Bases de Datos Relacionales

Bases de datos relacionales

</digitalmind>

- Una base de datos es una forma de almacenamiento persistente, sostenido en el concepto de sistematización de los datos que se almacenan. Siendo estos de un mismo origen (entorno) y representando una determinada realidad, la cual tiene un limite de entendimiento preestablecido en su diseño.
- Los datos pueden o no estar almacenados en formato digital y con un sistema informático.

- **Almacenamiento Digital**: Implica un sistema que organiza la información dentro de un medio de almacenamiento computarizado, utilizando Motores de base de datos (DBMS) tales como SqlServer, Oracle, DB2, MySql, etc.

- **Almacenamiento no Digital**: Organización de información que está sujeta a una sistematización que no implica la utilización de medios informáticos, tales como: Biblioteca Pública, Ficheros de historias clínicas de consultorios médicos.

Bases de datos relacionales

</digitalmind>

- Un motor de base de datos relacional (RDBMS), es el software que administra una base de datos relacional
- El modelo relacional presenta vistas tabulares (filas y columnas) de los datos almacenados.
- Para diseñar un modelo de datos relacional se utiliza una serie de normas o reglas.
- Esta normas se denominan “Formas Normales” y son las que establecen el fundamento teórico de la “Normalización”

Relacionales

</digitalmind>

- El modelo relacional esta basado en una estructura denominada "**Relación**".
- La Relación tiene una estructura similar a una tabla bidimensional, sólo que con mayor nivel de abstracción.
- La tabla está compuesta por filas y columnas y la relación por tuplas y atributos.
- La Relación cumple con un conjunto de restricciones que la diferencian de una tabla común.

Relacionales

- Las restricciones a las que está sometida una relación:
  - Las relaciones poseen un nombre propio irrepetible en el modelo al que pertenecen.
  - Los atributos dentro de una relación tiene también nombre propio irrepetible.
  - Los atributos no tiene un orden especifico dentro de la relación.
  - Las tuplas son conjuntos únicos. Una tupla representa un único conjunto de valores.
  - Las tuplas no tienen ningún orden especifico.

Clave Primaria

</digitalmind>

- La forma para poder individualizar y ubicar inequívocamente una tupla es la **clave primaria**. Por lo tanto, la clave primaria en una forma de localización de una tupla.
- La clave primaria es el **atributo**, o conjunto de estos, que permite una individualización, es decir la localización de una tupla dentro del espectro de posible tuplas de una relación.
- El valor de la clave primaria debe ser único en toda la vida de la relación.
- Si en la relación se observan distintas estructuras que permiten planificar más de una clave primaria, entonces éstas se denominan **claves candidatas** y sólo una de ellas será clave primaria.
- Entonces una clave primaria es también una clave candidata.
- En una relación de múltiples claves candidatas, sólo uno de ellas se llamará clave primaria y las demás alternas.

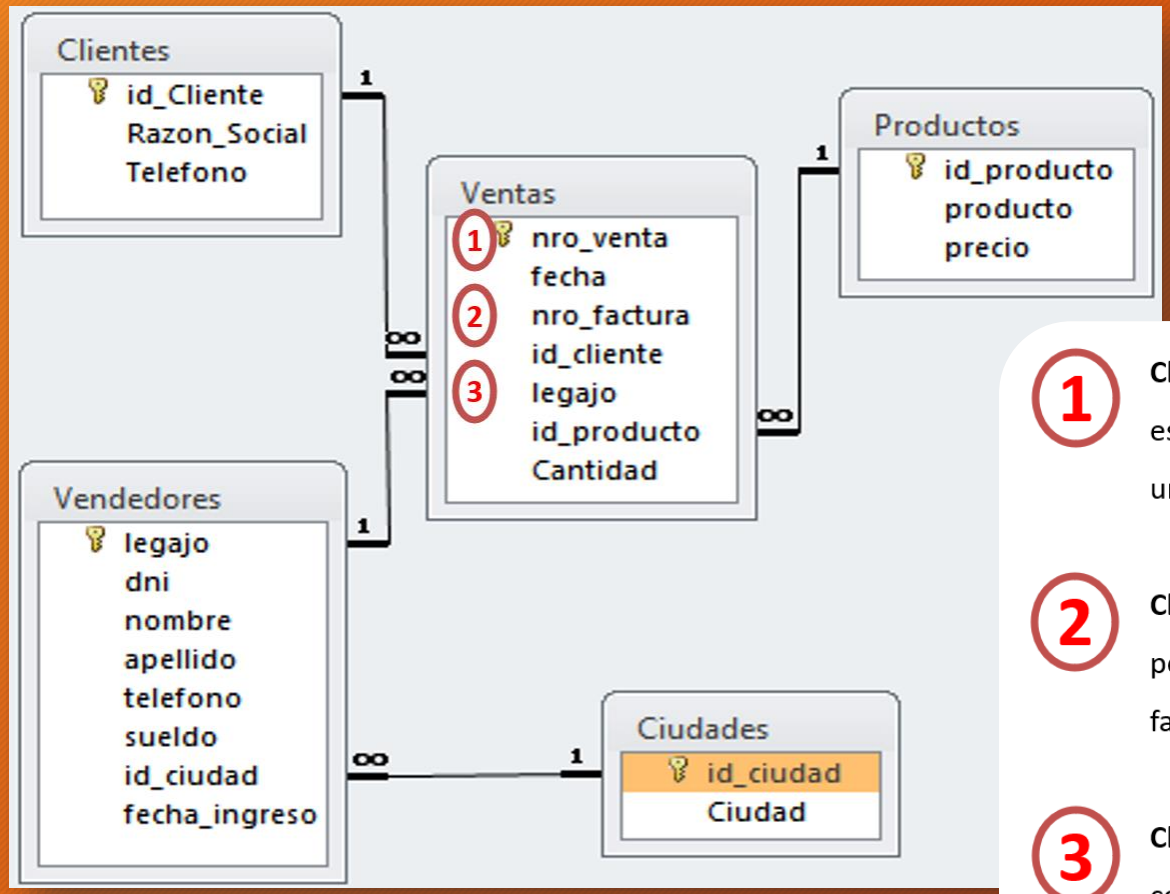
Clave Primaria

</digitalmind>

- Los atributos que participan de la clave primaria no pueden causar incertidumbre y por ello siempre toman valores ciertos.
- Una restricción de integridad es que los valores que componen la clave primaria no pueden ser **valores nulos**.
- Nulo, significa que ese atributo para la tupla en cuestión es desconocido. Nulo no es cero “0” para una atributo que pertenece a un dominio de tipo numérico, ni tampoco espacios en blanco para un atributo que pertenece a un dominio alfanumérico. Simplemente nulo es la inexistencia de un valor posible.

Clave Foránea

- **Clave Foránea** es una estructura que permite asociar dos relaciones.
- Esta asociación se produce entre la relación donde se declara la clave foránea y la relación que posee la clave primaria.
- Si se declara una clave foránea en una relación “A” que se relaciona con otra relación “B”, la clave primaria estructuralmente debe ser igual a la clave foránea de “A”, para poder establecer la vinculación.
- La clave foránea en una relación se declara sobre la totalidad de los atributos de la clave primaria de otra relación.
- La clave foránea al no ser un elemento de individualización de la relación puede repetirse ilimitadamente.
- La clave foránea también admite valor nulo. Esto trae como consecuencia que la tupla que posee valor nulo no se encuentra vinculada con la relación hacia la cual apunta la clave foránea. Esta situación puede ser revertida en cualquier momento.



1 **Claves Primaria**: En cada Relacion (tabla) hay una clave primaria (en este caso identificadas con una llave), la cual permite identificar univocamente a cada tupla (fila) de la relacion.

2 **Clave Candidatas**: En cada Relacion puede haber varios datos que permitan identificar univocamente una tupla (como el nro de factura en la tabla ventas o el dni en la tabla vendedores)

3 **Clave Foreanea**: Son los campos de una relacion que se relacionan con la clave primaria de otra tabla. EJ: una venta corresponde a un vendedor, los datos del vendedor estan almacenados en la tabla Vendedores entonces en la tabla Ventas se registra el legajo del vendedor (sera una clave foranea) que apunte al legajo de la tabla de Vendedores (donde es Clave Primaria)

Normalizaciones

¿Que es una normalización en BD?

Es una técnica que ayuda a desarrollar la estructura lógica de los datos de un sistema informático en el modelo relacional.

Fue introducida por E.F. Codd en 1972 donde define una estrategia de diseño de abajo hacia arriba, partiendo de los atributos y estos se van agrupando en tablas según su afinidad.

Los objetivos de la normalización son:

1. Controlar la redundancia de la información.
2. Evitar pérdidas de información.
3. Mantener la consistencia de los datos.
4. Capacidad para representar toda la información.

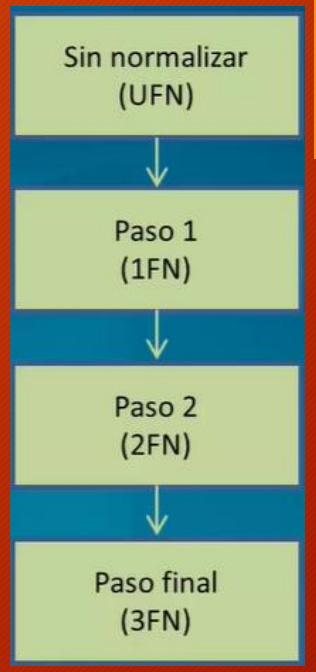
Pasos para realizar la normalización

Al aplicar esta técnica nos debemos asegurar que cada relación o tabla cumple con ciertas reglas o condiciones.

Si alguna regla o condición no se cumple, entonces esa tabla se debe descomponer en varias relaciones que si cumplan con esa regla.

Normalmente, con realizar una normalización de 1era, 2da, y 3era forma es suficiente para tener un modelo estable y menos susceptible a las anomalías estudiadas en este curso.

</digitalmind>



- **Pasos para realizar la normalización?**

1. Descomponer todos los grupos de datos en atributos atómicos (1FN), además de que cada tabla debe poseer su PK.

2. Eliminar todas las relaciones en la que los datos no dependan completamente de la llave primaria de la tabla (2FN).

3. Eliminar todas las relaciones que contengan dependencias transitivas (3FN).

CLIENTE		
PK	IDCliente	Numerico
	Nombre	Cadena
	Apellido	Cadena
	Direccion	Cadena
	Cedula	Cadena
	DOB	Fecha
	Email	Cadena

ID	Nombe
1	Miguel Fagundez
2	Carlos Gonzalez
3	Luis Alvarado
4	Andres Parra
5	Andrea Colmenares
6	Luisa Vialli

ID	Nombe	Apellido
1	Miguel	Fagundez
2	Carlos	Gonzalez
3	Luis	Alvarado
4	Andres	Parra
5	Andrea	Colmenares
6	Luisa	Vialli

SQL Commands

</digitalmind>

DDL

- CREATE
- ALTER
- DROP
- TRUNCATE
- RENAME

DML

- SELECT
- INSERT
- UPDATE
- DELETE

DCL

- GRANT
- REVOKE

TCL

- COMMIT
- ROLLBAK
- SET TRANSACTION

COMO CREAR UNA BASE DE DATOS

Archivos de la BD

Principal o Main Data File con extensión (.**mdf**) es el archivo de datos principal donde se encuentra la información inicial de la base de datos, en este archivo se almacenan los datos y los objetos del usuario, aunque también se puede realizar en archivos secundarios, toda base de datos cuenta con un archivo principal.

Secundarios o Secondary Data File con extensión (.**ndf**) son los archivos de datos secundarios donde la base de datos puede contener uno o varios, se pueden usar opcional, están definidos por el usuario y almacenan los datos de los usuarios, se pueden utilizar para repartir la información en varios discos agregando cada archivo en una unidad de disco diferente, además si una base de datos supera el límite de almacenamiento para un archivo de Windows se podrán utilizar archivos de datos secundarios para el crecimiento de la misma.

De registro o Log Data File con extensión (.log) son archivos del registro de transacciones, contiene la información de se utiliza para la recuperación de la base de datos cada BD debe contar con al menos un archivo de registro..

```
CREATE DATABASE [Northwind]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'Northwind',
    FILENAME = N'E:\datos\MSSQL12.MSSQLSERVER\MSSQL\DATA\northwnd.mdf'
    , SIZE = 7360KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'Northwind_log',
    FILENAME = N'E:\datos\MSSQL12.MSSQLSERVER\MSSQL\DATA\northwnd.ldf'
    , SIZE = 1536KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
```

Comandos DDL que usamos para la definición y creación de objetos en la base de datos (Tabla, Procedimiento, Vistas...). Estos comandos se utilizan principalmente para diseñar y definir la estructura de nuestra base de datos.

</digitalmind>

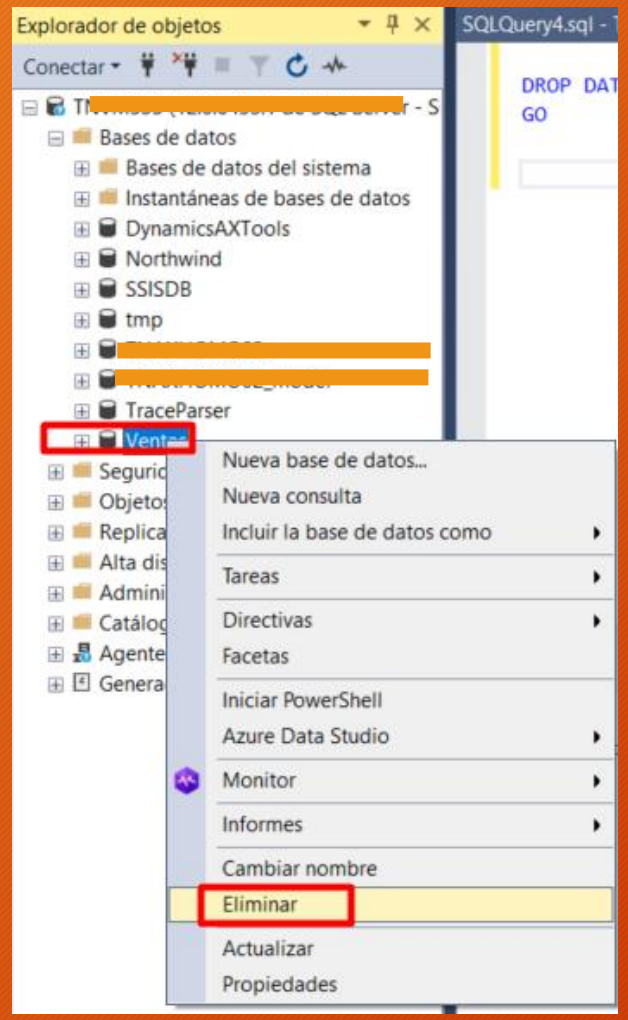
CREATE - we can create a new table, database, procedure, view, trigger...

ALTER - usually we use for editing database objects (table, procedure, view...) for example, add or delete column from table

DROP - we use for deleting database objects

RENAME is used to rename an object existing in database.

- Borrar la BD Desde la herramienta visual de **SQL Management Studio**



- Desde el editor de consultas **Transact SQL**

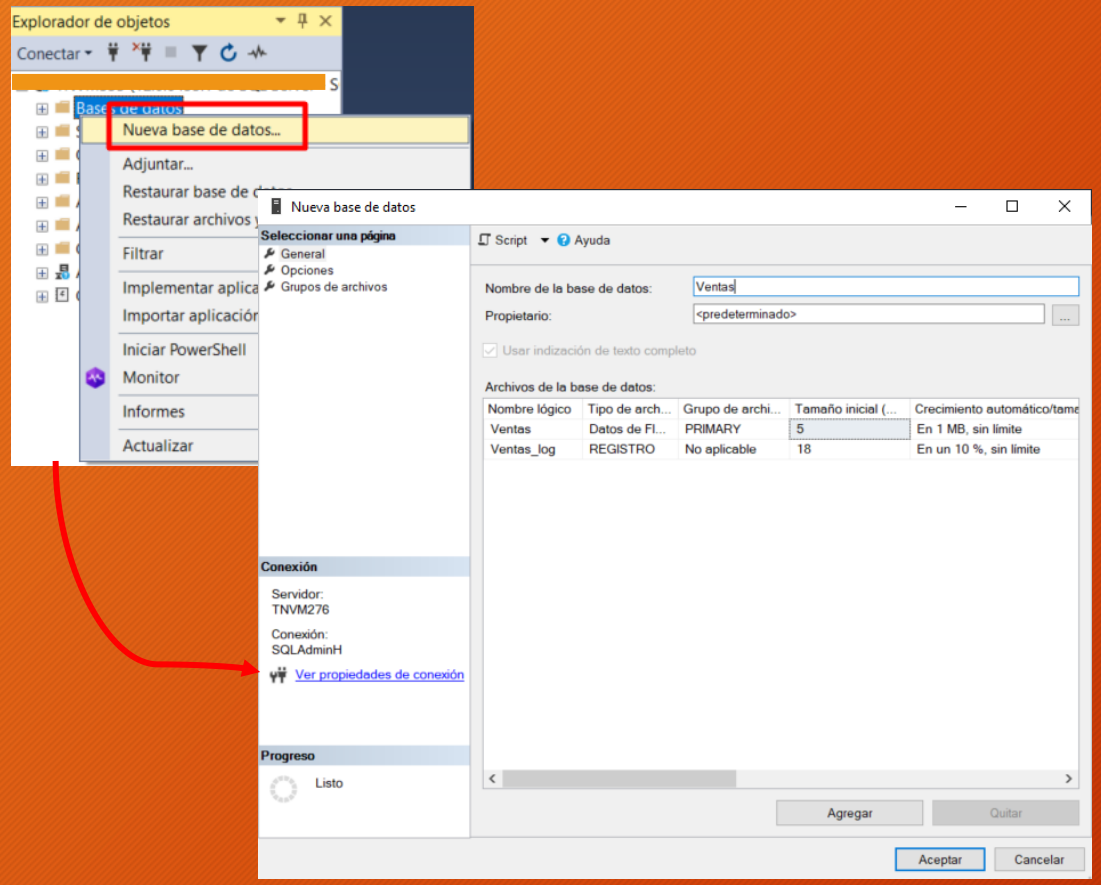
```
DROP DATABASE Ventas
GO
```

- Renombrar una **BD**

```
ALTER DATABASE Ventas SET SINGLE_USER WITH
ROLLBACK IMMEDIATE;
GO
ALTER DATABASE Ventas MODIFY NAME = VentasWeb;
GO
ALTER DATABASE VentasWeb SET MULTI_USER;
GO
```

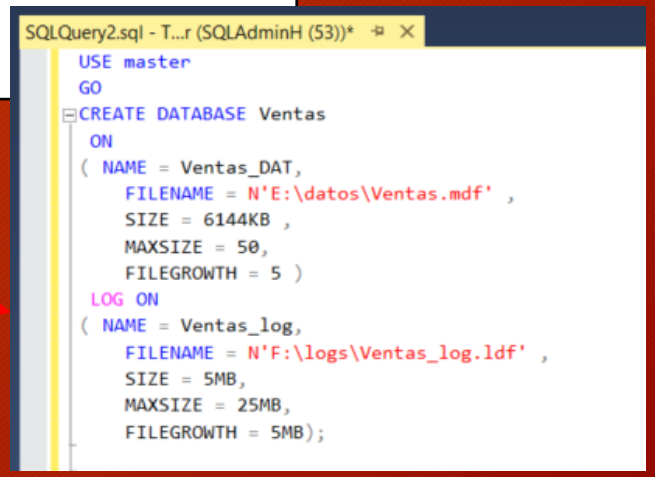
Ejemplo: Nos solicitan crear una BS llamada Ventas

- Desde la herramienta visual de **SQL Management Studio**



- Desde el editor de consultas **Transact**

```
USE master
GO
CREATE DATABASE Ventas
 ON
( NAME = Ventas_DAT,
FILENAME = N'E:\datos\Ventas.mdf',
SIZE = 6144KB ,
MAXSIZE = 50,
FILEGROWTH = 5 )
 LOG ON
( NAME = Ventas_log,
FILENAME = N'F:\logs\Ventas_log.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB);
```



Breve explicación sobre estos comandos:

```
4   -- SELECT
5   SELECT *
6   FROM employee
7
8   -- INSERT
9   INSERT INTO employee(emp_id, fname, minit, lname,
10                       job_id, job_lvl, pub_id, hire_date)
11              VALUES('0000000', 'Almir', 'M', 'Vuk',
12                       7, 12, 1207, 2009-05-09)
13
```

Con el **SELECT** podemos ver una sintaxis muy simple, después de escribir la palabra clave **SELECT** con el signo de asterisco (que reemplaza TODAS las columnas), por lo que con este comando en nuestra salida obtendremos todas las columnas del empleado de la tabla.

```
/*Ej. Sobre la base de datos DBDigital, inserte un nuevo registro
sobre la tabla Ciudad, con los valores IdCiudad = 1 y Ciudad = "Cordoba"  */
USE DBDigital
GO
INSERT INTO Ciudad(IdCiudad,Ciudad) VALUES(1,'Córdoba')
GO
```

El comando **INSERT** también es muy simple, después de la palabra clave **INSERT** seleccionamos el nombre de la tabla (la palabra clave **INTO** es opcional) después del nombre de la tabla entre paréntesis podemos especificar qué columnas usaremos en esta inserción, luego ponemos la palabra clave **VALUES** y luego entre paréntesis escribiendo los valores que queremos insertar. La disposición de las columnas en la parte del **INSERT** debe ser la misma que en la parte de VALUES para que podamos insertar los valores correctos en las columnas correctas

UPDATE es muy útil cuando queremos cambiar algunos datos en nuestro registro en la tabla, este comando tiene una sintaxis muy simple, como podemos ver en la imagen de arriba... después de la palabra clave **UPDATE**, enumeramos el nombre de la tabla después de eso con la palabra clave **SET** y la columna que queremos cambiar y luego asignar el signo con nuevos valores (podemos combinar más de una columna)... después de eso, lo más importante es usar la cláusula **WHERE** para especificar criterios para el registro que queremos cambiar que puede ser, por ejemplo, un ID de Persona en la tabla

</digitalmind>

```
13
14  -- UPDATE
15  UPDATE employee
16  SET fname = 'ALMIR'
17  WHERE emp_id = '0000000'
18
19  -- DELETE
20  DELETE
21  FROM employee
22  WHERE emp_id = '0000000'
```

```
/*Ej. Sobre la base de datos BDDigital, actualice el valor
de Ciudad "SALTA" para el IdCiudad = 1 */
UPDATE Ciudad
SET Ciudad = 'SALTA'
WHERE IdCiudad = 1
GO
```

```
/*Ej. sobre la base DBDigital, elimine la ciudad "SALTA" */
DELETE FROM Ciudad
WHERE IdCiudad = 1
GO
```

DELETE es un comando muy poderoso y simple, después de la palabra clave **DELETE** enumeramos el nombre de la tabla de la que queremos eliminar el registro y luego lo más importante es usar la cláusula **WHERE** para especificar los criterios para el registro que queremos cambiar que pueden ser por ejemplo, una identificación de persona en la tabla (igual que en ACTUALIZAR).

Resumen de Consultas SQL

```
SELECT < * | Lista_Columnas>
FROM <Lista_Tablas>
WHERE <Condición_Filtro>
```

COLUMNAS CACULADAS: permite definir columnas que no consumen espacio de almacenamiento, se denominan columna virtuales.

Ej1.

```
/*Ej. Sobre la Base de Datos DBCelsius, obtenga el Total Ventas
que se obtienen sobre la columnas Precio y Cantidad, para todas
las ventas realizadas posterior al 01/01/2021 */
USE DBCelsius
GO
SELECT item, producto, marca
      , (precio * cantidad) AS [Total Ventas]
FROM VENTAS
WHERE marca = 'Philips'
      AND [Fecha Venta] >= '01/01/2021'
```

USO DE LITERALES: use literales para lograr una salida mas legible, un literal es una cadena de caracteres rodeada por comillas simples o dobles.

Ej1.

```
      USE DBDigital
      SELECT IdCiudad "Cod.Ciudad", Ciudad "Provincia"
      FROM Ciudad
      GO
```

Ej2.

```
/*Query4 - Mostrar las columnas title, price y pubdate
de la tabla Titles, con encabezados mas legibles para el usuario,
por ejemplo --> price = "Precio Unitario" */

USE pubsDB
SELECT title_id AS IDENTIFICADOR
      , price AS 'PRECIO UNITARIO'
      , pubdate AS [FECHA DE PUBLICACION]
FROM titles
```

Test de Comparación:

/*Ej. - Mostrar de la tabla titles, las columnas
title_id, title y pubdate. Donde la fecha de publicación
sea mayor a 1990.-*/

```
use pubsDB
SELECT title_id, title, pubdate
FROM titles
WHERE pubdate > 1990
--WHERE pubdate > '31-12-1989'
```

Test de Pertenencia en un conjunto o Listas:

/*Ej - De la Tabla Titles, mostrar las columnas title_id, title y price, cuyos Title_Id sean 'BU1032', 'PC1035', 'PS3333'*/

```
SELECT title_id,title, price
FROM Titles
WHERE title_id IN ('BU1032', 'PC1035', 'PS3333' )
--WHERE title_id NOT IN ('BU1032', 'PC1035', 'PS3333' )
```

Test de Rango:

/*Ej - De la Tabla Titles, mostrar las columnas title_id, title y price, donde el price sea mayor a 20 y menor a 50 */

```
use pubsDB
SELECT title_id,title, price
FROM Titles
WHERE price BETWEEN (20) AND (50)
--WHERE price >= 20 AND price <= 50
```

Test de Correspondencia con un patrón:

/*Ej - De la Tabla Titles, mostrar las columnas title_id
, title y type, donde el valor de la columna type sea
igual a "Business" */

```
SELECT [title_id], [type], [title] FROM titles
WHERE type LIKE 'business%'
--WHERE type NOT LIKE 'business%' –Negamos el Like
```

</digitalmind>

Test de Valores Nulos:

/*Ej - De la Tabla Titles, mostrar las columnas title_id, title y price, cuyos Title_Id sean 'BU1032', 'PC1035', 'PS3333'*/

```
SELECT * FROM
[dbo].[titles]
WHERE royalty IS NULL
```

Test de Ordenamiento:

/*Query 7 - Mostrar todas las columnas de la tabla Sales, ordenadas de forma ascendente teniendo como indicar la columna payterms*/

```
SELECT *
FROM sales
ORDER BY payterms <ASC> <DESC>

SELECT stor_id, ord_num, payterms, title_id
FROM sales
ORDER BY 3 <ASC> <DESC>

SELECT stor_id, ord_num, payterms, title_id
FROM sales
ORDER BY 3 ASC
```

Test de Condiciones Compuestas:

- **AND**
- **OR**
- **NOT**

/*Ej - De la Tabla Titles, mostrar todas las columnas, cuyos Type = Business y los Precio se encuentre entre 10.90 o 18,99*/

```
SELECT * FROM
[dbo].[titles]
WHERE type = 'Business' AND price > 10.90 OR price < 18.99

SELECT * FROM
[dbo].[titles]
WHERE type = 'Business' AND NOT(price > 10.90)
```

Test de Conteos:

/*Ej - De la Tabla Employees, indicar la cantidad de empleados que se encuentra en USA*/

```
Use Northwind
SELECT COUNT(*) AS 'Empleados USA'
FROM Employees
WHERE Country = 'usa'
```

</digitalmind>

/*Ej - De la tabla Sales y Titles, mostrar las columnas title_id, ord_num (de la tabla Sales),titles,
type, price (de la tabla Titles), agregando una columna calculada entre (Qty * Price) con la etiqueta Importe.
teniendo en cuenta que el ord_num es el p2121.- */

```
SELECT sales.title_id, sales.ord_num, titles.title, titles.type, titles.price, Importe = (titles.price * sales.qty)
FROM sales, titles
WHERE
	sales.title_id = titles.title_id
	AND sales.ord_num = 'P2121’
```

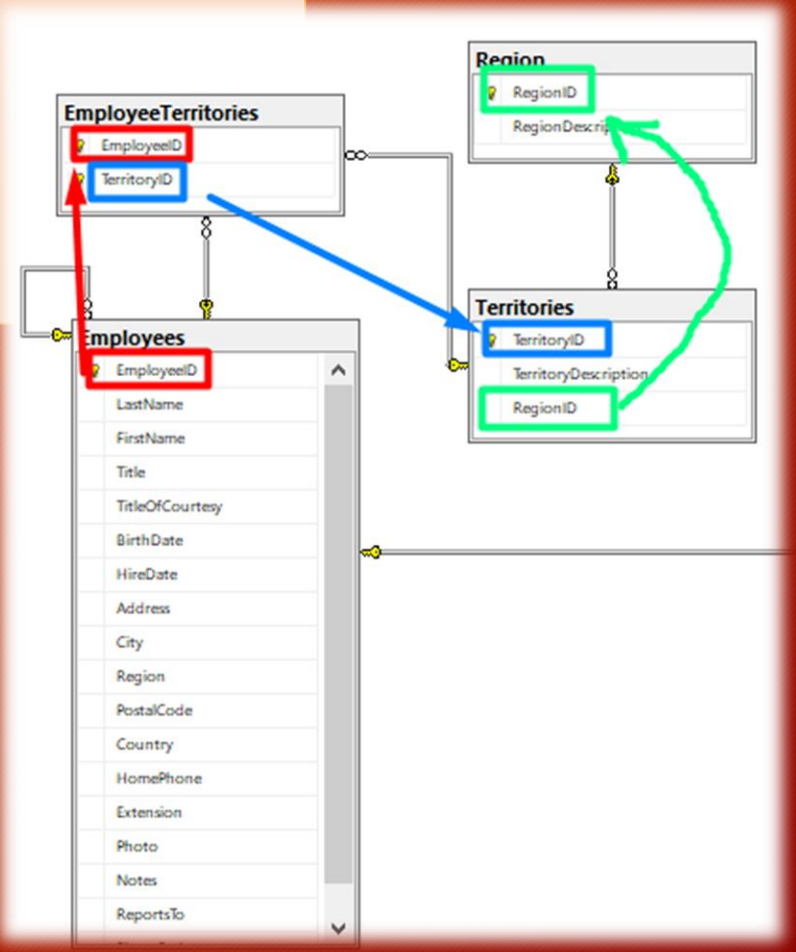
Resultado:

	title_id	ord_num	title	type	price	Importe
1	TC3218	P2121	Onions, Leeks, and Garlic: Cooking Secrets of t...	trad_cook	20,95	838,00
2	TC4203	P2121	Fifty Years in Buckingham Palace Kitchens	trad_cook	11,95	239,00
3	TC7777	P2121	Sushi, Anyone?	trad_cook	14,99	299,80

```
/*Ej -.- */
Use Northwind
SELECT Employees.EmployeeID, Employees.FirstName, EmployeeTerritories.TerritoryID
      , EmployeeTerritories.EmployeeID, Territories.TerritoryDescription
      , Region.RegionDescription
FROM Employees , EmployeeTerritories, Territories, Region
WHERE Employees.EmployeeID = EmployeeTerritories.EmployeeID
      AND EmployeeTerritories.TerritoryID IN(30346,44122,60601)
      AND Territories.TerritoryID = EmployeeTerritories.TerritoryID
      AND Territories.RegionID = Region.RegionID
```

Resultado:

	EmployeeID	FirstName	TerritoryID	EmployeeID	TerritoryDescription	RegionDescription
1	3	Janet	30346	3	Atlanta	Southern
2	7	Robert	60601	7	Chicago	Western
3	8	Laura	44122	8	Beachwood	Northern

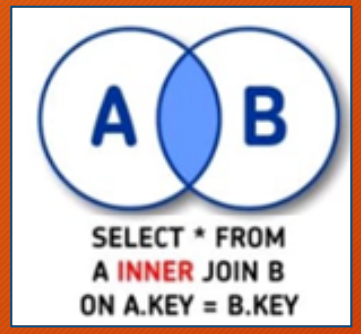


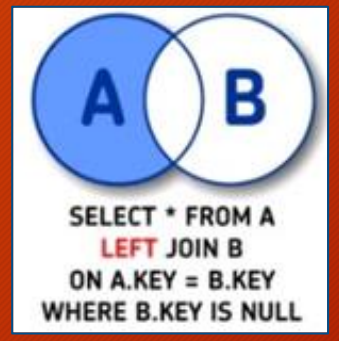
SQL JOINS

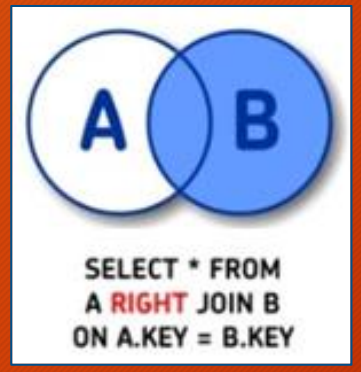
La sentencia JOIN permite combinar dos conjuntos de datos con una determinada condición.

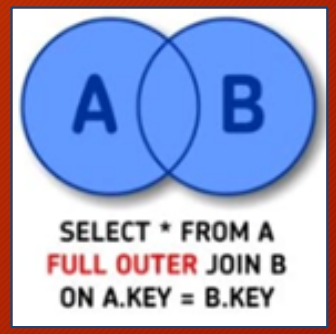
Tipos de JOINS:

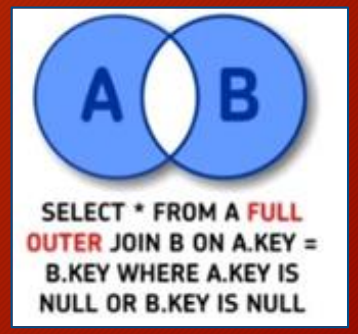
- ❑ JOIN
- ❑ INNER JOIN
- ❑ LEFT JOIN
- ❑ RIGHT JOIN
- ❑ FULL OUTER JOIN
- ❑ CROSS JOIN











JOIN: Esta cláusula está diseñada precisamente para reunir registros de varias tablas, en ella intervienen las claves primarias y foráneas, y no intervienen, o lo hacen en la cláusula WHERE, los filtros propiamente dichos.

Sintaxis:
SELECT p.ProductID, m.Name AS Model, p.Name AS Product
FROM SalesLT.Product AS p **JOIN** SalesLT.ProductModel AS m **ON** p.ProductModelID = m.ProductModelID

</digitalmind>

/* UNIONES BASADAS EN ANSI SQL-92 */

```
SELECT DISTINCT E.EmployeeID,E.FirstName, e.Title ,E.City, ET.TerritoryID
FROM Employees E  JOIN EmployeeTerritories ET ON E.EmployeeID = ET.EmployeeID
```

/* UNIONES BASADAS EN ANSI SQL-89*/

```
SELECT DISTINCT E.EmployeeID,E.FirstName, e.Title ,E.City, ET.TerritoryID
FROM Employees E, EmployeeTerritories ET
WHERE E.EmployeeID = ET.EmployeeID
```

EmployeeID	FirstName	Title	City	TerritoryID
1	Nancy	Sales Representative	Seattle	06897
1	Nancy	Sales Representative	Seattle	19713
2	Andrew	Vice President, Sales	Tacoma	01581
2	Andrew	Vice President, Sales	Tacoma	01730
2	Andrew	Vice President, Sales	Tacoma	01833
2	Andrew	Vice President, Sales	Tacoma	02116
2	Andrew	Vice President, Sales	Tacoma	02139
2	Andrew	Vice President, Sales	Tacoma	02184
2	Andrew	Vice President, Sales	Tacoma	40222
3	Janet	Sales Representative	Kirkland	30346
3	Janet	Sales Representative	Kirkland	31406
3	Janet	Sales Representative	Kirkland	32859
3	Janet	Sales Representative	Kirkland	33607
4	Margaret	Sales Representative	Redmond	20852

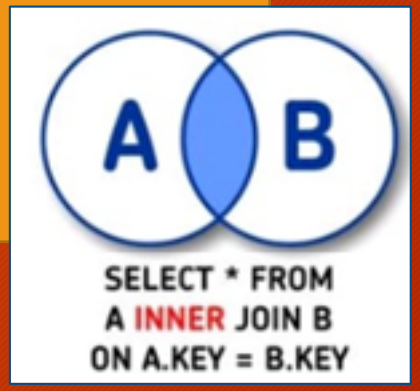
(49) filas afectadas)

* La sintaxis de ANSI SQL-92 dificulta la creación de productos cartesianos accidentales. Una vez que se ha agregado la palabra clave JOIN, se producirá un error de sintaxis si falta una cláusula ON, a menos que JOIN se especifique como CROSS JOIN.

INNER JOIN: Combina los registros de dos tablas si hay valores coincidentes en un campo común.

Sintaxis:

SELECT <lista de campos>
FROM tabla1 INNER JOIN tabla2 ON tabla1.campo1 *<operador_de_comparación>* tabla2.campo2



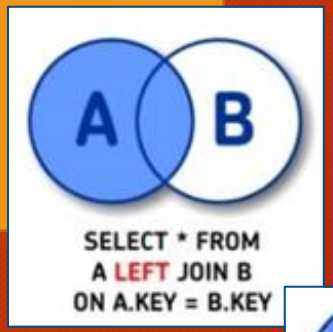

```
USE AdventureWorks2016
SELECT p.ProductID, m.Name AS Model, p.Name AS Product
FROM Production.Product AS p INNER JOIN Production.ProductModel AS m
   ON p.ProductModelID = m.ProductModelID
```

ProductID	Model	Product
680	HL Road Frame	HL Road Frame - Black, 58
706	HL Road Frame	HL Road Frame - Red, 58
707	Sport-100	Sport-100 Helmet, Red
708	Sport-100	Sport-100 Helmet, Black
709	Mountain Bike Socks	Mountain Bike Socks, M
710	Mountain Bike Socks	Mountain Bike Socks, L
711	Sport-100	Sport-100 Helmet, Blue
712	Cycling Cap	AWC Logo Cap
713	Long-Sleeve Logo Jersey	Long-Sleeve Logo Jersey, S
714	Long-Sleeve Logo Jersey	Long-Sleeve Logo Jersey, M
715	Long-Sleeve Logo Jersey	Long-Sleeve Logo Jersey, L
716	Long-Sleeve Logo Jersey	Long-Sleeve Logo Jersey, XL
717	HL Road Frame	HL Road Frame - Red, 62
718	HL Road Frame	HL Road Frame - Red, 44

LEFT JOIN: Indica que muestre todos los resultados de la columna de la izquierda.

Sintaxis:

SELECT <lista de campos>
FROM tabla1 LEFTJOIN tabla2 ON tabla1.campo1 <operador_de_comparación> tabla2.campo2



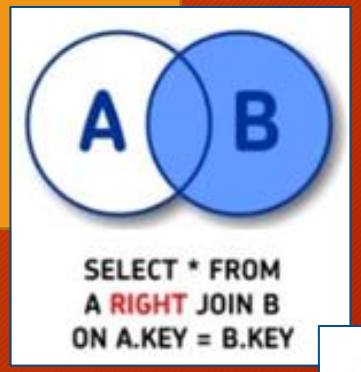

```
USE Northwind
SELECT P.ProductID, P.ProductName, P.UnitPrice
      ,P.CategoryID, C.CategoryName
FROM Products P LEFT JOIN Categories C ON P.CategoryID  = C.CategoryID
```

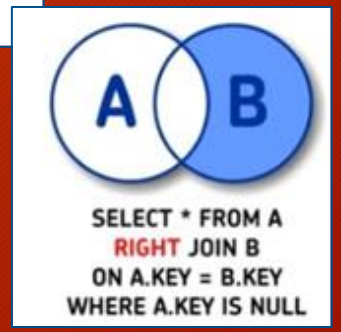
ProductID	ProductName	UnitPrice	CategoryID	CategoryName
1	Chai	18,00	1	Beverages
2	Chang	19,00	1	Beverages
3	Aniseed Syrup	10,00	2	Condiments
4	Chef Anton's Cajun Seasoning	22,00	2	Condiments
5	Chef Anton's Gumbo Mix	21,35	2	Condiments
6	Grandma's Boysenberry Spread	25,00	2	Condiments
7	Uncle Bob's Organic Dried Pears	30,00	7	Produce
8	Northwoods Cranberry Sauce	40,00	2	Condiments
9	Mishi Kobe Niku	97,00	6	Meat/Poultry
10	Ikura	31,00	8	Seafood
11	Queso Cabrales	21,00	4	Dairy Products
12	Queso Manchego La Pastora	38,00	4	Dairy Products
13	Konbu	6,00	8	Seafood
14	Tofu	23,25	7	Produce

RIGHT JOIN: Indica que muestre todos los resultados de la columna de la derecha

Sintaxis:

SELECT <lista de campos>
FROM tabla1 RIGHT JOIN tabla2 ON tabla1.campo1 <operador_de_comparación> tabla2.campo2





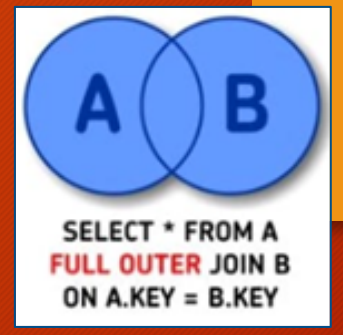

```
USE Northwind
SELECT P.ProductID, P.ProductName, P.UnitPrice
      ,P.CategoryID, C.CategoryName
FROM Products P RIGHT JOIN Categories C ON P.CategoryID  = C.CategoryID
```

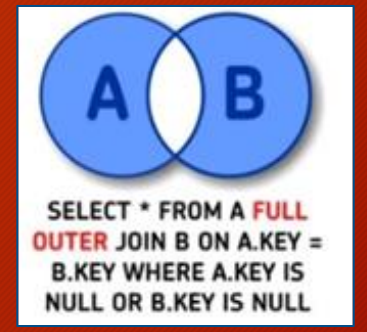
ProductID	ProductName	UnitPrice	CategoryID	CategoryName
1	Chai	18,00	1	Beverages
2	Chang	19,00	1	Beverages
24	Guaraná Fantástica	4,50	1	Beverages
34	Sasquatch Ale	14,00	1	Beverages
35	Steeleye Stout	18,00	1	Beverages
38	Côte de Blaye	263,50	1	Beverages
39	Chartreuse verte	18,00	1	Beverages
43	Ipoh Coffee	46,00	1	Beverages
67	Laughing Lumberjack Lager	14,00	1	Beverages
70	Outback Lager	15,00	1	Beverages
75	Rhönbräu Klosterbier	7,75	1	Beverages
76	Lakkalikööri	18,00	1	Beverages
3	Aniseed Syrup	10,00	2	Condiments
4	Chef Anton's Cajun Seasoning	22,00	2	Condiments

FULL OUTER JOIN: Indica que muestre todos los resultados de la columna de la derecha.

Sintaxis:

SELECT <lista de campos>
FROM tabla1 FULL OUTER JOIN tabla2 ON tabla1.campo1 <operador_de_comparación> tabla2.campo2






```
USE Northwind
SELECT P.ProductID, P.ProductName, P.UnitPrice
      ,P.CategoryID, C.CategoryName
FROM Products P FULL OUTER JOIN Categories C ON P.CategoryID  = C.CategoryID
```

ProductID	ProductName	UnitPrice	CategoryID	CategoryName
1	Chai	18,00	1	Beverages
2	Chang	19,00	1	Beverages
3	Aniseed Syrup	10,00	2	Condiments
4	Chef Anton's Cajun Seasoning	22,00	2	Condiments
5	Chef Anton's Gumbo Mix	21,35	2	Condiments
6	Grandma's Boysenberry Spread	25,00	2	Condiments
7	Uncle Bob's Organic Dried Pears	30,00	7	Produce
8	Northwoods Cranberry Sauce	40,00	2	Condiments
9	Mishi Kobe Niku	97,00	6	Meat/Poultry
10	Ikura	31,00	8	Seafood
11	Queso Cabrales	21,00	4	Dairy Products
12	Queso Manchego La Pastora	38,00	4	Dairy Products
13	Konbu	6,00	8	Seafood
14	Tofu	23,25	7	Produce

VISTAS / VIEW: Una vista es una consulta almacenada en la base de datos para reutilizarla cuando sea necesario como si de una tabla se tratara. Las vistas es otro caso de abstracción de tabla.

Sintaxis:

```
CREATE VIEW [dbo].[titleview]
AS
SELECT title, au_ord, au_lname, price, ytd_sales, pub_id
FROM authors, titles, titleauthor
WHERE authors.au_id = titleauthor.au_id   AND titles.title_id = titleauthor.title_id
GO
```

	title	au_ord	au_lname	price	ytd_sales	pub_id
1	The Busy Executive's Database Guide	1	Bennet	19,99	4095	1389
2	Fifty Years in Buckingham Palace Kitchens	1	Blotchet-Halls	11,95	15096	0877
3	But Is It User Friendly?	1	Carson	22,95	8780	1389
4	The Gourmet Microwave	1	DeFrance	2,99	22246	0877
5	Silicon Valley Gastronomic Treats	1	del Castillo	19,99	2032	0877
6	Secrets of Silicon Valley	1	Dull	20,00	4095	1389
7	The Busy Executive's Database Guide	2	Green	19,99	4095	1389
8	You Can Combat Computer Stress!	1	Green	2,99	18722	0736
9	Sushi, Anyone?	3	Gringlesby	14,99	4095	0877
10	Secrets of Silicon Valley	2	Hunter	20,00	4095	1389
11	Computer Phobic AND Non-Phobic Individuals: Beha...	1	Karsen	21,59	375	0877

VEAMOS EL SIGUIENTE REQUERIMIENTO:

/* Generar una vista en donde sólo se incluyan los libros, sus tipos, y el nombre de sus autores. */

IMPORTANTE: Asegúrese de trabajar con la base de datos Pubs.

</digitalmind>

	title_id	title	type	au_fname	au_lname
1	BU1032	The Busy Executive's Database Guide	business	Marjorie	Green
2	BU1032	The Busy Executive's Database Guide	business	Abraham	Bennet
3	BU1111	Cooking with Computers: Surreptitious Balance Sh...	business	Michael	O'Leary
4	BU1111	Cooking with Computers: Surreptitious Balance Sh...	business	Steams	MacFeather
5	BU2075	You Can Combat Computer Stress!	business	Marjorie	Green
6	BU7832	Straight Talk About Computers	business	Dean	Straight
7	MC2222	Silicon Valley Gastronomic Treats	mod_cook	Innes	del Castillo
8	MC3021	The Gourmet Microwave	mod_cook	Michel	DeFrance
9	MC3021	The Gourmet Microwave	mod_cook	Anne	Ringer
10	PC1035	But Is It User Friendly?	popular_comp	Cheryl	Carson
11	PC8888	Secrets of Silicon Valley	popular_comp	Ann	Dull

Procedimientos Almacenados (Store Procedure)

</digitalmind>

Un procedimiento almacenado de SQL Server es un grupo de una o más instrucciones Transact-SQL o una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework. Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos. Entre otras, pueden contener llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores, y el motivo de estos.

Ventajas de usar procedimientos almacenados

- **Tráfico de red reducido entre el cliente y el servidor**

Los comandos de un procedimiento se ejecutan en un único lote de código. Esto puede reducir significativamente el tráfico de red entre el servidor y el cliente porque únicamente se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin la encapsulación de código que proporciona un rocedimiento, cada una de las líneas de código tendría que enviarse a través de la red.

</digitalmind>

- **Mayor seguridad**

Varios usuarios y programas cliente pueden realizar operaciones en los objetos de base de datos a través de un procedimiento, aunque los usuarios y los programas no tengan permisos directos sobre esos objetos. El procedimiento controla qué procesos y actividades se llevan a cabo y protege los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.

- **Reutilización del código**

El código de cualquier operación de base de datos redundante resulta un candidato perfecto para la encapsulación de procedimientos. De este modo, se elimina la necesidad de escribir de nuevo el mismo código, se reducen las inconsistencias de código y se permite que cualquier usuario o aplicación que cuente con los permisos necesarios pueda acceder al código y ejecutarlo.

- **Mantenimiento más sencillo**

Cuando las aplicaciones cliente llaman a procedimientos y mantienen las operaciones de base de datos en la capa de datos, solo deben actualizarse los cambios de los procesos en la base de datos subyacente. El nivel de aplicación permanece independiente y no tiene que tener conocimiento sobre los cambios realizados en los diseños, las relaciones o los procesos de la base de datos.

Tipos de Procedimientos Almacenados

- **Definidos por el usuario**

Un procedimiento definido por el usuario se puede crear en una base de datos definida por el usuario o en todas las bases de datos del sistema excepto en la base de datos Resource . El procedimiento se puede desarrollar en Transact-SQL o como una referencia a un método de Common Runtime Language (CLR) de Microsoft .NETFramework.

- **Temporales**

Los procedimientos temporales son una forma de procedimientos definidos por el usuario. Los procedimientos temporales son iguales que los procedimientos permanentes salvo porque se almacenan en tempdb. Hay dos tipos de procedimientos temporales: locales y globales. Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Los procedimientos temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); solo son visibles en la conexión actual del usuario y se eliminan cuando se cierra la conexión. Los procedimientos temporales globales presentan dos signos de número (##) antes del nombre; son visibles para cualquier usuario después de su creación y se eliminan al final de la última sesión en la que se usa el procedimiento.

- **Sistema**

Los procedimientos del sistema se incluyen con SQL Server. Están almacenados físicamente en la base de datos interna y oculta Resource y se muestran de forma lógica en el esquema sys de cada base de datos definida por el sistema y por el usuario. Además, la base de datos msdb también contiene procedimientos almacenados del sistema en el esquema dbo que se usan para programar alertas y trabajos. Dado que los procedimientos del sistema empiezan con el prefijo sp_ , le recomendamos que no use este prefijo cuando asigne un nombre a losprocedimientos definidos por el usuario. Link documentación procedimientos extendidos.

Sintaxis de procedimiento almacenado

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

```
EXEC procedure_name;
```

Ejemplo de la base de datos Northwind

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Ej. crear un procedimiento almacenado denominado “TodosLosClientes" que selecciona todos los registros de la tabla "Customers":

```
CREATE PROCEDURE TodosLosClientes
AS
SELECT * FROM Customers
GO;
```

```
EXEC TodosLosClientes;
```

Procedimiento almacenado con un parámetro

Ej. crear un procedimiento almacenado que selecciona Clientes de una Ciudad en particular de la tabla "Clientes":

```
CREATE PROCEDURE TodosLosClientes @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

```
EXEC TodosLosClientes @City = 'London';
```

Procedimiento almacenado con múltiples parámetros

</digitalmind>

Ej. Crear un procedimiento almacenado que selecciona Clientes de una Ciudad particular con un Código Postal particular de la tabla "Clientes":

```
CREATE PROCEDURE TodosLosClientes @City nvarchar(30), @PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Como se ejecuta el procedimiento

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

Funciones SQL

Funciones de agregación:

Las funciones de agregado realizan un cálculo en un conjunto de valores y devuelven un valor único.

- **COUNT() devuelve el número de filas que coincide con un criterio específico.**
  Sintaxis
  SELECT COUNT (column_name) FROM table_name WHERE condition

- **AVG() devuelve el valor promedio de una columna numérica**
  Sintaxis
  SELECT AVG (column_name) FROM table_name WHERE condition

- **SUM() devuelve la suma total de una columna numérica**
  Sintaxis
  SELECT SUM (column_name) FROM table_name WHERE condition

- **MIN() devuelve el valor mas pequeño de la columna seleccionada**
  Sintaxis
  SELECT MIN (column_name) FROM table_name WHERE condition

- **MAX() devuelve el valor mas grande de la columna seleccionada**
  Sintaxis
  SELECT MAX (column_name) FROM table_name WHERE condition

Funciones definidas por el usuario:

Una función definida por el usuario es una rutina de Transact-SQL o Common Language Runtime (CLR) que acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como valor. El valor devuelto puede ser un valor escalar (único) o una tabla.

Función ESCALAR:

Las funciones escalares definidas por el usuario devuelven un único valor de datos del tipo definido en la cláusula RETURNS.
El tipo devuelto puede ser de cualquier tipo de datos excepto text, ntext
Sintaxis:

```
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
-- Returns the stock level for the product.
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
     IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
```

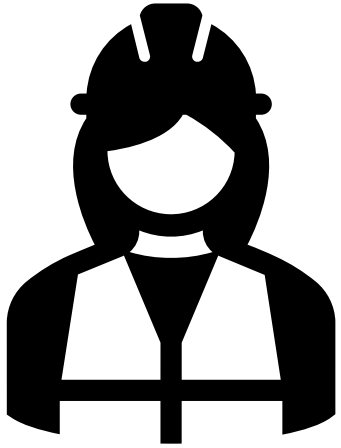
FUNCIÓN CON VALOR DE TABLA

Las funciones con valores de tabla definidas por el usuario devuelven un tipo de datos table.
La tabla es el conjunto de resultados de una sola instrucción SELECT.

Sintaxis

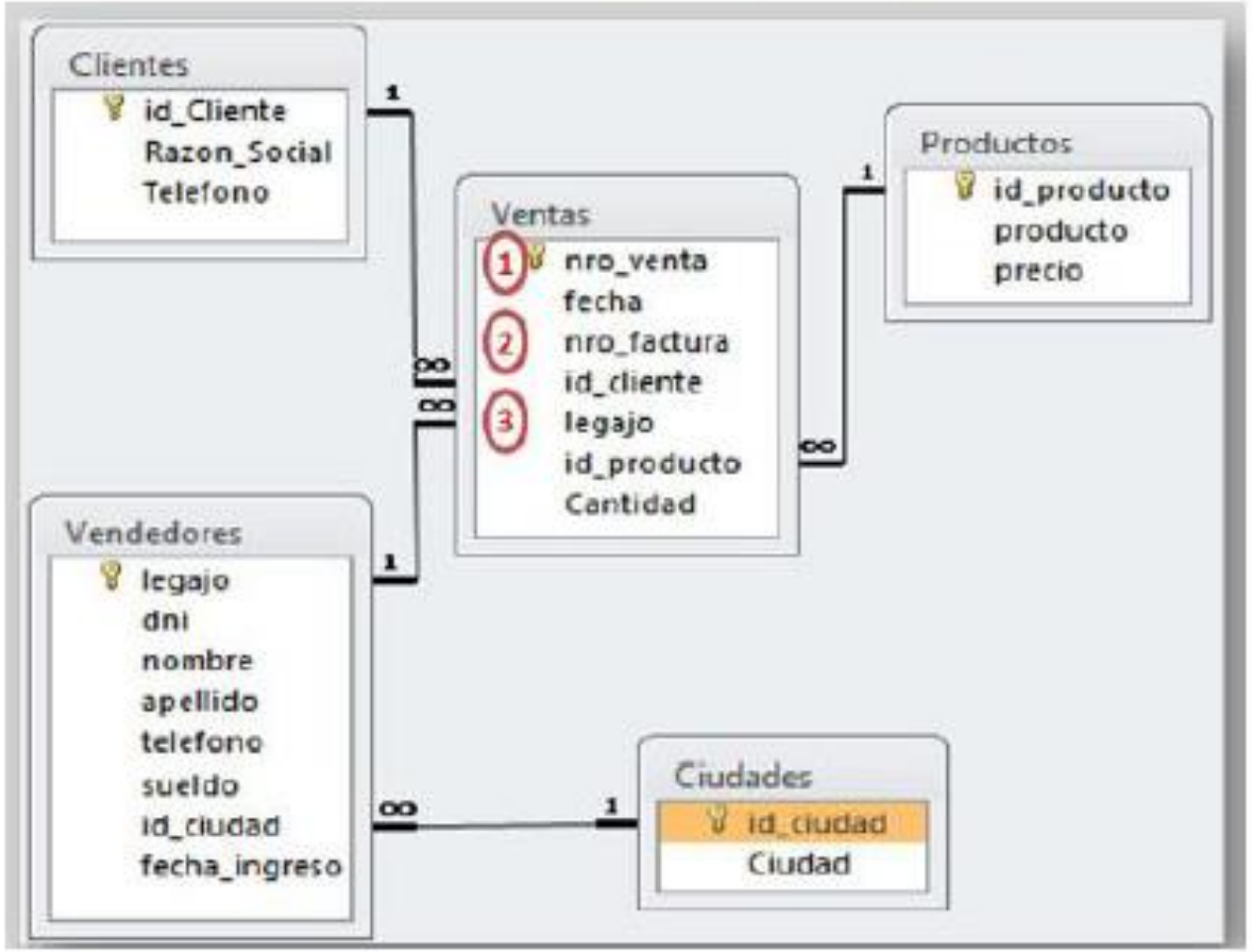
```
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
    SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
```

Ahora, manos a la obra….



Actividades:

Crear las siguientes tablas (entidades) con sus respectivos tipos de Datos



</digitalmind>

Teniendo en cuenta los siguientes tipos de datos.

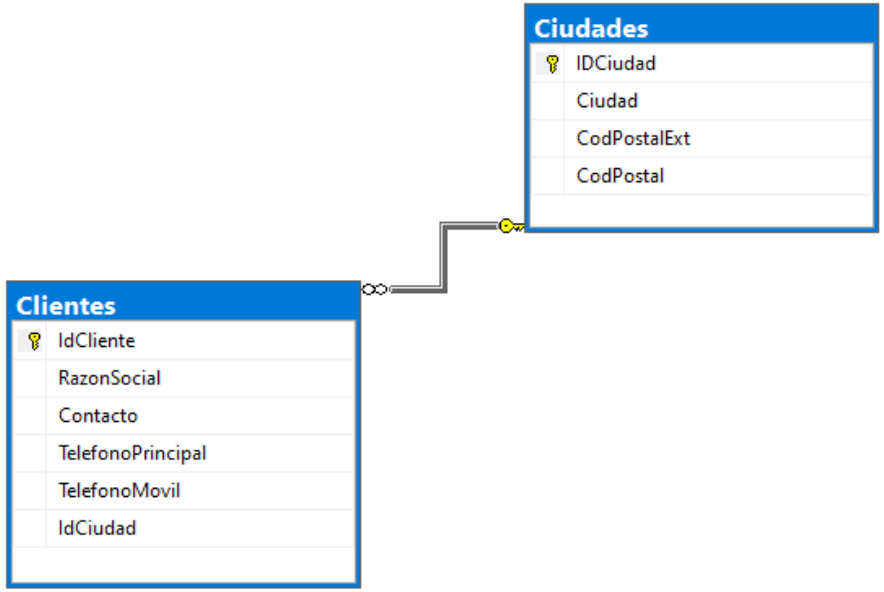
Tabla: Clientes
IdCliente -> Int
RazonSocial -> nvarchar(50)
Telefono -> Text

Tabla: Ciudad
IdCiudad -> Int
Ciudad -> nvarchar(50)

Tabla Vendedores
IdLegajo -> nchar(10)
DNI -> nchar(8)
Nombre -> nvarchar(50)
Apellido -> nvarchar(50)
Telefono -> Text
Sueldo -> decimal(18, 0)
IdCiudad -> Int
FechaIngreso -> Date

Tabla: Productos
idProducto -> int
Producto -> nvarchar(50)
Precio -> decimal(18,0)

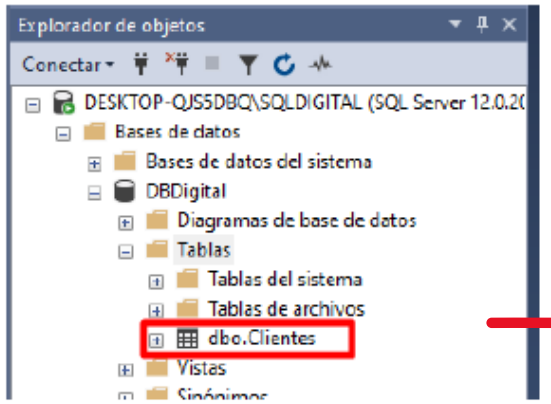
Tabla: Ventas
IdNroVenta -> int
Fecha -> date
NroFactura -> nchar(16)
idCliente -> int
IdLegajo -> nchar(10)
idProducto -> int
Cantidad -> int

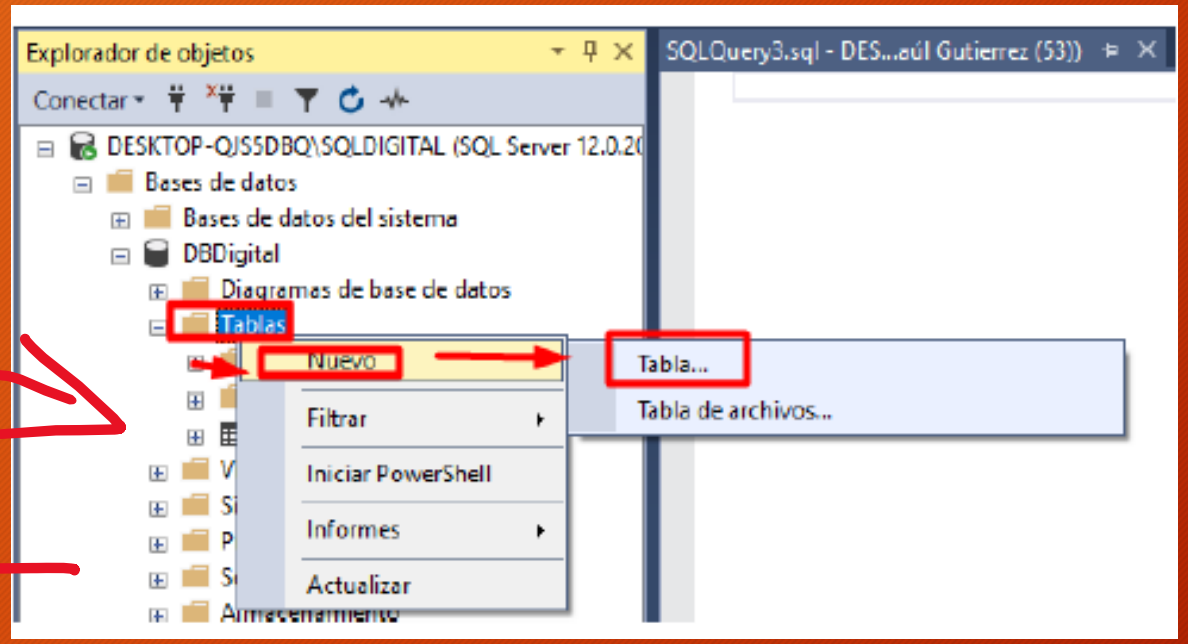


Site para diseñara Diagramas:
https://app.diagrams.net/

Creación Tabla Clientes :

- Creación Tabla (entidad) Clientes





- Creación de los Campos (fields) de la entidad, tipo de datos y generación de la clave principal