

## Los tipos de datos SQL

Se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos. Los tipos de datos primarios son:

Tipo de Datos	Longitud	Descripción
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos Binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de - 3.402823*10 <sup>38</sup> a -1.401298*10 <sup>-45</sup> para valores negativos, 1.401298*10 <sup>-45</sup> a 3.402823*10 <sup>38</sup> para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de - 1.79769313486232*10 <sup>308</sup> a -4.94065645841247*10 <sup>-324</sup> para valores negativos, 4.94065645841247*10 <sup>-324</sup> a 1.79769313486232*10 <sup>308</sup> para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.z
TEXT	1 byte por carácter	De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos de los tipos de datos definidos:

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT

CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEE SINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEE DOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR
VARIANT (No Admitido)	VALUE

## **SQL - DML(I)**

### **Instrucciones para manipular datos en una tabla**

#### **1. Agregar datos**

##### **Insertar una fila en una tabla**

```
INSERT INTO tabla(lista_de_columnas)  
VALUES(lista_de_valores);
```

##### **Insertar múltiples líneas en una tabla**

```
INSERT INTO tabla(lista_de_columnas)  
VALUES  
(lista_de_valores),  
(lista_de_valores)...;
```

#### **2. Modificar datos**

##### **Actualizando datos de un registros de toda una columna**

```
UPDATE tabla SET c1 = nuevo_valor
```

##### **Actualizar los datos de la columna c1 y c2 que cumplan con una condición**

```
UPDATE tabla  
SET  
c1 = nuevo_valor,  
c2 = nuevo_valor WHERE condición;
```

#### **3. Eliminar datos**

##### **Eliminar TODOS los datos de una tabla**

```
DELETE FROM tabla;
```

##### **Borrar una serie de datos**

```
DELETE FROM tabla WHERE condición;
```

## SQL - DML(II)

### 4. Búsquedas en la base de datos

**Obtener los datos de las columnas c1 y c2 de una tabla**

```
SELECT c1, c2 FROM tabla;
```

**Obtener todas las filas y columnas de una tabla**

```
SELECT * FROM tabla;
```

**Obtener datos y filtrar filas con una condición**

```
SELECT c1, c2 FROM tabla WHERE condición [AND/OR condición2];
```

**Obtener filas distintas de una tabla** (que no se repitan los datos)

```
SELECT DISTINCT c1 FROM tabla WHERE condición [AND/OR condición2];
```

**Ordenar los resultados de forma ascendente o descendente**

```
SELECT c1, c2 FROM tabla ORDER BY c1 ASC [DESC];
```

**Obtener los primeros 10 registros de una tabla con la columna c1 ordenada en forma descendente**

```
SELECT c1, c2 FROM tabla ORDER BY c1 DESC LIMIT 10;
```

**Ejemplo de búsquedas en múltiples tablas.** Complementar con el apunte de JOIN.

```
SELECT c1, c2 FROM tabla1 INNER JOIN tabla2 ON condición;
```

```
SELECT c1, c2 FROM tabla1 LEFT JOIN tabla2 ON condición;
```

```
SELECT c1, c2 FROM tabla1 RIGHT JOIN tabla2 ON condición;
```

**Obtener todas las columnas de los registros cuya columna *variable1* tenga un valor entre *nro1* y *nro2***

```
SELECT * FROM tabla WHERE variable1 BETWEEN nro1 AND nro2;
```

**Obtener todas las columnas de los registros cuya columna *variable1* contenga la cadena *una\_cadena***

```
SELECT * FROM tabla WHERE variable1 [NOT] LIKE una_cadena;
```

**Obtener todas las columnas de los registros cuya columna *variable1* contenga una cadena que contiene la letra 'm'**

```
SELECT * FROM tabla WHERE variable1 LIKE '%m%';
```

**Obtener todas las columnas de los registros cuya columna *variable1* contenga una cadena que termina con 's'**

```
SELECT * FROM tabla WHERE variable1 LIKE '%s';
```

**Obtener todas las columnas de los registros cuya columna *variable1* contenga una cadena que comienza con 'A'**

```
SELECT * FROM tabla WHERE variable1 LIKE 'A%';
```

## SQL - Alias

Los alias se utilizan en SQL para asignarle un nombre temporal a una columna o a una tabla.

Sirven para abreviar los nombres en sentencias muy extensas y para que el código sea más legible.

El alias se elimina en cuanto termina la consulta.

Para crear un alias, utilizamos la palabra clave **AS**.

### Sintaxis:

```
SELECT nombre_columna AS alias_col  
FROM table_name
```

```
SELECT nombre_columna  
FROM table_name AS alias_tabla
```

### Ejemplo:

TABLA CLIENTES		
cliente_nro	nombre	apellido
1	Sally	Thompson
2	Sally	Henderson
3	Harry	Henderson
4	Sandra	Wellington

TABLA PEDIDOS				
nro_orden	empleado_nro	cliente_nro	proveedor	precio
1	1	42	Harrison	\$235,00
2	4	1	Ford	\$234,00
3	1	68	Jason	\$415,00
4	2	112	Statham	\$350,00
5	3	42	Dwayne	\$235,00
6	2	112	Johnson	\$350,00
7	2	42	Diesel	\$235,00

```
SELECT p.nro_orden, p.proveedor, c.nombre, c.apellido  
FROM tabla_clientes AS c, tabla_pedidos AS p  
WHERE c.nombre = 'Sally'  
AND c.cliente_nro = p.cliente_nro
```

### Sin alias:

```
SELECT tabla_pedidos.nro_orden, tabla_pedidos.proveedor, tabla_clientes.nombre,  
tabla_clientes.apellido  
FROM tabla_clientes, tabla_pedidos  
WHERE tabla_clientes.nombre = 'Sally'  
AND tabla_clientes.cliente_nro = tabla_pedidos .cliente_nro
```

## Funciones en SQL

### 1. COUNT()

Devuelve el número de filas que coincide con un criterio.

#### Sintaxis:

```
SELECT COUNT(nombre_columna)
FROM nombre_tabla
WHERE condición;
```

### 2. AVG()

Devuelve el valor promedio de una columna con valores numéricos que cumplan la condición.

#### Sintaxis:

```
SELECT AVG(nombre_columna)
FROM nombre_tabla
WHERE condición;
```

### 3. SUM()

Devuelve el valor de la suma de una columna, siempre y cuando cumplan con la condición.

#### Sintaxis:

```
SELECT SUM(nombre_columna)
FROM nombre_tabla
WHERE condición;
```

### Ejemplo:

TABLA_CLIENTES		
cliente_nro	nombre	apellido
1	Sally	Thompson
2	Sally	Henderson
3	Harry	Henderson
4	Sandra	Wellington

TABLA_PEDIDOS				
nro_orden	empleado_nro	cliente_nro	proveedor	precio
1	1	42	Harrison	\$235,00
2	4	1	Ford	\$234,00
3	1	68	Jason	\$415,00
4	2	112	Statham	\$350,00
5	3	42	Dwayne	\$235,00
6	2	112	Johnson	\$350,00
7	2	42	Diesel	\$235,00

```
SELECT COUNT(nombre)
```

```
FROM tabla_clientes
```

```
WHERE nombre = 'Sally';
```

```
SELECT AVG(precio)
```

```
FROM tabla_pedidos
WHERE cliente_nro = 112;

SELECT SUM(precio)
FROM tabla_pedidos
WHERE empleado_nro = 2;
```

#### 4. Otras funciones y operaciones

**MIN()** / **MAX()** devuelven el menor y mayor valor de una columna.

Además, pueden “agregarse” provisoriamente columnas calculadas a partir de las existentes.

Si quisiera, por ejemplo, agregar una columna a la tabla de pedidos, donde me mostrara el IVA y el precio final:

```
SELECT precio, precio*0.21, precio + (precio*0.21)
FROM tabla_pedidos
```

El resultado de la búsqueda me devolverá una tabla de tres columnas: precio, iva y precio con iva.

La estructura de la tabla no se ve afectada. Como los alias, estas columnas se extinguen al terminar la consulta.

Podemos ponerle nombre a estas “nuevas columnas”, utilizando alias. En el ejemplo anterior:

```
SELECT precio, precio*0.21 AS IVA, precio + (precio*0.21)
AS Total FROM tabla_pedidos;
```