



# **Base de Datos Laboratorio**



# Índice

Presentación	5
Red de contenidos	6
Sesiones de aprendizaje	
<b>Unidad de aprendizaje 1.</b> Fundamentos de <i>SQL Server 2008</i>	
SEMANA 1 : El lenguaje <i>SQL Server 2008</i> . Implementación	7
SEMANA 2 : Creación de Tablas e Integridad de Relación	29
<b>Unidad de aprendizaje 2.</b> Modificación del contenido de una base de datos	
SEMANA 3 : Ingreso, modificación y eliminación de datos	47
SEMANA 4 : Creación y mantenimiento de Índices	55
<b>Unidad de aprendizaje 3.</b> Consultas	
SEMANA 5 : Implementación de Consultas Sencillas	61
SEMANA 6 : Consultas condicionales	71
SEMANA 7 : Examen Parcial de Teoría	
SEMANA 8 : Examen Parcial de Laboratorio	
SEMANA 9 : Funciones Agrupadas y búsqueda de grupos	81
SEMANA 10 : Consultas Multitablas	93
<b>Unidad de aprendizaje 4.</b> Subconsultas y Vistas	
SEMANA 11 : -Sub Consultas anidadas y correlacionadas -Creación de Vistas	105
SEMANA 12 : Continuación de creación de Vistas	119
<b>Unidad de aprendizaje 5.</b> Herramienta <i>ERWIN</i>	
SEMANA 12 : Uso del <i>ERWIN</i>	119
<b>Unidad de aprendizaje 6.</b> Programación avanzada en <i>SQL Server 2008</i>	
SEMANA 13 : Creación de Procedimientos Almacenados y empleo de lenguaje <i>Transact/SQL</i>	147
SEMANA 14 : Creación de Funciones y <i>Triggers</i>	167
SEMANA 15 : Examen Final de Laboratorio	



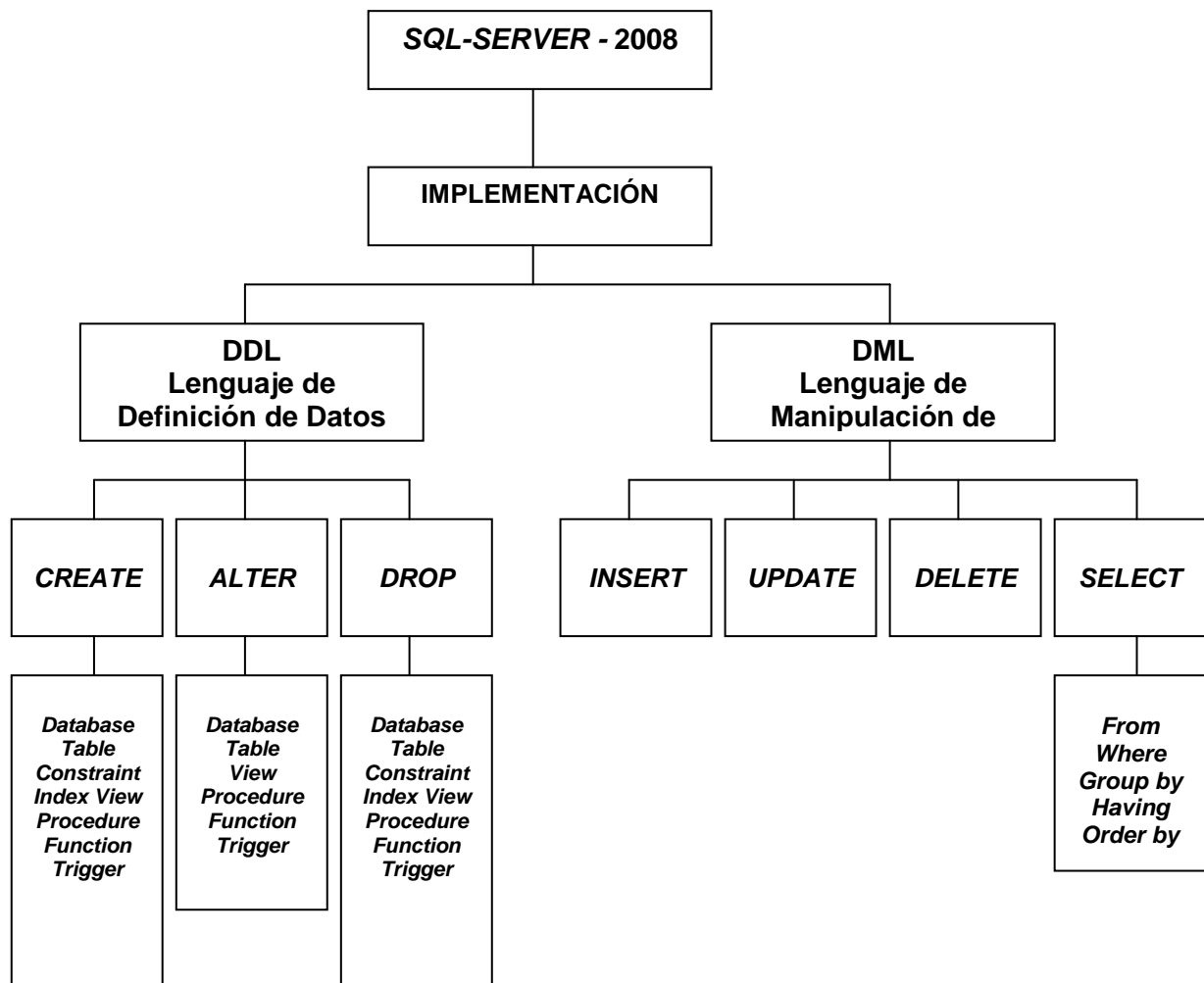
## Presentación

**Base de Datos** es un curso que pertenece a la Escuela de Tecnologías de Información y se dicta en las carreras de Administración y Sistemas, Computación e Informática y Redes y Comunicaciones. El presente manual ha sido desarrollado para que los alumnos del curso de Base de Datos laboratorio puedan aplicar los conocimientos adquiridos en el curso de teoría. Todo ello, en conjunto, le permitirá implementar una base de datos relacional previamente diseñada.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros, que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará actividades y/o autoevaluaciones que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es eminentemente práctico. Se inicia con la creación de la base de datos DEPARTAMENTOS usando el lenguaje *Transact/SQL* en el manejador de base de datos relacional *SQL Server 2008*. Se presenta el *script* que permite su creación, luego, su estructura que incluye tablas, llaves primarias y foráneas, y restricciones. Posteriormente, se efectúa la manipulación de datos (*Data Manipulation Lenguaje – DML*) para hacer uso de comandos que se emplean en la inserción, modificación y eliminación de los mismos. Una vez creada la base de datos y efectuada la inserción de registros, se ingresa a la etapa de las consultas (*SELECT*), las cuales parten de las más sencillas, una sola tabla, hasta su relación con otras tablas con el empleo del comando *INNER JOIN* y filtros (*WHERE*) con condiciones (*and*, mayor, menor, igual, etc.). Finalmente, en la última parte del manual se realizan consultas multitablas empleando funciones agrupadas, para luego grabarlas en una vista. También, se ha considerado una introducción a la creación y manipuleo de procedimientos almacenados, funciones y desencadenadores.

## Red de contenidos



UNIDAD DE  
APRENDIZAJE

**1**

## FUNDAMENTOS DE *SQL SERVER* 2008

---

### LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos construirán una base de datos relacional utilizando el gestor de base de datos *SQL Server* 2008 y los comandos del Lenguaje de Definición de Datos (DDL), asegurando la integridad de los datos mediante el empleo de restricciones tomando como caso un proceso de negocio real.

### TEMARIO

- 1.1. Creación de bases de datos
- 1.2. Identificación de los tipos de datos empleados en *SQL Server* 2008

### ACTIVIDADES PROPUESTAS

- Comprenden la visión general del curso.
- Deducen la importancia de la existencia de las bases de datos.
- Emplean los procedimientos necesarios para crear una base de datos.
- Identifican los tipos de datos que se emplean en el *SQL SERVER* 2008.

## 1. HISTORIA DEL LENGUAJE ESTRUCTURADO DE CONSULTAS (SQL)

### 1.1. Introducción

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.

### 1.2. 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, éste 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 sólo 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.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités *ANSI* e *ISO*, que debería terminar en la definición de lo que en este momento se conoce como *SQL3*. Las características principales de esta nueva encarnación de *SQL* deberían ser su transformación en un lenguaje



*stand-alone* (mientras ahora se usa como lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimedia.

### 1.3. Componentes del SQL

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

Existen dos (2) tipos de comandos SQL:

- Los comandos del Lenguaje de Definición de Datos (DDL) que permiten crear y definir nuevas bases de datos, campos e índices.
- Los comandos del Lenguaje de Manipulación de Datos (DML) que permiten modificar y generar consultas para insertar, modificar o eliminar, así como, ordenar, filtrar y extraer datos de la base de datos.

#### 1.3.1. Comandos del DDL

Comando	Descripción
<b>CREATE</b>	Utilizado para crear nuevas tablas, campos e índices
<b>DROP</b>	Empleado para eliminar tablas e índices
<b>ALTER</b>	Utilizado para modificar las tablas y agregar campos o cambiar la definición de los campos.

#### 1.3.2. Comandos del DML

Comando	Descripción
<b>SELECT</b>	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
<b>INSERT</b>	Utilizado para ingresar registros de datos en la base de datos en una única operación
<b>UPDATE</b>	Utilizado para modificar los valores de los campos y registros especificados
<b>DELETE</b>	Utilizado para eliminar registros de una tabla de una base de datos

#### 1.3.3. Cláusulas

Comando	Descripción
<b>FROM</b>	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
<b>WHERE</b>	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
<b>GROUP BY</b>	Utilizada para separar los registros seleccionados en grupos específicos
<b>HAVING</b>	Utilizada para expresar la condición que debe satisfacer cada grupo
<b>ORDER BY</b>	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico

#### 1.3.4. Operadores Lógicos

Comando	Descripción
<b>AND</b>	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas
<b>OR</b>	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta
<b>NOT</b>	Negación lógica. Devuelve el valor contrario de la expresión

#### 1.3.5. Operadores de Comparación

Comando	Descripción
<b>&lt;</b>	Menor que
<b>&gt;</b>	Mayor que
<b>&lt;&gt;</b>	Distinto de
<b>&lt;=</b>	Menor ó igual que
<b>&gt;=</b>	Mayor ó igual que
<b>=</b>	Igual que
<b>BETWEEN</b>	Utilizado para especificar un intervalo de valores
<b>LIKE</b>	Utilizado en la comparación de un modelo
<b>In</b>	Utilizado para especificar registros de una base de datos

#### 1.3.6. Funciones para el manejo de fechas y función de conversión

Función	Descripción
<b>GETDATE</b>	Devuelve la fecha del día
<b>DAY</b>	Devuelve un entero que representa el día (día del mes) de la fecha especificada
<b>MONTH</b>	Devuelve un entero que representa el mes de la fecha especificada
<b>YEAR</b>	Devuelve un entero que representa la parte del año de la fecha especificada
<b>DATEDIFF</b>	Devuelve el número de límites <i>datepart</i> de fecha y hora entre dos fechas especificadas
<b>DATEPART</b>	Devuelve un entero que representa el parámetro <i>datepart</i> especificado del parámetro <i>date</i> especificado
<b>CONVERT</b>	Convierte una expresión de un tipo de datos en otro

- 1.3.7.** Funciones de agregado: las funciones de agregado se usan dentro de una cláusula *SELECT*, en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

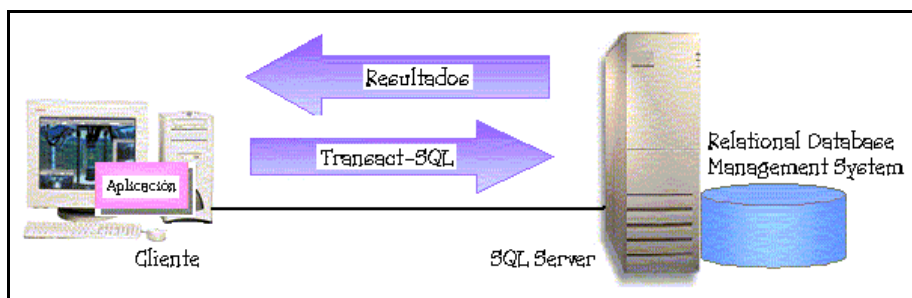
Comando	Descripción
<b>AVG</b>	Se emplea para calcular el promedio de los valores de un campo determinado
<b>COUNT</b>	Se emplea para devolver la cantidad de registros de la selección
<b>SUM</b>	Se emplea para devolver la suma de todos los valores de un campo determinado
<b>MAX</b>	Se emplea para devolver el valor más alto de un campo o expresión especificada
<b>MIN</b>	Se emplea para devolver el valor más bajo de un campo o expresión especificada

## 2. 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 éstos una herramienta bajo la cual puedan reducir considerablemente el tiempo del proceso de búsqueda en profundidad de los datos almacenados.

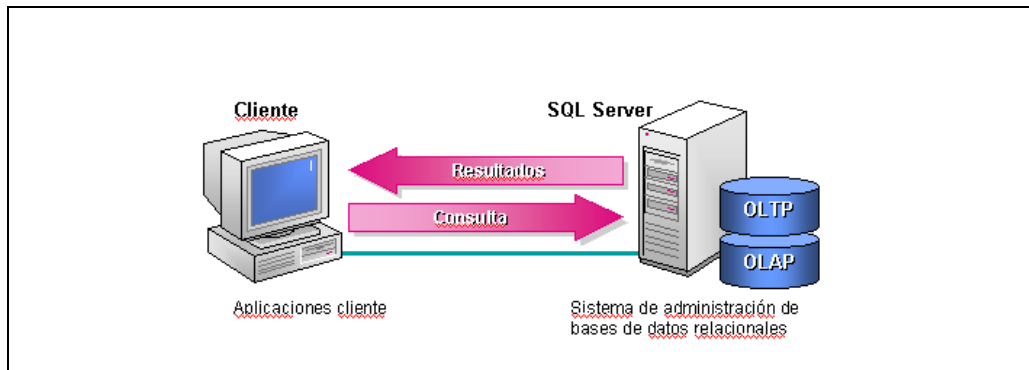
### 2.1. Implementación de las base de datos con **SQL SERVER**

*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*.



### 2.2. Arquitectura Cliente / Servidor

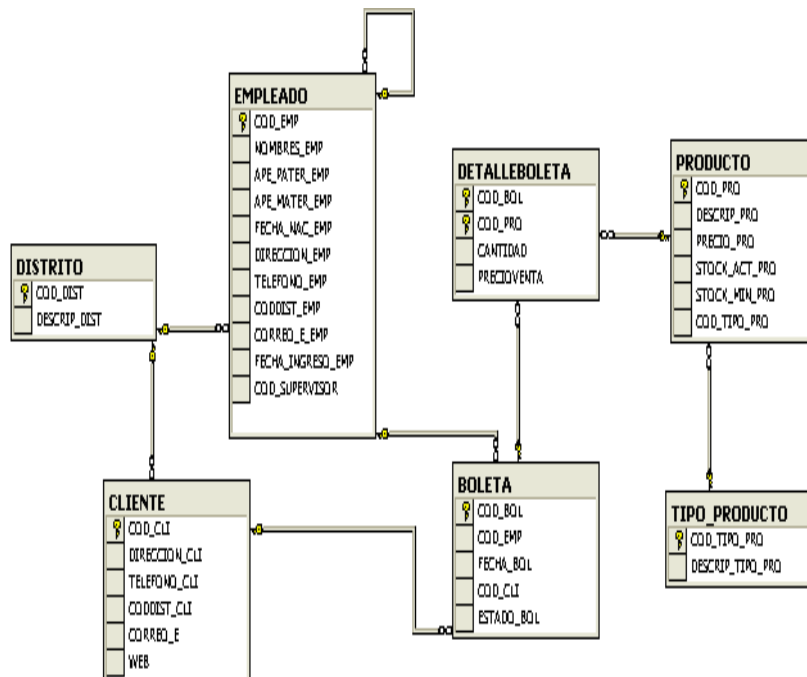
*SQL Server* usa la arquitectura Cliente / Servidor para separar la carga de trabajo en tareas que se ejecuten en computadoras tipo Servidor y tareas que se ejecuten en computadoras tipo Cliente:



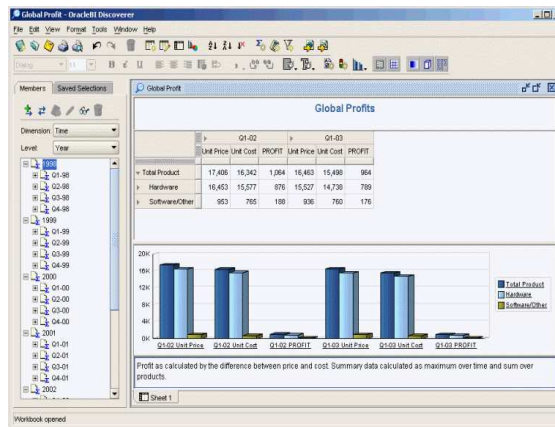
- El Cliente es responsable de la parte lógica y de presentar la información al usuario. Generalmente, el cliente ejecuta en una o más computadoras Cliente, aunque también puede ejecutarse en una computadora que cumple las funciones de Servidor con SQL Server.
- SQL Server administra bases de datos y 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 2008 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.



### 2.3. Sistema administrador para bases de datos Relacionales (RDBMS)

El RDBMS es responsable de:

- Mantener las relaciones entre la información y la base de datos.
- Asegurarse de que la información sea almacenada correctamente, es decir, que las reglas que definen las relaciones entre los datos no sean violadas.
- Recuperar toda la información en un punto conocido en caso de que el sistema falle.

### 2.4. TRANSACT SQL

Éste 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.

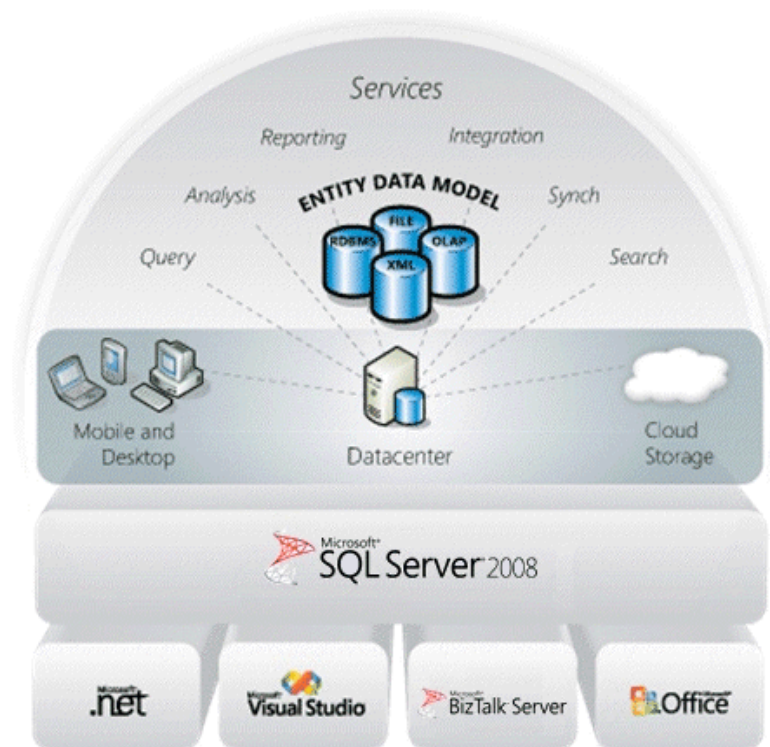
### 2.5. SQL SERVER 2008

Esta última versión del SQL Server Database Engine (Motor de base de datos de SQL Server) incluye nuevas características y mejoras que aumentan la eficacia y la productividad de los arquitectos, los programadores y los administradores que diseñan, desarrollan y mantienen sistemas de almacenamiento de datos.



Las ediciones disponibles de *SQL Server 2008* son:

- **SQL Server 2008 Enterprise.** Con soporte de escalabilidad de clase enterprise, data-warehousing, seguridad, análisis avanzado y reportería.
- **SQL Server 2008 Standard.** Para aplicaciones departamentales.
- **SQL Server 2008 Workgroup.** Para aplicaciones *branch* (sucursales).
- **SQL Server 2008 Web.** Para aplicaciones web. Diseñada para ser de bajo costo, tener alta disponibilidad y a gran escala.
- **SQL Server 2008 Developer.** Similar a la versión Enterprise, pero se licencia únicamente para efectos de desarrollo de aplicaciones y es por cada desarrollador.
- **SQL Server 2008 Express.** GRATUITA. Para aplicaciones de escritorio y efectos de aprendizaje.
- **SQL Server Compact 3.5.** GRATUITA. Para aplicaciones móviles.



### 3. CREACIÓN DE BASES DE DATOS

#### 3.1. Definición de base de datos

Una base de datos es un contenedor de objetos relacionados entre sí, de manera lógica y coherente. Estos objetos incluyen los orígenes de datos, dimensiones compartidas y funciones de base de datos.

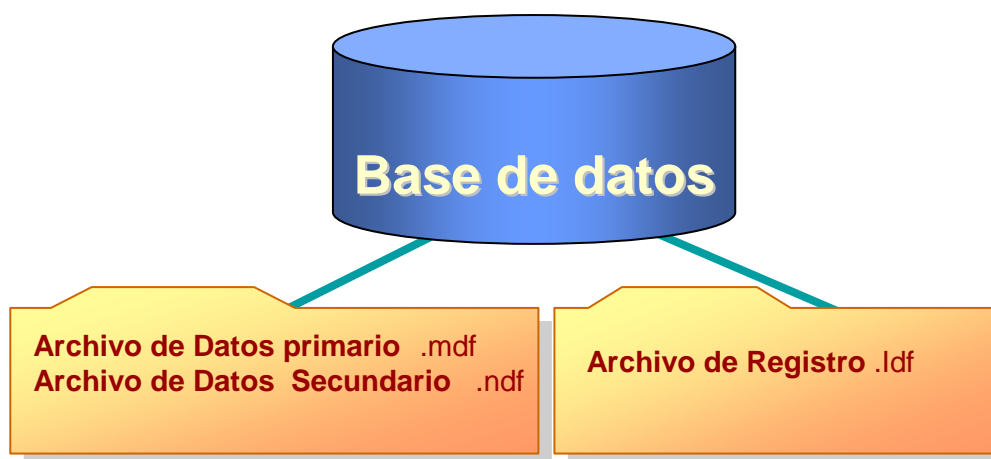
### 3.2. Aprendiendo a crear una base de datos

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 32.767 bases de datos.

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



- **Archivos principales**  
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.
- **Archivos secundarios**  
Estos archivos contienen todos los datos que no caben en el archivo de datos principal. No es necesario que las bases de datos tengan archivos de datos secundarios si el archivo principal es lo suficientemente grande como para contener todos los datos. Algunas bases de datos pueden ser muy grandes y necesitar varios archivos de datos secundarios o utilizar archivos secundarios en unidades de disco distintas, de modo que los datos estén distribuidos en varios discos. Tiene extensión .NDF.
- **Registro de transacciones**  
Estos archivos contienen la información de registro que se utiliza para recuperar la base de datos. Debe haber al menos un archivo de registro de transacciones para cada base de datos, aunque puede haber más de uno. El tamaño mínimo para un archivo de registro es 512 *kilobytes* (KB). Tiene extensión .LDF.

**Importante:** Los archivos de datos y de registro de transacciones de *Microsoft® SQL Server™ 2008* 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 suplantando los datos de los archivos ya eliminados que hubieran quedado en el disco. Aunque esto provoque que el proceso de creación de los archivos sea más largo, es mejor, pues así se evita que el sistema operativo tenga que llenar los archivos con ceros cuando se escriban por primera vez datos en los archivos durante las operaciones habituales con la base de datos. De esta manera, se mejora el rendimiento de las operaciones cotidianas.

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

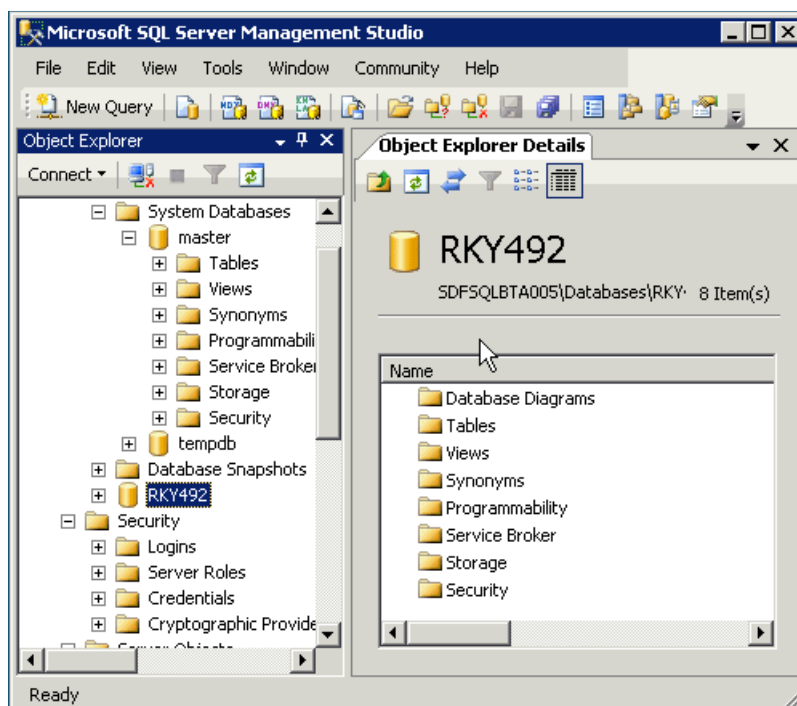
Después de crear una base de datos, **se recomienda** crear una copia de seguridad de la base de datos **MASTER**.

### 3.3. Creación de una base de datos usando la herramienta de *SQL SERVER 2008*

Para crear una base de datos, se deben seguir los siguientes pasos:

- Expandir un grupo de servidores y, a continuación, un servidor.
- Con el botón secundario del *mouse* (ratón) en bases de datos y, a continuación, en nueva base de datos.

Escriba un nombre para la nueva base de datos.





c) Haga clic

**Observación:**

Los archivos de registro de transacciones y de la base de datos principal se crean utilizando el nombre de base de datos que ha especificado como prefijo, por ejemplo, newdb\_Data.mdf y newwdb\_Log.ldf. Los tamaños iniciales de estos archivos son los mismos que los tamaños predeterminados especificados para la base de datos **MODEL**. El archivo principal contiene las tablas del sistema para la base de datos.

1. Para cambiar los valores predeterminados del nuevo archivo de base de datos principal, haga clic en la ficha General. Para cambiar los valores predeterminados correspondientes al nuevo archivo de registro de transacciones, haga clic en la ficha Registro de transacciones.
2. Para cambiar los valores predeterminados proporcionados en las columnas Nombre de archivo, Ubicación, Tamaño inicial (*megabytes*) y Grupo de archivos (no aplicable para el registro de transacciones), haga clic en la celda apropiada para cambiar y escribir el nuevo valor.
3. Para especificar cómo debe crecer el archivo, haga clic en una de estas opciones:
  - 3.1. Para permitir que el archivo actualmente seleccionado crezca cuando sea necesario más espacio para los datos, haga clic en Crecimiento automático del archivo.
  - 3.2. Para especificar que el archivo debe crecer en incrementos fijos, haga *click* en “*megabytes*” y especifique un valor.
  - 3.3. Para especificar que el archivo debe crecer en un porcentaje de su tamaño actual, haga clic en “por porcentaje” y especifique un valor.
  - 3.4. Para especificar el límite de tamaño del archivo, haga clic en una de estas opciones:
    - 3.4.1. Para que el archivo crezca tanto como sea necesario, haga clic en No limitar el crecimiento de los archivos.
    - 3.4.2. Para especificar el tamaño máximo que se debe permitir que alcance el archivo, haga clic en Limitar crecimiento de archivo a (*megabytes*) y especifique un valor.

**Nota:** El tamaño máximo de una base de datos está determinado por la cantidad de espacio de disco disponible y los límites de licencia establecidos por la versión de *SQL Server* que utilice.

### 3.4. ¿Cómo crear una base de datos usando *TRANSACT/SQL*?

#### ACTIVIDAD 3.4.1 Crear una base de datos sin especificar los archivos

Este ejemplo crea una base de datos llamada `bd_ejemplo`, automáticamente se crean los archivos principal y de registro de transacciones correspondientes en la carpeta asignada por defecto, por ejemplo en:

#### C:\Program Files\Microsoft SQL Server\MSSQL\Data

Debido a que la instrucción no tiene elementos `<filespec>`, el archivo principal de la base de datos toma el tamaño del archivo principal de la base de datos **MODEL**. El registro de transacciones toma el tamaño del archivo del registro de transacciones de la base de datos **MODEL**. Como no se ha especificado el parámetro *MAXSIZE*, los archivos pueden crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE bd_ejemplo
Go
```

**NOTA:** Puede usar el comando `sp_helpDB` para presentar información acerca de la base de datos especificada o de todas las bases de datos.

**Sintaxis:** `SP_HELPDB` [Nombre de la base de datos]

**Ejemplo:**

`SP_HELPDB bd_ejemplo`

SUITE306-ST1...QLQuery1.sql\*

```
create database bd_ejemplo
go
use bd_ejemplo
go
sp_helpdb bd_ejemplo
```

	name	db_size	owner	dbid	created	status	compatibility_level
1	bd_ejemplo	2.73 MB	sa	26	Dec 9 2008	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	90

	name	fileid	filename	filegroup	size	maxsize	growth	usage
1	bd_ejemplo	1	D:\Archivos de programa\Microsoft SQL Server\MSS...	PRIMARY	2240 KB	Unlimited	1024 KB	data only
2	bd_ejemplo_log	2	D:\Archivos de programa\Microsoft SQL Server\MSS...	NULL	560 KB	2147483648 KB	10%	log only

### 3.5 ESPECIFICANDO LAS PROPIEDADES PARA CADA ARCHIVO QUE CONFORMA UNA BASE DE DATOS:

Para el archivo primario, los archivos secundarios y los archivos de registro de transacciones se pueden especificar las siguientes propiedades:

- NAME:** Nombre lógico del archivo.
- FILENAME:** Nombre físico, en el cual se debe especificar la ruta (ubicación) donde será creado el archivo.
- SIZE:** Tamaño inicial, por defecto está dado en *megabytes*.
- MAXSIZE:** Tamaño máximo.
- FILEGROWTH:** Crecimiento del archivo.

#### ACTIVIDAD 3.5.1: Crear una base de datos individual

En este ejemplo se crea una base de datos llamada DEPARTAMENTOS en la carpeta D:\Data\ y se especifica un único archivo. El archivo especificado se convierte en el archivo principal y se crea automáticamente un archivo de registro de transacciones de 1 *megabytes*. Como no se especifican la unidad en el parámetro *SIZE* del archivo principal, se asigna por defecto en *megabytes*. Ya que no existe <filespec> para el archivo de registro de transacciones, éste no tiene el parámetro *MAXSIZE* y puede crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE Departamentos
ON
(
    NAME = Departamentos_Data ,
    FILENAME = 'D:\Data\Departamentos_Data.mdf' ,
    SIZE = 4 ,
    MAXSIZE = 10 ,
    FILEGROWTH = 1
)
GO
```

#### ACTIVIDAD 3.5.2: Crear una base de datos sin especificar tamaño (*size*), ni máximo tamaño (*maxsize*), ni crecimiento (*filegrowth*)

Esta actividad crea una base de datos llamada Departamentos2. El archivo departamentos2\_Data se convierte en el archivo principal, con un tamaño igual al tamaño del archivo principal de la base de datos **MODEL**. El archivo de registro de transacciones se crea automáticamente y es un 25 por ciento del tamaño del archivo principal, o 512 *kilobytes*, el que sea mayor. Como no se ha especificado *MAXSIZE*, los archivos pueden crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE Departamentos2
ON
(NAME = Departamentos2_Data,
FILENAME = 'D:\ Data \Departamentos2_Data.mdf'
)
GO
```

### ACTIVIDAD 3.5.3: Crear una base de datos especificando dos archivos, archivo de datos y archivo de transacciones

Esta actividad crea una base de datos llamada Departamentos3 en la carpeta D:\Data\, su archivo principal contará con un tamaño inicial de 40 *megabytes*, un tamaño máximo de 100 *megabytes* y un crecimiento de 1 *megabytes*. Su archivo de registro contará con un tamaño inicial de 5 *megabytes*, un tamaño máximo de 40 *megabytes* y un crecimiento de 10%.

```
CREATE DATABASE Departamentos3
ON
(NAME = Departamentos3_Data,
FILENAME = 'D:\ Data \Departamentos3_Data.mdf',
SIZE = 40,
MAXSIZE = 100,
FILEGROWTH = 1
)
LOG ON
(NAME = Departamentos3_Log,
FILENAME = 'D:\ Data \Departamentos3_Log.ldf',
SIZE = 5,
MAXSIZE = 40,
FILEGROWTH = 10%
)
GO
```

**ACTIVIDAD 3.5.4 Crear una base de datos especificando el archivo de datos, un archivo secundario y un archivo de transacciones:**

Esta actividad crea una base de datos llamada Departamentos3 en la carpeta D:\DATA\, su archivo principal contará con un tamaño inicial de 15MB, un tamaño máximo de 200 *megabytes* y un crecimiento de 20%. Su archivo secundario contará con un tamaño inicial de 10 *megabytes*, un tamaño máximo de 80 *megabytes* y un crecimiento de 2 *megabytes*. Su archivo de registro contará con un tamaño inicial de 10 *megabytes*, un tamaño máximo de 70 *megabytes* y un crecimiento de 5 *megabytes*.

```
CREATE DATABASE Departamentos4

ON

(NAME = Departamentos4_Data,
FILENAME = 'D:\ Data \Departamentos4_Data.mdf',
SIZE = 15, MAXSIZE = 200, FILEGROWTH = 20%
) ,

(NAME = Departamentos4_Sec,
FILENAME = 'D:\ Data \Departamentos4_Sec.ndf',
SIZE = 10, MAXSIZE = 80, FILEGROWTH = 2
)

LOG ON

(NAME = Departamentos4_Log,
FILENAME = 'D:\ Data \Departamentos4_Log.ldf',
SIZE = 10, MAXSIZE = 70, FILEGROWTH = 5
)

GO
```

**4. TIPOS DE DATOS**

Los tipos de datos definen el valor de datos que se permite en cada columna. *SQL Server* proporciona varios tipos de datos diferentes. Ciertos tipos de datos comunes tienen varios tipos de datos de *SQL Server* asociados. Se debe elegir los tipos de datos adecuados que permitan optimizar el rendimiento y conservar el espacio en el disco.

#### 4.1 Categorías de tipos de datos del sistema

La siguiente tabla asocia los tipos de datos comunes con los tipos de datos del sistema proporcionados por *SQL Server*. La tabla incluye los sinónimos de los tipos de datos por compatibilidad con ANSI<sup>1</sup>.

Tipos de datos comunes	Tipos de datos del sistema de SQL Server	Sinónimo ANSI	Número de Bytes
Entero	<b>Int</b>	Integer	4
	<b>bigint</b>	-	8
	<b>smallint, tinyint</b>	-	2,1
Numérico Exacto	<b>decimal[(p,s)]</b>	Dec	2-17
	<b>numeric[(p,s)]</b>	-	
Numérico Aproximado	<b>float[(n)]</b>	double precisión,	8
	<b>real</b>	float[(n)] para n=8-15	4
		float[(n)] para n=1-7	
Moneda	<b>money, smallmoney</b>	-	8,4
Fecha y hora	<b>datetime, smalldatetime</b>	-	8
			4
Carácter	<b>char[(n)]</b>	character [(n)]	0-8000
	<b>varchar[(n)]</b>	char VARYING[(n)]	
	<b>text</b>	character VARYING[(n)]	
Caracteres Unicote	<b>nchar[(n)]</b>	-	0-2 GB
	<b>nvarchar[(n)]</b>		0-8000
Binario	<b>binary [(n)]</b>		(4000 caracteres)
	<b>varbinary [(n)]</b>		0-2 GB
Imagen	<b>image</b>	binary VARYING[(n)]	0-8000
		-	0 a 2 GB

**Nota:** *SQL SERVER* 2008 agrega nuevos tipos de datos. Estos nuevos tipos de datos permiten una mejora en el almacenamiento y en el trabajo con tipos de datos fecha y hora, incluyendo múltiples zonas horarias y cálculos mejorados. Estos nuevos tipos de datos son: *datetime2*, *date*, *time* y *datetimeoffset*.

<sup>1</sup> American National Standards Institute

## ACTIVIDADES A DESARROLLAR EN CLASE

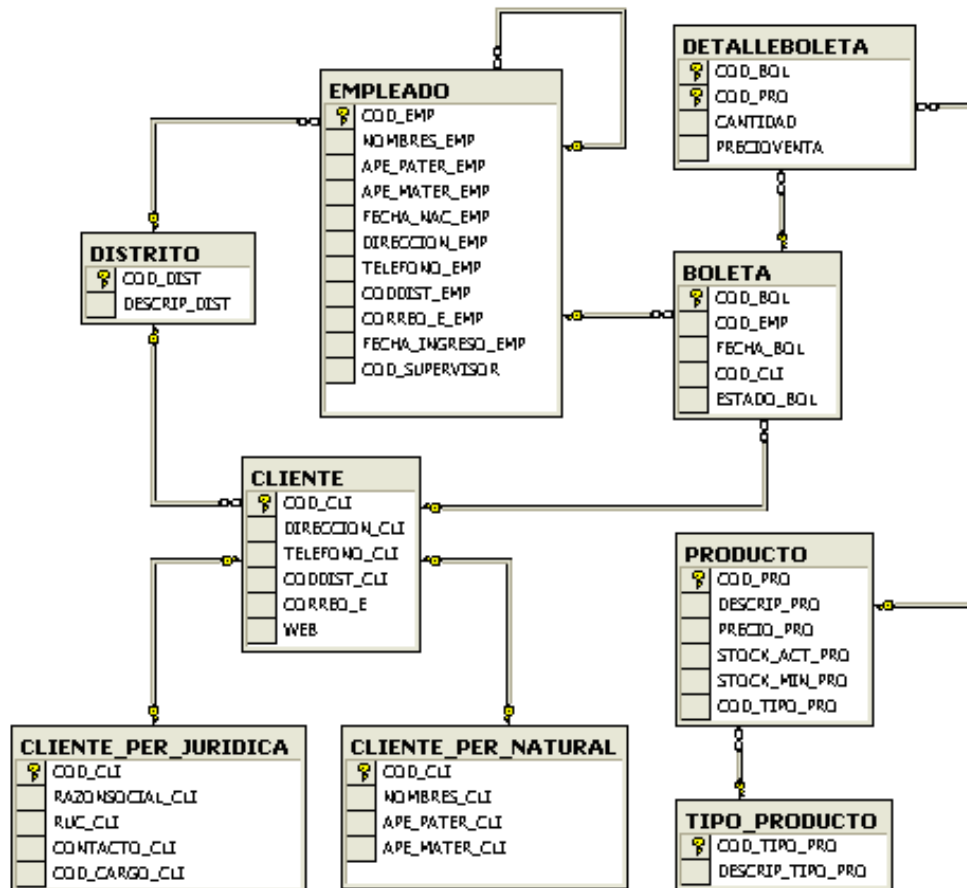
Usando **TRANSACT/SQL**, cree las siguientes bases de datos:

1. **VENTAS2009\_1**, en la carpeta **D:\DATABASE\**, considerando que el **archivo principal** tiene tamaño de 20 *megabytes*, un tamaño máximo de 80 *megabytes* y un incremento de 10 *megabytes*.
2. **VENTAS2009\_2**, en la carpeta **D:\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%.
3. **VENTAS2009\_3**, en la carpeta **D:\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%.

## ACTIVIDADES PROPUESTAS

### Caso: VENTAS

Se ha diseñado una base de datos para el control de las ventas realizadas en una empresa, como se detalla en el siguiente diagrama:



1. Cree la base de datos Ventas indicando propiedades para el archivo primario, un archivo secundario y un archivo de registro.
2. Identificar los tipos de datos que le corresponde a los campos de las tablas EMPLEADO, BOLETA y PRODUCTO.
3. Cree la base de datos para su **proyecto de investigación**, el cual tendrá definida la ruta para la creación de los archivos y la especificación de los tres (3) archivos (archivo de registro, archivo secundario y archivo de transacciones).



## ACTIVIDADES ADICIONALES

Los ejemplos mostrados a continuación han sido tomados de la Ayuda en Línea del *Microsoft SQL 2008*.

### A. Cree una base de datos sin especificar archivos

En este ejemplo se crea la base de datos *mytest*, y el archivo principal y de registro de transacciones correspondientes. Debido a que la instrucción no tiene elementos <filespec>, el archivo de la base de datos principal tiene el tamaño del archivo principal de la base de datos **model**. El registro de transacciones se establece en el mayor de estos valores: 512 KB o el 25% del tamaño del archivo de datos principal. Como no se ha especificado *MAXSIZE*, los archivos pueden crecer hasta llenar todo el espacio disponible en el disco.

```
USE master;
GO
IF DB_ID (N'mytest') IS NOT NULL
DROP DATABASE mytest;
GO
CREATE DATABASE mytest;
GO
-- Verify the database files and sizes
SELECT name, size, size*1.0/128 AS [Size in MBs]
FROM sys.master_files
WHERE name = N'mytest';
GO
```

### B. Cree una base de datos que especifica los archivos de datos y de registro de transacciones:

En el ejemplo siguiente se crea la base de datos *Sales* (Ventas). Debido a que no se utiliza la palabra clave *PRIMARY*, el archivo (*Sales\_dat*) se convierte en el archivo principal. Como no se especifica *megabytes* ni *kilobytes* en el parámetro *SIZE* del archivo *Sales\_dat*, se utiliza *megabytes* y el tamaño se asigna en *megabytes*. El tamaño del archivo *Sales\_log* se asigna en *megabytes* porque el sufijo MB se ha indicado explícitamente en el parámetro *SIZE*.

```
USE master;
GO
IF DB_ID (N'Sales') IS NOT NULL
DROP DATABASE Sales;
GO
-- Get the SQL Server data path
DECLARE @data_path nvarchar (256);
```

```

SET @data_path = (SELECT SUBSTRING (physical_name, 1, CHARINDEX(N'master.mdf',
LOWER (physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1);

-- execute the CREATE DATABASE statement
EXECUTE ('CREATE DATABASE Sales
ON
(NAME = Sales_dat,
FILENAME = "' + @data_path + 'saledat.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5)
LOG ON
(NAME = Sales_log,
FILENAME = "' + @data_path + 'salelog.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB)'
);
GO

```

### C. Cree una base de datos mediante la especificación de múltiples archivos de datos y de registro de transacciones:

En el ejemplo siguiente se crea la base de datos *Archive*, que tiene tres archivos de datos de 100 *megabytes* y dos archivos de registro de transacciones de 100 *megabytes*. El archivo principal es el primer archivo de la lista y se especifica explícitamente con la palabra clave *PRIMARY*. Los archivos de registro de transacciones se especifican a continuación de las palabras clave *LOG ON*. Tenga en cuenta las extensiones usadas para los archivos en la opción *FILENAME*: *.mdf* se usa para archivos de datos principales, *.ndf* para archivos de datos secundarios y *.ldf* para archivos de registro de transacciones.

```

USE master;
GO
IF DB_ID (N'Archive') IS NOT NULL
DROP DATABASE Archive;
GO
-- Get the SQL Server data path
DECLARE @data_path nvarchar(256);
SET @data_path = (SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf',
LOWER(physical_name)) - 1)

```

```
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1);

-- execute the CREATE DATABASE statement
EXECUTE ('CREATE DATABASE Archive
ON
PRIMARY
(NAME = Arch1,
FILENAME = "' + @data_path + 'archdat1.mdf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
( NAME = Arch2,
FILENAME = "' + @data_path + 'archdat2.ndf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
( NAME = Arch3,
FILENAME = "' + @data_path + 'archdat3.ndf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20)
LOG ON
(NAME = Archlog1,
FILENAME = "' + @data_path + 'archlog1.ldf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
(NAME = Archlog2,
FILENAME = "' + @data_path + 'archlog2.ldf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20)'
);
GO
```

Al ejecutar el código anterior, se mostrará lo siguiente:

The screenshot shows the Microsoft SQL Server Management Studio interface. The main window displays a SQL query that drops the 'Archive' database if it exists and then creates a new database with a primary data file and four log files. The query is as follows:

```
USE master;
GO
IF DB_ID (N'Archive') IS NOT NULL
DROP DATABASE Archive;
GO
-- Get the SQL Server data path
DECLARE @data_path nvarchar(256);
SET @data_path = (SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_r
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1);

-- execute the CREATE DATABASE statement
EXECUTE ('CREATE DATABASE Archive
ON
PRIMARY
(NAME = Arch1,
-- ... (rest of the query is truncated in the image)

The Results pane shows the following table:


|   | name    | db_size   | owner              | dbid | created     | status                                              | compatibility_level |
|---|---------|-----------|--------------------|------|-------------|-----------------------------------------------------|---------------------|
| 1 | Archive | 500.00 MB | CA311-ST19\Student | 9    | Dec 13 2008 | Status=ONLINE, Updateability=READ_WRITE, UserAcc... | 90                  |



Below this, a detailed view of the database files is shown:



|   | name     | fileid | filename                                             | filegroup | size      | maxsize   | growth   | usage     |
|---|----------|--------|------------------------------------------------------|-----------|-----------|-----------|----------|-----------|
| 1 | Arch1    | 1      | I:\Program Files\Microsoft SQL Server\MSSQL.1\MSS... | PRIMARY   | 102400 KB | 204800 KB | 20480 KB | data only |
| 2 | Archlog1 | 2      | I:\Program Files\Microsoft SQL Server\MSSQL.1\MSS... | NULL      | 102400 KB | 204800 KB | 20480 KB | log only  |
| 3 | Arch2    | 3      | I:\Program Files\Microsoft SQL Server\MSSQL.1\MSS... | PRIMARY   | 102400 KB | 204800 KB | 20480 KB | data only |
| 4 | Arch3    | 4      | I:\Program Files\Microsoft SQL Server\MSSQL.1\MSS... | PRIMARY   | 102400 KB | 204800 KB | 20480 KB | data only |
| 5 | Archlog2 | 5      | I:\Program Files\Microsoft SQL Server\MSSQL.1\MSS... | NULL      | 102400 KB | 204800 KB | 20480 KB | log only  |



The status bar at the bottom indicates: Query executed successfully. CA311-ST19 (9.0 RTM) CA311-ST19\Student (54) master 00:00:00 6 rows.


```

# Resumen

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

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

📖 Los tipos de datos definen el valor de datos que se permite en cada columna. *SQL Server* proporciona varios tipos de datos diferentes.

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

🔗 <http://mredison.wordpress.com/2008/05/27/tutorial-creacin-de-una-bd-desde-script-sql-server-2008/>

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

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

Libro en pantalla de *SQL Server 2008* - creación de una base de datos en *SQL*.

🔗 <http://msdn.microsoft.com/es-es/library/ms176061.aspx>

Libros en pantalla de *SQL Server 2008* – Ejemplos.





## FUNDAMENTOS DE *SQL SERVER* 2008

---

### LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos construirán una base de datos relacional utilizando el gestor de base de datos *SQL – Server* 2008 y los comandos del Lenguaje de Definición de Datos (DDL), asegurando la integridad de los datos mediante el empleo de restricciones tomando como caso un proceso de negocio real.

### TEMARIO

1.1 Creación de tablas e integridad de relación

1.2 Implementación de restricciones *PRIMARY KEY*, *FOREIGN KEY*, *CHECK*, *UNIQUE* y *DEFAULT*

### ACTIVIDADES PROPUESTAS

- Efectúan la creación de tablas.
- Definen las claves o llaves de cada tabla y las relacionan.
- Precisan restricciones sobre los campos definidos.

## 1. APRENDIENDO A CREAR TABLAS

### 1.1. DEFINICIÓN DE UNA TABLA

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.

Las columnas de la tabla recién creada se definen en el cuerpo de las sentencias *CREATE TABLE*. La definición de las columnas aparece en una lista separada por comas e incluida entre paréntesis. La definición de la columna determina el orden de izquierda a derecha de la columna en la tabla.

**a. Nombre de columna:**

Pueden ser iguales a los nombre de las columnas de otras tablas, pero no pueden tener el nombre de una columna existente en la misma tabla.

**b. Tipo de datos**

Identifica la clase de datos que la columna almacenará.

**c. Datos requeridos:**

Si la columna contiene datos requeridos se debe especificar si la columna no acepta valores nulos. La cláusula *NOT NULL* impide que aparezcan valores *NULL* en la columna. Por defecto se admiten valores *NULL*. Las tablas suelen estar relacionadas con otras tablas.

### 1.2. Clave Primaria y Clave Foránea

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.

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.

### 1.3. Definición de relaciones

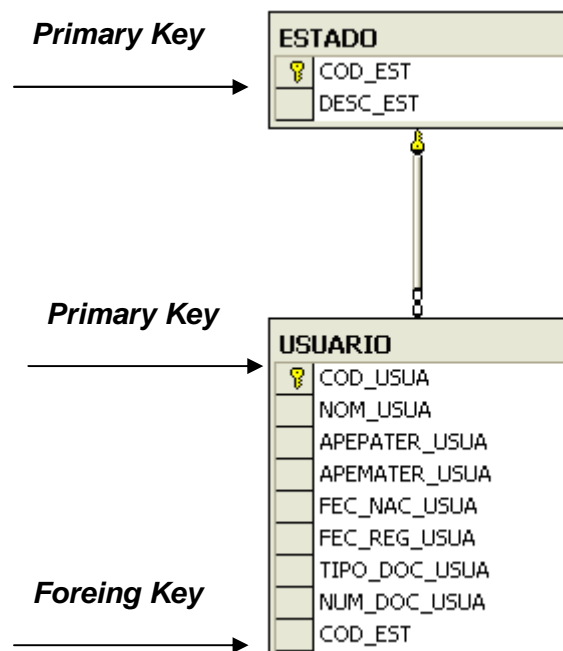
El término "relaciones" usualmente se refiere a las relaciones entre claves foráneas y primarias, y entre tablas. Estas relaciones deben ser definidas porque determinan qué columnas son o no claves primarias o claves foráneas. A continuación, veamos los tipos de relación que pueden existir entre las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

#### 1.2.1 Relación Uno-a-Varios:

La relación uno a varios (uno a muchos), es el tipo de relación más común. 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: cada editor produce muchos títulos, 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.

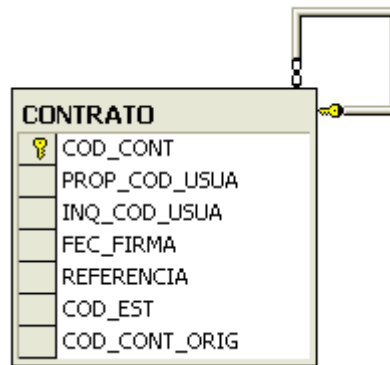
En el ejemplo tenemos:

**Un estado lo es de muchos usuarios pero un usuario tiene únicamente un estado.**



A continuación, se muestra la relación uno a muchos en una **relación recursiva**.

**Un contrato puede ser la extensión de otro contrato y un contrato puede tener muchas extensiones.**

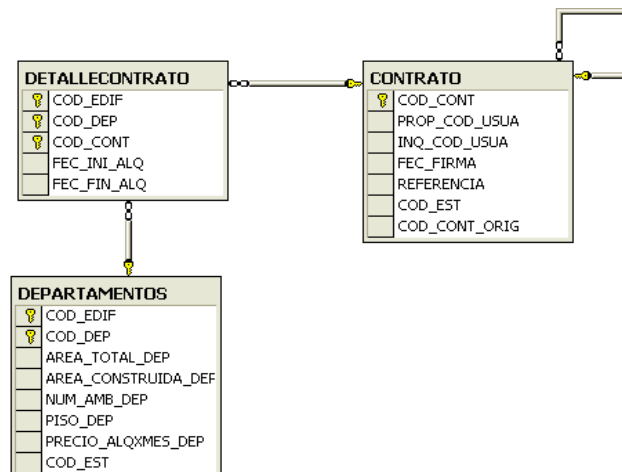


### 1.2.2 Relaciones de varios a varios

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, defina una tercera tabla, denominada tabla de unión, cuya clave principal está formada 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).

Otro ejemplo:

**Un contrato puede registrar muchos departamentos y un departamento puede estar registrado en muchos contratos. Para poder implementar esta relación compleja debemos adicionar una tabla de detalle (DetalleContrato).**



### 1.2.3 Relaciones de uno a uno

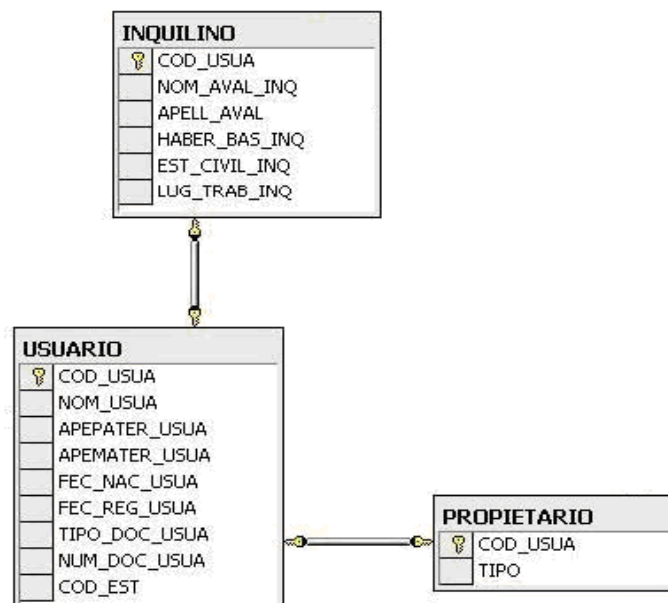
En una relación de uno a uno, una fila de la tabla A no puede tener más de una fila coincidente en la tabla B y viceversa. Una relación de uno a uno se crea si las dos columnas relacionadas son claves principales o tienen restricciones únicas.

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
- Aislar 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.

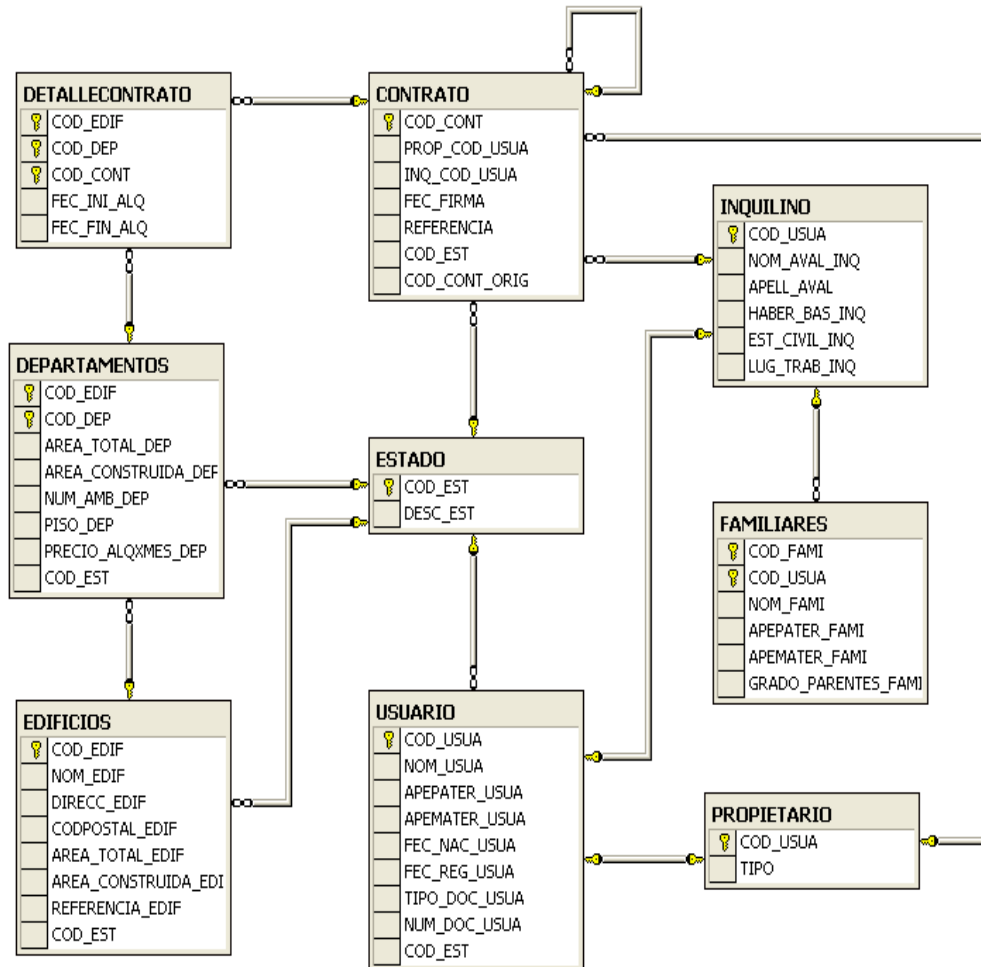
El ejemplo, a continuación, muestra a la tabla USUARIO (generalización) relacionándose con la tabla PROPIETARIO (especialización 1) y la tabla INQUILINO (especialización 2), de uno a uno.



Ahora que ya conoce el concepto de tablas y el concepto de relaciones, empezará a implementar algunas tablas para distintos casos.

## ACTIVIDADES A DESARROLLAR EN CLASE

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



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

1. Cree la base de datos Departamentos
2. Active la base de datos Departamentos
3. Cree las tablas mostradas
4. Agregue las llaves Primarias (*ADD PRIMARY KEY*)
5. Agregue las llaves Foráneas y Relaciones (*ADD FOREIGN KEY – REFERENCES*)

**SOLUCIÓN:**

Implemente las tablas, las llaves primarias, las llaves foráneas y las relaciones paso a paso:

**SCRIPT EN TRANSACT/SQL  
SQL SERVER 2008**

```

CREATE DATABASE DEPARTAMENTOS

GO

USE DEPARTAMENTOS

CREATE TABLE CONTRATO (
    COD_CONT          char(6) NOT NULL,
    PROP_COD_USUA     char(6) NULL,
    INQ_COD_USUA      char(6) NULL,
    FEC_FIRMA         datetime NOT NULL,
    REFERENCIA        varchar(100) NULL,
    COD_EST           char(6) NULL,
    COD_CONT_ORIG     char(6) NULL
)

CREATE TABLE DEPARTAMENTOS (
    COD_EDIF          char(6) NOT NULL,
    COD_DEP           char(6) NOT NULL,
    AREA_TOTAL_DEP    decimal NOT NULL,
    AREA_CONSTRUIDA_DEP decimal NULL,
    NUM_AMB_DEP       int NULL,
    PISO_DEP          int NULL,
    PRECIO_ALQXMES_DEP money NULL,
    COD_EST           char(6) NULL
)

CREATE TABLE DETALLECONTRATO (
    COD_EDIF          char(6) NOT NULL,
    COD_DEP           varchar(6) NOT NULL,
    COD_CONT          char(6) NOT NULL,
    FEC_INI_ALQ       datetime NOT NULL,
    FEC_FIN_ALQ       datetime NOT NULL
)

CREATE TABLE EDIFICIOS (
    COD_EDIF          char(6) NOT NULL,
    NOM_EDIF          varchar(60) NOT NULL,
    DIRECC_EDIF       varchar(60) NOT NULL,
    CODPOSTAL_EDIF    varchar(4) NOT NULL,
    AREA_TOTAL_EDIF   decimal NOT NULL,
    AREA_CONSTRUIDA_EDIF decimal NULL,
    REFERENCIA_EDIF   varchar(100) NULL,
    COD_EST           char(6) NULL
)

CREATE TABLE ESTADO (
    COD_EST           char(6) NOT NULL,
    DESC_EST          varchar(20) NOT NULL
)

```

```
CREATE TABLE FAMILIARES (
  COD_FAMI          char(6) NOT NULL,
  COD_USUA          char(6) NOT NULL,
  NOM_FAMI          varchar(25) NOT NULL,
  APEPATER_FAMI     varchar(25) NOT NULL,
  APEMATER_FAMI     varchar(25) NOT NULL,
  GRADO_PARENTES_FAMI varchar(50) NOT NULL
)
```

```
CREATE TABLE INQUILINO (
  COD_USUA          char(6) NOT NULL,
  NOM_AVAL_INQ      varchar(30) NOT NULL,
  APELL_AVAL        char(30) NOT NULL,
  HABER_BAS_INQ     int NOT NULL,
  EST_CIVIL_INQ     char(1) NULL,
  LUG_TRAB_INQ      varchar(50) NULL
)
```

```
CREATE TABLE PROPIETARIO (
  COD_USUA          char(6) NOT NULL,
  TIPO_PROP         varchar(30) NOT NULL,
  TELEFONO_PROP     char(10)
)
```

```
CREATE TABLE USUARIO (
  COD_USUA          char(6) NOT NULL,
  NOM_USUA          varchar(25) NOT NULL,
  APEPATER_USUA     varchar(25) NOT NULL,
  APEMATER_USUA     varchar(25) NOT NULL,
  FEC_NAC_USUA      datetime NULL,
  FEC_REG_USUA      datetime NULL,
  TIPO_DOC_USUA     varchar(20) NOT NULL,
  NUM_DOC_USUA      char(8) NOT NULL,
  COD_EST           char(6) NOT NULL
)
```

#### **Agregando las llaves primarias a todas las tablas**

**NONCLUSTERED:** Crea un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla.

```
ALTER TABLE CONTRATO
  ADD PRIMARY KEY NONCLUSTERED (COD_CONT)
```

```
ALTER TABLE DEPARTAMENTOS
  ADD PRIMARY KEY NONCLUSTERED (COD_EDIF, COD_DEP)
```

```
ALTER TABLE DETALLECONTRATO
  ADD PRIMARY KEY NONCLUSTERED (COD_EDIF, COD_DEP, COD_CONT)
```

```
ALTER TABLE EDIFICIOS
  ADD PRIMARY KEY NONCLUSTERED (COD_EDIF)
```

```
ALTER TABLE ESTADO
  ADD PRIMARY KEY NONCLUSTERED (COD_EST)
```

```
ALTER TABLE FAMILIARES
  ADD PRIMARY KEY NONCLUSTERED (COD_FAMI, COD_USUA)
```

```
ALTER TABLE INQUILINO
  ADD PRIMARY KEY NONCLUSTERED (COD_USUA)
```

```
ALTER TABLE PROPIETARIO  
  ADD PRIMARY KEY NONCLUSTERED (COD_USUA)
```

```
ALTER TABLE USUARIO  
  ADD PRIMARY KEY NONCLUSTERED (COD_USUA)
```

#### **Agregando llaves Foráneas y relacionando tablas**

##### **-- SE CREA LA RELACIÓN RECURSIVA.**

```
ALTER TABLE CONTRATO  
  ADD FOREIGN KEY (INQ_COD_USUA) REFERENCES INQUILINO,  
  FOREIGN KEY (COD_CONT_ORIG) REFERENCES CONTRATO
```

##### **-- SE CREAN LAS RELACIONES DEL RESTO DE TABLAS.**

```
ALTER TABLE CONTRATO  
  ADD FOREIGN KEY (PROP_COD_USUA) REFERENCES PROPIETARIO
```

```
ALTER TABLE CONTRATO  
  ADD FOREIGN KEY (COD_EST) REFERENCES ESTADO
```

```
ALTER TABLE DEPARTAMENTOS  
  ADD FOREIGN KEY (COD_EDIF) REFERENCES EDIFICIOS
```

```
ALTER TABLE DEPARTAMENTOS  
  ADD FOREIGN KEY (COD_EST) REFERENCES ESTADO
```

```
ALTER TABLE DETALLECONTRATO  
  ADD FOREIGN KEY (COD_CONT) REFERENCES CONTRATO
```

```
ALTER TABLE DETALLECONTRATO  
  ADD FOREIGN KEY (COD_EDIF, COD_DEP) REFERENCES DEPARTAMENTOS
```

```
ALTER TABLE EDIFICIOS  
  ADD FOREIGN KEY (COD_EST) REFERENCES ESTADO
```

```
ALTER TABLE FAMILIARES  
  ADD FOREIGN KEY (COD_USUA) REFERENCES INQUILINO
```

```
ALTER TABLE INQUILINO  
  ADD FOREIGN KEY (COD_USUA) REFERENCES USUARIO
```

```
ALTER TABLE PROPIETARIO  
  ADD FOREIGN KEY (COD_USUA) REFERENCES USUARIO
```

```
ALTER TABLE USUARIO  
  ADD FOREIGN KEY (COD_EST) REFERENCES ESTADO
```

## 1.4. Definición y uso de los *DEFAULTS*

Un *DEFAULT* es un valor por defecto que se puede asignar en un campo cuando el valor de este campo no es insertado en el registro.

Las definiciones *DEFAULT* se pueden utilizar de las siguientes maneras:

- 1.4.1** Generarlo cuando se crea la tabla, durante el proceso de definición de la misma.

```
CREATE TABLE PROPIETARIO (
    COD_USUA          char(6) NOT NULL,
    TIPO_PROP         varchar(30) NOT NULL
    TELEFONO_PROP     char(11) DEFAULT 'DESCONOCIDO'
)
```

- 1.4.2** Agregar a una tabla ya existente. Cada columna de una tabla puede contener una sola definición *DEFAULT*.

```
ALTER TABLE CONTRATO
    ADD DEFAULT 'DESCONOCIDO'
    FOR TELEFONO_PR
```

Pueden realizarse modificaciones o eliminaciones, si ya existen definiciones *DEFAULT*. Por ejemplo, puede modificar el valor que se inserta en una columna cuando no se escribe ningún valor.

No se puede crear definiciones *DEFAULT* para columnas definidas con:

- Una propiedad *IDENTITY*<sup>2</sup>
- Una definición *DEFAULT* o un objeto *DEFAULT* ya existentes

Cuando se agrega una definición *DEFAULT* a una columna existente en una tabla, *SQL Server 2008* aplica de forma predeterminada el nuevo valor predeterminado sólo a las nuevas filas de datos que se agregan a la tabla. Los datos existentes que se insertan mediante la definición *DEFAULT* anterior no se ven afectados. No obstante, cuando agregue una nueva columna a una tabla ya existente, puede especificar que *SQL Server* inserte en la nueva columna el valor predeterminado (especificado mediante la

<sup>2</sup> Crea una columna de identidad en una tabla. Esta propiedad se usa con las instrucciones *CREATE TABLE* y *ALTER TABLE* del lenguaje *Transact/SQL*. Su sintaxis es *IDENTITY* [(valor inicial, incremento)], donde el valor inicial es el valor entero a partir del cual empezará a contar el campo identity y el incremento es el valor que se agrega al valor de identidad de la fila anterior. Si no se coloca nada se asume que los dos valores son 1. Esta función será detallada en el punto 1.6.



definición *DEFAULT*) en vez de un valor *NULL* para las filas existentes en la tabla.

El *DEFAULT* crea un objeto denominado predeterminado. Cuando se enlaza a una columna o tipo de datos definido por el usuario, un valor predeterminado especifica un valor que debe insertarse en la columna a la que está enlazada el objeto (o en todas las columnas, que estén asociadas al tipo de datos en el caso de un tipo de datos definido por el usuario) cuando no se proporciona explícitamente un valor durante la inserción. Los valores predeterminados, que son una característica de compatibilidad con versiones anteriores, realizan algunas de las mismas funciones que las definiciones predeterminadas creadas mediante la palabra clave *DEFAULT* de las instrucciones *ALTER* o *CREATE TABLE*. Las definiciones predeterminadas son el método preferido y estándar para restringir los datos de columna, debido a que la definición se almacena con la tabla y se quita automáticamente cuando se quita ésta. Sin embargo, un valor predeterminado es útil cuando se utiliza múltiples veces en múltiples columnas.

### 1.4.3 Crear un objeto DEFAULT

Crea un objeto denominado valor predeterminado. Cuando se enlaza a un tipo de datos de columna o de alias, un valor predeterminado especifica un valor que debe insertarse en la columna a la que está enlazada el objeto (o en todas las columnas, en el caso de un tipo de datos de alias) si no se proporciona explícitamente un valor durante la inserción.

```
CREATE DEFAULT nombre_default
```

```
AS <Expresión>
```

#### 1.4.3.1 Argumentos

##### Nombre\_default

Es el nombre del valor predeterminado. Los nombres predeterminados deben cumplir las reglas de los identificadores. Especificar el nombre del propietario del valor predeterminado es opcional.

##### Expresión

Una expresión contiene sólo valores constantes (no puede contener el nombre de ninguna columna u otros objetos de base de datos). Se puede utilizar cualquier constante, función integrada o expresión matemática. Incluya las constantes de caracteres y fechas entre comillas simples ('); las constantes de moneda, de enteros y de signo flotante no necesitan comillas. Los datos binarios deben precederse de 0x y los datos de moneda deben precederse de un signo de dólar (\$). El valor predeterminado debe ser compatible con el tipo de datos de la columna.

### 1.4.3.2 Ejemplo:

#### A. Crear un valor predeterminado

Este ejemplo crea un valor predeterminado de carácter denominado 'desconocido'.

```
CREATE DEFAULT telefono AS 'desconocido'
```

#### B. Enlazar un valor predeterminado

Este ejemplo enlaza el valor predeterminado creado en el ejemplo A. El valor predeterminado sólo entra en efecto si no hay ninguna entrada en la columna Teléfono de la tabla ALUMNO. Observe que la falta de entrada no es lo mismo que un valor NULL explícito.

```
SP_BINDEFULT telefono, 'propietario.telefono_usu'
```

**Importante:** Esta característica se quitará en una versión futura de *Microsoft SQL Server*. Evite utilizar esta característica en nuevos trabajos de desarrollo y tenga previsto modificar las aplicaciones que actualmente la utilizan. En su lugar, use definiciones predeterminadas creadas con la palabra clave *DEFAULT* de *ALTER TABLE* o *CREATE TABLE* desarrollados inicialmente.

## 1.5. Definición y uso del **CHECK CONSTRAINT**

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, sólo puede hacer referencia a la columna restringida y una restricción *CHECK*, en el nivel de tabla, sólo 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.

#### Sintaxis:

```
ALTER TABLE tabla
```

```
ADD CONSTRAINT nombre_check CHECK (condición)
```

**Nombre\_check**

Es el nombre del *check constraint*

**Condición**

Es una expresión que contiene sólo valores constantes (aquí es necesario contener el nombre de alguna columna como parte de la condición).

**Ejemplo**

**A. Crear un *check constraint* que valide las descripciones de la tabla estado**

```
ALTER TABLE ESTADO ADD CONSTRAINT CHKDESC_EST  
  
CHECK ( DESC_EST IN ('OPERATIVO', 'INOPERATIVO', 'REGULAR'))
```

**B. Crear un *check constraint* desde la creación de la tabla ESTADO**

```
CREATE TABLE ESTADO (  
    COD_EST      char(6) NOT NULL,  
    DESC_EST     varchar(20) NOT NULL CHECK ( DESC_EST IN  
        ( 'OPERATIVO', 'INOPERATIVO', 'REGULAR' ) )  
)
```

## 1.6. Definición y uso del *IDENTITY*

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

**Sintaxis:**

IDENTITY [ (inicio , incremento) ]

**Argumentos:**

Inicio: es el valor que se utiliza para la primera fila cargada en la tabla.

Incremento: se trata del valor incremental que se agrega al valor de identidad de la anterior fila cargada.

Debe especificar tanto el valor de inicialización como el incremento, o bien ninguno de los dos. Si no se especifica ninguno, el valor predeterminado es (1,1).

**Ejemplo:**

**--Se activa la base de datos Ventas; luego, se verifica si existe la tabla new\_empleados para eliminarla y proceder a su creación.**

```
USE Ventas
IF OBJECT_ID ('dbo.new_empleados', 'U') IS NOT NULL
    DROP TABLE new_empleados
GO
CREATE TABLE new_empleados
(
    id_Emp int IDENTITY(1,1),
    nombreEmp varchar (20) not null,
    apePatEmp varchar(30) not null
    apeMatEmp varchar(30) not null
    sexo char(1) not null
)
```

**--Se ingresan datos a la tabla new\_empleados**

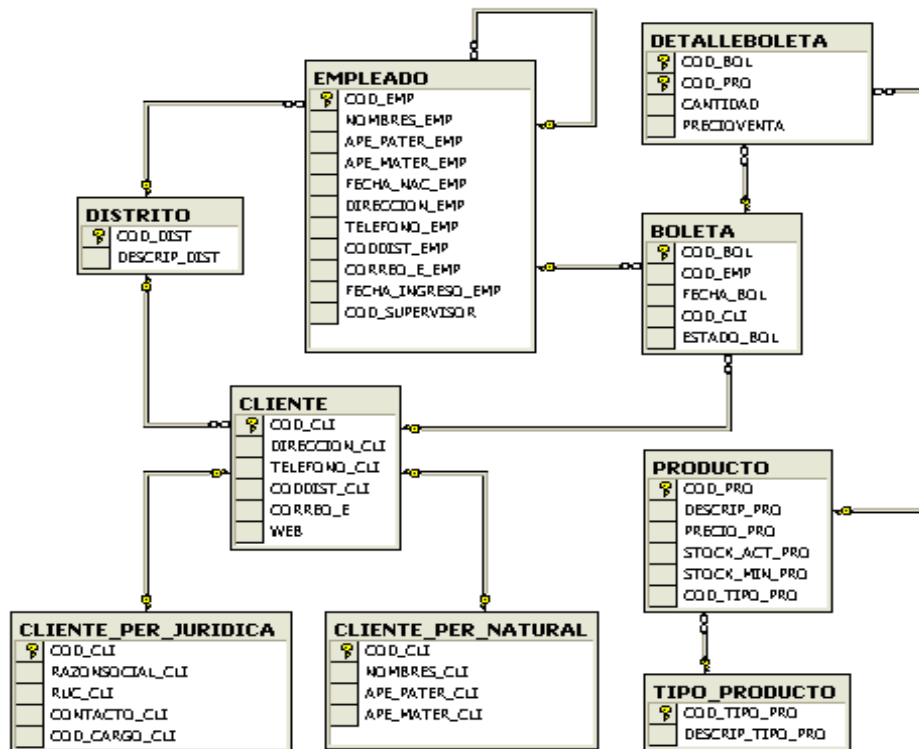
```
INSERT new_empleados ( nombreEmp, apePatEmp, apeMatEmp, sexo)
VALUES ('Lidia', 'Sanchez', 'Vargas', 'F')
```

```
INSERT new_empleados ( nombreEmp, apePatEmp, apeMatEmp, sexo)
VALUES ('Alfredo', 'Escalante', 'Sifuentes', 'M')
```

## ACTIVIDADES PROPUESTAS

### Caso: VENTAS

Se ha diseñado una base de datos para el control de las ventas realizadas en una empresa, como se detalla en el siguiente diagrama:



Se solicita:

1. Cree la base de datos Ventas y luego actívela.
2. Cree las tablas de la base de datos VENTAS, las **llaves primarias y foráneas deben ser creadas dentro de la generación de la tabla.**
3. Cree las siguientes restricciones:
  - a. Asigne el valor por defecto 'NO REGISTRA' en el campo CORREO\_E de la tabla CLIENTE.
  - b. El precio del producto, de la tabla PRODUCTO, debe ser mayor igual a cero (0) pero menor o igual a mil nuevos soles (1000).
  - c. La fecha de nacimiento del empleado debe ser menor a la fecha actual (obtener la fecha de sistema).
  - d. Por defecto establezca el valor 'TIPO01' en el campo COD\_TIPO\_PROD de la tabla Producto.
  - e. Cree la tabla CLIENTE\_BAK con los mismos campos de la tabla CLIENTE. Aplique la restricción *IDENTITY* al campo que será llave primaria e ingrese 3 registros.

## Resumen

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

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

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

Tutorial para investigar el comando *Create Table*.

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

Tutorial para investigar la propiedad Identidad (*Identity*)

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

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

**UNIDAD DE  
APRENDIZAJE****2****SEMANA****3**

## MODIFICACIÓN DEL CONTENIDO DE UNA BASE DE DATOS

---

### LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos modificarán el contenido de una base de datos mediante la aplicación de las tres sentencias del Lenguaje de Manipulación de Datos (DML) y emplean índices en la optimización del rendimiento de una base de datos haciendo uso de datos de prueba de un proceso de negocio real.

### TEMARIO

1.1 Ingreso, modificación y eliminación de datos: *INSERT*, *DELETE* y *UPDATE*

### ACTIVIDADES PROPUESTAS

- Emplean comandos *SQL* para ingresar, modificar o eliminar datos.

## 1. Sentencia *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 [(nombre_columna1, nombre_columna1,
nombre_columna n..)] VALUES (expr1, expr2, expr n...)
```

**Nombre\_Tabla** es únicamente el nombre de la tabla donde se desea ingresar los nuevos datos.

**Nombre\_Columna** es una lista opcional de nombres de campo en los que se insertarán valores en el mismo número y orden que se especificarán en la cláusula **VALUES**. Si no se especifica la lista de campos, los valores de **expr** en la cláusula **VALUES** deben ser tantos como campos tenga la tabla y en el mismo orden que se definieron al crear la tabla.

**Expr** es una lista de expresiones o valores constantes, separados por comas, para dar valor a los distintos campos del registro que se añadirá a la tabla. Las cadenas de caracteres deberán estar encerradas entre comillas.

### 1.1 Insertar un único registro

Añada un registro a la tabla USUARIO

#### A. Especificando todos los campos a ingresar.

```
INSERT INTO USUARIO ( COD_USUA, NOM_USUA, APEPATER_USUA,
APEMATER_USUA, FEC_NAC_USUA, FEC_REG_USUA, TIPO_DOC_USUA,
NUM_DOC_USUA, COD_EST)
VALUES ('U00001', 'LUIS', 'PEREZ', 'PRADO', '25/02/1989', '02/01/2009',
'DNI', '23453894', 'ACTIVO')
```

Cada sentencia **INSERT** añade un único registro a la tabla. En el ejemplo se han especificado los nueve (09) campos con sus respectivos valores. Si no se ingresara valores a un campo, este se cargará con el valor **DEFAULT** o **NULL** (siempre y cuando haya sido especificado en la estructura de la tabla). Un valor nulo –**NULL**– no significa blancos o ceros, sino que el campo nunca ha tenido un valor.

#### B. Especificando únicamente los valores de los campos.

```
INSERT INTO USUARIO VALUES ('U00001', 'LUIS', 'PEREZ', 'PRADO',
'25/02/1989', '02/01/2009', 'DNI', '23453894', 'ACTIVO')
```

El ejemplo anterior muestra que podría especificarse únicamente los valores cuando se ingresen en todos los campos de tabla. Si no se especifica la lista de



campos, los valores en la cláusula VALUES deben ser tantos como campos tenga la tabla y en el mismo orden que se definieron al crear la tabla.

Si se va a ingresar parcialmente los valores en una tabla, se debe especificar el nombre de los campos a ingresar, como en el ejemplo A.

## 1.2. Insertar Múltiples Registros

Además, existe la posibilidad de agregar múltiples registros con ayuda del comando SELECT anteriormente explicado. Veamos un ejemplo:

Se tiene la tabla USUARIOS con los siguientes datos:

COD_USUA	NOM_USUA	APEPATER_USUA	APEMATER_USUA	FEC_NAC_USUA	FEC_REG_USUA	COD_EST
U00001	LUIS	PEREZ	PRADO	25/02/1939	01/01/1970	ACTIVO
U0002	JUAN	MARTELL	TANTAS	12/01/1979	01/01/1970	ACTIVO
U0003	MIGUEL	PAEZ	CONNOR	12/108/1958	01/01/1980	INACTIVO
U0004	ANA	LOZA	LAOS	22/05/1945	01/01/1970	ACTIVO
U0005	CARLOS	ARIES	CERPA	12/01/1979	01/01/1980	INACTIVO
U0005	MARTHA	MARES	MARAVI	21/07/1977	01/01/1980	INACTIVO
U0006	MILAGROS	LLOSA	CORDOVA	05/02/1975	01/01/1970	ACTIVO
U0007	LUISA	GRADOS	DEL AGUILA	03/11/1988	01/01/1970	ACTIVO

También, tenemos una nueva tabla EJEMPLO, que ha sido creada por nosotros. Además, contiene los mismos campos o columnas de la tabla USUARIO.

### ACTIVIDAD 1.2.1

Trasladar todos los clientes de la tabla USUARIO, que tengan COD\_EST='ACTIVO', a nuestra nueva tabla EJEMPLO. La respuesta sería la siguiente:

```
INSERT INTO EJEMPLO (COD_USUA, NOM_USUA, APEPATER_USUA,
APEMATER_USUA, FEC_NAC_USUA, FEC_REG_USUA, COD_EST)
```

```

SELECT COD_USUA, NOM_USUA, APEPATER_USUA,
APEMATER_USUA, FEC_NAC_USUA, FEC_REG_USUA,
COD_EST
FROM USUARIO
WHERE COD_EST='ACTIVO'
```

Como se puede observar, es posible combinar el comando *INSERT* con las consultas de Selección para agregar datos específicos sin tener la necesidad de realizarlo uno por uno.

## 2. Sentencia *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 nombre_columna1 = expr1, nombre_columna2 = expr2, .....
[WHERE { condición }]
```

**Nombre\_Tabla** es únicamente el nombre de la tabla donde se desea ingresar los nuevos datos.

**Nombre\_columna** es el nombre de columna o campo cuyo valor se desea cambiar. En una misma sentencia *UPDATE* pueden actualizarse varios campos de cada registro de la tabla.

**Expr** es el nuevo valor que se desea asignar al campo que le precede. La expresión puede ser un valor constante o una subconsulta. Las cadenas de caracteres deberán estar encerradas entre comillas. Las subconsultas entre paréntesis.

La cláusula *WHERE* sigue el mismo formato que la vista en la sentencia *SELECT* y determina qué registros se modificarán.

**ACTIVIDAD 2.1:** se le solicita fijar un precio único para todos los DEPARTAMENTOS. No se utiliza la cláusula *WHERE*.

```
UPDATE DEPARTAMENTOS
SET PRECIO_ALQxMES_DEP = 2000
```

**ACTIVIDAD 2.2:** subir el precio de alquiler por mes de un departamento de la tabla de DEPARTAMENTOS en un 10% de aquellos que tengan más de 100 metros de área construida:

```
UPDATE DEPARTAMENTOS
SET PRECIO_ALQxMES_DEP = PRECIO_ALQxMES_DEP * 1.1
WHERE AREA_TOTAL_DEP > 100
```

**ACTIVIDAD 2.3:** se le solicita fijar la fecha de registro de un usuario (en la tabla USUARIO) de todos aquellos usuarios que tengan la fecha de registro vacía (nula) a la fecha actual, se escribiría:

```
UPDATE USUARIO
SET FEC_REG_USUA = TODAY()
WHERE FEC_REG_USUA IS NULL
```

**ACTIVIDAD 2.4:** se le solicita asignar precio a todos los libros que no lo tienen. Ese precio será el resultante de calcular la media entre los libros que sí lo tenían. El ejemplo utilizará una subconsulta:

```
UPDATE DEPARTAMENTOS
SET PRECIO_ALQXMES_DEP =
  ( SELECT AVG(PRECIO_ALQXMES_DEP)
    FROM DEPARTAMENTOS
    WHERE PRECIO_ALQXMES_DEP IS NOT NULL )
WHERE PRECIO IS NULL
```

### 3. DELETE

*DELETE* es especialmente útil cuando se desea eliminar varios registros. En una instrucción *DELETE* con múltiples tablas debe incluir el nombre de tabla (Tabla.\*). Si especifica más de una tabla para eliminar registros, todas deben tener una relación de muchos a uno. Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Se puede utilizar *DELETE* para eliminar registros de una única tabla o desde varios lados de una relación uno a muchos. Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación. Por ejemplo, en la base de datos VENTAS, la relación entre las tablas Clientes y Pedidos, la tabla Pedidos es la parte de muchos, por lo que las operaciones en cascada sólo afectarán a la tabla Pedidos. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si desea eliminar valores en un campo especificado, crea una consulta de actualización que cambie los valores a *Null*.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus

datos en todo momento. Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

El formato de la sentencia es:

```
DELETE FROM Nombre_Tabla  
  
[WHERE { condición }]
```

**Nombre\_Tabla** es únicamente el nombre de la tabla donde se desea borrar los datos.

La cláusula **WHERE** sigue el mismo formato que la vista en la sentencia **SELECT** y determina qué registros se borrarán.

Cada sentencia **DELETE** borra los registros que cumplen la condición impuesta o todos si no se indica cláusula **WHERE**.

**ACTIVIDAD 3.1:** con esta actividad, se borrarían todos los registros de la tabla FAMILIARES cuyo grado de parentesco sea "ESPOSA".









```
DELETE FROM FAMILIARES  
  
WHERE GRADO_PARENTES_FAMI='ESPOSA'
```

## ACTIVIDADES A DESARROLLAR EN CLASE

Usando **TRANSACT/SQL**, realice las siguientes actividades en la base de datos **VENTAS**:

- 1 Inserte tres (3) registros a todas las tablas.
- 2 Actualice la descripción del distrito con código 'L01' por 'LIMA 01'.
- 3 Inserte el empleado con código 'EMP1010' , asegúrese que resida en el distrito con código 'L01'.
- 4 Elimine el distrito con descripción 'LIMA 01'. ¿Es factible realizar esta actividad? Explique.
- 5 Cree una tabla llamada Copia\_Empleado con la misma estructura de la tabla Empleado.
- 6 Inserte a la tabla Copia\_Empleado, todos los empleados de la tabla EMPLEADO, cuyo año de ingreso se encuentre entre los años 2006 y 2008.
- 7 Actualice el campo CORREO\_E de la tabla CLIENTES de todos los clientes que no tengan correo electrónico (valor 'NO REGISTRA').
- 8 Actualice el campo fecha de ingreso y el teléfono del empleado con código 'EMP1001'. La fecha de ingreso actualícela por la fecha actual y el campo teléfono por el número '2471111'.

## Para recordar

-  La sentencia *INSERT* de una fila añade una fila de datos a una tabla. Los valores para la nueva fila se especifican en la sentencia como constantes. La sentencia *INSERT* multifila añade cero o más filas a una tabla. Los valores para las nuevas filas provienen de una consulta, especificada como parte de la sentencia *INSERT*.
-  La sentencia *UPDATE* modifica los valores de una o más columna en cero o más filas de una tabla. Las filas a actualizar son especificadas mediante una condición de búsqueda.
-  La sentencia *DELETE* suprime cero o más filas de datos de una tabla. Las filas a suprimir son especificadas mediante una condición de búsqueda.
-  A diferencia de la sentencia *SELECT*, que puede operar sobre múltiples tablas, las sentencias *INSERT*, *DELETE* y *UPDATE* funcionan solamente sobre una única tabla cada vez.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:
  -  <http://technet.microsoft.com/es-es/library/ms174335.aspx>  
Tutorial para el uso del comando *Insert*
  -  <http://technet.microsoft.com/es-es/library/ms177523.aspx>  
Tutorial para el uso del comando *Update*
  -  <http://technet.microsoft.com/es-es/library/ms189835.aspx>  
Tutorial para el uso del comando *Delete*

UNIDAD DE  
APRENDIZAJE**2**

SEMANA

**4**

## MODIFICACIÓN DEL CONTENIDO DE UNA BASE DE DATOS

---

### LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos modificarán el contenido de una base de datos mediante la aplicación de las tres sentencias del Lenguaje de Manipulación de Datos (DML) y emplean índices en la optimización del rendimiento de una base de datos haciendo uso de datos de prueba de un proceso de negocio real.

### TEMARIO

#### 1.1 Creación y mantenimiento de índices

### ACTIVIDADES PROPUESTAS

- Determinan la necesidad de empleo de índices en una base de datos.
- Emplean los comandos *CREATE* y *DROP* para el uso de índices.

## 1. INTRODUCCIÓN

Se puede decir que un índice sobre un atributo determinado en este caso "A" de una relación es una estructura de datos que permite encontrar rápidamente las tuplas que poseen un valor fijo en ese atributo.

Los índices generalmente facilitan las consultas en las que el atributo A se compara con una constante ( $A=3$ ) o incluso ( $A<3$ ).

## 2. CREACIÓN DE ÍNDICES

Un índice adecuadamente situado en una tabla, ayudará a la base de datos a recuperar más rápidamente los datos, sobre todo cuando las tablas son muy grandes. Los índices se asocian a una única tabla. Podrá haber un índice primario (índice por el cual se ordena la tabla de datos) y múltiples índices secundarios.

Para la creación de índices, se utiliza la siguiente sintaxis:

```
CREATE [ UNIQUE ] INDEX índice
ON tabla (campo [ASC | DESC][, campo [ASC | DESC],...])
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

donde:

Parte	Descripción
índice	Es el nombre del índice a crear
tabla	Es el nombre de una tabla existente en la que se creará el índice
campo	Es el nombre del campo o lista de campos que constituyen el índice
ASC/DESC	Indica el orden de los valores de los campos <i>ASC</i> indica un orden ascendente (valor predeterminado) y <i>DESC</i> un orden descendente
UNIQUE	Indica que el índice no puede contener valores duplicados
DISALLOW NULL	Prohíbe valores nulos en el índice
IGNORE NULL	Excluye del índice los valores nulos incluidos en los campos que lo componen
PRIMARY	Asigna al índice la categoría de clave principal, en cada tabla sólo puede existir un único índice que sea "Clave Principal". Si un índice es clave principal implica que no puede contener valores nulos ni duplicados



**Ejemplo:**

CREATE INDEX	--comando de creación de índice
MI_PRIMER_INDICE	--nombre de índice
ON	--ubicación
EDIFICIOS	--nombre de la tabla
(COD_EDIF , NOM_EDIF)	--atributos que van a conformar el índice

Del ejemplo propuesto, se desprende que se va a crear un índice con el nombre “MI\_PRIMER\_INDICE”, el cual va a estar ubicado en la tabla “EDIFICIOS” y va a comprender los atributos “COD\_EDIF” y “NOM\_EDIF”.

Se pueden utilizar también los siguientes comandos:

**2.1. UNIQUE:**

Indica que se trata de un índice único, es decir, no admite más de una tupla con el mismo valor en los atributos que forman el índice. Cuando se crea una tabla con clave primaria, automáticamente se crea un índice único.

**ON tabla (atrib1 [ASC | DESC][,atrib2 [ASC | DESC]]...):** Es posible crear índices compuestos, que se forman con más de una columna. Se emplea en caso de columnas que siempre se consultan juntas.

**Ejemplo:**

CREATE UNIQUE INDEX	--comando de creación de índice único
MI_SEGUNDO_INDICE	--nombre de índice
ON	--ubicación
EDIFICIOS	--nombre de la tabla
(COD_EDIF , NOM_EDIF)	--atributos que van a conformar el índice

**3. IMPORTANCIA DE LOS ÍNDICES**

La selección de índices requiere que el diseñador de la base de datos haga un compromiso:

- La existencia de un índice en un atributo agiliza enormemente las consultas en que se especifica un valor de él.
- En cambio, todos los índices contruidos para un atributo de alguna tabla hacen que inserciones, modificaciones y eliminaciones sean más complejas y lentas

A la hora de decidirse a crear los índices, se debe estimar cuál será la combinación normal de consultas y otras operaciones sobre la base de datos. Si una tabla se consulta con mucha mayor frecuencia de lo que se modifica, conviene utilizar índices sobre los atributos utilizados como filtro de consulta más a menudo.

Si las modificaciones son la operación predominante, hay que ser más cauto respecto a la creación de índices.

#### 4. ELIMINACIÓN DE ÍNDICES

Para eliminar un índice, se utiliza el comando ***DROP INDEX***

DROP INDEX 'table.index | view.index' [ ,...n ]

##### Ejemplo

DROP INDEX EDIFICIOS.MI\_PRIMER\_INDICE

#### 5. USO DE LOS ÍNDICES

##### a) Razones para crear un índice

Acelerar el acceso a datos

Fuerzan la unicidad de las filas

##### b) Razones para no crear un índice

Consumen espacio en disco

Generan costos de procesamiento

#### 6. TIPOS DE ÍNDICES

##### a) Índices Agrupados

- Cada tabla sólo puede tener un índice agrupado
- El orden físico de las filas de la tabla y el orden de las filas en el índice son el mismo
- La unicidad de los valores de clave se mantiene explícitamente o implícitamente

##### b) Índices No Agrupados

- Los índices no agrupados son los predeterminados de *SQL Server*
- Los índices no agrupados existentes se vuelven a generar automáticamente:
  - Se quita un índice agrupado existente
  - Se crea un índice agrupado
  - Se utiliza la opción *DROP\_EXISTING* para cambiar las columnas que definen el índice agrupado





## ACTIVIDADES A DESARROLLAR EN CLASE

Usando **TRANSACT/SQL**, realice las siguientes actividades en la base de datos **VENTAS**:

Identifique, cuáles son las tablas de donde es posible que se realicen la mayor cantidad de consultas, y luego cree los siguientes índices:

1. Para un rápido acceso a la tabla **CLIENTE**, cree el índice para la columna **APE\_PATER\_EMP**.
2. Para una rápida identificación de un producto cree el índice para la columna **DESCRIP\_PRO**.
3. Asegúrese de la unicidad de los nombres y apellidos de los empleados.

## Resumen

-  Las sentencias *CREATE INDEX* y *DROP INDEX* definen índices, los cuales aceleran las consultas de la base de datos notablemente, pero a cambio ellos añaden recargo a las actualizaciones.
-  *UNIQUE*, indica que se trata de un índice único, es decir, no admite más de una tupla con el mismo valor en los atributos que forman el índice. Cuando se crea una tabla con clave primaria, automáticamente se crea un índice único.
-  A la hora de decidirse a crear los índices, se debe estimar cuál será la combinación normal de consultas y otras operaciones sobre la base de datos. Si una tabla se consulta con mucha mayor frecuencia de lo que se modifica, conviene utilizar índices sobre los atributos utilizados como filtro de consulta más a menudo.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 <http://technet.microsoft.com/es-es/library/ms188783.aspx>

Tutorial para el uso de índices

**UNIDAD DE  
APRENDIZAJE****3****SEMANA****5**

## IMPLEMENTACIÓN DE CONSULTAS

---

### LOGROS DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos recuperan información de una base de datos con el empleo de comandos *SQL* haciendo uso de múltiples formas con el uso de algunas condiciones de comparación y funciones para el manejo de campos tipo fecha. Obtienen registros originados por la selección de uno o varios grupos haciendo uso de las funciones agrupamiento y columna procedentes de dos o más tablas.

### TEMARIO

1.1. Introducción a las consultas

1.2. Uso del *SELECT*, *FROM*, *WHERE*, *ORDER BY*, *INNER JOIN*

1.3. Funciones para el manejo de fechas: *DAY()*, *MONTH()*, *YEAR()*, *DATEDIFF()* y *DATEPART()*

### ACTIVIDADES PROPUESTAS

- Generan consultas sencillas a la base de datos con el comando *SELECT*.

## 1. CONSULTAS DE SELECCIÓN (*SELECT*)

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.

## 2. CONSULTAS SENCILLAS

La sintaxis básica de una consulta de selección es la siguiente:

```
SELECT Campos FROM Tabla
```

En donde campos es la lista de campos que se deseen recuperar y tabla es el origen de los mismos, por ejemplo:

```
SELECT *  
FROM CONTRATO
```

Esta consulta devuelve un *recordset* con todos los campos de la tabla "CONTRATO".

```
SELECT COD_CONT, PRO_COD_USUA  
FROM CONTRATO
```

Esta consulta devuelve un *recordset* con los campos COD\_CONT y PRO\_COD\_USUA de la tabla "CONTRATO".

## 3. ORDENAR LOS REGISTROS

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula *ORDER BY* (Lista de Campos). En donde Lista de campos representa los campos a ordenar.

Ejemplo:

```
SELECT *  
FROM USUARIO  
ORDER BY APEPATER_USUA
```

Esta consulta devuelve todos los campos de la tabla "USUARIO" ordenados por el campo apellido paterno del usuario.

Se pueden ordenar los registros por más de un campo, como por ejemplo:

```
SELECT APEPATER_FAMI, APEMATER_FAMI, NOM_FAMI
FROM FAMILIARES
ORDER BY APEPATER_FAMI, APEMATER_FAMI
```

Incluso, se puede especificar el orden de los registros: ascendente mediante la cláusula (*ASC* -se toma este valor por defecto) ó descendente (*DESC*)

```
SELECT COD_CONT, FEC_INI_ALQ, FEC_FIN_ALQ
FROM DETALLECONTRATO
ORDER BY COD_CONT ASC, FEC_INI_ALQ DESC
```

#### 4. CONSULTAS CON PREDICADO

El predicado se incluye entre la cláusula y el primer nombre del campo a recuperar, los posibles predicados son los siguientes:

Predicado	Descripción
<i>ALL</i>	Devuelve todos los campos de la tabla
<i>TOP</i>	Devuelve un determinado número de registros de la tabla
<i>DISTINCT</i>	Omite los registros cuyos campos seleccionados coincidan totalmente
<i>DISTINCTROW</i>	Omite los registros duplicados que están borrados en la totalidad del registro y no sólo en los campos seleccionados

##### 6.1. *ALL*

Si no se incluye ninguno de los predicados se asume *ALL*. El motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción *SQL*. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene. Por ello, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL * FROM FAMILIARES

SELECT * FROM FAMILIARES
```

##### 6.2. *TOP*

Devuelve un cierto número de registros que entran al principio o al final de un rango especificado por una cláusula *ORDER BY*. Supongamos que queremos recuperar los nombres de los cinco (05) primeros *recordset* de la tabla familiares ordenados por apellido paterno en forma descendente:

```
SELECT TOP 5    APEPATER_FAMI,
                APEMATER_FAMI,
                NOM_FAMI
FROM FAMILIARES
ORDER BY
APEPATER_FAMI DESC
```

Si no se incluye la cláusula *ORDER BY*, la consulta devolverá un conjunto arbitrario de cinco (05) registros de la tabla familiares. El predicado *TOP* no elige entre valores iguales.

### 6.3. TOP PERCENT

Se puede utilizar la palabra reservada *PERCENT* para devolver un cierto porcentaje de registros que están al principio o al final de un rango especificado por la cláusula *ORDER BY*. Supongamos que en lugar de los cinco (05) primeros familiares, deseamos el veinte por ciento (20%) por ciento de la tabla de familiares:

```
SELECT TOP 20 PERCENT    APEPATER_FAMI,
                        APEMATER_FAMI,
                        NOM_FAMI
FROM FAMILIARES
ORDER BY
APEPATER_FAMI DESC
```

El valor que va a continuación de *TOP* debe ser un *Integer* sin signo. “*TOP*” no afecta a la posible actualización de la consulta.

### 6.4. DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción *SELECT* se incluyan en la consulta deben ser únicos.

Por ejemplo, varios edificios listados en la tabla edificios pueden tener el mismo nombre. Si dos registros contienen Los Cipreses en el campo nombre, la siguiente instrucción *SQL* devuelve un único registro:

```
SELECT DISTINCT NOM_EDIF
FROM EDIFICIOS
```

En otras palabras, el predicado *DISTINCT* devuelve aquellos registros cuyos campos indicados en la cláusula *SELECT* posean un contenido diferente. El resultado de una consulta que utiliza *DISTINCT* no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.



### 6.5. **DISTINCTROW**

Devuelve los registros diferentes de una tabla. A diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo, independientemente de los campos indicados en la cláusula *SELECT*.

```
SELECT DISTINCTROW NOM_FAMI, APEPATER_FAMI, APEMATER_FAMI  
FROM FAMILIARES
```

Si la tabla empleados contiene dos registros: Antonio López y Marta López el ejemplo del predicado *DISTINCT* devuelve un único registro con el valor López en el campo Apellido ya que busca no duplicados en dicho campo. Este último ejemplo devuelve dos registros con el valor López en el apellido ya que se buscan no duplicados en el registro completo.

## 7. **ALIAS**

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada *AS* que se encarga de asignar el nombre que deseamos a la columna deseada. Tomado como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse apellido (igual que el campo devuelto) se llame Empleado. En este caso, procederíamos de la siguiente forma:

```
SELECT DISTINCTROW NOM_FAMI  
AS nombre_del_Familiar  
FROM FAMILIARES
```

## 8. **CONSULTAS DE UNIÓN INTERNAS (INNER)**

Las vinculaciones entre tablas se realizan mediante la cláusula *INNER* que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es de la siguiente forma:

```
SELECT campos  
FROM tb1  
    INNER JOIN tb2  
    ON tb1.campo1 comp tb2.campo2
```

En donde:

**tb1, tb2**

Son los nombres de las tablas desde las que se combinan los registros.

**campo1, campo2**

Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

**comp**

Es cualquier operador de comparación relacional: =, <, >, <=, >=, o <>.

Se puede utilizar una operación *INNER JOIN* en cualquier cláusula *FROM*. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones aquí son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar *INNER JOIN* con las tablas *DEPARTAMENTOS* y *EMPLEADOS* para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los *DEPARTAMENTOS* (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea *LEFT JOIN* o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso *RIGHT JOIN*.

Si se intenta combinar campos que contengan datos Memo u Objeto *OLE*<sup>3</sup>, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad *Size* de su objeto *Field* está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas *EDIFICIOS* y *DEPARTAMENTOS* basándose en el campo "COD\_EDIF.":

```
SELECT EDIFICIOS.NOM_EDIF,
        DEPARTAMENTOS.COD_DEP,
        DEPARTAMENTOS.AREA_TOTAL_DEP
FROM   EDIFICIOS
        INNER JOIN DEPARTAMENTOS
        ON EDIFICIOS.COD_EDIF = DEPARTAMENTOS.COD_EDIF
```

**Usando alias :**

```
SELECT E.NOM_EDIF as 'Nombre Edificio', D.COD_DEP as 'Código dep.',
        D.AREA_TOTAL_DEP as 'Área total dep.'
FROM   EDIFICIOS AS E
        INNER JOIN DEPARTAMENTOS AS D
        ON E.COD_EDIF = D.COD_EDIF
```

En el ejemplo anterior, COD\_EDIF es el campo combinado, pero no está incluido en la salida de la consulta ya que no se considera en la instrucción *SELECT*. Para incluir el campo combinado, se debe agregar el nombre del campo en la

<sup>3</sup> Un objeto OLE (*Object Linking and Embedding*) significa el estándar de vinculación e incrustación de objetos. OLE es un entorno unificado de servicios basados en objetos con la capacidad de personalizar esos servicios y de ampliar la arquitectura a través de servicios personalizados, con la finalidad global de permitir una integración entre los componentes. OLE proporciona un estándar consistente que permite a los objetos, aplicaciones y componentes *ActiveX*, comunicarse entre sí con la finalidad de usar el código de los demás. Los objetos no necesitan conocer por anticipado en qué objetos se van a comunicar, ni su código necesita estar escrito en el mismo lenguaje.

instrucción *SELECT*, en este caso *EDIFICIOS.COD\_EDIF* o *E.COD\_EDIF*, si usa alias.

También, se pueden enlazar varias cláusulas *ON* en una instrucción *JOIN*, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1
    INNER JOIN tabla2
    ON tb1.campo1 comp tb2.campo1
    and ON tb1.campo2 comp tb2.campo2)]
```

También, puede anidar instrucciones *JOIN* utilizando la siguiente sintaxis:

```
SELECT campos
FROM tabla1
    INNER JOIN tabla2
    ON tb1.campo1 comp tb2.campo2
    INNER JOIN tabla3
    ON tb2.campo2 comp tb3.campo3)]
```

Un *LEFT JOIN* o un *RIGHT JOIN* puede anidarse dentro de un *INNER JOIN*, pero un *INNER JOIN* no puede anidarse dentro de un *LEFT JOIN* o un *RIGHT JOIN*.

**Ejemplo :**

```
SELECT USUARIO.NOM_USUA, CONTRATO.FEC_FIRMA
FROM PROPIETARIO
INNER JOIN CONTRATO
    ON PROPIETARIO.COD_USUA = CONTRATO.PROP_COD_USUA
INNER JOIN USUARIO
    ON USUARIO.COD_USUA = PROPIETARIO.COD_USUA
```

Si se emplea la cláusula *INNER* en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de *INNER JOIN* que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave *INNER*, estas cláusulas son *LEFT* y *RIGHT*. *LEFT* toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. *RIGHT*

realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

## 9. EJEMPLOS DE CONSULTAS USANDO FUNCIONES DE FECHA

<i>Función</i>	<i>Descripción</i>
<b>DAY</b>	Devuelve un entero que representa el día (día del mes) de la fecha especificada
<b>MONTH</b>	Devuelve un entero que representa el mes de la fecha especificada
<b>YEAR</b>	Devuelve un entero que representa la parte del año de la fecha especificada
<b>DATEDIFF</b>	Devuelve el tiempo transcurrido desde una fecha inicial hasta una fecha final
<b>DATEPART</b>	Devuelve un entero que representa el parámetro <i>datepart</i> especificado del parámetro <i>date</i> especificado

### 9.1 Muestre el código, el apellido y la edad de los usuarios

```
SELECT COD_USUA, APEPATER_USUA,
       DATEDIFF(YY,FEC_NAC_USUA,GETDATE()) 'EDAD DEL USUARIO'
FROM USUARIO
```

### 9.2 Muestre el código y los meses transcurridos desde la firma del contrato

```
SELECT COD_CONT, DATEDIFF(MM,FEC_FIRMA,GETDATE()) AS 'MESES
TRANSCURRIDOS'
FROM CONTRATO
```

### 9.3 Muestre el código del contrato, el día, mes y año en que se firmó el contrato

--SOLUCION 1

```
SELECT COD_CONT, FEC_FIRMA,
       DATEPART(DD,FEC_FIRMA) AS DIA ,
       DATEPART(MM,FEC_FIRMA) AS MES ,
       DATEPART(YY,FEC_FIRMA) AS AÑO
FROM CONTRATO
```

--SOLUCION 2

```
SELECT COD_CONT, FEC_FIRMA, DAY(FEC_FIRMA) AS DIA,
       MONTH(FEC_FIRMA) AS MES, YEAR(FEC_FIRMA) AS AÑO
FROM CONTRATO
```

### 9.4 Muestre el código del contrato, el día, mes y año en que se firmó el contrato

```
SELECT COD_CONT,
       DATEPART(DW,FEC_FIRMA) AS 'DIA DE LA SEMANA (1-7)',
       DATEPART(DD,FEC_FIRMA) AS 'DIA DEL CONTRATO (1-31)',
       DAY(FEC_FIRMA) AS 'DIA DEL CONTRATO (1-31)',
       DATENAME(DW,FEC_FIRMA) AS 'NOMBRE DEL DIA (MONDAY-SUNDAY)'
FROM CONTRATO
```






## ACTIVIDADES A DESARROLLAR EN CLASE

Usando **TRANSACT/SQL**, realice las siguientes actividades:

De la base de datos **VENTAS**

1. Muestre los 5 primeros productos más caros.
2. Muestre **todos los datos** del 20% de los clientes del tipo persona jurídica registrada, ordenada por su razón social de manera ascendente.
3. Muestre la descripción de todos los empleados registrados y la descripción del distrito donde residen.
4. Muestre la descripción de todos los clientes que hayan nacido entre los meses de febrero y marzo.
5. Muestre los datos de las boletas con sus detalles emitidos entre los años 2007 y 2009.
6. Muestre los datos de todos los clientes que tienen correo electrónico, ordenados por el apellido paterno del cliente que residen en los distritos de Lince, San Isidro y Jesús María.

## Resumen

-  La instrucción Select se utiliza para expresar una consulta SQL. Toda sentencia Select produce una tabla de resultados que contiene una o más columnas y cero o más filas.
-  La cláusula From especifica la (s) tabla (s) que contiene (n) los datos a recuperar de una consulta.
-  La cláusula Where selecciona las filas a incluir en los resultados aplicando una condición de búsqueda a las filas de la base de datos.
-  La cláusula Order by especifica que los resultados de la consulta deben ser ordenados en sentido ascendente o descendente, basándose en los valores de una o más columnas.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 [http://www.aulaclie.es/sql/t\\_3\\_4.htm](http://www.aulaclie.es/sql/t_3_4.htm)

Tutorial para el uso de inner join

 <http://technet.microsoft.com/es-es/library/ms174420.aspx>

Tutorial para el uso de la función datepart()

 <http://technet.microsoft.com/es-es/library/ms189794.aspx>

Tutorial para el uso de la función datediff()

**UNIDAD DE  
APRENDIZAJE****3****SEMANA****6**

## CONSULTAS CONDICIONALES

---

### LOGROS DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos recuperan información de una base de datos con el empleo de comandos *SQL* haciendo uso de múltiples formas con el uso de algunas condiciones de comparación y funciones para el manejo de campos tipo fecha. Obtienen registros originados por la selección de uno o varios grupos haciendo uso de las funciones agrupamiento y columna procedentes de dos o más tablas.

### TEMARIO

- 1.1 Manipulación de consultas condicionales
- 1.2 Empleo condicionales *IF EXISTS*, *AND*, y operadores lógicos *>*, *<*, *=*, *<>*, *BETWEEN*, *IN*, *OR*, *NOT*, *DISTINCT*, *LIKE*.

### ACTIVIDADES PROPUESTAS

- Emplean operadores lógicos y condicionales en las consultas *SQL* a la base de datos.

## 1. CONSULTAS CON EL COMANDO *EXISTS*

El predicado *EXISTS* (con la palabra reservada *NOT* opcional) se utiliza en comparaciones de verdadero o falso para determinar si la subconsulta devuelve algún registro. Supongamos que se desea determinar el área total de un departamento en un edificio:

```
SELECT DEPARTAMENTOS.AREA_TOTAL_DEP
FROM   DEPARTAMENTOS
WHERE  EXISTS
      ( SELECT *
        FROM EDIFICIOS
        WHERE EDIFICIOS.COD_DEP = DEPARTAMENTOS.COD_DEP)
```

## 2. CRITERIOS DE SELECCIÓN

Anteriormente se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla. A lo largo de esta sesión se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas.

Antes de comenzar el desarrollo de esta sesión hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda es que no se puede establecer condiciones de búsqueda en los campos memo y la tercera hace referencia a las fechas. Las fechas se deben escribir siempre en formato mm-dd-aa en donde mm representa el mes, dd el día y aa el año. Se debe tomar en cuenta los separadores – (guión), no siendo útil la separación habitual de la barra (/), por lo que se debe utilizar el guión (-) y además la fecha debe ir encerrada entre almohadillas (#).

Por ejemplo si deseamos referirnos al día 3 de Septiembre de 1995 deberemos hacerlo de la siguiente forma; #09-03-95# ó #9-3-95#.

## 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> <b>operador</b> <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:



<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso
Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

El último operador denominado *Is* se emplea para comparar dos variables de tipo objeto <Objeto1> *Is* <Objeto2>. Este operador devuelve verdad si los dos objetos son iguales.

```

SELECT NOM_AVAL_INQ, EST_CIVIL_INQ
FROM INQUILINO
WHERE  HABER_BAS_INQ>500
      AND
      HABER_BAS_INQ<1000

```

```

SELECT  NOM_AVAL_INQ, EST_CIVIL_INQ
FROM    INQUILINO
WHERE   ( HABER_BAS_INQ>=500 AND HABER_BAS_INQ<=1000 )
        OR EST_CIVIL_INQ='C' --"casado"

----
SELECT  *
FROM    DEPARTAMENTOS
WHERE   NOT COD_DEP ="301"

----
SELECT  COD_DEP,  NUM_AMB_DEP, AREA_TOTAL_DEP
FROM    DEPARTAMENTOS
WHERE   (NUM_AMB_DEP>=1 AND NUM_AMB_DEP<=3)
        OR (AREA_TOTAL_DEP>100 AND AREA_TOTAL_DEP<150)

```

#### 4. INTERVALOS DE VALORES - *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 [Not] **Between** valor1 And valor2  
(la condición **Not** es opcional)

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.

```

SELECT NOM_AVAL_INQ, EST_CIVIL_INQ
FROM INQUILINO
WHERE  HABER_BAS_INQ Between 500 AND 1000

--Equivalente usando el operador lógico AND
SELECT NOM_AVAL_INQ, EST_CIVIL_INQ
FROM INQUILINO
WHERE  HABER_BAS_INQ >= 500 AND HABER_BAS_INQ <= 1000

```

*(Devuelve los "haberés básicos" entre 500 y 1000;  
se considera el intervalo)*

## 5. EL 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 una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador *Like* para encontrar valores en los campos que coincidan con el modelo especificado. De acuerdo con el modelo empleado, se puede especificar un valor completo (Ana María) o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (*Like* 'An%').

El operador *Like* se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena.

Por ejemplo, si introduce *Like* 'C%' en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

Like 'P[A-F]###'

Este ejemplo devuelve los campos cuyo contenido empieza con una letra de la A a la D seguidas de cualquier cadena.

Like '[A-D]%'

En la tabla siguiente se muestra cómo utilizar el operador *Like* para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a%a'	'aa', 'aBa', 'aBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab%'	'abcdefg', 'abc', 'ab'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

## 6. EL OPERADOR *IN*

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es la siguiente:

expresión [Not] In(valor1, valor2, . . .)

**Ejemplo :**

```
SELECT    NOM_FAMI, APEPATER_FAMI, APEMATER_FAMI
FROM      FAMILIARES
WHERE     APEPATER_FAMI IN ("PEREZ","CABANILLAS","SANCHEZ")
```

## 7. LA CLÁUSULA *WHERE*

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 expuestas en los puntos anteriores de la presente sesión. 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*.

```
SELECT    COD_DEP, AREA_TOTAL_DEP, AREA_CONSTRUIDA_DEP
FROM      DEPARTAMENTOS
WHERE     NUM_AMB_DEP>3

--

SELECT    COD_DEP, AREA_TOTAL_DEP, AREA_CONSTRUIDA_DEP
FROM      DEPARTAMENTOS
WHERE     AREA_CONSTRUIDA_DEP>=150 AND NUM_AMB_DEP>3
```

## ACTIVIDADES A DESARROLLAR EN CLASE

Usando **TRANSACT/SQL**, realice las siguientes actividades en la base de datos **VENTAS**:

1. Muestre los datos de los productos cuyo precio se encuentra entre 1 y 18.5, cuya descripción del producto empiece con las letras A a la D.
2. Muestre **todos los datos** de los clientes del tipo persona natural registrados cuyo teléfono empiece con los dígitos '247' y su correo termine con @hotmail.com, ordenados por su apellido paterno de manera ascendente y en coincidencias por su apellido materno de manera descendente.
3. Muestre la descripción de todos los empleados registrados y la descripción del distrito donde residen pero únicamente muestre a los que residen en los distritos de 'Magdalena', 'San Isidro' y 'Miraflores'.
4. Muestre la descripción de todos los clientes que hayan nacido entre el mes de enero y abril. Use la cláusula BETWEEN.
5. Muestre los datos de las boletas con sus detalles emitidos entre los años 2007 y 2008. Use la cláusula BETWEEN.
6. Muestre las boletas emitidas por los vendedores con apellido paterno 'ALVA', 'GOMEZ' o 'PEREZ'. Ordenado por fecha de emisión de las boletas, en coincidencias ordenarlo por apellido del vendedor.

## Resumen

Se debe recordar siempre lo siguiente:

- 📖 Cada vez que haga una consulta haciendo referencia a un valor numérico, la comparación será escrita de la siguiente manera:

WHERE CAMPO (=, >, <, etc) VALOR

- 📖 Cada vez que haga una consulta haciendo referencia a un valor no numérico, la comparación será escrita de la siguiente manera:

WHERE CAMPO (=, >, <, etc) ' VALOR '

Además, debe indicar los apostrofes al inicio y al final del texto a comparar.

- 📖 Es posible hacer una comparación a números guardados en formato *CHAR*, *VARCHAR*, (es decir, texto). Lo único que debemos tomar en cuenta es que cada número es representado por valores *ASCII* y en realidad es el valor de carácter *ASCII* el que prima para realizar un ordenamiento. Por ejemplo:


Campo X (número guardados en formato numérico)	Campo Y (número guardados en formato texto)
2	2
25	25
10	10
1	1
7	7


Si desea realizar un ordenamiento de menor a mayor, el resultado sería de la siguiente manera:

Campo X (número guardados en formato numérico)	Campo Y (número guardados en formato texto)
1	1
2	10
7	2
10	25
25	7

El código de los caracteres *ASCII* para los números son los siguientes:

Código	48	49	50	51	52	53	54	55	56	57
número	0	1	2	3	4	5	6	7	8	9

 Una condición de búsqueda puede seleccionar filas mediante comparación de valores, mediante comparación de un valor con rango o un grupo de valores, por correspondencia con un patrón de cadena o por comprobación de valores *NULL*. Las condiciones de búsqueda simples pueden combinarse mediante *AND*, *OR* y *NOT* para formar condiciones de búsqueda más complejas.

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

 <http://technet.microsoft.com/es-es/library/ms187922.aspx>

Tutorial para el uso del operador BETWEEN

 <http://technet.microsoft.com/es-es/library/ms179859.aspx>

Tutorial para el uso del operador LIKE





**UNIDAD DE  
APRENDIZAJE****3****SEMANA****9**

## CONSULTAS CONDICIONALES

---

### LOGROS DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos recuperan información de una base de datos con el empleo de comandos *SQL* haciendo uso de múltiples formas con el uso de algunas condiciones de comparación y funciones para el manejo de campos tipo fecha. Obtienen registros originados por la selección de uno o varios grupos haciendo uso de las funciones agrupamiento y columna procedentes de dos o más tablas.

### TEMARIO

1.1 Empleo de las cláusulas *GROUP BY* y *HAVING*

1.2 Empleo de las funciones *SUM*, *MIN*, *MAX*, *AVG* y *COUNT*

### ACTIVIDADES PROPUESTAS

- Determinan la necesidad de agrupar la salida de datos de una selección.
- Emplean funciones para calcular la suma, promedio, mínimo, máximo y cantidad de un rango de datos.

## 1. **GROUP BY** y **HAVING**

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro, se crea un valor sumario si se incluye una función SQL agregada, como *SUM* o *COUNT* (como veremos después), en la instrucción *SELECT*. Su sintaxis es la siguiente:

```
SELECT CAMPOS
FROM TABLA
WHERE CRITERIO
GROUP BY CAMPOS DEL GRUPO
```

**GROUP BY** es opcional. 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.

Se utiliza la cláusula *WHERE* para excluir aquellas filas que no desea agrupar, y la cláusula **HAVING** para filtrar los registros una vez agrupados.

A menos que contenga un dato Memo, un campo de la lista de campos *GROUP BY* puede referirse a cualquier campo de las tablas que aparecen en la cláusula *FROM*, incluso si el campo no está incluido en la instrucción *SELECT*, siempre y cuando esta instrucción incluya al menos una función SQL agregada.

Todos los campos de la lista de campos de la instrucción *SELECT* deben incluirse en la cláusula *GROUP BY* o como argumentos de una función SQL agregada.

### ACTIVIDAD 1.1

```
SELECT COD_EDIF, COUNT(COD_DEP)
FROM DEPARTAMENTOS
GROUP BY COD_EDIF
```

Una vez que *GROUP BY* ha combinado los registros, *HAVING* muestra cualquier registro agrupado por la cláusula *GROUP BY* que satisfaga las condiciones de la cláusula *HAVING*.

**HAVING** es similar a *WHERE*, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando *GROUP BY*, *HAVING* determina cuáles de ellos se van a mostrar. El siguiente ejemplo lista a los edificios que poseen más de 22 departamentos.

## ACTIVIDAD 1.2

```

SELECT COD_EDIF, COUNT(COD_DEP)
FROM DEPARTAMENTOS
GROUP BY COD_EDIF
HAVING COUNT(COD_DEP) > 22

```

## Ejemplo práctico:

Supongamos que tenemos los siguientes datos de la **tabla DEPARTAMENTOS**

COD_EDIF	COD_DEP	PRECIO_ALQXMES_DEP
EDF001	DEP101	200
EDF001	DEP102	250
EDF002	DEP202	200
EDF003	DEP103	230
EDF003	DEP304	190

Supongamos además que tenemos los siguientes datos de la **tabla DETALLECONTRATO**

COD_CONT	COD_EDIF	COD_DEP	FEC_INI_ALQ	FEC_FIN_ALQ
CNT001	EDF001	DEP101	11/04/2008	10/04/2010
CNT001	EDF001	DEP102	22/05/2007	22/08/2009
CNT002	EDF002	DEP202	01/07/2007	01/08/2009
CNT003	EDF003	DEP103	02/03/2008	02/06/2009
CNT003	EDF003	DEP304	01/05/2007	01/08/2009

### ACTIVIDAD 1.3

Se solicita mostrar el número de contrato y el monto acumulado por cada contrato.

```

SELECT DET.COD_CONT,
MONTO=SUM(DEP.PRECIO_ALQXMES_DEP *
(DATEDIFF(MM,DET.FEC_INI_ALQ,DET.FEC_FIN_ALQ)))
FROM DEPARTAMENTOS DEP, DETALLECONTRATO DET
WHERE  DEP.COD_EDIF.=DET.COD_EDIF.
        AND DEP.COD_DEP=DET.COD_DEP
GROUP BY DET.COD_CONT

```

### ACTIVIDAD 1.4

Haciendo uso de la cláusula **HAVING** se filtran los datos agrupados obtenidos y se muestra solamente los que tengan un MONTO ACUMULADO mayor de 1,000 soles.

La solución sería la siguiente:

```

SELECT DET.COD_CONT,
MONTO=SUM(DEP.PRECIO_ALQXMES_DEP *
        DATEDIFF ( MM,DET.FEC_INI_ALQ,DET.FEC_FIN_ALQ ) )
FROM DEPARTAMENTOS DEP, DETALLECONTRATO DET
WHERE DEP.COD_EDIF.=DET.COD_EDIF.
AND DEP.COD_DEP=DET.COD_DEP
GROUP BY DET.COD_CONT
HAVING SUM(DEP.PRECIO_ALQXMES_DEP *
        DATEDIFF(MM,DET.FEC_INI_ALQ,DET.FEC_FIN_ALQ)) > 1000

```

## 2. SUM

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es:

```
SELECT SUM (EXPR) FROM TABLA
```

En donde EXPR debe ser un valor numérico y representa el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de EXPR pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

### ACTIVIDAD 1.5

Mostrar el total de área construida por código de edificio.

```
SELECT COD_EDIF, SUM(AREA_CONSTRUIDA_DEP) AS  
AREATOTALCONSTRUIDA  
FROM DEPARTAMENTOS  
GROUP BY COD_EDIF
```

## 3. AVG

Calcula la media aritmética de un conjunto de valores CONTENIDO en un campo especificado de una consulta. Su sintaxis es la siguiente:

```
SELECT AVG(EXPR) FROM TABLA
```

En donde EXPR representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por AVG es la media aritmética (la suma de los valores dividido por el número de valores). La función AVG no incluye ningún campo NULL en el cálculo.

### ACTIVIDAD 1.6

Calcular la edad promedio de los usuarios.

```
SELECT AVG(DATEDIFF(YY,FEC_NAC_USUA,GETDATE()))
AS EDADPROMEDIO
FROM USUARIO
```

### 4. *MIN, MAX*

Devuelven el mínimo o el máximo de un conjunto de valores CONTENIDO en un campo específico de una consulta. Su sintaxis es la siguiente:

```
SELECT MIN(EXPR) FROM TABLA
---
SELECT MAX(EXPR) FROM TABLA
---
SELECT MIN(EXPR) , MAX(EXPR)
FROM TABLA
```

En donde EXPR es el campo sobre el que se desea realizar el cálculo. EXPR puede incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

### ACTIVIDAD 1.7

Calcular el haber mínimo y el haber máximo de los inquilinos.

```
SELECT MIN (HABER_BAS_INQ) AS ELMINIMOHABER
FROM INQUILINO
----
SELECT MAX (HABER_BAS_INQ) AS ELMAXIMOHABER
FROM INQUILINO
```

```
---  
  
SELECT MIN (HABER_BAS_INQ) AS ELMINIMOHABER ,  
        MAX (HABER_BAS_INQ) AS ELMAXIMOHABER  
FROM INQUILINO
```

## 5. COUNT

Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente:

```
SELECT COUNT(EXPR) FROM TABLA
```

En donde EXPR contiene el nombre del campo que desea contar. Los operandos de EXPR pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos, incluso texto.

Aunque EXPR puede realizar un cálculo sobre un campo, *COUNT* simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función *COUNT* no cuenta los registros que tienen campos *NULL* a menos que EXPR sea el carácter comodín asterisco (\*). Si utiliza un asterisco, *COUNT* calcula el número total de registros, incluyendo aquellos que contienen campos *NULL*. *COUNT(\*)* es considerablemente más rápido que *COUNT(Campo)*. No se debe poner el asterisco entre dobles comillas ('\*').

### ACTIVIDAD 1.8

Mostrar la cantidad de usuarios registrados.

```
SELECT COUNT(*) AS TOTAL FROM USUARIO
```

Si EXPR identifica a múltiples campos, la función *COUNT* cuenta un registro sólo si al menos uno de los campos no es *NULL*. Si todos los campos especificados son *NULL*, no se cuenta el registro. Hay que separar los nombres de los campos con '+ '.

```
SELECT COUNT(NOM_USUA + APEPATER_USUA) AS TOTAL  
FROM USUARIO
```

## ACTIVIDADES A DESARROLLAR EN CLASE

Usando **TRANSACT/SQL**, realice las siguientes actividades:

En la base de datos DEPARTAMENTOS.

1. Para cada inquilino muestre su nombre, apellido paterno y apellido materno y el número de familiares que posee.

```
SELECT U.NOM_USUA, U.APEPATER_USUA, COUNT(F.COD_FAMI) as
        NUMERO_FAMI
FROM INQUILINO I
INNER JOIN USUARIO U
        ON U.COD_USUA=I.COD_USUA
INNER JOIN FAMILIARES F
        ON I.COD_USUA=F.COD_USUA
GROUP BY U.NOM_USUA, U.APEPATER_USUA
```

2. Muestre el contrato de menor monto.

```
SELECT TOP 1 DET.COD_CONT,
        MONTO=SUM(DEP.PRECIO_ALQXMES_DEP *
        (DATEDIFF(MM,DET.FEC_INI_ALQ,DET.FEC_FIN_ALQ)))
FROM DEPARTAMENTOS DEP
INNER JOIN DETALLECONTRATO DET
        ON DEP.COD_EDIF.=DET.COD_EDIF
        AND DEP.COD_DEP=DET.COD_DEP
GROUP BY DET.COD_CONT
ORDER BY MONTO ASC
```

Algunas veces se desea recuperar los valores de estas funciones solamente para grupos que satisfacen ciertas condiciones. Para hacer esto, *SQL* provee la cláusula *HAVING*, la cual puede aparecer junto con la cláusula *GROUP BY*. *HAVING* provee una condición sobre el grupo de duplas asociadas con cada valor de los atributos agrupados y, en el resultado, aparecen solamente los grupos que satisfacen esta condición. Por ejemplo.



3. Listar el nombre del edificio, área total y área construida del edificio, de aquellos edificios que tienen al menos cinco (05) departamentos por piso.

```
SELECT E.NOM_EDIF, E.AREA_TOTAL_EDIF,  
E.AREA_CONSTRUIDA_EDIF, COUNT(D.COD_DEP)  
FROM EDIFICIOS E  
INNER JOIN DEPARTAMENTOS D  
ON E.COD_EDIF=D.COD_EDIF  
GROUP BY E.NOM_EDIF, E.AREA_TOTAL_EDIF,  
E.AREA_CONSTRUIDA_EDIF  
HAVING COUNT(D.COD_DEP) >= 5
```

4. Para el edificio de nombre LOS ANGELES, muestre el área total construida por piso.






```
SELECT D.PISO_DEP,  
SUM(D.AREA_CONSTRUIDA_DEP) AS TOTALCONSTRUIDO  
FROM EDIFICIOS E  
INNER JOIN DEPARTAMENTOS D  
ON E.COD_EDIF = D.COD_EDIF  
AND E.NOM_EDIF= 'LOS ANGELES'  
GROUP BY D.PISO_DEP
```

**ACTIVIDADES PROPUESTAS**

En la base de datos **VENTAS**, efectúe lo siguiente:

1. Muestre el producto más caro y el producto más barato.
2. Muestre el total de boletas emitidas en el año 2008.
3. Muestre la cantidad de boletas emitidas por vendedor (nombre y apellido paterno) en el primer trimestre del 2008.
4. Muestre los datos de los vendedores que han emitido 10 o más boletas en el año 2008.
5. Muestre el monto total vendido por cada vendedor (nombre y apellido paterno) pero únicamente los montos superiores a 1000 nuevos soles.
6. Muestre el total de productos vendidos por producto (código y descripción del producto) en el año 2008.

## Resumen

-  Las columnas sumarias utilizan funciones de columna SQL para condensar una columna de valores en un único valor.
-  Las funciones de columna pueden calcular el promedio, la suma, el valor mínimo, el valor máximo de una columna, contar el número de valores de datos de una columna o contar el número de filas de los resultados de la consulta.
-  La consultas sumarias sin una cláusula Group By genera una única fila de resultados, sumando todas las filas de una tabla o de un conjunto compuesto de tablas.
-  Una consulta sumaria con una cláusula Group By genera múltiples filas de resultados, cada una acumulando el valor de la suma en fila de un grupo en particular.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 <http://www.diginota.com/trucos-y-tutoriales/sql-funciones-sql.html>

Tutorial para el uso de diversas funciones



**UNIDAD DE  
APRENDIZAJE****3****SEMANA****10**

## CONSULTAS MULTITABLAS

---

### LOGROS DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos recuperan información de una base de datos con el empleo de comandos *SQL* haciendo uso de múltiples formas con el uso de algunas condiciones de comparación y funciones para el manejo de campos tipo fecha. Obtienen registros originados por la selección de uno o varios grupos haciendo uso de las funciones agrupamiento y columna procedentes de dos o más tablas.

### TEMARIO

1.1. Empleo de los operadores *LEFT JOIN*, *RIGHT JOIN*, *FULL JOIN* y *CROSS JOIN*

### ACTIVIDADES PROPUESTAS

- Emplean combinaciones entre una o más tablas para determinar ocurrencias o no.

## UTILIZANDO COMBINACIONES

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:

1. **COMBINACIONES INTERNAS** (la operación de combinación típica, que usa algunos operadores de comparación como = o <>). En este tipo se incluyen las combinaciones equivalentes y las combinaciones naturales.

Las combinaciones internas usan un operador de comparación para hacer coincidir las filas de dos tablas según los valores de las columnas comunes de cada tabla.

2. **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:

### 2.1 *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 sólo 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.

### 2.2 *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.

### 2.3 *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.

## 3. COMBINACIONES CRUZADAS

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.

Por ejemplo, a continuación, se muestra un ejemplo de una combinación cruzada:

```
SELECT COD_USUA, NOM_AVAL_INQ, COD_CONT  
FROM INQUILINO CROSS JOIN CONTRATO  
ORDER BY NOM_AVAL_INQ DESC
```

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.

## 4. COMBINACIONES INTERNAS

### 4.1 *INNER JOIN*

Una combinación interna es aquella en la que los valores de las columnas que se están combinando se comparan mediante un operador de comparación. En el estándar SQL, las combinaciones internas se pueden especificar en las cláusulas *FROM* o *WHERE*. Éste es el único tipo de combinación que *SQL SERVER 2000* admite en la cláusula *WHERE*. Las combinaciones internas especificadas en la cláusula *WHERE* se conocen como combinaciones internas al estilo antiguo.

Esta consulta de *Transact SQL* es un ejemplo de una combinación interna:

```
SELECT *  
  
FROM INQUILINO AS I  
  
INNER JOIN FAMILIARES AS F  
  
ON I.COD_USUA = F.COD_USUA  
  
ORDER BY I.HABER_BAS DESC
```

Esta combinación interna se conoce como una combinación equivalente. 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.

Es equivalente a la siguiente consulta:

```
SELECT *
FROM INQUILINO I , FAMILIARES F
WHERE I.COD_USUA = F.COD_USUA
ORDER BY I.HABER_BAS DESC
```

Comprobando los resultados en el analizador de consultas, observamos que la columna COD\_USUA aparece dos veces. Puesto que no tiene sentido repetir la misma información, se puede eliminar una de estas dos columnas idénticas si se cambia la lista de selección. El resultado se llama combinación natural. La consulta anterior de *Transact SQL* se puede volver a formular para que forme una combinación natural.

Por ejemplo:

```
SELECT I.*, F.NOM_FAMI, F.APEPATER_FAMI,
F.GRADO_PARENTES_FAMI
FROM INQUILINO I , FAMILIARES F
WHERE I.COD_USUA = F.COD_USUA
ORDER BY I.HABER_BAS DESC
```

En el ejemplo anterior, FAMILIARES.COD\_USUA no aparece en los resultados.

#### 4.1.1 Combinaciones con el operador no igual

La combinación no igual (< >) se usa con poca frecuencia. Como regla general, las combinaciones no igual sólo tienen sentido cuando se usan con una auto combinación.

En el siguiente ejemplo de *Transact SQL*, se usa una combinación no igual que se combina con una auto combinación para buscar todas las filas de la tabla DEPARTAMENTOS en la que dos o más filas tengan el mismo COD\_EDIF pero distintos números de COD\_DEP (es decir, edificios con más de un departamento):



```
SELECT DISTINCT D1.COD_DEP, D1.COD_EDIF  
FROM DEPARTAMENTOS D1 INNER JOIN  
DEPARTAMENTOS D2  
ON D1.COD_EDIF = D2.COD_EDIF  
WHERE D1.COD_DEP <> D2.COD_DEP  
ORDER BY D1.COD_DEP
```

**NOTA:** La expresión `NOT column_name = column_name` es equivalente a `column_name < > column_name`.

## 5. COMBINACIONES EXTERNAS

Las combinaciones internas sólo 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*. Todas las filas se recuperarán de la tabla izquierda a la que se haya hecho referencia con una combinación externa izquierda, y 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 2008 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**

SQL Server admite tanto la sintaxis de combinaciones externas como la sintaxis heredada para especificar las combinaciones externas que se basan en la utilización de los operadores `*` y `=*` en la cláusula *WHERE*.

Ejemplo práctico: asumiendo que se tiene los siguientes datos de la **tabla DEPARTAMENTOS**

COD_EDIF	COD_DEP	PRECIO_ALQXMES_DEP
EDF001	DEP101	200
EDF001	DEP102	250
EDF002	DEP202	200
EDF003	DEP103	230
EDF003	DEP304	190
EDF004	DEP101	200

Supongamos, además, que tenemos los siguientes datos de la **tabla DETALLECONTRATO**

COD_CONT	COD_EDIF	COD_DEP	FEC_INI_ALQ	FEC_FIN_ALQ
CNT001	EDF001	DEP101	11/04/2008	11/07/2008
CNT001	EDF001	DEP102	22/05/2008	22/08/2008
CNT002	EDF002	DEP202	01/07/2008	01/08/2008
CNT003	EDF003	DEP103	02/03/2008	02/06/2008
CNT003	EDF003	DEP304	01/05/2008	01/08/2008

### 5.1 **LEFT JOIN** (Utilizado en combinaciones exteriores izquierdas)

El operador de combinación exterior 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.

```
SELECT DET.COD_CONT, DEP.COD_EDIF, DEP.COD_DEP,
DEP.PRECIO_ALQXMES_DEP
FROM DETALLECONTRATO DET LEFT JOIN DEPARTAMENTOS DEP
ON DET.COD_EDIF = DEP.COD_EDIF
AND DET.COD_DEP=DEP.COD_DEP
ORDER BY DET.COD_EDIF ASC, DET.COD_DEP ASC
```

El siguiente es el conjunto de resultados:

COD_CONT	COD_EDIF	COD_DEP	PRECIO_ALQXMES_DEP
CONT001	EDF001	DEP101	200
CONT001	EDF001	DEP102	250
CONT002	EDF002	DEP202	200
CONT003	EDF003	DEP103	230
CONT003	EDF003	DEP304	190

Observamos que para este ejemplo hay coincidencia para todas las filas. Ello se produce porque la tabla DETALLECONTRATO es una tabla débil (Entidad Débil del diagrama entidad relación), pues depende de DEPARTAMENTOS y CONTRATO.

## 5.2 **RIGHT JOIN** (Utilizado en combinaciones exteriores derecha)

El operador de combinación exterior 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.

Para incluir todos los departamentos en los resultados, sin tener en cuenta si hay algún departamento en la tabla DETALLECONTRATO, use una combinación externa derecha. A continuación, se muestra la consulta de *Transact SQL* y los resultados de la combinación externa derecha:

```
SELECT DET.COD_CONT, DEP.COD_EDIF, DEP.COD_DEP,
DEP.PRECIO_ALQXMES_DEP
FROM DETALLECONTRATO DET
RIGHT JOIN DEPARTAMENTOS DEP
      ON DET.COD_EDIF = DEP.COD_EDIF
      AND DET.COD_DEP=DEP.COD_DEP
ORDER BY DET.COD_EDIF ASC, DET.COD_DEP ASC
```

El siguiente es el conjunto de resultados:

COD_CONT	COD_EDIF	COD_DEP	PRECIO_ALQXMES_DEP
CONT001	EDF001	DEP101	200
CONT001	EDF001	DEP102	250
CONT002	EDF002	DEP202	200
CONT003	EDF003	DEP103	230
CONT003	EDF003	DEP304	190
<b>NULL</b>	EDF004	DEP101	200

Observamos que el departamento DEP101 del edificio EDF004 no está en ningún contrato, pero es listado a raíz del operador RIGHT JOIN. Es lógico que, como el departamento no está en ningún contrato, éste sea nulo.

### 5.3 **FULL JOIN** (Utilizado en combinaciones externas completas)

Para retener la información que no coincida al incluir las filas no coincidentes en los resultados de una combinación, utilice una combinación externa completa. *SQL Server 2000* 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*.

```
SELECT DET.COD_CONT, DEP.COD_EDIF, DEP.COD_DEP,
DEP.PRECIO_ALQXMES_DEP
FROM DETALLECONTRATO DET FULL JOIN DEPARTAMENTOS DEP
ON DET.COD_EDIF = DEP.COD_EDIF
AND DET.COD_DEP=DEP.COD_DEP
ORDER BY DET.COD_EDIF ASC, DET.COD_DEP ASC
```

El siguiente es el conjunto de resultados:

COD_CONT	COD_EDIF	COD_DEP	PRECIO_ALQXMES_DEP
CONT001	EDF001	DEP101	200
CONT001	EDF001	DEP102	250
CONT002	EDF002	DEP202	200
CONT003	EDF003	DEP103	230
CONT003	EDF003	DEP304	190
NULL	EDF004	DEP101	200

Se aprecia que las cinco (05) primeras filas, se obtuvieron usando *LEFT JOIN* y las seis (06) últimas filas se obtuvieron usando *RIGHT JOIN*, es decir *FULL JOIN* combina ambos resultados.

#### 5.4 **CROSS JOIN** (Utilizado en combinaciones externas completas)

Se emplea *cross join* cuando se quieren 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.






```
SELECT DET.COD_CONT, DEP.COD_EDIF, DEP.COD_DEP,
DEP.PRECIO_ALQXMES_DEP
FROM DETALLECONTRATO CROSS JOIN DEPARTAMENTOS
ORDER BY DET.COD_EDIF ASC, DET.COD_DEP ASC
```

## ACTIVIDADES PROPUESTAS

### Caso: **VENTAS**

1. Muestre el nombre y apellido paterno de los empleados que no han emitido ninguna boleta. Use el operador *LEFT JOIN*.
2. Muestre el nombre de los productos que no se han vendido. Ordenado por descripción del producto. Use el operador *RIGHT JOIN*.
3. Muestre los datos de las boletas y el nombre del producto de las boletas emitidas en el primer trimestre del año 2008. ordenado por fecha de boleta y en coincidencias ordenarlo por código de cliente, de manera ascendente.

## Resumen

-  En una consulta multitabla, las tablas que contienen los datos son designadas en la cláusula FROM.
-  Cada fila de resultados es una combinación de datos procedentes de una única fila en cada una de las tablas, y es la única fila que extrae sus datos de esa combinación particular.
-  Las consultas multitablas más habituales utilizan las relaciones padre / hijo creadas por las claves primarias y claves foráneas.
-  Una tabla puede componerse consigo misma; las auto composiciones requieren el uso de alias.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 [http://www.aulacltic.es/sql/t\\_3\\_5.htm](http://www.aulacltic.es/sql/t_3_5.htm)

Tutorial para el uso de UNIONES EXTERNAS





UNIDAD DE  
APRENDIZAJE**4**

SEMANA

**11**

## SUB CONSULTAS ANIDADAS Y CORRELACIONADAS, CREACIÓN DE VISTAS

---

### LOGROS DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos emplean los resultados de una consulta como parte de otra en una base de datos de un proceso de negocio real. Asimismo, mejoran el procesamiento y la integridad de una base de datos utilizando vistas sencillas. Luego, crean y emplean vistas complejas de subconjunto, agrupadas y compuestas en una base de datos de un proceso de negocio real.

### TEMARIO

- 1.1. Uso de subconsultas anidadas y correlacionadas
- 1.2. Creación de vistas sencillas

### ACTIVIDADES PROPUESTAS

- Utilizan los resultados de una consulta como parte de otra.
- Comprenden que las subconsultas hacen más fácil la escritura de la sentencia *Select*, descomponiendo una consulta en partes.

## 1. SUBCONSULTAS ANIDADAS

Una subconsulta es una consulta *SELECT* que devuelve un valor único y está 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.

En este ejemplo, se utiliza una subconsulta con la idea de listar a todos los inquilinos que tienen hermanos:

```
SELECT COD_USUA, HABER_BAS_INQ FROM INQUILINO
WHERE COD_USUA IN
      ( SELECT COD_USUA
        FROM FAMILIARES
        WHERE GRADO_PARENTES_FAMI = 'HERMANO' )
```

Se llama también subconsulta a una consulta o selección interna, mientras que la instrucción que contiene una subconsulta también es conocida como consulta o selección externa.

Muchas de las instrucciones *Transact SQL* que incluyen subconsultas se pueden formular también como combinaciones. Otras preguntas se pueden formular sólo con subconsultas.

En *Transact SQL*, normalmente no hay diferencias de rendimiento entre una instrucción que incluya una subconsulta y una versión equivalente que no lo haga. Sin embargo, en algunos casos en los que se debe comprobar la existencia de algo, una combinación produce mejores resultados. De lo contrario, la consulta anidada debe ser procesada para cada resultado de la consulta externa con el fin de asegurar la eliminación de los duplicados. En tales casos, la utilización de combinaciones produciría los mejores resultados. Esto es un ejemplo que muestra una subconsulta *SELECT* y una combinación *SELECT* que devuelve el mismo conjunto de resultados:

```
1.  SELECT COD_USUA, HABER_BAS_INQ FROM INQUILINO
      WHERE COD_USUA IN
            ( SELECT COD_USUA
              FROM FAMILIARES
              WHERE GRADO_PARENTES_FAMI = 'HERMANO' )
```

```
2.  SELECT I.COD_USUA, I.HABER_BAS_INQ
      FROM INQUILINO I, FAMILIARES F
      WHERE I.COD_USUA = F.COD_USUA
      AND F.GRADO_PARENTES_FAMI = 'HERMANO '
```

Una subconsulta anidada en la instrucción externa *SELECT* tiene los siguientes componentes:

- Una consulta *SELECT* normal, que incluye los componentes normales de la lista de selección.
- Una cláusula normal *FROM* que incluye uno o más nombres de tablas o vistas.
- Una cláusula opcional *WHERE*.
- Una cláusula opcional *GROUP BY*.
- Una cláusula opcional *HAVING*.

La consulta *SELECT* de una subconsulta se incluye siempre entre paréntesis. No puede incluir una cláusula *COMPUTE* y sólo puede incluir una cláusula *ORDER BY* cuando se especifica también una cláusula *TOP*.

Una subconsulta puede anidarse dentro de la cláusula *WHERE* o *HAVING* de una instrucción externa *SELECT*, *INSERT*, *UPDATE* o *DELETE*, o dentro de otra subconsulta.

Se puede disponer de hasta 32 niveles de anidamiento, aunque el límite varía dependiendo de la memoria disponible y de la complejidad del resto de las expresiones de la consulta. Las consultas individuales pueden que no admitan anidamientos por encima de 32 niveles. Una subconsulta puede aparecer en cualquier parte en la que se pueda usar una expresión, si devuelve un valor individual.

Si una tabla aparece sólo en una subconsulta y no en la consulta externa, las columnas de esa tabla no se pueden incluir en el resultado (la lista de selección de la consulta externa).

Las instrucciones que incluyen una subconsulta normalmente tienen uno de estos formatos:

- *WHERE* expresión [*NOT*] *IN* (subconsulta)
- *WHERE* expresión\_de\_comparacion [*ANY* | *ALL*] (subconsulta)
- *WHERE* [*NOT*] *EXISTS* (subconsulta)

En algunas instrucciones, la subconsulta se puede evaluar como si fuera una consulta independiente. Conceptualmente, los resultados de la subconsulta se sustituyen en la consulta externa.

Hay tres (03) tipos básicos de subconsultas, con las cuales se realizan las siguientes operaciones:

- Operan en listas incorporadas con *IN*, o en aquellas que modificó un operador de comparación mediante *ANY* u *ALL*.
- Se introducen con un operador de comparación sin modificar y deben devolver un valor individual.
- Son pruebas de existencia que se introducen con *EXISTS*.

### 1.1. *ALL*

El predicado *ALL* se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia *ANY* por *ALL* en el ejemplo anterior, la consulta devolverá únicamente todos los datos de los inquilinos cuyo haber básico es mayor que el de todos los inquilinos cuyo estado civil es soltero. Esto es mucho más restrictivo.

### 1.2. *IN*

El predicado *IN* se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El siguiente ejemplo lista a los edificios que tengan departamentos desocupados:

```
SELECT * FROM EDIFICIOS  
  
WHERE COD_EDIF IN (SELECT COD_EDIF FROM DEPARTAMENTOS  
  
WHERE COD_EST IN (SELECT COD_EST FROM ESTADO  
  
WHERE DESC_EST='DESOCUPADO' ))
```

Inversamente se puede utilizar *NOT IN* para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

### 1.3. *EXISTS*

El predicado *EXISTS* (con la palabra reservada *NOT* opcional) se utiliza en comparaciones de verdadero / falso para determinar si la subconsulta devuelve algún registro.

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula *FROM* fuera de la subconsulta. El siguiente ejemplo lista a los propietarios que han firmado contratos de alquiler.

```
SELECT * FROM PROPIETARIOS  
  
WHERE EXISTS (SELECT * FROM CONTRATO  
  
WHERE PROP_COD_USUA= PROPIETARIO.COD_USUA)
```

El siguiente ejemplo lista a los edificios que tengan departamentos desocupados

```
SELECT * FROM EDIFICIOS  
  
WHERE EXISTS  
  
    (SELECT * FROM DEPARTAMENTOS  
  
    WHERE COD_EDIF=EDIFICIOS.COD_EDIF  
  
    AND EXISTS (SELECT * FROM ESTADO  
  
    WHERE COD_EST=DEPARTAMENTOS.COD_EST  
  
    AND DESC_EST='DESOCUPADO' ))
```

## 2. SUBCONSULTAS CORRELACIONADAS

Se pueden evaluar muchas consultas mediante la ejecución de la subconsulta una vez y la sustitución del valor o valores resultantes en la cláusula WHERE de la consulta externa. En las consultas que incluyen una subconsulta correlacionada (conocida también como una subconsulta repetida), la subconsulta depende de la consulta externa para sus valores. Esto significa que la subconsulta se ejecuta varias veces, una vez por cada fila que pueda ser seleccionada por la consulta externa.

Esta consulta busca el código del contrato y la fecha de firma del mismo para todos aquellos contratos suscritos con inquilinos con estado civil soltero.

```
SELECT COD_CONT, FEC_FIRMA  
  
FROM CONTRATO  
  
WHERE 'SOLTERO' IN  
  
    (SELECT EST_CIVIL_INQ  
  
    FROM INQUILINO  
  
    WHERE COD_USUA = CONTRATO.INQ_COD_USUA)
```

Al contrario que la mayor parte de las subconsultas mostradas anteriormente, la subconsulta de esta instrucción no se puede resolver con independencia de la consulta principal. Necesita un valor para `CONTRATO.INQ_COD_USUA`, pero este valor es una variable. Cambia a medida que *Microsoft SQL Server* examina distintas filas de la tabla `CONTRATO`.

### 3. VISTAS

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. Las vistas se utilizan para alguna de estas funciones, o para todas:

- Restringir el acceso del usuario a filas concretas de una tabla

Por ejemplo, permitir que un empleado sólo vea las filas que guardan su trabajo en una tabla de seguimiento de actividad laboral

- Restringir el acceso del usuario a columnas específicas

Por ejemplo, permitir que los empleados que no trabajen en el departamento de nóminas vean las columnas de nombre, oficina, teléfono y departamento de la tabla de empleados, pero no permitir que vean las columnas con los datos de salario u otra información personal

- Combinar columnas de varias tablas de forma que parezcan una sola tabla
- Agregar información en lugar de presentar los detalles

Por ejemplo, presentar la suma de una columna o el valor máximo o mínimo de una columna.

Las vistas se crean definiendo la instrucción `SELECT` que recupera los datos presentados por la vista. Las tablas de datos a las que hace referencia la instrucción `SELECT` se conocen como las tablas base para la vista. En este ejemplo, `MONTOXCONTRATO` es una vista que selecciona datos de dos tablas base para presentar una tabla virtual de datos frecuentemente utilizados:

```
CREATE VIEW MONTOXCONTRATO

AS

SELECT DET.COD_CONT,
MONTO=SUM(DEP.PRECIO_ALQXMES_DEP *
(DATEDIFF(MM,DET.FEC_INI_ALQ,DET.FEC_FIN_ALQ)))
```

```
FROM DEPARTAMENTOS DEP, DETALLECONTRATO DET  
  
WHERE DEP.COD_EDIF.=DET.COD_EDIF.  
  
AND DEP.COD_DEP=DET.COD_DEP  
  
GROUP BY DET.COD_CONT
```

Una vez creada, se puede hacer referencia a TITLEVIEW en las instrucciones de la misma forma que se hace referencia a una tabla:

```
SELECT * FROM MONTOXCONTRATO
```

### 3.1. Ventajas de las vistas:

- 3.1.1. **Seguridad:** cada usuario puede acceder a la base de datos únicamente a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver, restringiendo así el acceso al usuario a datos almacenados.
- 3.1.2. **Simplicidad de consulta:** una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla haciendo que las consultas multitablas se formulen como consultas de una sola tabla con respecto a la vista.
- 3.1.3. **Simplicidad estructurada:** las vistas pueden dar a un usuario una visión personalizada de la estructura de la base de datos presentando esta como un conjunto de tablas virtuales que tienen sentido para ese usuario.
- 3.1.4. **Aislamiento frente al cambio:** una vez vista puede presentar una imagen consistente inalterada de la estructura de la base de datos, incluso si las tablas fuente subyacentes se dividen.
- 3.1.5. **Integridad de datos:** si los datos se acceden y se introducen a través de una vista, el *DBMS* puede comprobar automáticamente los datos para asegurarse de que satisfacen restricciones de integridad específicas.

### 3.2. Desventajas de las vistas

- 3.2.1. **Rendimiento:** las vistas crean la apariencia de una tabla pero el *DBMS* debe traducir las consultas con respecto a la vista de consultas y con respecto a las tablas fuentes subyacentes. Si la vista se define mediante una consulta multitabla compleja, entonces una consulta sencilla con respecto a la vista se convierte en una composición complicada y puede tardar mucho tiempo en completarse.
- 3.2.2. **Restricciones de actualización:** cuando un usuario trata de actualizar filas de una vista, el *DBMS* debe traducir la petición de una actualización

sobre las filas de las tablas fuentes subyacentes. Esto es posible para vistas sencillas, pero para vistas más complejas no pueden ser actualizadas; son sólo de lectura

### 3.3. CLASIFICACIÓN DE LAS VISTAS

#### 3.3.1 Vistas horizontales

Un uso común de las vistas es restringir el acceso de un usuario a únicamente filas seleccionadas de una tabla. Por ejemplo, en la base de datos ejemplo, supongamos que se desea que los propietarios vean las filas de la tabla CONTRATO correspondientes a ciertos meses (ejemplo imaginario). Para lograr esto, se pueden definir dos vistas del modo siguiente:

```
CREATE VIEW VERCONTRATOSMESESIMPARES
```

```
SELECT * FROM CONTRATO
```

```
WHERE DATEPART(MM,FEC_FIRMA) IN (1,3,5,7,9,11)
```

```
CREATE VIEW VERCONTRATOSMESESPARES
```

```
SELECT * FROM CONTRATO
```

```
WHERE DATEPART(MM,FEC_FIRMA) IN (2,4,6,8,10,12)
```

Ahora se puede dar a cada propietario permiso para acceder a la vista VERCONTRATOSMESESIMPARES o bien a la vista VERCONTRATOSMESESPARES y negarles el permiso para acceder a la otra vista y a la propia tabla CONTRATO.

Una vista como VERCONTRATO se denomina normalmente vista horizontal. Una vista horizontal divide horizontalmente la tabla fuente para crear la vista.

#### 3.3.2 VISTAS VERTICALES

Otro uso habitual de las vistas es restringir el acceso de un usuario a sólo ciertas columnas de una tabla INQUILINOS, por ejemplo en nuestra base de datos.

CODUSUA	NOMAVALINQ	APELL_AVAL	HABER_BAS_INQ	EST_CIVIL_INQ
USU001	ANGEL	LIRA	1100	SOLTERO
USU002	DIANA	VEGA	3500	SOLTERO
USU003	MARCELO	TORRES	1900	CASADO
USU004	ZOILA	AGREDA	2500	SOLTERO
USU005	GABRIEL	DEL RIO	3000	SOLTERO
USU006	IGOR	ROJAS	1000	SOLTERO



Se desea crear una vista que muestre el código, nombre y apellido del aval de todos los inquilinos existentes, para que unos ciertos usuarios tengan acceso sólo a columnas específicas. Entonces, la vista se realizará de la siguiente manera.

**CREATE VIEW VERINQUILINOS AS**

```
SELECT CODUSUA, NOM_AVAL_INQ, APELL_AVAL  
  
FROM INQUILINO
```

Mostrará:

COD_USUA	NOM_AVAL_INQ	APELL_AVAL
USU001	ANGEL	LIRA
USU002	DIANA	VEGA
USU003	MARCELO	TORRES
USU004	ZOILA	AGREDA
USU005	GABRIEL	DEL RIO
USU006	IGOR	ROJAS

**ACTIVIDADES**

- En la base de datos DEPARTAMENTOS:
  - Muestre el nombre, apellido paterno y apellido materno de los inquilinos que tienes el haber básico más alto y más bajo.
  - Muestre el nombre y apellido paterno de los propietarios que no tienen contratos






- Muestre el nombre, dirección y área total de los edificios que tienen departamentos ubicados en el piso 7.

## ACTIVIDADES PROPUESTAS

### Caso: VENTAS

1. Muestre el nombre y apellido de los empleados que no emitieron ninguna boleta en el segundo trimestre de 2008.
2. Muestre la descripción de los productos que se no se han vendido en el mes de diciembre de 2008.
3. Cree una vista que muestre los datos de todos los empleados que no hayan emitido alguna boleta. Luego, efectúe una consulta sobre dicha vista, para mostrar el menor y el mayor año de ingreso de dichos empleado. Emplee etiquetas (alias de campo) para el encabezado del listado de campo
4. Cree una vista que muestre los datos de los productos vendidos en el año 2008. Luego, efectúe una consulta a dicha vista para listar aquellos productos cuya letra inicial de su descripción se encuentra entre la A y la F.
5. Cree una vista que muestre los datos de los productos vendidos. Luego, mediante una consulta a dicha vista, muestre los productos cuya descripción empiezan con la letra 'P' o contenga la cadena 'SA' y que hayan sido vendidos en el segundo trimestre del año 2008.

## Resumen

-  Una subconsulta es una consulta dentro de una consulta. Las subconsultas aparecen dentro de una de las condiciones de búsqueda.
-  Cuando aparece una subconsulta en la cláusula WHERE, los resultados de la subconsulta se utilizan para seleccionar las filas individuales que contribuyen a los datos de los resultados de la consulta.
-  Cuando una subconsulta aparece en la cláusula HAVING, los resultados de la subconsulta se utilizan para seleccionar los grupos de filas que contribuyen con datos a los resultados de la consulta.
-  Una subconsulta puede incluir una referencia externa a una tabla en cualquiera de las consultas que la contienen, enlazando la subconsulta con la fila actual de esa consulta.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 <http://technet.microsoft.com/es-es/library/ms187956.aspx>

Tutorial para la creación de vistas



## CONTINUACIÓN DE CREACIÓN DE VISTAS

---

### LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad 4, los alumnos crean y emplean vistas complejas de subconjunto, agrupadas y compuestas en una base de datos de un proceso de negocio real.

Al finalizar la unidad 5, los alumnos diseñan, crean e importan la estructura de una base de datos con el empleo de la herramienta *ERWIN* para un proceso de negocio real.

### TEMARIO

1.1 Creación de vistas utilizando consultas multitas y subconsultas

1.2 Uso de la herramienta *ERWIN*

### ACTIVIDADES PROPUESTAS

- Desarrollan vistas complejas a varias tablas.
- Emplean una herramienta informática para modelar una base de datos relacional.

## 1. CONTINUACIÓN DE VISTAS

### 1.1 VISTAS CON SUBCONJUNTOS FILA / COLUMNA

Cuando se define una vista *SQL*, no restringe a divisiones puramente horizontales o verticales de unas tablas. De hecho, el lenguaje *SQL* no incluye la noción de vistas horizontales y verticales. Estos conceptos ayudan a visualizar cómo la vista presenta la información a partir de la tabla fuente tanto por la dimensión horizontal como por la vertical.

#### Ejemplo

Se tiene la tabla **USUARIO** y se desea mostrar una vista que obtenga solamente el código, nombre y apellido paterno de los usuarios que hayan nacido después del año 1980.

```
CREATE VIEW USUR_LISTA
AS
SELECT      COD_USUA,
            NOM_USUA,
            APEPATER_USUA
FROM        USUARIO
WHERE       year(FEC_NAC_USUA) > 1980
```

La información devuelta sería:

CÓDIGO	NOMBRE	APELLIDO
U00001	CARLOS	SUÁREZ
U00002	ENRIQUE	SORIA
U00003	PABLO	VERA

Los datos visibles a través de esta vista son un subconjunto fila / columna de la tabla **USUARIO**. Sólo las columnas explícitamente designadas en la lista de selección de la vista y las filas que satisfacen la condición de búsqueda son visibles a través de la vista.

### 1.2 VISTAS AGRUPADAS

La consulta especificada en una definición de vista puede incluir una cláusula **GROUP BY**. Este tipo de vista se denomina agrupada, ya que los datos visibles a través de ella son el resultado de una consulta agrupada. Las vistas agrupadas efectúan la misma función que las consultas agrupadas; agrupan filas relacionadas de datos y producen una fila de resultados de consulta para cada grupo sumando los datos de ese grupo. Una vista agrupada reúne estos resultados de la consulta en una vista virtual y permite efectuar consultas adicionales sobre ella.



**Ejemplos:**

- a) Cree una vista que muestre el código, nombre y apellido paterno, así como el número de familiares que tienen cada inquilino.
  
- b) Cree una vista que muestre el código, nombre, dirección del edificio, código y piso del departamento, así como la cantidad de veces que haya sido alquilado cada departamento. Sin embargo sólo se mostrarán aquellos departamentos que hayan sido alquilados más de tres (03) veces.

- c) Cree una vista que muestre el código, nombre y apellido del inquilino, así como el número de veces que alquilo un departamento. Sin embargo, sólo se mostrarán aquellos inquilinos que hayan alquilado más de cuatro (04) veces un departamento.
  
- d) Cree una vista que muestre el código, nombre del propietario, así como el número de contratos que haya realizado en los últimos 4 meses. Sin embargo, sólo se mostrarán aquellos propietarios que hayan generado más de tres (03) contratos.

### 1.3 VISTAS COMPUESTAS

Una de las razones más frecuentes para utilizar vistas compuestas es simplificar las consultas multitablas. Para ello, hay que especificar una consulta de dos o tres tablas en la definición de una vista. Se puede crear una vista compuesta que extrae datos de dos o tres tablas diferentes y presenta los resultados de la consulta como única tabla virtual. Una vez definida la vista, con frecuencia, se puede utilizar como una consulta simple de una sola tabla.

#### Ejemplos:

- a) **Muestre el código y fecha firma del contrato, así como el código, nombre y apellido paterno de todos los propietarios cuyo nombre contenga en la tercera posición la letra igual a “N” y que los contratos hayan sido generados entre noviembre 2004 y abril 2008. La información debe ser ordenada por en nombre del propietario en forma descendente.**
  
- b) **Muestre el código, nombre y dirección del edificio, código y piso del departamento, así como la fecha de inicio y fin que fue alquilado de todos los departamentos que hayan sido alquilado en los últimos ocho (08) meses y que se encuentren ubicados en “SAN ISIDRO”.**

#### **1.4 ACTIVIDADES**

Muestre el total y el promedio de las áreas construidas de los departamentos cuyo inicio de alquiler se produjo a partir del centésimo día del año 2008. El resultado debe estar clasificado por el nombre del edificio y sólo debe incluir aquellos cuyo total de área construida sea mayor a 400.

--Creación de la vista

-- Ejecutar la vista

-- Recuperar el texto de una vista

-- Modificación de la vista con ENCRIPCIÓN

## 2. MODELADOR DE BASE DE DATOS – *ERWIN* No entra

*ERWIN* v. 7.0, es una herramienta de fácil utilización para el diseño de base de datos relacionados que permite, a la vez que el diseño, la generación y el mantenimiento.

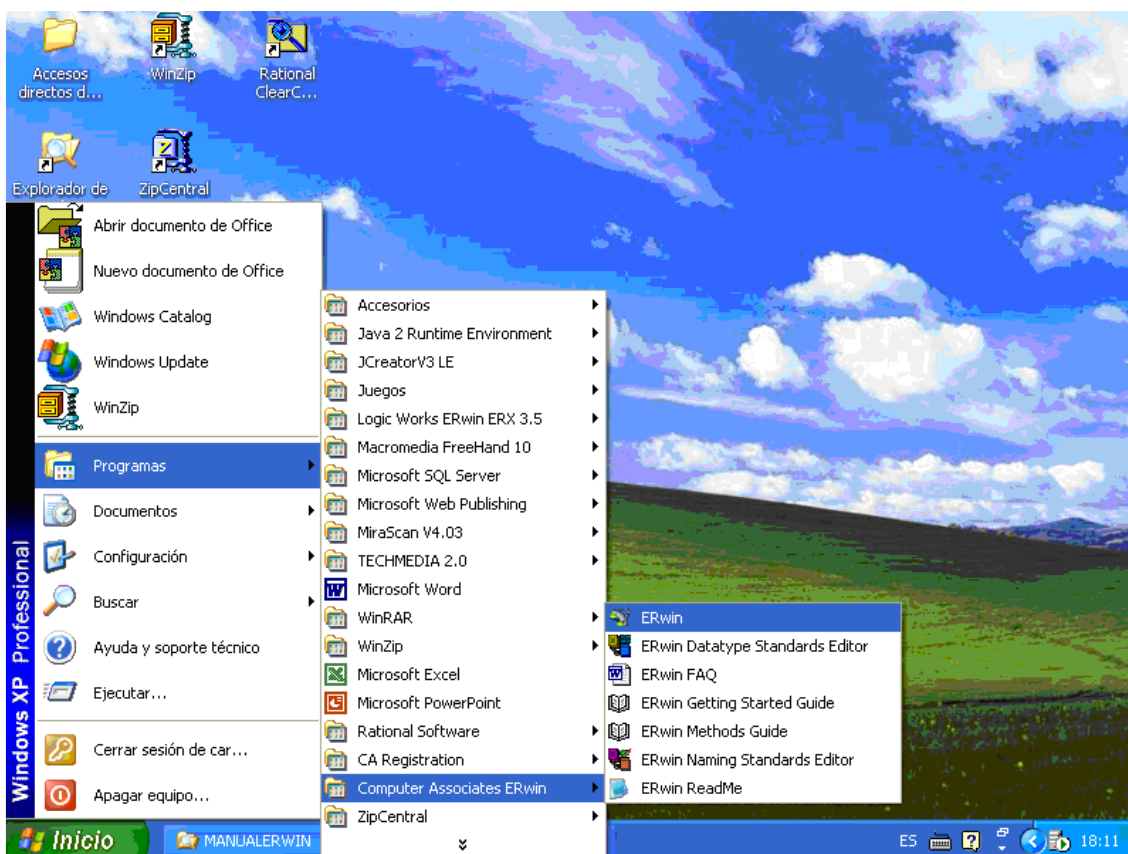
*ERWIN* permite visualizar la estructura, los elementos claves y el diseño optimizado de la base de datos, desde el modelo lógico en donde se reflejan los requerimientos del negocio, hasta el modelo físico con las características específicas de la base de datos elegida.

*ERWIN* no es sólo una herramienta de diseño de base de datos, sino que es una herramienta de desarrollo de *RDBMS* que permite generar automáticamente tablas y miles de líneas de código de procedimientos almacenados y *triggers*.

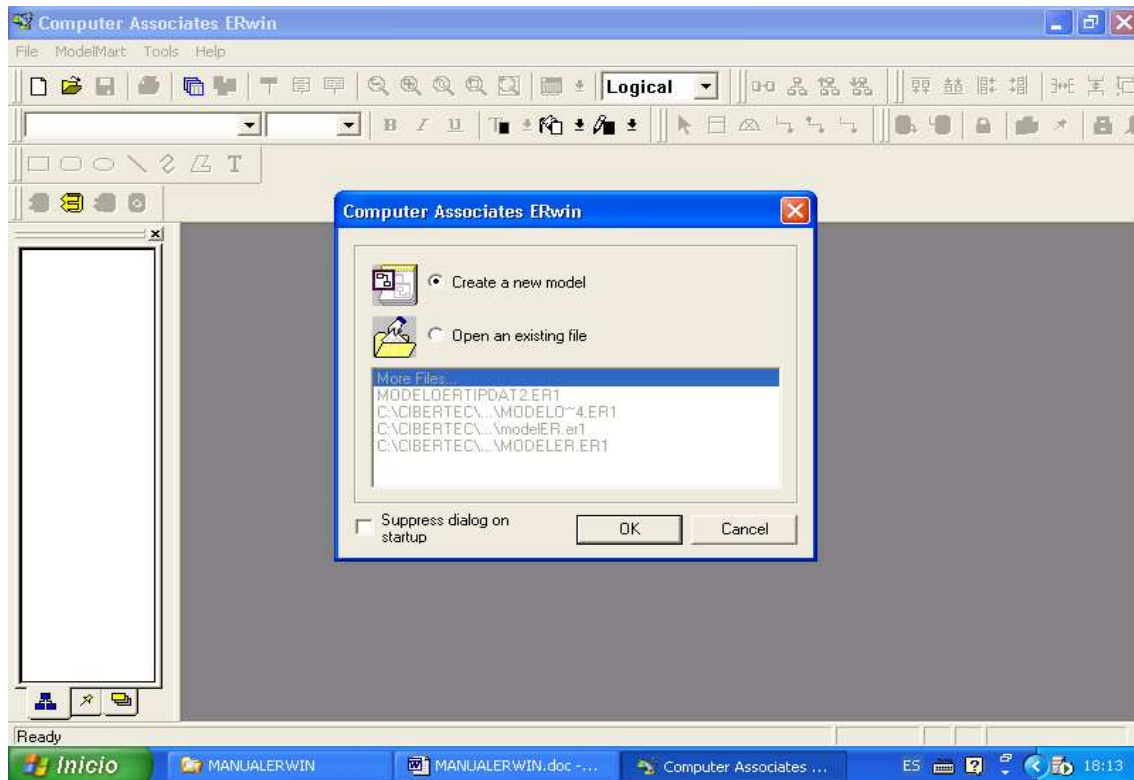
A continuación mostraremos la herramienta *ERWIN*.

### 2.1 INICIANDO EL PROGRAMA

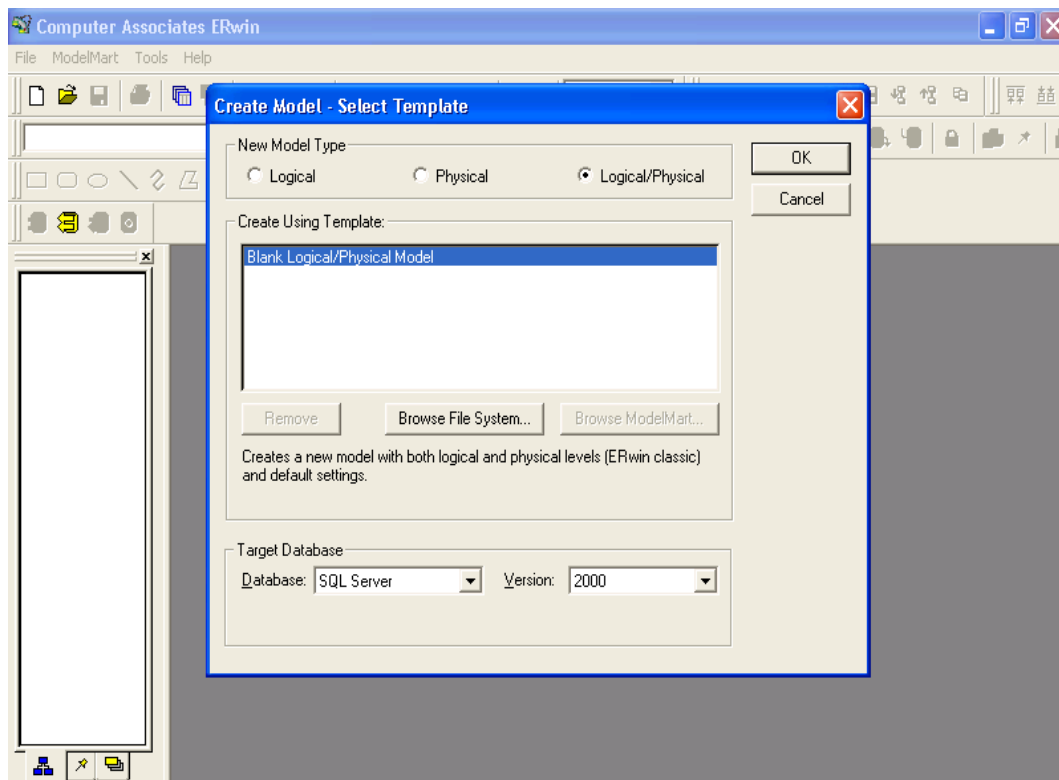
1. En el menú inicio, busque el icono correspondiente a *ERWIN*.



2. Al ingresar, le aparecerá una pantalla donde se le preguntará si desea abrir un archivo ya creado (aparecerá los que existen) o abrir uno en blanco. En nuestro caso, elegiremos uno en blanco.

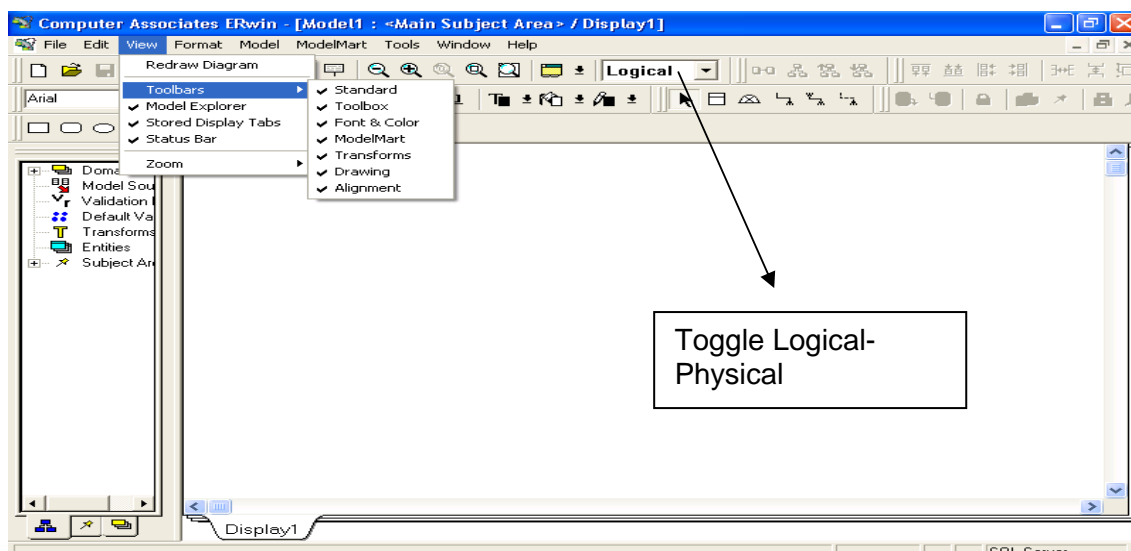


3. Luego aparecerá una nueva ventana donde se elegirá si se desea realizar el modelado lógico o físico de la base de datos. Elegiremos la opción lógico/físico, en la parte inferior aparece un combo en el cual donde elegiremos el tipo de base de datos y la versión del mismo. En nuestro caso, *SQL SERVER* versión 2008.

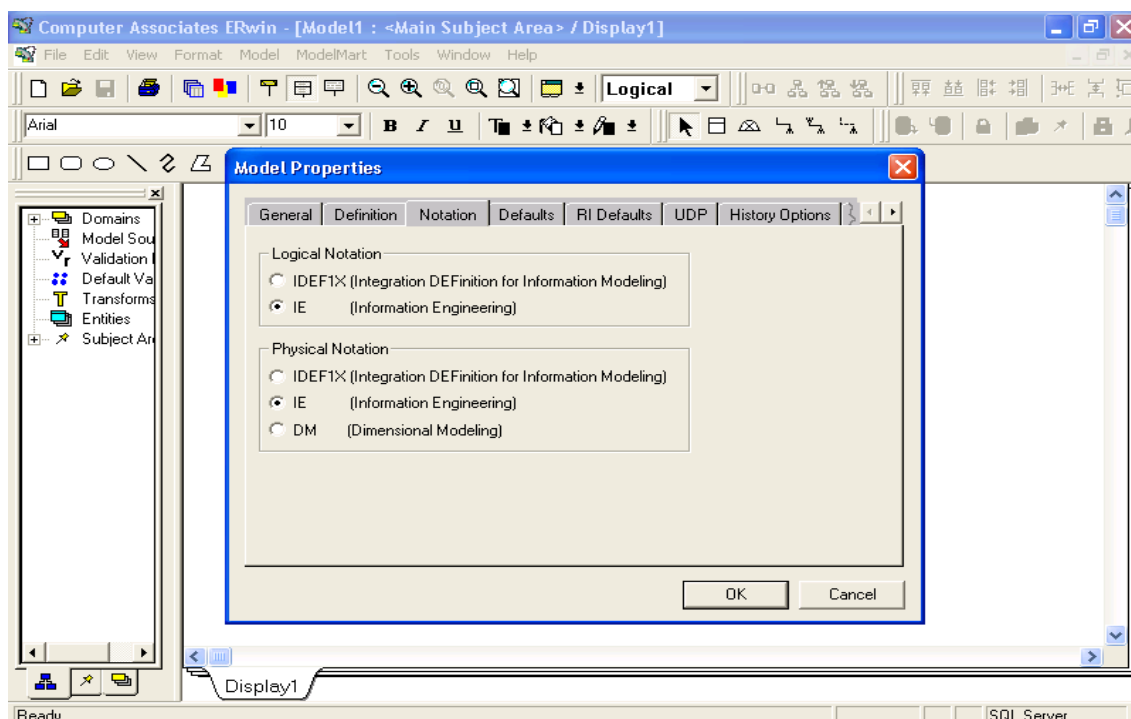


**PREPARACIÓN DEL ENTORNO DEL TRABAJO:**

1. En la interfase de usuario de *ERWIN*, asegúrese de que la barra de herramientas (*Toolbox*) esté visible. Si no lo estuviera, vaya al menú de *windows* elija *View – Toolbars – Toolbox* y verifique que tenga el *check* respectivo.
2. Verifique que en el control *Toggle Logical-Physical* de la barra de herramientas esté especificado *Logical*.



3. Para establecer el tipo de notación a utilizar en el menú *Model – Model Properties* en la pestaña *Notación*, seleccione tanto en *Logical Notation* como en *Physical Notation IE (Information Engineering)*. Luego haga clic en *OK*.

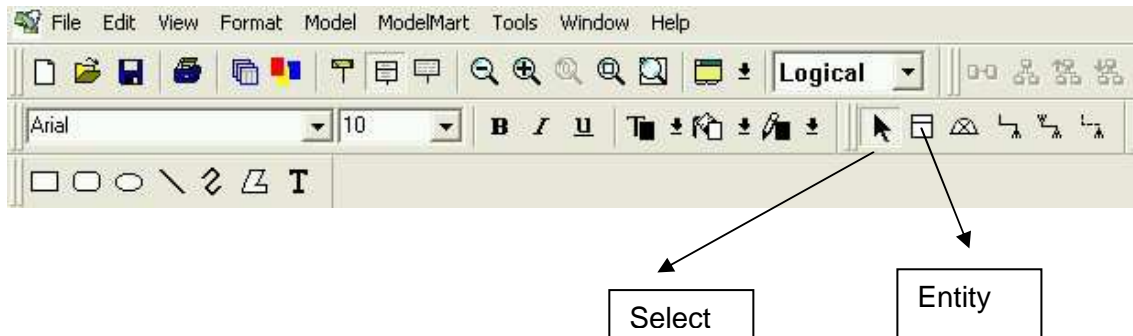


## DEFINICIÓN DE UNA NUEVA ENTIDAD:

**Entidad:** Algo o alguien acerca del cual se coleccionan y se procesan datos. Por ejemplo: Alumnos, clientes, productos, trabajadores.

### Creación de la entidad:

1. En *ERWIN Toolbox* haga clic sobre el botón *Entity*, y luego haga clic sobre la posición del diagrama en la que colocará la nueva entidad.



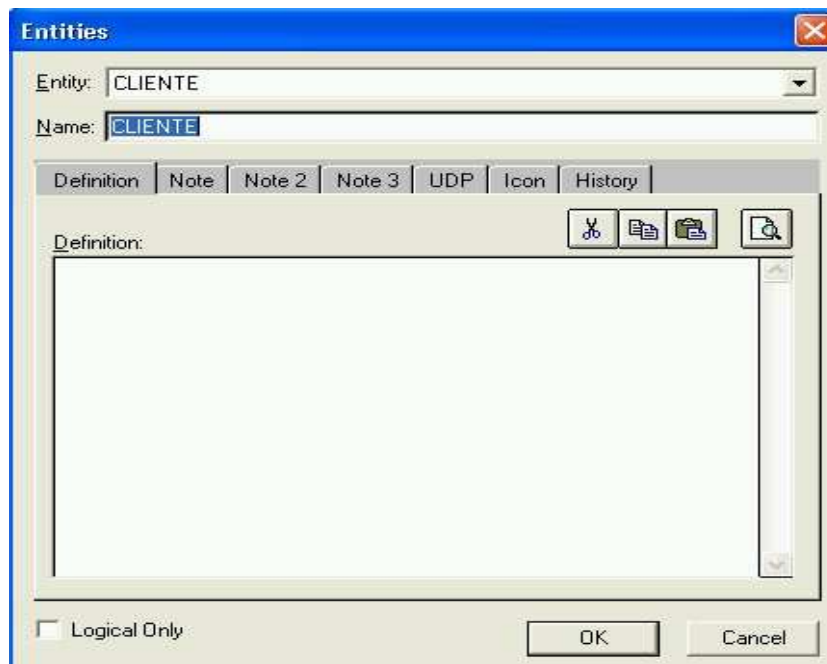
### Definición del nombre de la entidad:

1. Al insertar la nueva entidad, puede escribir el nombre de la misma en ese instante si no puede hacer clic sobre la entidad con el botón secundario del ratón y del menú contextual elegir *Entity Properties*.



2. En el control *Name* del diálogo *Entity Properties*, digite el nombre de la nueva entidad. Por ejemplo Cliente.
3. Haga clic en el botón OK.





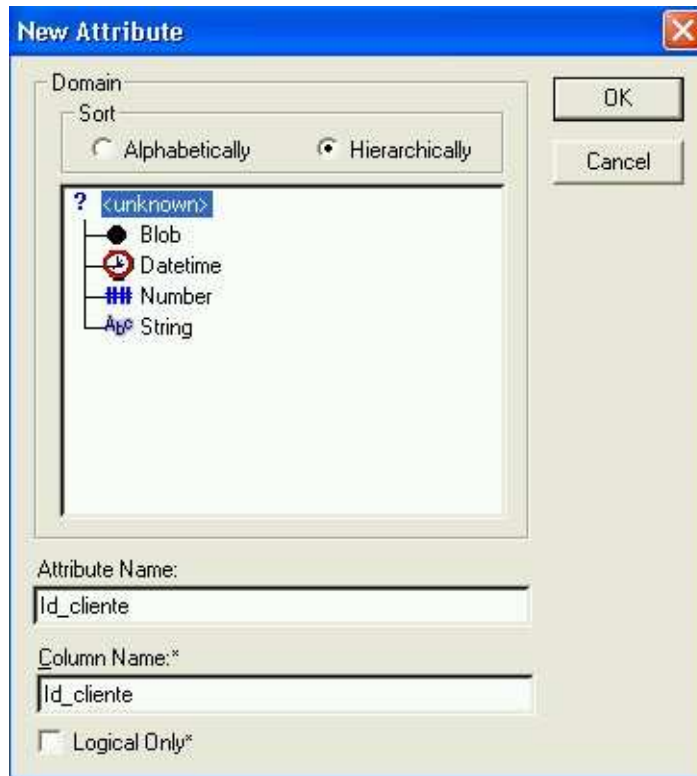
### Definición de los atributos de la entidad:

1. Haga clic sobre la entidad con el botón secundario del ratón.
2. En el menú contextual haga clic sobre *Atributtes*. Aparece el diálogo *Attribute Editor*.



### Para definir un nuevo atributo:

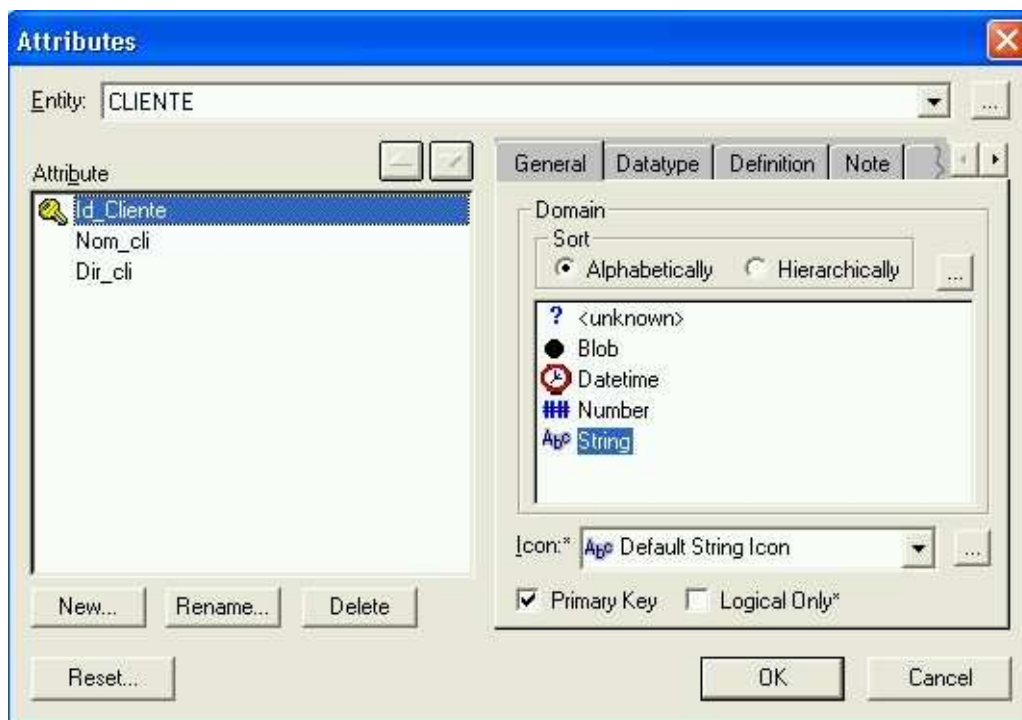
3. Haga clic en el botón *New* del diálogo *Attribute*.
4. En el control *Attribute Name* del diálogo *New Attribute*, digite el nombre del nuevo atributo. Por ejemplo, digite *IdCliente*.



5. Observe que, cuando el control *Attribute Name* pierde el enfoque, el control *Column Name* toma el mismo valor que *Attribute Name*. Si desea, redefina el nombre de la columna.
6. Haga clic en OK.

### Para definir el tipo de datos de un atributo:

7. En el diálogo *Attribute*, seleccione el atributo cuyo tipo de dato desea definir.
8. En la ficha General, seleccione el tipo entre *Blob*, *Datetime*, *Number* y *String*. El tipo de dato *Blob* es usado para los campos de sonido, imagen, video o un número binario de gran tamaño.
9. Si el valor del atributo (dato) es el *primary key*, marque en el recuadro respectivo dicha opción.



10. Para finalizar la definición de esta entidad, haga clic en el botón OK.

## DEFINICIÓN DE LAS RELACIONES:

ERWIN permite especificar tres tipos de relaciones:

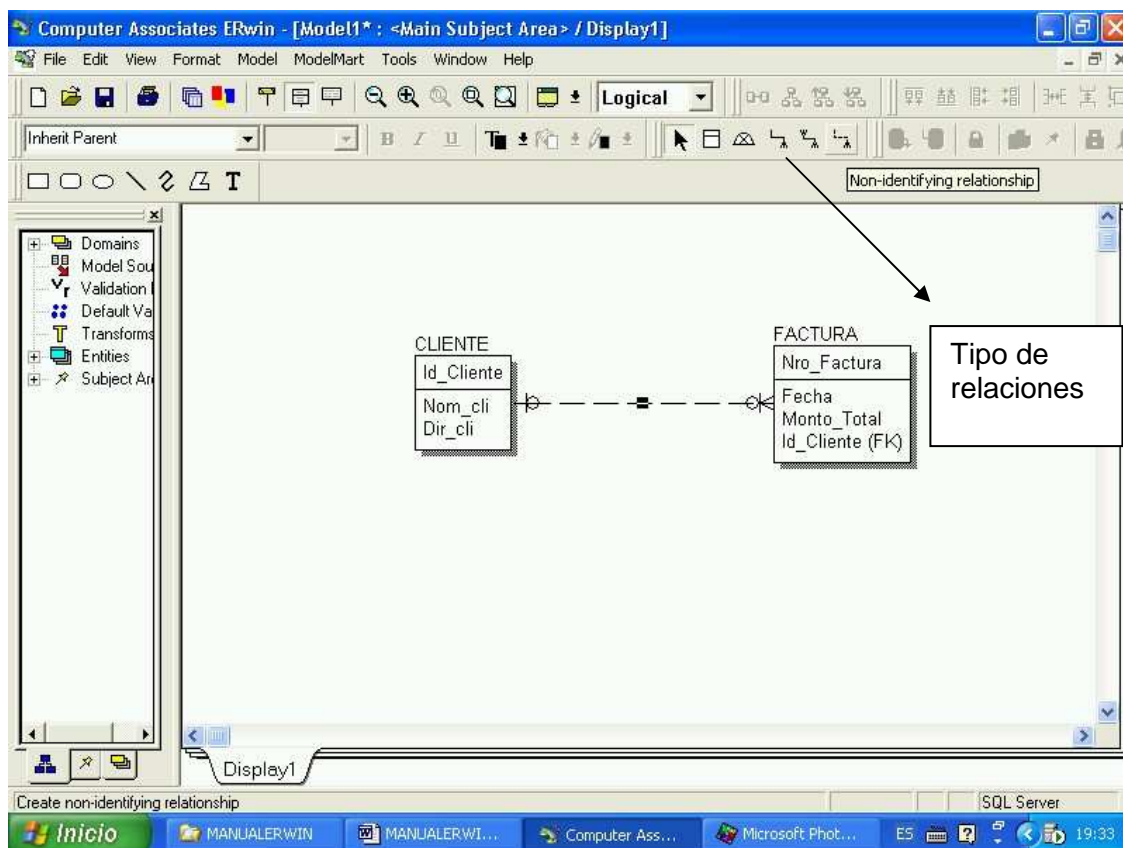
- Relación no identificada (*Non-identifying relationship*) – Uno a muchos
- Relación identificada (*Identifying relationship*) – Entidad débil
- Relación de muchos a muchos (*Many-to-many relationship*) – Sólo en el modelo lógico luego deberá ser reemplazada por el correspondiente detalle para romper dicha relación.

### **Definición de una relación no identificada :**

Se presenta cuando la entidad hija no depende de la entidad padre para su identificación. Una instancia de la entidad padre está relacionada a muchas instancias de la entidad hija. Por ejemplo, la relación entre las entidades DEPARTAMENTO Y EMPLEADO. Un departamento tiene muchos empleados y la identificación de un empleado no depende del departamento en el que trabaja. (Adolfo López sigue siendo Adolfo López independientemente del departamento en el que trabaje).

**Para definir la relación:** Definamos dos entidades Cliente y Factura con algunos atributos. Cliente: Id\_Cliente, Nom\_Cliente. y Factura: Nro\_Factura, Monto\_Total luego:

1. Verifique que en el cuadro *Toggle Logical-Physical* de la barra de herramientas esté especificado *Logical*.
2. En *Erwin Toolbox* haga clic en el botón *Non-Identifying relationship*.
3. Haga clic sobre la entidad padre (haga clic sobre CLIENTE).
4. Haga clic sobre la entidad hija (haga clic sobre FACTURA).



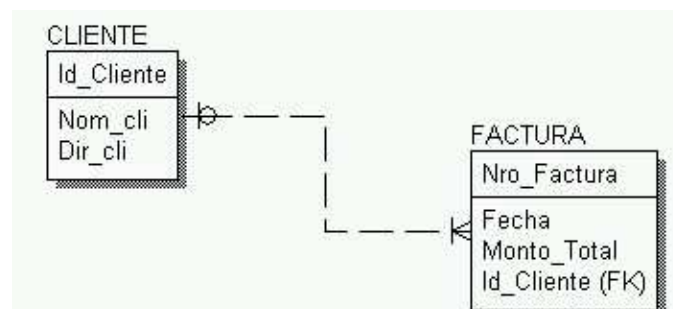
### Para definir la cardinalidad de la relación:

5. Haga clic sobre la relación utilizando el botón secundario del ratón.
6. En el menú contextual, ejecute *Relationship Properties*.
7. En la sección *Relationship Cardinality* de la ficha General del diálogo *Relationship Properties*, seleccione la *cardinalidad* entre las siguientes:
  - De cero a uno a cero, uno a muchos (*Zero, One or More*)
  - De cero o uno a uno o muchos (*One or More (P)*)
  - De cero o uno a cero o uno (*Zero or One (Z)*)
  - De cero o uno a exactamente n (*Exactly*).

Para nuestro ejemplo, seleccionaremos *One or More*. En la sección *Relationship Type*, se elige el tipo de relación (identificada o no identificada), además puede especificar si dicha relación aceptará nulos o no en la misma. En nuestro caso, elegimos permitir nulos (*Nulls Allowed*) pues puede existir un cliente que no posea factura.

8. Haga clic en el botón OK.

**MUY IMPORTANTE:** Observe que en una relación no identificada la llave foránea (FK) es un atributo no clave.

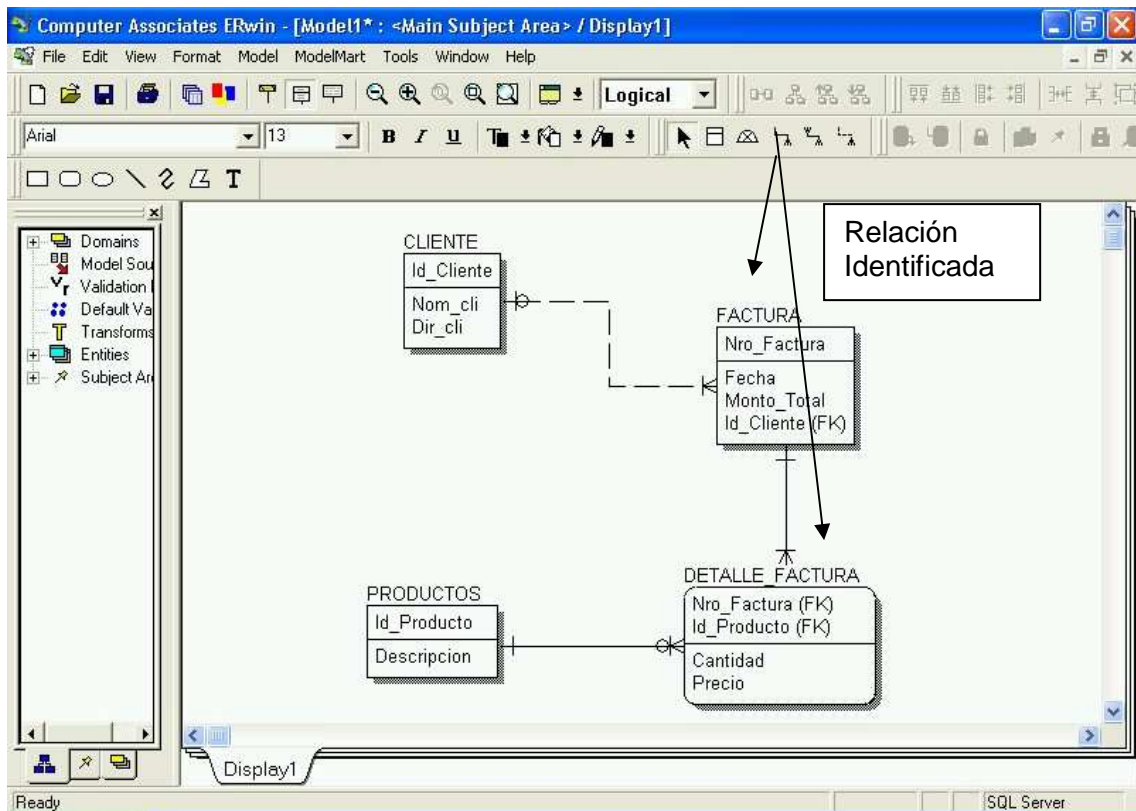


#### Definición de una relación identificada:

Se presenta cuando la entidad hija depende de la entidad padre para su identificación, es decir, una instancia de la entidad hija no puede existir sin su correspondiente instancia de la entidad padre. Una instancia de la entidad padre puede estar relacionada a muchas instancias de la entidad hija. Por ejemplo, la relación entre las entidades FACTURA Y DETALLE\_FACTURA, una factura puede tener muchas líneas de detalle y para identificar a cada una de las líneas de detalle es necesario especificar a cual de las facturas pertenece cada una de las líneas de detalle, por lo tanto, no puede existir una línea de detalle sin su correspondiente factura.

**Para definir la relación:** Definamos dos entidades Producto y Detalle\_Factura con algunos atributos. Producto: IdProducto (PK), Descripción, y Detalle\_Factura con cantidad y precio sin *primary key*. Luego:

1. Verifique que en el cuadro *Toggle Logical-Physical* de la barra de herramientas esté especificado *Logical*.
2. En *ERWIN Toolbox*, haga clic en el botón *Identifying relationship*.
3. Haga clic sobre la entidad padre (haga clic sobre FACTURA).
4. Haga clic sobre la entidad hija (haga clic sobre DETALLE\_FACTURA).
5. Haga clic sobre la otra entidad padre (haga clic sobre PRODUCTO).
6. Haga clic nuevamente sobre la entidad hija (haga clic sobre DETALLE\_FACTURA).



**Para definir la cardinalidad de la relación:**

7. En la relación entre FACTURA y DETALLE\_FACTURA, haga clic con el botón secundario del ratón y elija *Relationship Properties*.
8. En la sección *Relationship Cardinality* de la ficha General del diálogo *Relationship properties*, elija entre las siguientes:
  - De cero a uno a cero, uno a muchos (*Zero, One or More*)
  - De cero o uno a uno o muchos (*One or More (P)*)
  - De cero o uno a cero o uno (*Zero or One (Z)*)
  - De cero o uno a exactamente n (*Exactly*).

Para nuestro ejemplo, seleccionaremos *One or More*. Al elegir la relación identificada no se habilita la posibilidad de permitir nulos en la relación pues no es posible dicha relación al ser una dependiente de la otra.

Relationships

Relationship: FACTURA R/3 DETALLE\_FACTURA

New... Delete

General Definition Rolename RI Actions UDP

Verb Phrase

Parent-to-Child: R/3

Child-to-Parent:

Relationship Cardinality

Summary: One-to-One-or-More (P)

Cardinality

☐ Zero, One or More

☒ One or More (P)

☐ Zero or One (Z)

☐ Exactly:

Relationship Type

☒ Identifying

☐ Non-Identifying

Nulls

☐ Nulls Allowed

☒ No Nulls

☐ Logical Only

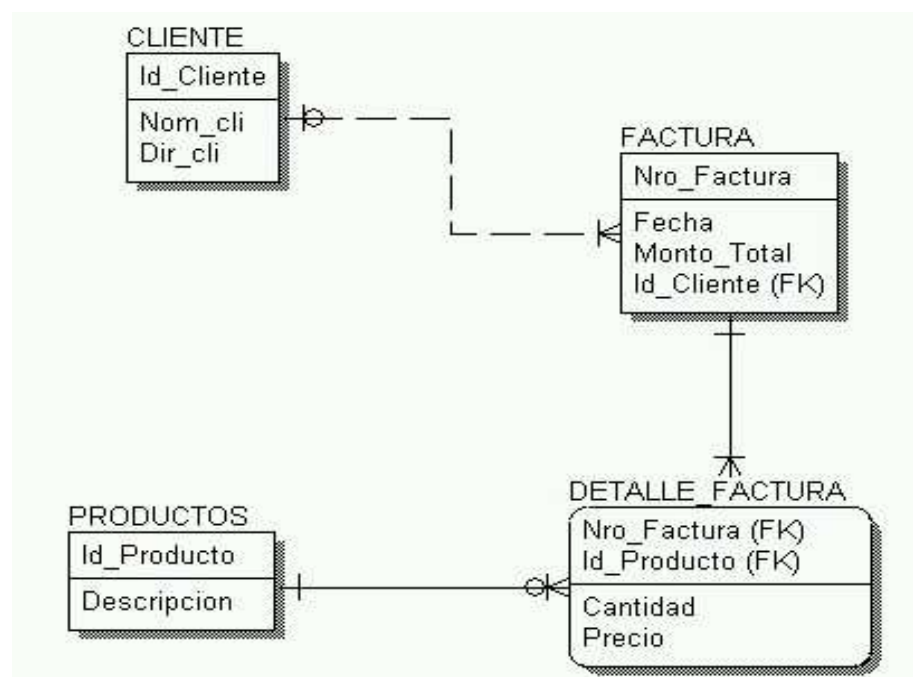
OK Cancel

9. Haga clic en el botón OK.

10. Haga lo mismo para la relación entre PRODUCTO y DETALLE\_FACTURA.

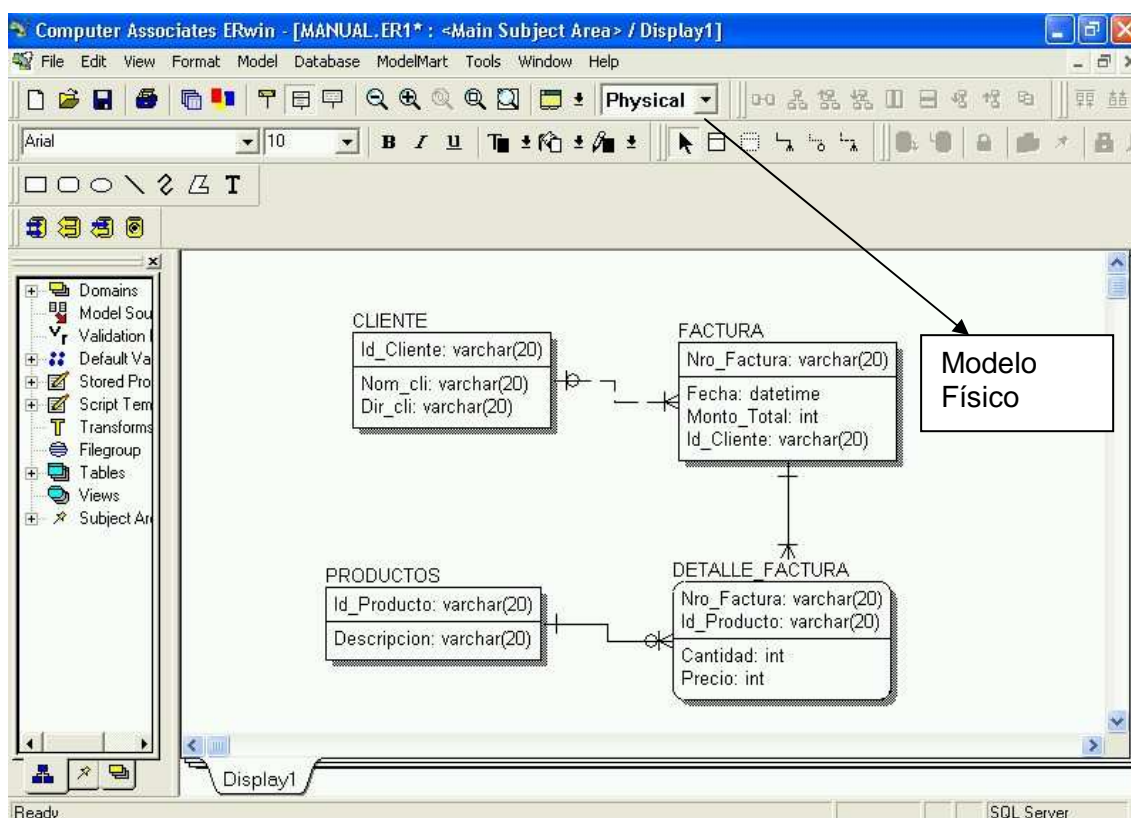
**MUY IMPORTANTE:** Observe que en una relación identificada la llave foránea forma parte de la llave primaria de la entidad hija. Además, las esquinas redondeadas de la entidad indican que es una entidad dependiente.





## INTRODUCCIÓN AL MODELAMIENTO FÍSICO CON *ERWIN*

1. En el control *Toggle Logical-Physical* de la barra de herramientas seleccione *Physical*.

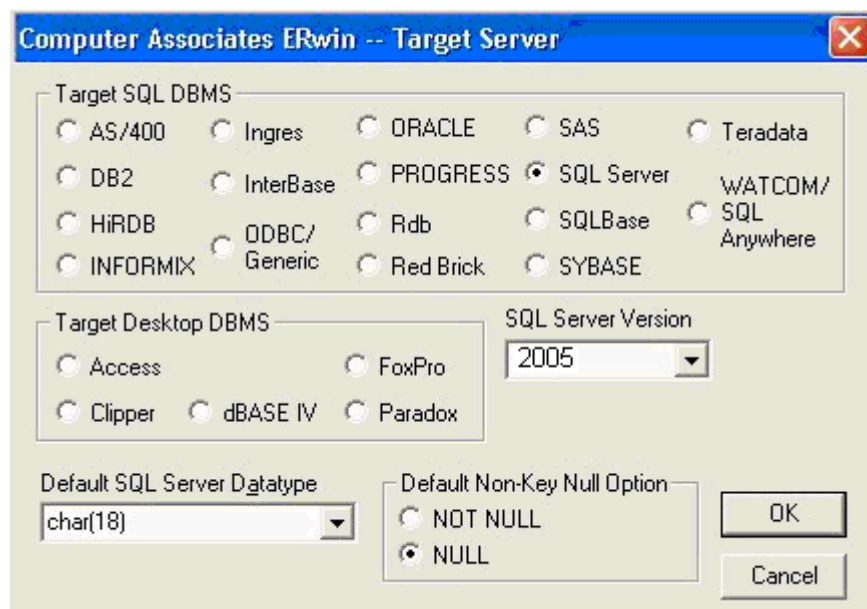




2. Observe que por defecto, el modelo físico utiliza el nombre de la entidad como nombre de la tabla, y que además las columnas han sido definidas con tipos predeterminados.

### SELECCION DEL SOFTWARE DE BASES DE DATOS EN QUE SE GENERARÁ LA NUEVA BASE DE DATOS

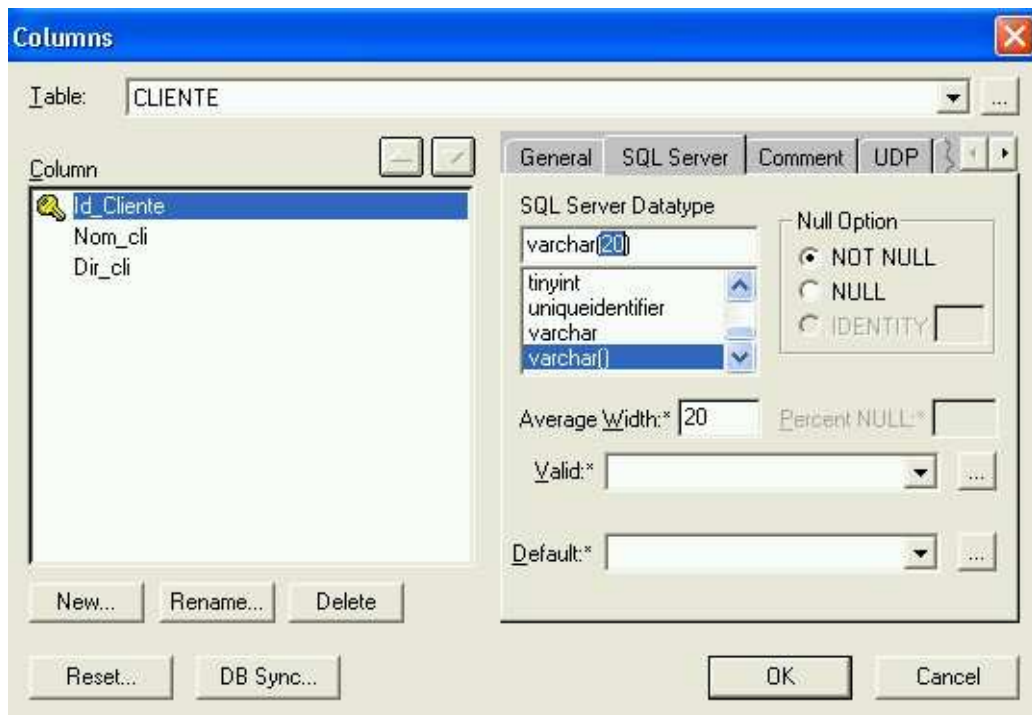
1. En el menú *Database*, ejecute la opción *Choose Database*.
2. En el diálogo *Target Server*, seleccione el software de bases de datos a emplear. De ser necesario, seleccione también la versión. Para nuestro ejemplo, seleccionaremos *SQL Server*.
3. En el control *Default Non-Key Null Option*, especifique si los atributos no claves permitirán o no valores nulos como valores predeterminados.



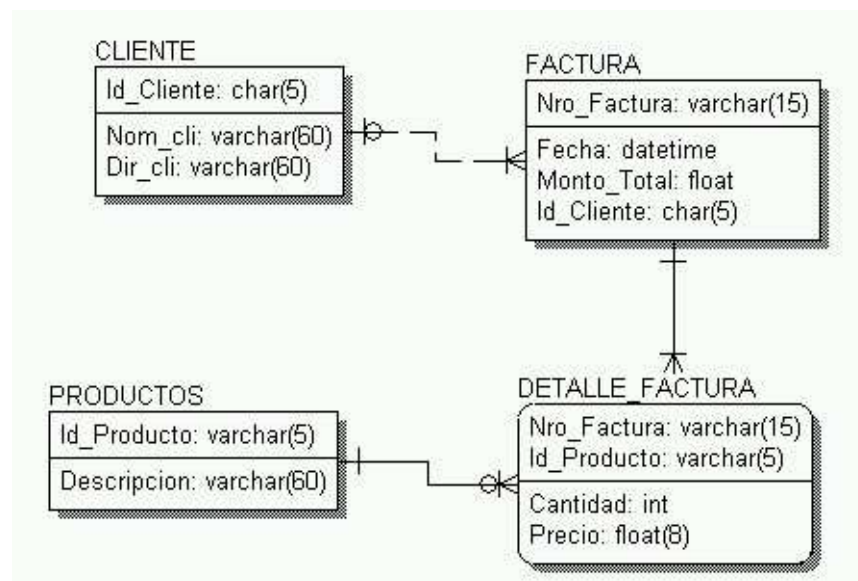
4. Haga clic en el botón OK.

### REDEFINICIÓN DE LOS TIPOS DE DATOS PARA EL MODELO FÍSICO

1. Haga clic sobre la entidad con el botón secundario del ratón.
2. En el menú contextual, seleccione *Columns*.
3. En el diálogo *Columns*, debe aparecer una pestaña *SQL Server* al costado de la pestaña *General*. Seleccione la pestaña *SQL Server*.



4. Utilizando la pestaña *SQL Server* redefine los tipos de datos de cada atributo de la entidad, especificando el tamaño del mismo.



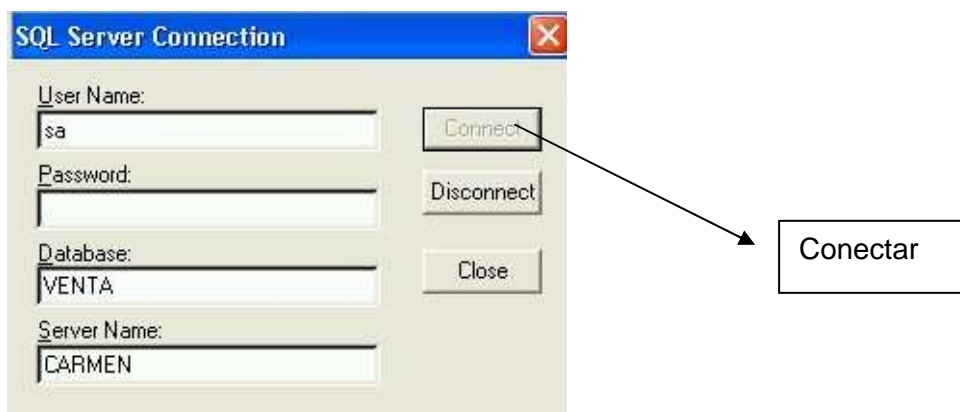
5. Al finaliza haga clic en OK.

## GENERACIÓN DE LOS OBJETOS DE LA BASE DE DATOS

En este punto con la ayuda del Administrador Corporativo, cree una base de datos llamada VENTA para nuestro ejemplo.

## CONEXIÓN DE ERWIN CON SQL SERVER

1. Regrese a su modelo físico en *ERWIN*, luego del menú *Database* ejecute la opción *Database Connection*.
2. En *User Name* y *Password* proporcione la identificación y la contraseña de una cuenta *SQL Server* válida. Para nuestro laboratorio, digite *sa* en *User Name* y deje *password* en blanco.
3. En *Database* escriba el nombre de la base de datos a la que desea conectarse (digite *VENTA*).
4. En *Server Name* digite el nombre de su servidor *SQL*.



5. Haga clic en el botón *Connect*. Si no recibe ningún mensaje la conexión se ha establecido.

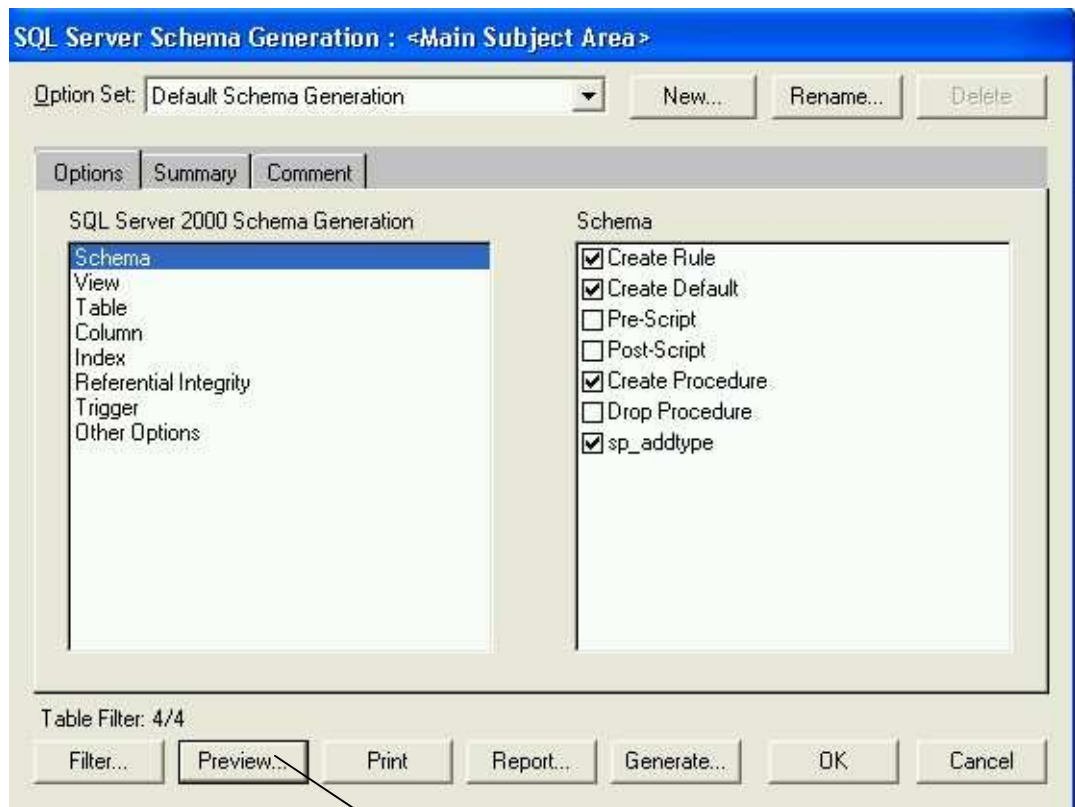
## CREACIÓN DE LOS OBJETOS DE LA BASE DE DATOS

Para crear los objetos de la base de datos, *ERWIN* crea un procedimiento que contiene todas las sentencias *SQL* que deben ejecutarse para crear los objetos definidos en el modelo Físico. Puede:

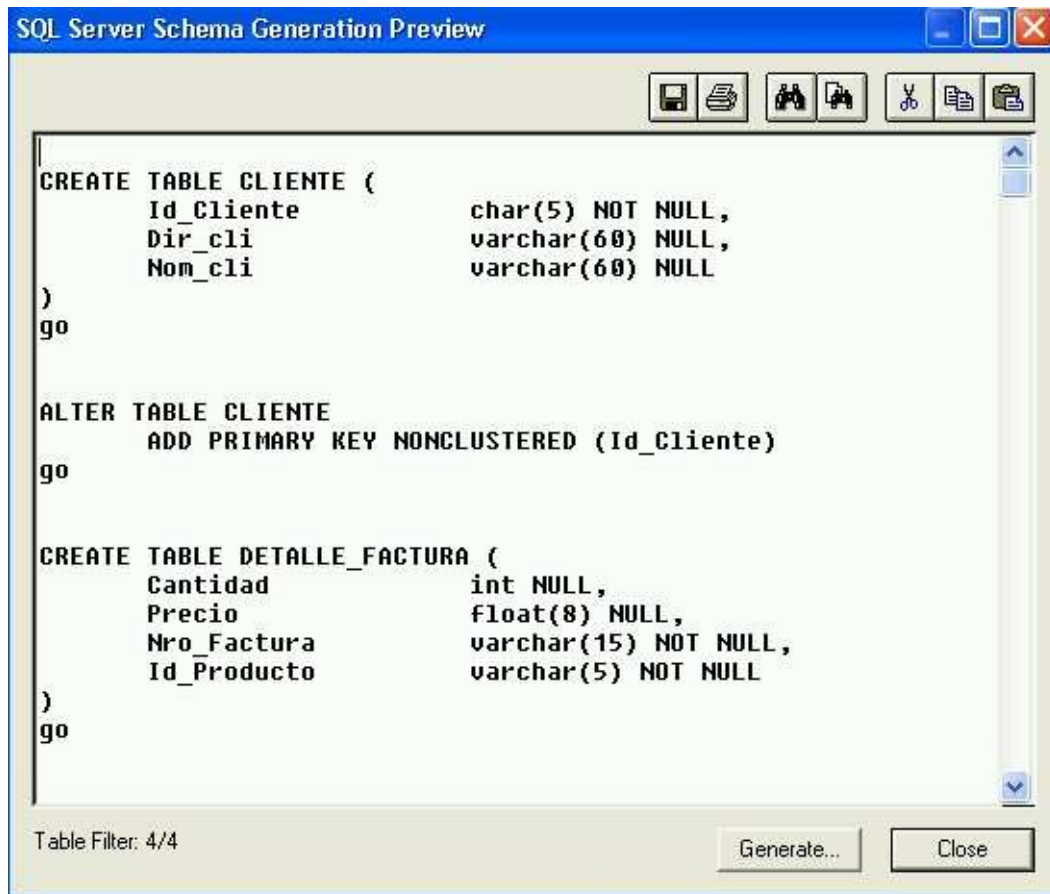
- Utilizar la conexión establecida para crear en este momento los objetos de la base de datos.
- Generar un archivo *script* (programa) que guarde el procedimiento que crea los objetos de la base de datos, y ejecutar posteriormente el procedimiento desde el cliente *SQL Server*.

**Para revisar el procedimiento que crea los objetos de la base de datos:**

1. En el menú *Tools* ejecute *Forward Engineer / Schema Generation*.
2. En el diálogo *SQL Server Schema Generation*, haga clic en el botón *Preview*. Podrá ver todas las instrucciones que se ejecutarán para crear los objetos definidos en el modelo físico.



Haga clic  
aquí.



3. Haga clic en el botón *Close* al finalizar la revisión.

Para crear un **script SQL** que puede ser editado y ejecutado posteriormente:

1. En el diálogo *SQL Server Schema Generation*, haga clic en el botón *Report*.
2. En el diálogo *Generate SQL Server/SQL Schema Report*, especifique la ubicación y el nombre del archivo que almacenará el *script* (guárdelo con extensión *sql*), por ejemplo *ObjetosVentas.sql*.

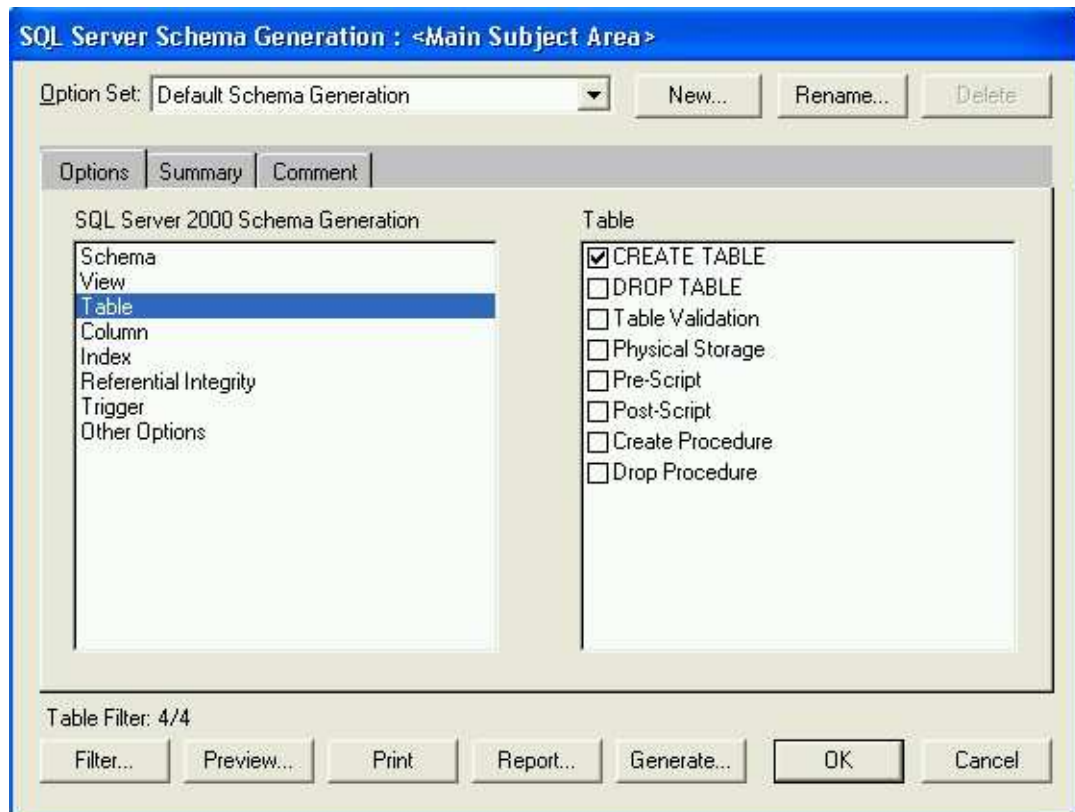


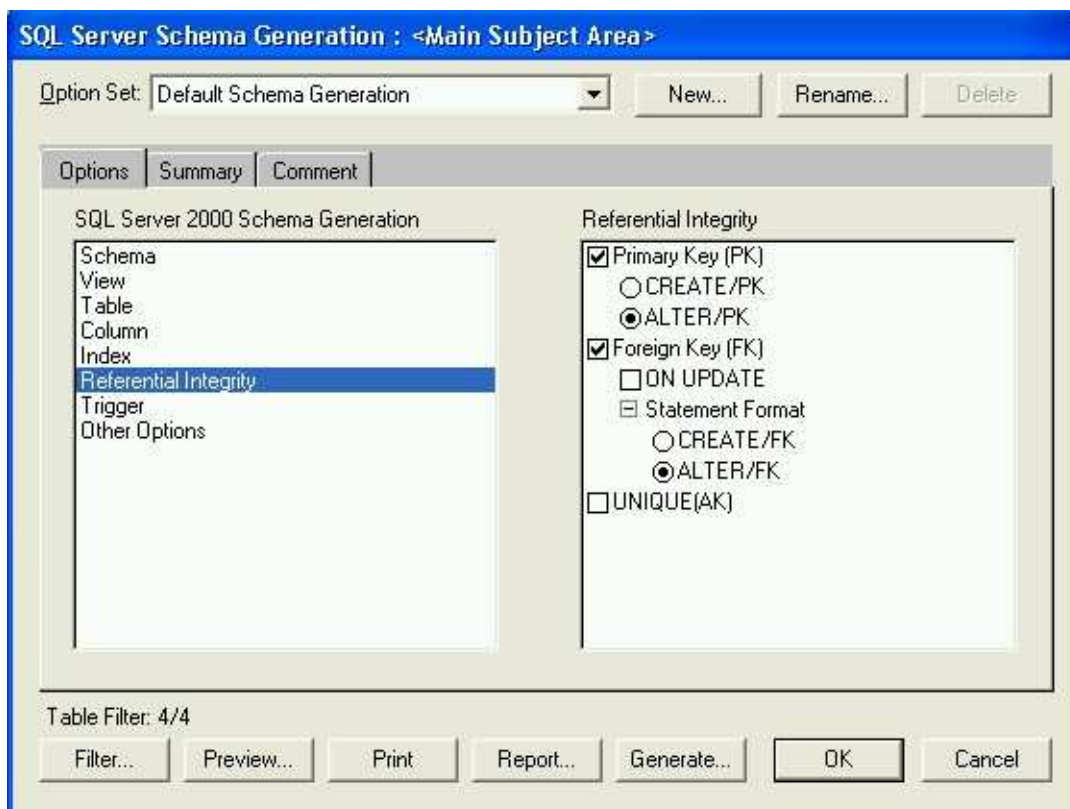
3. Haga clic en el botón *Guardar* y luego *OK*.

**Para generar los objetos de la base de datos desde ERWIN:**

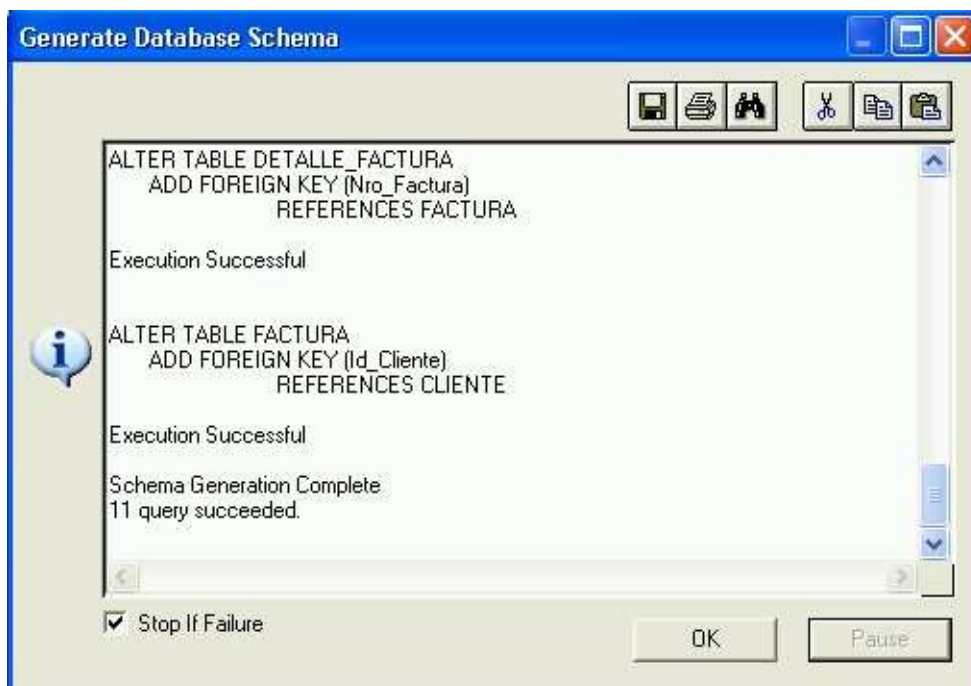
Antes de proceder a generar los objetos de la base de datos, se debe revisar el esquema de generación para seleccionar las sentencias SQL a ejecutar.

1. En la lista *SQL Server Schema Generation* de la ficha *Options*, seleccione *Schema*. Luego, en la lista *Schema Options* a la derecha quítele la marca *check* a todas las opciones que las tengan:
2. Ahora de la lista de la izquierda, seleccione *View* también quíteles la marca a las opciones seleccionadas y así con las demás opciones.
3. Las únicas opciones que deberán seleccionarse serán las siguientes:





4. Haga clic en el botón *Preview*. Note que algunas sentencias SQL han sido eliminadas del procedimiento original.
5. Para generar los objetos de la base de datos utilizando la conexión establecida, haga clic en el botón **Generate**.
6. La ventana siguiente le mostrará un reporte con el resultado de la generación. En este caso, todas las sentencias se ejecutaron sin problemas.



7. Haga clic en OK.



**Revisión del servidor *SQL* para comprobar la creación de los objetos:**





1. Cambie a *SQL Server 2008*.
2. Haga clic sobre la base de datos *Venta* utilizando el botón secundario del ratón.
3. Del menú contextual ejecute la opción *Refresh*.
4. Expanda *Venta* y haga doble clic sobre *Tablas*.



## ACTIVIDADES PROPUESTAS

1. Cree el modelo físico y lógico de la base de datos VENTAS en *ERWIN*.

## Resumen

-  Las vistas permiten redefinir la estructura de una base de datos proporcionando a cada usuario una vista personalizada de la estructura y los contenidos de la base de datos.
-  Una vista es una tabla virtual definida mediante una consulta. La vista parece contener filas y columnas de datos, al igual que una tabla real, pero los datos visibles a través de la vista son los resultados de la consulta.
-  Una vista puede ser un subconjunto simple fila / columna de una única tabla. Puede mostrar resultados obtenidos de funciones de agrupamiento o puede extraer los datos de dos o más tablas.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 <http://technet.microsoft.com/es-es/library/ms187956.aspx>

Tutorial para la creación de vistas

**UNIDAD DE  
APRENDIZAJE****5****SEMANA****13**

## **PROGRAMACIÓN AVANZADA EN SQL SERVER 2008**

---

### **LOGRO DE LA UNIDAD DE APRENDIZAJE**

Al terminar la unidad, los alumnos implementan instrucciones *SQL* y de programación mediante procedimientos almacenados, funciones para optimizar las operaciones en la base de datos y desencadenadores para optimizar las operaciones (*insert*, *delete* y *update*) de registros en una base de datos.

### **TEMARIO**

1.1. Creación de procedimientos almacenados.

### **ACTIVIDADES PROPUESTAS**

- Emplean herramientas de control de flujo para desarrollar programas.
- Utilizan los programas desarrollados desde un procedimiento almacenado creado.
- Ejecutan los procedimientos almacenados y analizan los resultados.

## PROGRAMACIÓN TRANSACT SQL

*Transact/SQL* permite agrupar una serie de instrucciones como lote, ya sea de forma interactiva o desde un archivo operativo. También, se puede utilizar estructuras de control de flujo por *Transact/SQL* para conectar las instrucciones utilizando estructuras de tipo de programación.

*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.

### 1. PROCEDIMIENTOS ALMACENADOS

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

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 de *SQL* de forma dramática. Además los procedimientos almacenados pueden ejecutarse en otro *SQL Server* si el servidor del usuario y el remoto están configurados para permitir *logins* remotos.

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.

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***. También, puede utilizar el nombre del procedimiento almacenado solo, siempre que sea la primera palabra de una instrucción o lote.

## 2. USO DEL LENGUAJE DE CONTROL DE FLUJO

El lenguaje de control de flujo se puede utilizar con instrucciones interactivas, en lotes y en procedimientos almacenados. El control de flujo y las palabras clave relacionadas y sus funciones son:

Control de flujo y palabras clave relacionadas	
Palabra Clave	Función
<i>IF</i>	Define una ejecución condicional
<i>..ELSE</i>	Define una ejecución alternativa cuando la condición <i>if</i> es falsa
<i>BEGIN</i>	Comienzo de un bloque de instrucciones
<i>...END</i>	Final de un bloque de instrucciones
<i>WHILE</i>	Repite la ejecución de instrucciones mientras la condición es verdadera
<i>BREAK</i>	Salida del final del siguiente bucle <i>while</i> más exterior
<i>...CONTINUE</i>	Reinicio del bucle <i>while</i>
<i>DECLARE</i>	Declara variables locales
<i>RETURN</i>	Salida de forma incondicional
<i>PRINT</i>	Imprime un mensaje definido por el usuario o una variable local en la pantalla del usuario
<i>/*COMENTARIO*/</i>	Inserta un comentario en cualquier punto de una instrucción <i>SQL</i>
<i>--COMENTARIO</i>	Inserta una línea de comentario en cualquier punto de una instrucción <i>SQL</i>
<i>CASE</i>	Permite que se muestre un valor alternativo

## 3. SINTAXIS

### 3.1 Declaración de variables locales

Una variable es una entidad a la que se asigna un valor. Este valor puede cambiar durante el lote o el procedimiento almacenado donde se utiliza la variable. *SQL Server* tiene dos tipos de variables: locales y globales. Las variables locales están definidas por el usuario, mientras que las variables globales las suministra el sistema y están predefinidas.

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.

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 o un tipo de dato suministrado por el sistema distinto de *text*, *image* o *sysname*.

**DECLARE @ NOMBRE\_DE\_VARIABLE TIPODEDATO**

**[, @ NOMBRE\_DE\_VARIABLE TIPODEDATO ] ...**

**DECLARACIÓN:**

```

DECLARE @ APEPATER_USUA      VARCHAR(25)
DECLARE @ COD_EST             CHAR(06)
DECLARE @ PISO_DEP            INTEGER
DECLARE @ FEC_FIRMA           DATETIME

```

**ASIGNACIÓN**

**e) Mostrar el apellido paterno del Usuario cuyo código sea igual a 'U000001'.**

**Usando set y print:**

```

Declare @APEPATER_USUA      varchar(25)

-- Obtener el Dato
Set    @APEPATER_USUA = APEPATER_USUA
From  Usuario
Where COD_USUA = 'U000001'

-- Mostrar el Dato
Print  @APEPATER_USUA

```

**Usando Select:**

```

Declare @APEPATER_USUA      varchar(25)

-- Obtener el Dato
Select @APEPATER_USUA = APEPATER_USUA
From  Usuario
Where COD_USUA = 'U000001'

-- Mostrar el Dato
Select @APEPATER_USUA

```

**f) Mostrar el Haber Básico del inquilino cuyo código de usuario es igual a U000002**

```

Declare @HABER_BAS_INQ float

-- Obtener el Haber Básico
Select @HABER_BAS_INQ = HABER_BAS_INQ
From  INQUILINO
Where COD_USUA = 'U000002'

-- Mostrar el Dato
Select @HABER_BAS_INQ

```

**g) Mostrar el Número de departamentos que tiene el Edificio cuyo código es igual a E00001**

```

Declare @NUM_DEPARTAMENTOS integer

```

```
-- Obtener el Número de Departamentos
Select @NUM_DEPARTAMENTOS = count(*)
From EDIFICIOS E inner Join DEPARTAMENTOS D
on E.COD_EDIF = D.COD_EDIF
Where E.COD_EDIF = 'E00001'
```

```
-- Mostrar el Dato
Select @NUM_DEPARTAMENTOS
```

### 3.2 Declaración de variables globales

Las variables globales son variables predefinidas suministradas por el sistema. Se distinguen de las variables locales por tener dos símbolos “@”.

Estas son algunas variables globales del servidor:

Variables globales de SQL Server	
Variable	Contenido
@@error	Contiene 0 si la última transacción se ejecutó de forma correcta; en caso contrario, contiene el último número de error generado por el sistema. La variable global @@error se utiliza generalmente para verificar el estado de error de un proceso ejecutado.
@@identity	Contiene el último valor insertado en una columna <b>IDENTITY</b> mediante una instrucción <i>insert</i>
@@Version	Devuelve la Versión del SQL Server
@@SERVERNAME	Devuelve el Nombre del Servidor
@@LANGUAGE	Devuelve el nombre del idioma en uso

### 3.3 CREATE / ALTER / DROP PROCEDURE

Sintaxis para crear un procedimiento almacenado:

```
CREATE PROCEDURE NOMBRE_PROCEDIMIENTO
AS
CONSULTA_SQL
```

Sintaxis para modificar un procedimiento almacenado:

```
ALTER PROCEDURE NOMBRE_PROCEDIMIENTO
AS
CONSULTA_SQL
```

### Sintaxis para eliminar un procedimiento almacenado:

```
DROP PROCEDURE NOMBRE_PROCEDIMIENTO
```

### Ejemplos:

- a) Cree un procedimiento almacenado que muestre todos los edificios.

```
CREATE PROCEDURE LISTAR_EDIFICIOS
AS
Select *
From EDIFICIOS
```

Como se aprecia, el procedimiento mostrado no tiene parámetros de entrada y para ejecutarlo deberá usar una de las siguientes sentencias:

```
EXECUTE LISTAR_EDIFICIOS
```

```
EXEC LISTAR_EDIFICIOS
```

```
LISTAR_EDIFICIOS
```

Ahora veamos cómo se define un procedimiento almacenado con parámetros.

- b) Cree un procedimiento almacenado que permita buscar los datos de un Usuario. El procedimiento tiene los siguientes parámetros de entrada: Nombre y Apellido Paterno.

```
CREATE PROCEDURE BUSCAR_USUARIO
@NOM_USUA          varchar(25),
@APEPATER_USUA     varchar(25)
AS
Select *
From USUARIO
Where NOM_USUA = @NOM_USUA
      and APEPATER_USUA = @APEPATER_USUA
```

Para ejecutarlo

```
EXECUTE BUSCAR_USUARIO 'ENRIQUE','VERA'
```

Como se aprecia el **procedimiento almacenado** recibe dos parámetros y realiza una búsqueda exacta dado el Nombre y Apellido del Usuario.



- c) Modifique el procedimiento anterior para que la búsqueda del Usuario sea por una secuencia de caracteres y no en forma exacta.

```
ALTER PROCEDURE BUSCAR_USUARIO
@NOM_USUA          varchar(25),
@APEPATER_USUA     varchar(25)
AS
Select *
From  USUARIO
Where NOM_USUA like '%' + @NOM_USUA + '%'
      and APEPATER_USUA like '%' + @APEPATER_USUA + '%'
```

Para ejecutarlo

```
EXECUTE BUSCAR_USUARIO 'EN','VE'
```

La sentencia **ALTER PROCEDURE** permite modificar el contenido del procedimiento almacenado y la sentencia **LIKE** permite hacer la búsqueda por una secuencia de caracteres.

- d) Cree un procedimiento almacenado que muestre el código, nombre y dirección del edificio así como el código, número de ambiente y piso del departamento. El procedimiento recibirá como parámetro de entrada el código del edificio.  
Se debe considerar que el parámetro de entrada tendrá un valor por defecto igual a vacío.

```
Create procedure BUSCAR_DEPARTAMENTOS
@COD_EDIF char(6) = ''
AS
Select E.COD_EDIF,
E.NOM_EDIF,
E.DIRECC_EDIF,
D.COD_DEP,
D.NUM_AMB_DEP,
D.PISO_DEP
From  EDIFICIOS E inner join DEPARTAMENTOS D
on E.COD_EDIF = D.COD_EDIF
Where E.COD_EDIF = @COD_EDIF
```

Para ejecutarlo

```
EXECUTE BUSCAR_DEPARTAMENTOS 'E00001'
```

En el parámetro de entrada **@COD\_EDIF**, se ha definido un valor por defecto igual a vacío. El procedimiento almacenado tomaría este valor cuando no se defina un valor al momento de ejecutarse.

```
EXECUTE BUSCAR_DEPARTAMENTOS
```

- e) Elimine el procedimiento almacenado **BUSCAR\_USUARIO**

```
Drop procedure BUSCAR_USUARIO
```

### 3.4 IF...ELSE

La palabra clave **IF** con o sin la compañía **ELSE** se utiliza para introducir una condición que determina si se ejecutará la instrucción siguiente. La instrucción SQL se ejecuta si la condición se cumple, es decir, si devuelve *TRUE* (verdadero)

La palabra clave **ELSE** introduce una instrucción SQL alternativa que se ejecuta cuando la condición **IF** devuelva *FALSE* (falso).

A continuación se presenta su sintaxis:

```

IF
    EXPRESION_BOOLEANA
        EXPRESION_SQL
[ ELSE
    [ IF EXPRESION_BOOLEANA ]
        EXPRESION_SQL ]

```

**Ejemplos:**

- a) Cree un procedimiento almacenado el cual reciba como parámetros dos códigos de edificios y el programa deberá evaluarlos e indicar que edificio tiene la mayor, menor o igual cantidad de departamentos.

```

Create procedure VERIFICAR_NUMERO_DEPARTAMENTOS
    @COD_EDIF1      char(6),
    @COD_EDIF2      char(6)
AS

```

-- Variables

```

Declare      @NUM_DEPARTAMENTOS1 integer
Declare      @NUM_DEPARTAMENTOS2 integer

```

```

-- Obtener el número de departamentos del Edificio 1
Select  @NUM_DEPARTAMENTOS1 = count(*)
From    EDIFICIOS E inner join DEPARTAMENTOS D
on E.COD_EDIF = D.COD_EDIF
Where E.COD_EDIF = @COD_EDIF1

```

```

-- Obtener el número de departamentos del Edificio 2
Select  @NUM_DEPARTAMENTOS2 = count(*)
From    EDIFICIOS E inner join DEPARTAMENTOS D
on E.COD_EDIF = D.COD_EDIF
Where E.COD_EDIF = @COD_EDIF2

```

```
-- Verificar el Número de Departamentos
If @NUM_DEPARTAMENTOS1 = @NUM_DEPARTAMENTOS2
Select 'Los Edificios tienen la misma cantidad de departamentos'
Else
If @NUM_DEPARTAMENTOS1 > @NUM_DEPARTAMENTOS2
    Select 'El Edificio 1 tiene la mayor cantidad de departamentos'
Else
    Select 'El Edificio 2 tiene la mayor cantidad de departamentos'
```

Para ejecutarlo

```
EXECUTE VERIFICAR_NUMERO_DEPARTAMENTOS 'E00001','E00002'
```

### 3.5 RETURN

La palabra clave **RETURN** sale de un lote o procedimiento de forma incondicional y puede usarse en cualquier momento de un lote o procedimiento.

**RETURN**

### 3.6 BEGIN...END

Las palabras clave **BEGIN** y **END** se utilizan para englobar una serie de instrucciones a fin de que sean tratadas como una unidad por las estructuras de control de flujo como *if...else*. Una serie de instrucciones por *begin* y *end* se denomina bloque de instrucciones.

La sintaxis de *begin...end* es:

```
BEGIN
    BLOQUE DE INSTRUCCIONES
END
```

**Ejemplos:**

- a) Cree un procedimiento almacenado que permita el ingreso de datos de un usuario. El procedimiento tendrá los siguientes parámetros de entrada: código, nombre, apellido paterno, apellido materno, fecha de nacimiento, fecha de registro, tipo de documento, número de documento y código de estado.

Considere las siguientes condiciones antes de ingresar un usuario:

- El código del usuario debe ser único
- No deberá ingresar usuarios cuya edad sea menor de 18 años

**CREATE PROCEDURE INSERTA\_USUARIO**

```
@COD_USUA      char(6),
@NOM_USUA      varchar(25),
@APEPATER_USUA varchar(25),
@PEMATER_USUA  varchar(25),
@FEC_NAC_USUA  datetime,
@FEC_REG_USUA  datetime,
@TIPO_DOC_USUA varchar(20),
@NUM_DOC_USUA  char(8),
@COD_EST       char(6)
```

**AS**

```
-- Verificar que el Usuario sea único
If Exists(
    Select *
    From Usuario
    Where COD_USUA = @COD_USUA
)
Begin
    Select 'Ya existe un Usuario con el mismo Código'
    Return
End

-- Verificar la Edad del Usuario
If datediff(yy, @FEC_NAC_USUA, getdate()) < 18
Begin
    Select 'La edad del Usuario debe ser mayor a 18 años'
    Return
End

-- Insertar los datos del Usuario
Insert into USUARIO (
    COD_USUA,
    NOM_USUA,
    APEPATER_USUA,
    PEMATER_USUA,
    FEC_NAC_USUA,
    FEC_REG_USUA,
    TIPO_DOC_USUA,
    NUM_DOC_USUA,
    COD_EST
)
Values (
    @COD_USUA,
```

```

@NOM_USUA,
@APEPATER_USUA,
@APEMATER_USUA,
@FEC_NAC_USUA,
@FEC_REG_USUA,
@TIPO_DOC_USUA,
@NUM_DOC_USUA,
@COD_EST
)

```

- b) Cree un procedimiento almacenado que permita modificar los datos de un inquilino. El procedimiento tendrá los siguientes parámetros de entrada: código de usuario, nombre de aval, apellido del aval, Haber básico, estado civil y lugar de trabajo del inquilino. Así mismo, sólo se podrán modificar aquellos inquilinos cuyo Haber básico sea menor a 600 soles.

#### CREATE PROCEDURE MODIFICA\_INQUILINO

```

@COD_USUA      char(6),
@NOM_AVAL_INQ  varchar(30),
@APELL_AVAL    char(30),
@HABER_BAS_INQ float,
@EST_CIVIL_INQ char(1),
@LUG_TRAB_INQ  varchar(50)

```

#### AS

-- Variables

Declare @HABER\_BAS\_INQ\_ACTUAL float

-- Obtener el Haber Básico Actual del Inquilino

```

Select @HABER_BAS_INQ_ACTUAL = HABER_BAS_INQ
From INQUILINO

```

Where COD\_USUA = @COD\_USUA

-- Verificar el Haber Básico del Inquilino

If @HABER\_BAS\_INQ\_ACTUAL > 600

Begin

Select 'El Haber Básico del Inquilino debe ser menor a 600 soles'

Return

End

-- Actualizar los Datos

Update INQUILINO

```

Set COD_USUA = @COD_USUA,
    NOM_AVAL_INQ = @NOM_AVAL_INQ,
    APELL_AVAL = @APELL_AVAL,
    HABER_BAS_INQ = @HABER_BAS_INQ,
    EST_CIVIL_INQ = @EST_CIVIL_INQ,
    LUG_TRAB_INQ = @LUG_TRAB_INQ

```

Where COD\_USUA = @COD\_USUA

- c) Cree un procedimiento almacenado que permita eliminar los datos de un Contrato. El procedimiento tendrá como parámetro de entrada el código del contrato.

Tome en cuenta las siguientes consideraciones antes de eliminar un producto:

- Sólo se podrán eliminar contratos que hayan sido generados en los últimos cuatro meses.
- Deberá verificar que el contrato no tenga detalle de contrato.

**CREATE PROCEDURE** ELIMINAR\_CONTRATO

@COD\_CONT char(6)

**AS**

--Variables

Declare @FEC\_FIRMA datetime

-- Obtener la Fecha de Firma

Select @FEC\_FIRMA = FEC\_FIRMA

From CONTRATO

Where COD\_CONT = @COD\_CONT

-- Verificar la Antigüedad del Contrato

If datediff(mm,@FEC\_FIRMA,getdate())>4

Begin

Select 'La antigüedad del Contrato no debe ser mayor a 4 meses'

Return

End

-- Verificar que el Contrato no se encuentre asociado algún Detalle

If Exists(

Select \*

From DETALLECONTRATO

Where COD\_CONT = @COD\_CONT

)

Begin

Select 'El Contrato tiene su Detalle por lo tanto no puede ser eliminado'

Return

End

-- Eliminar los Datos

Delete from CONTRATO

Where COD\_CONT = @COD\_CONT

### 3.7 WHILE

**WHILE** se utiliza para definir una condición para la ejecución repetida de una instrucción o un bloque de instrucciones. Las instrucciones se ejecutan reiteradamente siempre que la condición especificada es verdadera.

La sintaxis de *WHILE* es:

```
WHILE BOOLEAN_EXPRESION
    EXPRESION_SQL
```

**BREAK** y **CONTINUE** controlan el funcionamiento de las instrucciones dentro de un bucle *while*. **BREAK** permite salir del bucle *while*. **CONTINUE** hace que el bucle *while* se inicie de nuevo.

La sintaxis de **BREAK** y **CONTINUE** es:

```
WHILE BOOLEAN_EXPRESION
BEGIN
    EXPRESION_SQL
    [ EXPRESION_SQL ] ...
    BREAK
    [ EXPRESION_SQL ] ...
    CONTINUE
    [ EXPRESION_SQL ] ...
END
```

### 3.8 CASE

La función *CASE* es una expresión especial de *Transact SQL* que permite que se muestre un valor alternativo dependiendo de una columna. Este cambio es temporal, con lo que no hay cambios permanentes en los datos.

La función *CASE* está compuesta de:

- La palabra *CASE*
- El nombre de la columna que se va transformar
- Cláusulas *WHEN* que se especifican las expresiones que se van a buscar y cláusulas *THEN* que especifican las expresiones que las van a reemplazar
- La palabra *END*
- Una cláusula *AS* opcional que define un alias de la función *CASE*

**Ejemplo:**

```

Select  COD_USUA,
        NOM_USUA,
        APEPATER_USUA,
        APEMATER_USUA,
        Case Month(FEC_NAC_USUA)
            When 1 Then 'Enero'
            When 2 Then 'Febrero'
            When 3 Then 'Marzo'
            When 4 Then 'Abril'
            When 5 Then 'Mayo'
            When 6 Then 'Junio'
            When 7 Then 'Julio'
            When 8 Then 'Agosto'
            When 9 Then 'Septiembre'
            When 10 Then 'Octubre'
            When 11 Then 'Noviembre'
            When 12 Then 'Diciembre'
        End As Mes_Nacimiento
From    USUARIO

```

**3.9 Crear un sp que permita incrementar el haber básico de un inquilino en función de su estado civil (20% para casados o viudos y 10% para los solteros y divorciados). Asimismo, si el inquilino tiene una comisión aumentar su haber con ese monto.**

**Create Procedure splncHaberInquilino**

```

@Inquilino Char(6),
@Comision Numeric(10,2) = Null --Parametro con valor por defecto
As
Begin -- Inicio del sp
    -- Validar que el codigo y la descripcion no sean nulas
    If( @Inquilino Is Null )
    Begin
        Print 'El código del inquilino no puede ser NULO!!!'
        Return
    End

    -- Validar si el inquilino existe
    If Not Exists(Select * From Inquilino Where Cod_Usua = @Inquilino)
    Begin
        Print 'Actualizacion cancelada. El inquilino no existe.'
        Return
    End

    -- Variables locales para el proceso
    Declare @HaberBasico Int
    Declare @EstadoCivil Char(1)
    Declare @IncrementoPorEstadoCivil Int

    -- Determinar valores del inquilino
    Select
        @HaberBasico = Haber_Bas_Inq,

```



```
        @EstadoCivil = Est_Civil_Inq
    From Inquilino
    Where Cod_Usua = @Inquilino

    -- Determinar porcentaje de incremento
    If (@EstadoCivil In ('C','V'))
        Set @IncrementoPorEstadoCivil = @HaberBasico * 0.20
    Else
        Set @IncrementoPorEstadoCivil = @HaberBasico * 0.10

    -- Eliminar el departamento
    Update Inquilino
    Set Haber_Bas_Inq = Haber_Bas_Inq + @IncrementoPorEstadoCivil +
    IsNull(@Comision,0)
    Where Cod_Usua = @Inquilino
End -- Fin del splncHaberInquilino
```

#### **-- Ejecutar el splncHaberInquilino**

```
Select * From Inquilino Where Cod_Usua = 'USU002'
Exec splncHaberInquilino 'USU002'
```

```
Select * From Inquilino Where Cod_Usua = 'USU002'
```

```
Select * From Inquilino Where Cod_Usua = 'USU002'
Exec splncHaberInquilino 'USU002', 1
```

```
Select * From Inquilino Where Cod_Usua = 'USU002'
```

### 3.10 Procedimiento almacenado para páginas de datos

Cuando se tiene que mostrar una serie numerosa de resultados en una página *web*, es necesario mostrar estos de poco en poco. Para ello, se utiliza la paginación, dividir los resultados en grupos y mostrarlos en distintas páginas. Una parte fundamental es la consulta a la base de datos, la cual tiene que ser eficiente.

#### 3.10.1 Función ROW\_NUMBER ( )

Devuelve el número secuencial de una fila de una partición de un conjunto de resultados, comenzando con 1 para la primera fila de cada partición.

##### Sintaxis:

```
ROW_NUMBER ( ) OVER ( [ <partition_by_clause> ] <order_by_clause> )
```

##### Argumentos

###### **<partition\_by\_clause>**

Divide el conjunto de resultados generado por la cláusula FROM en particiones a las que se aplica la función ROW\_NUMBER.

###### **<order\_by\_clause>**

Determina el orden en el que se asigna el valor ROW\_NUMBER a las filas de una partición. Un entero no puede representar una columna cuando se usa la cláusula <order\_by\_clause> en una función de categoría.

##### Tipos de valor devueltos

bigint

##### Nota

La cláusula ORDER BY determina la secuencia en la que se asigna a las filas el ROW\_NUMBER único correspondiente en una partición especificada.

#### 3.10.2 Función CEILING()

Devuelve el entero más pequeño mayor o igual que la expresión numérica especificada.

##### Sintaxis

```
CEILING ( numeric_expression )
```

### 3.10.3 Ejemplo1 (Versión no recomendada)

```

Create Procedure ListaDptosPaginacion_Ejemplo1
    @PageSize int,          -- Tamaño de la pág(Cantidad de registros a devolver)

    @PageNumber int        -- Página que se desea devolver
As
Begin

-- Seleccionar los datos que se desean observar pero agregándoles el número de
-- fila
    Select Top ( @PageSize)
        * -- Obtiene todos los campos de la consulta cruda (RawData)
    From (
        Select
            Row_Number() Over (Order By Dep.Cod_Edif, Dep.Cod_Dep)-1
As RowNum, -- Agrega el número de fila a cada registro
        Dep.Cod_Edif As IDEdif,
        Dep.Cod_Dep As IDDpto,
        Edi.Nom_Edif As NombreEdif,
        Dep.Area_Total_Dep As AreaTotalEdif,
        Dep.Area_Construida_Dep As AreaContruidaDpto,
        Dep.Precio_AlqXMes_Dep As PrecioAlquilerDpto
        From Departamentos As Dep
            Inner Join Edificios As Edi On Dep.Cod_Edif = Edi.Cod_Edif
        ) As RawData
    Where RowNum >= @PageSize * (@PageNumber - 1)
    Order By RowNum
End;

```

#### -- Ejecutar el procedimiento almacenado

---

```

Exec ListaDptosPaginacion_Ejemplo1 50, 1

```

### 3.10.4 Ejemplo2 (Versión recomendada)

```

Create Procedure ListaDptosPaginacion_Ejemplo2
    @PageSize int,          -- Tamaño de la página (Cantidad de registros a
                            -- devolver)
    @PageNumber int,        -- Página que se desea devolver
    @PageMax int output     -- Parámetro de salida que devolverá el total de
                            -- páginas de la consulta base
As
Begin

--Seleccionar los datos que se desean observar pero agregándoles el número de
--fila

Select *, -- Obtiene todos los campos de la consulta cruda (RawData)
        Row_Number() Over (Order By IDEdif, IDDpto)-1 As RowNum

```

```
-  
- Agrega el número de fila a cada registro  
  Into #Data -- Almacena el resultado en una tabla temporal  
  From (  
    Select  
      Dep.Cod_Edif As IDEdif,  
      Dep.Cod_Dep As IDDpto,  
      Edi.Nom_Edif As NombreEdif,  
      Dep.Area_Total_Dep As AreaTotalEdif,  
      Dep.Area_Construida_Dep As AreaContruidaDpto,  
      Dep.Precio_AlqXMes_Dep As PrecioAlquilerDpto  
    From Departamentos As Dep  
      Inner Join Edificios As Edi On Dep.Cod_Edif = Edi.Cod_Edif  
    ) As RawData  
  
  -- Obtenemos los registros de la pagina solicitada  
  Select  
    *  
  From #Data  
  Where Ceiling(RowNum/@PageSize)+1 = @PageNumber  
  Order By RowNum  
  
  -- Obtenemos el número de páginas de la data cruda  
  Select @PageMax =(Max(RowNum)/@PageSize)+1 From #Data;  
End;
```

#### **-- Ejecutar el procedimiento almacenado**

-----

```
Declare @pages Int  
Exec ListaDptosPaginacion_Ejemplo2 50, 1, @pages Output  
Print @pages
```

## ACTIVIDADES PROPUESTAS





En la base de datos **VENTAS**:

1. Cree un procedimiento almacenado que permita buscar los datos de un producto. El procedimiento tiene como parámetro de entrada el código del producto.
2. Cree un procedimiento almacenado que muestre los datos de las boletas con su monto total por boleta. El procedimiento recibirá como parámetro de entrada el código del vendedor que emitió dicha boleta.
3. Cree un procedimiento almacenado que muestre el detalle de una boleta. El procedimiento recibirá como parámetro de entrada el código de dicha boleta.
4. Cree un procedimiento almacenado que permita ingresar datos a la tabla PRODUCTO y que reciba como parámetros de entrada los datos para cada campo. Considere que el código del producto es un valor generado por el procedimiento almacenado y tiene la forma tal como 'PRO001'. Adicionalmente, verifique que no se ingresen dos productos con la misma descripción, y si esto ocurriera, muestre un mensaje que indique 'Ya existe un producto con la misma descripción. Los mensajes de error deberán ser invocados desde otro procedimiento almacenado.
5. Cree un procedimiento almacenado que permita el ingreso de datos de un empleado. El procedimiento tendrá como parámetros de entrada todos los campos de la tabla empleado.  
Tome en cuenta las siguientes condiciones antes de ingresar un empleado:
  - El código del empleado debe ser único.
  - No deberá permitir ingresar empleados cuya edad sea menor a 18 años.
  - No deberá permitir ingresar empleados con el *email* repetido.

Si existieran errores se mostrarán los mensajes correspondientes.
6. Cree un procedimiento almacenado que permita actualizar el stock mínimo de un producto. El procedimiento tendrá los siguientes parámetros de entrada: código del producto y el stock mínimo a actualizar.  
Considere las siguientes condiciones antes de modificar un producto:
  - No permitir registrar *stocks* negativos.
  - Sólo puede modificarse aquellos productos con un stock actual menor a 100 unidades.

Si existieran errores se mostrarán los mensajes correspondientes.
7. Cree un procedimiento almacenado que permita eliminar un distrito. El procedimiento tendrá como parámetro de entrada el código del distrito. Un distrito será eliminado siempre y cuando no este asociado a algún empleado o cliente.

## Resumen

-  En esta sección se han explicado las extensiones de programación que van más allá de las implementaciones típicas de SQL y que hacen de Transact SQL un lenguaje de programación especializado.
-  Las instrucciones de Transact/SQL pueden agruparse en procesos o lotes, permanecer en la base de datos, ejecutarse repetidamente en forma de procedimientos almacenados.
-  Los procedimientos almacenados de Transact/SQL pueden ser bastante complejos y pueden llegar a ser una parte importante del código fuente de su aplicación.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:



<http://technet.microsoft.com/es-es/library/ms187926.aspx>

Tutorial para la creación de procedimientos almacenados



<http://www.devjoker.com/contenidos/Tutorial-de-Transact-SQL/238/Procedimientos-almacenados-en-Transact-SQL.aspx>

Tutorial para la creación de procedimientos almacenados

UNIDAD DE  
APRENDIZAJE**6**

SEMANA

**14**

# PROGRAMACIÓN AVANZADA EN SQL SERVER 2008

---

## LOGRO DE LA UNIDAD DE APRENDIZAJE

Al terminar la unidad, los alumnos implementan instrucciones SQL y de programación mediante procedimientos almacenados, funciones para optimizar las operaciones en la base de datos desencadenadores para optimizar las operaciones (*insert*, *delete* y *update*) de registros en una base de datos.

## TEMARIO

- 1.1 Creación de funciones escalares
- 1.2 Creación de desencadenadores

## ACTIVIDADES PROPUESTAS

- Comprenden la necesidad del empleo de funciones en el manejo de una base de datos
- Entienden la necesidad del uso del Trigger (desencadenador) para validar acciones en la base de datos

## 1. FUNCIONES

Lo más interesante para los programadores del *SQL* es la posibilidad de hacer funciones definidas por el usuario. Estas funciones del *SQL* solucionarán los problemas de reutilización del código y dará mayor flexibilidad al programar las consultas de *SQL*.

### TIPOS DE FUNCIONES

El servidor 2008 del *SQL* utiliza tres tipos de funciones: **las funciones escalares, tabla en línea y funciones de tabla de multi sentencias**. Los tres tipos de funciones aceptan parámetros de cualquier tipo excepto el *rowversion*. Las funciones escalares devuelven un solo valor, y las funciones tabla en línea y multisentencias devuelven un tipo de dato tabla. En este curso nos centraremos en las funciones escalares.

#### 1.1 Funciones Escalares

Las funciones escalares devuelven un tipo de datos tales como ***int***, ***money***, ***varchar***, ***float***, etc. Pueden ser utilizadas en cualquier lugar incluso incorporado dentro de sentencias *SQL*. La sintaxis para una función escalar es la siguiente:

```
CREATE FUNCTION [ owner_name. ] function_name
(
    [ { @parameter_name [ AS ] data_type } [ ,...n ] ]
)
    RETURNS data_type
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
```



## Ejemplos

- a) Cree una función que permita elevar al cubo un número natural.

```
CREATE FUNCTION ElevarCubo(  
    @Numero float  
)  
    RETURNS float  
AS  
BEGIN  
    RETURN(@Numero * @Numero * @Numero)  
END
```

**Para ejecutarlo:**

```
Declare    @Resultado  float  
  
-- Obtener el Valor  
select @Resultado = ElevarCubo(10)  
  
-- Mostrar el Resultado  
select @Resultado
```

- b) Muestre los datos del edificio así como el número de departamentos que tiene cada edificio.

Este ejercicio podrá ser resuelto usando *GROUP BY*, sin embargo resolveremos este problema usando funciones.

```
CREATE FUNCTION ObtenerNumeroDepartamentos(  
    @COD_EDIF char(06)  
)  
    RETURNS Integer  
AS  
BEGIN  
    -- Variables  
    Declare @NumeroDepartamentos integer  
  
    -- Obtener el Número de Departamentos  
    Select @NumeroDepartamentos = count(*)  
    From    DEPARTAMENTOS  
    Where COD_EDIF = @COD_EDIF  
  
    -- Retornar Valor  
    Return (@NumeroDepartamentos)  
  
END  
  
-- Mostrar Consulta  
Select COD_EDIF,  
       NOM_EDIF,  
       DIRECC_EDIF,  
       ObtenerNumeroDepartamentos(COD_EDIF)  
From EDIFICIOS
```

**c) Muestre los datos completos del contrato.**

```

CREATE FUNCTION ObtenerNombreUsuario(
    @COD_USUA char(06)
)
    RETURNS Varchar(100)
AS
BEGIN
    -- Variables
    Declare @NombreCompleto as varchar(100)

    -- Obtener el Nombre Completo del Usuario
    Select @NombreCompleto = NOM_USUA + ' ' + APEPATER_USUA + ' ' +
    APEMATER_USUA
    From USUARIO
    Where COD_USUA = @COD_USUA

    -- Retornar Valor
    Return (@NombreCompleto)

END

-- Mostrar los Datos del Contrato
Select COD_CONT,
    PROP_COD_USUA as CodigoPropietario,
    ObtenerNombreUsuario(PROP_COD_USUA)as NombrePropietario,
    INQ_COD_USUA as CodigoInquilino,
    ObtenerNombreUsuario(INQ_COD_USUA) as NombreInquilino,
    FEC_FIRMA, REFERENCIA
From CONTRATO
GO

```

## **2. TRIGGERS O DISPARADORES**

Los disparadores pueden usarse para imponer la integridad de referencia de los datos en toda la base de datos. Los disparadores también permiten realizar cambios “en cascada” en tablas relacionadas, imponer restricciones de columna más complejas que las permitidas por las reglas, compara los resultados de las modificaciones de datos y llevar a cabo una acción resultante.

### **2.1 DEFINICIÓN DISPARADOR**

Un disparador es un tipo especial de procedimiento almacenado que **se ejecuta cuando se insertan, eliminan o actualizan datos** de una tabla especificada. Los disparadores pueden ayuda a mantener la integridad de referencia de los datos conservando la consistencia entre los datos relacionados lógicamente de distintas tablas. Integridad de referencia significa

que los valores de las llaves primarias y los valores correspondientes de las llaves foráneas deben coincidir de forma exacta.

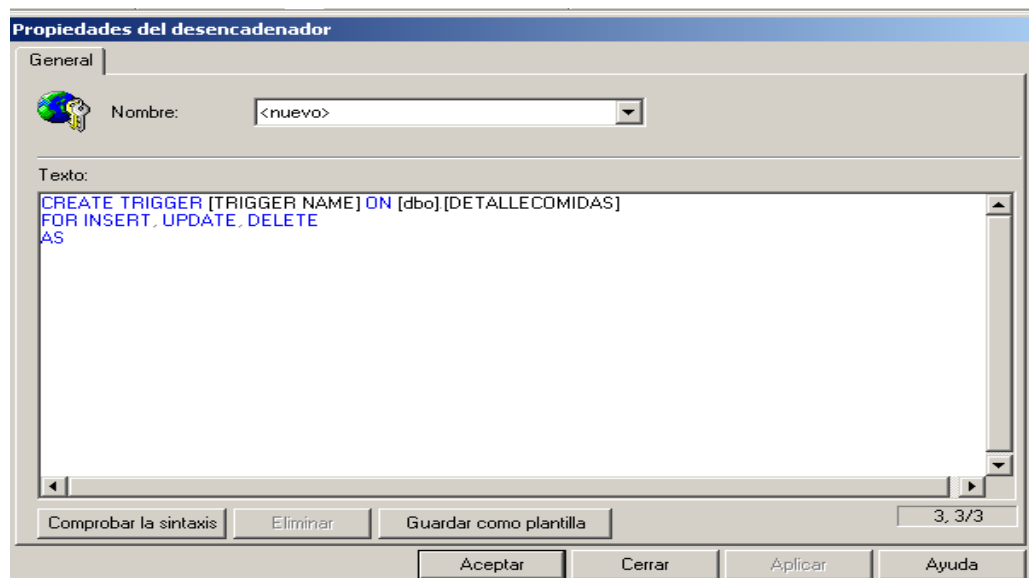
La principal ventaja de los disparadores es que son automáticos: funcionan cualquiera sea el origen de la modificación de los datos, una introducción de datos por parte de un empleado o una acción de una aplicación. Cada disparador es específico de una o más operaciones de modificación de datos, **UPDATE, INSERT o DELETE**. El disparador se ejecuta una vez por cada instrucción.

## 2.2 CREACIÓN DE DISPARADORES

Un disparador es un objeto de la base de datos. Cuando se crea un disparador, se especifica la tabla y los comandos de modificación de datos que deben “disparar” o activar el disparador. Luego, se indica la acción o acciones que debe llevar a cabo un disparador.

A continuación se muestra un ejemplo sencillo. Este disparador imprime un mensaje cada vez que alguien trata de insertar, eliminar o actualizar datos de la tabla Usuario

```
CREATE TRIGGER TX_USUARIO
ON USUARIO
FOR INSERT, UPDATE, DELETE
AS
PRINT "UD. ACABA DE MODIFICAR VALORES EN LA TABLA USUARIO"
```



Para modificar el *trigger*, se utiliza la siguiente sintaxis:

```
ALTER TRIGGER TX_USUARIO
ON USUARIO
FOR INSERT, UPDATE, DELETE
AS
PRINT "UD. ACABA DE MODIFICAR LOS DATOS DEL USUARIO"
```

Para borrar un trigger, se utiliza la siguiente sintaxis:

```
DROP TRIGGER TX_USUARIO
```

## 2.3 Funcionamiento de los Disparadores

### Ejemplo de Disparador de Inserción

Cuando se inserta una nueva fila en una tabla, *SQL Server* inserta los nuevos valores en la tabla ***INSERTED*** el cual es una tabla del sistema. Esta tabla toma la misma estructura del cual se originó el *trigger*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios.

- a) Cree un *trigger* que permita insertar los datos de un Usuario siempre y cuando el nombre y apellido paterno del usuario sean únicos

```
CREATE TRIGGER TX_USUARIO_INSERTA
ON USUARIO
FOR INSERT
AS
IF (Select count (*)
    From INSERTED,
         USUARIO
    Where INSERTED.NOM_USUA = USUARIO.NOM_USUA
        AND INSERTED.APEPATER_USUA =
USUARIO.APEPATER_USUA
) > 1
Begin
    Rollback transaction

    Select 'El Usuario existe en la base de datos, por favor verifique sus
datos'
End
Else
    Select 'El Usuario fue ingresado en la base de datos'
```

En este ejemplo, verificamos el número de usuarios que tienen el mismo nombre y apellido y de encontrarse más de un usuario no se deberá permitir ingresar los datos del usuario.

Este disparador imprime un mensaje si la inserción se revierte y otro si se acepta.

### Ejemplo de Disparador de Eliminación

Cuando se elimina una fila de una tabla, *SQL Server* inserta los valores que fueron eliminados en la tabla **DELETED** el cual es una tabla del sistema. Está tabla toma la misma estructura del cual se origino el *trigger*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios. En este caso la reversión de los cambios significará restaurar los datos eliminados.

- b) Cree un *trigger* el cual permita eliminar Inquilinos cuyo haber básico sea menor a 400 soles. De eliminarse algún inquilino que no cumpla con dicha condición la operación no deberá ejecutarse.

```
CREATE TRIGGER TX_INQUILINO_ELIMINA
ON INQUILINO
FOR Delete
AS

IF (select HABER_BAS_INQ from deleted ) > 400
Begin
    Rollback transaction
    print 'El Haber Básico del Inquilino debe ser menor a 400 soles'
End
```

En este ejemplo, verificamos el haber básico del inquilino y de superar los 400 soles la operación deberá ser cancelada.

### Ejemplo de Disparador de Actualización

Cuando se actualiza una fila de una tabla, *SQL Server* inserta los valores que antiguos en la tabla **DELETED** y los nuevos valores los inserta en la tabla **INSERTED**. Usando estas dos tablas se podrá verificar los datos y ante un error podrían revertirse los cambios.

- c) Cree un *trigger* que valide que la fecha de nacimiento de un Usuario no sea mayor a 1990.

```
CREATE TRIGGER TX_USUARIO_ACTUALIZA
ON USUARIO
FOR UPDATE
AS
If (Select      year(FEC_NAC_USUA) From Inserted) > 1990
Begin
    -- Mostrar los Datos
    Select year(dt_fechanacimiento) as fecha_anterior From deleted
    Select year(dt_fechanacimiento) as fecha_nueva From Inserted
    Rollback transaction
```

```
print 'La fecha de nacimiento debe ser menor a 1990'  
End
```

## 2.4 Uso de INSTEAD OF

Los desencadenadores INSTEAD OF pasan por alto las acciones estándar de la instrucción de desencadenamiento: INSERT, UPDATE o DELETE. Se puede definir un desencadenador INSTEAD OF para realizar comprobación de errores o valores en una o más columnas y, a continuación, realizar acciones adicionales antes de insertar el registro. Por ejemplo, cuando el valor que se actualiza en un campo de tarifa de una hora de trabajo de una tabla de planilla excede un valor específico, se puede definir un desencadenador para producir un error y revertir la transacción, o insertar un nuevo registro en un registro de auditoría antes de insertar el registro en la tabla de planilla.

## 2.5 RESTRICCIONES DE LOS DISPARADORES

A continuación, se describen algunas limitaciones o restricciones impuestas a los disparadores por SQL Server:

- Una tabla puede tener un máximo de tres disparadores: uno de actualización, uno de inserción y uno de eliminación.
- Cada disparador puede aplicarse a una sola tabla. Sin embargo, un mismo disparador se puede aplicar a las tres acciones del usuario: *update*, *insert* y *delete*.
- No se puede crear un disparador en una vista ni en una tabla temporal, aunque los disparadores pueden hacer referencia a las vistas o tablas temporales.
- Los disparadores no se permiten en las tablas del sistema. Aunque no aparece ningún mensaje de error si se crea un disparador en una tabla del sistema, el disparador no se utilizará.

## 2.6 DISPARADOR y RENDIMIENTO

En términos de rendimiento, la sobrecarga de disparador es generalmente muy baja. El tiempo invertido en ejecutar un disparador se emplea principalmente para hacer referencia a otras tablas que pueden estar en memoria o en el dispositivo de base de datos.

Las tablas de verificación de disparadores *deleted* e *inserted* siempre están en la memoria activa. La ubicación de otras tablas a las que hace referencia el disparador determina la cantidad de tiempo necesaria para realizar la operación.

## ACTIVIDADES A DESARROLLAR EN CLASE

En la base de datos **VENTAS**:

1. Cree una función del tipo escalar que reciba el código identificador de un empleado y devuelva sus datos concatenados (nombre y apellido). Si no existiera o si se envía un valor nulo que muestre un mensaje de salida que indique “Dato no disponible”.
2. Cree una función que permita mostrar los datos de los clientes del tipo persona natural. La función deberá recibir como parámetro el código de un cliente y la función deberá mostrar los siguientes valores: código, nombres, apellido paterno, dirección, y teléfono concatenados.
3. Crear una función que muestre la cantidad de boletas emitidas a un cliente en un rango de fechas enviada como parámetro. El código del cliente es enviado como parámetro.
4. Crear una función que devuelva el total de ítems que registra el detalle de una boleta cuyo código es enviado como parámetro.
5. Crear un disparador que impida la eliminación de un empleado que haya emitido boletas.
6. Agregue en la tabla Boleta el campo total. Luego, cree un disparador que actualice el campo total por el cálculo del monto total a pagar en una boleta. Este disparador se ejecutara el tratar de insertar un registro en la tabla boleta.
7. Cree un trigger que elimine en cascada una boleta y su detalle.

### 7.1 Solución :

```
CREATE TRIGGER TX_ BOLETA_ELIMINAR  
ON BOLETA  
INSTEAD OF DELETE
```

```
AS
```

```
DECLARE @BOLETA CHAR (5)
```

```
--SE OBTIENE EL CÓDIGO DE LA BOLETA QUE SE HA ELIMINADO
```

```
SELECT @BOLETA=COD_BOL FROM DELETED
```

```
--SE VERIFICA SI LA BOLETA TIENE DETALLE
```

```
IF ( SELECT COUNT (*) FROM DETALLEBOLETA  
WHERE COD_BOL=@BOLETA) > 0
```

```
BEGIN
```

-- SE ELIMINA EL DETALLE DE LA BOLETA

**DELETE** DETALLEBOLETA WHERE COD\_BOL=@BOLETA

*/\*SE ELIMINA LA CABECERA.. ESTO SE HACE PUESTO QUE AL HABER DECLARADO EL **TRIGGER** DEL TIPO **INSTEAD OF**, ES DECIR, QUE EL **DELETE** EN LA TABLA BOLETA NO SE EJECUTA DE FORMA NORMAL. POR LO TANTO, HAY QUE FORZAR LA ELIMINACIÓN. \*/*

**DELETE BOLETA** WHERE COD\_BOL=@BOLETA  
PRINT 'BOLETA SE ELIMINÓ SATISFACTORIAMENTE'

**END**

**ELSE**

-- SE ELIMINA SOLAMENTE LA CABECERA

**DELETE** BOLETA WHERE COD\_BOL=@BOLETA  
PRINT 'BOLETA ELIMINADA CARECE DE DETALLE'

*/\*EL **TRIGGER INSTEAD OF DELETE** PUEDE SER USADO PARA REEMPLAZAR LA ACCIÓN REGULAR DE LA SENTENCIA **DELETE** SOBRE UNA TABLA O UNA VISTA. \*/*




## 7.2 PRUEBA DEL **TRIGGER**

**EJECUTAR LO SIGUENTE:**

**DELETE** BOLETA WHERE COD\_BOL='BOL777'



## Resumen

-  Los Triggers de Transact SQL pueden ser bastante complejos y pueden llegar a ser una parte importante del código fuente de su aplicación.
-  Por lo visto, las funciones definidas por el usuario proporciona muchas más opciones de programación. Las ventajas de la reutilidad del código y corrección de problemas en una sola rutina pueden ser realizadas incorporando “Funciones de Usuario” en nuestros diseños.
-  Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

 <http://technet.microsoft.com/es-es/library/ms186755.aspx>

Tutorial para la creación de funciones