

# **BASE DE DATOS**



Semana 05

UPN.EDU.PE



# **Semana 05**

**Lenguaje de consulta estructurado, creación de base de datos, creación de tablas, relacionar tablas restricciones y elaboración de consultas.**



# Presentación de la sesión

## Logro de sesión

Al término de la sesión el estudiante comprende las características y estructura básica del lenguaje Transact-SQL, aplicando tales conocimientos en la herramienta SQL server 2014, con ejemplos prácticos.

## Temario

- Lenguaje de consulta estructurado SQL
- Creación de Base de Datos
- Creación de Tablas
- Relación entre tablas
- Inserción de datos
- Elaboración de consultas, ordenamiento y búsqueda de datos

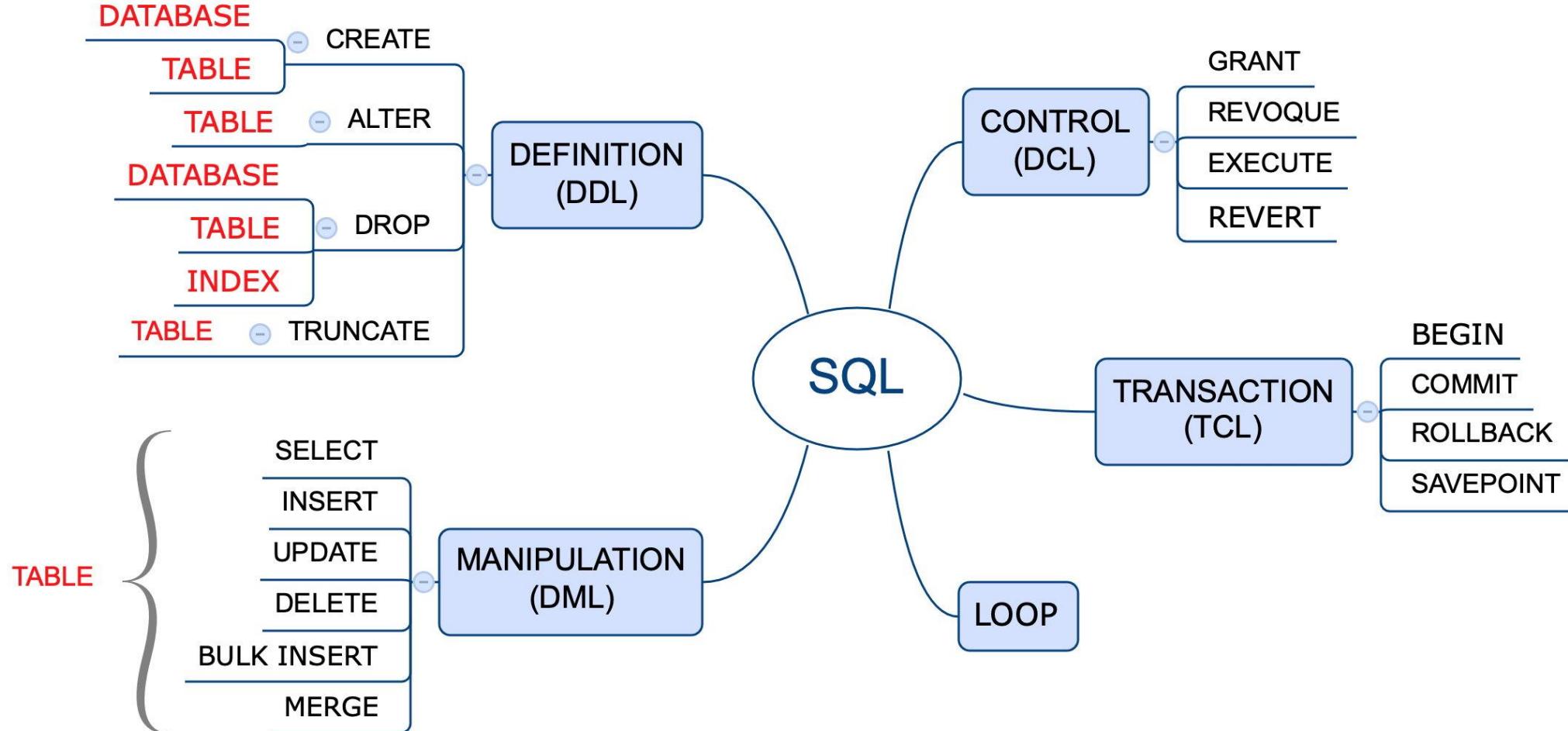


## Recogiendo saberes previos



¿Qué entiende por lenguaje estructurado de datos?

# Recogiendo saberes previos



1

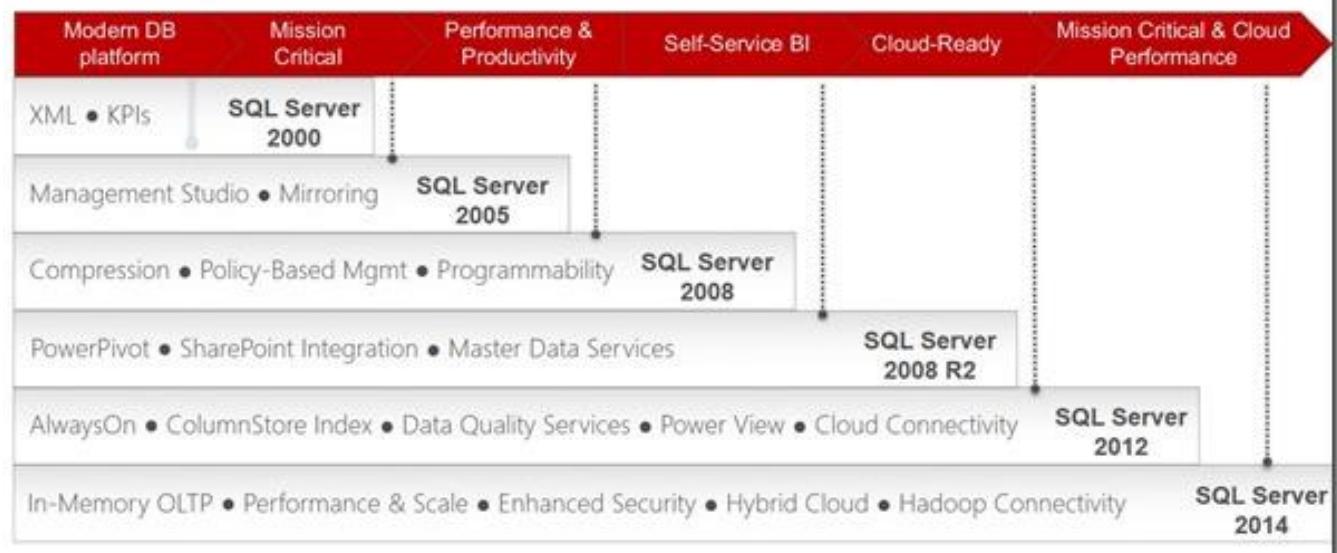
# **LENGUAJE ESTRUCTURADO DE CONSULTAS (SQL)**

# 11 HISTORIA DEL LENGUAJE ESTRUCTURADO

La historia de SQL empieza en 1974 con la definición por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL(Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975.

Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje(SEQUEL/2) que, a partir de ese momento, cambió de nombre por completo y se convirtió en SQL.

## The Evolution of SQL Server



# 1 LENGUAJE ESTRUCTURADO DE CONSULTAS (SQL)

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

```
SELECT TOP 1000 [IdCategoria]  
    ,[Categoria]  
    ,[Descripcion]  
FROM [MarketPERU].[dbo].[CATEGORIA]
```





# GENERALIDADES DE SQL



## QUE ES SQL?

SQL es un lenguaje de interacción con la base de datos que permite añadir, recuperar, editar y borrar la información almacenada en la base de datos. Esta diseñado para:



SIMPLE



FLEXIBLE

Las bases de datos son la base de un sinnúmero de sitios web y aplicaciones, y la mayoría de las aplicaciones centradas en datos utilizan bases de datos relacionales



Casi todas las bases de datos relacionales en el mundo son compatibles con SQL, lo que hace que sea una herramienta crucial para los programadores





# GENERALIDADES DE SQL

## ¿Como funciona?

Vamos a empezar con la forma en que las bases de datos están configuradas.

**Las bases de datos se componen de cientos o miles de tablas, y cada tabla se compone de filas y columnas.**

first_name	last_name	gender
John	Smith	M
Mark	Jones	M
Cindy	Remington	F
Joanne	Callahan	F

**Las bases de datos son básicamente un montón de información.**

**La forma en que se consulta los datos con SQL permite:**  
**Ordenar, organizar, y recuperar los datos de acuerdo a nuestras preferencias.**





# GENERALIDADES DE SQL

- Arquitectura Cliente / Servidor



- Transact SQL

```
DECLARE @SaleTax AS MONEY
DECLARE @VAT AS MONEY
SET @Amount = 1000
SET @SaleTax = @Amount * 0.12
SET @VAT = @Amount * 0.04
SELECT @Amount AS Amount, @SaleTax AS SaleTax, @VAT AS VAT
```

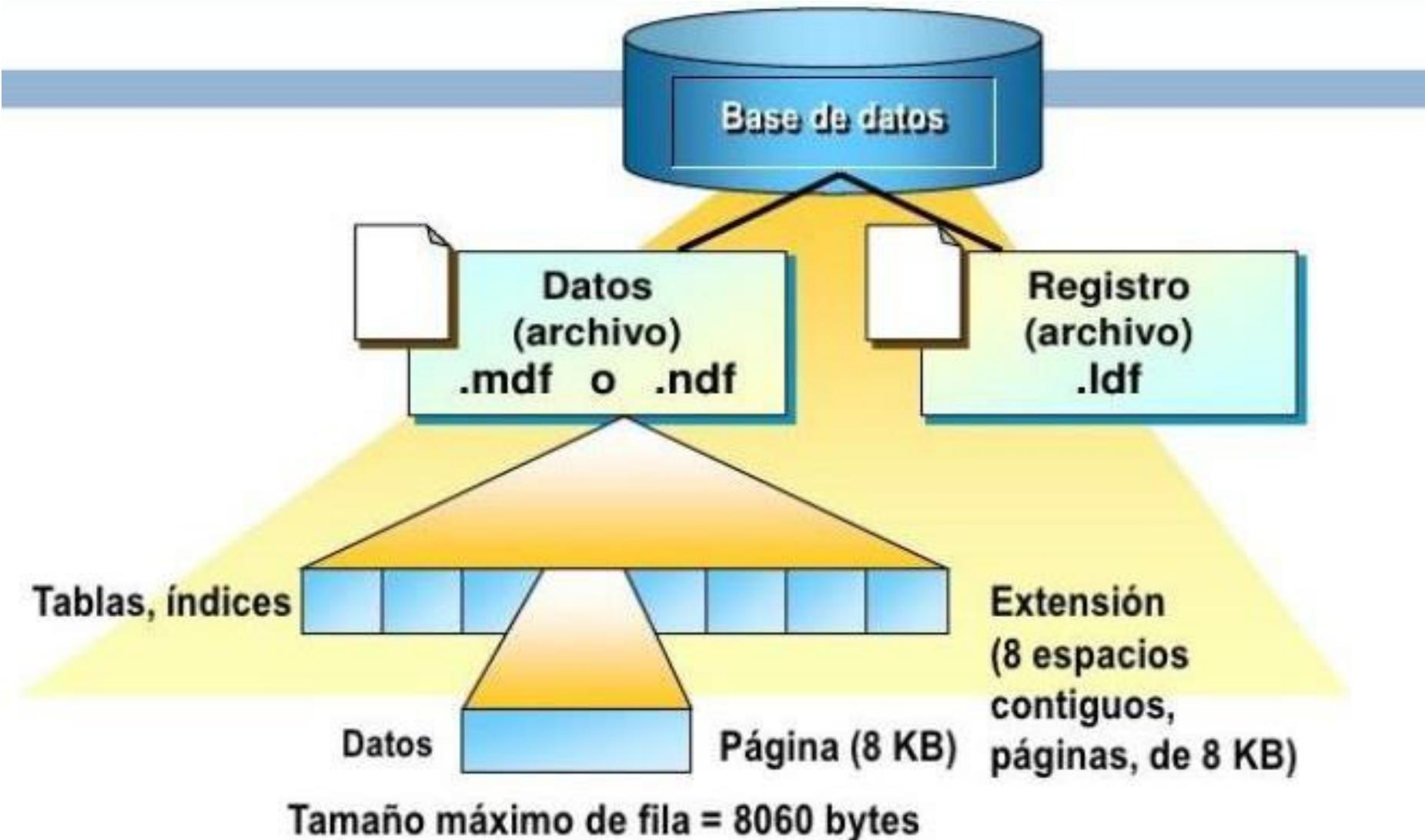


# VERSIONES SQL

**Historia de versiones**

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

# ESTRUCTURA DE UNA BASE DE DATOS

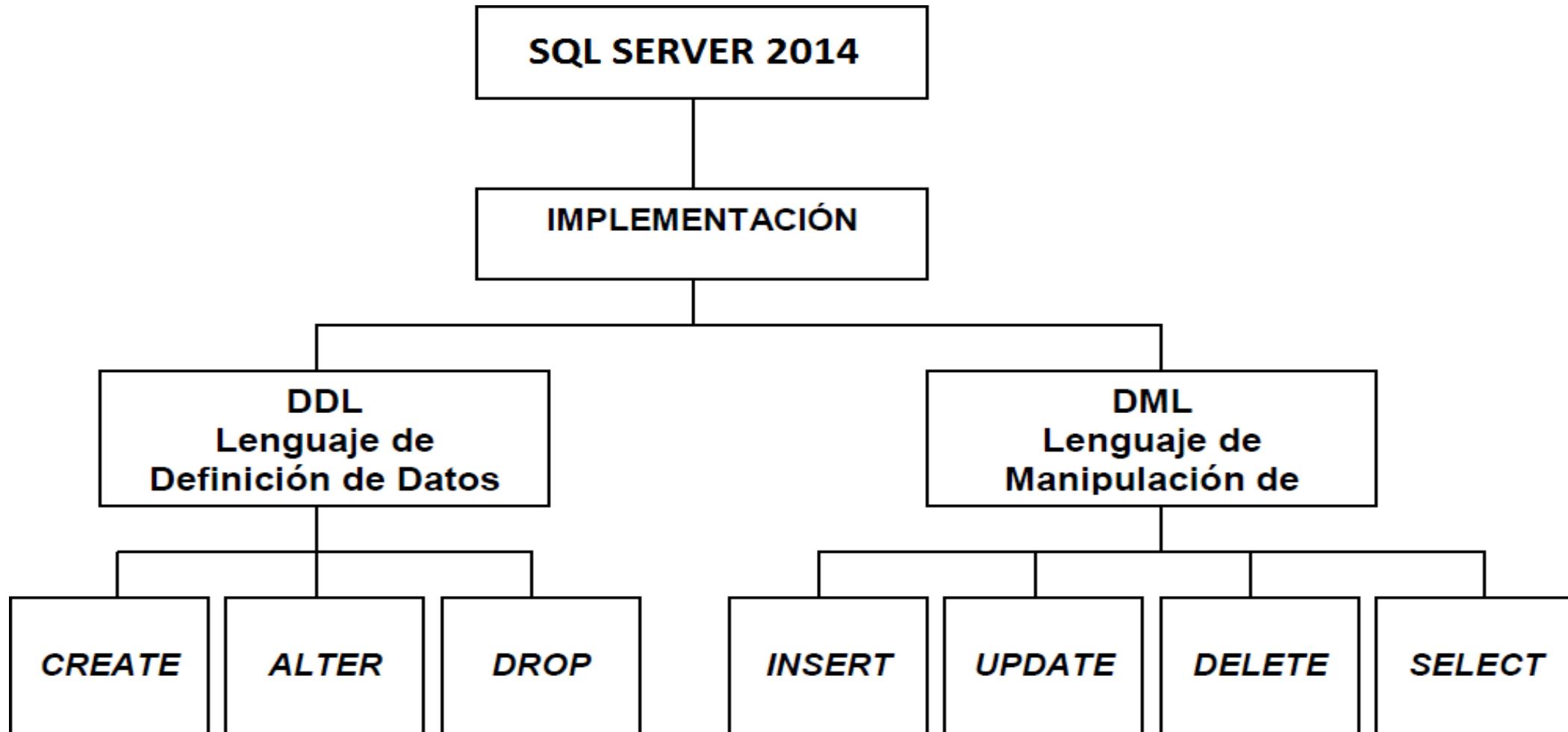


# 1 TIPOS DE ARCHIVOS DE UNA BASE DE DATOS:

<b>Archivo Principal</b>	Estos archivos contienen la información de inicio para la base de datos. Este archivo se utiliza también para almacenar datos. Cada base de datos tiene un único archivo principal. Tiene extensión .MDF.
<b>Archivo Secundario</b>	Estos archivos contienen todos los datos que no caben en el archivo de datos principal. No es necesario que las bases de datos tengan archivos de datos secundarios si el archivo principal es lo suficientemente grande como para contener todos los datos.
<b>Archivo de Transacciones</b>	Estos archivos contienen la información de registro que se utiliza para recuperar la base de datos. Debe haber al menos un archivo de registro de transacciones para cada base de datos, aunque puede haber más de uno. El tamaño mínimo para un archivo de registro es 512 kilobytes (KB). Tiene extensión .LDF.



# DIVISION DEL LENGUAJE DE CONSULTA ESTRUCTURADO (SQL)



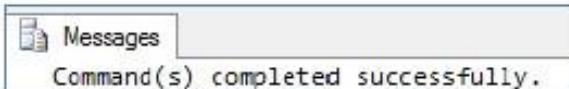


# **CREACION DE UNA BASE DE DATOS**

# 1 CREACIÓN DE UNA BASE DE DATOS: FORMA BÁSICA

**Caso 1:** Crear la base de datos BD\_COMERCIO de forma estándar.

```
CREATE DATABASE BD_COMERCIO  
GO
```



Verificar la existencia de la base de datos BD\_COMERCIO

```
SELECT * FROM SYS.sysdatabases WHERE NAME='BD_COMERCIO'
```

	name	dbid	sid	mode	status	status2	create	reserved	category	cmplvl	filename	version
1	BD_COMERCIO	45	0x010..	0	65536	1627385952	2015-08-13 10:04:24.853	1900-01-01 00:00:00.000	0	110	C:\Program Files\Microsoft SQL Server\M5SQL11 PC\MSQL\DATA\BD_COMERCIO.mdf	706

# 1 SENTENCIA BÁSICA PARA CREAR UNA BASE DE DATOS: FORMA ESTRUCTURADA

```
CREATE DATABASE [NOMBRE_BASE_DATOS]
ON(    NAME=NOMBRELOGICO_ARCHIVO,
      FILENAME='RUTA DEL ARCHIVO',
      SIZE=TAMAÑO_INICIAL,
      MAXSIZE=MÁXIMO_TAMAÑO,
      FILEGROWTH=TASA_DE_CRECIMIENTO)
GO
```

Donde:

- **NAME**: Define el nombre lógico del archivo.
- **FILENAME**: Define el nombre del archivo en disco.
- **SIZE**: Define el tamaño inicial de archivo (KB, MB, GB).
- **MAXSIZE**: Define el máximo tamaño que puede tener la BD (KB, MB, GB).
- **FILEGROWTH**: Define la tasa de ampliación del tamaño actual.



# CREACIÓN DE UNA BASE DE DATOS: FORMA ESTRUCTURADA

```
create database BD_ventas
on
primary (name=BD_ventas_data,
filename = 'C:\VENTAS_01\ventas.MDF',
SIZE=6,
MAXSIZE = 50,
FILEGROWTH = 15%)
LOG ON (name=BD_ventas_LOG,
filename = 'C:\VENTAS_01\ventas.LDF',
SIZE=3,
MAXSIZE = 10,
FILEGROWTH = 15%)
GO
```

# 1 CREACIÓN DE UNA BASE DE DATOS: FORMA ESTRUCTURADA

```
CREATE DATABASE VENTA2016  
ON  
    PRIMARY (NAME = VENTA2016_DATA,  
        FILENAME = 'C:\BDVENTAS\VENTA.MDF',  
        SIZE = 6,  
        MAXSIZE = 50,  
        FILEGROWTH = 15%)  
    LOG ON (NAME = VENTA2016_LOG,  
        FILENAME = 'C:\BDVENTAS\VENTA.LDF',  
        SIZE = 3,  
        MAXSIZE = 10,  
        FILEGROWTH = 15%)  
GO
```

# 1 CREACIÓN DE UNA BASE DE DATOS: FORMA ESTRUCTURADA

```
CREATE DATABASE DBPrueba2
ON PRIMARY(
    NAME = DBPrueba2_data,
    FILENAME = 'E:\SQLServer2014\Data\DBPrueba2.mdf',
    SIZE = 8MB,
    MAXSIZE = 15MB,
    FILEGROWTH = 1MB)
LOG ON(
    NAME = DBPrueba2_log,
    FILENAME = 'E:\SQLServer2014\Data\DBPrueba2_log.ldf',
    SIZE = 3MB,
    MAXSIZE = 8MB,
    FILEGROWTH = 10%)
go
```

1

# **CREAR, MODIFICAR Y ELIMINAR ESTRUCTURA DE DATOS**

# 1 SENTENCIAS DDL



## ¿Qué significa DDL?

**Significa:** Lenguaje de definición de datos

## ¿Para qué se utiliza las sentencias DDL?

Se utilizan para crear, modificar, eliminar la estructura de las tablas y las bases de datos, así como otros objetos de la base de datos.

## ¿Cuáles son las sentencias DDL?

CREATE

DROP

ALTER



# SENTENCIAS DDL PARA LA TABLA DE DATOS

## CREATE

Permite **crear** los objetos propios de la base de datos (Tablas, vistas, procedimientos almacenados, índices, Base de datos, etc.).

## DROP

Permite **eliminar** los objetos propios de la base de datos (Tablas, vistas, procedimientos almacenados, índices, Base de datos, etc.).

## ALTER

Permite **modificar** los objetos propios de la base de datos (Tablas, vistas, procedimientos almacenados, índices, Base de datos, etc.).





# SENTENCIA DDL: CREATE

## CONSIDERACIONES ANTES DE UTILIZAR



**PRIMERO:** Conocer la sintaxis

```
CREATE TABLE NOMBRE_TABLA  
(  
    CAMPO1 TIPO DE DATO  
)  
GO
```

**SEGUNDO:** Explicar las partes

- **NOMBRE\_TABLA:** Es el nombre de la tabla a crear, debemos considerar que dicho nombre cuenta con las mismas reglas que se aplican a las variables de un lenguaje de programación. **Por ejemplo: EMPLEADO, TB\_EMPLEADO,**
- **CAMPO1:** Es la especificación de las columnas que cuenta la tabla, así mismo son llamados columnas o atributos. Por ejemplo: **CÓDIGO, CODIGO\_EMP, COD\_EMP.**
- **TIPO:** Es la especificación del tipo de datos según el contenido del campo de una tabla; éstas podrían ser: **CHAR, VARCHAR, INT, MONEY, DATE, ETC.**

# SENTENCIA DDL: CREATE

## IMPLEMENTACIÓN DE LA TABLA: PRODUCTO

**Nota:** Para implementar la tabla **Producto**, debemos crear primero una base de datos. Para nuestro ejemplo será: **TIENDA2016**

1

```
--VALIDAR LA BASE DE DATOS  
IF DB_ID('TIENDA2016') IS NOT NULL  
    DROP DATABASE TIENDA2016  
GO  
--CREANDO LA BASE DE DATOS  
CREATE DATABASE TIENDA2016  
GO  
--POSICIONÁNDOTE EN LA BASE DE DATOS  
USE TIENDA2016  
GO
```

2

```
CREATE TABLE PRODUCTO  
(  
    ID_PRODUCTO     CHAR      (6) NOT NULL,  
    DESCRIPCION     VARCHAR   (45) NOT NULL,  
    PRECIO_VENTA    MONEY     NOT NULL,  
    STOCK_MINIMO    INT       NULL,  
    STOCK_ACTUAL    INT       NULL,  
    FECHA_VENC     DATE     NULL,  
    PRIMARY KEY     (ID_PRODUCTO)  
)  
GO
```

3

PRODUCTO	
ID_PRODUCTO	
DESCRIPCION	
PRECIO_VENTA	
STOCK_MINIMO	
STOCK_ACTUAL	
FECHA_VENC	

# SENTENCIA DDL: ALTER

## ¿CÓMO AGREGO UNO O MÁS CAMPOS A UNA TABLA?

**Nota:** Utilizaremos la sentencia **ALTER**

su sintaxis es:

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

### 1.) Agregar una columna a la tabla

```
ALTER TABLE NOMBRE_TABLA  
ADD COLUMNA NUEVA TIPO NULL | NOT NULL  
GO
```

**Ejemplo:** Agregar la columna STOCK MÁXIMO a la tabla **Producto**.

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

PRODUCTO	
ID_PRODUCTO	
DESCRIPCION	
PRECIO_VENTA	
STOCK_MINIMO	
STOCK_ACTUAL	
FECHA_VENC	
▶ STOCK_MAX	



# SENTENCIA DDL: ALTER

## ¿CÓMO AGREGO UNO O MÁS CAMPOS A UNA TABLA?

### 2.) Agregar varios campos a la tabla

```
ALTER TABLE NOMBRE_TABLA  
ADD COLUMNNUEVA1 TIPO      NULL | NOT NULL,  
                  COLUMNNUEVA2 TIPO    NULL | NOT NULL  
GO
```

**Ejemplo:** Agregar los campos unidad de medida, precio de costo y código de categoría a la tabla **Producto**

```
ALTER TABLE PRODUCTO  
ADD PRE_COSTO MONEY        NOT NULL,  
      UNI_MEDIDA VARCHAR(25) NOT NULL,  
      COD_CATE  CHAR(3)     NOT NULL  
GO
```

PRODUCTO	
ID_PRODUCTO	
DESCRIPCION	
PRECIO_VENTA	
STOCK_MINIMO	
STOCK_ACTUAL	
FECHA_VENC	
STOCK_MAX	
PRE_COSTO	
UNI_MEDIDA	
COD_CATE	



# SENTENCIA DDL: ALTER

¿CÓMO AGREGO UNA LLAVE PRIMARIA A UNA TABLA?

## 3.) Agregar llave primaria a la tabla

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

**Ejemplo:** Especificar la columna COD\_CATE como llave primaria de la tabla **CATEGORIAS**

1

CATEGORIAS	
COD_CATE	
NOMBRE	

2

```
ALTER TABLE CATEGORIAS  
ADD PRIMARY KEY (COD_CATE)  
GO
```

3

CATEGORIAS	
	COD_CATE
	NOMBRE

```
CREATE TABLE CATEGORIAS  
( COD_CATE CHAR(3) NOT NULL,  
NOMBRE VARCHAR(25) NOT NULL  
)  
GO
```

1

**PRIMARY KEY  
AND  
FOREING KEY**

# SENTENCIA DDL: ALTER

¿CÓMO AGREGO UNA LLAVE PRIMARIA A UNA TABLA?

## 3.) Agregar llave primaria a la tabla

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

**Ejemplo:** Especificar la columna COD\_CATE como llave primaria de la tabla **CATEGORIAS**

1

CATEGORIAS	
COD_CATE	
NOMBRE	

2

```
ALTER TABLE CATEGORIAS  
ADD PRIMARY KEY (COD_CATE)  
GO
```

3

CATEGORIAS	
	COD_CATE
	NOMBRE

```
CREATE TABLE CATEGORIAS  
( COD_CATE CHAR(3) NOT NULL,  
NOMBRE VARCHAR(25) NOT NULL  
)  
GO
```

# PRIMARY KEY



## ¿QUÉ ES?

Es un valor que identifica de manera única a cada **FILA**.

### DONDE:

- El valor puede ser de un campo (simple) o de la combinación de varios campos (compuesta)
- Los campos que identifican al **Primary Key** deben ser **Not Null**
- No permite valores nulos
- No permite valores duplicados



Primary Key

Restricción



# SENTENCIA DDL: ALTER

## ¿CÓMO AGREGO UNA LLAVE COMPUUESTA A UNA TABLA?

### 4.) Agregar llave compuesta a la tabla

```
ALTER TABLE NOMBRE_TABLA  
ADD PRIMARY KEY (COLUMNNA1,COLUMNNA2)  
GO
```

**Ejemplo:** Especificar la columna NUM\_BOLETA y ID\_PRODUCTO como llave compuesta de la tabla DETALLE\_BOLETA.

1

DETALLE_BOLETA	
NUM_BOLETA	
ID_PRODUCTO	
CANTIDAD	
IMPORTE	

```
CREATE TABLE DETALLE_BOLETA  
(  
    NUM_BOLETA CHAR (8) NOT NULL,  
    ID_PRODUCTO CHAR (6) NOT NULL,  
    CANTIDAD INT NOT NULL,  
    IMPORTE MONEY NOT NULL  
)  
GO
```

2

```
ALTER TABLE DETALLE_BOLETA  
ADD PRIMARY KEY  
(NUM_BOLETA,ID_PRODUCTO)  
GO
```

3

DETALLE_BOLETA	
NUM_BOLETA	KEY
ID_PRODUCTO	KEY
CANTIDAD	
IMPORTE	



# ↑ FOREIGN KEY



## CONSTRAINT: Foreign Key

### Foreign Key:

- Permite definir una clave externa, que es una columna o combinación de columnas, que se utiliza para establecer y exigir un vínculo entre los datos de dos tablas.
- Evidencia una relación física entre dos entidades
- En una relación “**Uno a Muchos**” la llave del lado “Uno” es referenciada desde el lado “Muchos”



# SENTENCIA DDL: ALTER

¿CÓMO AGREGO UNA LLAVE FORÁNEA A UNA TABLA?

## 5.) Agregar llave foránea a la tabla

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

Ejemplo: Vamos a relacionar la tabla **Producto** con la tabla **Categorías**.

1

CATEGORIAS	
PK	COD_CATE
	NOMBRE

2

```
ALTER TABLE PRODUCTO  
ADD FOREIGN KEY (COD_CATE) REFERENCES CATEGORIAS  
GO
```

3



PRODUCTO	
PK	ID_PRODUCTO
	DESCRIPCION
	PRECIO_VENTA
	STOCK_MINIMO
	STOCK_ACTUAL
	FECHA_VENC
	STOCK_MAX
	PRE_COSTO
	UNI_MEDIDA
FK	COD_CATE



# SENTENCIA DDL: ALTER

¿CÓMO AGREGO UNA LLAVE FORÁNEA A UNA TABLA?

Ejemplo 02: Vamos a relacionar la tabla **Producto** con la tabla **DETALLE\_BOLETA**.

1

PRODUCTO	
PK	ID_PRODUCTO
	DESCRIPCION
	PRECIO_VENTA
	STOCK_MINIMO
	STOCK_ACTUAL
	FECHA_VENC
	STOCK_MAX
	PRE_COSTO
	UNI_MEDIDA
	COD_CATE

2

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

DETALLE_BOLETA	
PK	NUM_BOLETA
	ID_PRODUCTO
	CANTIDAD
	IMPORTE

CLAVE  
COMPUESTA

3

PRODUCTO *	
PK	ID_PRODUCTO
	DESCRIPCION
	PRECIO_VENTA
	STOCK_MINI...
	STOCK_ACTUAL
	FECHA_VENC
	STOCK_MAX
	PRE_COSTO
	UNI_MEDIDA
	COD_CATE

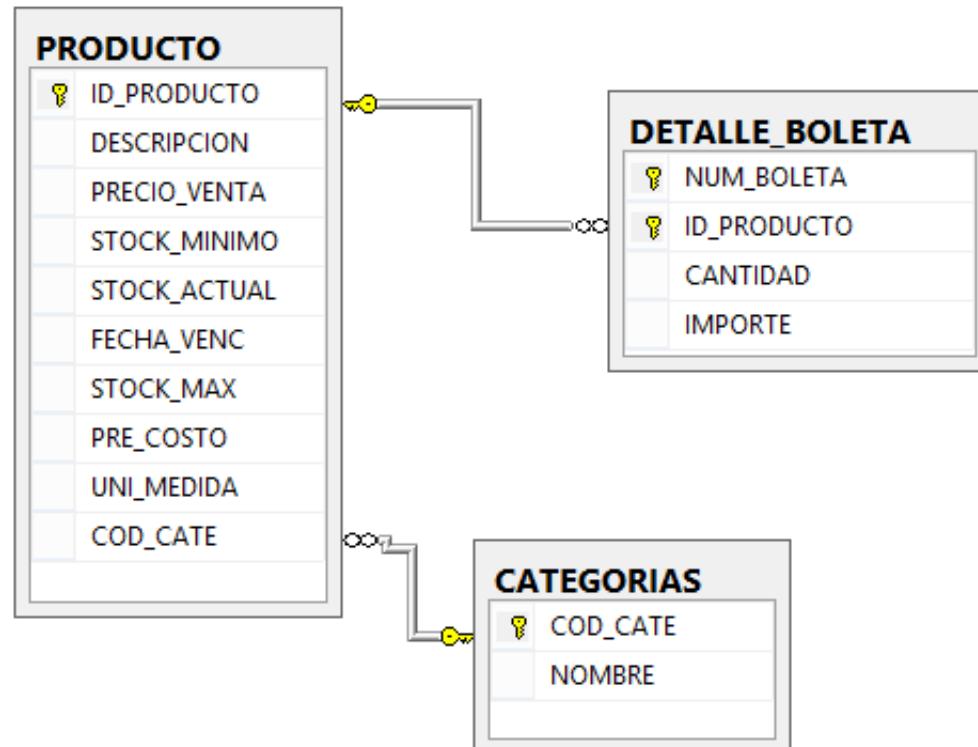
DETALLE_BOLETA	
PK	NUM_BOLETA
	ID_PRODUCTO
	CANTIDAD
	IMPORTE



# TABLAS RELACIONADAS

¿CÓMO QUEDARÍA LAS TABLAS RELACIONADAS?

**Importante:** La relación de las tablas: **Productos**, **Categorias** y **Detalle\_Boleta** quedaría así:





# MODIFICAR TIPO DE DATOS

¿CÓMO MODIFICO EL TIPO DE DATO DE UN CAMPO?

## 7.) Modificar tipo de datos de una campo de la tabla

```
ALTER TABLE NOMBRE_TABLA  
ALTER COLUMN COLUMNNA TIPO  
GO
```

**Ejemplo:** Modificar el tipo de datos de la columna **PRE\_COSTO** de la tabla **Producto** por **decimal(7,2)**.

1

PRODUCTO	
Nombre de columna	Tipo de datos
ID_PRODUCTO	char(6)
DESCRIPCION	varchar(45)
PRECIO_VENTA	money
STOCK_MINIMO	int
STOCK_ACTUAL	int
FECHA_VENC	date
STOCK_MAX	int
PRE_COSTO	money
COD_CATE	char(3)

2

```
ALTER TABLE PRODUCTO  
ALTER COLUMN PRE_COSTO DECIMAL(7,2)  
GO
```

3

PRODUCTO	
Nombre de columna	Tipo de datos
ID_PRODUCTO	char(6)
DESCRIPCION	varchar(45)
PRECIO_VENTA	money
STOCK_MINIMO	int
STOCK_ACTUAL	int
FECHA_VENC	date
STOCK_MAX	int
PRE_COSTO	decimal(7, 2)
COD_CATE	char(3)



# ELIMINAR ATRIBUTOS

¿CÓMO ELIMINAR UNA COLUMNA DE UNA TABLA?

## 6.) Eliminar una columna de una tabla

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

**Ejemplo:** Eliminar la columna **UNI\_MEDIDA** de la tabla **Producto**.

1

PRODUCTO	
ID_PRODUCTO	
DESCRIPCION	
PRECIO_VENTA	
STOCK_MINIMO	
STOCK_ACTUAL	
FECHA_VENC	
STOCK_MAX	
PRE_COSTO	
UNI_MEDIDA	

2

```
ALTER TABLE PRODUCTO  
DROP COLUMN UNI_MEDIDA  
GO
```

3

PRODUCTO	
ID_PRODUCTO	
DESCRIPCION	
PRECIO_VENTA	
STOCK_MINIMO	
STOCK_ACTUAL	
FECHA_VENC	
STOCK_MAX	
PRE_COSTO	
COD_CATE	



# OTRA FORMA DE AGREGAR KEY's

## IMPORTANTE



También es posible asignar una clave principal; así como relación al momento de crear las tablas.

```
CREATE TABLE PRODUCTO
(
    ID_PRODUCTO     CHAR(6) PRIMARY KEY      NOT NULL,
    DESCRIPCION     VARCHAR  (45)           NOT NULL,
    PRECIO_VENTA    MONEY                 NOT NULL,
    STOCK_MINIMO    INT                   NULL,
    STOCK_ACTUAL    INT                   NULL,
    FECHA_VENC     DATE                 NULL,
    COD_CATE        CHAR (3)             NOT NULL REFERENCES CATEGORIAS
)
GO
```



**SETENCIAS DML**

# OTRA FORMA DE AGREGAR KEY's

## IMPORTANTE



También es posible asignar una clave principal; así como relación al momento de crear las tablas.

```
CREATE TABLE PRODUCTO
(
    ID_PRODUCTO      CHAR(6) PRIMARY KEY      NOT NULL,
    DESCRIPCION      VARCHAR  (45)            NOT NULL,
    PRECIO_VENTA     MONEY                  NOT NULL,
    STOCK_MINIMO     INT                   NULL,
    STOCK_ACTUAL     INT                   NULL,
    FECHA_VENC      DATE                  NULL,
    COD_CATE         CHAR (3)             NOT NULL REFERENCES CATEGORIAS
)
GO
```

# DML



## ¿Qué significa DML?

**Significa:** Lenguaje de manipulación de datos.

## ¿Para qué se utiliza las sentencias DML?

Se utilizan para consultar, ingresar, modificar y eliminar la información que existe dentro de las tablas en la base de datos.

## ¿Cuáles son las sentencias DML?

SELECT

DELETE

UPDATE

INSERT

# 11 INSERT INTO

Para almacenar datos en una base de datos pone a nuestra disposición la sentencia **INSERT**.

Inserción de filas; consiste en añadir a una tabla una o más filas y en cada fila todos o parte de sus campos. Podemos distinguir dos formas de insertar filas:

- ✓ Inserción individual de filas.
- ✓ Inserción múltiple de filas.

## Inserción Individual de una FILA

Para realizar la inserción individual de filas SQL posee la instrucción **INSERT INTO**.

Su sintaxis es la siguiente:

```
INSERT INTO Nombre_tabla[ (nombre_columna1,  
    nombre_columna2, nombre_columna n...)]  
VALUES (expr1, expr2, expr n...);
```

**Nombre\_Tabla:** Tabla donde se desea ingresar los nuevos datos.

**Nombre\_Columna:** Es una lista opcional de nombres de campos de la tabla.

Debe considerar el orden de las columnas.

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

# ↑ INSERT INTO

Indique la tabla a la cual insertará los datos

Nombre las columnas de la tabla a las cuales insertará datos

1.

INSERT INTO <table> (column1, column2,...)

VALUES (value1, value2,...)

Aquí deberá añadir los valores que insertará a la tabla, cada valor debe corresponder a una columna mencionada

## Ejemplo del comando INSERT INTO:

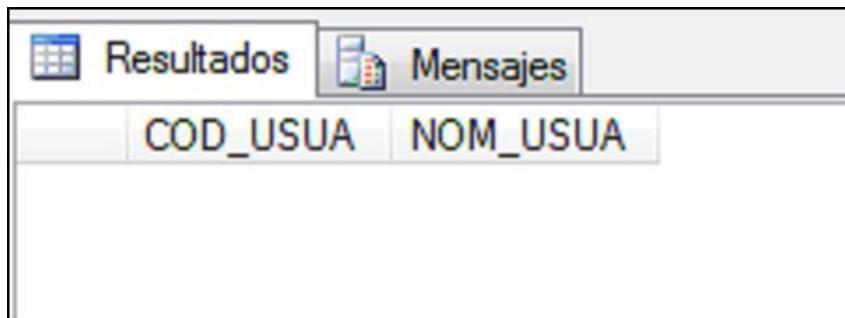
```
INSERT INTO employees (birth_date, first_name, last_name, gender, hire_date)
```

```
VALUES ('15-MAR-1995','John','Smith','M','01-FEB-2015');
```

★ Lo que esto significa: Añade el registro a la tabla "Empleados" con los datos fecha nacimiento 15-Mar-1995, nombre John Smith, genero M y fecha de contratación 1-FEB-2015.

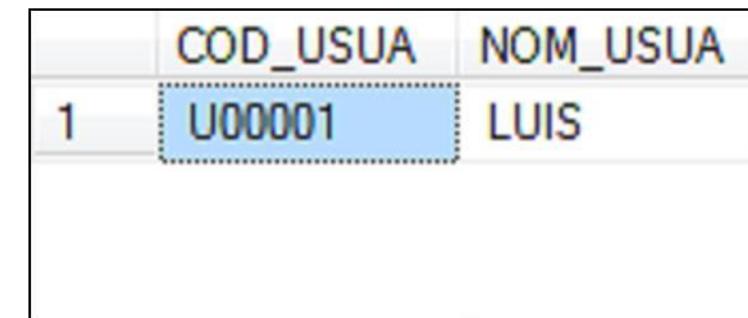
# ↑ INSERT INTO

```
--inserción de un registro a una tabla de la base de datos  
INSERT INTO USUARIO(COD_USUA,NOM_USUA)  
    VALUES ('U00001','LUIS')
```



COD_USUA	NOM_USUA

ANTES



COD_USUA	NOM_USUA
1	U00001 LUIS

DESPUÉS

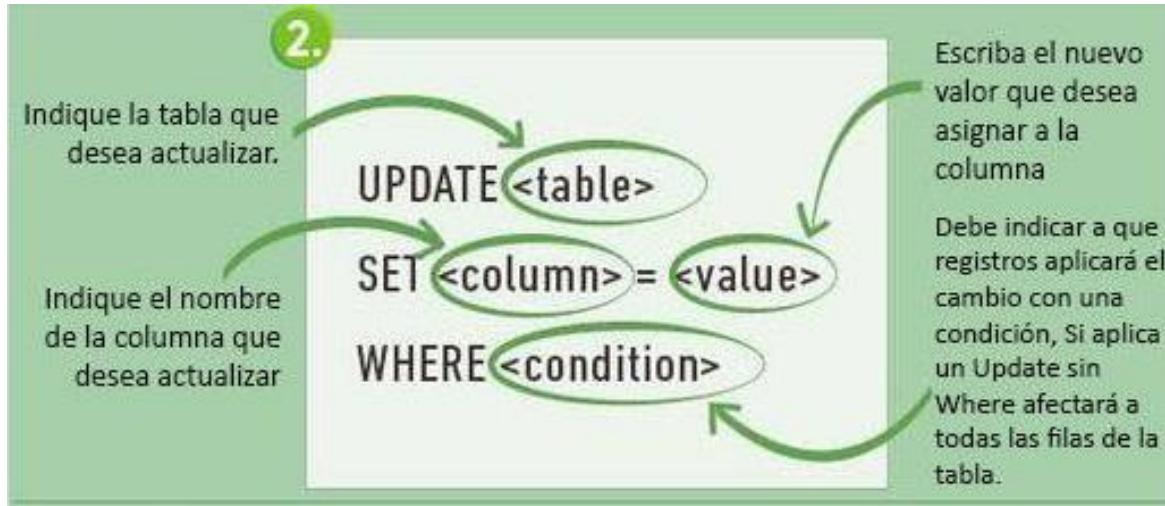
# UPDATE

- Para la actualización de datos SQL dispone de la sentencia UPDATE.
- La sentencia UPDATE permite la actualización de uno o varios registros de una única tabla.
- La sintaxis del UPDATE es la siguiente:

```
UPDATE Nombre_tabla  
      SET nombre_columna1 = expr1,  
          nombre_columna2 = expr2, ....  
      [WHERE { condición }]
```

- **Nombre\_Tabla:** Tabla donde se actualiza los datos.
- **Nombre\_columna:** Es el nombre de columna o campo cuyo valor se desea cambiar.
- **Expr:** El nuevo valor que se desea asignar al campo que le precede.

# 1 UPDATE



## Ejemplo del Comando UPDATE:

```
UPDATE employees  
SET monthly_gross = salary + bonus  
WHERE dept_id = 10
```



Lo que esto significa:  
Actualiza el campo monthly\_gross con la sumatoria de salario mas bono de los registros de la tabla Empleados donde dept\_id sea igual a 10

# UPDATE

- --Actualiza un campo de todos los registros
- **UPDATE DEPARTAMENTOS**
- **SET PRECIO\_ALQxMES\_DEP = 2000**

	COD_DEP	PRECIO_ALQXMES_DEP
1	DPT001	350
2	DPT002	240
3	DPT003	280
4	DPT004	250

**TABLA DEPARTAMENTOS  
ANTES**

	COD_DEP	PRECIO_ALQXMES_DEP
1	DPT001	2000
2	DPT002	2000
3	DPT003	2000
4	DPT004	2000

**TABLA DEPARTAMENTOS  
DESPUÉS**

# UPDATE

- --Actualiza varios campos de todos los registros
- **UPDATE DEPARTAMENTOS**
- **SET PRECIO\_ALQXMES\_DEP = 2500,**
- **NUM\_AMB\_DEP = 6**

COD_DEP	PRECIO_ALQXMES_DEP	NUM_AMB_DEP	
DPT001	2000	6	⬅
DPT002	2000	5	➡
DPT003	2000	5	➡
DPT004	2000	4	➡

**TABLA DEPARTAMENTOS  
ANTES**

COD_DEP	PRECIO_ALQXMES_DEP	NUM_AMB_DEP
1 DPT001	2500	6
2 DPT002	2500	6
3 DPT003	2500	6
4 DPT004	2500	6

**TABLA DEPARTAMENTOS  
DESPUÉS**

# UPDATE

- --Actualiza una o varias columnas utilizando la sentencia where.
- **UPDATE DEPARTAMENTOS**
- **SET PRECIO\_ALQxMES\_DEP = PRECIO\_ALQxMES\_DEP\*1.1**
- **WHERE AREA\_TOTAL\_DEP > 100**

	COD_DEP	PRECIO_ALQXMES_DEP	AREA_TOTAL_DEP	
1	DPT001	2500	250	➡
2	DPT002	2000	180	➡
3	DPT003	2000	90	➡
4	DPT004	2000	190	➡

**TABLA DEPARTAMENTOS  
ANTES**

	COD_DEP	PRECIO_ALQXMES_DEP	AREA_TOTAL_DEP	
	DPT001	2750	250	
	DPT002	2200	180	
	DPT003	2000	90	
	DPT004	2200	190	

**TABLA DEPARTAMENTOS  
DESPUES**

# DELETE

- Para borrar datos de una tabla, debemos utilizar la sentencia DELETE.
- Su sintaxis es la siguiente:

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

- **Nombre\_Tabla** nombre de la tabla donde se desea borrar los datos.
- La cláusula **WHERE** sigue el mismo formato que la vista en la sentencia
- **SELECT** y determina qué registros se borrarán.

# DELETE

- Permite eliminar uno o más registros de una tabla

--Eliminar todos los registros de la tabla

**DELETE FROM USUARIOS**

	COD_USUA	NOM_USUA	COD_EST
1	U00001	LUIS	NULL
2	U00002	OSCAR	ACTIVO
3	U00003	MARIELLA	ACTIVO
4	U00004	JUAN	INACTIVO
5	U00005	SUSAN	ACTIVO

**TABLA USUARIO ANTES**

	COD_USUA	NOM_USUA

**TABLA USUARIO DESPUÉS**

# DELETE

--Eliminar un registro específico

```
DELETE FROM USUARIO WHERE COD_USUA= 'U00001'
```

	COD_USUA	NOM_USUA	COD_EST
1	U00001	LUIS	NULL
2	U00002	OSCAR	ACTIVO
3	U00003	MARIELLA	ACTIVO
4	U00004	JUAN	INACTIVO
5	U00005	SUSAN	ACTIVO

TABLA USUARIO ANTES

	COD_USUA	NOM_USUA	COD_EST
1	U00002	OSCAR	ACTIVO
2	U00003	MARIELLA	ACTIVO
3	U00004	JUAN	INACTIVO
4	U00005	SUSAN	ACTIVO

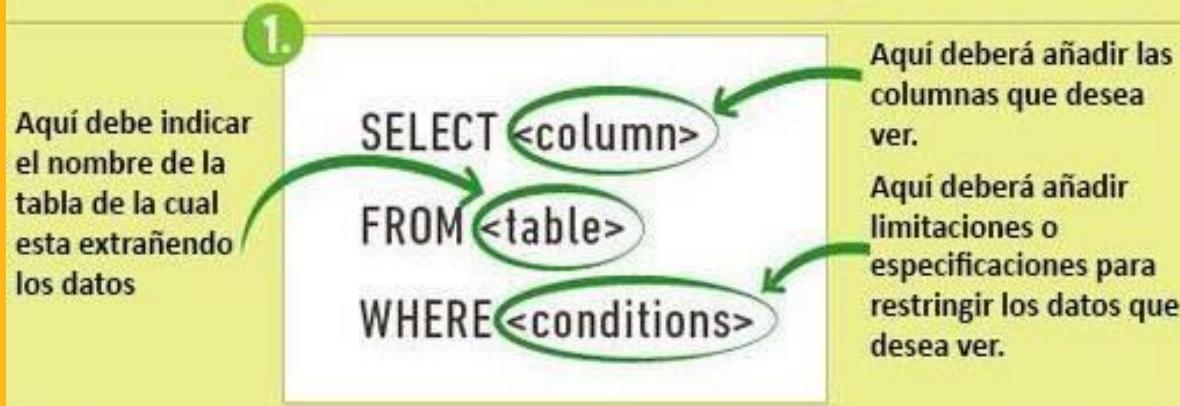
TABLA USUARIO DESPUÉS

# 1 SELECT

La lectura consisten en el comando **SELECT**, que tiene tres cláusulas comunes:

- **SELECT**
- **FROM**
- **WHERE**

Instrucciones que trabajan juntas



## Ejemplo del comando SELECT

```
SELECT first_name, last_name  
FROM employees  
WHERE first_name = 'John'
```

★ Lo que esto significa:  
Recupera nombre y apellido de la tabla llamada "Empleados" de cada persona con nombre igual a John



# **Semana 05**

**Operadores aritméticos, relacionales y lógicos. Consultas con operadores aritméticos, funciones de columna, funciones para el manejo de fechas.**



# Presentación de la sesión

## Logro de sesión

Al término de la sesión el estudiante comprende la estructura básica del lenguaje de manipulación de datos, aplicando tales conocimientos en la herramienta SQL server 2014, con ejemplos prácticos.

## Temario

- Uso del lenguaje de manipulación de datos (DML)
- Instrucción select.
- Funciones en base de datos.
- Resolución de ejercicios propuestos.



¿Qué entiende por  
lenguaje de manipulación de datos ?



# **LENGUAJE DE MANIPULACIÓN DE DATOS**

# INSTRUCCIÓN SELECT

La instrucción SELECT indica al motor de base de datos que devuelva información de la base de datos como un conjunto de registros.

Sintaxis:

```
SELECT<campo1>,<campo2>,<campo3>,..(*)  
FROM <nombre_tabla>
```

Cuando es un (\*) se mostrarán todos los datos de la tabla.

EJEMPLO: Se desea consultar todos los datos de la tabla:

```
SELECT *  
FROM LIBRO
```

# INSTRUCCIÓN SELECT ... FROM

```
SELECT *  
FROM LIBRO
```

isbn	titulo	apAutor	nomEdit	año
8420464988	Momo	Ende	Alfaguara	1982
8408049003	El retrato de Dorian Gray	Wilde	Planeta	2003
8477205302	El alquimista	Coelho	Obelisco	1996
8420432261	La historia interminable	Ende	Alfaguara	1998
8420616524	El fantasma de Canterville	Wilde	Alianza	1996
8408048783	Once minutos	Coelho	Planeta	2003

# 1 INSTRUCCIÓN SELECT ... FROM

LIBRO

isbn	titulo	apAutor	nomEdit	año
8420464988	Momo	Ende	Alfaguara	1982
8408049003	El retrato de Dorian Gray	Wilde	Planeta	2003
8477205302	El alquimista	Coelho	Obelisco	1996
8420432261	La historia interminable	Ende	Alfaguara	1998
8420616524	El fantasma de Canterville	Wilde	Alianza	1996
8408048783	Once minutos	Coelho	Planeta	2003

Se desea realizar una consulta a esta tabla para visualizar solo el campo **isbn** y el campo **título**:

SOLUCIÓN:

```
SELECT ISBN, TITULO  
FROM LIBRO
```

# 1 INSTRUCCIÓN SELECT ... FROM

```
SELECT ISBN, TITULO  
FROM LIBRO
```

isbn	titulo
8420464988	Momo
8408049003	El retrato de Dorian Gray
8477205302	El alquimista
8420432261	La historia interminable
8420616524	El fantasma de Canterville
8408048783	Once minutos

# 1 INSTRUCCIÓN SELECT ... FROM

LIBRO

isbn	titulo	apAutor	nomEdit	año
8420464988	Momo	Ende	Alfaguara	1982
8408049003	El retrato de Dorian Gray	Wilde	Planeta	2003
8477205302	El alquimista	Coelho	Obelisco	1996
8420432261	La historia interminable	Ende	Alfaguara	1998
8420616524	El fantasma de Canterville	Wilde	Alianza	1996
8408048783	Once minutos	Coelho	Planeta	2003

Se desea realizar una consulta a esta tabla para visualizar solo el campo **isbn** , campo **título** y el campo **año**:

SOLUCIÓN:

```
SELECT ISBN, TITULO, AÑO  
FROM LIBRO
```



TABLA: AULA\_A

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA

TABLA: AULA\_B

CODIGO	NOMBRE
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

```
SELECT CODIGO, NOMBRE  
FROM AULA_A  
UNION  
SELECT CODIGO, NOMBRE  
FROM AULA_B
```

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
005	PAUL
007	LUISA

# UNION ALL

TABLA: AULA\_A

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA

```
SELECT CODIGO,NOMBRE  
FROM AULA_A  
UNION ALL  
SELECT CODIGO,NOMBRE  
FROM AULA_B
```

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

TABLA: AULA\_B

CODIGO	NOMBRE
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

# 1 FUNCIONES BASICAS SQL

- **COUNT:** devuelve el número total de filas seleccionadas por la consulta.
- **MIN:** devuelve el valor mínimo del campo que especifiquemos.
- **MAX:** devuelve el valor máximo del campo que especifiquemos.
- **SUM:** suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **AVG:** devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

# 1 FUNCIONES BASICAS SQL

CODIGO	NOMBRE
2	ALVA
3	CASTRO
4	RAMIREZ
57	RUIZ
45	SANCHEZ

```
SELECT MAX(CODIGO)  
FROM AULA_C
```

```
SELECT MIN(CODIGO)  
FROM AULA_C
```

```
SELECT AVG(CODIGO)  
FROM AULA_C
```

```
SELECT SUM(CODIGO)  
FROM AULA_C
```

```
SELECT COUNT(CODIGO)  
FROM AULA_C
```



# AGRUPACION Y ORDENAMIENTO

La cláusula **GROUP BY** se utiliza para agrupar los campos que se encuentran en la cláusula SELECT del query en curso.

Sintaxis:

**GROUP BY** campo1, campo2, campo3, ..., campoN

La cláusula **ORDER BY** se utiliza para ordenar los campos que se encuentran en la cláusula SELECT por cualquier criterio.

Sintaxis:

**ORDER BY** campo1, campo2, campo3, ..., campoN

# ↑ GROUP BY

TABLA:

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

```
SELECT CODIGO, NOMBRE  
FROM AULA_D  
GROUP BY CODIGO,NOMBRE
```

RESULTADO:

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
005	PAUL
007	LUISA

# ↑ ORDER BY

TABLA: AULA\_D

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

```
SELECT CODIGO, NOMBRE  
FROM AULA_D  
ORDER BY CODIGO
```

RESULTADO:

CODIGO	NOMBRE
001	JUAN
002	RAUL
002	RAUL
003	ANA
004	BRENDA
004	BRENDA
004	BRENDA
005	PAUL
007	LUISA

# 1 GROUP BY – ORDER BY

TABLA: AULA\_D

CODIGO	NOMBRE
001	JUAN
002	RAUL
003	ANA
004	BRENDA
002	RAUL
004	BRENDA
005	PAUL
004	BRENDA
007	LUISA

```
SELECT CODIGO, NOMBRE  
FROM AULA_D  
GROUP BY CODIGO, NOMBRE  
ORDER BY NOMBRE
```

RESULTADO:

CODIGO	NOMBRE
003	ANA
004	BRENDA
001	JUAN
007	LUISA
005	PAUL
002	RAUL



# **RECUPERACIÓN DE DATOS**

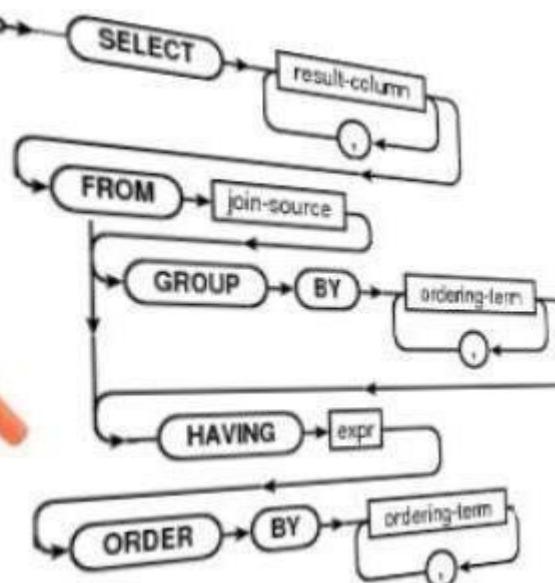


# RECUPERACION DE DATOS

El lenguaje de consulta estructurado (SQL) es un lenguaje de base de datos normalizado, utilizado por el motor de base de datos de Microsoft. SQL se utiliza para crear objetos QueryDef, como el argumento de origen del método OpenRecordSet y como la propiedad RecordSource del control de datos

# SQL

Funciones  
de agregación





## SENTENCIA SELECT

Es un comando que permite componer una consulta; este es interpretado por el servidor el cual recupera los datos especificados de las tablas. Su formato es:

```
SELECT  
[ALL | * | DISTINCT]  
[TOP VALOR [PERCENT]]  
[ALIAS.] COLUMNA [AS] CABECERA]  
  
FROM NOMBRE_TABLA [ALIAS]  
  
[WHERE CONDICION]  
[ORDER BY COLUMNA [ASC | DESC]]
```

# 1 SELECT

```
SELECT * | { [DISTINCT] column | expression [alias], ... }  
[FROM table]  
[WHERE condition(s)]  
[ORDER BY {column | expression} [ASC | DESC];]
```

todas las columnas  
cláusula de selección | suprime los duplicados | asigna el nombre de las cabeceras

especifica la tabla seleccionada  
restringe las filas que cumplen la condición  
indica las columnas por las que se realizará la ordenación  
determina el orden para cada columna

# 1 CONSULTAS BASICAS

A continuación se creara la base de datos y la tabla en donde se realizara las consultas:

Se tiene la base de datos DB\_TIENDA y dentro de ella la tabla productos:

```
USE MASTER  
GO  
SET DATEFORMAT DMY  
go  
  
if db_id('DB_TIENDA') is not null  
begin  
drop database DB_TIENDA  
end  
Go  
  
CREATE DATABASE DB_TIENDA  
GO  
  
USE DB_TIENDA  
go
```

```
create table productos  
(  
    CodPro      char(4) primary key,  
    descripcion varchar(30) not null,  
    StockInicial int not null,  
    unidadesVend int not null,  
    precioPro   money not null,  
    almacen     char(1) not null,  
    proveedor   varchar(30) not null,  
    fechalIngreso smalldatetime not null  
)  
go
```



## GENERANDO CONSULTAS BASICAS

**ingresar los siguientes datos a la tabla:**

```
insert into productos values('P001','Televisor',20,5,1800.00,'B','LG','02/05/2015')
```

```
insert into productos values('P002','Radio',30,12,900.00,'C','Sony','02/05/2015')
```

```
insert into productos values('P003','Cocina',40,15,600.00,'C','Imaco','03/05/2015')
```

```
insert into productos values('P004','Dvd',55,5,350.00,'D','Samsung','09/05/2015')
```

```
insert into productos values('P005','Horno',36,8,400.00,'A','Imaco','15/05/2015')
```

```
insert into productos values('P006','Refrigerador',53,34,2800.00,'B','Samsung','03/06/2015')
```

```
insert into productos values('P007','Lavadora',42,22,800.00,'C','Samsung','07/06/2015')
```

```
insert into productos values('P008','Plancha',31,15,200.00,'A','Imaco','15/06/2015')
```

```
insert into productos values('P009','Licuadora',76,23,180.00,'B','Oster','20/06/2015')
```

```
insert into productos values('P010','Computador',55,45,2200.00,'C','HP','23/06/2015')
```

```
insert into productos values('P011','Tostadora',19,6,90.00,'B','Oster','29/06/2015')
```

```
insert into productos values('P012','Microondas',22,16,450.00,'A','Samsung','05/07/2015')
```



# 1 MOSTRANDO DATOS USANDO OPERADOR (\*)

El operador asterisco cumple dos funciones dentro de la implementación de una consulta, la primera es mostrar todos los registros de la tabla y la segunda es mostrar todas las columnas que la componen con el orden especificado en su creación.

**Ejm 1: Crear el Script que permita mostrar todos los registros de la tabla productos:**

**Select \* from productos**

**Select** CodPro,descripcion,StockInicial,unidadesVend,precioPro,almacen,  
proveedor,fechalngreso  
**from** productos

**Select** productos.CodPro,productos.descripcion,productos.StockInicial,  
productos.unidadesVend,productos.precioPro,productos.almacen,  
productos.proveedor,productos.fechalngreso  
**from** productos



# CONSULTAS ESPECIFICANDO COLUMNAS

La especificación de las columnas permite determinar el orden de los campos en el resultado de la consulta, su objetivo es meramente visual; pues al final el resultado es el mismo

**Ejm 2:** Script que permita mostrar los campos CodPro, descripcion, unidadesVend y proveedor de la tabla productos.

Select **CodPro, descripcion, unidadesVend, proveedor** from  
productos

	CodPro	descripcion	unidadesVend	proveedor
1	P001	Televisor	5	LG
2	P002	Radio	12	Sony
3	P003	Cocina	15	Imaco
4	P004	Dvd	5	Samsung
5	P005	Homo	8	Imaco
6	P006	Refrigera...	34	Samsung
7	P007	Lavadora	22	Samsung
8	P008	Plancha	15	Imaco
9	P009	Licuadora	23	Oster
10	P010	Computa...	45	HP
11	P011	Tostadora	6	Oster
12	P012	Microondas	16	Samsung

# 1 CONSULTAS ESPECIFICANDO COLUMNAS Y UTILIZANDO ALIAS A LA TABLA.

El Alias se utiliza para identificar con un seudonimo la tabla de donde se obtienen las columnas.

**Ejm 3:** Script que permita mostrar los campos CodPro, descripcion y almacén de la tabla productos. Utilizar el alias “P” para la tabla productos

Select **p.codPro, p.descripcion, p.almacen** from productos**p**

	CodPro	descripcion	almacen
1	P001	Televisor	B
2	P002	Radio	C
3	P003	Cocina	C
4	P004	Dvd	D
5	P005	Horno	A
6	P006	Refrigerador	B
7	P007	Lavadora	C
8	P008	Plancha	A
9	P009	Licuadora	B
10	P010	Computador	C
11	P011	Tostadora	B
12	P012	Microondas	A

# 11 CONSULTAS ESPECIFICANDO ALIAS A LOS NOMBRES DE COLUMNAS

También se puede especificar un “alias” a los nombres de columnas mostradas en el resultado de una consulta SELECT

Ejm 4: Crear un script que permita mostrar los campos CodPro con el alias código, descripción con el alias Producto y unidadesvend con el alias Ventas de la tabla productos.

Select **CodPro as codigo, descripcion as producto, unidadesvend as ventas** from productos

	codigo	producto	ventas
1	P001	Televisor	5
2	P002	Radio	12
3	P003	Cocina	15
4	P004	Dvd	5
5	P005	Horno	8
6	P006	Refrigerador	34
7	P007	Lavadora	22
8	P008	Plancha	15
9	P009	Licuadora	23
10	P010	Computador	45
11	P011	Tostadora	6
12	P012	Microondas	16

# EJEMPLO

Crear un script que permita mostrar los campos codpro(código), descripcion(producto), StockInicial(Cantidad), UnidadesVend(Ventas) y proveedor de la tabla productos, utilizar alias para la tablaproductos

Select p.codPro Código, p.descripcion Producto,  
p.StockInicial Cantidad, p.unidadesvend Ventas,  
p.proveedor

from productos p

	Codigo	Producto	Cantidad	Ventas	proveedor
1	P001	Televisor	20	5	LG
2	P002	Radio	30	12	Sony
3	P003	Cocina	40	15	Imaco
4	P004	Dvd	55	5	Samsung
5	P005	Homo	36	8	Imaco
6	P006	Refrigerador	53	34	Samsung
7	P007	Lavadora	42	22	Samsung
8	P008	Plancha	31	15	Imaco
9	P009	Licuadora	76	23	Oster
10	P010	Computador	55	45	HP
11	P011	Tostadora	19	6	Oster

# 1 CONSULTA DISTINGUIDAS

Este tipo de consulta permita mostrar solo una ocurrencia de un conjunto de valores repetidos a partir de una columna de la tabla.

**Ejm 6:** Crear un script que permita mostrar todos los almacenes con los que cuenta la tabla productos.

Select **distinct p.almacen** from productos p

	almacen
1	A
2	B
3	C
4	D

# 1 CONSULTAS ORDENADAS

Este tipo de consulta permita mostrar la información de los registros ordenados de tal forma que podemos especificar que columnas deben ordenarse tanto de forma ascendente como descendente.

**Ejm 7:** Crear un script que permita mostrar todos los productos ordenados por descripción de forma descendente

Select p.\*

From productos p

order by p.descripcion desc

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalngreso
1	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
2	P010	Computador	55	45	2200.00	C	HP	2015-06-23 00:00:00
3	P004	Dvd	55	5	350.00	D	Sansung	2015-05-09 00:00:00
4	P005	Homo	36	8	400.00	A	Imaco	2015-05-15 00:00:00
5	P007	Lavadora	42	22	800.00	C	Sansung	2015-06-07 00:00:00
6	P009	Licuadora	76	23	180.00	B	Oster	2015-06-20 00:00:00
7	P012	Microondas	22	16	450.00	A	Sansung	2015-07-05 00:00:00
8	P008	Plancha	31	15	200.00	A	Imaco	2015-06-15 00:00:00
9	P002	Radio	30	12	900.00	C	Sony	2015-05-02 00:00:00
10	P006	Refrigerador	53	34	2800.00	B	Sansung	2015-06-03 00:00:00
11	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
12	P011	Tostadora	19	6	90.00	B	Oster	2015-06-29 00:00:00

# 1 CONSULTAS ORDENADAS

Crear un script que permita mostrar todos los productos ordenados primero por almacén de forma ascendente y luego por proveedor de forma descendente. Utilizar números para las columnas

Select p.\* from productos **p order by 6 asc, 7 desc**

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechaIngreso
1	P012	Microondas	22	16	450.00	A	Samsung	2015-07-05 00:00:00
2	P005	Horno	36	8	400.00	A	Imaco	2015-05-15 00:00:00
3	P008	Plancha	31	15	200.00	A	Imaco	2015-06-15 00:00:00
4	P006	Refrigerador	53	34	2800.00	B	Samsung	2015-06-03 00:00:00
5	P009	Licuadora	76	23	180.00	B	Oster	2015-06-20 00:00:00
6	P011	Tostadora	19	6	90.00	B	Oster	2015-06-29 00:00:00
7	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
8	P002	Radio	30	12	300.00	C	Sony	2015-05-02 00:00:00
9	P007	Lavadora	42	22	800.00	C	Samsung	2015-06-07 00:00:00
10	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
11	P010	Computador	55	45	2200.00	C	HP	2015-06-23 00:00:00
12	P004	Dvd	55	5	350.00	D	Samsung	2015-05-09 00:00:00



# CLÁUSULA WHERE DE LA SENTENCIA SELECT

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones que deben cumplir dichos resultados para su muestra

**Ejm 9:** Crear un script que permita mostrar todos los productos del almacén A

Select p.\* from productos p **where p.almacen='A'**

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalIngreso
1	P005	Horno	36	8	400.00	A	Imaco	2015-05-15 00:00:00
2	P008	Plancha	31	15	200.00	A	Imaco	2015-06-15 00:00:00
3	P012	Microondas	22	16	450.00	A	Samsung	2015-07-05 00:00:00

# EJEMPLO

Ejm 10: Crear un script que permita mostrar todos los productos que sus unidades vendidas sean igual a 5

Select p.\* from productos p **where p.unidadesVend=5**

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalIngreso
1	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
2	P004	Dvd	55	5	350.00	D	Samsung	2015-05-09 00:00:00



# OPERADORES DE COMPARACION

Operador	Descripción
=	Determina la igualdad entre dos valores.
>	Determina si el primer valor es mayor que el segundo.
>=	Determina si el primer valor es mayor o igual que el segundo.
<	Determina si el primer valor es menor que el segundo.
<=	Determina si el primer valor es menor o igual que el segundo.

Crear un script que permita mostrar todos los productos que sus unidades vendidas son mayores que 15

Select p.\* from productos p **where p.unidadesvend>15**

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalngreso
1	P006	Refrigerador	53	34	2800.00	B	Samsung	2015-06-03 00:00:00
2	P007	Lavadora	42	22	800.00	C	Samsung	2015-06-07 00:00:00
3	P009	Licuadora	76	23	180.00	B	Oster	2015-06-20 00:00:00
4	P010	Computador	55	45	2200.00	C	HP	2015-06-23 00:00:00
5	P012	Microondas	22	16	450.00	A	Samsung	2015-07-05 00:00:00



# OPERADORES LOGISTICOS

Los operadores lógicos soportados por SQL son: AND, OR, XOR, Eqv, Imp, Is y Not.

Ejm 12: Crear un script que permita mostrar todos los productos que sean del almacén c y el stock inicial sea mayor a 30

```
Select p.* from productos p where p.almacen='c' and  
p.StockInicial>30
```

	CodPro	descripcion	StockInicial	unidadesVend	precio Pro	almacen	proveedor	fechalngreso
1	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
2	P007	Lavadora	42	22	800.00	C	Samsung	2015-06-07 00:00:00
3	P010	Computador	55	45	2200.00	C	HP	2015-06-23 00:00:00

# EJEMPLO

Crear un script que permita mostrar todos los productos que sean de proveedor Imaco o LG

```
Select p.* from productos p where p.proveedor='imaco' or  
p.proveedor='lg'
```

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalIngreso
1	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
2	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
3	P005	Horno	36	8	400.00	A	Imaco	2015-05-15 00:00:00
4	P008	Plancha	31	15	200.00	A	Imaco	2015-06-15 00:00:00

# EJEMPLO

Crear un script que permita mostrar todos los productos que sus unidades vendidas estén entre 5 y 10 o sus unidades vendidas estén entre 20 y 40

```
Select p.* from productos p where (p.unidadesVend>=5 and  
p.unidadesVend<=10) or (p.unidadesVend>=20 and  
p.unidadesVend<=40)
```

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechaIngreso
1	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
2	P004	Dvd	55	5	350.00	D	Samsung	2015-05-09 00:00:00
3	P005	Horno	36	8	400.00	A	Imaco	2015-05-15 00:00:00
4	P006	Refrigerador	53	34	2800.00	B	Samsung	2015-06-03 00:00:00
5	P007	Lavadora	42	22	800.00	C	Samsung	2015-06-07 00:00:00
6	P009	Licuadora	76	23	180.00	B	Oster	2015-06-20 00:00:00
7	P011	Tostadora	19	6	90.00	B	Oster	2015-06-29 00:00:00

# ↑ OPERADOR LIKE

## expresión SQL

Se utiliza para comparar una expresión de cadena con un modelo en una

Crear un script que permita mostrar todos los productos que su descripción comience con D, luego siga una V,y sea solo de 3 dígitos.

```
select p.* from productos p where p.descripcion like 'D[v]_'
```

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalIngreso
1	P004	Dvd	55	5	350.00	D	Samsung	2015-05-09 00:00:00

# EJEMPLO

Crear un script que permita mostrar todos los productos que la descripción del producto comience con R, luego siga una e y el resto de caracteres podrá ser cualquiera

```
select p.* from productos p where p.descripcion like 'R[e]%'
```

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalIngreso
1	P006	Refrigerador	53	34	2800.00	B	Samsung	2015-06-03 00:00:00

# 1 OPERADOR BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo

Crear un script que permita mostrar todos los productos que su stock inicial este entre 20 y 40.

select p.\* from productos p **where p.StockInicial between 20 and 40**

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechaIngreso
1	P001	Televisor	20	5	1800.00	B	LG	2015-05-02 00:00:00
2	P002	Radio	30	12	900.00	C	Sony	2015-05-02 00:00:00
3	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
4	P005	Horno	36	8	400.00	A	Imaco	2015-05-15 00:00:00
5	P008	Plancha	31	15	200.00	A	Imaco	2015-06-15 00:00:00
6	P012	Microondas	22	16	450.00	A	Samsung	2015-07-05 00:00:00

# OPERADOR IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los valores que se encuentran en la lista

Crear un script que permita mostrar todos los productos que su precio sea 600, 800, 900

```
select p.* from productos p where p.precioPro in(600,800,900)
```

	CodPro	descripcion	StockInicial	unidadesVend	precioPro	almacen	proveedor	fechalngreso
1	P002	Radio	30	12	900.00	C	Sony	2015-05-02 00:00:00
2	P003	Cocina	40	15	600.00	C	Imaco	2015-05-03 00:00:00
3	P007	Lavadora	42	22	800.00	C	Samsung	2015-06-07 00:00:00

# 1 FUNCION DAY

Devuelve un valor numérico entero que representa el día según el mes de una determinada fecha

Script que permita mostrar el día correspondiente a la fecha actual,

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

	DIA
1	8

# EJEMPLO

Script que permita mostrar los datos de los productos indicando la fecha, día de ingreso y el código del producto:

```
SELECT p.FechalIngreso AS fecha, DAY(p.FechalIngreso) AS [DIA DE REGISTRO], p.codpro  
AS [CODIGO DE PRODUCTO] FROM productos P GO
```

	fecha	DIA DE REGISTRO	CODIGO DE PRODUCTO
1	2015-05-02 00:00:00	2	P001
2	2015-05-02 00:00:00	2	P002
3	2015-05-03 00:00:00	3	P003
4	2015-05-09 00:00:00	9	P004
5	2015-05-15 00:00:00	15	P005
6	2015-06-03 00:00:00	3	P006
7	2015-06-07 00:00:00	7	P007
8	2015-06-15 00:00:00	15	P008
9	2015-06-20 00:00:00	20	P009

# 1 FUNCION MONTH

Devuelve un valor numérico entero que representa al mes de una determinada fecha.

Script que permite mostrar el mes correspondiente a la fecha actual

```
select month(getdate()) AS Mes
```

	Mes
1	9

Script que permite mostrar el día y mes correspondiente al campo fechalingreso

```
select day(p.fechaingreso) as [DIA DE INGRESO],  
month(p.fechaingreso) as [MES  
DE INGRESO], p.codpro as [CODIGO DE PRODUCTO] from  
productos p  
GO
```

	DIA DE INGRESO	MES DE INGRESO	CODIGO DE PRODUCTO
1	2	5	P001
2	2	5	P002
3	3	5	P003
4	9	5	P004
5	15	5	P005
6	3	6	P006
7	7	6	P007
8	15	6	P008

# ↑ FUNCION YEAR

Devuelve un valor numérico entero que representa al año de una determinada fecha.

Script que permita mostrar el año actual.

```
select year(getdate()) as AÑO
```

AÑO
2015

Script que permita mostrar el día, mes, año y código correspondiente al campo fechalIngreso

```
select day(p.fechalingreso) AS [DIA DE INGRESO], month(p.fechalingreso) AS [MES DE INGRESO], year(p.fechalingreso) AS [AÑO DE INGRESO], p.codpro AS [CODIGO PRODUCTO] FROM productos p
```

	DIA DE INGRESO	MES DE INGRESO	AÑO DE INGRESO	CODIGO PRODUCTO
1	2	5	2015	P001
2	2	5	2015	P002
3	3	5	2015	P003
4	9	5	2015	P004
5	15	5	2015	P005
6	3	6	2015	P006
7	7	6	2015	P007
8	15	6	2015	P008

# 1 FUNCION DATEPART

Devuelve un valor numérico entero que representa una parte específica de una determinada fecha

Script que permita mostrar la fecha actual separada por día, mes y año

```
SELECT cast(datepart(DD,fechaingreso) as varchar(2))+'/'+
cast(datepart(MM,fechaingreso)as varchar(2))+'/'+
cast(datepart(YYYY,fechaingreso)as char(4)) as [FECHA DE INGRESO.],
fechaingresoAS [FECHA DE INGRESO] FROM productos
GO
```

	FECHA DE REG.	FECHA DE REG.
1	2/5/2015	2015-05-02 00:00:00
2	2/5/2015	2015-05-02 00:00:00
3	3/5/2015	2015-05-03 00:00:00
4	9/5/2015	2015-05-09 00:00:00
5	15/5/2015	2015-05-15 00:00:00
6	3/6/2015	2015-06-03 00:00:00
7	7/6/2015	2015-06-07 00:00:00
8	15/6/2015	2015-06-15 00:00:00



**VISTAS**



## ¿QUE ES UNA VISTA?

- Las vistas son consultas donde podemos visualizar los datos seleccionados, si actualizamos una vista, actualizaremos la tabla, y si actualizamos la tabla se actualizará la vista.
- La ventaja que tienen es que se pueden almacenar como objetos de la base de datos.

# ↑ CREACIÓN DE VISTAS

- Para crear una vista debemos utilizar la sentencia CREATE VIEW, debiendo proporcionar un nombre a la vista y una sentencia SQL SELECT válida.

```
CREATE VIEW <nombre_vista>  
AS  
(<sentencias_select>);
```



## EJEMPLO:

- Crear una vista sobre nuestra tabla alquileres, en la que se nos muestre el nombre y apellidos del cliente en lugar de su código.

**CREATE VIEW** vAlquileres

**AS**

**(**

**SELECT** nombre,  
apellidos,  
matricula

**FROM** tAlquileres, tClientes

**WHERE** ( tAlquileres.codigo\_cliente = tClientes.codigo )



## MODIFICACIÓN DE VISTAS

- Si queremos, modificar la definición de nuestra vista podemos utilizar la sentencia ALTER VIEW, de forma muy parecida a como lo hacíamos con las tablas. En este caso queremos añadir los campos fx\_alquiler y fx\_devolucion a la vista.

```
ALTER VIEW vAlquileres
```

```
AS
```

```
(
```

```
    SELECT nombre,  
          apellidos,  
          matricula,  
          fx_alquiler,  
          fx_devolucion
```

```
    FROM tAlquileres,
```

```
        tClientes
```

```
    WHERE ( tAlquileres.codigo_cliente = tClientes.codigo )
```

```
)
```



# ELIMINACIÓN DE VISTAS (DROP VIEW)

## DROP VIEW vAlquileres

- DROP VIEW elimina una o más vistas de la base de datos. Se debe poseer el privilegio DROP en cada vista a eliminar.



# **SUB CONSULTAS**



## SUBCONSULTAS

Las sub consultas son construcciones especiales que difícilmente se podrían realizar con JOINS.

Es una sentencia dentro de otra sentencia, por lo general se utiliza dentro del WHERE o inclusive dentro de la lista de selección

```
SELECT COD_VEN, NOM_VEN  
FROM TB_VENDEDOR  
WHERE COD_VEN IN ( SELECT COD_VEN  
                    FROM TB_FACTURA  
                    WHERE NUM_FAC = 'FA003' )
```



SUBCONSULTA

Lo que devuelve esta sentencia es:

Código y nombre del Vendedor que ha generado la factura 'FA003'



## SUBCONSULTA

**UPDATE** TB\_PRODUCTO

**SET** PRE\_PRO = (**SELECT** PRE\_PRO  
**FROM** TB\_PRODUCTO  
**WHERE** COD\_PROD='P007')

**WHERE** UNI\_MED='Uni'

**SUBCONSULTA**

Lo que genera esta sentencia es:

Actualiza el precio de los productos que será igual al precio del producto de Código 'P007', solo si la unidad de medida es UNI



## SUBCONSULTA - PREDICADOS

- a) ANY | SOME: recupera registros de consulta principal que satisfagan la **comparación** con cualquier otro registro recuperado en la sub-consulta
- b) ALL: recupera únicamente **aquellos** registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la sub-consulta
- c) IN: recupera únicamente aquellos registros de la consulta principal para los que algunos registros de la sub-consulta **contienen** un valor igual
- d) EXISTS: se utiliza en **comparaciones** de verdad/falso, para determinar si la sub-consulta devuelve datos



## SUBCONSULTA (ANY)

```
SELECT *  
FROM TB_PRODUCTO  
WHERE PRE_PRO > ANY
```

```
SELECT PRE_VEN  
FROM TB_DETALLE_FACTURA  
WHERE CAN_VEN>10)
```

PREDICADO ANY  
(CUALQUIERA)



## SUBCONSULTA

Esta Consulta devuelve los Productos cuyo precio son mayores a cualquiera de los precios de ventas de la tabla Detalle de Factura cuya cantidad vendida es mayor a 10



## SUBCONSULTA (EJEMPLO)

En una base de datos comercial, se pide mostrar los pedidos del cliente 000001, cuyo monto sea mayor al promedio de pedidos general

Observe el campo total y el cliente 000001, el promedio del campo total es 133 aproximadamente.  
El desarrollo de la consulta es el siguiente

```
SELECT idpedido,idcliente,total  
FROM PEDIDO  
WHERE idcliente='000001' AND total>(SELECT AVG(total) FROM pedido)
```

idpedido	ped_fecha	idcliente	total	estado
0000000001	02/10/2008	000001	100.00	1
0000000002	10/10/2008	000017	100.00	1
0000000003	01/06/2009	000001	200.00	1

idpedido	idcliente	total
0000000003	000001	200.00



**JOIN**

# JOIN

**INNER JOIN** Recupera información de dos o más tablas

## Sintaxis

```
SELECT campos  
FROM tabla1 INNER JOIN tabla2 ON tabla1.campo=tabla2.cmpo
```

<b>INNER JOIN</b>	cláusula para definir la tabla a relacionar
<b>ON</b>	Permite establecer la relación con el campo similar en las dos tablas, por lo general la clave primaria

# JOIN

Autor	Nacionalidad
Mario Marengo	Peru
Maria Velarde	Chile
Carla Bernales	Perú

```
SELECT n.nac_descripcion, a.nombre  
FROM AUTOR a INNER JOIN NACIONALIDAD n  
ON (n.idnacionalidad=a.idnacionalidad)
```

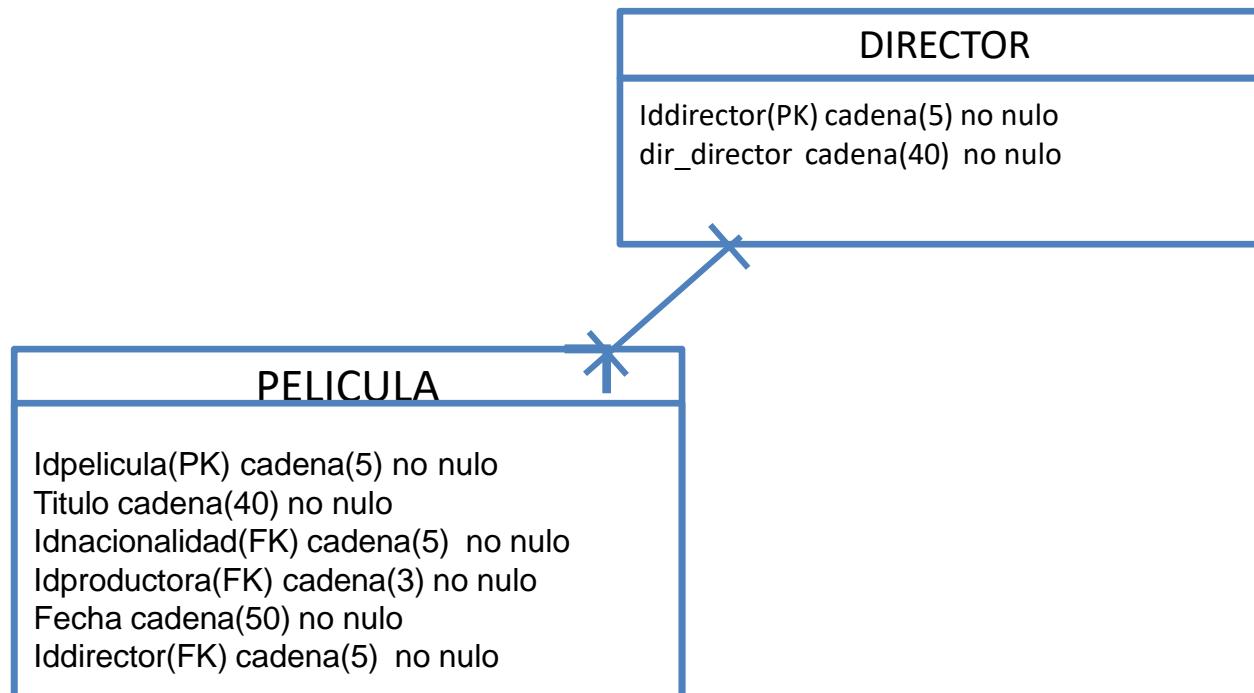
Observe que tanto la tabla NACIONALIDAD y AUTOR, ha cambiado el nombre AUTOR por a y NACIONALIDAD por n.

Estos nombre cortos se conocen como alias, sirven para tener nombres más reducidos, tanto de tablas como de campos.

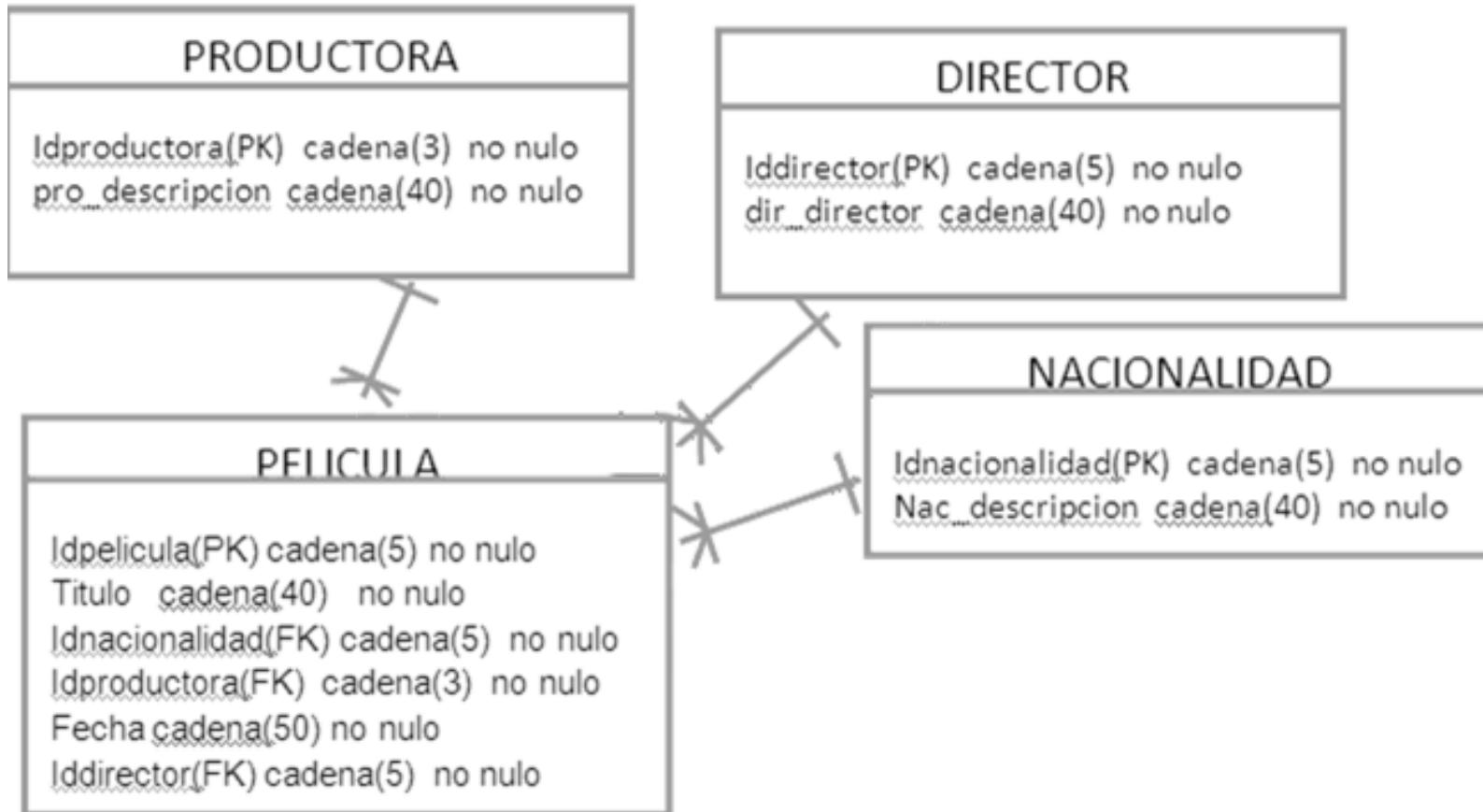


## JOIN - EJERCICIO

Elaborar una consulta SQL que muestre las películas con sus respectivos directores



# 1 JOIN - EJERCICIO

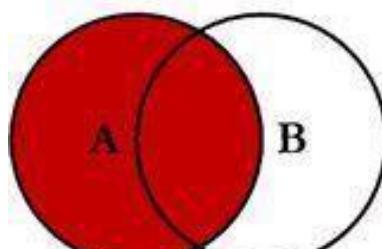


# 1 JOIN - EJERCICIO

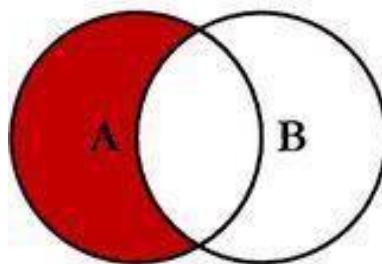
```
SELECT PE.TITULO,N.NAC_DESCRIPCION,D.DIR_DIRECTOR,P.PRO_DESCRIPCION  
FROM PELICULA PE INNER JOIN PRODUCTORA P ON (P.IDPRODUCTORA=PE.IDPRODUCTORA)  
          INNER JOIN DIRECTOR D ON (D.IDDIRECTOR=PE.IDDIRECTOR)  
          INNER JOIN NACIONALIDAD N ON (N.IDNACIONALIDAD=PE.IDNACIONALIDAD)
```

Otro aspecto importante es el campo para relacionar las tablas, por lo general, es el campo que se repite en las dos tablas, en la tabla **PRODUCTORA** y **PELICULA**, se repite **idproductora**, entonces este campo es el que se relaciona. Siguiendo este modelo se relacionan las demás tablas.

# SQL JOINS



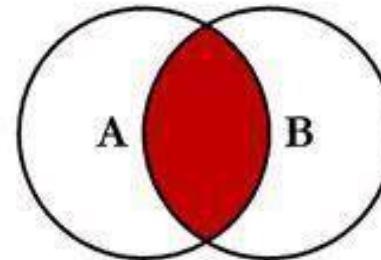
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



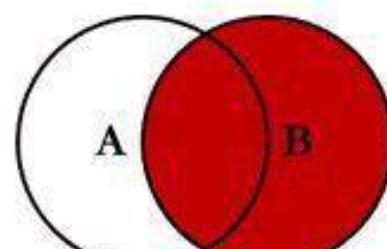
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

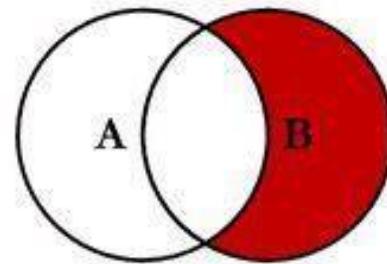
# SQL JOINS



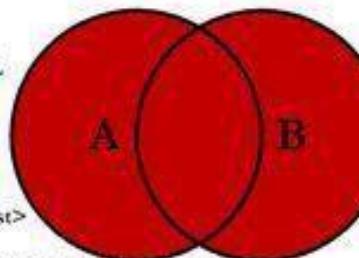
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



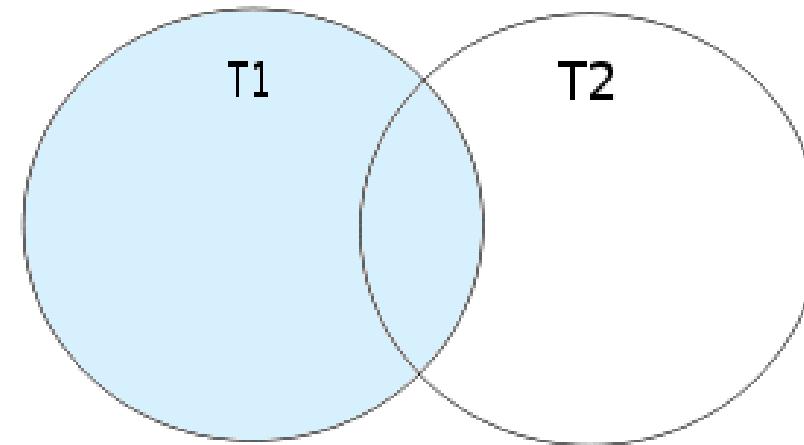
© C.L. Moffett, 2008

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# ↑ LEFT JOIN

Una LEFT JOIN devuelve los registros que están en la tabla de la izquierda (T1) tanto si tienen pareja en T2 como no.

Si tienen pareja, devuelve el dato relacionado, sino rellena los huecos con NULL.



# ↑ LEFT JOIN

PELICULAS

id	PELICULA	AÑO	idDirector
1	Four Rooms	1995	3
2	Die Hard	1988	1
3	The Hunt for Red October	1990	1
4	Psycho	1960	2

DIRECTORES

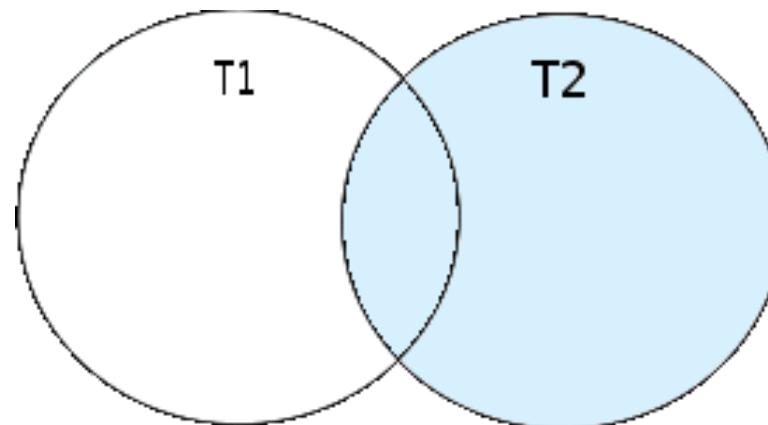
id	NOMDIRECTOR
1	John McTiernan
2	Alfred Hitchcock
3	Quentin Tarantino
4	Lidia Sanchez
5	Juan Pérez

```
SELECT D.id, D.NOMBREDIR , P.PELICULA  
FROM DIRECTORES AS D -- todos los directores  
LEFT JOIN PELICULAS AS P  
ON P.IDDIRECTOR = D.ID
```

```
SELECT D.id, D.NOMBREDIR , P.PELICULA  
FROM DIRECTORES AS D  
LEFT JOIN PELICULAS AS P  
ON P.IDDIRECTOR = D.ID  
WHERE P.PELICULA IS NULL
```

# RIGHT JOIN

Una RIGHT JOIN devuelve los registros que están en la tabla de la derecha (T2) tanto si tienen pareja en T1 como no.  
Si tienen pareja, devuelve el dato relacionado, sino rellena los huecos con NULL.



# RIGHT JOIN

PELICULAS

id	PELICULA	AÑO	idDirector
1	Four Rooms	1995	3
2	Die Hard	1988	1
3	The Hunt for Red October	1990	1
4	Psycho	1960	2

DIRECTORES

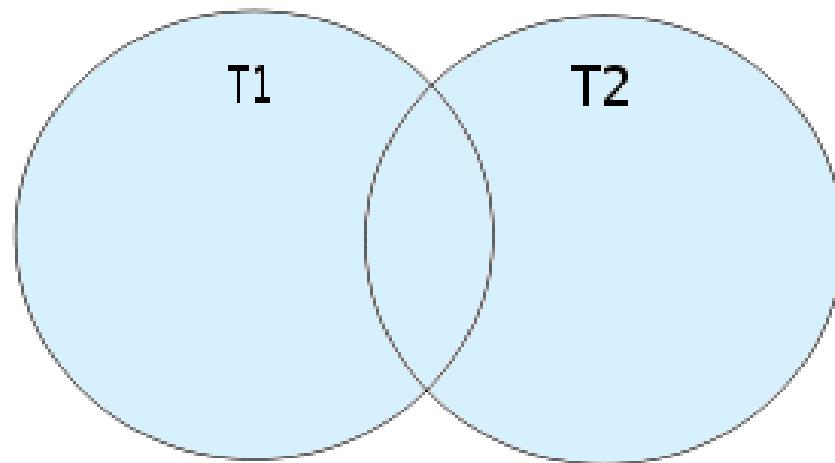
ID	NOMDIRECTOR
1	John McTiernan
2	Alfred Hitchcock
3	Quentin Tarantino
4	Lidia Sanchez
5	Juan Pérez

```
SELECT D.id, D.NOMBREDIR , P.PELICULA  
FROM PELICULAS AS P  
RIGHT JOIN DIRECTORES AS D—todos los directores  
ON P.IDDIRECTOR = D.ID
```

```
SELECT D.id, D.NOMBREDIR , P.PELICULA  
FROM PELICULAS AS P  
RIGHT JOIN DIRECTORES AS D  
ON P.IDDIRECTOR = D.ID  
WHERE P.PELICULA IS NULL
```

# ↑ FULL JOIN

Una FULL JOIN devuelve los registros que están tanto en la tabla de la derecha (T2) como en la tabla de la izquierda en T1 como los que están en ambas tablas.





# **INDICE**



- Un índice es una estructura de datos que permite acceder a diferentes filas de una misma tabla a través de un campo (o campos clave).
- Un índice permite un acceso mucho más rápido a los datos.
- Para entender lo que es un índice debemos saber primero como se almacena la información internamente en las tablas de una base de datos.
- Cada tabla se divide en páginas de datos, imaginemos un libro, podríamos escribirlo en "una sola hoja enorme" al estilo pergamo egipcio, o bien en páginas a las que podemos acceder rápidamente a través de un índice.



- Si queremos buscar la palabra zapato en un diccionario , ¿qué hacemos?
  - Leemos todo el diccionario hasta encontrar la palabra, con lo que nos habremos leído el diccionario enterito (¡seguro que aprenderíamos un montón!)
  - Buscamos en el índice en que página está la letra z, y es en esa página donde buscamos.



- Los índices se actualizan automáticamente cuando realizamos operaciones de escritura en la base de datos.
- Este es un aspecto muy importante de cara al rendimiento de las operaciones de escritura, ya que además de escribir los datos en la tabla se escribirán también en el índice.
- Las claves primarias son índices.
- Los nombres de los índices deben ser únicos.

- Las sentencias de SQL para manipular índices son:
  - CREATE INDEX;
  - DROP INDEX;



- La sintaxis para la creación de índices es la siguiente:

```
CREATE [UNIQUE] INDEX <nombre_index> ON <nombre_tabla>(  
<nombre_campo> [ASC | DESC]  
{,<nombre_campo> [ASC | DESC]} );
```

- La palabra clave UNIQUE especifica que no pueden existir claves duplicadas en el índice.  
ASC | DESC especifican el criterio de ordenación elegido, ascendente o descendente, por defecto es ascendente.

- Creamos la tabla CLIENTES, este ejemplo crea un índice único en el campo IdCliente. Esto nos permitirá buscar mucho mas rápido por el campo IdCliente y nos asegurará que no tengamos dos IdCliente iguales.
- **CREATE UNIQUE INDEX UIX\_CLIENTES\_IdCliente ON CLIENTES (IdCliente);**

- Para eliminar un índice debemos emplear la sentencia DROP INDEX.
- `DROP INDEX <nombre_tabla>.<nombre_indice>;`
- Ejemplo: Para eliminar el índice creado anteriormente.
- **`DROP INDEX CLIENTES.UIX_CLIENTES_IdClientes;`**



# ¿Preguntas o comentarios?



# ↑ CONCLUSIONES

- SQL nos permite ingresar comandos o sentencias de tal manera que podemos administrar o crear una base de datos.
- Los comandos SQL permiten generar información desde la creación, modificación o mantenimiento a tablas las cuales también nos permiten recuperar datos o importarlos.



# REFERENCIAS BIBLIOGRÁFICAS

## REFERENCIAS

Coronel, Carlos; Morris Steven y Rob, Peter. Bases de Datos: Diseño, Implementación y Administración. 2011

**GRACIAS**

