¿ QUÉ ES UNA BASE DE DATOS ?

Es una colección de tablas con datos y otros objetos tales como: vistas, índices, procedimientos almacenados y disparadores que son organizados con el propósito de proveer una fuente central de información significativa para apoyar las funciones empresariales.

OBJETIVOS:

- Compartir los datos entre usuarios, apoyando gran variedad de aplicaciones.
- Mantener la Integridad de los Datos.
- Asegurar la disponibilidad de los datos en cualquier momento.
- Permitir cambios a la base de datos para ajustarse a las necesidades del entorno sin afectar su desempeño.

INTEGRIDAD DE DATOS

Las reglas de Integridad de Datos aseguran que los datos sean exactos y consistentes. Aunque la Integridad de los datos es a menudo lograda al construir restricciones en los programas de la aplicación, muchos desarrolladores aplican restricciones durante el modelamiento de datos, disminuyendo asi los requerimientos de los programas.

Existen cuatro tipo de reglas de integridad de los datos que son usados en el modelamiento de datos. Algunas de estas reglas están embebidas en la misma estructura de datos, otras se incorporan con explícitas definiciones externas.

- **Integridad de la Entidad.**
- **Integridad Referencial.**
- **Integridad de dominio.**
- **Integridad definida por el usuario.**

INTEGRIDAD DE DATOS

Integridad de la Entidad.
Esta regla establece que cada fila de la entidad posea una clave primaria única y que ninguna clave primaria puede ser nula. El valor Nulo es definido como "vacio" o "ausencia de valor". Un valor nulo no es "0" (cero) o un espacio en blanco, estos valores poseen un valor identificable. Los motores de bases de datos automáticamente aseguran esta regla permitiendo sólo claves primarias únicas y prohibiendo valores nulos en claves primarias.

Integridad Referencial.
Integridad referencial es el mecanismo que asegura que los valores de la clave primaria y de la clave foránea usados para establecer la relación nunca estén fuera de sincronismo. De otra manera, sería posible tener entidades hijas que no tengan una entidad padre correspondiente. Si esto sucede se denomina huérfana a la fila que no tiene una fila padre correspondiente.

En cualquier momento que los datos son insertados, removidos o actualizados existen riesgos de violación a la integridad referencial.

INTEGRIDAD DE DATOS

Problema de Inserción.

Esta violación ocurre cuando una fila es insertada dentro de una tabla hija con un valor de clave foránea que no concuerda con el valor de una clave primaria en la tabla padre. En este escenario tiene dos opciones que están disponibles para mantener la Integridad Referencial:

- **Restringir la inserción de la fila, no se permite que suceda.**
- **Establecer el valor de clave foránea a nulo.**

En la práctica, normalmente la inserción es restringida.

Problema de Eliminación.

Si una fila es eliminada de una tabla padre, entonces las filas de tabla hija que se refieren a esa fila padre quedan huérfanas. En este escenario tiene tres opciones que están disponibles para mantener integridad referencial:

- **La eliminación puede operar en cascada a las filas hijas, en este caso ellas también son eliminadas.**
- **La eliminación puede ser restringida.**
- **Los valores de las claves foráneas de las filas hijas son cambiadas a nulo.**

Típicamente la eliminación es restringida.

INTEGRIDAD DE DATOS

Problema de Actualización.

Si el valor de una clave primaria de una tabla padre es cambiado, entonces la fila de la tabla hija que se refieren a la tabla padre quedan huérfanas. Para este caso las siguientes alternativas existen:

- **La actualización puede operar en cascada a las filas hijas.**
- **La actualización puede estar restringida.**
- **Los valores de las claves foráneas de las filas hijas son cambiados a nulos.**

Existen un segundo caso de problema de actualización que sucede cuando el valor de la clave foránea en una fila de una tabla hija es modificado para referirse a una fila padre inexistente. En este caso la fila queda huérfana. Las opciones son:

- **La actualización puede estar restringida.**
- **Los valores de las claves foráneas de la fila hija son cambiados a nulo.**

•Es una práctica estándar inhabilitar modificaciones a la clave primaria y a las claves foráneas de tal manera que sólo valores válidos pueden ser ingresados.

INTEGRIDAD DE DATOS

Integridad de dominio.

La integridad de dominio asegura que los valores de las columnas de una tabla son válidos para las definiciones del dominio físico y lógico. Un dominio físico identifica el formato de la columna, como tipo de campo y longitud; mientras que un dominio lógico identifica el conjunto de valores válidos. Por ejemplo la columna CodigoCliente tiene un dominio físico definido como: tipo de dato numérico con longitus de 4 caracteres; y un dominio lógico: “rango de números entre 1000 y 4999”.

Integridad definida por el usuario.

Los usuarios pueden definir reglas (reglas de negocios complejas) de acuerdo a las políticas del negocio.Estas reglas pueden ser implementadas en el modelo de datos por medio de las declaraciones explícitas para que se incorporen en la implementación física de la base de datos. Un ejemplo de políticas de negocio es que ninguna factura pueda ser eliminada de la base de datos y que los valores adeudados por el cliente en estos casos siempre sean corregidos por notas de crédito. Otro ejemplo puede ser, si el monto total de la factura supera los $ 2000, le hago un descuento.

ARQUITECTURA DE SQL Server

Fundamentos

SQL Server es una base de datos relacional cliente-servidor basada en SQL (Lenguaje de consulta estructurado).

Microsoft® SQL Server™ está diseñado para operar de forma eficiente en varios entornos:

- Como sistema de base de datos cliente-servidor de dos estratos o de varios estratos
- Como sistema de base de datos de escritorio

Sistemas de bases de datos cliente-servidor

Los sistemas cliente-servidor están construidos de tal modo que la base de datos puede residir en un equipo central, llamado **servidor** y ser compartida entre varios usuarios. Los usuarios tienen acceso al servidor a través de una aplicación de cliente o de servidor:

- En un sistema cliente-servidor de dos estratos, los usuarios ejecutan una aplicación en su equipo local, llamado **cliente**, que se conecta a través de la red con el servidor que ejecuta SQL Server; también se conoce como cliente amplio.

ARQUITECTURA DE SQL Server

En un sistema cliente-servidor de varios componentes, la lógica de la aplicación de cliente se ejecuta en dos ubicaciones:

- El cliente reducido se ejecuta en el equipo local del usuario y se encarga de presentar resultados al usuario.
- Los clientes reducidos solicitan funciones a la aplicación de servidor, que, a su vez, es una aplicación multiproceso capaz de operar con varios usuarios simultáneos. La aplicación de servidor es la que abre las conexiones con el servidor de la base de datos y se puede ejecutar en el mismo servidor que la base de datos, o se puede conectar a través de la red con otro servidor que opere como servidor de base de datos. Éste es el escenario típico de las aplicaciones de Internet.

El tener los datos almacenados y administrados en una ubicación central ofrece varias ventajas:

- Todos los elementos de datos están almacenados en una ubicación central en donde todos los usuarios pueden trabajar con ellos.

No se almacenan copias separadas del elemento en cada cliente, lo que elimina los problemas de hacer que todos los usuarios trabajen con la misma información.

ARQUITECTURA DE SQL Server

• Las reglas de la organización y las reglas de seguridad se pueden definir una sola vez en el servidor para todos los usuarios.
Esto se puede hacer en una base de datos mediante el uso de restricciones, procedimientos almacenados y desencadenadores. También se puede hacer en una aplicación de servidor.

• Los servidores de base de datos relacionales optimizan el tráfico de la red al devolver sólo los datos que la aplicación necesita.
• Las gastos en hardware se pueden minimizar.
Como los datos no están almacenados en los clientes, éstos no tienen que dedicar espacio de disco a almacenarlos. El servidor se puede configurar para optimizar la capacidad de E/S de disco necesaria para obtener los datos y los clientes se pueden configurar para optimizar el formato y presentación de los datos obtenidos desde el servidor.
• Las tareas de mantenimiento como las copias de seguridad y restauración de los datos son más sencillas porque están concentradas en el servidor central .

En los sistemas cliente-servidor grandes, miles de usuarios pueden estar conectados con una instalación de SQL Server al mismo tiempo. SQL Server tiene una protección completa para dichos entornos, con barreras de seguridad que impiden problemas como tener varios usuarios intentando actualizar el mismo elemento de datos a la vez.

ARQUITECTURA DE SQL Server

Lenguaje estructurado de consulta (SQL)

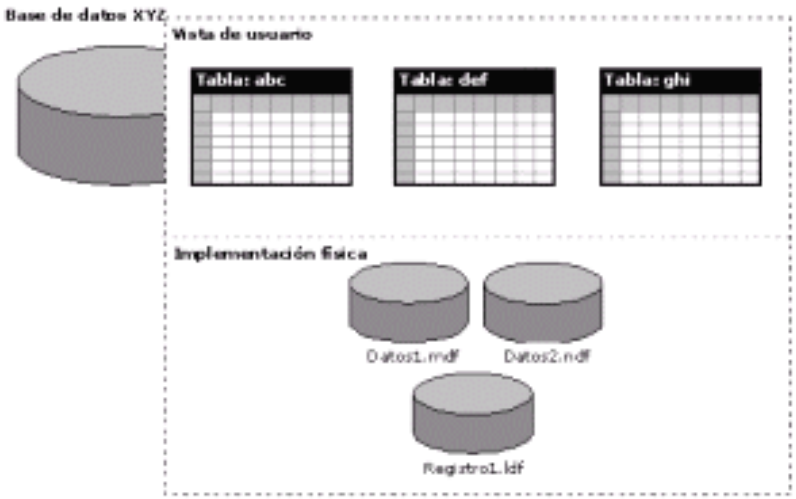
Para definir las estructuras de datos y manipularlas se utiliza un conjunto de lenguaje definido por el software de bases de datos. En las bases de datos relacionales el lenguaje utilizado es el SQL. Los estándares de SQL están definidos por el American National Standards Institute (ANSII) y la International Standars Organization (ISO).

Componentes Físicos y Lógicos de SQL Server

Los datos de Microsoft® SQL Server™ están organizados en bases de datos. Los datos de una base de datos están organizados en los componentes lógicos y componentes físicos.

Una base de datos está implementada físicamente como dos o más archivos de disco, es decir como archivos del System Operativo. Generalmente sólo el administrador de la base de datos tiene que trabajar con la implementación física.

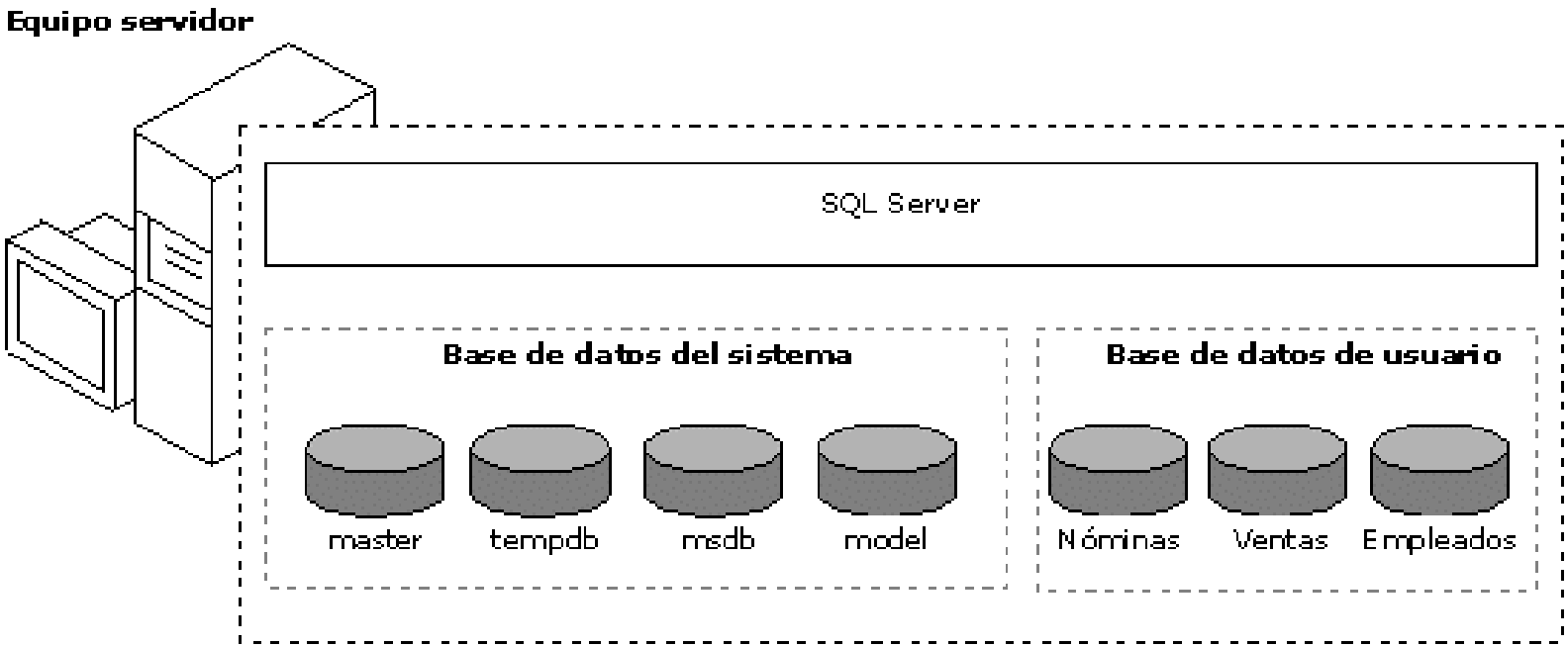
Los componentes lógicos son los componentes u objetos visibles por los usuarios tales como tablas, vistas, procedimientos, disparadores, tipos de datos.



Datos lógicos y físicos de una base de datos

Bases de Datos de SQL Server

Cada instalación de SQL Server tiene varias bases de datos. SQL Server tiene cinco bases de datos del sistema (**master**, **model**, **tempdb, msdb** y **distribution(**opcional para hacer replicación**)**) y cada instalación de SQL Server tiene una o varias bases de datos de usuario.



Bases de Datos del Sistema

Master

Master registra toda la información del sistema de SQL Server. Registra todas las cuentas de inicio de sesión y los valores de configuración del sistema. Se recomienda poseer siempre disponible una copia de seguridad actualizada de la misma.

La base de datos Master almacena la siguiente información:

- Cuentas de usuarios.
- Servidores remotos
- Procesos en ejecución.
- Mensaje de error del sistema.
- Bases de datos definidos en el servidor.
- Almacenamiento asignado a cada base de datos.
- Bloqueos activos.
- Archivos de respaldo y de almacenamiento.
- Procedimientos almacenados del sistema, que son usados principalmente para la administración del sistema.

Bases de Datos del Sistema

Model

Esta base de datos (plantilla) provee el modelo o prototipo en que las nuevas bases de datos de usuarios se basarán para su creación. Cada vez que una base de datos es creada, SQL Server realiza una copia de la base de datos Model y luego la extiende hasta el tamaño indicado en el momento de su creación. Una base de datos núnca puede ser más pequeña que la base de datos Model.

Cada vez que SQL Server es iniciado, la bases de datos TempDB es recreada por lo que la base de datos Model siempre debe existir.

TempDB

Las bases de datos TempDB provee almacenamiento para tablas y procedimientos almacenados temporales y otras necesidades de almacenamiento temporal tales como resultados intermedios de sentencias GROUP BY, ORDER BY, DISTINCT y cursores (permite procesar una tabla registro por registro).

Bases de Datos del Sistema

Msdb

Provee soporte para el servicio de SQL Server Agent(para ejecución de tareas) y provee un área de almacenamiento para la información de programación de Alertas y Jobs.

En resumen, permite definir acciones cuando ocurre un evento.

Distribution

Cuando un proceso de replicación es definido, una base datos de distribución es automáticamente creada en algunos servidores por el sistema.

Definiendo los objetos de la Base

Tipos de Dato Binario

Los datos binarios se componen de números hexadecimales. Por ejemplo, el número decimal 245 equivale al hexadecimal F5.

Tipo de Dato	Descripción
Binary[(n)]	Datos binarios de longitud fija de n bytes. El argumento n debe ser un valor comprendido entre 1 y 8.000.

Tipos de Dato Caracter

Se define como dato de carácter o alfanuméricos a cualquier combinación de letras, símbolos y caracteres numéricos. Por ejemplo, son datos de carácter válidos “928”, “Jimenez” y “(0*&(%B99nh jkJ.”.

Tipo de Dato	Descripción
char[(n)]	Datos de caracteres de longitud fija, con n caracteres. El valor de n tiene que estar comprendido entre 1 y 8.000. El tamaño de almacenamiento es n bytes.
varchar[(n)]	Datos de caracteres de longitud variable, con una longitud máxima de n caracteres. El valor de n tiene que estar comprendido entre 1 y 8.000.

Definiendo los objetos de la Base

Tipos de Dato Fecha y Hora

Constan de combinaciones válidas de fecha y hora. Por ejemplo, datos válidos de fecha y hora pueden ser “4/01/98 12:15:00 p.m.” y “1:28:29:15:01 a.m. 17/8/98”

Tipo de Dato	Descripción
datetime	Datos de fecha y hora comprendidos entre el 1 de enero de 1753 y el 31 de diciembre de 9999. El tamaño de almacenamiento es de 8 bytes.
smalldatetime	Datos de fecha y hora desde el 1 de enero de 1900 al 6 de Junio de 2079. El tamaño de almacenamiento es de 4 bytes.

Tipos de Dato Numérico

Constan de datos que almacenan con tanta precisión como permite el sistema de numeración binario.

Tipo de Dato	Descripción
real	Datos numéricos en coma flotante entre –3.04E + 38 y 3.40E + 38. El tamaño de almacenamiento es de 4 bytes. En SQL Server, el sinónimo de real es float(24).

Definiendo los objetos de la Base

Tipos de Dato Entero

Son datos enteros positivos y negativos.

Tipo de Dato	Descripción
int	Datos enteros comprendidos entre -2^{31} (-2.147.483.648) y 2^{31} - 1 (2.147.483.647). El sinónimo en SQL-92 para int es integer. El tamaño de almacenamiento es de 4 bytes.
smallint	Datos enteros comprendidos entre 2^{15} (-32.768) y 2^{15} - 1 (32.767). El sinónimo en SQL-92 para int es integer. El tamaño de almacenamiento es de 2 bytes.
tinyint	Datos enteros comprendidos entre 0 y 255. El tamaño es de almacenamiento es de 1 byte.

Decimales

Los datos ddecimales se componen de datos de los que se almacenan hasta el dígito menos significativo.

Definiendo los objetos de la Base

Tipo de Dato	Descripción
Numeric[(p[, s]]	Números de precisión y escala fijas. Cuando se utiliza precisión máxima, los valores permitidos están comprendidos entre - 10^{38} - 1 y 10^{38} - 1. p(precisión) Especifica el número máximo de dígitos que se puede almacenar. s(escala) Especifica el número máximo de decimales que se puede almacenar a la derecha del separador decimal.

Creando Bases de Datos

SQL Server 7.0 maneja 3 tipos de archivos:

Archivos de Datos Primarios

Este archivo posee las tablas del sistema y los datos en sí de la base de datos. Cada base de datos posee un solo archivo primario. La extensión recomendada para el archivo físico es **mdf** (master data file)

Archivos de Datos Secundarios

Los archivos de datos secundarios son todos los archivos de datos menos el archivo de datos primario. Estos archivos son necesarios si el archivo primario no posee suficiente espacio para almacenar todos los datos de la base de datos. La extensión recomendada para el archivo secundario es **ndf**.

Archivos Log

Este tipo de archivos contiene la información necesaria para el proceso de recuperación de la base de datos. Una base de datos puede poseer más de un archivo log pero al menos debe poseer uno. La extensión recomendada para el archivo físico es **ldf** (log data file)

Sentencia CREATE DATABASE

CREATE DATABASE nombrebaseDatos
[ON [PRIMARY]
[<especArchivo> [,..n]]
[, <filespec> [,..n]]
]
[LOG ON { <filespec> [,..n]}]
[FOR LOAD | FOR ATTACH]
[filespec> ::=
([NAME=nombreArchivoLógico,]
FILENAME = 'nombreArchivoSO'
[, SIZE=tamaño]
[, MAXSIZE={ tamañoMáximo | UNLIMITED }]
[, FILEGROWTH=incrementoCrecimiento]) [,..n]
<grupoArchivos>::=
FILEGROUP nombreGrupoArchivos <filespec> [,..n]

Sentencia CREATE DATABASE

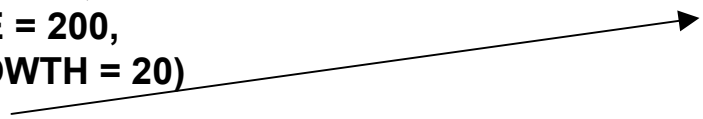
Creando una Base de datos especificando un archivos de datos y de log.

```
USE master
GO
CREATE DATABASE Sales
ON
( NAME = Sales_dat,
FILENAME = 'c:\mssql7\data\saledat.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
FILENAME = 'c:\mssql7\data\salelog.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB )
GO
```

Sentencia CREATE DATABASE

b) Creando una Base de Datos especificando múltiples archivos de datos y logs

```
USE master
GO
CREATE DATABASE Archive
ON
PRIMARY ( NAME = Arch1,
FILENAME = 'c:\mssql7\data\archdat1.mdf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
( NAME = Arch2,
FILENAME = 'c:\mssql7\data\archdat2.ndf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
( NAME = Arch3,
FILENAME = 'c:\mssql7\data\archdat3.ndf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20)
LOG ON
```



```
( NAME = Archlog1,
FILENAME = 'c:\mssql7\data\archlog1.ldf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20),
( NAME = Archlog2,
FILENAME = 'c:\mssql7\data\archlog2.ldf',
SIZE = 100MB,
MAXSIZE = 200,
FILEGROWTH = 20)
GO
```

Sentencia CREATE DATABASE

c) Creando una simple Base de Datos

Este ejemplo crea una base de datos de nombre Products y especifica un solo archivo. prods_dat es por default el archivo primario con un tamaño de 4 MB, y 1 MB de tamaño del archivo log creado automáticamente. El parámetro MAXSIZE para el archivo log no esta especificado y por default puede crecer hasta que haya espacio en el disco.

```
USE master
GO
CREATE DATABASE Products
ON
( NAME = prods_dat,
FILENAME = 'c:\mssql7\data\prods.mdf',
SIZE = 4,
MAXSIZE = 10,
FILEGROWTH = 1 )
GO
```

Sentencia CREATE DATABASE

d) Creando una Base de Datos sin archivos

Este ejemplo crea una base de datos con nombre mytest y automáticamente crea un archivo primario y un archivo log. El tamaño del archivo primario será igual al tamaño del archivo primario de la base de datos model, de igual manera para el tamaño del archivo log el mismo tamaño del archivo log de la base de datos model. El crecimiento de los archivos no esta especificado, es decir que por default crecerán hasta que se llene el disco.

```
CREATE DATABASE mytest
```

e) Creando una Base de Datos sin especificar el tamaño

Este ejemplo crea una base de datos con nombre products2. El archivo prods2_dat por default es el archivo primario con un tamaño igual al del archivo primario de la base de datos model. El archivo log es creado automáticamente con un tamaño del 25% de tamaño del archivo primario o 512 KB. El parámetro MAXSIZE no es especificado y por default estos archivos crecerán hasta que haya espacio en el disco.

```
USE master
GO
CREATE DATABASE Products2
ON
( NAME = prods2_dat,
FILENAME = 'c:\mssql7\data\prods2.mdf' )
GO
```

Sentencia DROP DATABASE

Eliminado Bases de Datos

Cuando se elimina una base de datos, todos sus archivos y datos son eliminados físicamente. Una vez eliminada una Base no podrá recuperar la información a menos que haya hecho una copia de seguridad

```
DROP DATABASE baseDeDatos[,…n]
```

a) Eliminando una base de datos

DROP DATABASE Ventas

b) Eliminando varias base de datos

DROP DATABASE pubs, newpubs

Sentencia CREATE TABLE

La sentencia CREATE TABLE

- ♦ Es empleada para la creación de nuevas tablas.
- ♦ Los nombres de las columnas deben ser únicos dentro de la tabla.
- ♦ Cada columna debe poseer un tipo de dato.
- ♦ En la creación de tablas se puede definir hasta:
- 2 Billones de tablas por base de datos.
- 1024 Columnas por Tabla.
- Máximo Longitud por Registro: 8060 bytes sin incluir tipos de datos image, text y ntext.

Sentencia CREATE TABLE

a) Creando una tabla con clave primaria, identity, default, check

Este ejemplo crea las tablas la primera de nombre **tareas**, la columna tareas_id es la clave primaria y se va a empezar desde 1 con un incremento de 1 porque tiene la restricción identity; la columna tareas_desc tiene un default de "Nueva posición"; y las columnas min_lvl, max_lvl tienen la restricción de chequeo.

```
CREATE TABLE tareas
(
tareas_id   smallint IDENTITY(1,1) PRIMARY KEY,
tareas_desc varchar(50) NOT NULL DEFAULT 'Nueva Posición',
min_lvl     int NOT NULL CHECK (min_lvl >= 10),
max_lvl     int NOT NULL CHECK (max_lvl <= 250)
)
```

Sentencia CREATE TABLE

b) Creando una tabla y relacionando con la tabla (a)

Ester ejemplo crea una tabla de nombre estudiante con una clave primaria est_id; est_nombre el nombre del estudiante, la columna tareas_id que es la referencia (relación) a la tabla tareas; y la fecha de ingreso del estudiante fecha_ing.

```
CREATE TABLE estudiante
(
est_id      smallint CONSTRAINT PK_emp_id PRIMARY KEY,
est_nombre varchar(30) NOT NULL,
tareas_id  smallint NOT NULL DEFAULT 1
REFERENCES tareas(tareas_id),
fecha_ing  datetime NOT NULL
DEFAULT (getdate())
)
```

Sentencias: Data Manipulation Language (DML)

☑ La sentencia SELECT

♦ Es el comando que más se utiliza y es la forma fundamental de consultar datos.

♦ Selecciona Registros y Columnas de las tablas.

♦ Básicamente se compone de tres palabras reservadas:

- SELECT - Especifica las columnas.
- FROM - Especifica las tablas.
- WHERE - Especifica los registros.

♦ SELECT * Recupera todas las columnas

Sintaxis básica de SELECT

SELECT [**ALL | DISTINCT**] <Columnas que van a ser escogidas, operaciones y variables>
[**FROM**] <Lista de Tablas que serán evaluadas>
[**WHERE**] <Criterios que deberán cumplirse para la selección de registros>
[**GROUP BY**] <columnas para agrupar funciones agregadas>
[**HAVING**] **<**criterios que deben cumplirse para las funciones agregadas**>**
[**ORDER BY**] <especificación opcional de como debe ordenarse los resultados>

SELECT

Seleccionando Columnas

Una sencilla consulta que recupera las columnas: pub_id, pub_name, city, state de la tabla publishers de la base de datos ***Pubs.***

♦ Los nombres de las columnas son separados por una coma. Ejemplo:

```
SELECT pub_id, pub_name, city, state
FROM publishers
```

El resultado es:

pub_id	pub_name	city	state
0736	New Moon Books	Boston	MA
0877	Binnet & Hardley	Washington	DC
1389	Algodata Infosystems	Berkeley	CA
1622	Five Lakes Publishing	Chicago	IL
1756	Ramona Publishers	Dallas	TX
9901	GGG&G	München	
9952	Scootney Books	New York	NY
9999	Lucerne Publishing	Paris	

.....

SELECT

Utilizando literales

- Resultados del SELECT más descriptivos.
- Delimitados por comillas simples o dobles. Ejemplo:

```
SELECT pub_id, pub_name, 'Ciudad:', city, state
FROM publishers
```

El resultado es:

pub_id	pub_name		city	state
0736	New Moon Books	Ciudad:	Boston	MA
0877	Binnet & Hardley	Ciudad:	Washington	DC

.....

SELECT

MANIPULANDO DATOS

En esta unidad vamos a ver la manipulación de los siguientes datos:

- Manipulando Datos Numéricos.
  - Operadores Aritméticos
  - Funciones Matemáticas.
- Manipulando Datos Alfanuméricos.
- Manipulando Datos de Fecha y Tiempo.
- Funciones del Sistema.

SELECT

Datos Numéricos: Operadores Numéricos

Ejecutan cómputos a partir de columnas ó constantes numéricas.
Los operadores numéricos son:

Símbolo	Significado	Tipos de Datos
+	Adición	int, smallint, tinyint, numeric, decimal, float, real, money, y smallmoney
-	Substracción	
*	Multiplicación	
/	División	
%	Módulo	int, smallint y tinyint

Datos Numéricos: Operadores Comparativos

Los operadores comparativos contrastan una específica diferencia entre dos expresiones. Estos son los operadores comparativos:

Símbolo	Significado
=	Igual
>	Mayor Que
<	Menor Que
>=	Mayor Igual Que
<=	Menor Igual Que
<>	Distinto

SELECT

Investigar:

- Funciones Matemáticas
- Funciones Alfanuméricas
- Funciones Fecha y Hora

SELECT

SeleccionandoRegistros

La cláusula WHERE en la sentencia SELECT determina los registros a recuperar de acuerdo a las condiciones de búsqueda.

Condiciones de búsqueda:

Operadores	
Comparación	= > < >= <= <> != !> !<
Rangos	BETWEEN AND NOT BETWEEN
Listas	IN y NOT IN
Cadena	LIKE y NOT LIKE
Valores Desconocidos	IS NULL y IS NOT NULL
Combinaciones	AND, OR
Negaciones	NOT

SELECT

Seleccionando Registros Mediante Comparaciones

La palabra reservada BETWEEN permite realizar búsquedas en un rango de valores. Delimite los valores con comillas simples ó dobles para tipos de datos: char, varchar, text, nchar, nvarcahr, ntext, datetime y smalldatetime.

Sintaxis:

```
SELECT Select_list
FROM table_list
WHERE expresión [NOT] BETWEEN expresión AND expresión
```

SELECT

a)Seleccionando datos dentro de un rango

```
USE pubs
GO
SELECT title_id, ytd_sales
FROM titles
WHERE ytd_sales BETWEEN 4095 AND 12000
GO
```

El resultado es:

```
title_id ytd_sales
-------- -----------
BU1032           4095
BU7832           4095
PC1035           8780
PC8888           4095
TC7777           4095
```

SELECT

b)Seleccionando datos fuera de un rango

```
USE pubs
GO
SELECT title_id, ytd_sales
FROM titles
WHERE ytd_sales NOT BETWEEN 4095 AND 12000
GO
```

El resultado es:

```
title_id ytd_sales
-------- -----------
BU1111  3876
BU2075  18722
MC2222 2032
MC3021 22246
PS1372   375
PS2091   2045
PS2106   111
PS3333   4072
PS7777   3336
TC3218  375
TC4203  15096
```

SELECT

Seleccionando Registros Mediante Listas

La palabra reservada IN permite realizar búsqueda que coincida con una lista de valores.

Sintaxis:

```
SELECT Select_list
FROM table_list
WHERE expresión [NOT] IN (Lista_Valores)
```

```
USE pubs
SELECT au_lname, state
FROM authors
WHERE state IN ('IN', 'MD')
```

El resultado es:

```
au_lname            state
------------------- -----
DeFrance            IN
Panteley            MD
```

SELECT

Seleccionando Registros Mediante Cadenas

La palabra reservada LIKE permite realizar búsqueda con subcadenas.

Sintaxis:

```
SELECT Select_list
FROM table_list
WHERE expresión [NOT] LIKE “String”
```

Posee 4 caracteres ‘Wildcard’:

Carácter	Descripción
%	Cadena con cero ó más caracteres.
_	Cualquier carácter
[]	Cualquier carácter dentro del rango especificado.
[^]	Cualquier carácter fuera del rango especificado

SELECT

Ejemplo:

LIKE “AR%” Nombres que comiencen con “AR”.
LIKE “_on%” Nombres de tres letras que terminen en “on”
LIKE “[AP]%” Nombres que comiencen con “A” ó “P”.
LIKE “[^A]%” Nombres que no comiencen con “A”.

```
USE pubs
GO
SELECT phone
FROM authors
WHERE phone LIKE '415%'
ORDER by au_lname
GO
```

El resultado es:

```
phone
------------
415 658-9932
415 548-7723
415 836-7128
415 986-7020
415 836-7128
415 534-9219
415 585-4620
415 354-7128
415 834-2919
415 843-2991
415 935-4228
```

```
USE pubs
GO
SELECT phone
FROM authors
WHERE phone NOT LIKE '415%'
ORDER BY au_lname
GO
```

El resultado es:

```
phone
------------
503 745-6402
219 547-9982
615 996-8275
615 297-2723
707 938-6445
707 448-4982
408 286-2428
301 946-8853
801 826-0752
801 826-0752
913 843-0462
408 496-7223
```

SELECT

Seleccionando Registros sobre Valores Desconocidos

Un NULL (Valor Nulo) significa ausencia de valor para una determinada columna.Para seleccionar registros con valores NULOS utilice la palabra reservada ***IS NULL*** en la cláusula WHERE. En Ordenación Ascendente el NULL se presenta primero.

Sintaxis:

```
SELECT * FROM titles
WHERE ytd_sales IS NULL
```

SELECT

Ejemplo:

```
USE pubs
SELECT title_id, advance
FROM titles
WHERE advance < $5000 OR advance IS NULL
ORDER BY title_id
```

El resultado es:

```
title_id advance
-------- -------------------------
MC2222   0.00
MC3026   (null)
PC9999   (null)
PS2091   2,275.00
PS3333   2,000.00
PS7777   4,000.00
TC3218   (null)
```

SELECT

Seleccionando Registros sobre varios Argumentos de Búsqueda

Se puede combinar varios argumentos de búsqueda a través de operadores lógicos como AND, OR y NOT. Orden de evaluación: NOT, AND y OR. Se pueden cambiar el orden de evaluación mediante el uso de Paréntesis.

Ejemplo:

```
USE pubs
SELECT au_id, au_lname, au_fname,phone
FROM authors
WHERE (au_lname LIKE 'G%' OR au_lname LIKE 'D%')
AND state = 'CA'
```

El resultado es:

```
au_id        au_lname         au_fname    phone
-----------  ---------------  ----------- ------
427-17-2319  Dull             Ann         415 -7128
213-46-8915  Green            Marjorie    415 986-7020
472-27-2349  Gringlesby       Burt        707 938-644
```

SELECT

Eliminando Valores Duplicados

La Palabra reservada DISTINCT elimina los valores duplicados de alguna expresión del Select_list.

Sintaxis:

```
SELECT [ALL|DISTINCT] Select_List
FROM tabla
WHERE condiciones
```

SELECT

Ejemplo:

```
SELECT DISTINCT address
FROM authors
WHERE state NOT IN ('CA')
```

El resultado es:

```
address
----------------------
10 Mississippi Dr.
1956 Arlington Pl.
22 Graybar House Rd.
2286 Cram Pl. #86
3 Balding Pl.
55 Hillsdale Bl.
67 Seventh Av.
```

ORDENANDO RESULTADOS (ORDER BY)

La Cláusula ORDER BY ordena los resultados por una o más columnas. Ordenamiento Ascendente (ASC) ó Descendente (DESC).

Ejemplo:

```
SELECT au_id, au_lname, city, zip
FROM authors
ORDER BY 3, 4 DESC
```

El resultado es:

au_id	au_lname	city	zip
712-45-1867	del Castillo	Ann Arbor	48105
238-95-7766	Carson	Berkeley	94705
409-56-7008	Bennet	Berkeley	94705
648-92-1872	Blotchet-Halls	Corvallis	97330
472-27-2349	Gringlesby	Covelo	95428
722-51-5454	DeFrance	Gary	46403
341-22-1782	Smith	Lawrence	66044
--	--	--	--

GENERANDO DATOS SUMARIZADOS

- Funciones Agregadas
- Cláusulas GROUP BY y HAVING.

Funciones Agregadas

Las funciones agregadas (llamadas a veces *funciones de conjunto*) permiten resumir una columna de resultados.

En la tabla siguiente se presenta un resumen de las funciones agregadas de SQL Server.

Función agregada	Descripción
AVG(*expresión*)	Devuelve el promedio (media) de todos los valores, o solo de los valores DISTINCT, de la expresión AVG. Solo se puede utilizarse en columnas numéricas. Se ignoran los valores nulos.

GENERANDO DATOS SUMARIZADOS

Función agregada	Descripción
COUNT(*expresión*)	Devuelve el número de valores no nulos en la expresión. COUNT puede utilizarse con columnas tanto numéricas como de tipo carácter. Se ignoran los valores nulos.
COUNT(*)	Devuelve el número de filas. COUNT(*) no tiene parámetros y no puede utilizarse con el DISTINCT, se cuentan todas las filas, incluso las que tienen valores nulos.
MAX(*expresión*)	Devuelve el valor máximo en la expresión. MAX puede utilizarse con columnas numéricas y datetime, pero no con columnas de tipo *bit*. MAX ignora los valores nulos.
MIN(*expresión*)	Devuelve el valor mínimo en la expresión. MIN puede utilizarse con columnas numéricas y datetime, pero no con columnas de tipo *bit*. MIN ignora los valores nulos.
SUM(*expresión*)	Devuelve la suma de todos los valores, o sólo de los valores DISTINCT, de la expresión. SUM sólo puede utilizarse con columnas numéricas e ignoran los valores nulos.

GENERANDO DATOS SUMARIZADOS

Las funciones agregadas sólo se aceptan en expresiones tales como:

- Lista de relación de una sentencia SELECT.
- Cláusula COMPUTE ó COMPUTE BY
- Cláusula HAVING

Ejemplo:

```
SELECT PrecioProm = AVG(Precio),
       TotalVtas = SUM(Total)
FROM Productos
WHERE Descripcion LIKE 'Computador%'
```

El resultado es:

PrecioProm	TotalVtas
3.25	89100.00

(1 fila(s) afectadas)

GENERANDO DATOS SUMARIZADOS

Cláusulas GROUP BY y HAVING

La cláusula *GROUP BY* organiza los datos en forma de Grupos.

- Puede agrupar sobre una columna ó una expresión.
- Típicamente usado con funciones agregadas.
- Produce un registro por cada grupo diferente.

La cláusula *HAVING* restringe los grupos a presentarse basado en una condición.

- Se puede aplicar sobre una columna ó una expresión.
- Permite funciones agregadas.
- Similar a la cláusula *WHERE*.

GENERANDO DATOS SUMARIZADOS

Condiciones de GROUP BY y HAVING

- Cada columna no agregada del Select_list debe ser mencionada en la cláusula GROUP BY.
- GROUP BY puede incluir expresiones.
- GROUP BY ALL presenta todos los grupos, a pesar de que sean excluidos por la cláusula WHERE.
- Las columnas del HAVING deben retornar un solo valor.
- Un query con una cláusula HAVING debe tener una cláusula GROUP BY.

CORRELACIONANDO DATOS

- Implementando Joins.
- Inner Joins.
- Cross Joins.
- Outer Joins.
- Joins con más de dos tablas.

CORRELACIONANDO DATOS

Implementando JOINS

```
SELECT 'Código' = TA.title_id, 'Titulo del Libro' = T.title
FROM titles as  T, titleauthor as TA
WHERE T.title_id=TA.title_id AND T.title_id LIKE 'P%'
ORDER BY T.title_id
```

El resultado es:

Código	Titulo del Libro
PC1035	But Is It User Friendly?
PC8888	Secrets of Silicon Valley
PC8888	Secrets of Silicon Valley
PC9999	Net Etiquette
PS1372	Computer Phobic AND Non-Phobic Individuals: Behavior Variations
PS1372	Computer Phobic AND Non-Phobic Individuals: Behavior Variations
PS2091	Is Anger the Enemy?
PS2091	Is Anger the Enemy?
PS2106	Life Without Fear
PS3333	Prolonged Data Deprivation: Four Case Studies
PS7777	Emotional Security: A New Algorithm

CORRELACIONANDO DATOS

Inner JOINS

Relaciona dos tablas e incluye en una tercera sólo los registros que satisfacen la condición del Join.

Existen dos tipos de Inner Joins:

- ***EquiJoin***: Los valores de las columnas enlazadas son comparadas por igualdad, y todas las columnas de las tablas enlazadas son presentadas.
- ***Natural Join***: En el resultado se visualiza una sola vez las columnas Joins.

CORRELACIONANDO DATOS

Ejemplo EquiJoin:

```
SELECT *
FROM authors, publishers
WHERE authors.city = publishers.city
```

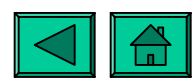
Este ejemplo presenta todas las columnas enlazadas.

Ejemplo Natural Join:

```
SELECT authors.au_lname, publishers.*
FROM  publishers, authors
WHERE publishers.city = authors.city
```

El resultado es:

au_lname	pub_id	pub_name	city	state	country
Carson	1389	Algodata Infosystems	Berkeley	CA	USA
Bennet	1389	Algodata Infosystems	Berkeley	CA	USA



CORRELACIONANDO DATOS

Cross Joins:

Produce un set de registros que incluye todas las combinaciones de todos los registros de las tablas que participan en el join.

Total Registros: #Reg Tabla1 * #Reg TablaN

ANSI-SQL:

```
SELECT pub_name, title
FROM   titles CROSS JOIN publishers
```

TRANSACT-SQL:

```
SELECT pub_name, title
FROM   titles, publishers
```

CORRELACIONANDO DATOS

Outer Joins:

Permite *restringir los registros en una tabla mientras no restringe los de la otra tabla*. Es útil para determinar qué datos de claves primarias y foráneas están fuera de sincronización.

ANSI-SQL incluye tres tipos de operadores Outer Joins:

- ***LEFT OUTER JOIN:*** Incluye todos los registros de la primera tabla especificada.

- ***RIGHT OUTER JOIN:*** Incluye todos los registros de la segunda tabla especificada.

- ***FULL OUTER JOIN:*** Incluye todos los registros de la primera tabla especificada que no existen en la segunda tabla especificada y viceversa.

CORRELACIONANDO DATOS

Outer Joins:

TRANSACT-SQL Incluye dos operadores Outer Joins:

*= Incluye todos los registros de la primera tabla especificada.
=* Incluye todos los registros de la segunda tabla especificada.

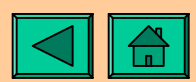
Pueden desaparecer en siguientes versiones de SQL Server. Pero SQL Server 7.0 si lo reconoce en la cláusula WHERE.

Ejemplo ANSI-SQL:

```
SELECT title, stor_id, ord_num, qty, ord_date
FROM   titles LEFT OUTER JOIN sales
ON     titles.title_id = sales.title_id
```

Ejemplo TRANSACT-SQL:

```
SELECT title, stor_id, ord_num, qty, ord_date
FROM   titles, sales
WHERE  titles.title_id *= sales.title_id
```



CORRELACIONANDO DATOS

Joins con más de dos tablas

```
SELECT  au_lname, title
FROM authors as A, titleauthor as TA, titles as T
WHERE A.au_id=TA.au_id AND
T.title_id=TA.title_id AND
A.state <> 'CA'
ORDER BY au_lname, au_fname
```

El resultado es:

au_lname	title
Blotchet-Halls	Fifty Years in Buckingham Palace Kitchens
DeFrance	The Gourmet Microwave
del Castillo	Silicon Valley Gastronomic Treats
Panteley	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
Ringer	Is Anger the Enemy?
Ringer	Life Without Fear
Ringer	The Gourmet Microwave
Ringer	Is Anger the Enemy?

GENERANDO DATOS SUMARIZADOS

Ejemplos:

Se tiene la tabla de Cliente, Producto, Cab_Factura, Det_Factura

Prod_Codigo	Prod_Descripcion	Prod_Estado
1	A	A
2	B	I
3	C	A
4	D	I
5	E	A
6	F	I

Producto

Num_Factura	Num_Linea	Prod_Codigo	Cantidad	Total
1201	1	1	2	100
1201	2	2	3	200
1201	3	3	5	500
1201	4	4	5	500
1202	1	5	5	500
1202	2	6	3	300
1203	1	1	3	300
1203	2	2	2	200
1203	3	3	4	400

Det_Factura

GENERANDO DATOS SUMARIZADOS

Ejemplos:

Se requiere:

• Mostrar los detalles de la Factura reemplazando el codigo del producto por el respectivo nombre del mismo.

• Mostrar por Factura en la tabla de Det_Factura, Cuántas líneas de detalle contiene.

• Mostrar la cantidad total y el valor total de venta por Producto.

• Mostrar la cantidad total y el valor total de venta por Factura.

• Se tiene el siguiente formato de consulta. Genere su sentencia sql.

Producto	TotalCant	TotalVtas	UltFactura
A	5	400	1203
B	5	400	1203
C	9	900	1203
D	5	500	1201
E	5	500	1202
F	3	300	1202

GENERANDO DATOS SUMARIZADOS

Utilizando HAVING

```
Select      Producto = A.Prod_Descripcion,
            TotalCant = SUM(B.Cantidad),
            TotalVtas = SUM(B.Total),
            UltFactura = MAX(B.Num_Factura)
From        Producto A, Det_Factura B
Where       A.Prod_Codigo = B.Prod_Codigo
Group By    A.Prod_Descripcion
HAVING      SUM(B.Total) >= 500
```

Producto	TotalCant	TotalVtas	UltFactura
C	9	900	1203
D	5	500	1201
E	5	500	1202

Resultado

SUBQUERIES

Definición Básica:

Una subconsulta es una sentencia SELECT que devuelve uno o varios valores, y es especificada (como una expresión) en una sentencia SELECT, INSERT, UPDATE o DELETE, o dentro de otra subconsulta.

Subconsultas usando IN

La palabra clave IN permite evaluar si el valor de una columna se encuentra en el conjunto de valores retornados por una subsonsulta.

La siguiente sentencia retorna todos los autores que al menos poseen un libro cuyo porcentaje de distribución sea menor a 50.

```
SELECT ApellidoAutor, NombreAutor
FROM Autores
WHERE CodAutor IN (SELECT CodAutor
                   FROM TituloAutores
                   WHERE Porc_Dist < 50)
```

SUBQUERIES

Subconsultas usando NOT IN

La palabra clave NOT IN permite evaluar si el valor de una columna no se encuentra en el conjunto de valores retornados por una subsonsulta.

La siguiente sentencia retorna todos los editores que no hayan publicado libros acerca de 'Empresa'.

```
SELECT NombreEditor
FROM Editores
WHERE CodEditor NOT IN (SELECT CodEditor
                        FROM Libros
                        WHERE Tipo = 'Empresa')
```

SUBQUERIES

Subconsultas con Operadores de comparación

Para usar una subconsulta presentada con un operador de comparación sin modificar, debe estar seguro que la Subconsulta devolverá exactamente un solo valor.

Si supone que cada editor se encuentra en una sola ciudad, y desea retornar el apellido y nombre de los autores que viven en la ciudad donde reside el editor 'Algodata Infosystema', para este caso puede presentar una subconsulta utilizando el operador de comparación simple (=).

```
SELECT ApellidoAutor, NombreAutor
FROM Autores
WHERE Ciudad=(SELECT Ciudad
              FROM Editores
              WHERE NombreEditor = 'AlgodataInfoSystema')
```

Si Algodata estuviese en varias ciudades, se produciría un mensaje de error. Se podría usar en vez de (=) el operador IN.

SUBQUERIES

Subconsultas con Operadores de comparación

Las Subconsultas presentadas con operadores de comparación pueden incluir funciones agregadas. Por ejemplo, la siguiente instrucción busca los nombres de todos los libros con un precio superior al precio mínimo.

```
SELECT DISTINCT Titulo
FROM Libros
WHERE Precio > (SELECT MIN(Precio)
                FROM Libros)
```

SUBQUERIES

Subconsultas usando EXISTS

La palabra clave EXISTS permite evaluar si una subconsulta retorna al menos un registro o ninguno, es decir, funciona como una prueba de existencia. La cláusula WHERE de la cláusula externa comprueba la existencia de los registros devueltos por la subconsulta.

```
SELECT Código, Descripción
FROM Productos a
WHERE NOT EXISTS (SELECT *
                  FROM Ventas b
                  WHERE b.IdProd = a.IdProd)
```

SUBQUERIES

```
SELECT Título = title_id, Cantidad = qty,
total = (SELECT SUM(qty) FROM sales),
porcentaje = (CONVERT(float, qty) /
                    (SELECT SUM(qty) FROM sales) * 100)
FROM sales
```

El resultado es:

```
Título  Cantidad  total  porcentaje
------- --------  -----  ----------
BU1032       5    493    1.0141987829614605
PS2091       3    493    0.6085192697768762
PC8888      50    493    10.141987829614605
...         ..    ..       ....
```

SUBQUERIES

```
SELECT title_id, au_id, royaltyper
FROM titleauthor
WHERE royaltyper = (SELECT MAX(royaltyper)
                    FROM titleauthor)
```

El resultado es:

Title_id	au_id	royaltyper
PS3333	172-32-1176	100
BU2075	213-46-8915	100
PC1035	238-95-7766	100
BU7832	274-80-9391	100
PC9999	486-29-1786	100
PS7777	486-29-1786	100
TC4203	648-92-1872	100
MC2222	712-45-1867	100
TC3218	807-91-6654	100
PS2106	998-72-3567	100

MODIFICANDO DATOS

- Insertando Registros.
- Actualizando Registros.
- Eliminando Registros.

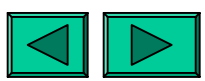
MODIFICANDO DATOS

Insertando Registros DEFAULT

La cláusula DEFAULT ingresa valor predeterminado para una columna. Las columnas deben poseer propiedad timestamp, permitir NULL o DEFAULT asignado.

Ejemplo:

```
INSERT employee
VALUES ('KLT91469F', 'Katrina', 'L', 'Thompson',
          DEFAULT, DEFAULT, DEFAULT, '01/14/95')
```



MODIFICANDO DATOS

Insertando Datos Parciales

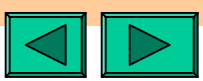
Puede especificar las columnas a ingresar valor, omitiendo aquellas que permiten NULL, timestamp, IDENTITY ó DEFAULT.

Sintaxis:

```
INSERT [INTO]
      {tabla | vista} [(lista_columnas)]
{DEFAULT VALUES | lista_valores |
sentencia_select}
```

Ejemplo:

```
INSERT titles(title_id, title, type, pub_id,
notes, pubdate)
VALUES ('BU1237', 'Get Going!', 'business', '138',
'great', '06/18/86')
```



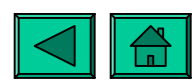
MODIFICANDO DATOS

Insertando Registros con SELECT *(INSERT/SELECT)*

Emplee SELECT para insertar registros a partir de los datos de otra tabla o de la misma.
La tabla del INSERT debe ser compatible con los resultados del SELECT (Número, Orden, Tipo de Dato).

Ejemplo:

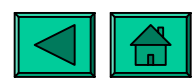
```
INSERT INTO newauthors
SELECT *
FROM authors
WHERE city = 'San Francisco'
```



MODIFICANDO DATOS

Insertando Registros con EXECUTE *(INSERT/EXECUTE)*

```
INSERT INTO MejoresClientes
EXECUTE sprConsultaMejoresClientes
```



MODIFICANDO DATOS

Actualizando Registros

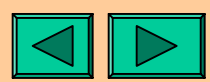
La sentencia **UPDATE** modifica los datos de determinados registros de una tabla. La cláusula *SET* indica las columnas a modificar. Generalmente utiliza la cláusula *WHERE* que limita la actualización a subconjunto de registros de la tabla.

Sintaxis:

```
UPDATE {tabla | vista}
SET nombre_columna = expresión
FROM lista_tablas
WHERE Condiciones
```

Ejemplo:

```
UPDATE authors
SET au_lname = 'Yokohama'
FROM authors
WHERE au_lame = 'Yokomoto'
```



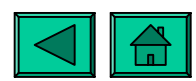
MODIFICANDO DATOS

Modificando Registros en base a otras tablas

La cláusula FROM lista los orígenes de los datos y la tabla en sí a ser actualizada.
Una sentencia UPDATE nunca actualiza el mismo registro dos veces (Minimiza Log).

Ejemplo:

```
UPDATE titles
SET    ytd_sales = ytd_sales + qty
FROM   titles, sales
WHERE  titles.title_id= sales.title_id AND
       sales.ord_date   = (SELECT MAX(sales.ord_date)
FROM sales)
```



MODIFICANDO DATOS

Eliminando Registros

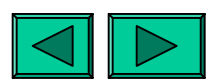
La sentencia **DELETE** elimina uno ó más registros de una tabla. Cómo se eliminan registros no columnas (campos), nunca hay que especificar nombres de columnas en una instrucción DELETE como se hacen con INSERT o UPDATE. Pero en otros aspectos DELETE actúa de manera muy parecida a UPDATE. Hay que especificar una cláusula WHERE para determinar los registros a eliminar.

La sentencia TRUNCATE TABLE remueve todos los registros de una tabla. Es más rápido que DELETE, sólo registra la desasignación de las páginas de datos y libera inmediatamente el espacio.

Sintaxis:

```
DELETE [FROM] {tabla | vista}
[WHERE condiciones]

TRUNCATE TABLE {tabla}
```



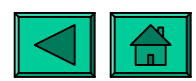
MODIFICANDO DATOS

Eliminando Registros a partir de otras Tablas

La cláusula WHERE determina los registros a eliminar de acuerdo a los Joins a otras tablas. Solamente los registros de la tabla del DELETE son eliminados.

Ejemplo:

```
DELETE FROM titleauthor
FROM authors a, titles t
WHERE a.au_id = titleauthor.au_id AND
            titleauthor.title_id = t.title_id AND
        t.title LIKE '%computers%'
```



Funciones Nativas de SQL Server

Funciones para manejo de cadenas
Funciones para manejo de valores aritméticos
Funciones para manejo de fechas

Funciones de conversión de tipos de Datos

La función CONVERT y CAST permiten convertir expresiones de un tipo de dato a otro. La función CONVERT a diferencia de CAST permite definir un estilo de formato cuando se convierten tipos de dato fecha (datetime y smalldatetime) y numéricos (float, real, money, o smallmoney) a tipos de dato carácter (nchar, nvarchar, char, varchar)

Sintaxis:

CAST (expresión AS tipoDato)
CONVERT (tipoDato [(longitud)] , expresión [, estilo])

Funciones Nativas de SQL Server

Ejemplo usando CAST

```
SELECT Producto        = descripcion,
       Moneda          = CodMoneda,
       PrecioVenta     = PrecioVenta
FROM Porductos
WHERE CAST(PrecioVenta AS char(20)) LIKE '1%'
```

Ejemplo usando CONVERT

```
SELECT Producto        = descripcion,
       Moneda          = CodMoneda,
       PrecioVenta     = PrecioVenta
FROM Porductos
WHERE CONVERT(char(20), PrecioVenta) LIKE '1%'
```

PROGRAMACIÓN DE LA BASE DE DATOS

Definición y Componentes de un Batch

1.- Script y Batch

Un **Script** es un conjunto de batches ejecutados para realizar una operación específica, generalmente almacenados como archivos con extensión **sql**.

Un **Batch** es un grupo de sentencias SQL que son compiladas en un solo plan de ejecución y ejecutadas como un todo.

Si alguna sentencia que pertenece al batch posee un error de sintaxis que evita ser compilada, entonces la compilación del batch falla y ninguna de las sentencias se ejecutan. El comando **GO** (no es una sentencia Transact-SQL) indica el final de un batch en las herramientas **osql, isql** y **SQL Query Analyzer**.

PROGRAMACIÓN DE LA BASE DE DATOS

Definición y Componentes de un Batch

2.- Variables

TSQL soporta tipos de variables locales o variables globales; éste último tipo en SQL Server es llamado – Funciones del Sistema.

3.- Sentencias de Control de Flujo

Permiten tener control sobre el código TSQL que se ejecuta o se deja de ejecutar.

4.- Manejo de Errores

SQL Server soporta la capacidad de controlar los errores que surgen y tomar respuestas automáticas a condiciones predefinidas o simplemente puede asegurarse que el error sea mostrado al usuario para que tome las acciones apropiadas.

PROGRAMACIÓN DE LA BASE DE DATOS

Usando Variables

Existen dos tipos de variables:

- Variables locales definidas por el usuario
- Variables globales del sistema

Variables definidas por el usuario

Se declaran en el cuerpo de un batch o de un procedimiento mediante la sentencia DECLARE, y se le asigna un valor mediante una instrucción SET o SELECT.

Sintaxis de la sentencia DECLARE para declarar una variable

```
DECLARE {@VariableLocal tipoDato} [,...n]
```

Parámetro	Descripción
@variableLocal	Es el nombre de la variable. Comienzan con un signo de arroba (@) y deben seguir las reglas de los identificadores.
tipoDato	Definido por el usuario o proporcionado por el sistema a excepción de los tipos **text**, **ntext** o **image**.
N	Se puede añadir más variables con la misma sentencia DECLARE

PROGRAMACIÓN DE LA BASE DE DATOS

Sintaxis de la sentencia SELECT y SET para asignar un valor a una variable

---

SELECT {@ variableLocal = expresion } [,...n]

SET {@ variableLocal = expresion }

Parámetro	Descripción
@variableLocal	Es el nombre de la variable
Expresión	Es cualquier expresión válida.

Ejemplo: Empleando la sentencia SELECT para asignar un valor a una variable

---

```
DECLARE @Maximo int

/* Consulta el máximo código asignado a un cliente */
SELECT      @Maximo = MAX(CodCliente)
FROM        Clientes
```

PROGRAMACIÓN DE LA BASE DE DATOS

Variables del Sistema

Poseen el prefijo (@@) y para consultar su contenido se debe emplear la sentencia SELECT.

A continuación se detallan las variables del sistema que retornan información acerca de valores, objetos y configuraciones de SQL Server.

Variable	Descripción
@@ERROR	Devuelve el número de error de la última instrucción Transact-SQL ejecutada.
@@IDENTITY	Devuelve el último valor ingresado a una columna identity.
@@ROWCOUNT	Devuelve el número de filas afectadas por la última instrucción.
@@TRANCOUNT	Devuelve el número de transacciones activas de la conexión actual

PROGRAMACIÓN DE LA BASE DE DATOS

Variables del Sistema

A continuación se detallan algunas variables que retornan información acerca de las opciones de configuración actual.

Variable	Descripción
@@CONNECTIONS	Devuelve el número de conexiones o intentos de conexión desde la última vez que se inició SQL Server.
@@LANGUAGE	Devuelve el nombre del lenguaje en uso.
@@SERVERNAME	Devuelve el nombre del servidor SQL Server
@@SERVICENAME	Devuelve el nombre de la clave de registro bajo la cual se ejecuta SQL Server.
@@VERSION	Devuelve la fecha, versión y tipo de procesador de la instalación actual de SQL Server.

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencias de Control de Flujo

Sentencia	Descripción
BEGIN...END	Define un bloque de instrucciones
BREAK	Sale del bucle WHILE más cercano
CONTINUE	Reinicia el bucle WHILE.
GOTO etiqueta	Continúa el proceso en la instrucción que sigue a la etiqueta definida por *etiqueta*.
IF...ELSE	Define una ejecución condicional y, opcionalmente, una ejecución alternativa si la condición es FALSE.
RETURN	Sale incondicionalmente.
WAITFOR	Establece un tiempo de espera.
WHILE	Ejecuta un conjunto de instrucciones mientras una condición específica se cumpla.

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencias de Control de Flujo

Existen otras sentencias que se pueden utilizar con las sentencias de control de flujo, tales como:

Sentencia
/*...*/ (comentario)
-- (comentario)
CASE
DECLARE @ variableLocal
EXECUTE
PRINT
RAISERROR

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia IF - ELSE

Sintaxis

IF expresionBooleana
{instrucciónSQL | bloqueInstrucción}
ELSE
{instrucciónSQL | bloqueInstrucción}

Ejemplo:

```
IF (SELECT SUM(V.total)
    FROM      Productos P, Ventas V
    WHERE     P.CodProducto = V.CodProducto AND
              P.Descripcion = 'Windows 98') > 5000
  PRINT 'El producto Win98 registra ventas por más de $5000'
ELSE
 PRINT 'El producto Win98 registra ventas menor o igual a $5000'
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia BEGIN y END

Sintaxis

```
BEGIN
{
instrucciónSQL
| bloqueInstrucción
}
END
```

Ejemplo:

```
IF (@@ERROR <> 0)
   BEGIN
       SET @NumeroError = @@ERROR
       PRINT 'Error encontrado, ' +
       CAST(@NumeroError AS VARCHAR(10))
   END
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia WHILE

Sintaxis

WHILE expresiónBooleana
{instrucciónSQL | bloqueInstrucción}
[BREAK]
{instrucciónSQL | bloqueInstrucción}
[CONTINUE

Ejemplo:

```
DECLARE @Contador int
SET @Contador = 1
WHILE @Contador < 6
   BEGIN
       SELECT 'Esta sentencia se ejecutará 5 veces'
         SET @Contador = @Contador + 1
   END
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia GOTO

Sintaxis

Definición de la etiqueta:

etiqueta:

Alteración de la ejecución:

GOTO etiqueta

Ejemplo:

```
SELECT 'Este comando será ejecutado'
GOTO Salto

SELECT 'Este comando no será ejecutado'

Salto:
SELECT 'Este comando será ejecutado'
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia RETURN

Permite terminar inmediatamente la ejecución de un procedimiento almacenado, batch o bloque de sentencias. Las sentencias que se encuentran a continuación de RETURN no se ejecutan.

Sintaxis

RETURN [expresiónEntera]

Parámetro	Descripción
expresiónEntera	Es el valor entero que se retorna. Los procedimientos almacenados pueden retornar un valor entero al procedimiento o a la aplicación que realiza la llamada.

Ejemplo:

```
CREATE PROCEDURE ExisteProducto
@Codigo varchar(10)
AS
  IF EXISTS(SELECT Codigo
            FROM Productos
            WHERE codigo=@Codigo)
     RETURN 1
  ELSE
     RETURN 2
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia WAITFOR

Se establece un tiempo, intervalo de tiempo o suceso de espera previo a la ejecución de un bloque de sentencias, procedimiento almacenado o transacción.

Sintaxis

```
WAITFOR { DELAY 'tiempo' | TIME 'tiempo' }
```

Parámetro	Descripción
DELAY	Es un indicador que especifica un tiempo, intervalo de tiempo o suceso de espera hasta un máximo de 24 horas.
`'tiempo'`	Es el tiempo o intervalo de tiempo de espera. El argumento *tiempo* se puede especificar en cualquier formato del tipo de dato **datetime** o en una variable local.
`TIME`	Especifica una espera hasta una hora especificada.

Ejemplo:

El siguiente ejemplo establece una espera de 000 horas, 00 minutos y 10 segundos.

```
WAITFOR DELAY '000:00:10'
```

PROGRAMACIÓN DE LA BASE DE DATOS

Sentencia CASE

Evalúa un conjunto de condiciones y retorna como resultado una expresión.

CASE tiene dos formatos:

- La función CASE sencilla evalúa sólo una expresión con un conjunto de expresiones sencillas para determinar el resultado.
- La función CASE de búsqueda evalúa un conjunto de expresiones booleanas para determinar el resultado.

Ambos formatos aceptan el argumento ELSE opcional.

Sintaxis función CASE sencilla:

```
CASE expresiónEntrada
WHEN expresiónCuando THEN expresiónResultado [...n]
[ELSE expresiónResultadoElse]
END
```

Sintaxis función CASE búsqueda:

```
CASE
WHEN expresiónBooleana THEN expresiónResultado [...n]
[ELSE expresiónResultadoElse]
END
```

PROGRAMACIÓN DE LA BASE DE DATOS

Ejemplo se CASE sencilla:

```
SELECT Codigo,
       Nombre,
       Tipo= CASE Tipo
                  WHEN 'N' THEN 'Nacional'
                  WHEN 'E' THEN 'Extranjero'
                  ELSE 'Otro'
             END
FROM Proveedores
```

Ejemplo se CASE búsqueda:

```
SELECT Codigo,Cantidad,Precio,
       'Nombre Zona' =
           CASE WHEN Zona >= 1 AND Zona <=10 THEN 'Zona Norte'
                WHEN Zona >= 11 AND Zona <=20 THEN 'Zona Centro'
                WHEN Zona >= 21 AND Zona <=50 THEN 'Zona Sur'
                ELSE 'Zona Indefinida'
           END
FROM Ventas
```

TRIGGERS

Creación de Triggers

Los triggers son usados frecuentemente para forzar las reglas del negocio y la integridad referencial de los datos. Microsoft ® SQL Server™ permite la creación de múltiples triggers para cualquier sentencia INSERT, UPDATE, DELETE.

Sintaxis

```
CREATE TRIGGER trigger_name
ON tabla
{FOR {[,] [DELETE] [,] [INSERT] [,] [UPDATE] }
AS sentencias_sql [... n]
```

TRIGGERS

Tipos de TRIGGERS

Un trigger puede acceder a las imágenes anterior y posterior de los datos por medio de las tablas lógicas (seudotablas especiales).

Triggers	Tabla lógica
INSERT	INSERTED
DELETE	DELETED
UPDATE	INSERTED-DELETED

Estas dos tablas tienen las mismas columnas que la tabla subyacente que se esta cambiando. Es posible comprobar los valores de anterior y posterior de columnas específicas y tomar acciones en funciones de lo que se encuentre. Estas tablas no son estructuras físicas (SQL Server las construye a partir del registro de las transacciones). Ese es el motivo por el que operaciones no registradas como una copia masiva o SELECT INTO no provocan la activación de los triggers.

TRIGGERS

Limitaciones de los TRIGGERS

Un trigger es creado sólo en la base de datos actual. No obstante un trigger puede referenciar a objetos de otras bases de datos.

La misma acción trigger puede ser definido por más de una acción de usuario (por ejemplo INSERT y UPDATE) en la misma sentencia CREATE TRIGGER.

Un trigger no puede ser creado en una vista.

Esta es la lista de sentencias Transact-SQL que no permite un trigger:

ALTER DATABASE	CREATE TRIGGER	DROP VIEW
ALTER PROCEDURE	CREATE VIEW	GRANT
ALTER TABLE	DENY	LOAD DATABASE
ALTER TRIGGER	DISK INIT	LOAD LOG
ALTER VIEW	DISK RESIZE	RESTORE DATABASE
CREATE DATABASE	DROP DATABASE	RESTORE LOG
CREATE DEFAULT	DROP DEFAULT	REVOKE
CREATE INDEX	DROP INDEX	RECONFIGURE
CREATE PROCEDURE	DROP PROCEDURE	TRUNCATE TABLE
CREATE RULE	DROP RULE	UPDATE STATISTICS
CREATE SCHEMA	DROP TABLE	
CREATE TABLE	DROP TRIGGER	