primaria.
- Diseñe el modelo de datos relacional normalizando las relaciones hasta la 3FN. Deben aparecer todas las relaciones que se vayan obteniendo en cada paso de normalización.

CASO 01: EMPRESA DE PROYECTOS



Se desea controlar la actividad de una empresa de proyectos. Para ello se cuenta con la siguiente información:

Del trabajador	Del proyecto
Documento de identidad	Numero de proyecto
Nombre completo	Fecha de inicio
Salario	Fecha de finalización

Además, se conoce las horas en plan de trabajo que cada trabajador dedica a cada proyecto. Se sabe que un trabajador puede laborar en varios proyectos y en un proyecto participan varios trabajadores.

Finalmente, cada proyecto tiene una fecha de inicio-terminación y cada trabajador tiene un nombre y un salario, aunque un mismo salario o nombre puede serlo de varios trabajadores.

CASO 02: CONTROL DE VENTAS



En una tienda se desea controlar las ventas y se conoce:

- Código de cada mercancía, que la identifica.
- Descripción de cada mercancía.
- País de procedencia de cada mercancía.
- Moneda de cada país.
- Precio de venta de cada mercancía de una calidad dada.

Una mercancía tiene varias calidades (1, 2, 3, etc.) y una calidad puede serlo de varias mercancías, esta procede de un país y de un país proceden varias mercancías. Una mercancía tiene una descripción, pero una descripción puede serlo de mercancías con diferentes códigos.

CASO 03: CONTROL DE PUBLICACIONES



En una empresa editorial de libros se desea controlar las publicaciones que se realizan. Para ello, se tiene la siguiente información:

- De cada autor se conoce su código, que lo identifica, nombre y dirección.
- Para cada libro se conoce su título, género literario y año en que fue escrito.
- Un libro puede ser escrito por varios autores y un autor puede escribir varios libros. Un libro puede tener varias ediciones y por cada edición de un libro, cada autor recibe un determinado pago.
- De cada edición de un libro se conoce su fecha y cantidad de ejemplares. (Considere que los títulos de los libros no se repiten).

CASO 04: FACTURA DE VENTA

En el formulario que se presenta a continuación determine lo siguiente:

- Normalice cada documento.
- Genere el modelo relacional correspondiente

R.T.N. 08019995298341			FACTURA №	VENDEDOR
DIA	MES	AÑO		
16	OCTUBRE	2007	00254	LENIN SALGADO
NOMBRE DEL CLIENTE			TIPO DE FACTURA	
AUTO FRENOS BOULEVARD			CONTADO	
CANT.	CODIGO	DESCRIPCION	PRECIO UNITARIO	TOTAL
10	7313570013	ADITIVO DE MOTOR 14.5 OZS	40.00	400.00
3	731357012	ETER	60.00	180.00
10	AD-35	BANDAS DENTADAS	2.50	25.00
25	8-94218-497-0	RELAY PARA ISUZU 5P 12V	25.00	625.00
75	40222-18000	TORNILLO DE RUEDA DATSUN 210	2.50	187.50
10	90311-180754	RETENEDOR RUEDA DELANTERA TOYOTA	75.00	750.00
2	4651	SILVINES CAJA DE 12 UNIDADES	50.00	100.00
30	1034AMARILLO	FOCOS DE STOP AMARILLO	7.20	216.00
10	AT-111-35	FUSES HACHITA MINIATURA 35 AMP	1.80	18.00
1- No se aceptan cambios ni devoluciones 2- Despues de la fecha de vencimiento de esta factura devengara un recargo equivalente al interes maximo del sistema bancario 3- Por cada cheque devuelto por el banco se cobrarán Lps. 300.00 de recargo			SUB TOTAL	L. 2,501.50
"ORIGINAL"			I.S.V.	L. 300.18
			TOTAL	L. 2,801.68

Figura 39: Factura
Fuente.- Tomado de http://www.sistematic21.com/factura_honduras.jpg

CASO 05: PRESUPUESTO DE OBRA

En el formulario que se presenta a continuación determine lo siguiente:

- a. Normalice cada documento.
- b. Genere el modelo relacional correspondiente

El siguiente documento muestra un análisis de precios y requerimientos tanto de materiales, personal, así como de los equipos y herramientas necesarios para realizar una obra.

PRESUPUESTO DE OBRA				
Nº PRESUPUESTO	FECHA / /			
CLIENTE	DIRECCIÓN	DISTRITO		
TELÉFONO				
MATERIALES				
DESCRIPCIÓN	UNIDAD	CANTIDAD	PRECIO UNITARIO	SUBTOTAL
PERSONAL				
CARGO DEL PERSONAL	UNIDAD	CANTIDAD	COSTO UNITARIO	SUBTOTAL
EQUIPOS Y HERRAMIENTAS				
DESCRIPCIÓN	UNIDAD	CANTIDAD	PRECIO UNITARIO	SUBTOTAL
UNIDADES VALIDAS: U1: UNI - U2: ROLL - U3:GLB - U4: GLN - U5:DÍA - U6:HRS				TOTAL PRESUPUESTO

Figura 40: Presupuesto de obra
Fuente. - Tomado desde la aplicación Excel

CASO 06: CONTROL DE PRODUCCIÓN

En el formulario que se presenta a continuación determine lo siguiente:

- Normalice cada documento.
- Genere el modelo relacional correspondiente

El siguiente documento muestra el control de prendas que realiza un determinado empleado, así como el control del tiempo y las ocurrencias por semana.

CONTROL DE PRODUCCION								
NUMERO DE EMPLEADO			SUPERVISOR					
NOMBRE DEL EMPLEADO								
NUMERO DE SEMANA								
CONTROL DE PRENDAS								
TIPO DE PRENDA	DOM	LUN	MAR	MIÉ	JUE	VIE	SÁB	
TOTAL DE PRENDAS A PAGAR								
CONTROL DE TIEMPO								
ITEM	TIPO HORA	DOM	LUN	MAR	MIÉ	JUE	VIE	SÁB
TOTAL DE HORAS A PAGAR								
OCURRENCIAS DE LA SEMANA								
ITEM	DESCRIPCIÓN DE LA OCURRENCIA				DÍA	HORA		

Figura 41: Hoja de control de producción
Fuente.- Tomado desde la aplicación Excel

3.3. Normalización parte 3

3.3.1. Detalle del detalle / ítems

Un paso importante dentro de la normalización de documentos, es la aplicación del detalle de ítems en la normalización, para eso veremos dos casos el primero aplicará las 3FN a un documento que presente detalle-ítem; mientras que el segundo aplicará las 3FN a un documento que presente detalle del detalle.

Caso: INFORME TÉCNICO (Detalle Items)

Del siguiente informe aplique las 3 formas normales.

INFORME TECNICO			
Nº			
CURSO			
SEMESTRE		SECCIÓN	
ESPECIALIDAD			
ITEM	PROBLEMAS		
1			
2			
3			
4			
ITEM	CAUSAS		
1			
2			
3			
4			
ITEM	EFFECTOS		
1			
2			
3			
4			
ITEM	ALTERNATIVAS		
1			
2			
3			
4			

Figura 42: Hoja de informe técnico
Fuente.- Tomado desde la aplicación Excel

Los números de ítems (problemas, causas, efectos y alternativas) se repiten en diferentes informes técnicos. Para dar solución al caso debemos seguir los siguientes pasos:

Pasos:

1. LISTANDO TODOS LOS ATRIBUTOS

INFORME (<u>NRO_INF</u> , <u>CURSO_INF</u> , <u>SEMESTRE_INF</u> , <u>SECCION_INF</u> , <u>ESPECIALIDAD_INF</u> , <u>FECHA_INF</u> ,

```
NRO_ITEM_PRO, DESC_PRO,  
NRO_ITEM_CAU, DESC_CAU,  
NRO_ITEM_EFE, DESC_EFE,  
NRO_ITEM_ALT, DESC_ALT )
```

2. IDENTIFICANDO ANOMALÍA DE DATOS NO ATÓMICOS

```
INFORME ( NRO_INF, CURSO_INF, SEMESTRE_INF, SECCION_INF,  
          ESPECIALIDAD_INF, FECHA_INF,  
          NRO_ITEM_PRO, DESC_PRO,  
          NRO_ITEM_CAU, DESC_CAU,  
          NRO_ITEM_EFE, DESC_EFE,  
          NRO_ITEM_ALT, DESC_ALT )
```

3. APLICANDO 1FN

```
INFORME ( NRO_INF, CURSO_INF, SEMESTRE_INF, SECCION_INF,  
          ESPECIALIDAD_INF, FECHA_INF )  
  
PRO-INF ( NRO_INF, NRO ITEM PRO, DESC_PRO )  
  
CAU-INF ( NRO_INF, NRO ITEM CAU, DESC_CAU )  
  
EFEC-INF ( NRO_INF, NRO ITEM EFE, DESC_EFE )  
  
ALT-INF ( NRO_INF, NRO ITEM ALT, DESC_ALT )
```

4. APLICANDO 2FN

Identificando anomalía de dependencia transitiva:

```
INFORME ( NRO_INF,  
          CURSO_INF,  
          SECCION_INF, ESPECIALIDAD_INF,  
          SEMESTRE_INF,  
          FECHA_INF )  
  
PRO-INF ( NRO_INF, NRO ITEM PRO, DESC_PRO )  
  
CAU-INF ( NRO_INF, NRO ITEM CAU, DESC_CAU )  
  
EFEC-INF ( NRO_INF, NRO ITEM EFE, DESC_EFE )  
  
ALT-INF ( NRO_INF, NRO ITEM ALT, DESC_ALT )
```

5. APLICANDO 3FN

```

INFORME ( NRO_INF, SEMESTRE_INF, FECHA_INF,
          COD_CUR, COD_ESP, COD_SEC )

CURSO ( COD_CUR, DESCRIPCIÓN )

ESPECIALIDAD ( COD_ESP, DESCRIPCIÓN )

SECCION ( COD_SEC, DESCRIPCIÓN )

PRO-INF ( NRO_INF, NRO_ITEM_PRO, DESC_PRO )

PRO-INF ( NRO_INF, NRO_ITEM_PRO, DESC_PRO )

CAU-INF ( NRO_INF, NRO_ITEM_CAU, DESC_CAU )

EFEC-INF ( NRO_INF, NRO_ITEM_EFE, DESC_EFE )

ALT-INF ( NRO_INF, NRO_ITEM_ALT, DESC_ALT )

```

Caso: INFORME TÉCNICO (Detalle de Detalle)

Del siguiente informe aplique las 3 formas normales.

CONTROL DE ATENCIÓN				
1. INFORMACIÓN INICIAL				
ESPECIALIDAD	<input type="text"/>	FECHA	/	<input type="text"/>
DOCTOR	<input type="text"/>			
2. INFORMACIÓN DE ATENCIÓNES				
PACIENTE	DIAGNOSTICO	MEDICAMENTO		
		CÓDIGO	DESCRIPCIÓN	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	

Figura 43: Hoja de control de atención
Fuente.- Tomado desde de la aplicación Excel

Para dar solución al caso debemos seguir los siguientes pasos:

Pasos:

1. LISTANDO TODOS LOS ATRIBUTOS

```
CONTROL_ATENCION(NRO_CON, COD_ESP, DESC_ESP,
COD_DOC, NOM_DOC,
FECHA_CON,
COD_PAC, NOM_PAC,
DIA_PAC,
COD_MED, DESC_MED )
```

Donde:

Atributo	Descripción
NRO_CON	Número de control de atención por doctor.
COD_ESP	Código de la especialidad.
DESC_ESP	Descripción de la especialidad del doctor.
FEC_CON	Fecha de registro del control de atención.
COD_PAC	Código del paciente.
NOM_PAC	Nombre completo del paciente.
DIA_PAC	Diagnóstico especificado por el doctor.
COD_MED	Código de medicamento.
DESC_ME	Descripción del medicamento especificado por el doctor según el diagnóstico realizado.

2. IDENTIFICANDO ANOMALÍAS DE DATOS NO ATÓMICO

```
CONTROL_ATENCION(NRO_CON, COD_ESP, DESC_ESP,
COD_DOC, NOM_DOC,
FECHA_CON,
```

```
COD_PAC, NOM_PAC,
DIA_PAC,
```

```
COD_MED, DESC_MED
```

Grupos repetidos

3. APLICANDO FN1

```
CONTROL_ATENCION(NRO_CON, COD_ESP, DESC_ESP,
COD_DOC, NOM_DOC,
FECHA_CON )
```

```
CONTROL_ATENCION_PACIENTE(NRO_CON, COD_PAC, NOM_PAC )
```

```
CONTROL_ATENCION_MEDICAMENTO(NRO_CON, COD_MED, DESC_MED )
```

4. IDENTIFICANDO ANOMALÍA DE DEPENDENCIA PARCIAL

```

CONTROL_ATENCION(NRO_CON, COD_ESP, DESC_ESP,
                  COD_DOC, NOM_DOC,
                  FECHA_CON )

CONTROL_ATENCION_PACIENTE(NRO_CON, COD_PAC, NOM_PAC )

CONTROL_ATENCION_MEDICAMENTO(NRO_CON, COD_MED, DESC_MED)

```

Esto se debe a que **NOM_PAC** solo depende de la llave **COD_PAC** y no de **NRO_CON**. Y que **DESC_MED** solo depende de **COD_MED** y no de **NRO_CON**.

5. APlicando FN2

```

CONTROL_ATENCION(NRO_CON, COD_ESP, DESC_ESP,
                  COD_DOC, NOM_DOC,
                  FECHA_CON )

CONTROL_ATENCION_PACIENTE(NRO_CON, COD_PAC )

PACIENTE(COD_PAC, NOM_PAC )

CONTROL_ATENCION_MEDICAMENTO(NRO_CON, COD_MED )

MEDICAMENTO(COD_MED, DESC_MED )

```

6. IDENTIFICANDO ANOMALÍA DE DEPENDENCIA TRANSITIVA

```

CONTROL_ATENCION(NRO_CON, COD_ESP,
                  DESC_ESP,
                  COD_DOC,
                  NOM_DOC,
                  FECHA_CON )

CONTROL_ATENCION_PACIENTE(NRO_CON, COD_PAC )

PACIENTE(COD_PAC, NOM_PAC )

CONTROL_ATENCION_MEDICAMENTO(NRO_CON, COD_MED )

MEDICAMENTO(COD_MED, DESC_MED )

```

Esto se debe a que el atributo **DESC_ESP** depende del atributo **COD_ESP** y no de la llave **NRO_CON**. Mientras que el atributo **NOM_DOC** depende del atributo **COD_DOC** y no de la llave principal **NRO_CON**.

7. APPLICANDO FN3

```
CONTROL_ATENCION(NRO CON, COD_ESP, COD_DOC, FECHA_CON )  
ESPECIALIDAD(COD ESP, DESC_ESP )  
DOCTOR(COD DOC, NOM_DOC )  
CONTROL_ATENCION_PACIENTE(NRO CON, COD PAC )  
PACIENTE(COD PAC, NOM_PAC )  
CONTROL_ATENCION_MEDICAMENTO(NRO CON, COD MED )  
MEDICAMENTO(COD MED, DESC_MED )
```

3.4. Implementación de detalle de detalle en SQL Server 2014

3.4.1 Obtención del modelo lógico-global de los datos a partir del DER

Para obtener el modelo lógico global de los datos, según el enfoque relacional a partir del **DER**, se sigue un procedimiento que iremos describiendo paso a paso y aplicándolo, así mismo, al ejemplo siguiente:

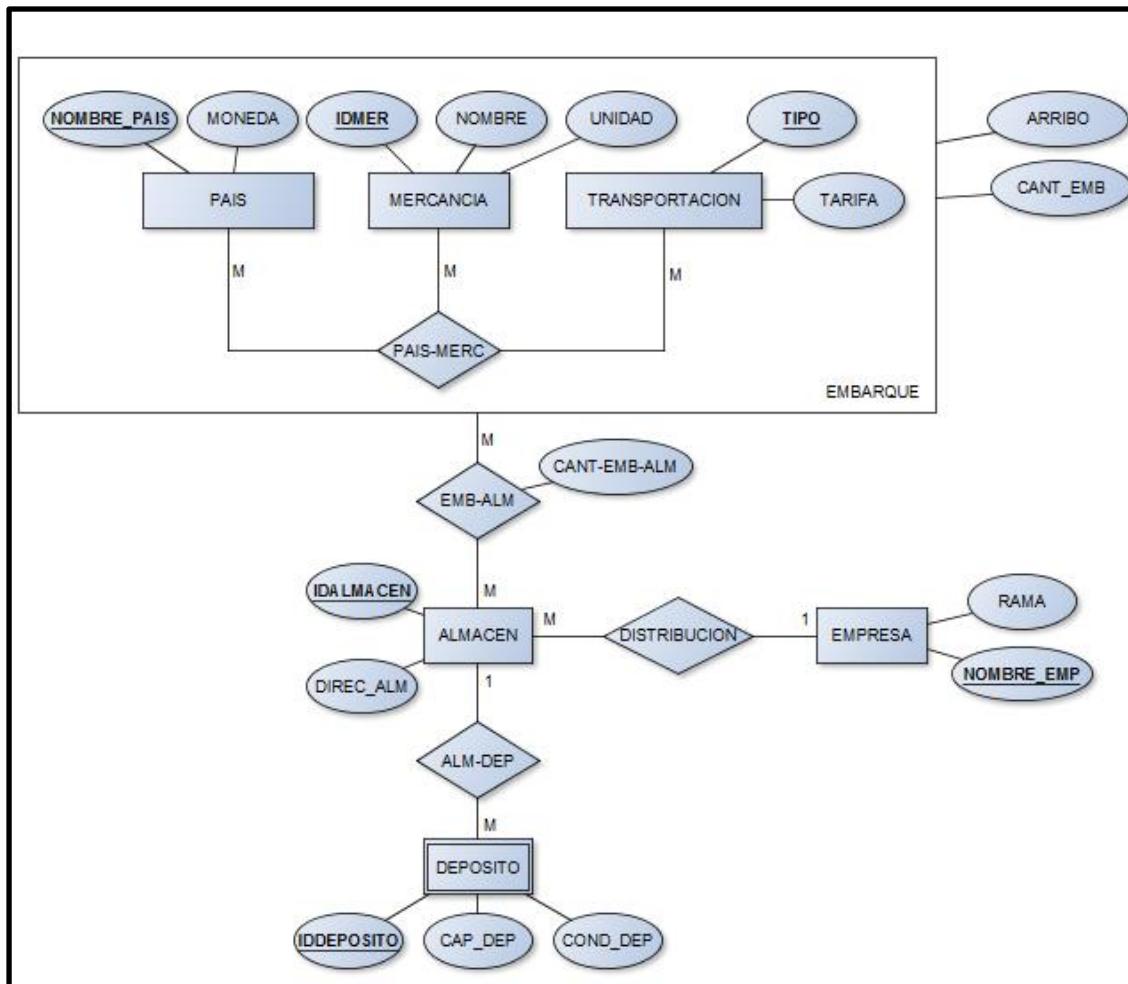


Figura 44: DER Movimiento Mercantil
Fuente.- Tomado desde la aplicación yEd

En el DER anterior se representa el movimiento mercantil de un organismo. En el organismo existen mercancías de las que se conoce su código, nombre y unidad de medida. Las mercancías proceden de diferentes países de los que se sabe nombre y tipo de moneda. Para la transportación de las mercancías existen diversas formas, cada una de las cuales se caracteriza por su tipo (barco, avión, tren, etc.) y tarifa.

Para cada mercancía de diferentes países existen diferentes formas de transportación; para cada país existen diferentes mercancías que son transportadas en diferentes formas de transportación; y una forma de transportación puede serlo de diferentes mercancías de diferentes países. Una mercancía procedente de un país y transportada de una forma dada constituye un embarque, y para éste se conoce su fecha de arribo y cantidad.

Un embarque se distribuye entre diferentes almacenes y en un almacén se tienen diferentes embarques, cada uno en cierta cantidad. De cada almacén se tiene su código y dirección. Un almacén envía sus productos a una sola empresa y cada empresa recibe productos de diferentes almacenes. Una empresa se caracteriza por su nombre y rama económica.

Cada almacén tiene distintos depósitos subordinados. De cada depósito se conoce su número (que se puede repetir en diferentes almacenes), capacidad y condiciones técnicas.

Pasos:

1. Representar las entidad regulares

PAÍS	(<u>NOMBRE_PAIS</u> , MONEDA)
MERCANCÍA	(<u>IDMER</u> , NOMBRE, UNIDAD)
TRANSPORTACIÓN	(<u>TIPO</u> , TARIFA)
ALMACÉN	(<u>IDALMACEN</u> , DIREC_ALM)
EMPRESA	(<u>NOMBRE_EMP</u> , RAMA)

2. Representar las entidades agregadas

Representar, en una tabla relacional, cada entidad agregada con sus correspondientes atributos (entre ellos un identificador si fue definido) y las llaves de las entidades que forman la agregación. La llave primaria de esta tabla es la llave de la relación que da origen a la agregación si no hay un identificador definido; pero si lo hay, entonces la llave será dicho identificador

EMBARQUE (NOMBRE_PAIS, IDMER, TIPO, ARRIBO, CANT_EMB)

Nótese que la llave estaría formada por las llaves de las 3 entidades regulares que intervienen en la agregación.

Pero, podía haberse definido un identificador para la entidad embarque. (Supóngase añadido en el DER un atributo de la entidad embarque, que sería su llave: IDEMBARQUE). Entonces se añadiría como atributo llave en la agregación y los 3 atributos NOMBRE_PAIS, IDMER y TIPO permanecerían en la relación, pero no como llave, así por ejemplo:

EMBARQUE (IDEMBARQUE, NOMBRE_PAIS, IDMER, TIPO, ARRIBO, CANT_EMB)

3. Representar las entidades generalizadas y especializadas

Representar cada entidad generalizada en una tabla que contendrá sus atributos (sólo los de la generalizada) y, entre ellos, la llave.

Representar cada entidad especializada en una tabla que contendrá, como llave primaria, la llave de la generalización y los atributos propios sólo de la especialización.

Como este ejemplo no incluye ninguna generalización /especialización, analicemos cómo se llevaría a tablas el siguiente DER, en que se tiene la entidad generalizada estudiante y los casos especiales, becario y practicante.

4. Finalmente, las entidades resultantes son:

PAÍS	(NOMBRE_PAÍS, MONEDA)
MERCANCÍA	(IDMER, NOMBRE, UNIDAD)
TRANSPORTACIÓN	(TIPO, TARIFA)
EMPRESA	(NOMBRE_EMP, RAMA)
ALMACÉN	(IDALMACEN, DIREC_ALM, NOMBRE_EMP)
DEPOSITO	(IDALMACEN, IDDEPOSITO, CAP_DEP, COND_DEP)
EMBARQUE	(NOMBRE_PAÍS, IDMER, TIPO, ARRIBO, CANT_EMB)
EMBARQUE-ALMACEN	(NOMBRE_PAÍS, IDMER, TIPO, IDALMACEN, CANT_EMB_ALM)

5. El diagrama de base de datos en SQL es el siguiente:

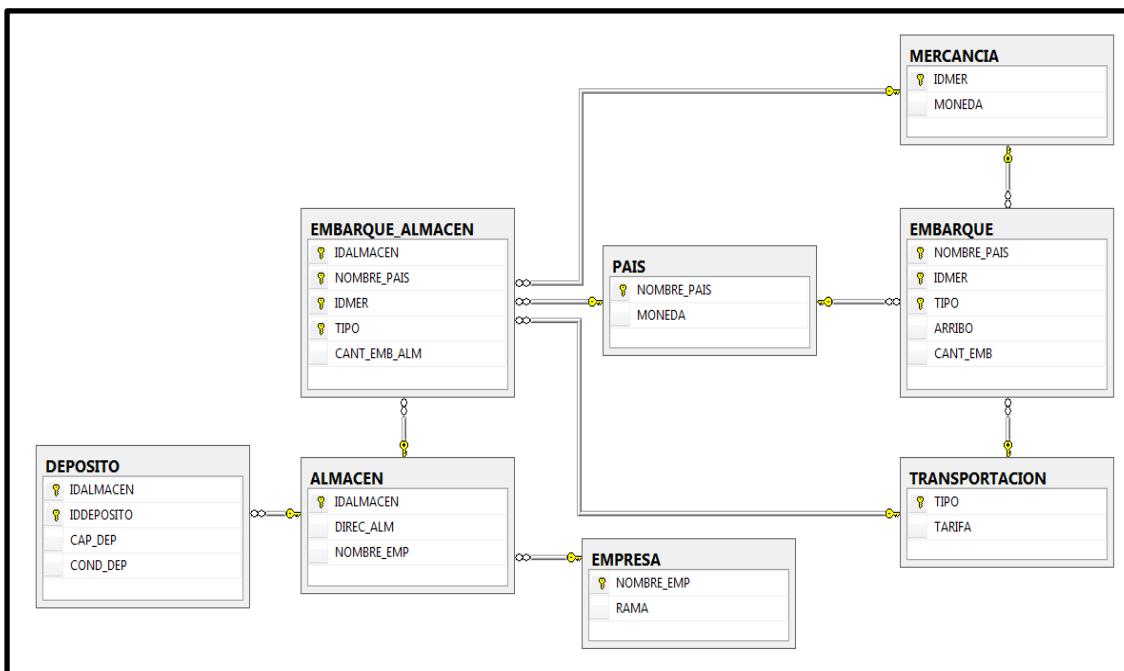


Figura 45: Diagrama de base de datos Movimiento Mercantil implementado en SQL
Fuente.- Tomado desde SQL Server 2014

Script en SQL Server 2014

```

USE MASTER
GO

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

CREATE DATABASE MERCANTIL
GO

USE MERCANTIL
GO

CREATE TABLE PAÍS(
    NOMBRE_PAÍS      VARCHAR(30) NOT NULL PRIMARY KEY,
    MONEDA          VARCHAR(30) NOT NULL
)

```

```
)  
GO  
CREATE TABLE MERCANCIA(  
    IDMER           CHAR(5)      NOT NULL PRIMARY KEY,  
    MONEDA          VARCHAR(30) NOT NULL  
)  
GO  
CREATE TABLE TRANSPORTACION(  
    TIPO            VARCHAR(30) NOT NULL PRIMARY KEY,  
    TARIFA          MONEY       NOT NULL  
)  
GO  
CREATE TABLE EMPRESA(  
    NOMBRE_EMP     VARCHAR(30) NOT NULL PRIMARY KEY,  
    RAMA            VARCHAR(30) NOT NULL  
)  
GO  
CREATE TABLE ALMACEN(  
    IDALMACEN      INT         NOT NULL PRIMARY KEY,  
    DIREC_ALM     VARCHAR(50) NOT NULL,  
    NOMBRE_EMP     VARCHAR(30) NOT NULL REFERENCES EMPRESA  
)  
GO  
CREATE TABLE DEPOSITO(  
    IDALMACEN      INT         NOT NULL REFERENCES ALMACEN,  
    IDDEPOSITO     VARCHAR(50) NOT NULL,  
    CAP_DEP        DECIMAL(8,2)NOT NULL,  
    COND_DEP       VARCHAR(50) NOT NULL,  
    PRIMARY KEY (IDALMACEN, IDDEPOSITO)  
)  
GO  
CREATE TABLE EMBARQUE(  
    NOMBRE_PAIS    VARCHAR(30) NOT NULL REFERENCES PAIS,  
    IDMER          CHAR(5)    NOT NULL REFERENCES MERCANCIA,  
    TIPO           VARCHAR(30) NOT NULL REFERENCES TRANSPORTACION,  
    ARRIBO          VARCHAR(50) NOT NULL,  
    CANT_EMB       INT        NOT NULL,  
    PRIMARY KEY (NOMBRE_PAIS, IDMER, TIPO)  
)  
GO  
CREATE TABLE EMBARQUE_ALMACEN(  
    IDALMACEN      INT         NOT NULL REFERENCES ALMACEN,  
    NOMBRE_PAIS    VARCHAR(30) NOT NULL REFERENCES PAIS,  
    IDMER          CHAR(5)    NOT NULL REFERENCES MERCANCIA,  
    TIPO           VARCHAR(30) NOT NULL REFERENCES TRANSPORTACION,  
    CANT_EMB_ALM   INT        NOT NULL,  
    PRIMARY KEY (IDALMACEN, NOMBRE_PAIS, IDMER, TIPO)  
)  
GO
```

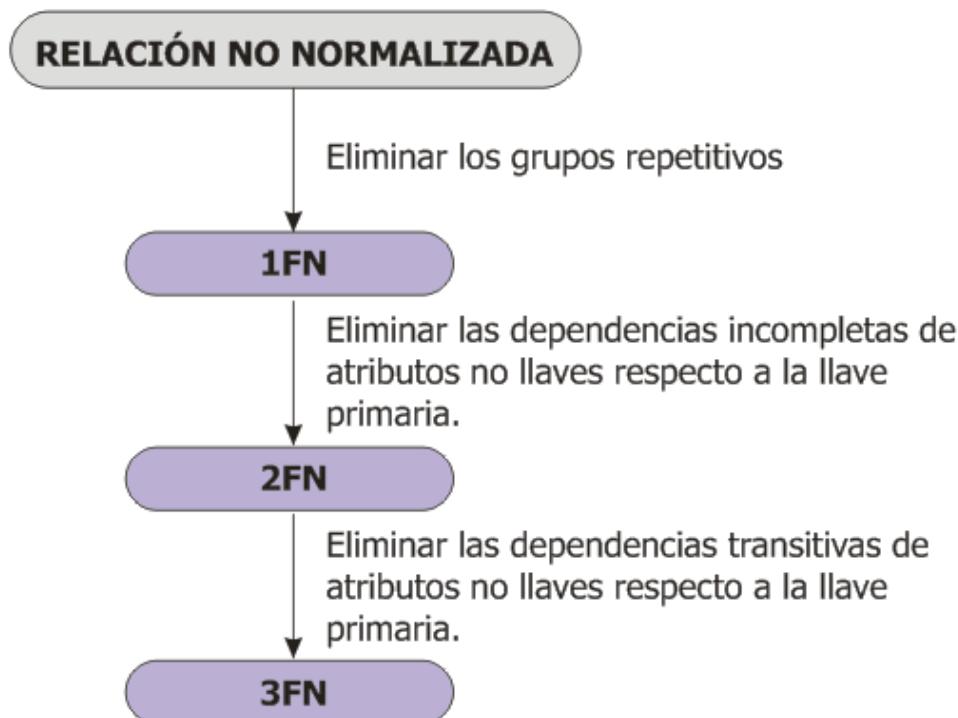
Resumen

1. En el modelo relacional, tanto las entidades como las relaciones se representan como relaciones (tablas). Las ocurrencias (de las entidades o de las relaciones) se almacenan como filas y las columnas son los atributos (de las entidades o de las relaciones).
2. Para toda tabla del modelo relacional se cumple lo siguiente:
 - a) No hay dos filas (tuplas) iguales.
 - b) El orden de las filas no es significativo.
 - c) El orden de las columnas no es significativo.
 - d) Cada valor de un atributo es un dato atómico (o elemental).
3. La normalización es la expresión formal del modo de realizar un buen diseño. Con la aplicación de la teoría de la normalización se evitan los problemas de actualización.
4. La normalización está constituida por pasos sucesivos:
 - a) Primera Forma Normal (1FN)
 - b) Segunda Forma Normal (2FN)
 - c) Tercera Forma Normal (3FN)
 - d) Forma Normal de *Boyce-Codd* (FNBC)

“Para que una relación esté en 2FN tiene que estar en 1FN, para que esté en 3FN tiene que estar en 2FN y así sucesivamente”.

5. Una relación está en 1FN si no incluye ningún grupo repetitivo.
6. Para transformar una relación que no está en 1FN en otras que sí lo estén y representen mejor el , se deben eliminar los grupos repetitivos, para lo cual:
 - a) Se dejan en la relación original sólo los atributos que no son repetitivos.
 - b) Se extraen en una nueva relación los grupos repetitivos, escogiendo adecuadamente la llave primaria, que siempre estará formada por la llave de la relación original más uno o varios de los atributos repetitivos que permitan distinguir cada valor diferente en dichos campos.
7. Dada una relación R, se dice que el atributo Y de R es funcionalmente dependiente del atributo X de R, si y sólo si cada valor X en R tiene asociado a él, precisamente, un valor de Y en R en cualquier momento del tiempo.
8. El atributo Y es funcionalmente y completamente dependiente del atributo X, si es funcionalmente dependiente de X y no es funcionalmente dependiente de algún subconjunto de X.
9. Una relación R se dice que está en 2FN si está en 1FN y si, y sólo si, los atributos no llaves (ni primarias, ni candidatas) de R, si los hubiese, son funcional y completamente dependientes de la llave primaria de R.

10. Una relación R está en 3FN si está en 2FN y si, y sólo si, los atributos no llaves son independientes de cualquier otro atributo no llave primaria. Esto es lo mismo que decir que se deben eliminar las dependencias transitivas de atributos no llaves respecto a la llave primaria, estando ya la relación en 2FN.
11. Un determinante es cualquier atributo o conjunto de atributos del cual depende funcional y completamente cualquier otro atributo. Mejor dicho, la parte izquierda de la implicación cuando la dependencia funcional es completa.
12. De modo resumido, se puede decir que los pasos que se deben dar en el proceso de normalización son:



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

- <http://sistemas.itlp.edu.mx/tutoriales/basedat1/>
En esta página web hallará conceptos complementarios a los mostrados en el manual sobre la el Modelo Relacional, así como ejemplos didácticos de su aplicación en los lenguajes de programación de base de datos.
- <http://www.mitecnologico.com/Main/ModeloErYNormalizacion>
En esta página web encontrará definiciones complementarias a la normalización de datos.
- http://www-gist.det.uvigo.es/~martin/bd/ejemplo_norm.pdf
En esta página web se muestra la normalización de una fuente de datos paso a paso.
- <http://sistemas.itlp.edu.mx/tutoriales/basedat1/>
En esta página web se muestra conceptos complementarios de cada paso de la normalización de datos.

- http://www-gist.det.uvigo.es/~martin/bd/ejemplo_norm.pdf
En esta página web se muestra la normalización de una fuente de datos paso a paso.



MANIPULACIÓN DE DATOS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno modifica el contenido de una base de datos mediante la aplicación de las tres sentencias del Lenguaje de Manipulación de Datos (DML).

TEMARIO

4.1 Tema 16 : Restricciones

- 4.1.1 : Restricción DEFAULT
- 4.1.2 : Restricción CHECK
- 4.1.3 : Restricción UNIQUE
- 4.1.4 : Restricción IDENTITY

4.2 Tema 17 : Sentencias DML

- 4.2.1 : Inserción de datos INSERT
- 4.2.2 : Inserción de datos UPDATE
- 4.2.3 : Inserción de datos DELETE

4.3 Tema 18 : Integración de sentencias SQL

ACTIVIDADES PROPUESTAS

- Los alumnos implementan cláusulas de restricciones a las tablas de una base de datos.
- Los alumnos implementan sentencias DML sobre una base de datos.

4.1. Restricciones

4.1.1. 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:

4.1.1.1 Al crear una tabla

```
CREATE TABLE NOMBRE_TABLA(
    COLUMNNA1      TIPO      DEFAULT VALOR
    COLUMNNA2      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', '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

```

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 46: Listado de empleados desde la tabla EMPLEADO
Fuente.- Tomado de SQL SERVER 2014

4.1.1.2 Agregar 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	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1 EMPLEADO	DEFAULT on column EST_CIVIL_EMP	DF_EMPLEADO_EST_CI_3B75D760	(n/a)	(n/a)	(n/a)	(n/a)	(S)
	DEFAULT on column NUM_HIJOS_EMP	DF_EMPLEADO_NUM_HI_3D5E1FD2	(n/a)	(n/a)	(n/a)	(n/a)	(0)
	DEFAULT on column SUELDO_EMP	DF_EMPLEADO_SUELDO_3C69FB99	(n/a)	(n/a)	(n/a)	(n/a)	(0)
	DEFAULT on column TELEFONO_EMP	DF_EMPLEADO_TELEFO_3A81B327	(n/a)	(n/a)	(n/a)	(n/a)	(000-000000)
	PRIMARY KEY (clustered)	PK_EMPLEADO_E014C316ABC1380	(n/a)	(n/a)	(n/a)	(n/a)	IDEMPLEADO

Figura 47: Listado de valores por defecto asignada a la tabla EMPLEADO

Fuente.- Tomado de SQL SERVER 2014

4.1.2. 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:

4.1.2.1 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','Salinas',
                            '582-158258','D',2500,-1);

```

4.1.2.2 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:

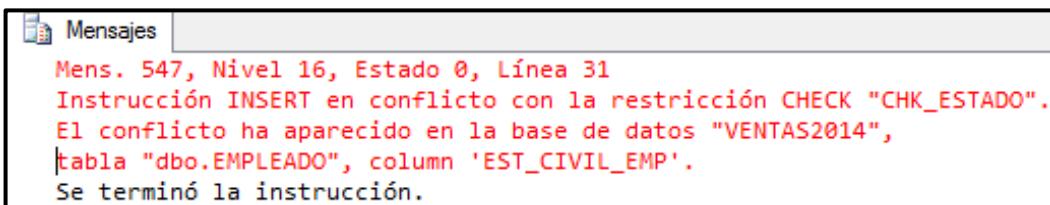


Figura 48: Rompiendo del constraint estado civil
Fuente.- Tomado de SQL SERVER 2014

- 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:

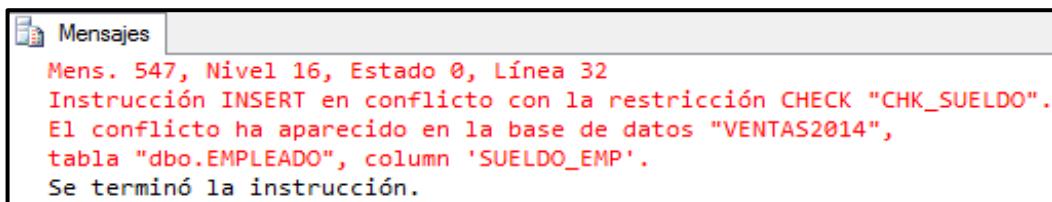


Figura 49: Rompimiento del constraint sueldo del empleado
Fuente.- Tomado de SQL SERVER 2014

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

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

El mensaje desde el servidor sería:

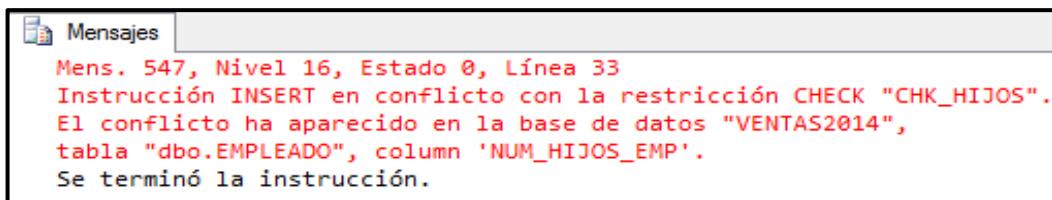


Figura 50: Rompimiento del constraint número de hijos del empleado
Fuente.- Tomado de SQL SERVER 2014

4.1.2.3 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 51: 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);
```

4.1.3. 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:

4.1.3.1 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, 'mreenteria@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 52: Rompimiento del constraint UNIQUE
Fuente.- Tomado de SQL SERVER 2014

4.1.3.2 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, 'mrenteria@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'

```

4.1.4. Restricción IDENTITY

Asigna un valor incremental a una sola columna dentro de una tabla, la asignación Identity lo podra 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_DECREMENTO)
)

```

El valor de inicio, indica el punto de partida de la secuencia de números; debemos mencionar que no siempre será el numero 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 53: Visualizando los registros de la tabla EMPLEADO
Fuente.- Tomado de SQL SERVER 2014

4.2. Sentencias DML

4.2.1. 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 un 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.

4.2.1.1 Formas de inserción de registros

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

4.2.1.1.1 Inserción de una sola fila de registro:

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	Jose	Blanco	525-5685	C	1850.00	jblanco@cibertec.edu.pe
2	Maria	Rengifo	985-698569	S	2500.00	mrenfigo@cibertec.edu.pe
3	Milagros	Acosta	998-562563	D	1100.00	macosta@cibertec.edu.pe
4	Angela	Torres	NULL	S	1800.00	atordes@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. Asimismo 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'

-- Envíando 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 un 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 qué 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	Jose	Blanco	525-5685	C	1850.00	jblanco@cibertec.edu.pe
2	Maria	Rengifo	985-698569	S	2500.00	mrenfigo@cibertec.edu.pe
3	Milagros	Acosta	998-562563	D	1100.00	macosta@cibertec.edu.pe
4	Janeth	Rengifo	985-698569	S	2500.00	mrenfigo@cibertec.edu.pe
5	Milagros	Acosta	998-562563	D	1100.00	macosta@cibertec.edu.pe
6	7	Lucero	Erazo	985-966858	S	1450.00
7	100	Carlos	Ramos	(01)522-1523	S	1520.00
8	101	David	Femandez	(01)425-5454	C	2500.00
9	102	Selena	Susaya	(01)421-1523	V	2200.00
10	103	Jhon	Hernandez	(01)485-9588	C	1800.00
11	104	Carlos	Pacheco	(01)523-1523	C	1750.00

Figura 56: 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.

4.2.2. 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 COLUMNNA = 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 columnas, en caso se quiera actualizar mas de dos columnas se deberá separar por comas:

```
SET COLUMNNA=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.

4.2.3. 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.

4.3. Integración de sentencias SQL

Actividad:

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

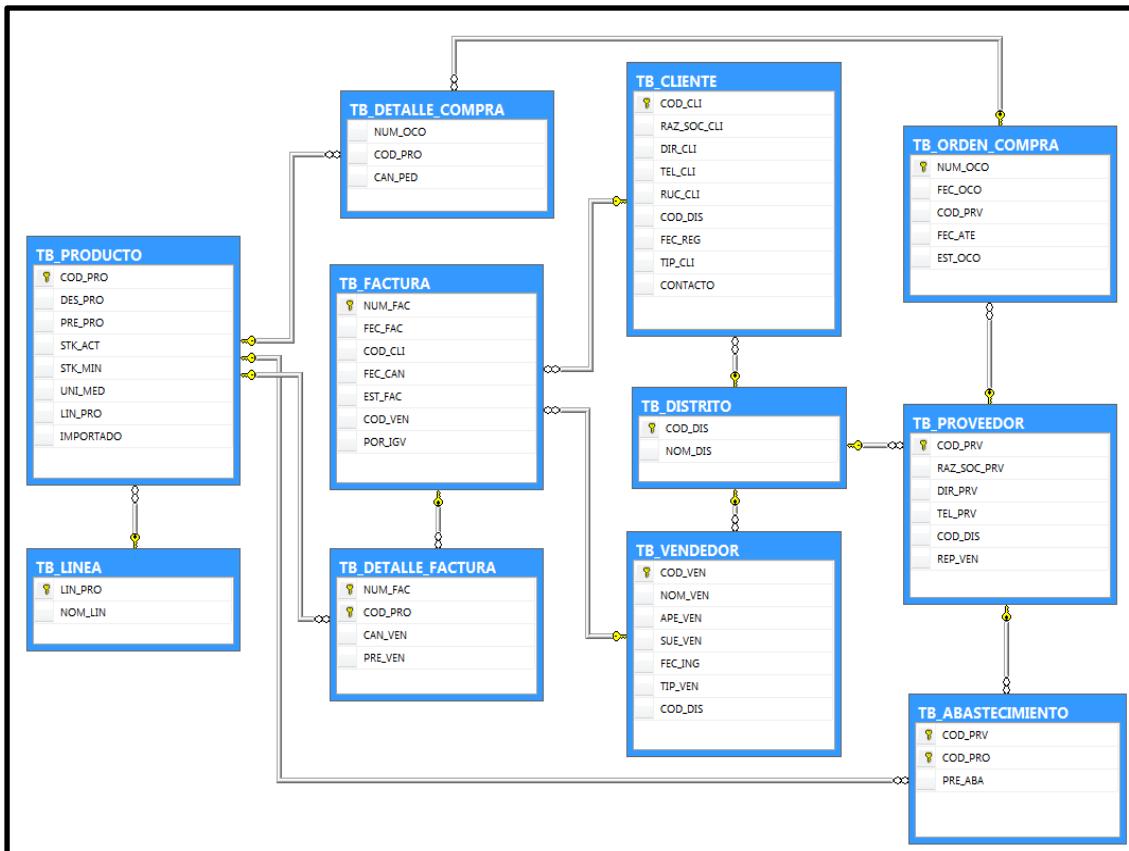


Figura 57: Visualizando el diagrama de base de datos COMERCIO
Fuente.- Tomado de SQL SERVER 2014

1. 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)  NOT NULL,
    PRIMARY KEY(COD_DIS)
);

CREATE TABLE TB_CLIENTE(
    COD_CLI int          NOT NULL,
    RAZ_SOC_CLI varchar(50) NOT NULL,
    DIR_CLI varchar(100),
    TEL_CLI int,
    RUC_CLI varchar(15),
    COD_DIS int,
    FEC_REG date,
    TIP_CLI varchar(50),
    CONTACTO varchar(100),
    PRIMARY KEY(COD_CLI)
);

CREATE TABLE TB_VENDEDOR(
    COD_VEN int          NOT NULL,
    NOM_VEN varchar(50)  NOT NULL,
    APE_VEN varchar(50),
    SUE_VEN decimal(10,2),
    FEC_ING date,
    TIP_VEN varchar(50),
    COD_DIS int,
    PRIMARY KEY(COD_VEN)
);

CREATE TABLE TB_ABASTECIMIENTO(
    COD_PRV int          NOT NULL,
    COD_PRO int          NOT NULL,
    PRE_ABA decimal(10,2),
    PRIMARY KEY(COD_PRV, COD_PRO)
);

CREATE TABLE TB_PRODUCTO(
    COD_PRO int          NOT NULL,
    DES_PRO varchar(100) NOT NULL,
    PRE_PRO decimal(10,2),
    STK_ACT int,
    STK_MIN int,
    UNI_MED varchar(50),
    UNI_PRO varchar(50),
    IMPORTADO varchar(50),
    PRIMARY KEY(COD_PRO)
);

CREATE TABLE TB_LINEA(
    LIN_PRO int          NOT NULL,
    NOM_LIN varchar(50)  NOT NULL,
    PRIMARY KEY(LIN_PRO)
);

CREATE TABLE TB_DETALLE_FACTURA(
    NUM_FAC int          NOT NULL,
    FEC_FAC date,
    COD_CLI int,
    FEC_CAN date,
    EST_FAC varchar(50),
    COD_VEN int,
    POR_JGV decimal(10,2),
    PRIMARY KEY(NUM_FAC)
);

CREATE TABLE TB_DETALLE_COMPRA(
    NUM_OCO int          NOT NULL,
    FEC_OCO date,
    COD_PRV int,
    FEC_ATE date,
    EST_OCO varchar(50),
    PRIMARY KEY(NUM_OCO)
);

```

```
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_IVG 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('MII','Uni','Cie','Doc'))
GO

/* INSERCIONES CORRECTAS */
INSERT TB_PRODUCTO VALUES
    ('P001','Papel Bond A-4',35.00,200,1500,
     'MII',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', 'Serviesma', '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

```

6. 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 Martín 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,
                           '2014-05-07', 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 Aguirre')
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 Gonzales')
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
Rodriguez')
INSERT TB_PROVEEDOR VALUES ('PR09', 'Dipropor',
                             'Av. Del Aire 901',
                             '4742046', 'D24', 'Roberto
Ronceros')
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 Zegarra')
INSERT TB_PROVEEDOR VALUES ('PR12', 'Sumigraf',
                             'Av. Manco Cápac 754',
                             '3320343', 'D13', 'Karinna Paredes')
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', 'Serviesma', '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 58: 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 59: 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 60: 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 61: 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 62: 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 63: 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 64: 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 65: 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 66: 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	Borrador de Tinta	8.00	100	500	Doc	3	FALSO
10	P010	Borrador 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 67: 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

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

EMPLEADO		
	Column Name	Condensed Type
1	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 68: 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 69: 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 validos
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 70: 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 seria:

	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 71: Visualizando los registros de la tabla EMPLEADO después de la actualización
Fuente.- Tomado de SQL SERVER 2014

El resultado seria 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:

	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

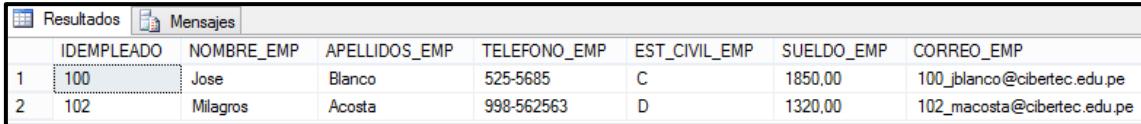
Figura 72: Visualizando los registros de la tabla EMPLEADO después de la actualización en el correo electrónico
Fuente.- Tomado de SQL SERVER 2014

- 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



	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 73: Visualizando los registros de la tabla EMPLEADO después de la eliminación

Fuente.- Tomado de SQL SERVER 2014

Resumen

1. Las restricciones se aplican a reglas de negocio especificadas en el análisis del proceso de negocio.
2. 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.
3. 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.
4. La cláusula UNIQUE permite definir un valor único entre un conjunto de valores contenidos en una tabla.
5. Las sentencias de tipo DML permiten manipular la información contenida en una tabla de base de datos; es a partir de aquí que podemos registrar, actualizar y eliminar.

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

- <http://www.ub.edu.ar/catedras/ingenieria/Datos/capitulo4/cap43.htm>
Sentencias DDL, DML del SQL Server.
- <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



IMPLEMENTACIÓN DE CONSULTAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno 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

5.1 Tema 19 : Recuperación de datos

- 5.1.1 : Introducción al lenguaje de consultas SQL.
- 5.1.2 : Uso de la sentencia SELECT, FROM, WHERE, ORDER BY
- 5.1.3 : Manipulación de consultas condicionales. Uso de condicionales: operadores lógicos AND, OR. Operadores de comparación >, <, =, <>, <=, >=. Operador para manejo de cadenas LIKE. Otros operadores: IN, BETWEEN.
- 5.1.4 : Funciones para el manejo de fechas: DAY(), MONTH(), YEAR(), DATEPART().

ACTIVIDADES PROPUESTAS

- Los alumnos implementan sentencias de consultas con la función SELECT.
- Los alumnos implementan funciones de fecha integrándolas a las consultas.

5.1. Recuperación de datos

5.1.1. 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.

5.1.2. Uso de la sentencia SELECT, FROM, WHERE, ORDER BY

5.1.2.1 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]]
```

5.1.2.2 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 permita mostrar los todos los registros de la tabla Cliente.

```
SELECT *
    FROM TB_CLIENTE
```

- Script que permita 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
```

5.1.2.3 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	XXXXXXXXXXXX	XXXXXXX	99/99/9999	9
XXXX	XXXXXXXXXXXX	XXXXXXX	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_IVG	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_IVG, F.EST_FAC, F.COD_CLI
FROM TB_FACTURA F
```

5.1.2.4 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	XXXXXXXXXXXX	XXXXXXX	99/99/9999	9
XXXX	XXXXXXXXXXXX	XXXXXXX	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	XXXXXXXX XXXXXXXXXXXX	99/99/9999
XXXX	XXXXXXXX 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 74: Listado de vendedores
Fuente.- Tomado desde SQL Server 2014

5.1.2.5 Consultas distinguida

Este tipo de consulta permite 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 fecha 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
```

5.1.2.6 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 permita 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 75: Listado de clientes
Fuente.- Tomado desde SQL Server 2014

- Script que permita 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 76: Listado de vendedores
Fuente.- Tomado desde SQL Server 2014

5.1.3. Manipulación de consultas condicionales. Uso de condicionales: operadores lógicos AND, OR. Operadores de comparación >, <, =, <>, <=, >=. Operador para manejo de cadenas LIKE. Otros operadores: IN, BETWEEN.

5.1.3.1 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 permita 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 77: Listado de detalle de la factura 100
Fuente.- Tomado desde SQL Server 2014

- Script que permita 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. Collagaté 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 78: Listado de clientes con tipo 2
Fuente.- Tomado desde SQL Server 2014

- Script que permita 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 79: Listado de productos con descripción Papel periódico
Fuente.- Tomado desde SQL Server 2014

5.1.3.2 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	MII	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 Plastico	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 80: 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 81: Listado de facturas con numero de factura superior a 116
Fuente.- Tomado desde SQL Server 2014

5.1.3.3 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

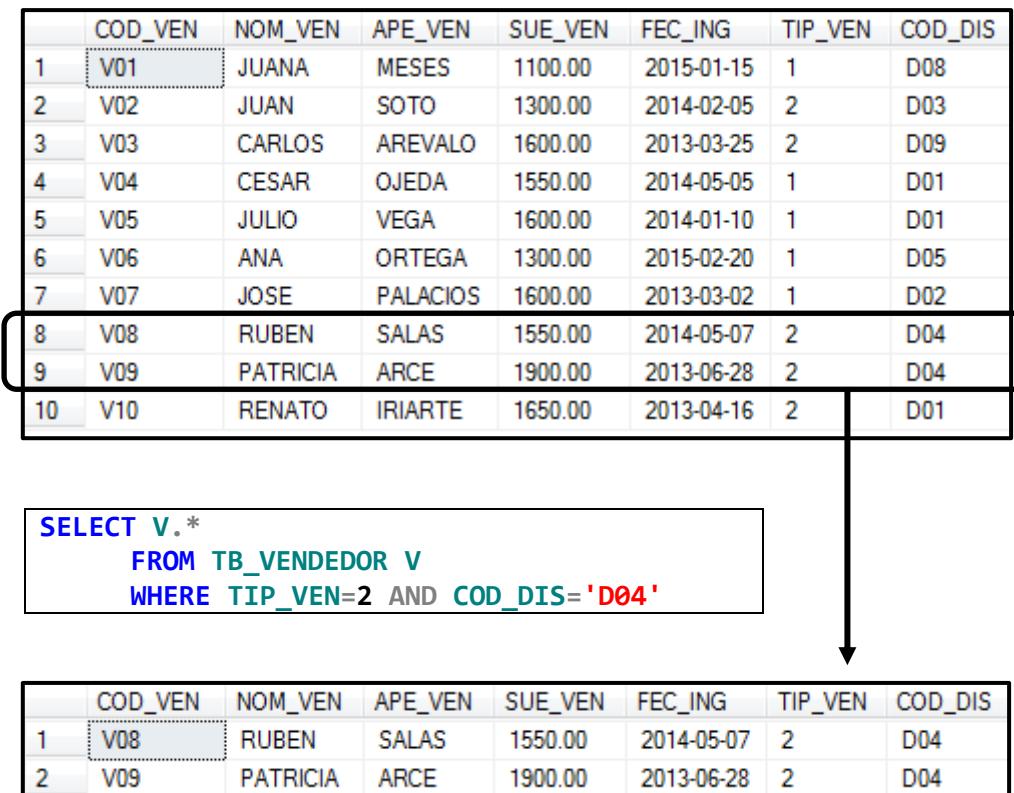
Expresión1	Expresión2	Resultado
V	V	V
V	F	F
F	V	F
F	F	V

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 **distrito cuyo código es D04**.



**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	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

	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 82: 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 83: 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) en el 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 84: 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 85: 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 86: 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 87: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

- Script que permita **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	Serviensa	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 88: 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 89: Listado de productos
Fuente.- Tomado desde SQL Server 2014

- Script que permita **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 90: Listado de órdenes de compra del año 1999 y estado 1
Fuente.- Tomado desde SQL Server 2014

5.1.3.4 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 comienzan 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 Marsano 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 91: 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 Marsano 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 92: Listado de clientes
Fuente.- Tomado desde SQL Server 2014

5.1.3.5 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 clausula 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 93: 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_IVG
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

Figura 94: 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 95: Listado de productos con inicial A y F en la descripción

Fuente.- Tomado desde SQL Server 2014

5.1.3.6 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 96: 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 97: Listado de vendedores del distrito D08 y D04
Fuente.- Tomado desde SQL Server 2014

5.1.4. Funciones para el manejo de fechas: DAY(), MONTH(), YEAR(), DATEPART()

5.1.4.1 Función Day

Devuelve un valor numérico entero que representa el dia según el mes de una determinada fecha. Veamos algunas consultas:

- Script que permita mostrar el dia correspondiente a la fecha actual.

```
SELECT DAY(GETDATE()) AS DIA
```

DIA	
1	13

Figura 98: 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 dia correspondiente a una determinada fecha obtenida a partir de una variable local.

```
DECLARE @FECHA DATE = '2016/02/05'
SELECT DAY(@FECHA) AS DIA
```

DIA	
1	5

Figura 99: 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 permita 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 100: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

5.1.4.2 Función Month

Devuelve un valor numérico entero que representa al mes de una determinada fecha. Veamos algunas consultas:

- Script que permita mostrar el mes correspondiente a la fecha actual.

```
SELECT MONTH(GETDATE()) AS MES
```

MES	
1	
	7

Figura 101: 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 = '2016/02/05'
SELECT MONTH(@FECHA) AS MES
GO
```

MES	
1	
	2

Figura 102: 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 103: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

5.1.4.3 Función Year

Devuelve un valor numérico entero que representa al año de una determinada fecha.
Veamos algunas consultas:

- Script que permita mostrar el año actual.

```
SELECT YEAR(GETDATE()) AS AÑO
```

AÑO
2015

Figura 104: Mostrando el año actual
Fuente.- Tomado desde SQL Server 2014

- Script que permita mostrar el año correspondiente a una determinada fecha obtenida a partir de una variable local.

```
DECLARE @FECHA DATE = '2016/02/05'
SELECT YEAR(@FECHA) AS AÑO
GO
```

AÑO
2016

Figura 105: Mostrando el año 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.],
       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	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

Figura 106: 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 107: Listado de facturas
Fuente.- Tomado desde SQL Server 2014

5.1.4.4 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 108: 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 109: Listado de fechas desde la tabla factura
Fuente.- Tomado desde SQL Server 2014

5.1.4.5 Función DateDiff

Devuelve un valor numérico entero que representa el recuento entre dos fechas especificadas. Su formato es:

DATEDIFF (ARGUMENTO , FECHAINICIO , FECHAFINAL)

- Script que permita 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 110: 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 111: Listado de fechas desde las órdenes de compra
Fuente.- Tomado desde SQL Server 2014

Resumen

1. SQL es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft SQL.
2. La sentencia SELECT permite componer una consulta que puede ser interpretada por el servidor de base de datos; el cual emitira un resultado expresado en registros.
3. 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.
4. Las consultas condicionadas soportan operadores lógicos como AND, OR, XOR, IS y NOT.
5. 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.



INTRODUCCIÓN A LA PROGRAMACIÓN TRANSACT SQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno implementa instrucciones SQL y de programación mediante procedimientos almacenados.

TEMARIO

6.1 Tema 20 : Introducción a la programación en SQL Server 2014

- 6.1.1 : Declaración de variables locales
- 6.1.2 Procedimientos almacenados con una tabla
- 6.1.3 Aplicación usando procedimientos almacenados con uno y dos parámetros

ACTIVIDADES PROPUESTAS

- Los alumnos entienden la importancia de la implementación de procedimientos almacenados en una base de datos.
- Los alumnos implementan procedimientos almacenados usando uno o más parámetros de entrada.
- Los alumnos integran las consultas dentro de los procedimientos almacenados.

6.1 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.

6.1.1 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, reciben 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 numero 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 mas 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 112: 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 113: 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 114: 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	XXXXXXXXXXX

```
DECLARE @COD CHAR(3)='V01', @CDDIS CHAR(3), @DIST VARCHAR(30)

--Obteniendo el código del distrito del vendedor
SELECT @CDDIS = 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 115: Listado del vendedor con código V01
Fuente.- Tomado desde SQL Server 2014

6.1.2 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**.

6.1.2.1 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  
AS  
    SENTENCIAS  
GO
```

O también podríamos usar el siguiente formato:

```
CREATE PROC NOMBRE  
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 ') IS NOT NULL  
    DROP PROC NOMBRE  
GO  
CREATE PROCEDURE NOMBRE  
AS  
    SENTENCIAS  
GO
```

Finalmente, veremos la sentencia que permite ejecutar los procedimientos almacenados:

```
EXECUTE NOMBRE
```

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'
```

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 116: Sentencias de un procedimiento almacenado
Fuente.- Tomado desde SQL Server 2014

6.1.2.2 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
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.

Veamos la modificación del procedimiento almacenado SP_LISTAPROVEEDOR en la cual se listaran 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
```

6.1.2.3 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
```

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
```

6.1.3. Aplicación de los procedimientos almacenados con uno y dos parámetros

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 varia 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'
```

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 117: Listado de proveedores cuyo nombre de su representante inicie con “V”
Fuente.- Tomado desde SQL Server 2014

6.1.3.1 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
```

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 118: Listado de vendedores del distrito D06
Fuente.- Tomado desde SQL Server 2014

6.1.3.2 Estructura condicional múltiple CASE

Evaluá 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 COLUMNA
    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
```

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_LISTA CLIENTES') IS NOT NULL
    DROP PROC SP_LISTA CLIENTES
GO

--Creando el procedimiento almacenado
CREATE PROC SP_LISTA CLIENTES
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_LISTA CLIENTES
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 119: Listado de clientes y su tipo
Fuente.- Tomado desde SQL Server 2014

6.1.3.3 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
```

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
```



Figura 120: Listado de números pares e impares usando While
Fuente.- Tomado desde SQL Server 2014

Actividad

Usando la base de datos **COMERCIO** implementaremos los siguientes procedimientos almacenados:

Procedimientos almacenados básicos

1. 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 121: Listando la fecha y hora actual
Fuente.- Tomado desde SQL Server 2014

2. 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	MII
2	P002	Papel Bond Oficio	28.00	50	1500	MII
3	P003	Papel Bulky	10.00	498	1000	MII
4	P004	Papel Periódico	7.20	4285	1000	MII
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 122: Listado de productos
Fuente.- Tomado desde SQL Server 2014

3. 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_LISTA CLIENTES') IS NOT NULL
    DROP PROC SP_LISTA CLIENTES
GO

CREATE PROC SP_LISTA CLIENTES
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_LISTA CLIENTES
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	Serviems	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 123: Listado de clientes
Fuente.- Tomado desde SQL Server 2014

4. 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 124: Listado de vendedores
Fuente.- Tomado desde SQL Server 2014

5. 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 125: Listado de productos de unidad de medida “Doc”
Fuente.- Tomado desde SQL Server 2014

6. 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. 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 126: Listado de clientes del distrito con código D05
Fuente.- Tomado desde SQL Server 2014

7. 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 127: Listado de órdenes de compra registradas en el año 1999
 Fuente.- Tomado desde SQL Server 2014

8. 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 128: Listando el detalle de la factura 103 y código de producto P002
 Fuente.- Tomado desde SQL Server 2014

9. 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_LISTAFACTURAS')IS NOT NULL
    DROP PROC SP_LISTAFACTURAS
GO

CREATE PROC SP_LISTAFACTURAS
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_LISTAFACTURAS
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 129: Listado de facturas registradas en el primer semestre del año 1998

Fuente.- Tomado desde SQL Server 2014

10. 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	M	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 130: 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

1. 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 131: Listado de productos de tipo Falso en la columna importado (producto nacional)
Fuente.- Tomado desde SQL Server 2014

2. 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_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

Figura 132: Listado de facturas registradas en el año 1998
Fuente.- Tomado desde SQL Server 2014

3. Procedimiento Almacenado que liste todos los datos de los clientes según el código del distrito.

```

IF OBJECT_ID('SP_LISTA CLIENTES') IS NOT NULL
    DROP PROC SP_LISTA CLIENTES
GO

CREATE PROC SP_LISTA CLIENTES (@DIS CHAR(3))
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.COD_DIS=@DIS
GO

--Probando el procedimiento
EXEC SP_LISTA CLIENTES '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 Faucet 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez

Figura 133: Listado de clientes del distrito cuyo código es D16
Fuente.- Tomado desde SQL Server 2014

4. Procedimiento Almacenado que liste todos los datos de los clientes según la letra inicial en su razón social.

```

IF OBJECT_ID('SP_LISTA CLIENTES') IS NOT NULL
    DROP PROC SP_LISTA CLIENTES
GO

CREATE PROC SP_LISTA CLIENTES (@INICIAL CHAR(1))
AS
    SELECT C.*
        FROM TB_CLIENTE C
        WHERE C.RAZ_SOC_CLI LIKE @INICIAL + '%'
GO

--Probando el procedimiento
EXEC SP_LISTA CLIENTES 'M'
GO

```

5. Procedimiento Almacenado que liste todos los datos de los vendedores cuyo sueldo sea inferior a un determinado monto.

```

IF OBJECT_ID('SP_LISTA VENDEDOR') IS NOT NULL
    DROP PROC SP_LISTA VENDEDOR
GO

CREATE PROC SP_LISTA VENDEDOR (@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 134: Listado de vendedores con sueldo inferior a S/. 1250
Fuente.- Tomado desde SQL Server 2014

6. 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 135: Listado de órdenes de compra del proveedor PR01
Fuente.- Tomado desde SQL Server 2014

7. 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 136: Listado de productos cuyo rango de precio es 15 y 20
Fuente.- Tomado desde SQL Server 2014

8. 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 137: Listando los clientes del distrito con código D14 y tipo de cliente 1
Fuente.- Tomado desde SQL Server 2014

9. 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 138: 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_LISTAFACUTURAS') IS NOT NULL
    DROP PROC SP_LISTAFACUTURAS
GO

CREATE PROC SP_LISTAFACUTURAS (@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_LISTAFACUTURAS 2015
GO

```

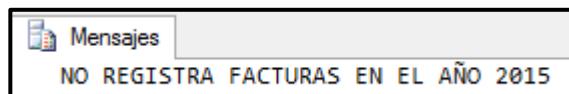


Figura 139: Mensaje desde la validación del año
Fuente.- Tomado desde SQL Server 2014

```
EXEC SP_LISTAFACUTURAS 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 140: Listado de facturas registradas en el año 1998
Fuente.- Tomado desde SQL Server 2014

2. 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 141: Listado de registro de orden de compra del proveedor PR07
Fuente.- Tomado desde SQL Server 2014

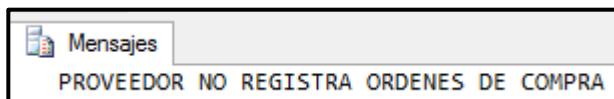


Figura 142: Mensaje desde la validación del proveedor
Fuente.- Tomado desde SQL Server 2014

3. 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 143: Mensaje de validación obtenidos 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 144: Listado de productos con precios entre S/. 20 y S/. 30
Fuente.- Tomado desde SQL Server 2014

4. 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 “CODIGO 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 'CODIGO 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
```

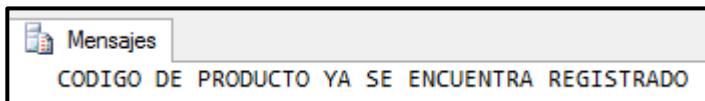


Figura 145: Mensaje desde la validación del producto
Fuente.- Tomado desde SQL Server 2014

```
EXEC SP_REGISTRAPRODUCTO 'P022', 'Cartulina Duplex',
                           0.5, 1000, 100, 'Doc', 2, 'FALSO'
GO
```

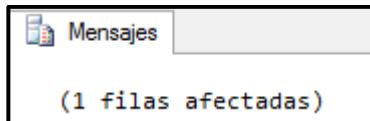


Figura 146: Mensaje de inserción correcta del producto
Fuente.- Tomado desde SQL Server 2014

- 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 147: Mensaje de inserción correcta de vendedor
Fuente.- Tomado desde SQL Server 2014

```
EXEC SP_REGISTRAVENDEDOR 'V01','MARIA','ZAMORA',
                           1950,'2015/02/05',1,'D07'
```

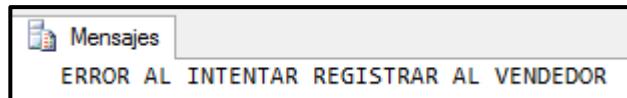


Figura 148: Mensaje desde la validación del vendedor no valido
Fuente.- Tomado desde SQL Server 2014

6. 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

```

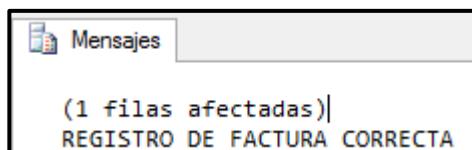


Figura 149: Mensaje de registro correcto
Fuente.- Tomado desde SQL Server 2014

```
EXEC SP_REGISTRAFACTURA '2016/06/05','C003','2016/07/03',1,'V31'
```



Figura 150: 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'
```

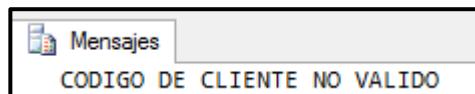


Figura 151: Mensaje desde la validación de cliente no valido
Fuente.- Tomado desde SQL Server 2014

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 152: 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 DE 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	B
8	V08	RUBEN	SALAS	1450,00	2014-05-07	2	D04	B
9	V09	PATRICIA	ARCE	1800,00	2013-06-28	2	D04	A
10	V10	RENATO	IRIARTE	1550,00	2013-04-16	2	D01	B
11	V11	MARIA	ZAMORA	1950,00	2015-02-05	1	D07	A
12	V12	MARIA	ZAMORA	1950,00	2015-02-05	1	D07	A

Figura 153: Listado de vendedores mostrando su categoría
Fuente.- Tomado desde SQL Server 2014

3. 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'
        WHEN 4 THEN 'A'
    END) AS CORREO_Corporativo
    FROM TB_VENDEDOR V
GO

```

```

        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	jsotoa@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	ARCE	1800.00	2013-06-28	2	D04	parceb@cibertec.edu.pe
10	V10	RENATO	IRIARTE	1550.00	2013-04-16	2	D01	ririaa@cibertec.edu.pe

Figura 154: Listado de vendedores con sus correos autogenerados

Fuente.- Tomado desde SQL Server 2014

Manejo de la sentencia While

1. Script que permita 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 155: Listando las N últimas facturas controladas por variables locales
 Fuente.- Tomado desde SQL Server 2014

2. 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 156: Listado de distritos entre un determinado rango de registros
 Fuente.- Tomado desde SQL Server 2014

Resumen

1. 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.
2. Las instrucciones de Transact/SQL pueden agruparse en procesos o lotes, permanecer en la base de datos, ejecutarse repetidamente en forma de procedimientos almacenados.
3. 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)



CONSULTAS MULTITABLAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno recupera información de una base de datos utilizando SQL, obtiene registros originados por la selección de uno o varias columnas procedentes de dos o más tablas.

TEMARIO

7.1. Tema 21: Uniones internas y externas (INNER JOIN)

- 7.1.1. Combinaciones internas con Inner Join
- 7.1.2. Combinaciones externas con Left, Right, Full, Cross Join.

ACTIVIDADES PROPUESTAS

- Los alumnos implementan consultas usando dos o más tablas e integradas a los procedimientos almacenados.
- Los alumnos implementan consultas que permiten hacer uso de las uniones internas y externas integradas a un procedimiento almacenado con parámetros.

7.1. Uniones internas y externas (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.

7.1.1. 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.

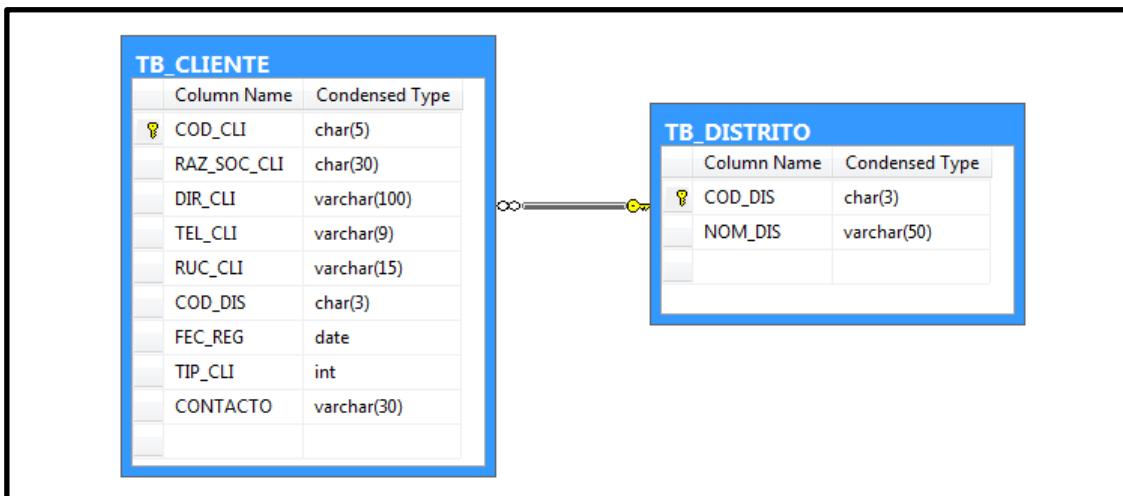
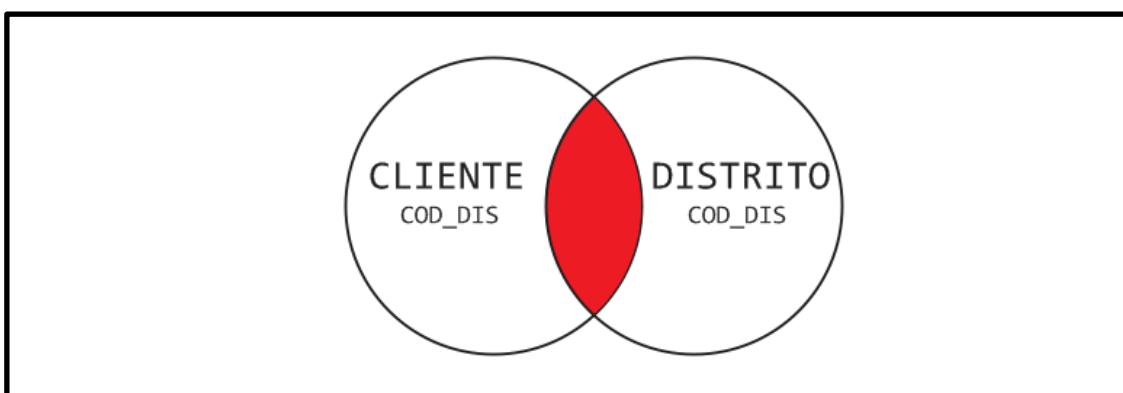


Figura 157: Diagrama de la tabla cliente y distrito
Fuente.- Tomado desde SQL Server 2014

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 158: 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	XXXXXXXXXXXX	XXXXXXXXXXXX	99999999	XXXXXXXXXXXX
XXXX	XXXXXXXXXXXX	XXXXXXXXXXXX	99999999	XXXXXXXXXXXX

```
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	4342318	San Miguel
2	C002	Orbi	Av. Emilio Cavenecia 225	4406335	La Molina
3	C003	Serviems	Jr. Collagate 522	75012403	San Miguel
4	C004	Issa	Calle Los Aviadores 263	3725910	Surco
5	C005	Mass	Av. Tomas Marsano 880	4446177	Surquillo

Figura 159: 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 160: 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**.

7.1.2. Combinaciones externas con Left, Right, Full, Cross Join.

Las combinaciones internas solo devuelven filas cuando hay una fila de ambas tablas, como mínimo, que coincide con la condición de la combinación. Las combinaciones internas eliminan las filas que no coinciden con alguna fila de la otra tabla. Sin embargo, las **combinaciones externas** devuelven todas las filas de una de las tablas o vistas mencionadas en la cláusula FROM, como mínimo, siempre que tales filas cumplan con alguna de las condiciones de búsqueda de WHERE o HAVING. Entonces podemos mencionar que:

- Todas las filas se recuperarán de la tabla izquierda a la que se haya hecho referencia con una combinación externa izquierda.
- Todas las filas se recuperarán de la tabla derecha a la que se haya hecho referencia con una combinación externa derecha.
- En una combinación externa completa, se devuelven todas las filas de ambas tablas.

SQL Server 2014 utiliza las siguientes palabras clave para las combinaciones externas especificadas en una cláusula FROM:

- LEFT OUTER JOIN o LEFT JOIN
- RIGHT OUTER JOIN o RIGHT JOIN
- FULL OUTER JOIN o FULL JOIN

7.1.2.1 Manejo de la combinación LEFT JOIN

El operador de combinación externa izquierda, LEFT JOIN, indica que todas las filas de la primera tabla se deben incluir en los resultados, con independencia si hay datos coincidentes en la segunda tabla. Es decir, la tabla de la izquierda fuerza a la derecha a mostrar todos los registros a pesar de no encontrar coincidencias y es justamente aquí que la tabla de la derecha mostrará el término NULL en los valores no coincidentes.

Ahora, mostraremos la combinación externa entre las tablas Distrito y Cliente:

```
SELECT *
  FROM TB_DISTRITO D
LEFT JOIN TB_CLIENTE C ON C.COD_DIS=D.COD_DIS
GO
```

COD_DIS	NOM_DIS	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO
1	D01	Surco	C004	Issa	Calle Los Aviadores 263	3725910	46720159	D01	1992-09-12	1 Luis Apumayta
2	D02	Jesús María	C020	Cramer	Jr. Mariscal Miller 1131	4719061	46031783	D02	1996-08-11	1 Christopher Ramos
3	D03	San Isidro	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
4	D04	La Molina	C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	D04	1990-02-01	2 Alfonso Beltran
5	D05	San Miguel	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1 Alicia Barreto
6	D05	San Miguel	C003	Serviemsa	Jr. Collagte 522	75012403	NULL	D05	1995-06-03	2 Christian Laguna
7	D05	San Miguel	C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1 Judith Aste
8	D05	San Miguel	C010	Landu	Av. Nicolas de Ayllon 1453	3267840	30405261	D05	1989-11-08	2 Raquel Espinoza
9	D05	San Miguel	C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1 Karina Vega
10	D06	Miraflores	C014	Kadia	Av. Santa Cruz 1332 Of.201	4412418	22202915	D06	1995-05-04	1 Miguel Arce
11	D07	Barranco	C017	Payet	Calle Juan Fanning 327	4779834	70594032	D07	1993-05-12	1 Paola Uribe
12	D08	Chorrillos	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 161: Listado de clientes y distritos combinados con Left Join

Fuente.- Tomado desde SQL Server 2014

Si observamos el resultado, podemos definir que empezando por los primeros registros el distrito de San Isidro, Chorrillos, Magdalena, Rímac y Pueblo Libre no cuentan con registro de clientes.

Así mismo, si analizamos el concepto aplicado al LEFT JOIN notamos que la tabla de la izquierda (Distrito) fuerza a la tabla de la derecha (Cliente) a mostrar información así no encuentre coincidencias como es el caso de los distritos mencionados anteriormente.

Finalmente, si necesitamos mostrar solo los distritos que aun no registran cliente, el script sería el siguiente:

```
SELECT D.*
  FROM TB_DISTRITO D
LEFT JOIN TB_CLIENTE C ON C.COD_DIS=D.COD_DIS
WHERE C.COD_CLI IS NULL
GO
```

	COD_DIS	NOM_DIS
1	D03	San Isidro
2	D08	Chorrillos
3	D12	Magdalena
4	D13	Rímac
5	D15	Pueblo Libre
6	D18	San Martín de Porres
7	D19	Santa Anita
8	D21	Independencia
9	D25	El Agustino
10	D27	Ate - Vitarte
11	D28	San Juan de Miraflores
12	D29	Carmen de la Legua
13	D30	Comas
14	D31	Villa María del Triunfo
15	D32	Villa el Salvador
16	D33	La Perla
17	D34	Ventanilla
18	D35	Puente Piedra

Figura 162: Listado de distritos que no registran clientes con Left Join
Fuente.- Tomado desde SQL Server 2014

Como observamos en el script en la selección de las columnas por mostrar hacemos referencia solo a la tabla Distrito (D.*) y agregamos la condición que muestre los registros solo si el código del cliente es nulo, pero si observamos la imagen XX notaremos que hay varias columnas con valores nulos; por lo tanto el script podría variar de la siguiente manera:

```
SELECT D.*
  FROM TB_DISTRITO D
LEFT JOIN TB_CLIENTE C ON C.COD_DIS=D.COD_DIS
WHERE C.COD_DIS IS NULL
GO
```

```
SELECT D.*
FROM TB_DISTRITO D
LEFT JOIN TB_CLIENTE C ON C.COD_DIS=D.COD_DIS
WHERE C.TEL_CLI IS NULL
GO
```

```
SELECT D.*
FROM TB_DISTRITO D
LEFT JOIN TB_CLIENTE C ON C.COD_DIS=D.COD_DIS
WHERE C.FEC_REG IS NULL
GO
```

7.1.2.2 Manejo de la combinación RIGHT JOIN

El operador de combinación externa derecha, RIGHT JOIN, indica que todas las filas de la segunda tabla se deben incluir en los resultados, con independencia si hay datos coincidentes en la primera tabla. Es decir, la tabla de la derecha fuerza a la izquierda a mostrar todos los registros a pesar de no encontrar coincidencias y es justamente aquí que la tabla de la izquierda mostrará el término NULL en los valores no coincidentes.

Ahora, mostraremos la combinación externa entre las tablas Distrito y Cliente:

```
SELECT *
FROM TB_CLIENTE C
RIGHT 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 C004	Issa	Calle Los Aviadores 263	3725910	46720159	D01	1992-09-12	1	Luis Apumaya	D01	Surco
2 C020	Cramer	Jr. Mariscal Miller 1131	4719061	46031783	D02	1996-08-11	1	Christopher Ramos	D02	Jesús María
3 NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D03	San Isidro
4 C002	Orbi	Av. Emilio Cavenecia 225	4406335	57031642	D04	1990-02-01	2	Alfonso Beltran	D04	La Molina
5 C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D05	San Miguel
6 C003	Serviensa	Jr. Collagate 522	75012403	NULL	D05	1995-06-03	2	Christian Laguna	D05	San Miguel
7 C006	Berker	Av. Los Proceres 521	3810322	54890124	D05	1989-07-05	1	Judith Aste	D05	San Miguel
8 C010	Landu	Av. Nicolás de Ayllón 1453	3267840	30405261	D05	1989-11-08	2	Raquel Espinoza	D05	San Miguel
9 C012	Sucerte	Jr. Grito de Huaura 114	4206434	62014503	D05	1990-10-05	1	Karina Vega	D05	San Miguel
10 C014	Kadia	Av. Santa Cruz 1332 Of.201	4412418	22202915	D06	1995-05-04	1	Miguel Arce	D06	Miraflores
11 C017	Payet	Calle Juan Fanning 327	4779834	70594032	D07	1993-05-12	1	Paola Uribe	D07	Baranco
12 NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D08	Chorrillos
13 C008	Intech	Av. San Luis 2619 5to P	2249493	34021824	D09	1997-01-07	2	Carlos Villanueva	D09	San Borja
14 C016	Cardeli	Jr. Bartolome Herrera 451	2658853	26403158	D10	1991-12-03	2	Giancarlo Bonifaz	D10	Lince
15 C009	Prominent	Jr. Iquique 132	43233519	NULL	D11	1993-06-11	1	Jorge Valdivia	D11	Breña
16 NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D12	Magdalena
17 NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D13	Rímac
18 C005	Mass	Av. Tomás Marsano 880	4446177	83175942	D14	1992-10-01	1	Katia Armejo	D14	Surquillo
19 NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D15	Pueblo Libre

Figura 163: Listado de clientes y distritos combinados con Right Join
Fuente.- Tomado desde SQL Server 2014

Si observamos el resultado, podemos definir que empezando por los primeros registros del lado derecho el distrito de San Isidro, Chorrillos, Magdalena, Rímac y Pueblo Libre no cuentan con registro de clientes.

Así mismo, si analizamos el concepto aplicado al RIGHT JOIN notamos que la tabla de la derecha (Distrito) fuerza a la tabla de la izquierda (Cliente) a mostrar información así no encuentre coincidencias; como es el caso de los distritos mencionados anteriormente.

Finalmente, si necesitamos mostrar solo los distritos que aun no registran cliente usando Right Join, el script seria el siguiente:

```
SELECT D.*
FROM TB_CLIENTE C
RIGHT JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS
WHERE C.COD_CLI IS NULL
GO
```

	COD_DIS	NOM_DIS
1	D03	San Isidro
2	D08	Chorillos
3	D12	Magdalena
4	D13	Rimac
5	D15	Pueblo Libre
6	D18	San Martin de Porres
7	D19	Santa Anita
8	D21	Independencia
9	D25	El Agustino
10	D27	Ate - Vitarte
11	D28	San Juan de Miraflores
12	D29	Carmen de la Legua
13	D30	Comas
14	D31	Villa María del Triunfo
15	D32	Villa el Salvador
16	D33	La Perla
17	D34	Ventanilla
18	D35	Puente Piedra

Figura 164: Listado de distritos que aun no registra clientes con Right Join
Fuente.- Tomado desde SQL Server 2014

7.1.2.3 Manejo de la combinación FULL JOIN

Para retener la información que no coincide al incluir las filas no coincidentes en los resultados de una combinación, utilice una combinación externa completa. SQL Server 2014 proporciona el operador de combinación externa completa, **FULL JOIN**, que incluye todas las filas de ambas tablas, con independencia de que la otra tabla tenga o no un valor coincidente. En forma práctica, podemos decir que FULL JOIN es una combinación de LEFT JOIN y RIGHT JOIN.

Ahora, mostraremos la combinación externa entre las tablas Distrito y Cliente:

```
SELECT *
FROM TB_CLIENTE C
FULL 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
15	C015	Meba	Av. Elmer Faucett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez	D16	Bellavista
16	C016	Cardeli	Jr. Bartolome Herrera 451	2658853	26403158	D10	1991-12-03	2	Giancarlo Bonifaz	D10	Lince
17	C017	Payet	Calle Juan Fanning 327	4779834	70594032	D07	1993-05-12	1	Paola Uribe	D07	Barranco
18	C018	Dasin	Av. Saenz Peña 338 - B	4657574	35016752	D17	1991-12-03	1	Angela Barreto	D17	Callao
19	C019	Corefo	Av. Canada 3894 - 3898	4377499	57201691	D24	1998-01-03	2	Rosalyn Cortez	D24	San Luis
20	C020	Cramer	Jr. Mariscal Miller 1131	4719061	46031783	D02	1996-08-11	1	Christopher Ramos	D02	Jesús María
21	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D03	San Isidro
22	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D08	Chorrillos
23	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D12	Magdalena
24	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D13	Rimac
25	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D15	Pueblo Libre
26	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D18	San Martin de Porres
27	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D19	Santa Anita
28	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D21	Independencia
29	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D25	El Agustino
30	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	D27	Ate - Vitarte

Figura 165: Listado de clientes y distritos combinado con Full Join
Fuente.- Tomado desde SQL Server 2014

En Full Join no se considera importante la posición de las tablas, pues, al ser una combinación Full, podemos hacer referencia a cualquier columna de la combinación, así mismo debemos mencionar que los valores nulos se presentarán cuando los valores en ambas tablas no coincidan como lo mencionado en el Left y Right Join; con la diferencia que estos se presentarán al final de los registros.

```
SELECT D.*  

  FROM TB_CLIENTE C  

    FULL JOIN TB_DISTRITO D ON C.COD_DIS=D.COD_DIS  

   WHERE C.COD_CLI IS NULL  

      GO
```

	COD_DIS	NOM_DIS
1	D03	San Isidro
2	D08	Chorrillos
3	D12	Magdalena
4	D13	Rimac
5	D15	Pueblo Libre
6	D18	San Martin de Porres
7	D19	Santa Anita
8	D21	Independencia
9	D25	El Agustino
10	D27	Ate - Vitarte
11	D28	San Juan de Miraflores
12	D29	Carmen de la Legua
13	D30	Comas
14	D31	Villa María del Triunfo
15	D32	Villa el Salvador
16	D33	La Perla
17	D34	Ventanilla
18	D35	Puente Piedra

Figura 166: Listado de distritos
Fuente.- Tomado desde SQL Server 2014

7.1.2.4 Manejo de la combinación CROSS JOIN

Se emplea Cross Join cuando se quieran combinar todos los registros de una tabla con cada registro de otra tabla.

Se puede usar un cross join si se quiere una manera rápida de enviar una tabla con información. Este tipo de joins también se les conoce como producto cartesiano. Ahora, mostraremos la combinación externa entre las tablas Distrito y Cliente:

```
SELECT *
  FROM TB_CLIENTE C
CROSS JOIN TB_DISTRITO D
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	D01	Surco
2	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D02	Jesús María
3	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D03	San Isidro
4	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D04	La Molina
5	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D05	San Miguel
6	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D06	Miraflores
7	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D07	Barranco
8	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D08	Chomillos
9	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D09	San Borja
10	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D10	Lince
11	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D11	Breña
12	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D12	Magdalena
13	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D13	Rimac
14	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D14	Surquillo
15	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	D15	Pueblo Libre

Figura 167: Listado de registro de clientes
Fuente.- Tomado desde SQL Server 2014

Actividad

Usando la base de datos **COMERCIO** y por medio de procedimientos almacenados, implemente las siguientes consultas:

Inner Join simple

1. 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_IGV 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	Serviems	1998-03-11	2	CESAR OJEDA	0.19
4	103	1998-06-09	Cardeli	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 168: Listado de registro de facturas
Fuente.- Tomado desde SQL Server 2014

2. 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 169: Listado de órdenes de compra
Fuente.- Tomado desde SQL Server 2014

Inner Join condicionado

3. 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 Plastico	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 Plastico A-4	50.00	3080	1100	Cie	3	FALSO

Figura 170: Listado de productos de tipo útiles de oficina
Fuente.- Tomado desde SQL Server 2014

4. Script que permita 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 171: Listando las órdenes de compra de un determinado proveedor
Fuente.- Tomado desde SQL Server 2014

5. 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	Finsteth	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	Beker	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 172: Listado de clientes del distrito de San Miguel
Fuente.- Tomado desde SQL Server 2014

6. 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 173: Listado de facturas del año 1998
Fuente.- Tomado desde SQL Server 2014

7. 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 174: Listado de facturas de un determinado vendedor
Fuente.- Tomado desde SQL Server 2014

8. 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 175: Listado de facturas de un determinado cliente y vendedor
Fuente.- Tomado desde SQL Server 2014

Left Join

9. Script que permita listar los datos de los productos que aun no registran venta.

```

IF OBJECT_ID('SP_LISTADO9')IS NOT NULL
    DROP PROC SP_LISTADO9
GO

CREATE PROC SP_LISTADO9
AS
    SELECT P./*
        FROM TB_PRODUCTO P
        LEFT JOIN TB_DETALLE_FACTURA DF ON P.COD_PRO=DF.COD_PRO
        WHERE DF.NUM_FAC IS NULL
GO

--PRUEBA
EXEC SP_LISTADO9
GO

```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
2	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
3	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
4	P021	Protector de Pantalla	50.00	20	5	Uni	1	FALSO

Figura 176: Listado de productos que aun no se venden
Fuente.- Tomado desde SQL Server 2014

10. Script que permita listar los datos de los proveedores que aun no registran órdenes de compra.

```

IF OBJECT_ID('SP_LISTADO10')IS NOT NULL
    DROP PROC SP_LISTADO10
GO

CREATE PROC SP_LISTADO10
AS
    SELECT P./*
        FROM TB_PROVEEDOR P
        LEFT JOIN TB_ORDEN_COMPRA O ON O.COD_PRV=P.COD_PRV
        WHERE O.NUM_OCO IS NULL
GO

--PRUEBA
EXEC SP_LISTADO10
GO

```

	COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
1	PR02	Atlas	Av. Lima 471	5380926	D13	Cesar Torres
2	PR13	Limmsa	Prolg. Huaylas 670	2546995	D08	Laura Ortega
3	PR15	Crosland	Av. Argentina 3206 - 3250	4515272	D17	Juan Ramirez
4	PR17	Reawse	Av. Santa Rosa 480	NULL	D19	Maria Perez

Figura 177: Listado de proveedores que no han registrado órdenes de compra

Fuente.- Tomado desde SQL Server 2014

11. Script que permita listar las órdenes de compra que no registran detalle, además de condicionarlo por un determinado año.

```

IF OBJECT_ID('SP_LISTADO11')IS NOT NULL
    DROP PROC SP_LISTADO11
GO

CREATE PROC SP_LISTADO11 (@AÑO INT)
AS
    SELECT OC.*
        FROM TB_ORDEN_COMPRA OC
        LEFT JOIN TB_DETALLE_COMPRA DC ON DC.NUM_OCO=OC.NUM_OCO
        WHERE DC.NUM_OCO IS NULL AND YEAR(OC.FEC_OCO)=@AÑO
GO

--PRUEBA
EXEC SP_LISTADO11 1998
GO

```

	NUM_OCO	FEC_OCO	COD_PRV	FEC_ATE	EST_OCO
1	OC006	1998-02-01	PR19	1998-02-01	1
2	OC007	1998-06-02	PR20	1998-05-04	3
3	OC015	1998-03-11	PR18	1998-10-12	1

Figura 178: Listando las órdenes de compra del año 1998

Fuente.- Tomado desde SQL Server 2014

Right Join

12. Script que permita listar los datos de los productos que aún no registran venta.

```

IF OBJECT_ID('SP_LISTADO12')IS NOT NULL
    DROP PROC SP_LISTADO12
GO

CREATE PROC SP_LISTADO12
AS
    SELECT P.*
        FROM TB_DETALLE_FACTURA D
        RIGHT JOIN TB_PRODUCTO P ON D.COD_PRO=P.COD_PRO
        WHERE D.NUM_FAC IS NULL
GO

--PRUEBA
EXEC SP_LISTADO12
GO

```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
2	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
3	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
4	P021	Protector de Pantalla	50.00	20	5	Uni	1	FALSO

Figura 179: Listando los productos que aun no se venden
Fuente.- Tomado desde SQL Server 2014

13. Script que permita listar los datos de los proveedores que aún no registran órdenes de compra.

```
IF OBJECT_ID('SP_LISTADO13')IS NOT NULL
    DROP PROC SP_LISTADO13
GO

CREATE PROC SP_LISTADO13
AS
    SELECT P./*
        FROM TB_ORDEN_COMPRA O
        RIGHT JOIN TB_PROVEEDOR P ON O.COD_PRV=P.COD_PRV
        WHERE O.NUM_OCO IS NULL
GO

--PRUEBA
EXEC SP_LISTADO13
GO
```

	COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
1	PR02	Atlas	Av. Lima 471	5380926	D13	Cesar Torres
2	PR13	Limmsa	Prolg. Huaylas 670	2546995	D08	Laura Ortega
3	PR15	Crosland	Av. Argentina 3206 - 3250	4515272	D17	Juan Ramirez
4	PR17	Reawse	Av. Santa Rosa 480	NULL	D19	Maria Perez

Figura 180: Listando los proveedores que no registran órdenes de compra
Fuente.- Tomado desde SQL Server 2014

Full Join

14. Script que permita listar los datos de los productos que aún no registran venta.

```
IF OBJECT_ID('SP_LISTADO14')IS NOT NULL
    DROP PROC SP_LISTADO14
GO

CREATE PROC SP_LISTADO14
AS
    SELECT P./*
        FROM TB_DETALLE_FACTURA D
        FULL JOIN TB_PRODUCTO P ON D.COD_PRO=P.COD_PRO
        WHERE D.NUM_FAC IS NULL
GO
```

```
--PRUEBA
EXEC SP_LISTADO14
GO
```

	COD_PRO	DES_PRO	PRE_PRO	STK_ACT	STK_MIN	UNI_MED	LIN_PRO	IMPORTADO
1	P010	Borrador Blanco	8.00	2000	400	Doc	3	FALSO
2	P014	Folder Manila A-4	20.00	150	150	Cie	3	VERDADERO
3	P018	Lapicero Azul	10.00	2010	1500	Doc	3	FALSO
4	P021	Protector de Pantalla	50.00	20	5	Uni	1	FALSO

Figura 181: Listado de productos que aun no se venden
Fuente.- Tomado desde SQL Server 2014

15. Script que permita listar los datos de los proveedores que aun no registran órdenes de compra.

```
IF OBJECT_ID('SP_LISTADO15')IS NOT NULL
    DROP PROC SP_LISTADO15
GO

CREATE PROC SP_LISTADO15
AS
    SELECT P./*
        FROM TB_ORDEN_COMPRA O
        FULL JOIN TB_PROVEEDOR P ON O.COD_PRV=P.COD_PRV
        WHERE O.NUM_OCO IS NULL
GO

--PRUEBA
EXEC SP_LISTADO15
GO
```

	COD_PRV	RAZ_SOC_PRV	DIR_PRV	TEL_PRV	COD_DIS	REP_VEN
1	PR02	Atlas	Av. Lima 471	5380926	D13	Cesar Torres
2	PR13	Limmsa	Prolg. Huaylas 670	2546995	D08	Laura Ortega
3	PR15	Crosland	Av. Argentina 3206 - 3250	4515272	D17	Juan Ramirez
4	PR17	Reawse	Av. Santa Rosa 480	NULL	D19	María Perez

Figura 182: Listado de proveedores que no registran órdenes de compra
Fuente.- Tomado desde SQL Server 2014

Cross Join

16. Script que permita mostrar la combinación de columnas entre las tablas Cliente y Factura.

```
IF OBJECT_ID('SP_LISTADO16')IS NOT NULL
    DROP PROC SP_LISTADO16
GO

CREATE PROC SP_LISTADO16
AS
    SELECT *
        FROM TB_CLIENTE
```

```
CROSS JOIN TB_FACTURA
GO

--PRUEBA
EXEC SP_LISTADO16
GO
```

	COD_CLI	RAZ_SOC_CLI	DIR_CLI	TEL_CLI	RUC_CLI	COD_DIS	FEC_REG	TIP_CLI	CONTACTO	NUM_FAC	FEC_FAC	COD_CLI	FEC_CAN	EST_FAC	COD_VEN	POR_IVA
1	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	100	1998-06-07	C001	1998-05-08	2	V01	0.19
2	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	101	1998-06-09	C019	1998-05-08	3	V02	0.19
3	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	102	1998-01-09	C003	1998-03-11	2	V04	0.19
4	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	103	1998-06-09	C016	1998-05-11	2	V07	0.19
5	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	104	1998-01-10	C015	1998-12-10	2	V08	0.19
6	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	105	1998-10-10	C009	1998-05-08	3	V05	0.19
7	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	106	1998-05-10	C019	1998-05-08	1	V09	0.19
8	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	107	1998-09-10	C012	1998-06-11	2	V10	0.19
9	C001	Finseth	Av. Los Viñedos 150	4342318	48632081	D05	1991-12-10	1	Alicia Barreto	108	1998-03-10	C008	1998-11-11	2	V09	0.19

Figura 183: Listado de clientes a partir de un Cross Join

Fuente..- Tomado desde SQL Server 2014

Resumen

6. En una consulta multitable, las tablas que contienen los datos son designadas en la cláusula FROM.
7. 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.
8. Las consultas multitable más habituales utilizan las relaciones padre / hijo creadas por las claves primarias y claves foráneas.
9. 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:

- http://www.aulaclic.es/sql/t_3_5.htm
Tutorial para el uso de UNIONES EXTERNAS
- http://www.aulaclic.es/sql/t_3_4.htm
Tutorial para el uso de inner join



SUBCONSULTAS, VISTAS Y AGRUPAMIENTO

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno crea y emplea vistas complejas de subconjunto, agrupadas y compuestas en una base de datos de un proceso de negocio real.

TEMARIO

8.1 Tema 22 : Subconsultas

8.1.1 : Subconsultas.

8.2 Tema 23 : Vistas

8.2.1 : Vistas multitabla.
8.2.2 Clasificación de las vistas.

8.3 Tema 24 : Agrupamiento de datos

8.3.1 Empleo de funciones agregadas
8.3.2 Empleo de GROUP BY, HAVING
8.3.3 Aplicación con procedimiento almacenado

ACTIVIDADES PROPUESTAS

- Los alumnos usan los resultados de una consulta como parte de otra en un determinado proceso.
- Los alumnos implementan vistas simples y complejas haciendo uso de varias tablas.
- Los alumnos integran las subconsultas y agrupamiento de datos en procedimientos almacenados.

8.1. Subconsultas

8.1.1. 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	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 184: Listado de clientes a partir de una subconsulta

Fuente..- Tomado desde SQL Server 2014

8.2. 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.

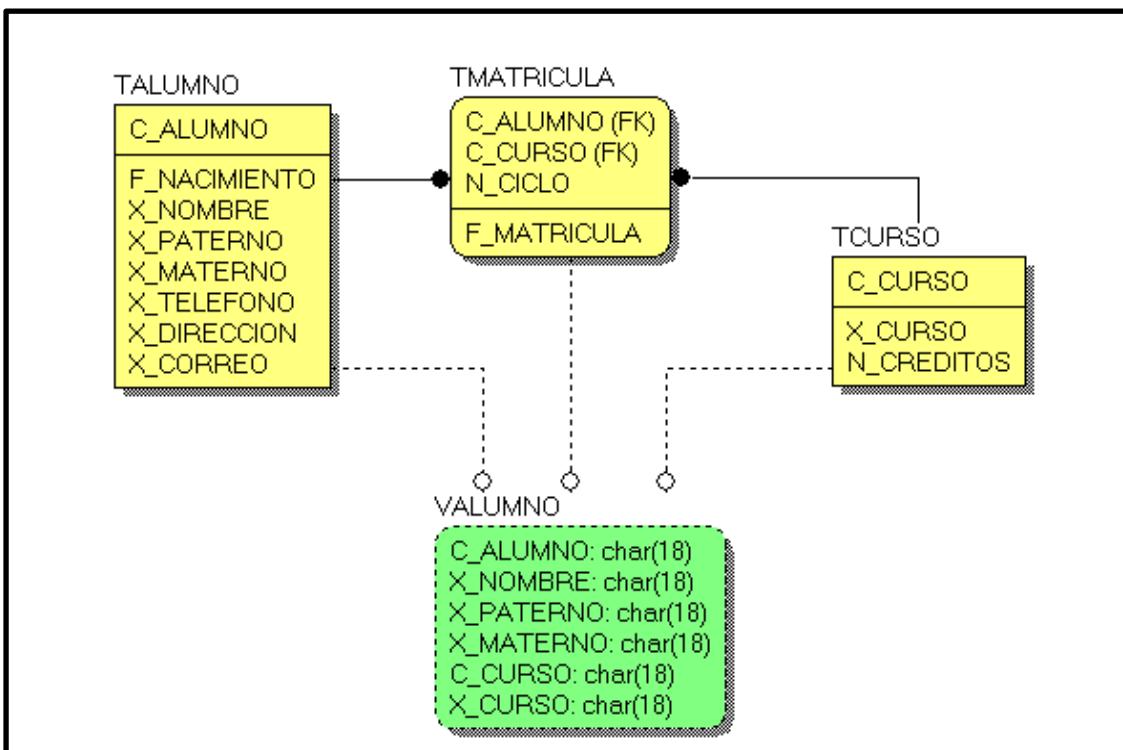


Figura 185: Implementación de la vista Alumno a partir de otras tablas
Fuente.- Tomado desde SQL Server 2014

8.2.1. 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
```

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	Serviems	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 186: Listando los registros de clientes
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 187: 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 188: 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 189: Listando los clientes cuyo segundo número telefónico sea 4 desde una vista
Fuente.- Tomado desde SQL Server 2014

8.2.2. Clasificación de las vistas.

8.2.2.1 Vistas horizontales

Un uso común de las vistas es restringir el acceso de un usuario a únicamente filas seleccionadas de una tabla.

Veamos por ejemplo, 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 Marsano 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	Serviems	Jr. Collagate 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	Emesto Uehara
7	C015	Meba	Av. Elmer Faucett 1638	4641234	50319542	D16	1993-05-12	2	Ricardo Gomez
8	C016	Cardeli	Jr. Bartolome Hemera 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 190: Listado de clientes con vistas horizontales

Fuente.- Tomado desde SQL Server 2014

8.2.2.2 Vistas verticales

Una vista vertical se caracteriza por mostrar ciertas columnas que el diseñador desea que visualice el usuario. Por ejemplo, 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	Borador de Tinta	10.00	100	500	Doc	3	FALSO

Figura 191: Listado de registros de productos

Fuente.- Tomado desde SQL Server 2014

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
```

8.3 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 192: 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 193: 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	MII	4
4	Uni	5

Figura 194: 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 195: Listando el total de productos por línea de producto
Fuente.- Tomado desde SQL Server 2014

8.3.1. 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)

8.3.1.1 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
```

8.3.1.2 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
```

8.3.1.3 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
```

8.3.1.4 Función máximo (max)

Devuelve el máximo valor de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
MAX(EXPRESSION)
```

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
```

8.3.1.5 Función mínimo (min)

Devuelve el mínimo valor de un grupo estrictamente numérico, el cual omite los valores de tipo NULL.

Su formato es:

```
MIN(EXPRESSION)
```

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
```

8.3.2. 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 196: 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 197: Total de productos importados y nacionales
Fuente.- Tomado desde SQL Server 2014

La imagen muestra que existen 16 productos nacionales y 5 importados.

8.3.3. Aplicación con procedimiento almacenado

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, para lo cual presentaremos un ejemplo en donde se implemente un procedimiento 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

Figura 198: 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

Figura 199: Listando el total de facturas registradas en el año 1999
Fuente.- Tomado desde SQL Server 2014

Actividad

Usando la base de datos COMERCIO implemente los siguientes casos:

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 200: 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
VARCHAR(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_IGV
1	102	1998-01-09	C003	1998-03-11	2	V04	0.19

Figura 201: Listando las facturas de un determinado cliente y vendedor
Fuente.- Tomado desde SQL Server 2014

3. 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 202: Listando las órdenes del proveedor Faber Castell en el año 1998
Fuente.- Tomado desde SQL Server 2014

Vistas

4. 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 Católica 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 203: Listando los datos del proveedor desde una vista
Fuente.- Tomado desde SQL Server 2014

5. 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 204: Listando las facturas mediante una vista del año 1998

Fuente.- Tomado desde SQL Server 2014

6. 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 205: Listando las órdenes de compra con estado 1
Fuente.- Tomado desde SQL Server 2014

7. 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 206: Listando las facturas registradas en el año 1999
Fuente.- Tomado desde SQL Server 2014

Agrupamiento de datos

8. 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

```

9. 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 207: Listando la cantidad de clientes por distrito
Fuente.- Tomado desde SQL Server 2014

10. 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 208: Listando el total de facturas por vendedor
Fuente.- Tomado desde SQL Server 2014

11. 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 209: Listando el total de facturas por año y por cliente
Fuente.- Tomado desde SQL Server 2014

12. 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 210: Listando los máximos montos de cada factura
Fuente.- Tomado desde SQL Server 2014

13. Procedimiento almacenado que permita 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 211: Listando el monto total acumulado de ventas por años
Fuente.- Tomado desde SQL Server 2014

14. 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 212: Listando el total de facturas por año y mes
Fuente.- Tomado desde SQL Server 2014

Resumen

1. 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.
2. Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis.
3. 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.
4. Cuando se ejecuta una consulta que contiene una subconsulta, la subconsulta se ejecuta por cada fila de la consulta principal.
5. Una vista es una consulta que es accesible como una tabla virtual en una base de datos relacional.
6. 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.