

# **Program BI**

## Curso SQL Básico Intermedio

*Apunte*

Autor:  
Juan Carrasco Pastrián

2021

# 1 Introducción a SQL

## 1.1 Introducción a las bases de datos

Una Base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados metodológicamente para su uso a futuro.

Actualmente gracias al desarrollo tecnologico que se esta dando y a la gran cantidad de datos que se tienen, han surgido varias soluciones al almacenamiento de datos. Dichas soluciones se conocen formalmente como **sistemas de gestion de bases de datos, o DBMS** por sus siglas en inglés. Estos sistemas permiten gestionar las bases de datos, sus sub-componentes como tablas o consultas, y utilizan un lenguaje estándar de consulta llamado SQL que significa “Lenguaje estructurado de consultas (Structured Query Language)” Los principales Motores de bases que estan en el mercado son: ORACLE, MySQL, SQL Server, PostGreSQL, SQLite.

## 1.2 Instalación de DBMS en máquina local

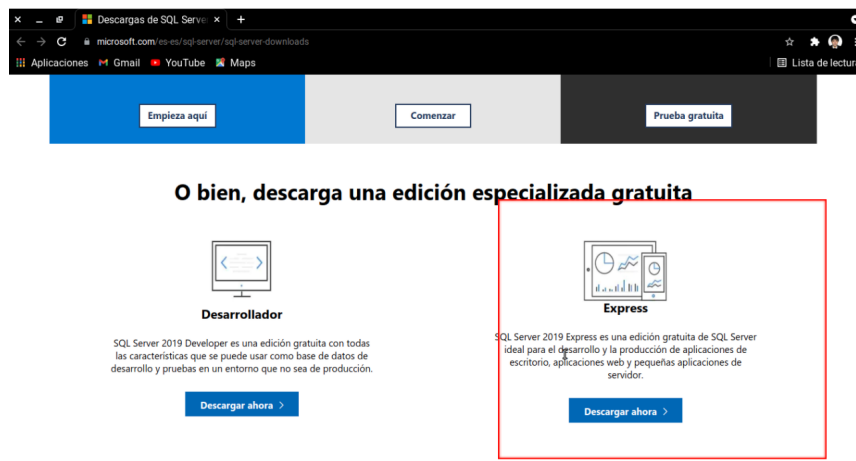
Los sistemas DBMS funcionan con un paradigma cliente-servidor, el cual es ampliamente frecuente en sistemas computacionales. En este caso, por medio de la instalación de SQL Server lo que haremos será transformar nuestro PC local en un servidor de SQL (vale la pena repetir: local y además limitado a las capacidades de nuestro PC). Para hacer peticiones a este servidor, instalaremos un programa de gestión de base de datos que actuará como cliente; es decir, en nuestro PC tendremos el servidor y el cliente instalado a la vez.

Para este curso usaremos como servidor el SQL Server de Microsoft, y como clientes podremos usar SQL Server Management Studio (SSMS)

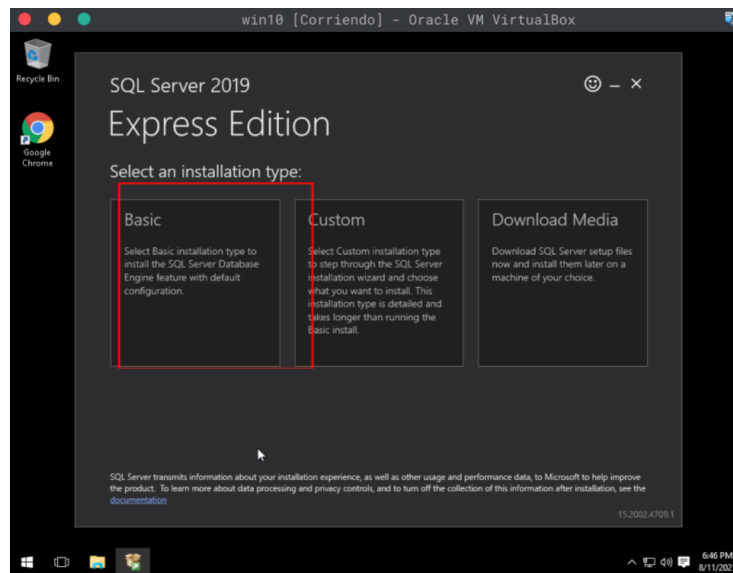
o Azure Data Studio, los cuales procederemos a explicar su instalación en el PC local.

### 1.2.1 Instalación del servidor

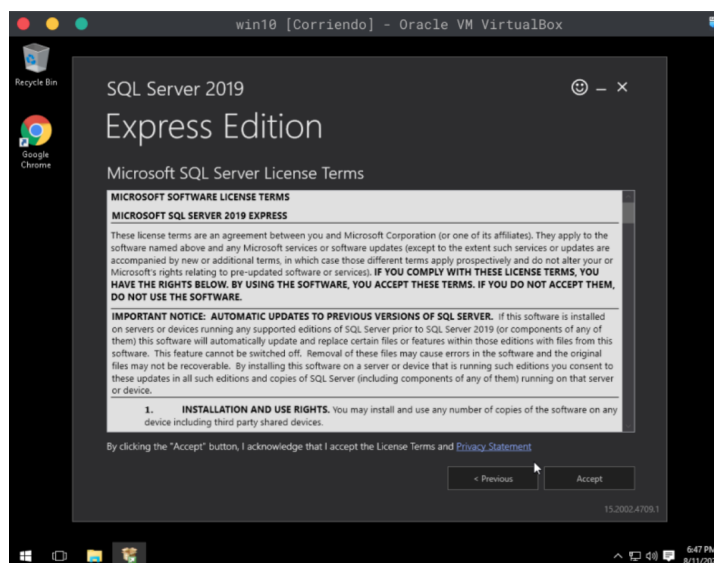
1. Lo primero que haremos será transformar nuestro PC en un servidor local de SQL. Para ello descargamos de <https://www.microsoft.com/es-es/sql-server/sql-server-downloads> la versión Express de SQL Server 2019:



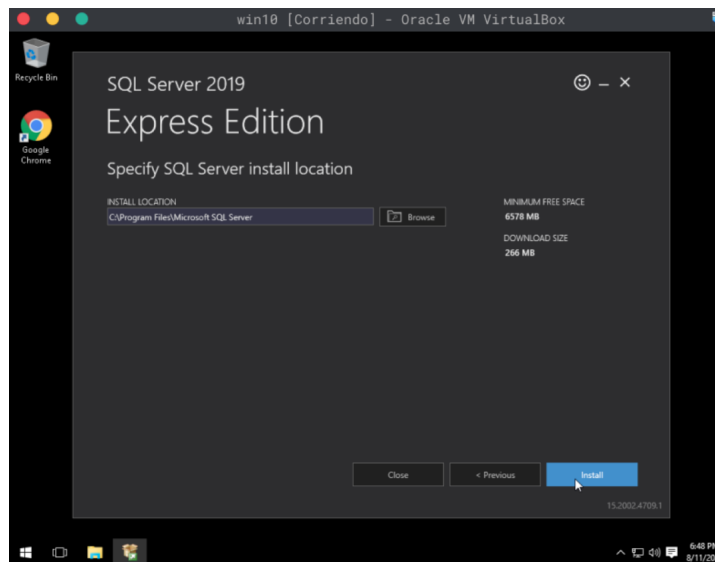
2. En la ventana anterior, presionar descargar ahora.
3. Abrimos el archivo descargado y seleccionamos la opción “Básica” de instalación.



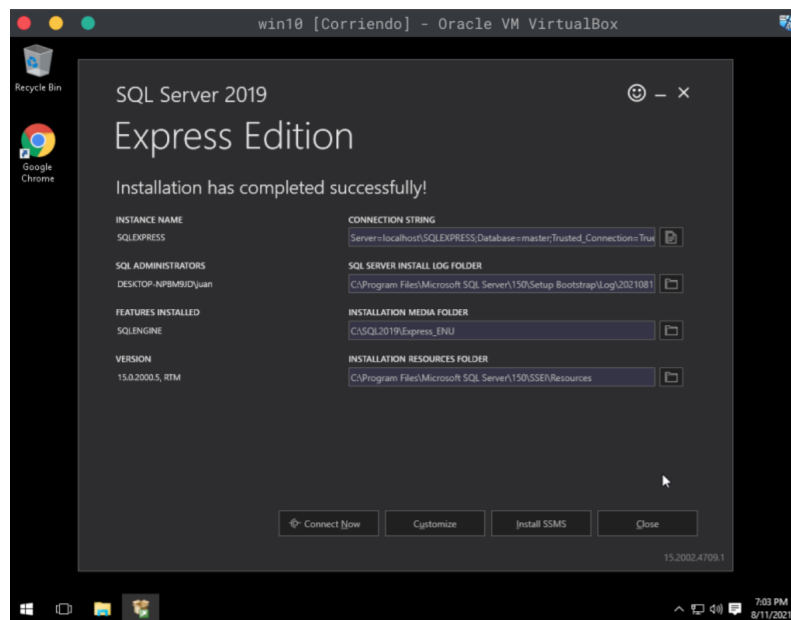
#### 4. Aceptamos los términos de la licencia



#### 5. Nos indicará la ruta donde se instalará SQL Server, le damos siguiente.



6. El instalador empezará a descargar e instalar la plataforma. Asegurarse de tener una conexión estable para que el instalador proceda sin inconvenientes. Cuando el instalador haya terminado aparecerá la siguiente ventana

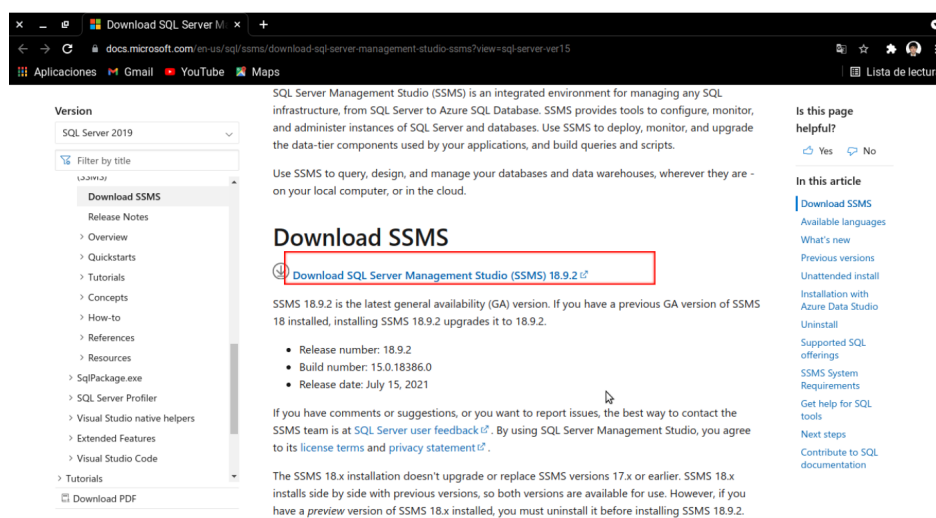


7. Se nos dá la opción de instalar SSMS inmediatamente, lo podemos hacer pulsando el botón “Install SSMS” que nos llevará a la página de descarga, y seguir de manera manual las instrucciones del próximo apartado.

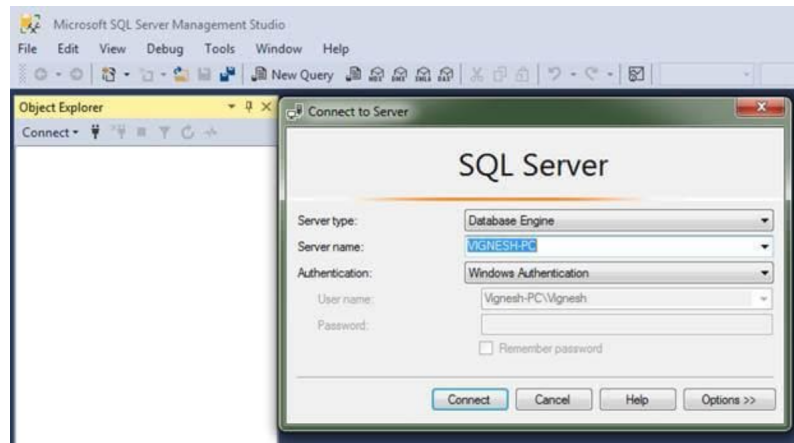
## 1.2.2 Instalación de SSMS

**IMPORTANTE:** el orden es fundamental. Se requiere primero instalar SQL Server(apartado anterior) antes de instalar el SQL Management Studio.

1. Descargamos el programa desde aquí <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>



2. Se iniciará la descarga del SQL Server Management Studio (archivo de más de 600 MB) Aceptamos los términos de la licencia y procedemos a instalar.
3. Al abrir SSMS nos aparecerá una ventana como la siguiente si la instalación es correcta. Importante que el programa reconozca como nombre del servidor el PC local, ya que eso indica que reconoce una instancia local de SQL server (instalada anteriormente según la primera parte de este documento). De ser así, basta con autenticación de Windows (en Authentication) para poder conectarse al servidor local.



4. Al presionar “Connect” ya estamos exitosamente conectados al servidor local de SQL y podremos trabajar en las actividades del curso.

### 1.2.3 Instalación de Azure Data Studio

**IMPORTANTE:** el orden es fundamental. Se requiere primero instalar SQL Server(apartado anterior) antes de instalar Azure Data Studio

1. Ingresamos al sitio de descarga de Azure Data Studio <https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15#download-azure-data-studio> y bajamos la versión correspondiente a nuestro sistema.

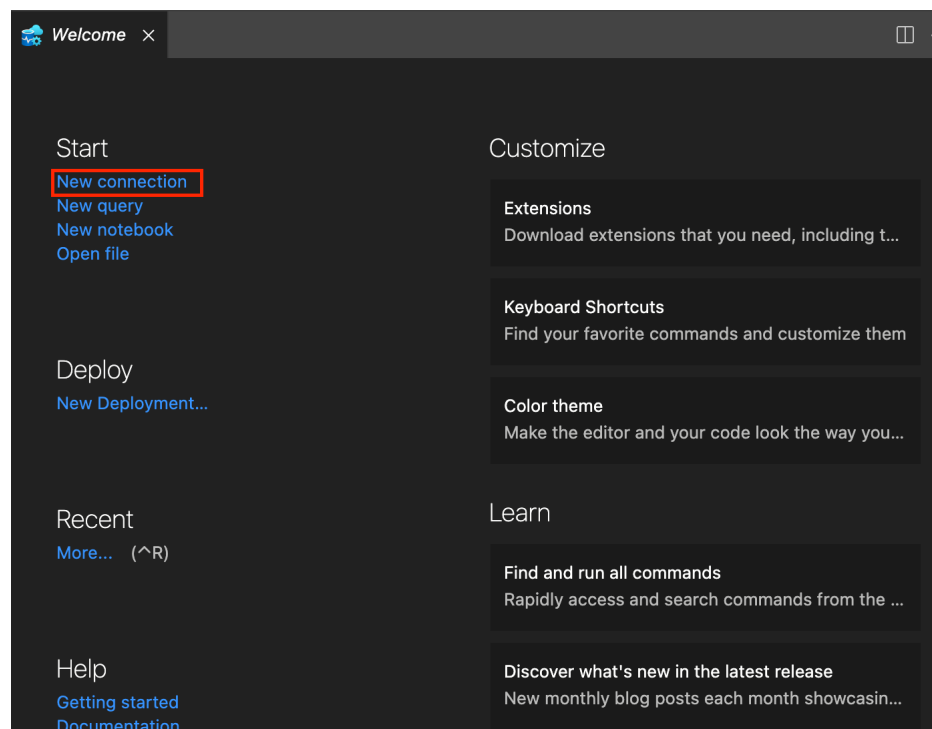
## Download Azure Data Studio

Azure Data Studio 1.32.0 is the latest general availability (GA) version.

- Release number: 1.32.0
- Release date: August 18, 2021

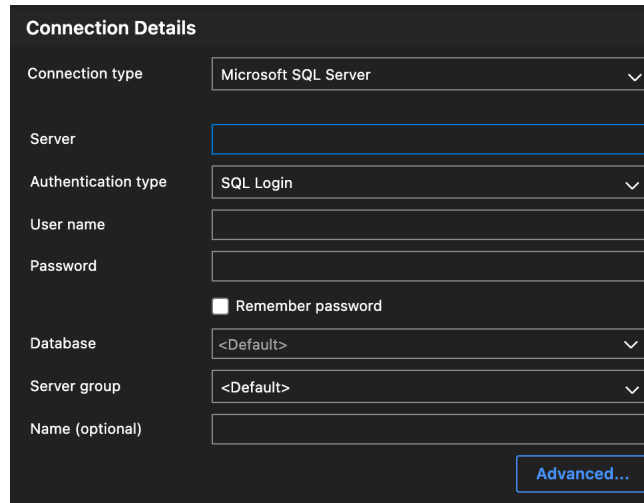
Platform	Download
Windows	<a href="#">User installer</a> <sup>↗</sup> (recommended) <a href="#">System installer</a> <sup>↗</sup> <a href="#">.zip file</a> <sup>↗</sup>
macOS	<a href="#">.zip file</a> <sup>↗</sup>
Linux	<a href="#">.deb file</a> <sup>↗</sup> <a href="#">.rpm file</a> <sup>↗</sup> <a href="#">.tar.gz file</a> <sup>↗</sup>

2. Abrimos el instalador y se comenzará a instalar el programa.
3. Una vez instalado debemos crear una conexión con nuestro servidor local de SQL Server. Para ello en la ventana bienvenida de Azure Data Studio, seleccionamos la opción "New Connection"



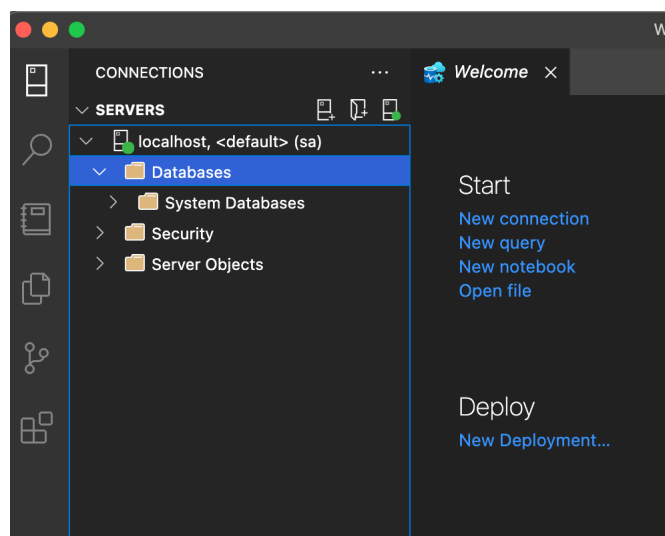


4. Aparecera una ventana emergente donde ingresaremos los parámetros de la conexión:



A lo que respondemos: Server=localhost / Authentication Type=Windows Authentication / Database=<default> / Server group =<default>.

5. Una forma de chequear que el proceso haya sido correcto es revisando la pestaña Servers de Azure, donde debiera aparecer nuestro localhost conectado y con las subcarpetas de trabajo, tal como se muestra en la imagen:



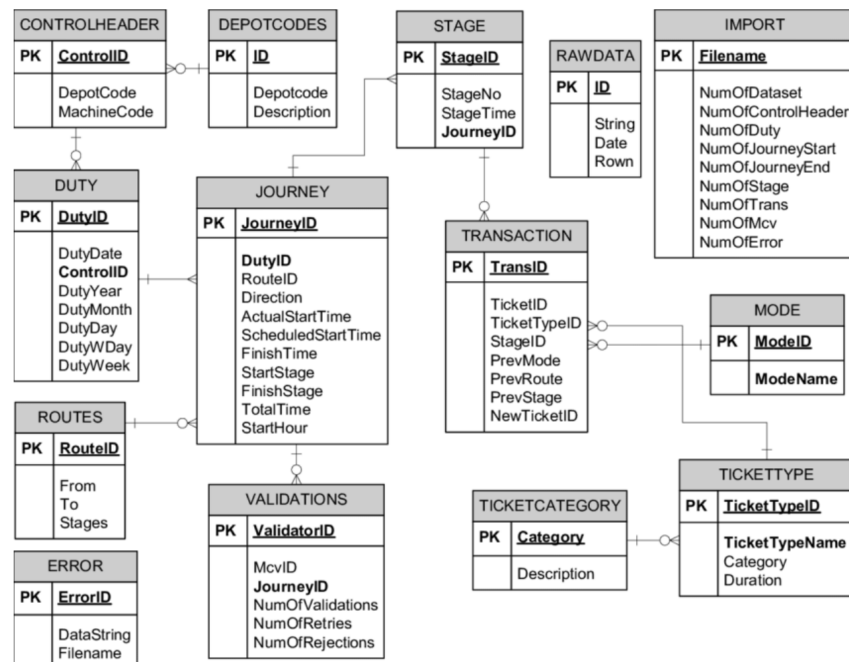
## 1.3 Tablas

Como objeto clave de una BBDD, Una tabla es un objeto que esta formado por columnas, las cuales están interrelacionadas a través de la información que contienen.

En jerga de bases de datos a una columna se le conoce como campo, mientras que una fila se le conoce como registro.

En una BBDD, las tablas tienen un nombre único. Una BBDD puede contener varias tablas, y las tablas estar relacionadas entre sí por campos que comparten, llaves (un tipo de campo que revisaremos más adelante) y relaciones de cardinalidad entre sus registros (que también veremos más adelante).

El conjunto de tablas y relaciones en una BBDD se conoce como el Esquema de una BBDD y se puede representar gráficamente en un diagrama que se conoce como diagrama Entidad-Relación (como el que se muestra a continuación)



## 1.4 Tipos de Datos

Las bases de datos en las que se puede consultar con SQL se les conoce como bases de datos relacionales, mientras que los datos que se almacenan en ellas son datos estructurados; es decir, corresponden a un tipo bien específico de dato que caerá en las siguientes categorías:

- Tipos Numéricos
  - **Int:** Número enteros, en un rango de  $-2^{31}$  a  $2^{31}$ .
  - **Bigint:** Número entero entre  $-2^{63}$  y  $2^{63}$
  - **Bit:** Para un número entero que puede ser 0 ó 1.
  - **Float:** Números decimales
- Tipos de Texto/Fecha
  - **Char:** Este tipo es para cadenas de longitud fija. Su longitud va desde 1 a 255 caracteres. Siempre se ocupara el largo que hayamos dado (añadiendo espacios en el caso que faltasen caracteres).
  - **Varchar:** Para una cadena de caracteres de longitud variable de hasta 8.060.
  - **Nvarchar:** Para una cadena de caracteres de longitud variable de hasta 4000, y soporta formato Unicode.
  - **Text:** Texto de longitud variable que puede tener hasta 65.535 caracteres.
  - **Date:** Para almacenar fechas. El formato por defecto es yyyyymmdd.
  - **Datetime:** Combinacion de fecha y hora.

## 1.5 Declaraciones DDL

Las declaraciones DDL o Data Definition Language corresponden a aquellas que intervienen de alguna manera el esquema de la BBDD o manipulan nuevos objetos adyacentes a las tablas. Las declaraciones DDL suelen empezar con un comando CREATE en caso de que se crean objetos, ALTER para modificarlos y DROP para borrarlos.

Por ejemplo, al crear una base de datos se puede usar un comando CREATE como sigue:

```
1 CREATE DATABASE mi_base_de_datos
```

Acto seguido puedo exigirle a SQL que apunte a esa base de datos de ahora en adelante para trabajar sobre ella:

```
1 USE mi_base_de_datos
```

Use una declaración CREATE TABLE para especificar el diseño de su tabla:

```
1 CREATE TABLE nombre-tabla (campo1 tipo1, campo2  
    tipo2 ... )
```

ALTER TABLE cambia la estructura de una tabla. Por ejemplo, puede agregar o eliminar columnas, crear o borrar índices, cambiar el tipo de columnas existentes o cambiar el nombre de las columnas o de la tabla en sí.

```
1 ALTER TABLE nombre-tabla (campo1 tipo1, campo2  
    tipo2 ... )
```

También puede ir acompañado de borrar columnas:

```
1 ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d
```

Para borrar una tabla, usar comando DROP TABLE. Para vaciar usar TRUNCATE TABLE:

```
1 DROP TABLE nombre-tabla
2 TRUNCATE TABLE nombre-tabla
```

Ojo: operaciones CREATE/ALTER/DROP requieren permisos de escritura

## 1.6 Inserción de registros y populating

INSERT crea una fila con cada columna establecida en su valor pre-determinado

```
1 INSERT INTO tbl_name ( campos...) VALUES(
    valores...);
```

Es decir, se cargan una lista de campos determinados (que pueden ser todos) y sus valores respectivos. Un valor de un campo también se le conoce como *Instancia*.

Para poblar tablas desde archivos externos, los datos deben respetar las convenciones geográficas del cliente(PC) que hace la carga.

- Las cadenas de texto siempre deben ir entre comillas.
- Por ejemplo las fechas en SQL son en formato yyyy-mm-dd. Deben ir entre comillas como cadenas de texto.
- Los números decimales separan el número con la parte decimal con un punto.

Desde SSMS se pueden importar archivos excel, planos de texto (txt) o separados por comas (csv) por medio de un asistente. Acceder a un asistente se puede utilizando el *boton derecho del mouse sobre la base de datos* → *Import Data* o *Import Flat File*.

En el caso de Azure Data Studio, se puede importar instalando una extensión llamada *SQL Server Import* que permitirá hacer tareas

similares al asistente de SSMS. Más info en <https://docs.microsoft.com/en-us/sql/azure-data-studio/extensions/sql-server-import-extension?view=sql-server-ver15>

## 1.7 Consultas en SQL

La sintaxis clásica de una consulta en SQL es la siguiente (vamos a revisar sus partes en las próximas slides)

```
1  --Sintaxis consulta
2  SELECT campo1, campo2, ...
3  FROM tabla
4  WHERE condicion
5  ORDER BY campo1 ASC/DESC
6  --Seleccionar toda una tabla
7  SELECT * FROM tabla
```

Adicionalmente puedo copiar valores de una tabla\_1 a otra tabla\_2 utilizando una sintaxis combinada de INSERT y SQL:

```
1  --copia de tabla_1 a tabla_2
2  INSERT INTO tabla_2 SELECT * FROM tabla_1
```

## 1.8 Condicionales en cláusula WHERE

La cláusula WHERE involucra condiciones. En general una condicion es una proposición lógica, es decir un enunciado cuyo valor es verdadero o falso. Siendo WHERE una instrucción en bloque (como SELECT también), evaluará qué registros de un campo cumplen con la condición, y los filtrará del bloque final.

```
1  --CONDICIONALES
2  --operadores
3  WHERE campo > valor  -- mayor que
4  WHERE campo < valor  --menor que
5  WHERE campo >= valor --mayor o igual
6  WHERE campo <= valor --menor o igual
7  WHERE campo <> valor -- distinto
8  WHERE campo = valor  --igual
9  WHERE campo LIKE patron (veremos en Seccion 2)
10 WHERE campo IN (valor1, valor2...) --si el campo
    esta en un conjunto de valores
11 WHERE campo BETWEEN rangomin AND rangomax --si
    el campo esta entre rangomin y rango max
12 WHERE campo IS NULL  -- filas en donde el campo
    seleccionado es null
13 WHERE campo IS NOT NULL --filas donde el campo
    no es null (vacio)
```

Respecto a lo anterior, entenderemos los valores NULL, como un dato vacío que posee algún campo para un registro específico en un tabla. Es decir, una celda donde no hay nada (ni siquiera espacios en blanco, por lo que una celda en blanco podría no ser NULL; en general las celdas vacías mostrarán un NULL para evitar confusiones).

Como WHERE evalúa proposiciones lógicas, sus valores de verdad pueden ser sujeto a los operadores tradicionales lógicos (de conjunción, disyunción, negación).

```

1  -Operadores Logicos
2  --operador AND
3  SELECT campo1, campo2, ...
4  FROM tabla
5  WHERE condicion1 AND condicion2 AND condicion3
   ...
6  --operador OR
7  SELECT campo1, campo2, ...
8  FROM tabla
9  WHERE condicion1 OR condicion2 OR condicion3...
10 --operador NOT
11 SELECT campo1, campo2, ...
12 FROM tabla
13 WHERE NOT condicion;

```

## 1.9 Comando ORDER BY

Cuando usa la instrucción SELECT para consultar datos de una tabla, el conjunto de resultados no se ordena. Significa que las filas del conjunto de resultados pueden estar en cualquier orden.

Para ordenar el conjunto de resultados, agrega la cláusula ORDER BY a la instrucción SELECT. A continuación se ilustra la sintaxis de la cláusula ORDER BY:

```

1  SELECT
2      (lista)
3  FROM
4      tabla_nombre
5  WHERE ...
6  ORDER BY columna1 ASC columna2 DESC

```



## 1.10 Comando DISTINCT

Dentro de una tabla, una columna a menudo contiene muchos valores duplicados; a veces solo desea enumerar los valores distintos. El comando `DISTINCT` elimina repeticiones de una tupla de datos en una fila.

```
1 SELECT DISTINCT columna1, columna2, ...  
2 FROM tabla_nombre;
```

## 2 Clase 2: de los Wildcards a las Vistas

### 2.1 Wildcards o patrones

Como mencionamos anteriormente, el comparador LIKE se utiliza para encontrar patrones o declarar filtros que permiten de forma eficiente obtener sub-conjuntos de los datos. Para representar patrones que serán utilizados en un comparador LIKE, se utilizan los siguientes caracteres:

- %: Representa cero o más caracteres
- \_: Representa 1 sólo caracter
- []: Representa un conjunto o intervalo de caracteres, según lo que se especifique dentro de los corchetes.

De esta manera podemos encontrar algunos ejemplos como los siguientes:

```
1 WHERE campo LIKE 'a%' --Busca cualquier valor
   que comience con "a"
2 WHERE campo LIKE '%a' --Encuentra cualquier
   valor que termine con "a"
3 WHERE campo LIKE '%or%' --Busca cualquier valor
   que tenga "o" en cualquier posicion
4 WHERE campo LIKE '_r%' --Encuentra cualquier
   valor que tenga "r" en la segunda posicion
5
6 WHERE campo LIKE 'a_-%' --Busca cualquier valor
   que comience con "a" y tenga al menos 3
   caracteres de longitud
7 WHERE campo LIKE 'a%o' --Busca cualquier valor
   que comience con "a" y termine con "o"
```

## 2.2 Cláusula GROUP BY y funciones

La declaración GROUP BY agrupa las filas que tienen los mismos valores en un campo determinado, como "encontrar el número de clientes en cada país". La instrucción GROUP BY se utiliza a menudo con funciones de agregado (COUNT(), MAX(), MIN(), SUM(), AVG()) y agrupando el conjunto de resultados de una o más columnas.

```
1 SELECT *, funcion de agregado
2 FROM tabla
3 WHERE condicion
4 GROUP BY *
```

Nota: importante que al especificar una función de agregado según un lote de campos previos, el lote de campos deberá ser escrito en GROUP BY, que es lo que se señala con \* en la query anterior.

## 2.3 Llave primaria y campo autoincremental

Una llave primaria (o *primary key*, PK) es un campo (o combinación de campos) que identifica de manera única a cada fila en una tabla

```
1 CREATE TABLE nombre(f1 t1, f2 t2, PRIMARY KEY(
   campo_llave))
2 --alternativa
3 CREATE TABLE nombre(f1 t1 PRIMARY KEY, f2 t2...
```

La forma general de crear una tabla con un campo autoincremental es la siguiente

```
1 CREATE TABLE nombre(f1 int IDENTITY, f2 t2,...)
```

En el caso anterior, el campo f1 es autoincremental, y siempre es un tipo de dato int.

## 2.4 Borrar y actualizar registros

Como parte de las operaciones de escritura (que deben estar debidamente autorizadas) se pueden borrar filas o actualizar filas según una condición en particular. La sintaxis de estas operaciones es la siguiente:

```
1  --Borrar registro
2  DELETE FROM tabla WHERE condicion
3  -- Actualizar registro
4  UPDATE tabla SET campo1=.., campo2=..
5  WHERE condicion
```

Recomendación: trabaje con esta programación en ambientes de desarrollo, ya que puede perder datos irreversiblemente si lo hace en un ambiente de producción.

Nota: Cuando usamos comando DELETE en nuestras tablas es necesariamente un cambio irreversible. Mientras que algunas operaciones con UPDATE pueden ser reversibles (matemáticamente u operacionalmente); por ejemplo, duplicar una fila con un UPDATE puede ser deshecho con otro UPDATE que divida a la mitad.

## 2.5 Declaración de Variables y Funciones

Podemos usar la instrucción DECLARE para indicar o declarar una o más variables. A partir de ahí, podemos utilizar el comando SET para inicializar o asignar un valor a la variable.

```
1  DECLARE @variable tipo
2  SET @variable=valor
```

Una función definida por el usuario es una rutina que acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como un valor.

$$inputs(x_1, x_2, \dots) \rightarrow f(x_1, x_2, \dots) \rightarrow output$$

El valor de retorno puede ser un valor escalar (único) o una tabla. Una sintaxis para una *función escalar* es la siguiente:

```

1 CREATE FUNCTION nombre_funcion(par1 t1, par2 t2
   ...)
2 RETURNS tipo_variable_salida
3 BEGIN
4 ....
5 returns @variable_salida
6 END

```

Nota sobre el código anterior: en resumen, creamos una variable de salida dentro del código de la función, que va a recibir el resultado del procedimiento que ejecute la función.

## 2.6 Condicionales usando CASE-WHEN

La instrucción CASE pasa por condiciones y devuelve un valor cuando ante la primera condición que se cumpla. Entonces, una vez que una de las condiciones es verdadera, dejará de leer y devolverá el resultado. Si no se cumple ninguna condición, devuelve el valor de la cláusula ELSE. Una sintaxis para un CASE-WHEN es la siguiente:

```
1 SELECT campo1, campo2, ....
2 CASE WHEN condicion1 THEN resultado1
3 WHEN condicion2 THEN resultado2
4
5 ...
6
7 ELSE resultadoN
8 END
9 FROM tabla
```

## 2.7 Introducción a las Vistas

En SQL, una vista es una tabla virtual basada en el conjunto de resultados de una declaración SQL. Una vista contiene filas y columnas, como una tabla real. Los campos de una vista son campos de una o más tablas reales de la base de datos.

Puede agregar sentencias y funciones SQL a una vista y presentar los datos como si vinieran de una sola tabla. Se crea una vista con la instrucción `CREATE VIEW`. También se puede modificar una vista (con `ALTER VIEW`) o eliminarla con `DROP VIEW`

```
1 CREATE VIEW nombre_vista AS
2 ..query con inputs de vista
3 --SE PUEDE USAR ALTER/DROP
4 DROP VIEW nombre_vista
5 ALTER VIEW nombre_vista AS
6 ...query con cambios
```

Notas importantes sobre las vistas:

- Si se modifican los datos de una tabla, la vista alimentada por dicha tabla se actualizará automáticamente (ya que es una query

almacenada que se ejecuta cuando se usa).

- Puedo ver cómo está constituida una vista en SSMS, botón derecho en la vista → opción "Design".

## 3 Clase 3: Subqueries, Tablas Temporales y Consultas de cruce

### 3.1 Subconsultas

Una subconsulta es una consulta SQL anidada dentro de una consulta más grande. Una subconsulta puede ocurrir en:

- Una cláusula SELECT
- Una cláusula FROM
- Una cláusula WHERE

La consulta interna se ejecuta primero antes que su consulta principal para que los resultados de una consulta interna se puedan pasar a la consulta externa.

```
1  --Sintaxis
2  SELECT subconsulta.columnas
3  FROM (SELECT columnas FROM tabla WHERE ...) as
        subconsulta
4  WHERE ....
```

```
1  --Subqueries en WHERE
2  SELECT ... FROM... WHERE campo IN (Subquery)
3  --Subquery en SELECT
4  SELECT campo=subquery FROM .... WHERE....
```

Observación: es importante referenciar la subconsulta y sus campos con el comando as, ya que al requerirla después en el SELECT de afuera debemos llamarla por sus nombres.



## 3.2 Tablas Temporales

Permiten generar una tabla a partir de una consulta. Se le denomina tabla temporal por que queda almacenada en memoria caché (temporal) mientras la sesión está abierta o el servidor permanece corriendo; una vez cerrada la sesión o apagado el servidor, la tabla desaparece con sus datos ingestados. La sintaxis es la siguiente:

```
1 SELECT ....
2 INTO #tablatemporal
3 FROM ...
4 WHERE ...
```

Nota: usar un # para acceso local, ## para acceso global **Importante:** La tabla temporal siempre quedará en una BBDD temporal del servidor que se llama tempdb; no es un objeto que quede en la BBDD que estamos usando.

## 3.3 Llaves Foráneas

Una clave foránea es una columna o grupo de columnas de una tabla que contiene valores que coinciden con la clave primaria de otra tabla. Las claves foráneas se utilizan para unir tablas.

Generalmente las llaves foráneas se utilizan para construir referencias en tablas de paso (es decir, tablas que consolidan registros que relacionan 2 o más tablas).

```
1 create table tabla2 (campo1 tipo1, campo2 tipo2,
...campollave REFERENCES tabla1(campollave))
```

## 3.4 Consultas de Cruce

### 3.4.1 INNER JOIN

Esta expresión selecciona registros que tienen valores coincidentes en ambas tablas.

```
1 SELECT columna(s)
2 FROM tabla1
3 INNER JOIN tabla2
4 ON tabla1.columna_pivote = tabla2.columna_pivote
   ;
```

### 3.4.2 LEFT JOIN

Devuelve todos los registros de la tabla izquierda (tabla1) y los registros coincidentes de la tabla derecha (tabla2). El resultado es 0 registros del lado derecho, si no hay coincidencia.

```
1 SELECT columna(s)
2 FROM tabla1
3 LEFT JOIN tabla2
4 ON tabla1.columna_pivote= tabla2.columna_pivote;
```

### 3.4.3 RIGHT JOIN

Devuelve todos los registros de la tabla derecha (tabla2) y los registros coincidentes de la tabla izquierda (tabla1). El resultado es 0 registros del lado izquierdo, si no hay coincidencia.

```
1 SELECT columna(s)
2 FROM tabla1
3 RIGHT JOIN tabla2
4 ON tabla1.columna_pivote = tabla2.columna_pivote
   ;
```

### 3.4.4 FULL JOIN

Devuelve todos los registros cuando hay una coincidencia en los registros de la tabla izquierda (tabla1) o derecha (tabla2).

```
1 SELECT columna(s)
2 FROM tabla1
3 FULL JOIN tabla2
4 ON tabla1.columna_pivote = tabla2.columna_pivote
5 WHERE condicion
```

## 3.5 Unión de queries

Con el comando **UNION ALL** puedo unir el resultado de una o varias consultas a mismos campos. Observación: los campos se unen incluso si están duplicados.

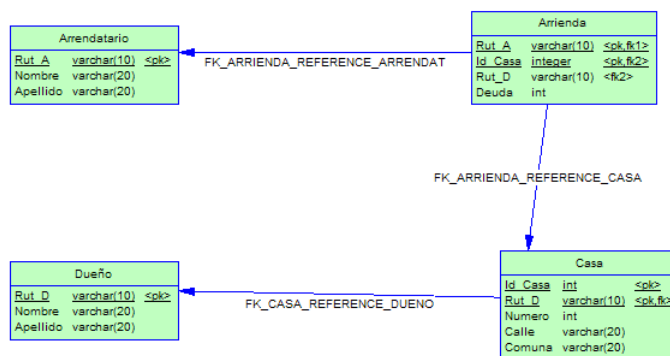
```
1 SELECT columna(s) FROM tabla1
2 UNION ALL
3 SELECT columna(s) FROM tabla2;
```

### 3.6 Manos a la obra: Ejercicio # 1

“Arriendo Seguro S.A” es una de las asociaciones encargadas de administrar el proceso de arrendar casas en Santiago. Para evitar las molestias de cobranzas para los propietarios ellos se encargan del cobro, por lo cual han diseñado el siguiente modelo de datos, con el fin de obtener toda la información necesaria y llevar el control del proceso.

1. Cree las tablas e ingrese datos acordes a cada una de ellas (ver anexos)
2. Qué Casas están en la comuna de Macul
3. Muestre cantidad adeudada por cada arrendatario
4. Muestre a los arrendatarios con deuda mayor a 100000
5. Cantidad de casas por comunas
6. Deudas por dueño
7. Ordenar 6) por deuda de mayor a menor

#### Modelo ERD



## Tabla Arrendatario:

Rut_A	Nombre	Apellido
11246890-4	Emilio	Gaete
12349840-4	Melissa	Torres
1243235-8	Liliana	Sotela
5345678-8	Marcos	Urrutia
5432345-6	Tamara	Romero
6789765-0	Francisco	Rodriguez
7987657-9	Carla	Matus
9654789-k	Sulema	Garrido

## Tabla Arrienda:

Rut_A	Id_Casa	Deuda
11246890-4	1	20000
11246890-4	9	145000
12349840-4	8	100000
1243235-8	5	320000
1243235-8	12	187000
5345678-8	3	123000
5432345-6	4	0
6789765-0	6	87000
7987657-9	7	0
9654789-k	2	34000
9654789-k	10	67000
9654789-k	11	0

## Tabla Casa:

Id_Casa	Rut_D	Numero	Calle	Comuna
1	13678567-9	2243	Las Torres	Macul
2	12567298-5	123	Guillermo Mann	Nunoa
3	11876984-2	5467	P.de Valdivia	Nunoa
4	8765432-1	7485	Los Olmos	Macul
5	8647299-k	876	Los Platanos	Quilicura
6	10234567-5	5546	Los Espinos	San Ramon
7	6783456-7	6657	Zafartu	Recoleta
8	7890987-3	4059	Los Alerces	Maipu
9	13678567-9	987	Av.Grecia	Macul
10	12567298-5	7657	Los Trucados	Nunoa
11	8765432-1	778	Almirante la Torre	Maipu
12	8647299-k	7854	Irarrazaval	Nunoa
13	6783456-7	4444	Marathon	Peñafior
14	7890987-3	3335	Manuel de Salas	Santiago

## Tabla Dueño:

Tabla Dueño:		
Rut_D	Nombre	Apellido
10234567-5	Leonardo	Opazo
11876984-2	Maria	Mercedes
12567298-5	Cristian	fuentes
13678567-9	Carlos	Gutty
6783456-7	Silvia	Hernandez
7890987-3	Eduardo	Lizama
8647299-k	Patricio	Rojas
8765432-1	Gloria	Sura

### 3.6.1 Observaciones

Comentar que al montar el esquema se debe hacer en **orden de dependencias** (de menor a mayor); por ejemplo la tabla casa depende de 1 tabla (ver diagrama ERD), mientras que arrienda depende de 2 tablas, y arrendatario/dueno dependen de 0 tablas. De esta manera, se crean arrendatario/dueno primero, luego casa y finalmente arrienda:

```

1  -- El orden SI IMPORTA!
2
3  CREATE TABLE Arrendatario (Rut_A varchar(10)
    PRIMARY KEY,
4  Nombre varchar(20),
5  Apellido varchar(20))
6
7  CREATE TABLE dueno (Rut_D varchar(10) PRIMARY
    KEY,
8  Nombre varchar(20),
9  Apellido varchar(20))
10
11 CREATE TABLE casa (Id_Casa int PRIMARY KEY,
12 Rut_D varchar(10) REFERENCES dueno(RUT_D),
13 Numero int,
14 Calle varchar(20),
15 Comuna varchar(20))
16
17 CREATE TABLE Arrienda (Rut_A varchar(10)
    REFERENCES Arrendatario(Rut_A),
18 Id_Casa int REFERENCES casa(Id_Casa),
19 Deuda int,
20 Rut_D varchar(10) REFERENCES dueno(Rut_D))

```

La ingesta de datos debe hacerse en el mismo orden.

## 4 Clase 4: Comenzando el nivel intermedio

### 4.1 Funciones Tabulares y Multisentencia

Como vimos en la sección anterior, una función puede tener un output escalar (valor único) o vectorial (entrega como resultado una tabla). A este último tipo se le conoce como **función tabular**.

La sintaxis para crear una función tabular es similar a lo revisado en clases anteriores, solo que como resultado se obtiene una tabla:

```
1 CREATE FUNCTION nombre_funcion (@param1 tipo1
  ..)
2 RETURNS TABLE
3 AS
4 RETURN
5 (...codigo para generar tabla...)
```

Además, podemos crear una tabla que sea definida en una función, y que por ende usará más comandos que un SELECT, sino también INSERT, DELETE, UPDATE, etc. A este tipo de bloques se les denomina **funciones multisentencia**:

```
1 CREATE FUNCTION nombre_funcion (@param1 tipo1
  ....)
2 RETURNS @tabla_retorna table(campos a retornar )
3 AS BEGIN
4     INSERT INTO @tabla_retorna
5     SELECT ...
6     RETURN
7 END
```



## 4.2 Triggers

Un Trigger o desencadenador es un tipo de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento de Lenguaje de Manipulación de Datos (DML), o sea cuando se ejecuta un INSERT, UPDATE o DELETE en una tabla o un evento de Lenguaje de Definición de Datos (DDL) por ejemplo un DROP TABLE.

Un gatillo se activa cuando ocurre un evento, por ende es condicionado a que el evento ocurra. El gatillo se puede activarse antes o después de que el evento ocurra, aunque para nivel intermedio de SQL consideramos gatillos con activación posterior a la ocurrencia del evento.

Para crear un Trigger, usar la siguiente sintaxis:

```
1 CREATE TRIGGER nombre_trigger
2 ON tabla_inicial FOR tipo_evento
3 AS
4 BEGIN
5 ...
6 END
```

En este caso el tipo de evento puede ser un DML o DDL, para los DML seria INSERT/UPDATE/DELETE. El evento que activa el gatillo entregará un objeto inserted/updated/deleted que corresponderá a una tabla resultante o que recibe la acción: por ejemplo, si quiero gatillar una acción a partir de una inserción realizada, puedo obtener sus características llamando al objeto inserted.

Los triggers pueden ser modificados con ALTER TRIGGER y eliminados con DROP TRIGGER.

## 4.3 Ejecución y Programación en SQL

Una ventaja importante de SQL es que no sólo es un lenguaje de consulta, sino también un lenguaje de programación: podemos crear programas que se ejecutan línea por línea, pueden iterar, agregar condicionales, etc.

En SQL Server podemos imponer condiciones sobre la ejecución de una instrucción SQL. La instrucción SQL que sigue a una palabra clave IF y su condición se ejecuta si se cumple la condición. La palabra clave ELSE opcional introduce otra instrucción SQL que se ejecuta cuando no se cumple la condición IF:

```
1 IF <condicion>
2     BEGIN
3         Ejecucion ...
4     END
5 ELSE
6     BEGIN
7         Ejecucion ...
8     END
```

Así mismo podemos almacenar conjuntos de instrucciones y ejecutarlas cuando queramos. Esto se puede hacer por medio de un **Procedimiento Almacenado**: si tiene una consulta SQL que escribe una y otra vez o tareas de DML, guárdela como un procedimiento almacenado y luego llámela para ejecutarla.

También se pueden pasar parámetros a un procedimiento almacenado, como una función (y recalamos "como", ya que un procedimiento no exige retornar valores):

```
1  --sintaxis
2  CREATE PROCEDURE nombre_procedimiento (@param1
    tipo1 ,
3  @param2 tipo2 ...)
4  AS
5  BEGIN
6  ...
7  END
8  --ejecute el procedimiento
9  EXEC nombre_procedimiento
```

## 4.4 Cursores

Una introducción a los objetos iterativos en programación en SQL son los cursores. El cursor es un objeto de base de datos para recuperar datos de un conjunto de resultados una fila a la vez, en lugar de los comandos vistos hasta ahora que operan en todas las filas del conjunto de resultados a la vez. En consecuencia, **el cursor es el unico objeto que recorre una consulta línea por línea.**

```

1 DECLARE nombre_cursor CURSOR
2 FOR especificacion_consulta
3  --activar_cursor
4 OPEN nombre_cursor
5  --recorrer fila por fila
6 FETCH NEXT FROM nombre:cursor INTO
   variables_cursor
7 WHILE @@FETCH_STATUS = 0
8     BEGIN
9         ...
10        FETCH NEXT FROM nombre:cursor INTO
           variables_cursor
11    END
12 CLOSE nombre_cursor  --cierra cursor
13 DEALLOCATE nombre_cursor  -- desposicionar

```

## 4.5 Ingesta de datos desde código

Una carga de archivo plano con separadores puede hacerse usando el asistente (en SSMS) pero también por medio de instrucciones en SQL. Para ello utilizamos el comando BULK INSERT

```

1 BULK INSERT tabla_receptora
2 FROM 'ruta_archivo'
3 WITH (FIELDTERMINATOR='sep_columnas',
4 ROWTERMINATOR='sep_filas',
5 FIRSTROW='fila_de_partida')

```

Algunos detalles importantes de ésta instrucción:

- Generalmente el FIELDTERMINATOR son comas, mientras que el ROWTERMINATOR es un salto de línea (representado por \n).

- `BULK INSERT` como instrucción carga los datos en el formato que están en el archivo de origen; por ende, debemos preocuparnos de que los datos vengan en el formato correcto y consistente con la tabla donde se hará la ingesta. Si los datos no vienen en un formato consistente con la tabla, pueden en primer lugar cargarse en una tabla temporal como texto (sólo campos `varchar(max)`) y luego utilizar funciones de conversión (tema a revisar sección 5)

## 5 Clase 5: Funciones específicas

### 5.1 Comandos de conversión e IIF

La función CONVERT () convierte un valor (de cualquier tipo) en un tipo de datos específico.

```
1 CONVERT(tipodato , dato)
2 -- sintaxis alternativa
3 CAST(valor AS tipo_datos)
```

Además de los comandos case-when, existe una sintaxis simple para un if dentro de un campo determinado, para esto se usa el comando IIF

```
1 IF(condicion , valor_if_true , valor_if_false)
```

Una forma astuta de reemplazar el siguiente case-when

```
1 case when var=x then 'a'
2 when var=y then 'b'
3 else 'c'
4 end
```

Es la siguiente:

```
1 iif(var=x, 'a', iif(var=y, 'b', 'c'))
```

Es decir podemos anidar para aprovechar la sintaxis resumida de la instrucción iif.

### 5.2 Funciones de Texto/Fecha

Existen una gran cantidad de funciones para operar con texto en SQL, que están ampliamente documentadas. Aquí mencionaremos las más importantes:

- `CHARINDEX()`: La función `CHARINDEX()` busca una subcadena en una cadena y devuelve la posición `CHARINDEX(substring, string, start)`
- `CONCAT`: La función `CONCAT()` agrega dos o más cadenas juntas. `CONCAT(string1, string2, ....., string_n)`
- `LEFT/RIGHT`: La función `LEFT ()/RIGHT ()` extrae varios caracteres de una cadena (empezando por la izquierda/derecha). `LEFT/RIGHT(string, no_caracteres)`
- `LTRIM/RTRIM` elimina los espacios iniciales de una cadena. `LTRIM/RTRIM(string)`
- `REPLACE`: reemplaza todas las apariciones de una subcadena dentro de una cadena, con una nueva subcadena. `REPLACE(string, vieja_subcadena, nueva_subcadena)`
- `STUFF`: elimina una parte de una cadena y luego inserta otra parte en la cadena, comenzando en una posición especificada. `STUFF(string, partida, largo, nuevo_string)`
- `LOWER/UPPER` : lleva todo a minúsculas o mayúsculas respectivamente
- `SUBSTRING`: La función extrae algunos caracteres de una cadena. `SUBSTRING(string, partida, largo)`

Análogamente, entre las funciones de fecha a destacar tenemos:

- `CURRENT_TIMESTAMP` entrega la fecha y hora actual
- `DATEDIFF(unidad, fecha1, fecha2)` entrega la diferencia entre 2 fechas en la unidad especificada (por ejemplo 'd' días, 'm' meses, 'yy' años, 'ww' semanas, etc.)

- DATEADD(unidad, cantidad, fecha) agrega a la fecha una cantidad de unidades de fecha (análogo a anterior)
- EOMONTH(fecha, meses\_agregar), entrega la fecha de fin de mes para una cantidad de meses atras (-) o adelante (+); si quiere el fin del mes actual, meses\_agregar=0
- DAY(), MONTH(), YEAR() obtiene día, mes, y año de una fecha en particular.
- ISDATE() permite verificar si un texto es fecha o no.
- @@DATEFIRST es una constante del sistema que define cual es el primer día de la semana.
- DATEPART(unidad, fecha) entrega una parte de la fecha según definición. Por ejemplo para saber qué día de la semana es, colocar 'dw'

## 5.3 Instrucciones Dinámicas

Se puede ejecutar una instruccion SQL a partir de una cadena de texto que esté parametrizada

```
1 EXEC ('Instruccion como cadena de texto')
```

NOTA: esto habilita para crear instrucciones dinámicas que puedan usar funciones de texto.

## 5.4 Consultando al esquema

Lo anterior puede usarse para hacer subrutinas que recorran toda la BBDD. Para obtener información de la BBDD que estamos usando, podemos consultar al INFORMATION\_SCHEMA



```

1  --obtiene tablas
2  SELECT * FROM nombre_bbdd.INFORMATION_SCHEMA .
    TABLES
3  --obtiene tablas y sus columnas
4  SELECT * FROM nombre_bbdd.INFORMATION_SCHEMA .
    COLUMNS
5  --Contar numero de tablas en bbdd
6  SELECT COUNT(TABLE_NAME) FROM nombre_bbdd .
    INFORMATION_SCHEMA.TABLES

```

## 5.5 Tablas dinámicas

Como sabemos, el instrumento de excelencia para operar con tablas dinámicas es MsExcel. Sin embargo, cuando los volúmenes de datos superan el orden de  $10^7$  registros, una tabla dinámica en excel comenzará a ponerse lenta e inmanejable. Para esto existe la función PIVOT que emula las funciones de excel mencionadas:

```

1  SELECT    <columna_no_pivote>,
2            <lista_columnas_a_pivoteear>
3  FROM
4  (<SELECT query que produce datos>)
5  AS <nombre_alias>
6  PIVOT
7  (
8  <funcion de agregado>(<columna agregado>)
9  FOR
10 [<campo desde el cual sale
    lista_columnas_a_pivoteear>]
11 IN ( [ <lista_columnas_a_pivoteear> ] )
12 ) AS <alias_pivot>

```

## 6 Ejercicio Final

El archivo `series_precios.csv` contiene los precios diarios de 5 índices ampliamente utilizados en el mercado financiero. En el se incluyen además los días no hábiles, en cuyo caso se repite el precio de cierre del último día hábil disponible. Respecto a este archivo:

1. Construya una tabla temporal `#fechas` que tenga la fecha de fin de mes anterior (fecha inicio) y la fecha de fin de mes (fecha término) para todos los meses que dispone la data.
2. Construya una consulta que entregue el retorno mensual de una de las series (por ejemplo, `serie1`), donde sus campos sean la fecha de término del mes y el retorno, el cual calcularemos usando la expresión  $Retorno = 100 * \left( \frac{PrecioFinal - PrecioInicial}{PrecioInicial} \right)$
3. Utilice la consulta realizada en (3) para construir una consulta dinámica donde al cambiar el nombre de la serie, se obtenga una tabla de retornos.
4. Elabore una consulta que obtenga el máximo retorno para una serie, y en qué mes se da.
5. Elabore una consulta para obtener el segundo mayor retorno (sólo el retorno, sin fecha)
6. Para la `serie1` de precios, elaborar un *flag*(o campo binario) donde valga 1 si el día es hábil y 0 si no
7. Calcular el promedio para los días hábiles (sólo serie de precios)
8. Elabore un control que permita detectar si hay precios duplicados.