

Curso AABD-SQL

Notas del curso

Con F5 se corren las líneas de código

Los comentarios se hacen con --

CREATE DATABASE

CREATE DATABASE [nombre] ;

+ Se crea una data base con el nombre asignado

CREATE TABLE

```
CREATE TABLE [nombre] (  
  [nombre columna 1] [tipo de datos] [condiciones]  
  ...  
  [nombre columna n] [tipo de datos] [condiciones] );
```

```
CREATE TABLE tabla_clientes(  
  id_cliente int,  
  nombre varchar,  
  apellido varchar,  
  edad varchar,  
  correo varchar  
);
```

```
CREATE TABLE users(id serial primary key, name carácter  
varying);
```

+ Se crea una tabla con n columnas con los nombres y tipos de datos. Las condiciones son opcionales.

+ Las tablas creadas se ven en
Schemas > Tablas (dentro del database)

Tipo de datos: int, char, char(n), varchar

Condiciones: NOT NULL, UNIQUE, PRIMARY KEY

Con esto logramos que ponga los números consecutivos

INSERT INTO - VALUES

```
INSERT INTO [nombre de la tabla] (  
[nombre columna 1] ,[nombre columna 2], ... )  
VALUES ( [valor1], [valor2], ... ) ,  
VALUES ( [valor1], [valor2], ... ) ;
```

```
INSERT INTO tabla_clientes  
VALUES ( 1,'Cintia', 'Cee' 32, 'ab@xyz.com' );
```

```
INSERT INTO tabla_clientes ( id_cliente, nombre, correo, edad )  
VALUES ( 2, 'Diana', 'd@xyz.com', 32 );
```

+ Dada una tabla creada, se pueden insertar valores de forma “manual” a una tabla con esta sentencia

+ Dentro del primer paréntesis, se **puede** especificar el orden en que se desea guardar los valores que aparecen en el segundo paréntesis

+ Se agregan VALUES como si cada uno fuera un vector

COPY - FROM

COPY [nombre de la tabla] (
[nombre columna 1] , [nombre columna 2], ...)
FROM 'dirección y nombre.csv' **DELIMITER** ','
CSV HEADER;

COPY tabla_clientes (id_cliente, nombre, apellido, edad, correo)
FROM ' C:\Users\Public\Documents\aabd_sql_20121\copy.csv'
DELIMITER ',' **CSV HEADER;**

+ Es una forma de hacer un importado de una tabla en formato .txt o .csv a SQL

+ El archivo que contiene la tabla debe estar en la parte pública de la computadora.

+ Si la tabla ya fue creada con los nombres de las columnas en CREATE TABLE, se puede evitar el primer paréntesis

SELECT

SELECT [nombre alguna columna] , [nombre alguna columna] ,
...

FROM [nombre de la tabla] ;

SELECT nombre, apellido **FROM** tabla_clientes;

+ Muestra los **registros** de las columnas seleccionadas.

+ Poniendo * se puede mostrar la tabla completa.

SELECT DISTINCT

SELECT DISTINCT [nombre alguna columna] , [nombre alguna columna] , ...

FROM [nombre de la tabla] ;

SELECT DISTINCT nombre **FROM** tabla_clientes;

+ Muestra los **registros únicos** de las columnas seleccionadas, es decir, valores sin duplicados.

+ Los datos no los muestra por orden ni frecuencia, esto tiene que ver con la normalización de las bases de datos.

WHERE

SELECT [nombre de la columna]
FROM [nombre de la tabla]
WHERE [condición] ;

SELECT nombre, apellido **FROM** tabla_clientes
WHERE edad>25 ;

SELECT ***FROM** tabla_clientes
WHERE nombre='Gabriela' ;

+ Permite seleccionar aquellas *subtablas* donde se cumpla alguna condición

+ Para poner más de una condición se deben usar operadores lógicos

OR, AND, NOT

SELECT [nombre de la columna]
FROM [nombre de la tabla]
WHERE [condición con OR, AND o NOT] ;

SELECT nombre, apellido **FROM** tabla_clientes
WHERE edad>20 **AND** edad<30 ;

SELECT ***FROM** tabla_clientes
WHERE (edad<=25 **OR** edad>30) **AND** nombre='Gabriela' ;

SELECT ***FROM** tabla_clientes
WHERE apellido **IS NULL**;

+ Estos operadores lógicos se pueden combinar con **WHERE** para realizar múltiples filtros.

+ Para una desigualdad se puede usar "<>", "!=" o un **NOT**

+ Las proposiciones (AUB)∩C se escribe como (A OR B) AND C

+ Esto es para seleccionar registros donde cierta columna tenga valores nulos.

UPDATE - SET

UPDATE [nombre de la tabla]
SET [nombre de la columna] = [valor],
[nombre de la columna] = [valor]
WHERE [condición] ;

UPDATE tabla_clientes
SET apellido='Perez', edad=17 **WHERE** id_cliente=2 ;

UPDATE tabla_clientes **SET** correo='gee@xyz.com' ;

+ Es útil para cambiar el valor de algunas columnas (campos) para un registro en específico (o conjunto de ellos que cumplan una condición)

+ Las filas actualizadas se mandan hasta abajo de la tabla

+ Estamos cambiando dos valores de un mismo registro

+ Actualizamos todos los valores de una columna

DELETE FROM - WHERE

DELETE FROM [nombre de la tabla]
WHERE [condición] ;

DELETE FROM tabla_clientes
WHERE id_cliente=6 ;

DELETE FROM tabla_clientes ;

+ Esta sentencia se usa para eliminar registros de una tabla

+ Si no se especifica el **WHERE** entonces se **vaciará la tabla completa**

+ Borraremos todos los registros de la tabla

ALTER TABLE

ALTER TABLE [nombre de la tabla]
[especificar acciones]

El conjunto de acciones que se pueden hacer es

1. Respecto a *columnas*

ADD COLUMN [nombre de la columna] [tipo de datos] ;

DROP COLUMN [nombre de la columna] ;

ALTER COLUMN [nombre de la columna] **TYPE** [nuevo tipo de datos] ;

RENAME COLUMN [nombre de la columna] **TO** [nuevo nombre] ;

+ Esta sentencia nos permite realizar diversas acciones con las columnas o con las condiciones que estas tienen

ALTER TABLE

ALTER TABLE tabla_clientes **ADD COLUMN** prueba varchar;

ALTER TABLE tabla_clientes **ALTER COLUMN** edad **TYPE** varchar;

ALTER TABLE tabla_clientes **RENAME COLUMN** correo **TO** correo_cliente;

+ Esta sentencia nos permite realizar diversas acciones con las columnas o con las condiciones que estas tienen

ALTER TABLE

ALTER TABLE [nombre de la tabla]
[especificar acciones]

El conjunto de acciones que se pueden hacer es

2. Respecto a las *condiciones*

ALTER COLUMN [nombre de la columna] **SET** NOT NULL ;

ALTER COLUMN [nombre de la columna] **DROP** NOT NULL ;

ADD CONSTRAINT [nombre de la columna] **CHECK** [condición] ;

ADD PRIMARY KEY [nombre de la columna];

+ Esta sentencia nos permite realizar diversas acciones con las columnas o con las condiciones que estas tienen

ALTER TABLE

```
ALTER TABLE tabla_clientes ALTER COLUMN id_cliente SET NOT NULL;
```

```
ALTER TABLE tabla_clientes ADD CONSTRAINT id_cliente CHECK (id_cliente>0);
```

```
ALTER TABLE tabla_clientes ADD PRIMARY KEY (id_cliente);
```

+ Esta sentencia nos permite realizar diversas acciones con las columnas o con las condiciones que estas tienen

IN

SELECT [nombre de la columna]
FROM [nombre de la tabla]
WHERE [nombre de una columna] **IN** ('Valor1', 'Valor2', ...)

SELECT * **FROM** customer **WHERE** city **IN** ('Philadelphia', 'Seattle');

SELECT * **FROM** customer
WHERE city **IN** ('Philadelphia', 'Seattle') **AND** segment **IN** ('Corporate');

+ Esta sentencia nos ayuda a reducir el uso del **OR**. Es compatible con **SELECT**, **INSERT**, **UPDATE** o **DELETE**

BETWEEN

SELECT [nombre de la columna]
FROM [nombre de la tabla]
WHERE [nombre de una columna]
BETWEEN 'Valor1' **AND** 'Valor2' ;

SELECT * **FROM** customer **WHERE** age **BETWEEN** 20 **AND** 30;

SELECT * **FROM** sales
WHERE ship_date **BETWEEN** '2015-04-01' **AND** '2016-04-01' ;

+ Es útil para filtrar valores usando un rango. Es compatible con **SELECT**, **INSERT**, **UPDATE** o **DELETE**.

+ Toma en cuenta los extremos, es decir, es un “menor o igual” y un “mayor o igual”

+ Esta sentencia se puede hacer con fechas, pero las fechas se deben poner entre comillas.

LIKE - ILIKE

SELECT [nombre de la columna]
FROM [nombre de la tabla]
WHERE [nombre de una columna] **LIKE** [patrón]

A% : Empiece con A y después haya cualquier cosa

%A: Termine con A y antes haya cualquier cosa

A%B: Empiece con A, termine con B y en medio haya cualquier cosa

Cuando ponemos “_” es que permitimos que haya UNA única cosa

SELECT city **FROM** customer **WHERE** city **NOT LIKE** ‘S%’ ;

SELECT * **FROM** customer **WHERE** customer_name **LIKE** ‘____ %’ ;

+ Es usada para filtrar mediante valores de coincidencia por patrones haciendo uso de *comodines*. Es compatible con **SELECT**, **INSERT**, **UPDATE** o **DELETE**.

+ Los *comodines* son “%” y “_”

+ Para hacer búsqueda con % se debe emplear “/”

+ **LIKE** hace distinción entre mayúsculas y minúsculas, si quisiéramos omitir eso podríamos usar **ILIKE**

+ Estamos pidiendo nombres de cuatro caracteres y que después haya lo que sea

ORDER BY

SELECT [nombre de la columna]
FROM [nombre de la tabla]
(**WHERE** si fuera necesaria)
ORDER BY [alguna columna] [**ASC, DESC**] [alguna columna], ...

```
SELECT * FROM customer  
WHERE age>25 ORDER BY city ASC, customer_name DESC;
```

```
SELECT * FROM customer  
WHERE state='California' ORDER BY customer_name ;
```

```
SELECT * FROM customer ORDER BY age ;
```

+ Es utilizada para ordenar los registros de un conjunto de resultados. Solo se puede usar con **SELECT**.

+ Si no se especifica, el valor por default es **ASC**. En caso de empate, el criterio de desempate lo determina la segunda columna escrita en la instrucción **ORDER BY**

LIMIT

SELECT [nombre de la columna]
FROM [nombre de la tabla]
(**WHERE** en caso de necesitarlo)
(**ORDER BY** en caso de necesitarlo)
LIMIT [número de registros a mostrar]

```
SELECT * FROM customer
WHERE age>=25
ORDER BY age DESC
LIMIT 8 ;
```

+ Nos sirve para visualizar una pequeña parte de la tabla.

AS

SELECT [nombre de la columna] **AS** [alias de la columna], ...
FROM [nombre de la tabla]
(**WHERE**, **ORDER BY**, **LIMIT**, ...)

SELECT customer_id **AS** “Num de cliente”, customer_name **AS** nombre,
age **AS** “Edad cliente”
FROM customer
ORDER BY nombre;

CREATE TABLE [nombre de la tabla]
AS
[Query]

+ Es útil para dar un nombre provisional a una columna o a una tabla.

+ Las comillas dobles son usadas para colocar espacios en los alias, pero al realizar consultas puede lanzar error

+ Con este código se crea una tabla con la query que se escriba justo después del **AS**.

COUNT, SUM, AVG, MIN, MAX

```
SELECT FUNCIONAGG( [nombre de la columna] )  
FROM [nombre de la tabla]  
( WHERE, ORDER BY, LIMIT, ... )
```

```
SELECT COUNT(*) FROM sales ;
```

```
SELECT COUNT( DISTINCT order_id ) AS “Número de órdenes distintas”  
FROM sales;
```

```
SELECT MIN( sales ) AS “Mínimo de ventas en junio”  
FROM sales WHERE order_date BETWEEN ‘2015-06-01’ AND ‘2015-06-30’ ;
```

+ Se denominan *funciones agregadas* porque funcionan junto con el **SELECT**.

+ Cuenta todos los registros que tiene la tabla *sales*

+ Cuenta todos los registros diferentes entre sí de la columna *order_id*. Pensar que primero se pone **SELECT DISTINCT** *order_id* y luego se hace un **COUNT()**

+ Notar el uso de comilas para hacer un **BETWEEN** con fechas

GROUP BY

```
SELECT [nombre de la columna]  
FUNCIONAGG( [nombre de otra columna] ), ...  
FROM [nombre de la tabla]  
( WHERE en caso de ser necesario)  
GROUP BY [nombre de la columna] ;
```

```
SELECT region AS "Región", COUNT(customer_id) AS "Total de clientes"  
FROM customer GROUP BY region ;
```

```
SELECT region AS "Región", state AS "Estado",  
COUNT(customer_id) AS "Total de clientes", AVG(age) AS "Edad promedio"  
FROM customer GROUP BY region state;
```

+ Se utiliza junto con **SELECT** para agrupar el conjunto de resultados, se pueden agrupar por una o más columnas

+ Es importante notar que la misma columna que se escribe en **GROUP BY** se tiene que escribir en el **SELECT**

+ Esta sentencia hará que se muestren los registros agrupados por los distintos **valores categóricos** que pueda tener la columna en rojo.

+ Es como si los valores categóricos de la columna definieran una partición

+ Se puede realizar el **GROUP BY** usando dos columnas, pero ambas se deben agregar al **SELECT**

	Región character varying 🔒	Total de clientes bigint 🔒
1	South	134
2	West	255
3	East	220
4	Central	184

+ Agrupa las cuatro regiones y muestra el resultado de realizar **COUNT(customer_id)**

Este ejemplo no es tan intuitivo o no ayuda mucho a clarificar lo que ocurre

	Región character varying 🔒	Estado character varying 🔒	Número de clientes bigint 🔒	Promedio de edad numeric 🔒
1	East	New York	87	45.0459770114942529
2	Central	Texas	77	46.1558441558441558
3	West	Colorado	20	47.9500000000000000
4	South	Virginia	15	50.9333333333333333
5	Central	Missouri	6	44.5000000000000000

+ Muestra los 41 estados, además muestra a qué región pertenecen y muestra las columnas correspondientes a **COUNT(customer_id)** y **AVG(age)**

HAVING

```
SELECT [nombre de la columna]
FUNCIONAGG( [nombre de otra columna] ), ...
FROM [nombre de la tabla]
( WHERE en caso de ser necesario)
GROUP BY [nombre de la columna]
HAVING [condición];
```



```
SELECT region, COUNT(customer_id) AS "Total de clientes"
FROM customer
WHERE customer_name LIKE 'A%'
GROUP BY region
HAVING COUNT( customer_id ) BETWEEN 15 AND 20 ;
```

+ Se utiliza en combinación con **GROUP BY** para restringir los registros, se mostrará aquellos que cumplan la condición del **HAVING**



+ La sentencia **HAVING** se aplica sobre la **función agregada**.

+ Primero va a regresar aquellos registros que cumplan la condición del **WHERE** y sobre ellos se aplicará el **HAVING** y este último hace referencia a la función de agregación.

Sin el HAVING

	Región character varying 	Total de clientes bigint 
1	South	13
2	West	18
3	East	22
4	Central	11

Con el HAVING

	Región character varying 	Total de clientes bigint 
1	West	18

CASE

CASE

WHEN [condición] **THEN** [resultado]

WHEN [condición] **THEN** [resultado]

ELSE [resultado]

END

SELECT *,

CASE

WHEN age<30 **THEN** 'Joven'

WHEN age>60 **THEN** 'Mayor'

ELSE 'Medio'

END AS 'Categoría de edad' **FROM** customer ;

+ Se usa como expresión condicional, funciona tal como el if-else

+ En aquellos casos donde no se tenga una instrucción para realizar, terminarán cayendo en el **ELSE**

+ Muestra la tabla completa y añade una columna más con el nombre de *Categoría de edad* , donde se hace la clasificación de acuerdo a la edad

+ Notar que la sentencia **CASE** se emplea para crear una columna, por lo que se le puede asignar un alias con la sentencia **AS**

INNER JOIN

```
SELECT [columnas]  
FROM [tabla1]  
INNER JOIN [tabla2]  
ON tabla1.columnajoin=tabla2.columnajoin
```

```
SELECT  
a.order_line, a.producto_id, a.customer_id, a.sales,  
b.customer_name, b.age  
FROM sales_2015 AS a  
INNER JOIN customer_20_60 AS b  
ON a.customer_id=b.customer_id  
ORDER BY customer_id;
```

+ Hace una intersección de ambas tablas mediante la **columna join**

+ Se usa un “.” para acceder al nombre de las columnas de cada tabla.

+ Los alias a cada tabla se asignan en la misma query

col1A	col2A	col3A
A	dato A	o
B	dato B	p
C	dato C	q
D	dato D	r

col1B	col2B
C	s
E	t
F	u

=

tabla 1			tabla 2	
col1A	col2A	col3A	col1B	col2B
C	dato C	q	C	s

LEFT JOIN

```
SELECT [columnas]
FROM [tabla1]
LEFT JOIN [tabla2]
ON tabla1.columnajoin=tabla2.columnajoin
```

+ Regresa los registros de la **tabla de la izquierda** aun cuando no exista coincidencia con los de la segunda tabla (cuyo caso asignará valores nulos)

col1A	col2A	col3A
A	dato A	o
B	dato B	p
C	dato C	q
D	dato D	r

col1B	col2B
C	s
E	t
F	u

=

col1A	col2A	col3A	col1B	col2B
A	dato A	o	NULL	NULL
B	dato B	p	NULL	NULL
C	dato C	q	C	s
D	dato D	r	NULL	NULL

RIGHT JOIN

```
SELECT [columnas]
FROM [tabla1]
RIGHT JOIN [tabla2]
ON tabla1.columnajoin=tabla2.columnajoin
```

+ Regresa los registros de la **tabla de la derecha** aun cuando no exista coincidencia con los de primera tabla(cuyo caso asignará valores nulos)

col1A	col2A	col3A
A	dato A	o
B	dato B	p
C	dato C	q
D	dato D	r

col1B	col2B
C	s
E	t
F	u

=

col1A	col2A	col3A	col1B	col2B
C	dato C	q	C	s
NULL	NULL	NULL	E	t
NULL	NULL	NULL	F	u

FULL JOIN

```
SELECT [columnas]
FROM [tabla1]
FULL JOIN [tabla2]
ON tabla1.columnajoin=tabla2.columnajoin
```

+ Regresa los registros **todos los registros** de ambas tablas, aun cuando no exista coincidencia

col1A	col2A	col3A
A	dato A	o
B	dato B	p
C	dato C	q
D	dato D	r

col1B	col2B
C	s
E	t
F	u

=

col1A	col2A	col3A	col1B	col2B
A	dato A	o	NULL	NULL
B	dato B	p	NULL	NULL
C	dato C	q	C	s
D	dato D	r	NULL	NULL
NULL	NULL	NULL	E	t
NULL	NULL	NULL	F	u

SELECT * FROM a
INNER JOIN b ON a.key = b.key



SELECT * FROM a
LEFT JOIN b ON a.key = b.key



SELECT * FROM a
RIGHT JOIN b ON a.key = b.key

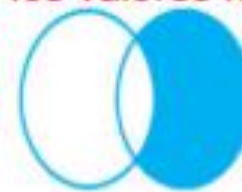


Con estos dos estamos quitando los valores null

SELECT * FROM a
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL



SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL



SELECT * FROM a
FULL JOIN b ON a.key = b.key



SELECT * FROM a
FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL



Este filtro de WHERE es para devolver aquellos que sí encontró en ambos lados

CROSS JOIN

```
SELECT [columnas]  
FROM [tabla1], [tabla2], ...
```

```
CREATE TABLE month( MM int);  
CREATE TABLE year( YYYY int);
```

```
INSERT INTO month  
VALUES (1), (2), (3), (4), ... , (12);  
INSERT INTO year  
VALUES (2011), (2012), (2013), (2014), ... , (2021);
```

```
SELECT a.YYYY, b.MM FROM year AS a, month AS b;
```

+ El resultado de un *cross join* de dos tablas consiste en combinar cada registro de la tabla1 con cada registro de la tabla2

col1A	col2A	col3A
A	dato A	o
B	dato B	p
C	dato C	q
D	dato D	r

col1B	col2B
C	s
E	t
F	u

=

col1A	col2A	col3A	col1B	col2B
A	dato A	o	C	s
A	dato A	o	E	t
A	dato A	o	F	u
B	dato B	p	C	s
B	dato B	p	E	t
B	dato B	p	F	u
C	dato C	q	C	s
C	dato C	q	E	t
C	dato C	q	F	u
D	dato D	r	C	s
D	dato D	r	E	t
D	dato D	r	F	u

CONSULTA COMBINADA

SELECT [columnas]

FROM [tabla1]

COMANDO

SELECT [columnas]

FROM [tabla2]

Dentro de las sentencias que se pueden usar podemos encontrar

INTERSECT, INTERSECT ALL

EXCEPT

UNION, UNION ALL

+ Permite combinar en un solo conjunto de resultados (output de una query) las salidas de dos consultas de tipo **SELECT**

CONSULTA COMBINADA

INTERSECT: se utiliza para encontrar filas en común de ambas consultas, sin valores duplicados. Si se añade la sentencia **ALL** muestra aquellos valores duplicados de la primera tabla.

EXCEPT: se utiliza para encontrar filas que están en una tabla, pero no en la otra.

UNION: se utiliza para juntar todas las filas de ambas consultas. Si se añade la sentencia **ALL** muestra aquellos valores duplicados.

```
SELECT customer_id FROM sales_2015
```

```
INTERSECT
```

```
SELECT customer_id FROM customer_20_60 ;
```

Col1A	Col2A
A	dato A
B	dato B
C	dato C
D	dato D
C	dato C

Col1B	Col2B
C	s
C	dato C
F	u
C	dato C
G	v
C	dato C

INTERSECT

Intersección simple

Col1	Col2
C	dato C

Intersección completa

Col1	Col2
C	dato C
C	dato C

INTERSECT ALL

EXCEPT

Resta A-B

Col1	Col2
A	dato A
B	dato B
D	dato D

UNION

Unión simple

Col1A	Col2A
A	dato A
B	dato B
C	dato C
D	dato D
C	s
F	u
G	v

Unión completa

Col1A	Col2A
A	dato A
B	dato B
C	dato C
D	dato D
C	dato C
C	s
C	dato C
F	u
C	dato C
G	v
C	dato C

UNION ALL

SUBCONSULTAS

En **WHERE**:

```
SELECT [columnas]
FROM [nombre tabla 1]
WHERE [nombre columna 1] [operador de comparación]
      ( SELECT [nombre columna 2]
        FROM [nombre tabla 2]
        WHERE [condiciones] );
```

```
SELECT * FROM sales
WHERE customer_id IN
      ( SELECT customer_id FROM customer
        WHERE age>60 );
```

+ Consiste en realizar una consulta dentro de otra consulta que se está realizando en ese momento.

+ Estas subconsultas se pueden realizar en el **SELECT**, **FROM** o en el **WHERE**

+ Devolverá aquellos registros cuyos valores de customer_id coincidan con los valores que aparecen después de la sentencia **IN**

SUBCONSULTAS

En **FROM**:

SELECT

a.producto_id,
a.product_name,
a.category,
b.cantidad

FROM product **AS** a

LEFT JOIN

(**SELECT** product_id, **SUM**(quantity) **AS** cantidad **FROM** sales
GROUP BY product_id) **AS** b

ON a.product_id=b.product_id

ORDER BY b.cantidad;

+ Se pudo haber puesto “**ORDER BY** cantidad” y de igual forma hubiera funcionado

+ Se pudo haber puesto “**ORDER BY** cantidad” y de igual forma hubiera funcionado

SUBCONSULTAS

En **SELECT**:

SELECT

customer_id,

order_id,

(**SELECT** customer_name **FROM** customer

WHERE sales.customer_id=customer.customer_id)

FROM sales

ORDER BY customer_id **DESC**;

+ En este ejemplo se emplea la subconsulta en una columna que queremos que se muestre.

+ La subconsulta está medio rara, pero esta es la idea que se busca ejemplificar. Ya dependerá qué instrucciones necesitemos en la subconsulta.

VIEWS

```
CREATE [o REPLACE] VIEW [nombre de la vista] AS  
SELECT [columnas]  
FROM [nombre de la tabla]  
( WHERE en caso de ser necesario) ;
```

```
CREATE VIEW logística AS
```

```
SELECT a.order_line,  
        b.customer_name,  
        b.state
```

```
FROM sales AS a
```

```
LEFT JOIN customer AS b
```

```
ON a.customer_id=b.customer_id ;
```

+ Es una tabla virtual resultado de una consulta.

+ Se almacenan en
Schemas > Views

+ La query para crear la view puede ser tan compleja como lo necesitemos

+ Las **VIEW** no son tan fáciles de actualizar, lo mejor sería tratarlas como tablas.

VIEWS

DROP VIEW [nombre de la vista] ;

SELECT * FROM [nombre de la vista] ;

+ Este resultado sí elimina la **VIEW** por completo de la memoria de la computadora. A diferencia de **DELETE FROM**, este último solo hacía un vaciado de tabla.

+ Para llamar a una **VIEW** se usa la misma sentencia que si estuviéramos llamando una tabla

FUNCIONES DE CARACTER

LENGTH

SELECT LENGTH(texto);

+ Devuelve la longitud del carácter que se le ingresa como argumento. Es una función vectorizada.

UPPER/LOWER

SELECT UPPER(texto);

SELECT LOWER(texto);

+ Convierte todos los caracteres a mayúsculas o minúsculas. Son funciones vectorizadas.

REPLACE

SELECT REPLACE(texto donde se va a reemplazar, texto que se va a reemplazar, texto con el que se va a reemplazar);

+ Reemplaza todas las apariciones de un texto específico. Es sensible a mayúsculas y minúsculas. Es una función vectorizada.

FUNCIONES DE CARACTER

TRIM

TRIM(LEADING 'texto a quitar' FROM 'texto donde se va a quitar')

SELECT TRIM(LEADING 'A' FROM 'AAA Yo soy Popeye');

TRIM(TRAILING 'texto a quitar' FROM 'texto donde se va a quitar')

SELECT TRIM(TRAILING 'B' FROM 'Yo soy Popeye BB');

TRIM(BOTH 'texto a quitar' FROM 'texto donde se va a quitar')

SELECT TRIM(BOTH '' FROM ' Yo soy Popeye BB ');

+ **TRIM** elimina la cadena máxima de todos los caracteres especificados de un texto específico

+ **LEADING** es para que busque desde la izquierda

+ **TRAILING** es para que busque desde la derecha

+ Cuando se topen un carácter distinto al especificado, detendrán el borrado

FUNCIONES DE CARACTER

LTRIM

LTRIM('texto donde se va a quitar', 'texto que se va a quitar')

```
SELECT RTRIM( ' Yo soy Popeye', '' );
```

RTRIM

RTRIM('texto donde se va a quitar', 'texto que se va a quitar')

```
SELECT RTRIM( 'Yo soy Popeye ', '' );
```

+ **LTRIM** elimina todos los caracteres especificados de un texto desde la izquierda

+ **RTRIM** elimina todos los caracteres especificados de un texto desde la derecha

CONCATENACIÓN

[texto] || [texto] || [texto]

SELECT customer_name, city || ' ' || state || ' ' || country **AS** dirección
FROM customer ;

+ El operador || nos permite concatenar textos

+ Muestra dos columnas, la primera el customer_name y la segunda columna es el resultado de concatenar con el operador ||

SUBSTRING

SUBSTRING('texto original', **FROM** [posición original],
FOR [posición final])

```
SELECT customer_id, customer_name,  
SUBSTRING(customer_id FOR 2) AS customer_grupo  
FROM customer  
WHERE SUBSTRING(customer_id FOR 2)='AB';
```

+ Nos permite extraer un subtexto de un texto especificado.

+ **FROM** indica desde qué posición empieza a extraer. **FOR** indica cuántos caracteres extraerá.

+ **FROM** indica desde qué posición empieza a extraer. **FOR** indica cuántos caracteres extraerá.

STRING_AGG

STRING_AGG([expresión], 'delimitador')

```
SELECT state STRING_AGG(producto_id, ',' )  
FROM customer  
GROUP BY state;
```

+ Concatena caracteres como una lista, separados por un delimitador especificado. Es una **función vectorizada**

+ El delimitador es arbitrario, puede ser cualquiera que necesitemos

+ Con este se pueden concatenar elementos de la misma columna

FUNCIONES MATEMÁTICAS

FLOOR / CEIL

FLOOR([número]);

CEIL([número]);

+ Son las funciones piso y techo que se conocen en matemáticas

SELECT order_line, sales, **FLOOR**(sales), **CEIL**(sales)

FROM sales

WHERE discount>0 ;

RANDOM

RANDOM();

CEIL([número]);

+ Se usa para generar un número aleatorio entre 0, inclusive, y 1, exclusive.

SELECT **RANDOM**(), **RANDOM**()*(50-10)+10

FUNCIONES MATEMÁTICAS

SETSEED

SEETSEED([semilla]);

SELECT SETSEED(0.5);

SELECT RANDOM(), **RANDOM()***(50-10)+10;

+ Es útil para fijar el mismo número aleatorio, por ejemplo, para cuando se quiere repetir las simulaciones

+ El setseed únicamente aplicará al primer comando que contenga aleatoriedad

ROUND

ROUND([número],[cantidad de decimales]);

SELECT order_line, sales, **ROUND**(sales), **ROUND**(sales,2)

FROM sales;

+ Redondea números hasta una cierta cantidad de decimales. Si no se especifica un número de decimales, lo redondea al siguiente entero si su parte decimal es mayor a 0.5

FUNCIONES MATEMÁTICAS

POWER

POWER(m,n);

SELECT age, **POWER**(age,2)

FROM customer **ORDER BY** age;

+ Sirve para calcular potencias. Con esta sentencia estamos calculando m^n

FUNCIONES DE FECHA

CURRENT_DATE

CURRENT_DATE

+ Nos devuelve la **fecha actual** en formato
YYYY-MM-DD

CURRENT_TIME

CURRENT_TIME([precisión]);

+ Nos devuelve la **hora actual** en formato
HH:MM:SS.GMT+TZ

CURRENT_TIMESTAMP

CURRENT_TIME([precisión]);

+ Nos devuelve la **fecha y hora actuales** en
formato
HH:MM:SS.GMT+TZ

SELECT CURRENT_DATE,
CURRENT_TIME,
CURRENT_TIME(3),
CURRENT_TIMESTAMP(3);

AGE

AGE([fecha 1], [fecha 2]);

SELECT

AGE('2014-04-25', '2014-01-01'),

AGE(**TIMESTAMP** '2014-04-25 17:00:10');

SELECT order_line, order_date, ship_date,

AGE(ship_date, order_date) **AS** tiempo

FROM sales;

+ Devuelve la diferencia que existe entre dos fechas en número de años, meses, días, horas, minutos y segundos.

+ Lo calcula como fecha1 - fecha2

+ **TIMESTAMP** permite indicar que queremos considerar el tiempo actual.

+ En caso de poner las fechas al revés, devolverá el resultado con un signo negativo

EXTRACT

EXTRACT([unidades] FROM TIMESTAMP [fecha])

SELECT EXTRACT (day FROM TIMESTAMP '2014-04-25')

Unidad	Significado
day	Día del 1 al 31
decade	Año dividido entre 10
doy	Día en el año (acepta bisiestos)
epoch	Número de segundos desde 1 de enero de 1970 UTC (si es una fecha) Número de segundos en un intervalo (si es un intervalo de tiempo)
hour	Horas (0-23)
minute	Minutos (0-59)
second	segundos (con fracciones)
month	Número del mes (1-12) si es una fecha Número del mes (0-11) si es un intervalo de tiempo
year	año con 4 dígitos



- + Permite extraer partes de una fecha
- + Devuelve 25.
- + Las “unidades” que pueden usarse aparecen en la siguiente tabla

REGEX

Existen muchos comodines. Aquí mostraremos los más importantes y útiles.

	Es el operador 'o'
*	Denota repetición de la secuencia previa 0 o mas veces
+	Denota repetición de la secuencia previa 1 o mas veces
?	Denota repetición de la secuencia previa 0 o una vez
{m}	Denota repetición de la secuencia previa <i>m</i> veces
{m,}	Denota repetición de la secuencia previa <i>m</i> o más veces
{m, n}	Denota repetición de la secuencia previa al menos <i>m</i> pero no más de <i>n</i> veces
^,\$	Denotan inicio y final del texto
[texto]	Una expresión de caracteres. Denota coincidencia con cualquier elemento dentro de los corchetes
\s	Denota espacio en blanco
~,*	Denotan caso sensitivo o insensitivo a mayúsculas



+ Son una búsqueda de patrones por codificación

TO CHAR

TO_CHAR([valor], [parámetro])

SELECT sales, **TO_CHAR**(sales, '9999.99') **FROM** sales ;

SELECT sales, **TO_CHAR**(sales, '\$9,999.99') **FROM** sales ;

SELECT order_date, **TO_CHAR**(order_date, 'Month DD, YY') **FROM** sales ;

+ Convierte números o fechas **a texto**

+ Estamos diciendo que queremos dos decimales (los trunca), y que la parte entera sea de 4 lugares, por lo que "14" aparecerá como " 14"

+ Especificamos que el separador de miles sea una coma y que ponga el signo de pesos.

Parámetro	Significado
9	Valores (sin 0 precedente)
0	Valores (con 0 precedente)
.	Decimales
,	Separador en grupos
PR	Valores negativos en corchetes
S	Signo
L	Cualquier símbolo
MI	Signo menos para negativos
PL	Signo mas para positivos
SG	Signos mas y signo -
EEEE	Notación científica

Parámetro	Significado
YYYY	Año en cuatro dígitos
MM	Mes en número
Mon	Nombre abreviado del mes iniciando con mayúscula
Month	Nombre completo del mes iniciando con mayúscula
DAY	Nombre del día en mayúsculas
Day	Nombre del día iniciando con mayúscula
DDD	Día del año
DD	Día del mes
HH	Hora del día
HH12	Hora del día
HH24	Hora del día
MI	Minutos
SS	Segundos
am,AM,pm,PM	indicador meridiano

TO DATE

TO_DATE([valor], [parámetro])

SELECT TO_DATE ('2019/01/15', 'YYYY/MM/DD');

Devuelve 2019-01-05

SELECT TO_DATE ('26122018', 'DDMMYYYY');

Devuelve 2018-12-26

+ Convierte textos **a fechas**

+ Se tiene que especificar en qué formato viene el texto que estamos metiendo como input para que SQL la ponga en formato YYYY-MM-DD